

details found in the physical description. That portion will hide the differences between non-volatile storage on-chip and non-volatile storage off-chip. The myriad differences found in off-chip non-volatile storage hardware (i.e., so-called “flash” parts) will also be declaratively described, as was the case pattern illustrated for the Power Supply example.

**[0099]** Most of the embedded firmware/software part logic depends only on the prototype scheme facets, and not on either the logical description details/values or the physical description. Likewise, most of the embedded firmware/software part logic does not depend on any of the physical description details/values. Only the prototype scheme facets are visible. Almost all of the embedded software/firmware can treat the block of bits as opaque. Only a focused portion of lower-level implementation logic ultimately switches to the appropriate logic (based on the logical and/or physical description details in question).

#### Portability Across Host Hardware Environments via Declarations

**[0100]** The declarative description of the embedded firmware/software host hardware environment provides a novel degree of portability. Accordingly, this invention achieves a unique degree of portability without requiring any change to the firmware/software image (presuming a micro-controller core with instruction set similarities that are expected to appear across a family/line of ASICs). **FIG. 4** illustrates one possible use case that leverages fixed firmware/software image portability according to embodiments of the present invention. As shown in **FIG. 4**, a single, possibly fixed-size, embedded firmware/software part image can be customized for a large variety of products. Initially the fixed firmware/software image is created (“Design Product”, “Install” and “Consult” steps). The Consult step is used to review and consult firmware/software programming guides (API documents and the like). When these steps are complete, the system is implemented, followed by the “Compile, Link, Test” steps (repeated as needed). Typically, this is a cycle that is repeated as many times as necessary to produce a desired executable image.

**[0101]** The initial steps are preferably done only once. The “add” (or edit) descriptions step is done once for each target product. Embodiments of the present invention allows for the automation of this step which results in true, off-the-shelf reuse of a firmware/software part. The last depicted step involves a load of the part into one or more target products. Essentially, this step is the assembly of a final, finished product which includes the re-usable firmware/software part.

#### Example

**[0102]** The following example depicts operation of aspects of the present invention for a given microcontroller core. As shown above, at a physical level, there are abstractions (e.g., general purpose input/output (GPIO) pins) found on any given ASIC (these are the physical pins on the underside of the ASIC chip). On any particular host chip, one (or more) of these pins might be directly available for a logical GPIO signal. Sometimes there are not enough pins for all the various needs (of the embedded firmware/software image). One solution is to limit the number of logical GPIO signal descriptions (and thus reduce the number of

paired physical GPIO pin descriptions). This approach causes the fixed image to sacrifice features and functions in order to conform to the limitations of the host ASIC. Simple changes to the appropriate descriptions suffice.

**[0103]** Alternatively, some product designers (i.e., board designers) may employ an off-ASIC pin expander mechanism—a distinct chip/part (a second, complementary chip). Along with the primary host chip, this expander chip becomes an integral part of the operational host hardware environment. This complication is introduced to this example to illustrate how this invention simply abstracts away such diverse host hardware environment details. In addition, this invention permits this to be done without requiring any change to a given firmware/software image.

**[0104]** Returning to the running Power Supply example, as illustrated above (in Table 1 and Table 2), the Power Supply uses a certain host ASIC pins for the implementation of various logical GPIO signals. These signals are used to implement one logical facet—i.e., the “Power On” behavior. Other GPIO signals (and corresponding GPIO pins) are used to implement other logical behavior facets.

**[0105]** We described above how the Power Supply scheme mapped logical GPIO signals to physical GPIO pins (on some specific host ASIC). We now describe how a logical GPIO signal can map into a logical bus. A collection of GPIO signals (and their corresponding GPIO pins) can be used together to affect a “bit-banged” bus. Some pin expander chips work this way. For instance, four GPIO signals might encode up to sixteen ( $2^4$ ) different signal combinations. In these cases, the GPIOBusFlag listed in Table 2 is true, and so the GPIO bus related attributes of Table 2 are not moot.

**[0106]** Thus can be seen some of descriptive power of the use of the block of bits. It can be used to describe a series of simple GPIO interfaces. It can be used to describe which physical ASIC pins are used (for various GPIO signals) with the Power Supply in question. It can be used to describe how multiple GPIO signals are used together to form a crude bit-banged bus interface (to either a Power Supply or to an off-chip pin expander which is itself wired to a power supply).

**[0107]** The Power Supply prototype scheme comes with a generic, default implementation. This implementation logic drives off the contents of the block of bits. If the default behaviors are inappropriate for some specific Power Supply, each behavioral facet can be replaced with alternate behaviors (which are available for as-needed use with the Power Supply scheme). Simple edits to the prototype scheme can replace any of the default prototype behaviors.

**[0108]** Only a few, isolated, focused bits of embedded software/firmware logic need to know anything about this particular block of bits interpretation. All of the logic associated with the Power Supply scheme form a natural module. This module contains the only logic that should have any knowledge of how the block of bits should be interpreted (for a Power Supply scheme instance). While some applications of this invention might want to dynamically load modules, a given range of desired applicability can map to a certain, fixed set of modules. Such modules can be pre-loaded into a fixed-size executable image. Once this is done, declarative changes to the description block are all