

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
23 February 2006 (23.02.2006)

PCT

(10) International Publication Number
WO 2006/020948 A1

- (51) International Patent Classification⁷: **C02F 1/18**, B01D 1/00, B01B 1/06
- (21) International Application Number: PCT/US2005/028898
- (22) International Filing Date: 12 August 2005 (12.08.2005)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
60/601,225 13 August 2004 (13.08.2004) US
11/200,619 10 August 2005 (10.08.2005) US
- (71) Applicant (for all designated States except US): **CY-PRESS SEMICONDUCTOR CORPORATION** [US/US]; 3901 North First Street, San Jose, CA 95134-1599 (US).
- (72) Inventors; and
- (75) Inventors/Applicants (for US only): **SNYDER, Warren** [CA/US]; 6405 158St. SE, Snohomish, WA 98296 (US). **MAHESHWARI, Dinesh** [IN/US]; 2997 Bavlis Street, Fremont, CA 94538 (US). **OGAMI, Kenneth** [US/US]; 14232 93rd Avenue NE, Bothell, WA 98011 (US). **HASTINGS, Mark** [US/US]; 9728 50th PI West, Mukilteo, WA 98275 (US).
- (74) Agents: **MALLIE, Michael, J.** et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 12400 Wilshire Boulevard, 7th Floor, Los Angeles, CA 90025 (US).
- (81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.
- (84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).
- Published:**
— with international search report
- For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: PROVIDING HARDWARE INDEPENDENCE TO AUTOMATE CODE GENERATION OF PROCESSING DEVICE FIRMWARE

(57) Abstract: A user application is generated in response to user input, wherein the user application is described in a user application description. Processing device code is generated for a targeted processing device based at least in part on the user application description without user intervention, wherein the processing device code includes a system layer, wherein functionality of the system layer is independent of the targeted processing device.



WO 2006/020948 A1

PROVIDING HARDWARE INDEPENDENCE TO AUTOMATE CODE GENERATION OF PROCESSING DEVICE FIRMWARE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/601,225, filed August 13, 2004, and incorporated herein by reference in its entirety.

BACKGROUND

Field

Embodiments of the invention relate to the field of processing devices and more specifically, but not exclusively, to providing hardware independence to automate generation of processing device firmware.

Background Information

Processing devices, such as microcontrollers, are widely used in the industry as control elements in many solutions. Most microcontrollers are general in purpose, and are designed for use in a wide variety of problem solutions. As microcontrollers become more programmable and more widely applicable, a designer needs more specific device knowledge to use the microcontroller to solve a problem.

In a conventional solution, hardware and software are usually created for a specific microcontroller, and may be redesigned (sometimes completely) following a change in requirements. A common sequence of events is to first determine the system requirements (often incompletely when time is short) to address a problem, then second to determine hardware and software requirements, then third to determine microcontroller and interfacing circuitry requirements, and fourth to find a suitable microcontroller and design suitable interfaces. Finally, the user

must manually configure the microcontroller and write device specific firmware, and the user may have to re-write firmware, redesign circuitry, or choose another microcontroller based upon new/changing requirements.

Design changes during this conventional approach may result in costly and inefficient code changes, may cause software and hardware architecture changes and may even require a change in microcontroller and a significant redesign. Such a redesign may be costly and may delay design and production schedules. Also, designing and redesigning conventional microcontrollers may require specialists in a variety of areas including hardware engineering, firmware coding, and system design.

BRIEF DESCRIPTION OF THE DRAWINGS

Non-limiting and non-exhaustive embodiments of the present invention are described with reference to the following figures, wherein like reference numerals refer to like parts throughout the various views unless otherwise specified.

Figure 1A is a flowchart illustrating the logic and operations to provide automated code generation of processing device firmware in accordance with one embodiment of the present invention.

Figure 1B is a diagram illustrating a system to provide automated code generation of processing device firmware in accordance with one embodiment of the present invention.

Figure 1C is a Graphical User Interface (GUI) in accordance with one embodiment of the present invention.

Figure 2 is a diagram illustrating a firmware stack in accordance with one embodiment of the present invention.

Figure 3 is a diagram illustrating a firmware stack in accordance with one embodiment of the present invention.

Figure 4A is a diagram illustrating a driver in accordance with one embodiment of the present invention.

Figure 4B is a diagram illustrating a base project in accordance with one embodiment of the present invention.

Figure 5 is a diagram illustrating a processing device in accordance with one embodiment of the present invention.

Figure 6 is a diagram illustrating a computer system in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

In the following description, numerous specific details are set forth to provide a thorough understanding of embodiments of the invention. One skilled in the relevant art will recognize, however, that embodiments of the invention can be practiced without one or more of the specific details, or with other methods, components, materials, etc. In other instances, well-known structures, materials, or operations are not shown or described in detail to avoid obscuring understanding of this description.

Reference throughout this specification to “one embodiment” or “an embodiment” means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrases “in one embodiment” or “in an embodiment” in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

In the following description and claims, the term “coupled” and its derivatives may be used. “Coupled” may mean that two or more elements are in direct contact (physically, electrically, magnetically, optically, etc.).

“Coupled” may also mean two or more elements are not in direct contact with each other, but still cooperate or interact with each other.

Turning to Figures 1A-1B, embodiments to provide automated code generation of processing device firmware is shown. Figure 1A shows a flowchart 100 in accordance with an embodiment of the invention. Figure 1B shows a system 120 in accordance with an embodiment of the invention. In one embodiment, the logic of flowchart 100 and components of system 120 may be embodied in instructions executable by a computer system.

Starting in a block 102 of flowchart 100, a user application is generated in response to user input. Referring to Figure 1B, user 122 may use a processing device maker GUI 124 to construct user application 127, also referred to as a system, for implementation on a processing device 140. Processing device maker GUI 124 overlays a processing device maker 126. User application 127 may be described in a user application description 128. In one embodiment, user application description 128 is a text file that describes the user's application.

An embodiment of processing device maker GUI 124 is shown in Figure 1C. GUI 124 may be generated in a variety of ways, such as using HyperText Markup Language (HTML), JavaScript, or the like. GUI 124 may include an interface such for Programmable System on a Chip™ (PSoC™) Express™ of the Cypress Semiconductor Corporation. However, it will be understood that embodiments of GUI 124 are not limited to the interface shown in Figure 1C or to any interfaces of PSoC Express™.

Tabs for various stages in designing a user application are shown at 152. These tabs include Design, Simulation, Bill of Materials/Schematic (BOM/Schem), and Build. The Design tab is where user 122 may use the GUI to create a user application. The Simulation

tab provides a software simulation of the user's application. The BOM/Schem tab is used to generate a bill of materials, schematic and/or a datasheet for the user's application. The Build tab may invoke the automated code functionality to provide a user with code to implement the user's application without coding by the user.

In Figure 1C, user 122 is creating a user application to turn a Light Emitting Diode (LED) on and off using a push-button switch. User 122 may use a tray 154 at the bottom of GUI 124 to drag-and-drop desired input/output devices and set up transfer functions between the input/output devices.

In the example of Figure 1C, user 122 has dragged push-button switch 156 and LED 158 into the design area. User 122 has also set up a transfer function 160 between switch 156 and LED 158. Transfer function 160 may be defined such that when switch 156 is pressed, LED 158 is on, and when switch 156 is released, LED 158 is off.

In general, a transfer function defines the behavior of an output device. The transfer function may define the behavior of an output device in response to one or more inputs. In one embodiment, transfer function types include a truth table transfer function for mapping specific output actions and/or output states to permutations of discrete inputs. In another embodiment, the transfer function types include a setpoint transfer function to convert a continuous input signal to a number of discrete values. User 122 may provide a number of inputs to the transfer function and assign an output state to each combination of the inputs. After the transfer function is defined, processing device maker 126 may validate the transfer function. If the transfer function is determined to be invalid, an error message may be displayed to user 122.

In one embodiment, after block 102 of flowchart 100, user 122 may perform a simulation of user application 127 using system 120, as shown

at a block 103. In one embodiment, user 122 selects the Simulation tab of GUI 124 to enter a simulation mode. The simulation capability enables hardware independence by verifying the transfer function behavior without requiring the user to compile and debug the firmware on the targeted processing device. Simulation also includes the ability to create complex input files to exhaustively test the transfer function behavior with arbitrarily large combinations of input values. The simulation logs the outputs based on the transfer function behavior so that the results may be analyzed by the user.

After block 103, the logic proceeds to a block 104 to generate processing device code for a targeted processing device without user intervention. After user 122 has completed the user application 127, user 122 may use processing device maker GUI 124 to request that processing device code be automatically generated. User 122 does not have to perform any actual coding.

In one embodiment, user 122 selects the Build tab of GUI 124. In one embodiment, user 122 may select a targeted processing device from a list of devices. The list may include other information regarding the targeted processing devices, such as processing device cost. After user 122 selects the targeted processing device, code is automatically generated for the device.

Turning to Figure 1B, user application description 128 is handed-off to a processing device designer 130 for the generation of processing device code. In one embodiment, processing device designer 130 includes PSoC Designer™ of the Cypress Semiconductor Corporation.

Processing device designer 130 may include an automatic code generator 132. Automatic code generator 132 assembles the code for the user's application 127 based on user application description 128. Automatic code generator 132 generates processing device code 135. In

one embodiment, code 135 may include a high-level language, such as C, a low-level code, such as Assembly, or a combination thereof. In one embodiment, automatic code generator 132 includes a Controller Maker engine (CMX) of the Cypress Semiconductor Corporation.

In one embodiment, to assemble code 135, automatic code generator 132 may reference libraries 136. Libraries 136 include code blocks that may be combined to form code 135. Automatic code generator 132 may use at least a portion of user application description 128 as a guide in gathering together various code blocks. Some of the code blocks may be selected based at least in part on the targeted processing device.

Returning to flowchart 100, after block 104, the logic may continue to a block 106 where code 135 is compiled. In system 120, code 135 may be compiled by compiler 134 of processing device designer 130 to generate a binary 138, also known as a binary image or a Read-Only Memory (ROM) image.

Continuing to a block 108, the binary is loaded into the targeted processing device. In Figure 1B, binary 138 is loaded into a Non-Volatile Storage (NVS) 142 of processing device 140. In one embodiment, NVS 142 includes Flash memory.

Embodiments of processing device 140 may include one or more general-purpose processing devices, such as a microprocessor or central processing unit, a network processor, a microcontroller, an embedded Programmable Logic Device (PLD), or the like. In one embodiment, processing device 140 includes a Cypress MicroSystem's PSoC™ microcontroller. Alternatively, the processing device may include one or more special-purpose processing devices, such as a Digital Signal Processor (DSP), an Application Specific Integrated Circuit (ASIC), a Field Programmable Gate Array (FPGA), or the like. The processing device

may also include any combination of a general-purpose processing device and a special-purpose processing device.

It will be appreciated that processing device code 135 may be generated without user intervention. User 122 does not have to write any code, so there is no need for C coding or Assembly coding experts. Also, because the code is constructed from pre-built and pre-tested code libraries, time wasted on debugging, such as finding syntax errors, is eliminated.

It will also be appreciated that user 122 generated user application 127 without referencing a targeted processing device. Instead of choosing a processing device to implement a user application and then writing code for that processing device, embodiments of the present invention allow a user application to be created and then code automatically generated for a particular processing device. Moreover, a user may take a user application, make revisions to the user application, and quickly generate revised programming device code. User 122 no longer has to waste time rewriting and debugging code just because a single sensor has been added to a user application.

In flowchart 100, after block 108, the logic may continue to a block 110, to generate processing device code for a second targeted processing device, where the second targeted processing device is different than the first targeted processing device. User 122 may take user application 127 that has already been constructed and use it to generate code for a different processing device. Automated code generator 132 is aware of the specific coding requirements of the second targeted device and assembles the appropriate code blocks from libraries 136. Thus, user 122 may easily move user application 127 to other processing devices without spending time and money recoding for a new targeted processing device.

Continuing to a block 112 of flowchart 100, one or more libraries used by processing device designer 130 may be updated. These updates may be received on disk, such as a Compact Disk Read-Only Memory (CD-ROM), or downloaded over a network. Libraries 136 may be updated with new components, such as a new temperature sensor, or updated with new targeted processing devices that have come onto the market. Thus, if a new processing device comes on the market, user 122 does not have to worry about learning new firmware coding requirements for the new processing device since all the coding is taken care of by processing device designer 130.

Referring to Figure 2, a firmware stack 200 in accordance with one embodiment of the invention is shown. Firmware stack 200 shows a logical structure of at least a portion of the processing device code 135. As discussed below, a portion of the stack is abstracted away from specific hardware. Such hardware independency provides the automatic code generator a consistent architecture for stitching together various code blocks.

Firmware stack 200 includes a system layer 202, a hardware encapsulation layer 204, and a base project layer 206. As will be described further below, the functionality of the system layer 202 is independent of the targeted processing device. Also, interfaces, such as Application Program Interfaces (APIs), made between system layer 202 and the remaining layers of firmware stack 200 are standardized regardless of the targeted processing device. The term "standardized" refers to the hardware independence of the APIs. This abstraction away from specific hardware allows system layer 202 to function without regard to the particular hardware. The low layers of firmware stack 200 have "knowledge" of the specific hardware and take care of the implementation details for the system layer 202.

The hardware encapsulation layer 204 and the base project layer 206 are generated based at least in part on the targeted processing device. Hardware encapsulation layer 204 represents the underlying hardware to system layer 202. Base project layer 206 includes a set of standard functions associated with the targeted processing device hardware. Base project layer 206 may include functionality at the register level of the targeted processing device.

System layer 202 may include transfer functions 208 and drivers 212. System layer 202 is targeted by an application level mapping function. Transfer functions 208 invoke the transfer functions defined by user 122.

Drivers 212 are usually associated with a hardware component of the processing device. In one embodiment, drives 212 may include three types: input, output, or interface. An output driver may be used with a device that is controlled by the user application, such as a fan or heater. Input drivers may be used for sensors, such as temperature or voltage sensors. Interface drivers may be used for devices that allow access to system variables and status, such as an Inter-Integrated Circuit (I2C) or a Serial Peripheral Interface (SPI).

Transfer functions 208 and drivers 212 may communicate with each other using APIs 222. Embodiments of an API include *DriverName-Instantiate* to initialize a device, *DriverName_GetValue* to return a value from an input device, and *DriverName_SetValue* to set an output of an output device to a specific value. Such APIs are defined such that they may be invoked regardless of the particular hardware.

Drivers 212 communicate with hardware encapsulation layer 204 using channels 224. Channels 224 are hardware independent. A channel may be further defined by a channel type, such as an input voltage

channel, an output voltage channel, or the like. In one embodiment, channels 224 are implemented as APIs.

Hardware encapsulation layer 204 may include low level drivers 214 and system initialization 210. Low level drivers 214 provide the implementation of channels 224. In one embodiment, all drivers 212 use one or more channels 224 to communicate with low level drivers 214. In one embodiment, a channel may have associated parameters assigned by a low level driver, and the associated driver must conform to those parameters.

In one embodiment, the base project layer 206 includes User Modules (UMs) 216 and PSoC™ processing device hardware 218. User modules 216 are used with block arrays in PSoC™ processing device hardware 218 to form hardware components, such as an Analog-Digital Converter (ADC) (discussed further below). It will be understood that embodiments of the invention are not limited to PSoC™ processing devices. In alternative embodiments, base project layer 206 includes a non-PSoC™ processing device.

Figure 2 also shows low level drivers 214, user modules 216, and PSoC™ Hardware 218 grouped into a PSoC™ Base Project 220. Embodiments using a PSoC™ Base Project will be discussed below.

Turning to Figure 3, a diagram illustrating the interaction between layers of a firmware stack 300 is shown. The embodiment of Figure 3 shows a transfer function between a single input device and a single output device for the sake of clarity. However, it will be appreciated that embodiments of the invention may be used with other user applications having various configurations of inputs, outputs, and transfer functions.

Firmware stack 300 includes a transfer function 308 layered on a driver 306. Transfer function 308 and driver 306 communicate via APIs 307. Driver 307 is layered on a low level driver 304. Driver 306 and low

level driver 304 communicate using volt channels 305. In one embodiment, a volt channel may be used by driver 306 for receiving a voltage from low level driver 304.

Low level driver 304 is layered on a user module/processing device 302. User module/processing device 302 includes a temperature sensor 309 and a fan 318.

In the embodiment of Figure 3, fan 318 is controlled in response to the temperature read by temperature sensor 309. At user module/processing hardware 302, a raw count value 310, corresponding to a reading from sensor 309, is sent to low level driver 304. Low level driver 304 takes the raw count value and converts it to a volt value 312.

Driver 306 may request and receive the volt value 312 using volt channels 305. Driver 306 may respond to temperature sensor requests from transfer function 308 and return a temperature value 314 converted from voltage value 312. Transfer function may then issue a command 316, if any, for fan 318 based on the temperature received. In Figure 3, transfer function 308 issues command 316 to turn on fan 318 in response to the temperature at temperature sensor 309. This "turn on fan" command may travel down through the firmware stack to execute the command on fan 318.

It will be appreciated that transfer function 308 and driver 306, as well as APIs 307 and volt channels 305, are hardware independent. Low level driver 304 and UM/processing device 302 have "knowledge" of how to interact with the particular hardware components. Since transfer function 308 and driver 306 are hardware independent and their communication interfaces are standardized, they can be easily hooked to low level drivers and UM/processing devices.

Turning to Figure 4A, an embodiment of driver 400 is shown. In one embodiment, driver 400 is compatible with the PsoC™ family of

products. Driver 400 includes several components for device selection and code generation. These components may be grouped into ControllerMaker Interface files 402, Data files 404, and Source Code files 406. In one embodiment, processing device designer 130 includes PsoC™ ControllerMaker.

Interface files 402 include files to generate the visual interface in processing device maker GUI 124. Data files 404 are used for driver type selection, resource allocation, user parameter selection, and code generation. Data files 404 may include a *DriverName.cmx* file that includes information about channel type, resource requirements, association with image files, and user selectable parameters. Source code files 406 include the driver firmware. Driver 400 may be written in C, Assembly, or a combination thereof.

Turning to Figure 4B, an embodiment of a base project 420 is shown. In one embodiment, base project 420 is compatible with the PSoC™ family of products. A PSoC™ Base Project includes low level drivers and UMs.

In one embodiment, base project 420 is built around the targeted processing device. Automatic code generator 132 refers to user application description 128 and adds the appropriate code for the drivers and transfer functions. The standardized calls to and from the drivers provide a consistent architecture for reliably connecting base project 420 to drivers.

Base project 420 may include any system design project using any of the processing device family of parts that support one or more driver channel types. In one embodiment, if a base project supports a given channel type, it must support it fully. The channel may support an external interface to the input or output signal as well as low level firmware to support the signal acquisition and signal conditioning required

for reliable signal measurement. Processing device designer 130 (in one embodiment called "Controller Maker" or "PSoC Designer™") adds drivers, such as driver 400, to a base project to support the input/output devices selected by user 122 in user application 127. A project summary file is included in a base project to define what channel types are support and how many channels are available.

Base project 420 may include standard project and UM files 422, resource files 424, and low level driver files 426. Standard project and UM files 422 may include common files that are contained in any PSoC Designer™ project. These files describe the specific user modules employed in the project and their specific placement. The UM's and their placement determine the mapping of the channels to the external pins. With the UM configuration file, specific UM API files are included in so much as they are needed to control specific interrupt behavior required for successful signal acquisition and conditioning. By themselves these files create an empty project.

Low level driver files 426 provide the implementation of the channels supported by a given base project. A variety of ways can be used to implement a given channel type. For example, a volts channel can be supported by a wide range of ADCs, multiplexers, and ADC resolutions, as long as the API for the specific channel is fully supported. Some base projects may offer more resolution than others for certain channel types. These differences may be specified in a project summary file (in one embodiment, a *CMXProjectSummary.xml* file).

Resource files may include a *CMXProjectSummary.xml* file. This eXtensible Markup Language (XML) file contains metadata that communicate a list of resources provided by the base project to the CMX engine. The XML file indicates the types and count of each of the resource channels and interfaces supported. It also determines the

channel assignment order and prioritization, in the case of channels competing for similar resources. It may also provide specifications of each of the channel types, where applicable.

Turning to Figure 5, an embodiment of a processing device 500 is shown. Processing device 500 includes a PSoC™ microcontroller. Processing device 500 includes Input/Output (I/O) ports 502. In one embodiment, I/O ports 502 are programmable. I/O ports 502 are coupled to a Programmable Interconnect and Logic (PIL) 504 which is coupled to a digital block array 506. In Figure 5, digital block array 506 includes a UM 508 that has been configured as a Universal Asynchronous Receive/Transmitter (UART). Digital block array 506 is coupled to a system bus 512.

A Static Random Access Memory (SRAM) 510 and a processing core 514 are also coupled to system bus 512. Processing core 514 is coupled to NVS 516 which has stored a binary 517. In one embodiment, binary 517 includes instructions generated as described herein. In another embodiment, binary 517 may include instructions executable by processing core 514 as well as instructions for configuring block arrays 516 and 518.

Analog block array 518 is coupled to system bus 512. In the embodiment of Figure 5, analog block array 518 includes a UM 520 configured as a filter and a UM 522 configured as an ADC. Analog block array 518 is also coupled to an analog I/O unit 524 which is coupled to I/O ports 502. Processing device 500 may also include other components, not shown for clarity, including a clock generator, an interrupt controller, an I2C, or the like.

Embodiments of the present invention provide automatic generation of processing device code. A user is presented with a user application design tool for constructing a user application. Processing

device code for the user's application is automatically generated without user intervention. The processing device code includes layers of hardware abstraction that progressively isolate hardware interactions to specific hardware components. Embodiments herein produce firmware with deterministic results, eliminate the need for debugging, and significantly reduce project completion time. Design verification may be performed at the user application level instead of struggling with assembly-language and register level verification.

Further, changes may be made to the user application and new code generated without the user having to endure the painful processing of revising code written previously, and in some cases, code written by someone else. Embodiments herein automatically stitch code pieces together from various library files. Embodiments herein also provide expansion capability by updating files to support code for new processing devices and hardware components.

Figure 6 is an illustration of one embodiment of an example computer system 600 on which embodiments of the present invention may be implemented. Computer system 600 includes a processor 602 and a memory 604 coupled to a chipset 606. Storage 612, Non-Volatile Storage (NVS) 605, network interface (I/F) 614, and Input/Output (I/O) ports 618 may also be coupled to chipset 606. Embodiments of computer system 600 include, but are not limited to, a desktop computer, a notebook computer, a server, a personal digital assistant, a network workstation, or the like. In one embodiment, computer system 600 includes processor 602 coupled to memory 604, processor 602 to execute instructions stored in memory 604.

Memory 604 may include, but is not limited to, Dynamic Random Access Memory (DRAM), Static Random Access Memory (SRAM),

Synchronized Dynamic Random Access Memory (SDRAM), Rambus Dynamic Random Access Memory (RDRAM), or the like.

Chipset 606 may include a memory controller and an input/output controller. Chipset 606 may also include system clock support, power management support, audio support, graphics support, or the like. In one embodiment, chipset 606 is coupled to a board that includes sockets for processor 602 and memory 604.

Components of computer system 600 may be connected by various interconnects. Such interconnects may include a Peripheral Component Interconnect (PCI), a System Management bus (SMBUS), a Low Pin Count (LPC) bus, a Serial Peripheral Interface (SPI) bus, an Accelerated Graphics Port (AGP) interface, or the like.

I/O ports 616 may include ports for a keyboard, a mouse, a display, a printer, a scanner, or the like. Embodiments of I/O ports 616 include a Universal Serial Bus port, a Firewire port, a Video Graphics Array (VGA) port, a Personal System/2 (PS/2) port, or the like.

Processing device 140 may be coupled to computer system 600 via I/O ports 616. Computer system 600 may have stored computer-readable instructions, in accordance with embodiments described herein, to allow user 122 to automatically generate processing device code for processing device 140 using computer system 600. This code may be compiled into a binary and loaded into NVS 142.

Computer system 600 may interface to external systems through network interface 614. Network interface 614 may include, but is not limited to, a modem, a Network Interface Card (NIC), or other interfaces for coupling a computer system to other computer systems. A carrier wave signal 623 may be received/transmitted by network interface 614. In the embodiment illustrated in Figure 6, carrier wave signal 623 is used to interface computer system 600 with a network 624, such as a Local Area

Network (LAN), a Wide Area Network (WAN), the Internet, or any combination thereof. In one embodiment, network 624 is further coupled to a computer system 625 such that computer system 600 and computer system 625 may communicate over network 624.

Computer system 600 also includes non-volatile storage 605 on which firmware and/or data may be stored. Non-volatile storage devices include, but are not limited to, Read-Only Memory (ROM), Flash memory, Erasable Programmable Read Only Memory (EPROM), Electronically Erasable Programmable Read Only Memory (EEPROM), Non-Volatile Random Access Memory (NVRAM), or the like. Storage 612 includes, but is not limited to, a magnetic disk drive, a magnetic tape drive, an optical disk drive, or the like. It is appreciated that instructions executable by processor 602 may reside in storage 612, memory 604, non-volatile storage 605, or may be transmitted or received via network interface 614.

It will be appreciated that in one embodiment, computer system 600 may execute Operating System (OS) software. For example, one embodiment of the present invention utilizes Microsoft Windows® as the operating system for computer system 600. Other operating systems that may also be used with computer system 600 include, but are not limited to, the Apple Macintosh operating system, the Linux operating system, the Unix operating system, or the like.

For the purposes of the specification, a machine-readable medium includes any mechanism that provides (i.e., stores and/or transmits) information in a form readable or accessible by a machine (e.g., a computer, network device, personal digital assistant, manufacturing tool, any device with a set of one or more processors, etc.). For example, a machine-readable medium includes, but is not limited to, recordable/non-recordable media (e.g., Read-Only Memory (ROM), Random Access Memory (RAM), magnetic disk storage media, optical storage media, a

flash memory device, etc.). In addition, a machine-readable medium may include propagated signals such as electrical, optical, acoustical or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.).

Various operations of embodiments of the present invention are described herein. These operations may be implemented by a machine using a processor, an Application Specific Integrated Circuit (ASIC), a Field Programmable Gate Array (FPGA), or the like. In one embodiment, one or more of the operations described may constitute instructions stored on a machine-readable medium, that when executed by a machine will cause the machine to perform the operations described. The order in which some or all of the operations are described should not be construed as to imply that these operations are necessarily order dependent. Alternative ordering will be appreciated by one skilled in the art having the benefit of this description. Further, it will be understood that not all operations are necessarily present in each embodiment of the invention.

The above description of illustrated embodiments of the invention, including what is described in the Abstract, is not intended to be exhaustive or to limit the embodiments to the precise forms disclosed. While specific embodiments of, and examples for, the invention are described herein for illustrative purposes, various equivalent modifications are possible, as those skilled in the relevant art will recognize. These modifications can be made to embodiments of the invention in light of the above detailed description. The terms used in the following claims should not be construed to limit the invention to the specific embodiments disclosed in the specification. Rather, the following claims are to be construed in accordance with established doctrines of claim interpretation.

CLAIMS

What is claimed is:

1. A method, comprising:
generating a user application in response to user input, wherein the user application is described in a user application description; and
generating processing device code for a targeted processing device based at least in part on the user application description without user intervention, wherein the processing device code includes a system layer, wherein functionality of the system layer is independent of the targeted processing device.
2. The method of claim 1 wherein the processing device code includes a hardware encapsulation layer to represent the targeted processing device to the system layer, wherein the hardware encapsulation layer is generated based at least in part on the targeted processing device.
3. The method of claim 2 wherein the system layer and the hardware encapsulation layer communicate using hardware independent interfaces.
4. The method of claim 2 wherein the processing device code includes a base project layer to support the hardware encapsulation layer, wherein the base project layer is generated based at least in part on the targeted processing device.
5. The method of claim 1 wherein generating the processing device code includes obtaining processing device code from libraries.
6. The method of claim 5, further comprising updating the libraries with new processing device code.

7. The method of claim 1, further comprising generating second processing device code for a second targeted processing device based at least in part on the user application description without user intervention, wherein the second targeted processing device is different than the targeted processing device, wherein the second processing device code includes a second system layer that is substantially similar to the system layer.
8. The method of claim 1, further comprising performing a simulation of the user application, wherein the simulation is performed independent of the targeted processing device.
9. An article of manufacture, comprising:
a machine-readable medium including a plurality of instructions which when executed perform operations comprising:
presenting a processing device maker graphical user interface (GUI) to allow a user to construct a user application using a processing device maker, wherein the user application is described in a user application description; and
generating processing device code for a targeted processing device based at least in part on the user application description without user intervention, wherein the processing device code includes a system layer, wherein functionality of the system layer is independent of the targeted processing device.
10. The article of manufacture of claim 9 wherein the system layer includes:
a transfer function to describe behavior of an output device of the targeted processing device; and

a driver associated with the output device.

11. The article of manufacture of claim 10 wherein the transfer function and the driver communicate using hardware independent application program interfaces.

12. The article of manufacture of claim 9 wherein the processing device code includes a hardware encapsulation layer to represent the targeted processing device to the system layer, wherein the hardware encapsulation layer is generated based at least in part on the targeted processing device.

13. The article of manufacture of claim 12 wherein the system layer communicates with the hardware encapsulation layer using hardware independent channels.

14. The article of manufacture of claim 12 wherein the processing device code includes a base project layer to operate at a register level of the targeted processing device to support the hardware encapsulation layer, wherein the base project layer is generated based at least in part on the targeted processing device.

15. The article of manufacture of claim 14 wherein the base project layer includes a user module to configure a portion of a block array of the targeted processing device.

16. The article of manufacture of claim 9 wherein execution of the plurality of instructions further perform operations comprising:
generating second processing device code for a second targeted processing device based at least in part on the user application

description without user intervention, wherein the second targeted processing device is different than the targeted processing device, wherein the second processing device code includes the system layer, a second hardware encapsulation layer, and a second base project layer, wherein the second hardware encapsulation layer and the second base project layer are generated for the second targeted processing device.

17. A system, comprising:

a targeted processing device including a non-volatile storage device; and

a computer system coupled to the targeted processing device, the computer system having stored a plurality of instructions which when executed by the computer system perform operations comprising:

presenting a processing device maker graphical user interface (GUI) to allow a user to construct a user application using a processing device maker, wherein the user application is described in a user application description; and

generating processing device code for the targeted processing device based at least in part on the user application description without user intervention, wherein the processing device code includes a system layer, wherein functionality of the system layer is independent of the targeted processing device.

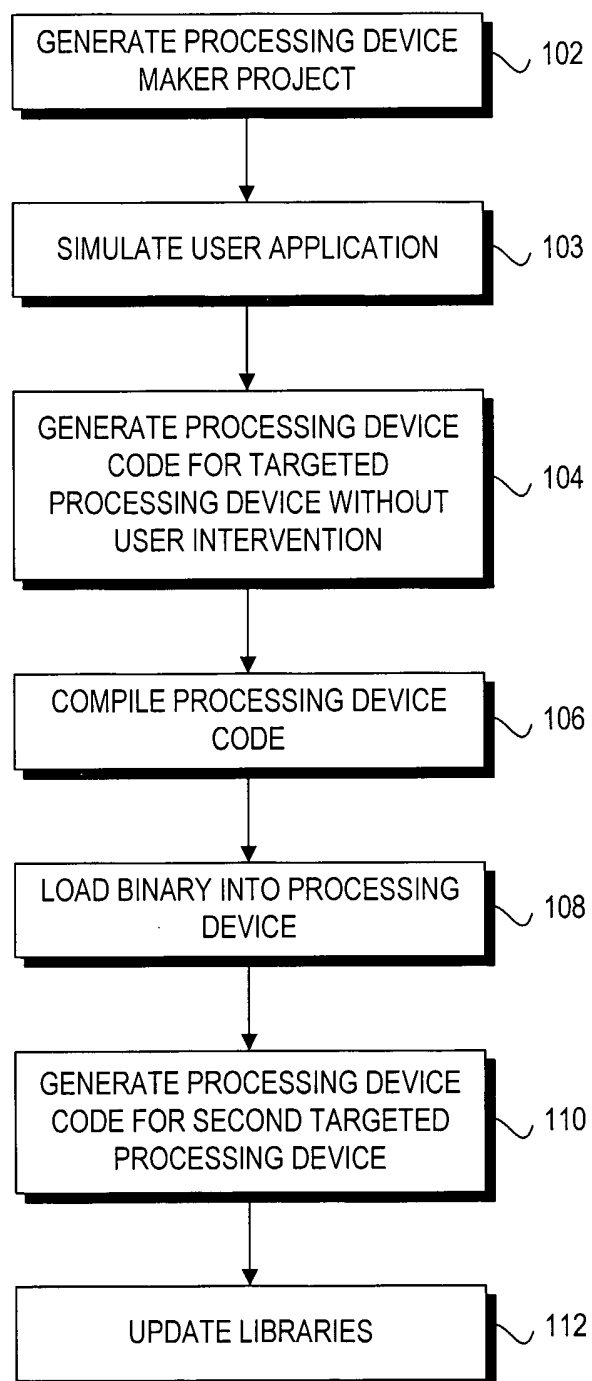
18. The system of claim 17 wherein the processing device code further includes:

a hardware encapsulation layer; and

a base project layer, wherein the hardware encapsulation layer and the base project layer are generated based on the targeted processing device.

19. The system of claim 18 wherein the system layer communicates with the hardware encapsulation layer using hardware independent channels.
20. The system of claim 17 wherein execution of the plurality of instructions further perform operations comprising:
- compiling the processing device code into a binary image; and
 - loading the binary image into the non-volatile storage of the targeted processing device.

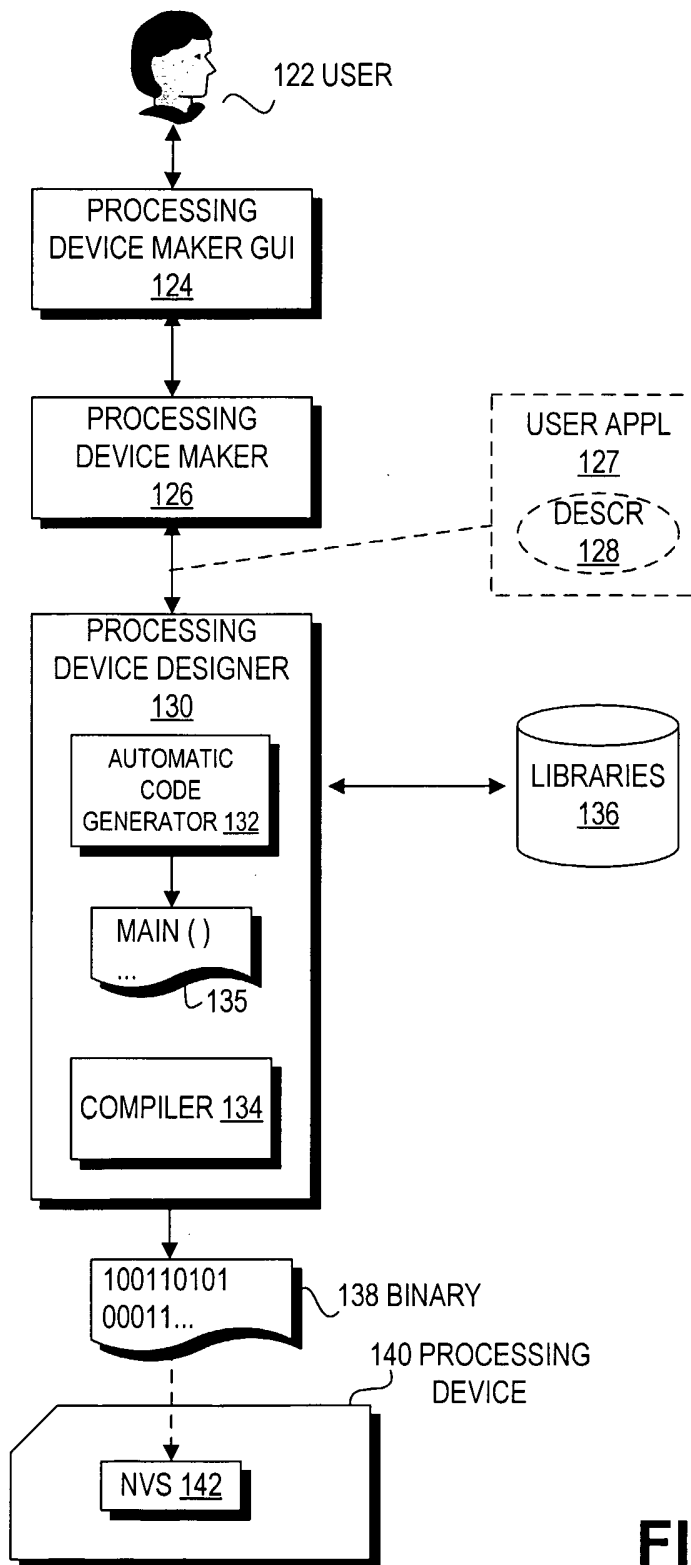
1/9



100

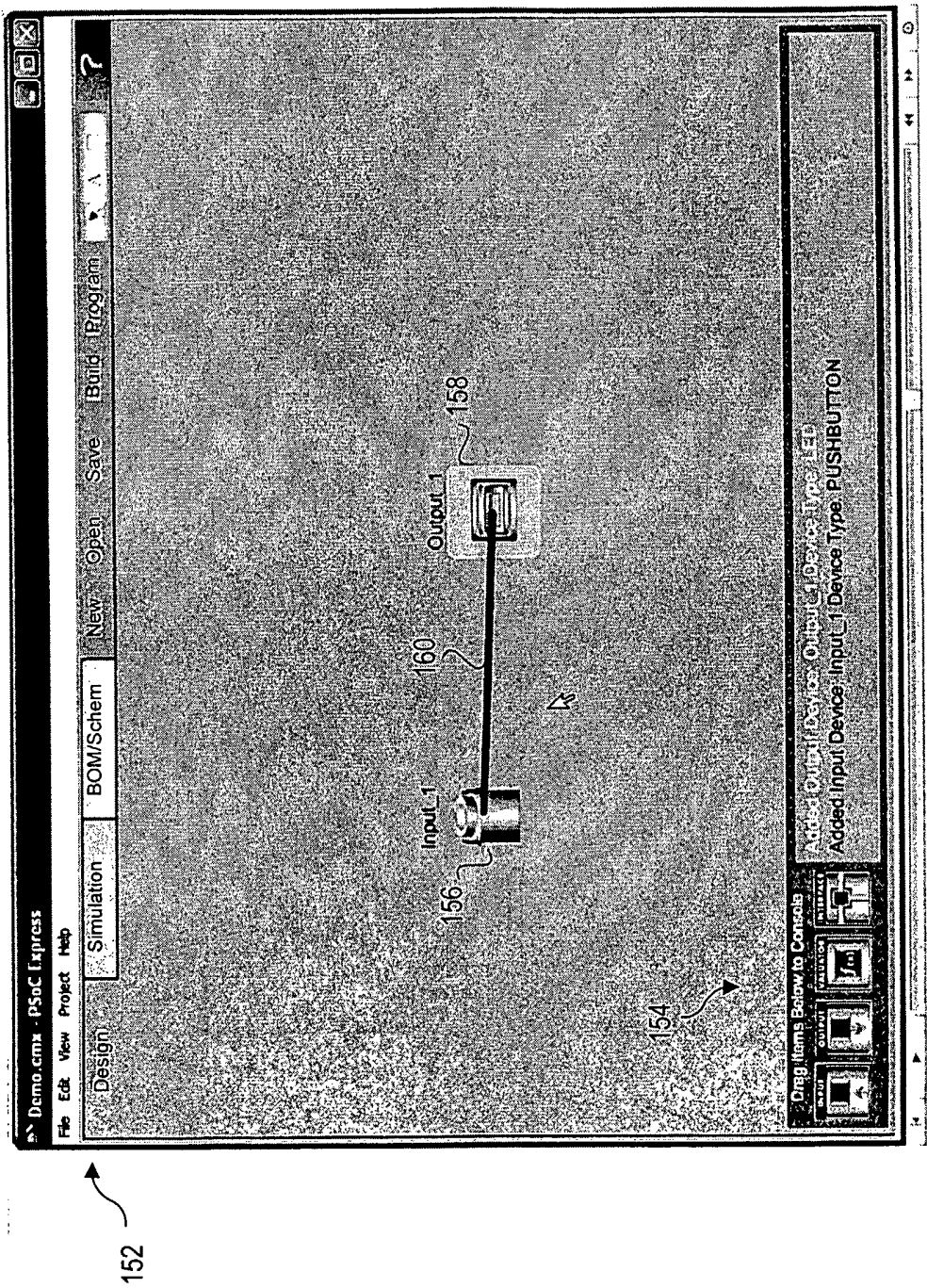
FIG. 1A

2/9



120

FIG. 1B



124

FIG. 1C

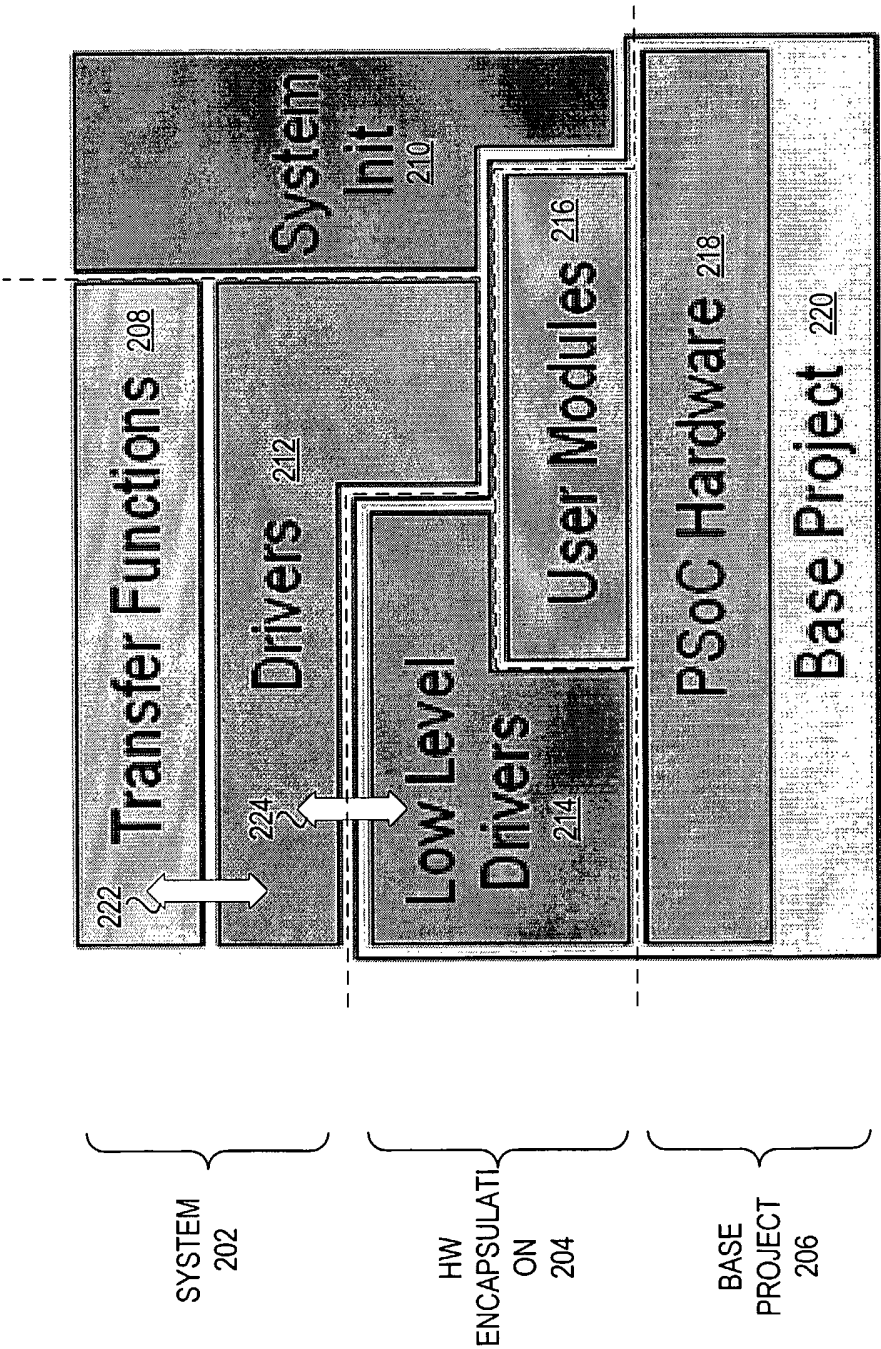
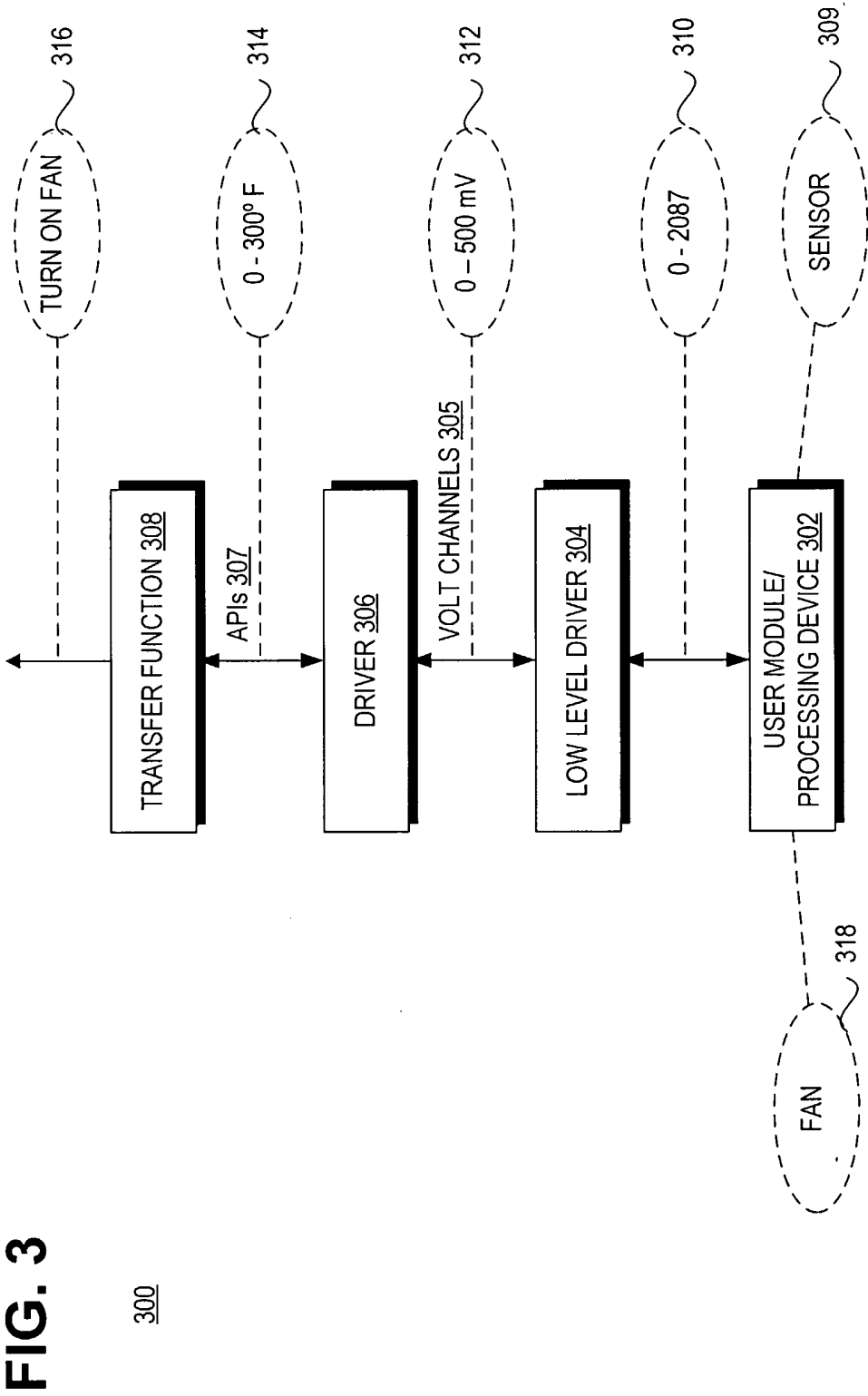
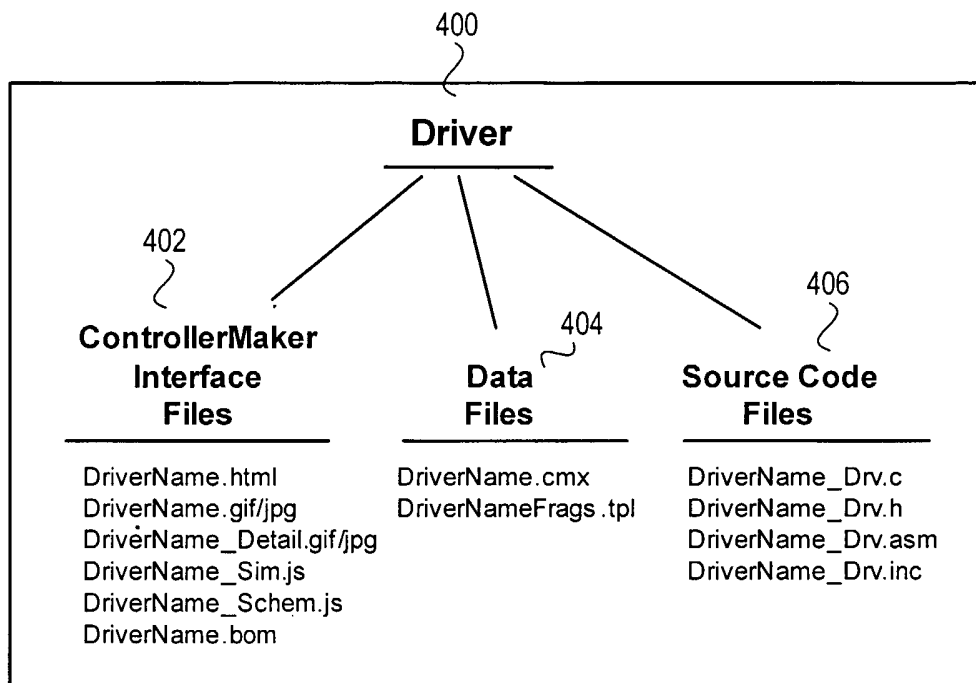
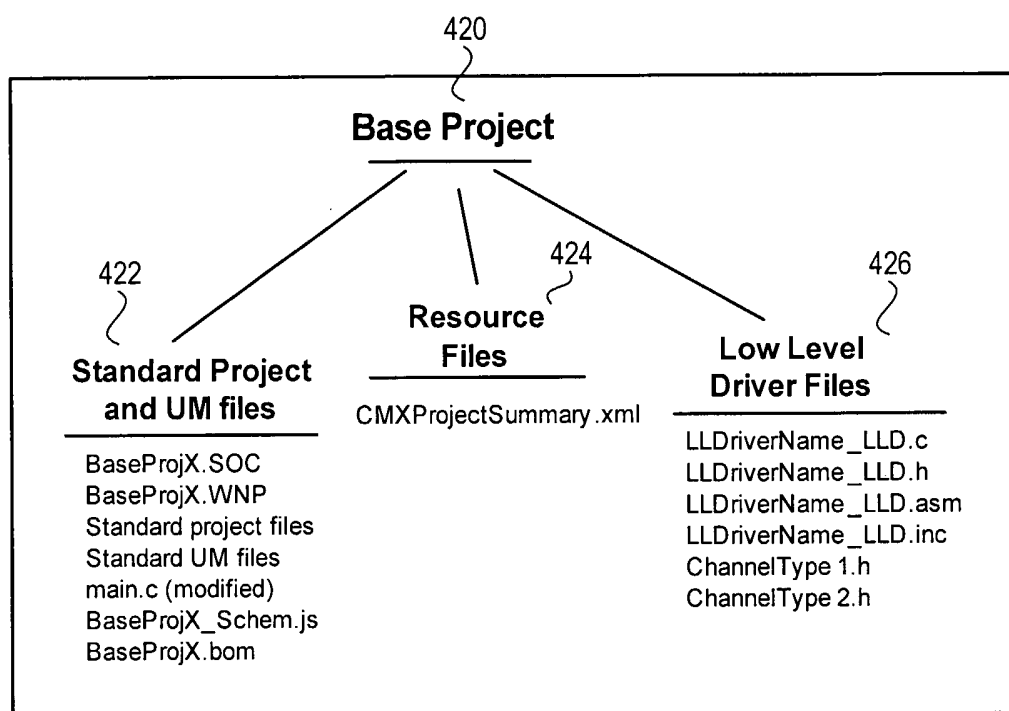


FIG. 2



6/9

**FIG. 4A**

**FIG. 4B**

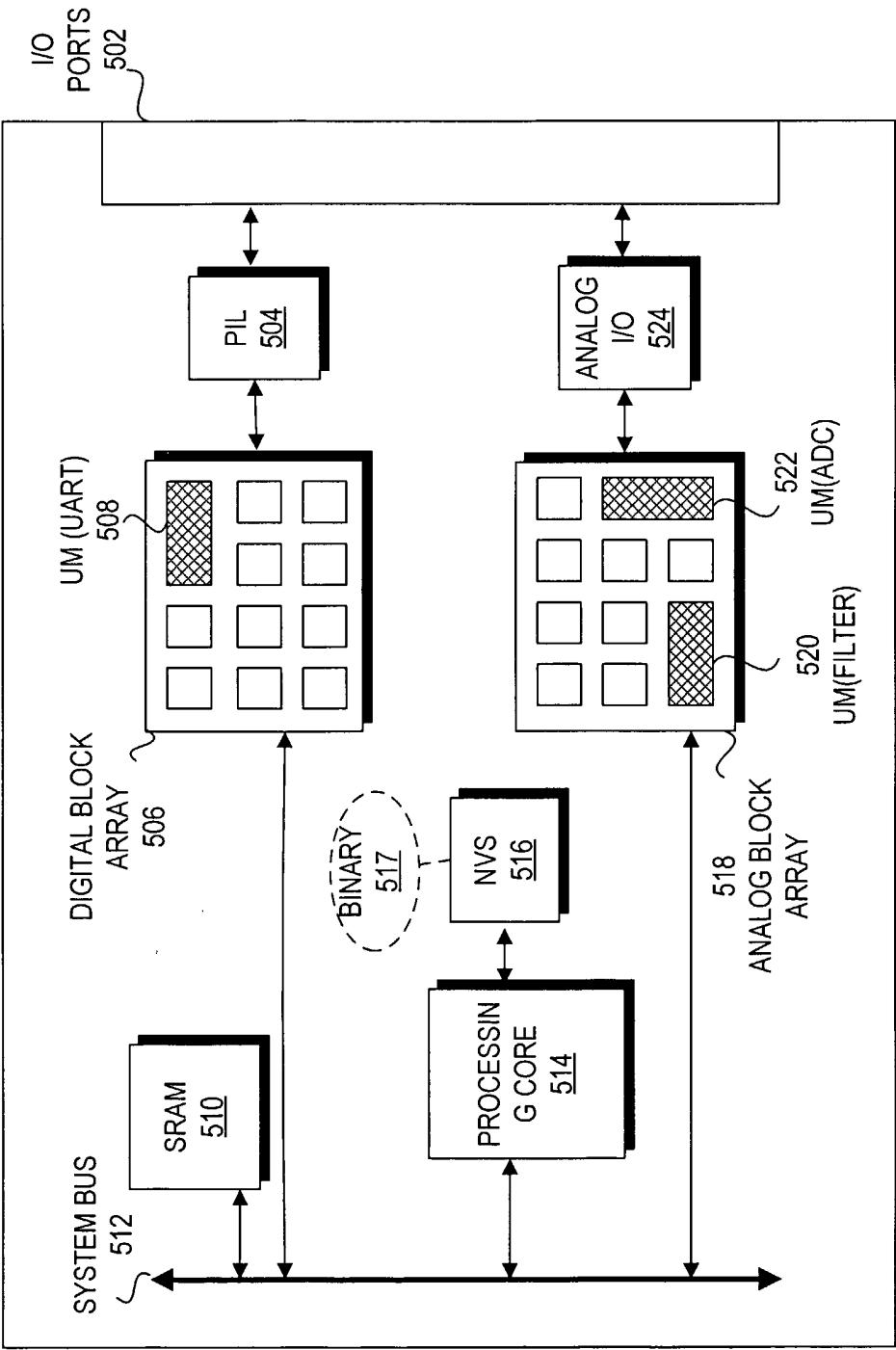
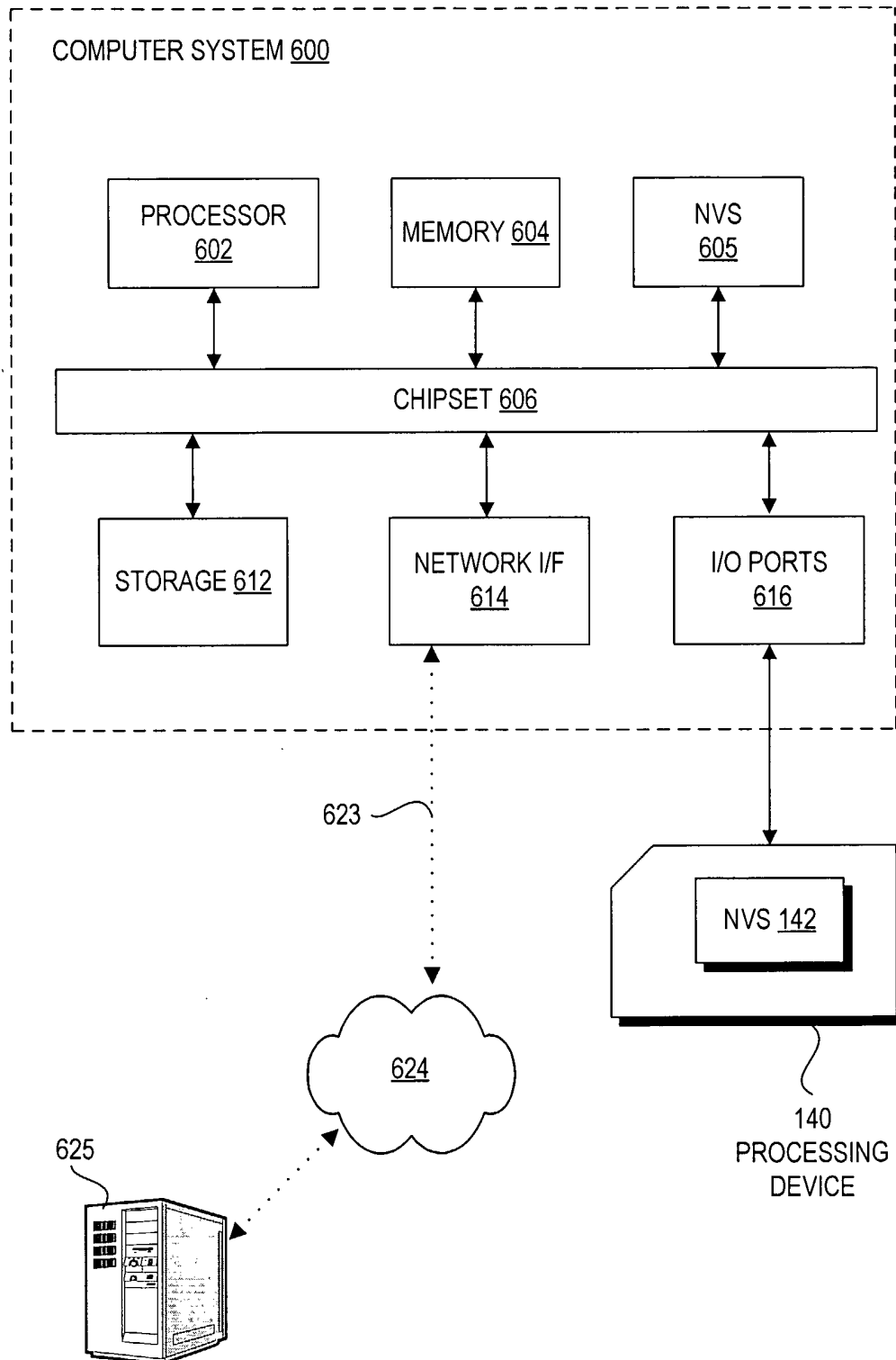


FIG. 5

PROCESSING DEVICE 500

**FIG. 6**

INTERNATIONAL SEARCH REPORT

International application No.

PCT/SE 2005/001011

A. CLASSIFICATION OF SUBJECT MATTER

IPC7: C02F 1/18, B01D 1/00, B01B 1/06

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC7: B01B, B01D, C02F

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

SE,DK,FI,NO classes as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-INTERNAL, WPI DATA, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 0189999 A1 (HVR WATER PURIFICATION AB), 29 November 2001 (29.11.2001) --	1-8
A	SE 510287 C2 (HVR WATER PURIFICATION AB), 10 May 1999 (10.05.1999) -- -----	1-8

☐ Further documents are listed in the continuation of Box C.☒ See patent family annex.

* Special categories of cited documents:

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search

12 October 2005

Date of mailing of the international search report

14-10-2005

Name and mailing address of the ISA/

Swedish Patent Office

Box 5055, S-102 42 STOCKHOLM

Facsimile No. +46 8 666 02 86

Authorized officer

Bengt Christensson/Eö

Telephone No. +46 8 782 25 00

INTERNATIONAL SEARCH REPORT

Information on patent family members

01/10/2005

International application No.

PCT/SE 2005/001011

WO	0189999	A1	29/11/2001	AT	279376	T	15/10/2004
				AU	6092801	A	03/12/2001
				AU	7980100	A	10/05/2001
				CN	1182041	C	29/12/2004
				CN	1430580	A,T	16/07/2003
				DE	60106437	D	00/00/0000
				EA	4021	B	00/00/0000
				EP	1289888	A,B	12/03/2003
				ES	2230315	T	01/05/2005
				HK	1054912	A	13/05/2005
				JP	2003534126	T	18/11/2003
				SE	518388	C	01/10/2002
				SE	0001950	A	23/11/2001
				US	20040222138	A	11/11/2004
SE	510287	C2	10/05/1999	AU	9099598	A	05/04/1999
				DE	19882674	T	16/11/2000
				SE	9801558	A	10/05/1999
