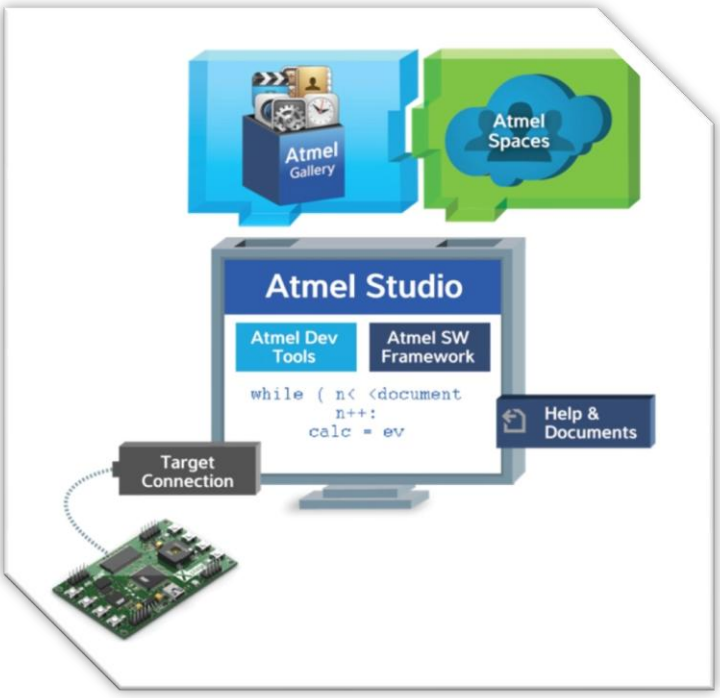
Beyond the IDE

Introducing a platform to facilitate reliable and highly productive embedded developments

Author:
Joerg Bertholdt, Director of Marketing, MCU Tools and Software, Atmel Corporation



Beyond the IDE

Software plays an increasingly dominant role in the success of an end product as the processing capability of embedded systems increases. Developers working on enterprise or web-based software have benefited from integrated tools dealing with software complexity for year, a requirement that is now strongly emerging for emerging, microcontroller based products.

As microcontroller (MCU) vendors have integrated more functions into their platforms, embedded developers realize they need to achieve higher levels of productivity, which can only be achieved with increasing the degree of code reuse. Marketing continues to demand faster responsiveness from engineering as the need for an agile development operation is firmly linked to an organization's success. Moving to a new MCU may involve downloading new support code archives and documentation and learning new APIs. Code that would otherwise be identical still requires rewriting because of changes in the application programming interface (API) between MCU versions. However, this is problematic in an environment that requires scaling and tuning on the fly in order to react quickly to changes in demand.

In today's complex and feature-rich embedded developments, a comprehensive MCU software framework and a method of easily integrating additional software libraries and tools are becoming increasingly more important for design teams.

Software Frameworks

In the enterprise application development, frameworks are now very popular and have enabled programmers to re-use common algorithms and functions for various projects. However, it is not until recently that designers have realized the true advantages of frameworks, and the approach for embedded development frameworks is just becoming established. For instance, new IDEs such as Atmel Studio 6 not only provide core language syntax support, but improved context-sensitive hints on parameters and functions from within a framework.

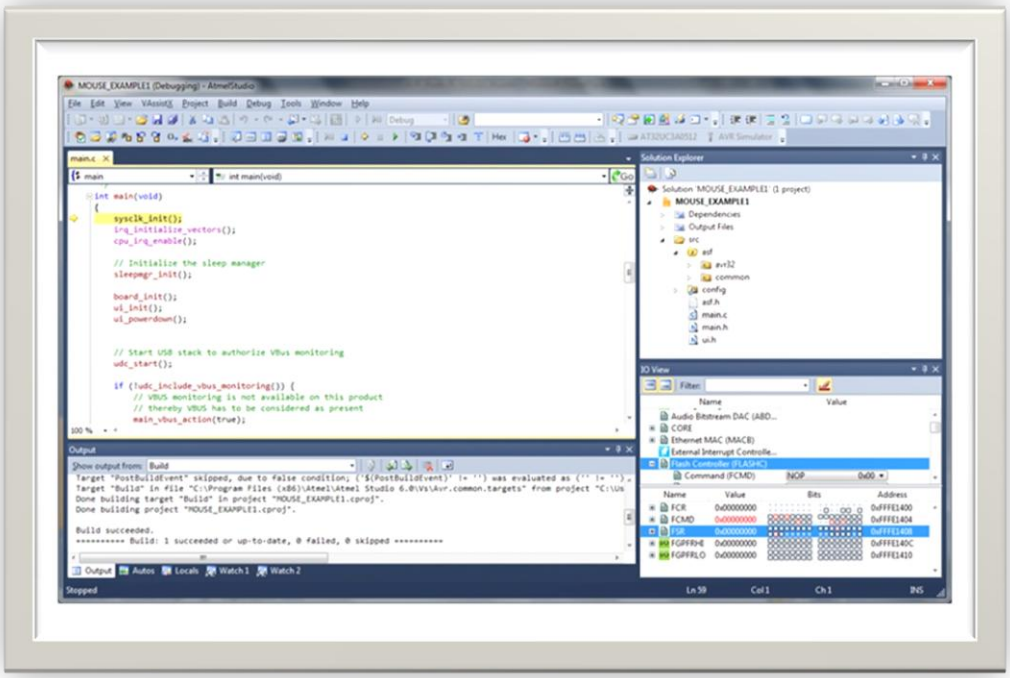


Figure 1: Context aware code editing in Atmel Studio 6

An example of this approach is Atmel's Software Framework (ASF). Used in conjunction with the Atmel Studio 6 IDE, it facilitates a top-down design approach to embedded systems development that fully leverages the accumulated intellectual property (IP) and expertise of the organization by avoiding the requirement to rewrite significant portions of the code for each port to a different MCU variant or architecture.

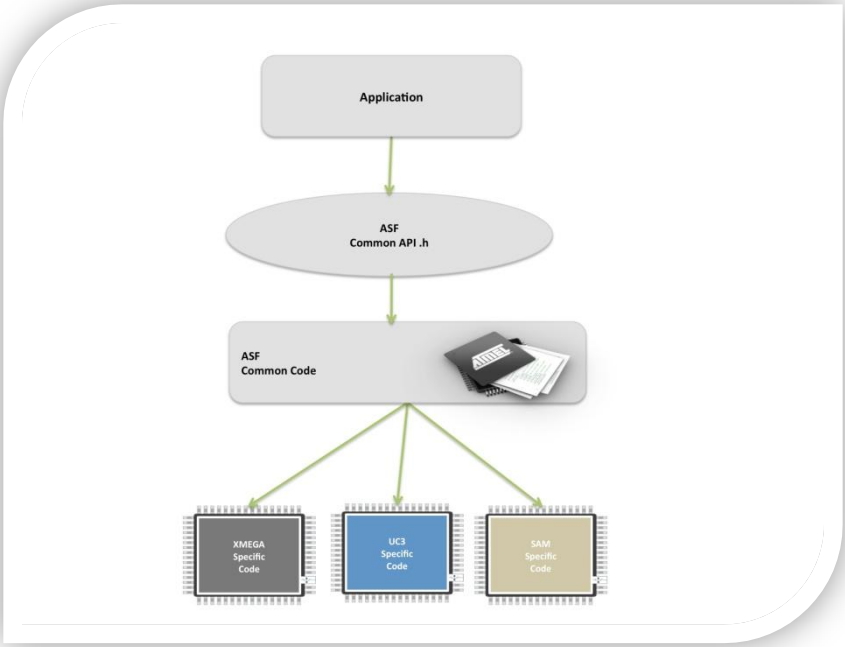


Figure 2. Example of a software framework – Atmel Software Framework architectural design

ASF has been architected from the ground-up to provide a clean interface between user application code and the software stacks that enable the application to run on a variety of different embedded target MCUs. ASF provides everything required between the application and the hardware design.

The functions and code examples that use them are all optimized for code size for each target architecture and work with a range of ANSI-C compilers. The ASF code is also architecture-optimized by Atmel experts, ensuring not just high performance but low power. The functions take full advantage of Atmel MCU features such as low-power modes and the Peripheral Event System.

Chip-specific features in the protocol stacks and functions are used in a way that maximizes portability for application-level code. But the functions are implemented using a common API that abstracts away the target-specific details, allowing developers to take code developed on one Atmel device and compile it for a new Atmel target practically unchanged.

For example, the API uses an intuitive format in which devices are programmed using function calls that follow a consistent naming convention: <device>_init(); <device>_enable(); <device>_disable(); <device>_start(); <device>_stop(); <device>_read(); <device>_write().

ASF modules are arranged in a layered architecture that allows application code to call functions that are most appropriate to the task at hand. This architecture also makes it easy to add support for complex protocols, such as USB, to a product. There are four types of layers: component, service, peripheral and board.

The component and service modules are called directly by the application, unless it needs direct access to low-level device functions provided by the peripheral and board layers. The service layer provides the user with application-oriented software stacks such as USB class drivers, file systems, architecture-optimized digital signal processing functions and graphics libraries. This layer also takes advantage of the MCU's hardware features. Components are high-level drivers that provide intuitive and direct control over MCU and board-level peripherals, such as display, sensors and wireless interfaces. The code in the component layer is written with a focus on providing the functionality that a typical user will need in each of the on-chip peripherals. If the application calls for a peripheral to operate in a way that is not directly supported by the component layer, it is easy for the user to modify the existing source code, or add an extra function to the API.

The component and service modules communicate with low-level drivers in the peripheral layer that provide register-level control over hardware interfaces. Applications can also call these drivers directly to facilitate tight hardware integration in a way that maximizes portability between members of the Atmel MCU portfolio.

Finally, the board module provides the hardware view of the MCU in its target environment. The board code abstracts the modules above the board from the physical wiring and initialization functions that take care of I/O and external devices. The board code also identifies which board features are available to the modules higher up in the software stack.

The board definition provides a convenient way to assign digital and analog peripherals to each I/O pin. An application, such as an audio interface for a PC, may call for the allocation of a USB port, ADC and DAC channels, an SPI port and several general-purpose digital I/O channels that, on the PCB, are connected to buttons and LEDs.

For example, an LED that shows whether the MCU is asleep or active might be called GPIO4. Instead of having to remember this name, the developer can assign the more logical name of ACTIVITY_LED to the bitmask that is used to access the specific bits within the actual I/O port's register that controls the LED state. Once defined, this constant can be used consistently throughout the application to provide access from I/O function calls to the LED.

To ensure consistency in the way that module APIs are used, code provided by the ASF uses standard techniques for initialization and other management tasks. This helps reduce training time on new functions as programmers can expect to use API calls in a similar way to known modules when they incorporate a new function. For example, starting and stopping a module is normally performed by module_start(…) and module_stop(…) API calls. When encountering an A/D converter module, the programmer can expect to use functions of the type adc_start(…) and adc_stop(…).

Function-call conventions and parameters remain the same across the range of Atmel MCUs, including the Atmel AVR UC3, megaAVR®, AVR XMEGA® and the SAM Cortex™-M processor-based product lines. The ASF takes full advantage of the IDE and the structure of C code to ensure maximum applications compatibility—even across architectures, allowing common code development for 8-bit and 32-bit targets, even using different compilers.

For example, differences between compilers and the way they interpret information in the source files are absorbed into architecture-specific header files. This structure allows ASF to support GCC and IAR compilers for both 8-bit and 32-bit AVR and ARM® processor-based MCUs. Similarly, differences between peripherals across the various MCUs in the Atmel portfolio are absorbed into header and C source files using rules developed by the ASF architects to ensure maximum commonality and portability. In drawing up the ASF, the software architects employed a series of rules and techniques that ensure common application code does not have to be changed to deal with a change in target MCU (Figure 3).

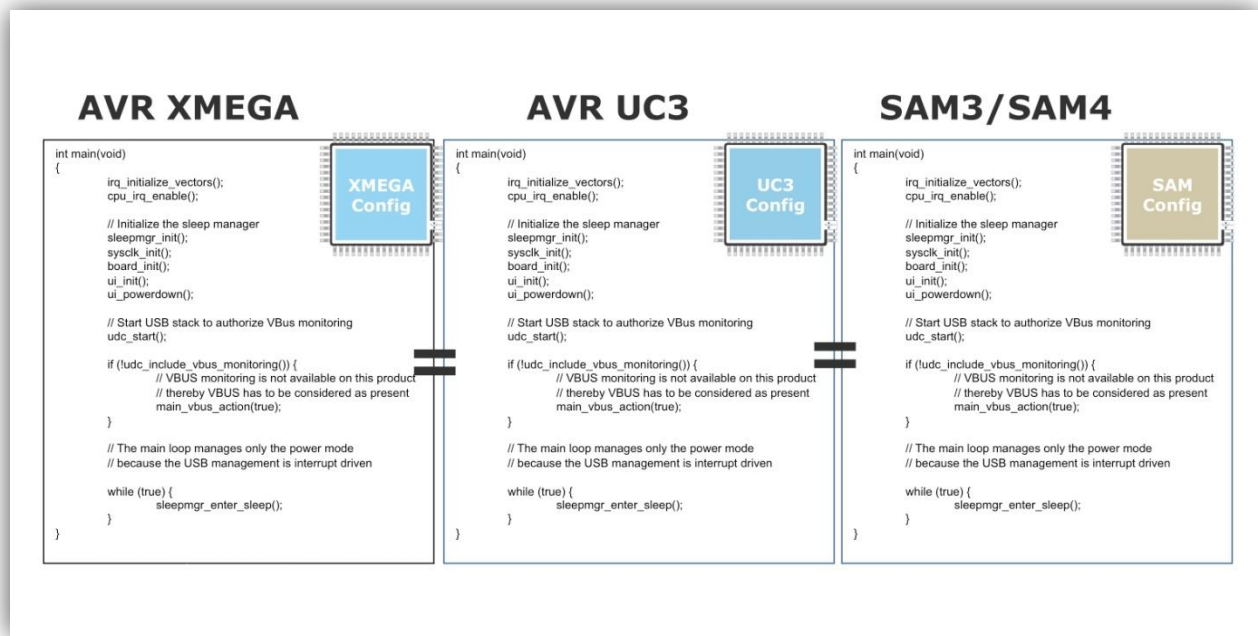


Figure 3. Identical C-code applications across Atmel's AVR, XMEGA, UC3, SAM3, SAM4 MCUs

This commonality assures ease of scaling as market demands change. It also streamlines the process of moving from prototype to production. Very often, developers will choose a more flexible, higher performance device for prototyping, so they can be assured of having sufficient headroom for the application code and potential changes during the project. As the project nears completion, it may become clear that there is potential for using a lower cost part for production – one that may even use a different core architecture. Using this framework-based approach makes it extremely straightforward for a different target to be accommodated quickly and provides marketing the engineering agility they desire.

Accommodating Code Libraries and Tools

Any complex embedded application is unlikely to be written totally from scratch. In order to speed development, there are a host of third party code libraries, tool chain extensions, configuration routines and application-specific or host related middleware that might be incorporated. The Internet provides free access to these materials but from a development process perspective they can become difficult to manage since they are typically outside the IDE environment.

Similar to how App Stores have become a popular method of adding media content and applications to tablets and smartphones, this concept is well suited to complement an IDE. For example, Atmel Gallery provides a moderated App Store feature for Atmel Studio 6.0 extensions. This method allows for free, evaluation and paid-for content to be easily and quickly integrated into an embedded development. Developers can be confident that the extensions are moderated for content quality from Atmel-certified third party development partners. Atmel can also provide content in this way. Atmel Gallery content is grouped by category. Example categories include Toolchain, Code Editing and Management, Analysis/Visualization tools, Application Tools and RTOS/Middleware.

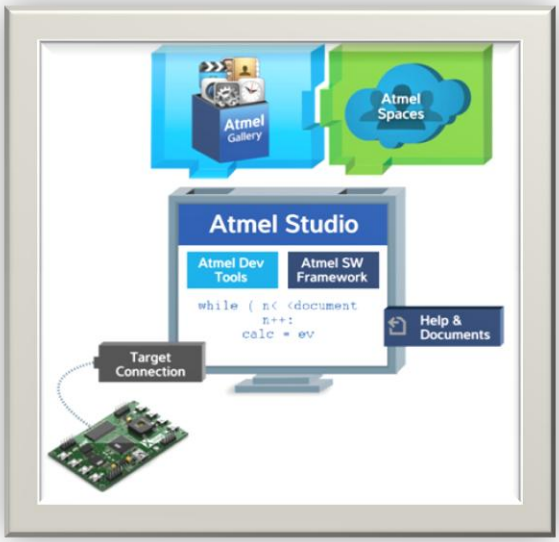


Figure 4 – Atmel Studio Platform

One of the RTOSes available within Atmel Gallery is FreeRTOS. By accessing FreeRTOs in Atmel Gallery, designers can implement a hassle-free deployment in their application without having to worry about driver integration. Since this is integrated into ASF, it comes complete with a project configuration wizard and example projects to ease the application development process.

Truly Integrated Approach

A truly integrated development platform includes a combination of software and hardware including the software framework and an app store. Recently, Atmel launched the Xplained Pro evaluation kits, a development board and kit that works seamlessly with Atmel's IDE, software framework and Atmel Gallery. The new kits feature a range of professional ARM-Cortex-M4 based Atmel SAM4 microcontroller boards that are complemented by optional interface, display and prototyping boards. These boards are fully supported across Atmel Studio 6, Atmel Software Framework and Atmel Gallery, and provides developers with immediate access to over 2,000 ready-to-run project examples. Using the Atmel development platform of tools, designers can readily prototype, test and bring their designs to market with ease and on-time.

Integrated Development Tools Platform

Moving beyond the traditional integrated development environment, the platform-based approach yields yet further developer productivity and efficiency. By combining the editor, compiler and debug functions with quick and easy access to a host of libraries, middleware and specialist tools the integrated development tools platform approach yields efficiencies across the whole design, development, test, and prototype process.

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Bldg.
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370