IN THE UNITED STATES DISTRICT COURT
FOR THE NORTHERN DISTRICT OF TEXAS
DALLAS DIVISION

LUCIO DEVELOPMENT LLC,	§	
	§	
Plaintiff,	§	Case No:
	§	
vs.	§	PATENT CASE
	§	
MICROCHIP TECHNOLOGY INCORPORATED	§	
	§	
Defendant.	§	
	§	

COMPLAINT

Plaintiff Lucio Development LLC ("Plaintiff" or "Lucio") files this Complaint against Microchip Technology Incorporated ("Defendant" or "MTI") for infringement of United States Patent No. 7,069,546 (hereinafter "the '546 Patent").

PARTIES AND JURISDICTION

1. This is an action for patent infringement under Title 35 of the United States Code. Plaintiff is seeking injunctive relief as well as damages.

2. Jurisdiction is proper in this Court pursuant to 28 U.S.C. §§ 1331 (Federal Question) and 1338(a) (Patents) because this is a civil action for patent infringement arising under the United States patent statutes.

3. Plaintiff is a Texas limited liability company with its office address at 555 Republic Dr., Suite 200, Plano, Texas 75074.

4. On information and belief, Defendant is a Delaware corporation with a place of business at 2355 W. Chandler Blvd., Chandler, AZ 85224. Microchip Technology may be served via its registered agent for service of process: The Corporation Trust Company,

Corporation Trust Center 1209 Orange St., Wilmington, Delaware 19801.

5. This Court has personal jurisdiction over Defendant because Defendant has committed, and continues to commit, acts of infringement in this District, has conducted business in this District, and/or has engaged in continuous and systematic activities in this District.

6. On information and belief, Defendant's instrumentalities that are alleged herein to infringe were and continue to be used, imported, offered for sale, and/or sold in this District.

VENUE

7. Venue is proper in this District pursuant to 28 U.S.C. §1400(b) because acts of infringement are occurring in this District and Defendant has a regular and established place of business in this District. For instance, on information and belief, Defendant has a regular and established place of business at 16200 Addison Rd. # 260, Addison, TX 75001.

COUNT I
(INFRINGEMENT OF UNITED STATES PATENT NO. 7,069,546)

8. Plaintiff incorporates paragraphs 1 through 7 herein by reference.

9. This cause of action arises under the patent laws of the United States and, in particular, under 35 U.S.C. §§ 271, *et seq*.

10. Plaintiff is the owner by assignment of the '546 Patent with sole rights to enforce the '546 Patent and sue infringers.

11. A copy of the '546 Patent, titled "Generic Framework for Embedded Software Development," is attached hereto as Exhibit A.

12. The '546 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code.

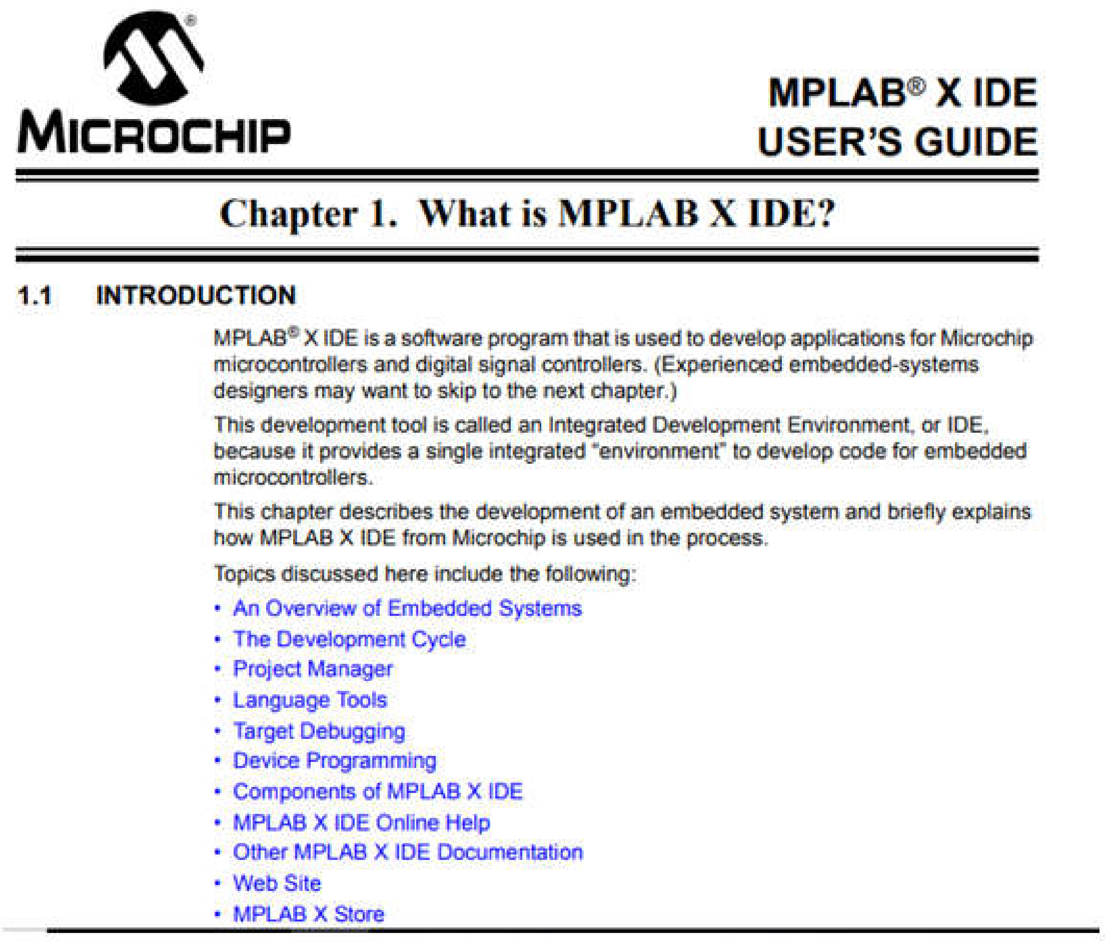
13. On information and belief, Defendant has infringed and continues to infringe

one or more claims, including at least Claim 1, of the ‘546 Patent by making, using, importing, selling, and/or offering for sale a software platform for embedded software development, which is covered by at least Claim 1 of the ‘546 Patent. Defendant has infringed and continues to infringe the ‘546 Patent directly in violation of 35 U.S.C. § 271.

14. Defendant, sells, offers to sell, and/or uses embedded software development packages including, without limitation, MPLAB X IDE, Atmel Studio 7, Atmel START, and any similar products (“Product”), which infringe at least Claim 1 of the ‘546 Patent.

MPLAB X IDE

15. MPLAB X IDE is a software program used to develop applications for microchip microcontrollers and digital signal controllers. It also includes MPLAB Harmony Configurator (MHC) as a plug-in, which accelerates development of highly capable and reusable PIC32 embedded firmware applications. MPLAB X IDE provides one or more generic application handler programs, each such program comprising computer program code for performing generic application functions common to multiple types of hardware modules used in a communication system. For example, MPLAB Harmony Configurator (MHC) uses a plug-in in MPLAB X IDE comprising a Hardware Abstraction Layer (HAL) containing Hardware Access Functions for defined functions (“generic application handler”). Hardware Access Functions include source code comprising functions and data structure, which are uniform across all supported MPLAB Harmony hardware modules. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



MICROCHIP

MPLAB® X IDE
USER'S GUIDE

Chapter 1. What is MPLAB X IDE?

1.1 INTRODUCTION

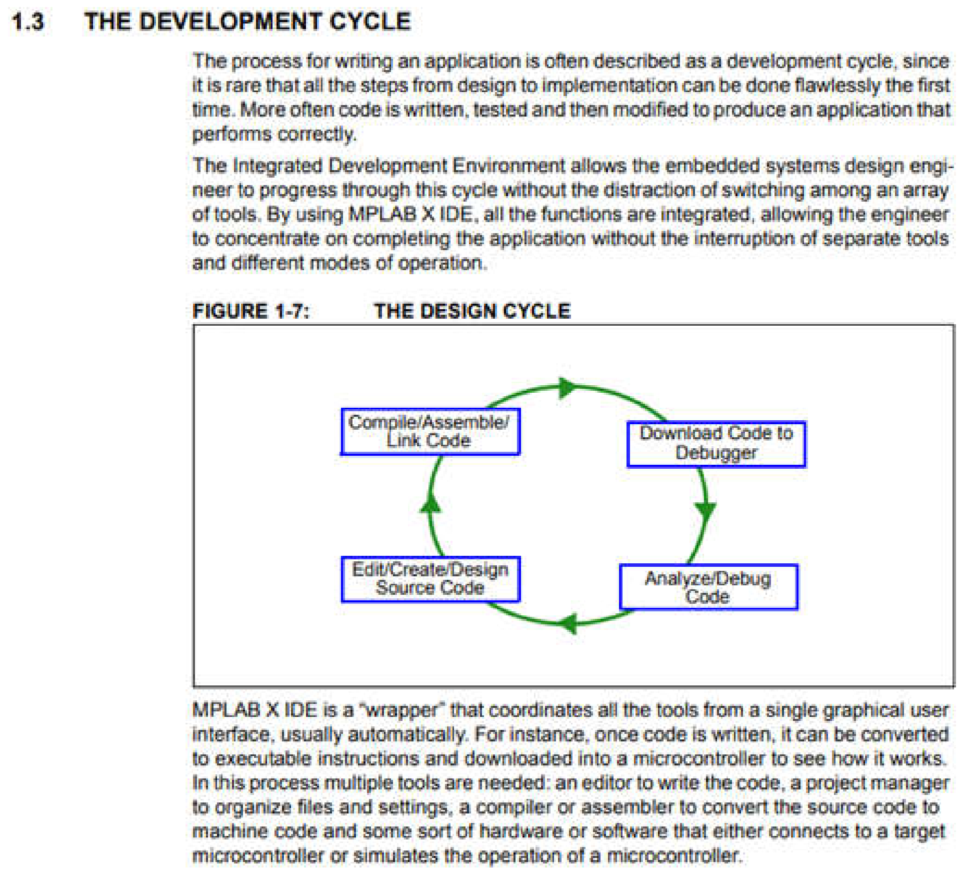
MPLAB® X IDE is a software program that is used to develop applications for Microchip microcontrollers and digital signal controllers. (Experienced embedded-systems designers may want to skip to the next chapter.)

This development tool is called an Integrated Development Environment, or IDE, because it provides a single integrated "environment" to develop code for embedded microcontrollers.

This chapter describes the development of an embedded system and briefly explains how MPLAB X IDE from Microchip is used in the process.

Topics discussed here include the following:

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/50002027D.pdf, page 13.

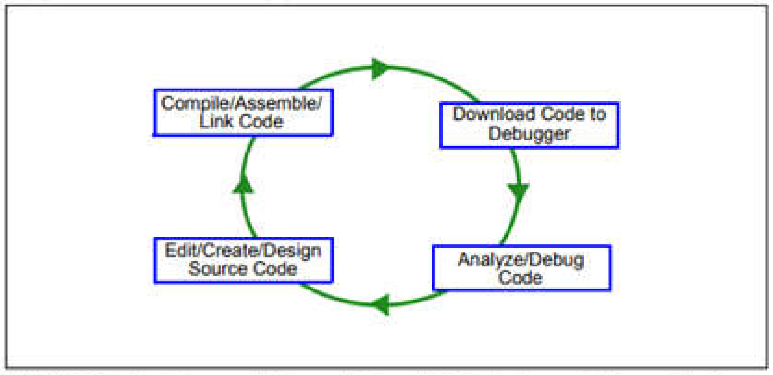


1.3 THE DEVELOPMENT CYCLE

The process for writing an application is often described as a development cycle, since it is rare that all the steps from design to implementation can be done flawlessly the first time. More often code is written, tested and then modified to produce an application that performs correctly.

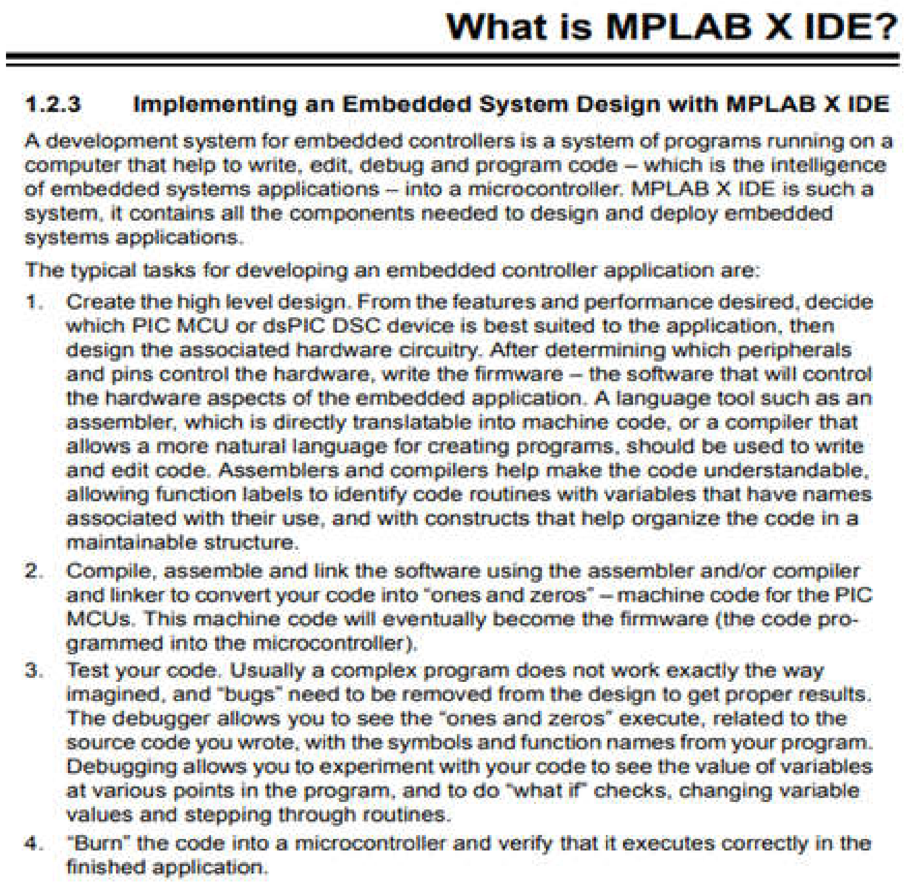
The Integrated Development Environment allows the embedded systems design engineer to progress through this cycle without the distraction of switching among an array of tools. By using MPLAB X IDE, all the functions are integrated, allowing the engineer to concentrate on completing the application without the interruption of separate tools and different modes of operation.

FIGURE 1-7: THE DESIGN CYCLE


Compile/Assemble/ Link Code
Download Code to Debugger
Analyze/Debug Code
Edit/Create/Design Source Code

MPLAB X IDE is a "wrapper" that coordinates all the tools from a single graphical user interface, usually automatically. For instance, once code is written, it can be converted to executable instructions and downloaded into a microcontroller to see how it works. In this process multiple tools are needed: an editor to write the code, a project manager to organize files and settings, a compiler or assembler to convert the source code to machine code and some sort of hardware or software that either connects to a target microcontroller or simulates the operation of a microcontroller.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/50002027D.pdf, page 21.



What is MPLAB X IDE?

1.2.3 Implementing an Embedded System Design with MPLAB X IDE

A development system for embedded controllers is a system of programs running on a computer that help to write, edit, debug and program code – which is the intelligence of embedded systems applications – into a microcontroller. MPLAB X IDE is such a system, it contains all the components needed to design and deploy embedded systems applications.

The typical tasks for developing an embedded controller application are:

1. Create the high level design. From the features and performance desired, decide which PIC MCU or dsPIC DSC device is best suited to the application, then design the associated hardware circuitry. After determining which peripherals and pins control the hardware, write the firmware – the software that will control the hardware aspects of the embedded application. A language tool such as an assembler, which is directly translatable into machine code, or a compiler that allows a more natural language for creating programs, should be used to write and edit code. Assemblers and compilers help make the code understandable, allowing function labels to identify code routines with variables that have names associated with their use, and with constructs that help organize the code in a maintainable structure.
2. Compile, assemble and link the software using the assembler and/or compiler and linker to convert your code into "ones and zeros" – machine code for the PIC MCUs. This machine code will eventually become the firmware (the code programmed into the microcontroller).
3. Test your code. Usually a complex program does not work exactly the way imagined, and "bugs" need to be removed from the design to get proper results. The debugger allows you to see the "ones and zeros" execute, related to the source code you wrote, with the symbols and function names from your program. Debugging allows you to experiment with your code to see the value of variables at various points in the program, and to do "what if" checks, changing variable values and stepping through routines.
4. "Burn" the code into a microcontroller and verify that it executes correctly in the finished application.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/50002027D.pdf, page 19.

1.2 Components of Hardware Abstraction Layer

Figure 1 shows a representative block diagram of the Hardware Abstraction Layer in relation to the MC Application Framework and the customer application.

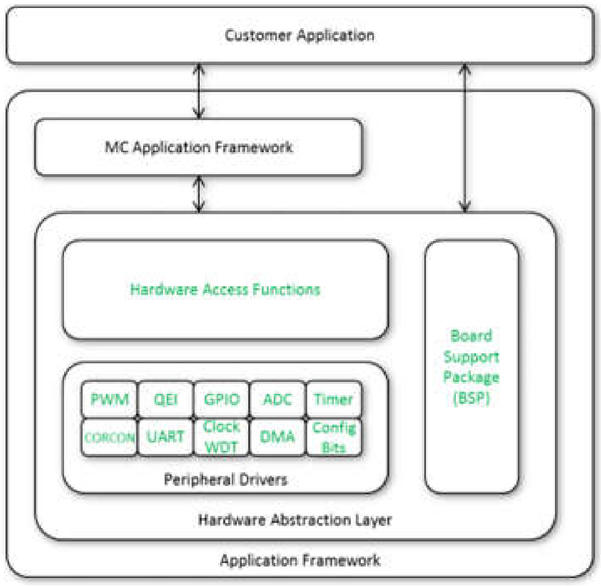
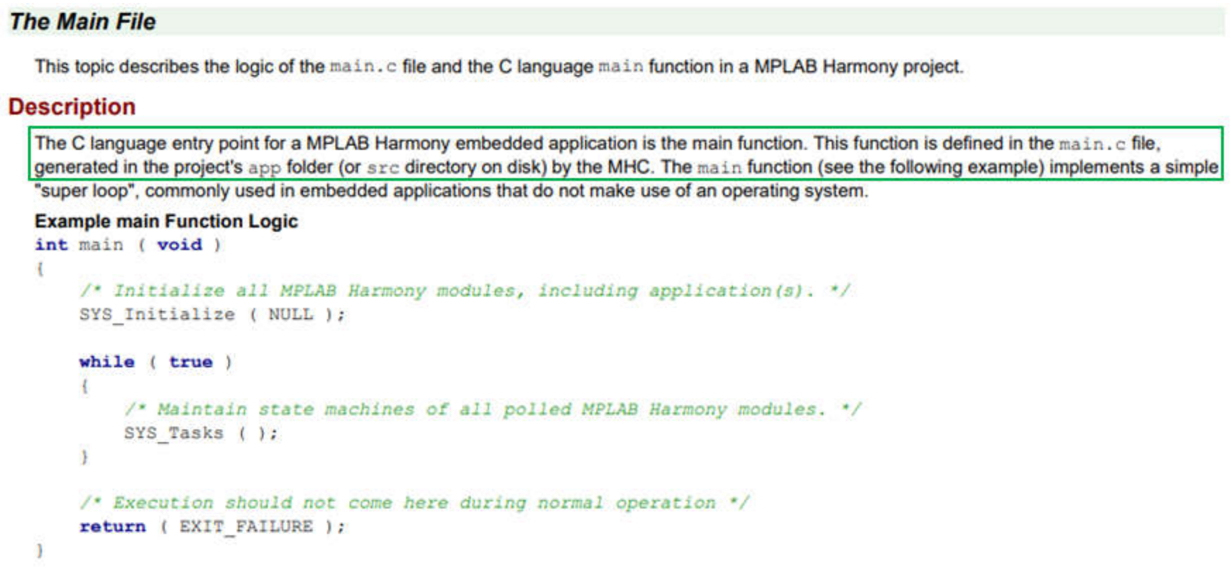


Figure 1: Block diagram of Hardware Abstraction Layer

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/hardware-abstraction-layer.pdf, page 5.



The Main File

This topic describes the logic of the `main.c` file and the C language `main` function in a MPLAB Harmony project.

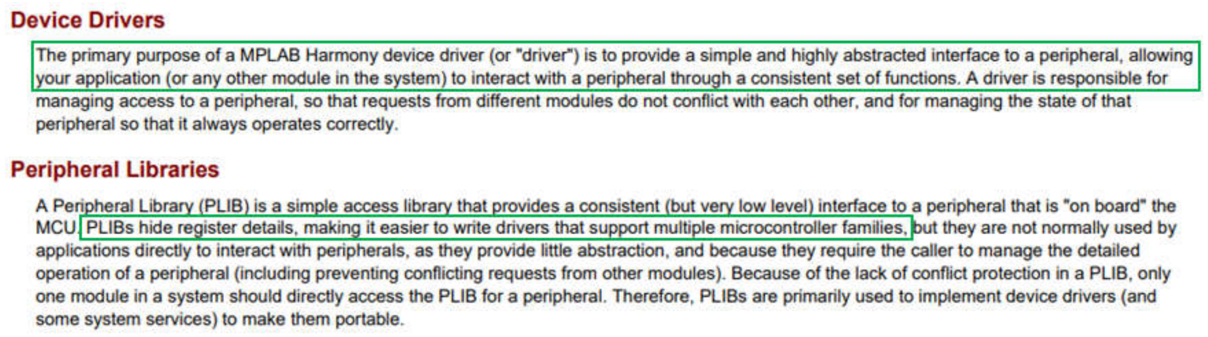
Description

The C language entry point for a MPLAB Harmony embedded application is the main function. This function is defined in the `main.c` file, generated in the project's `app` folder (or `src` directory on disk) by the MHC. The `main` function (see the following example) implements a simple "super loop", commonly used in embedded applications that do not make use of an operating system.

Example main Function Logic

```
int main ( void )
{
    /* Initialize all MPLAB Harmony modules, including application(s). */
    SYS_Initialize ( NULL );

    while ( true )
    {
        /* Maintain state machines of all polled MPLAB Harmony modules. */
        SYS_Tasks ( );
    }

    /* Execution should not come here during normal operation */
    return ( EXIT_FAILURE );
}
```

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Understanding%20MPLAB%20Harmony_v2.04.pdf, page 17.



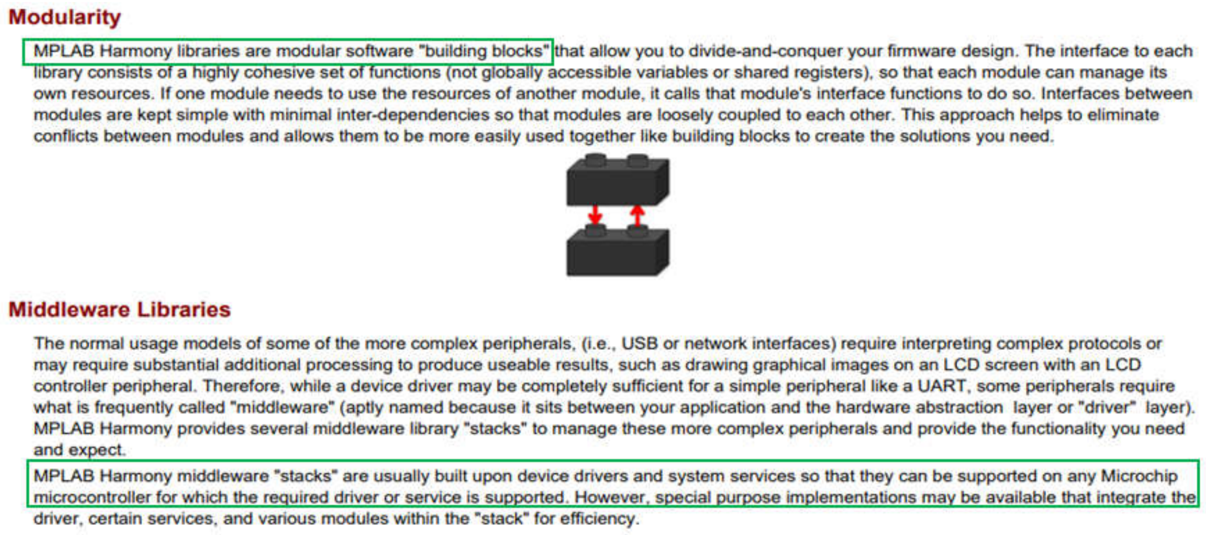
Device Drivers

The primary purpose of a MPLAB Harmony device driver (or "driver") is to provide a simple and highly abstracted interface to a peripheral, allowing your application (or any other module in the system) to interact with a peripheral through a consistent set of functions. A driver is responsible for managing access to a peripheral, so that requests from different modules do not conflict with each other, and for managing the state of that peripheral so that it always operates correctly.

Peripheral Libraries

A Peripheral Library (PLIB) is a simple access library that provides a consistent (but very low level) interface to a peripheral that is "on board" the MCU. PLIBs hide register details, making it easier to write drivers that support multiple microcontroller families, but they are not normally used by applications directly to interact with peripherals, as they provide little abstraction, and because they require the caller to manage the detailed operation of a peripheral (including preventing conflicting requests from other modules). Because of the lack of conflict protection in a PLIB, only one module in a system should directly access the PLIB for a peripheral. Therefore, PLIBs are primarily used to implement device drivers (and some system services) to make them portable.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Understanding%20MPLAB%20Harmony_v2.04.pdf, page 10.



Modularity

MPLAB Harmony libraries are modular software "building blocks" that allow you to divide-and-conquer your firmware design. The interface to each library consists of a highly cohesive set of functions (not globally accessible variables or shared registers), so that each module can manage its own resources. If one module needs to use the resources of another module, it calls that module's interface functions to do so. Interfaces between modules are kept simple with minimal inter-dependencies so that modules are loosely coupled to each other. This approach helps to eliminate conflicts between modules and allows them to be more easily used together like building blocks to create the solutions you need.

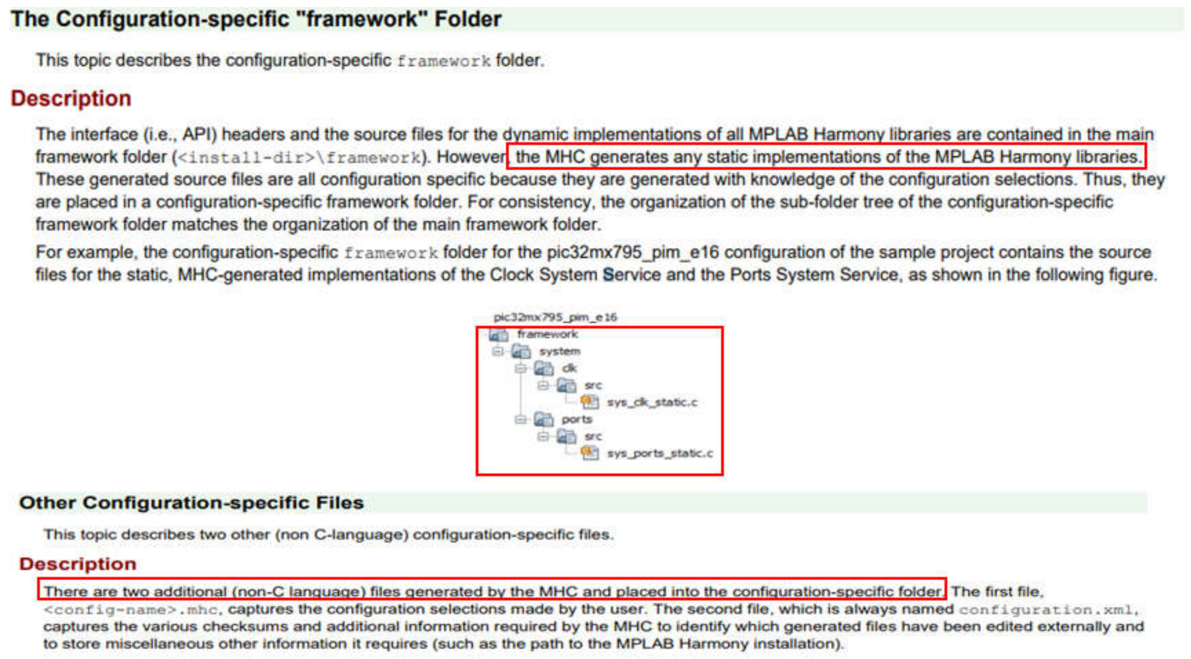
Middleware Libraries

The normal usage models of some of the more complex peripherals, (i.e., USB or network interfaces) require interpreting complex protocols or may require substantial additional processing to produce useable results, such as drawing graphical images on an LCD screen with an LCD controller peripheral. Therefore, while a device driver may be completely sufficient for a simple peripheral like a UART, some peripherals require what is frequently called "middleware" (aptly named because it sits between your application and the hardware abstraction layer or "driver" layer). MPLAB Harmony provides several middleware library "stacks" to manage these more complex peripherals and provide the functionality you need and expect.

MPLAB Harmony middleware "stacks" are usually built upon device drivers and system services so that they can be supported on any Microchip microcontroller for which the required driver or service is supported. However, special purpose implementations may be available that integrate the driver, certain services, and various modules within the "stack" for efficiency.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Understanding%20MPLAB%20Harmony_v2.04.pdf, page 11.

16. MPLAB X IDE generates specific application handler code to associate the generic application functions with specific functions of a device driver for at least one of the types of the hardware modules. For example, in addition to the generic drivers and HAL, MPLAB X IDE includes a plug-in MPLAB Harmony Configurator (MHC) comprising a specific application handler code that is specific to the application and specific to particular microcontroller families. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



The Configuration-specific "framework" Folder

This topic describes the configuration-specific `framework` folder.

Description

The interface (i.e., API) headers and the source files for the dynamic implementations of all MPLAB Harmony libraries are contained in the main framework folder (`<install-dir>\framework`). However, the MHC generates any static implementations of the MPLAB Harmony libraries. These generated source files are all configuration specific because they are generated with knowledge of the configuration selections. Thus, they are placed in a configuration-specific framework folder. For consistency, the organization of the sub-folder tree of the configuration-specific framework folder matches the organization of the main framework folder.

For example, the configuration-specific `framework` folder for the pic32mx795_pim_e16 configuration of the sample project contains the source files for the static, MHC-generated implementations of the Clock System Service and the Ports System Service, as shown in the following figure.

```
pic32mx795_pim_e16
framework
  system
    clk
      src
        sys_clk_static.c
    ports
      src
        sys_ports_static.c
```

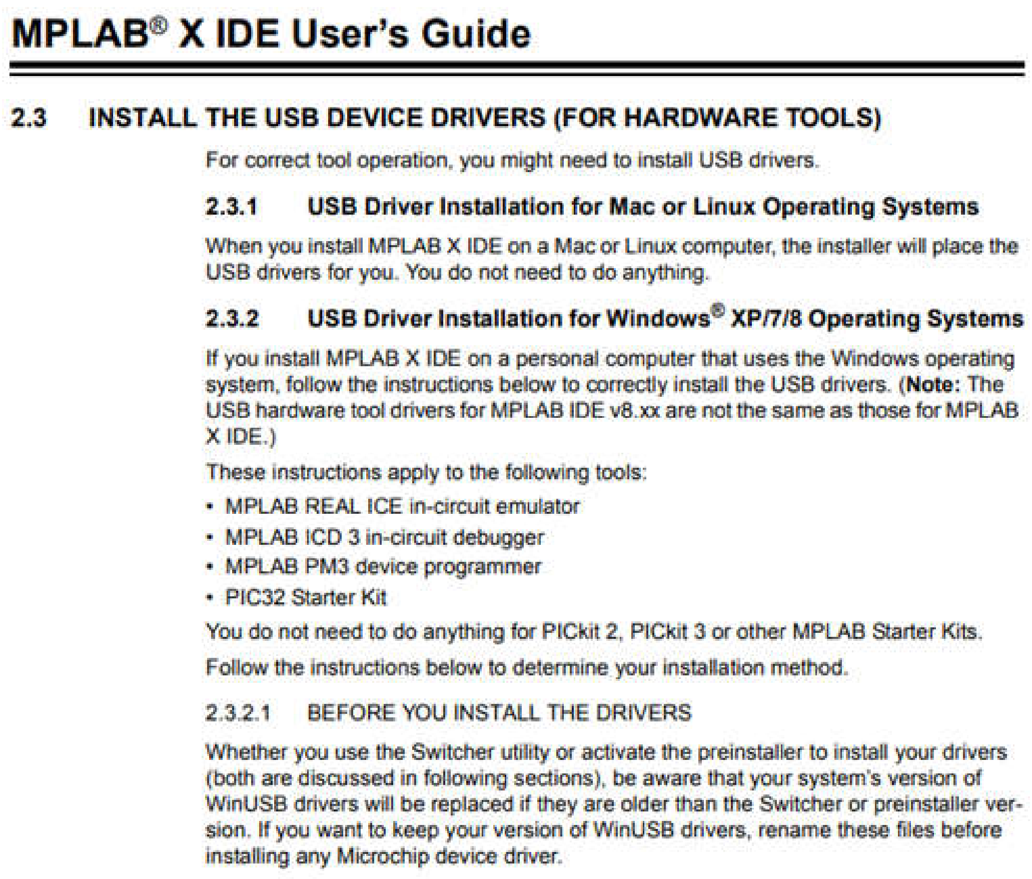
Other Configuration-specific Files

This topic describes two other (non C-language) configuration-specific files.

Description

There are two additional (non-C language) files generated by the MHC and placed into the configuration-specific folder. The first file, `<config-name>.mhc`, captures the configuration selections made by the user. The second file, which is always named `configuration.xml`, captures the various checksums and additional information required by the MHC to identify which generated files have been edited externally and to store miscellaneous other information it requires (such as the path to the MPLAB Harmony installation).

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Understanding%20MPLAB%20Harmony_v2.04.pdf, page 24-25.



MPLAB® X IDE User's Guide

2.3 INSTALL THE USB DEVICE DRIVERS (FOR HARDWARE TOOLS)

For correct tool operation, you might need to install USB drivers.

2.3.1 USB Driver Installation for Mac or Linux Operating Systems

When you install MPLAB X IDE on a Mac or Linux computer, the installer will place the USB drivers for you. You do not need to do anything.

2.3.2 USB Driver Installation for Windows® XP/7/8 Operating Systems

If you install MPLAB X IDE on a personal computer that uses the Windows operating system, follow the instructions below to correctly install the USB drivers. (**Note:** The USB hardware tool drivers for MPLAB IDE v8.xx are not the same as those for MPLAB X IDE.)

These instructions apply to the following tools:

- MPLAB REAL ICE in-circuit emulator
- MPLAB ICD 3 in-circuit debugger
- MPLAB PM3 device programmer
- PIC32 Starter Kit

You do not need to do anything for PICkit 2, PICkit 3 or other MPLAB Starter Kits.

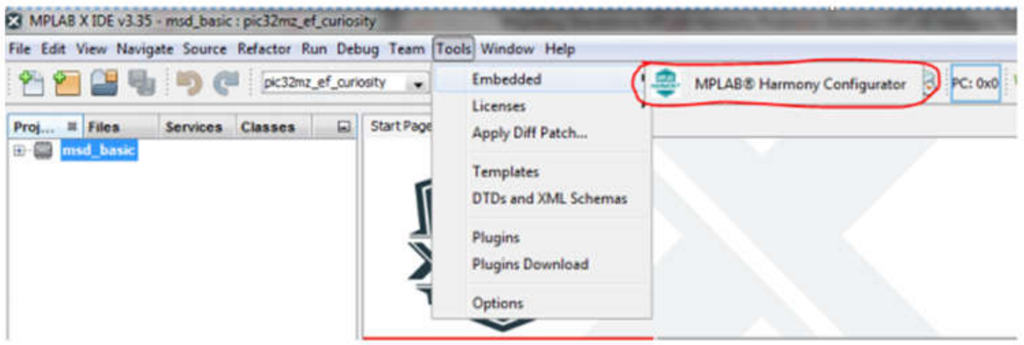
Follow the instructions below to determine your installation method.

2.3.2.1 BEFORE YOU INSTALL THE DRIVERS

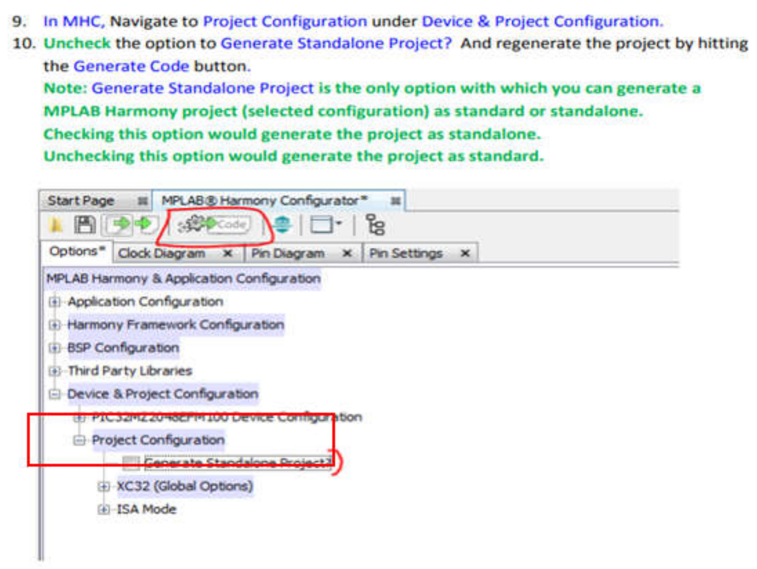
Whether you use the Switcher utility or activate the preinstaller to install your drivers (both are discussed in following sections), be aware that your system's version of WinUSB drivers will be replaced if they are older than the Switcher or preinstaller version. If you want to keep your version of WinUSB drivers, rename these files before installing any Microchip device driver.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/50002027D.pdf, page 32.

Open MHC by clicking Tools->Embedded->MPLAB Harmony Configurator.


MPLAB X IDE v3.35 - msd_basic : pic32mz_ef_curiosity
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
pic32mz_ef_curiosity
Embedded
MPLAB® Harmony Configurator
PC: 0x0
Licenses
Apply Diff Patch...
Templates
DTDs and XML Schemas
Plugins
Plugins Download
Options
Proj... Files Services Classes
msd_basic
Start Page

Source:
http://ww1.microchip.com/downloads/en/DeviceDoc/Migrating%20Standalone%20MPLAB%20Harmony%20Project%20to%20Standard%20MPLAB%20Harmony%20Project.pdf, page 4.


9. In MHC, Navigate to Project Configuration under Device & Project Configuration.
10. Uncheck the option to Generate Standalone Project? And regenerate the project by hitting the Generate Code button.
Note: Generate Standalone Project is the only option with which you can generate a MPLAB Harmony project (selected configuration) as standard or standalone.
Checking this option would generate the project as standalone.
Unchecking this option would generate the project as standard.

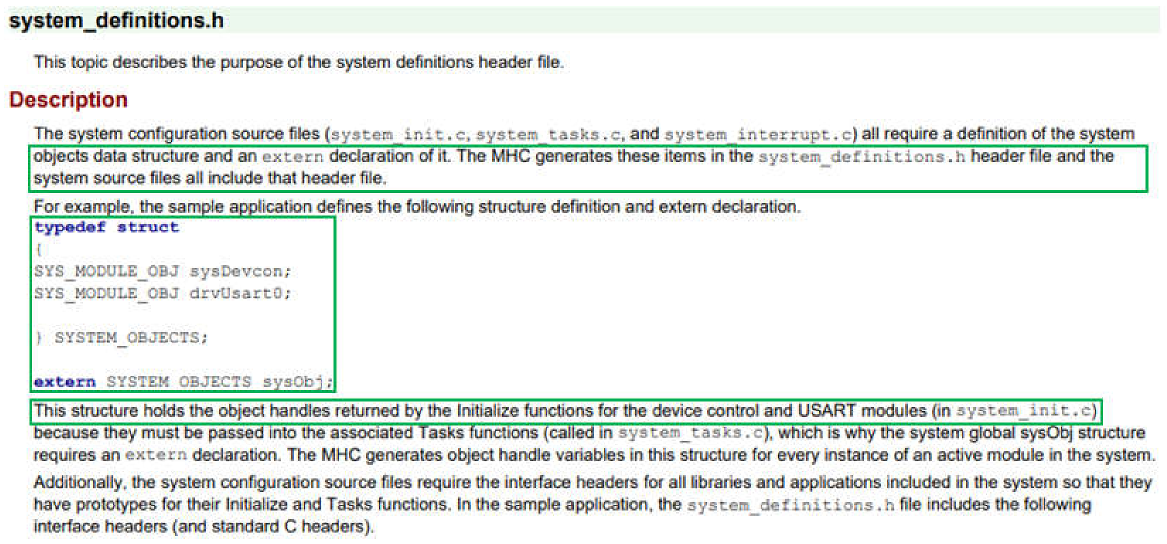
Start Page | MPLAB® Harmony Configurator*

Options* | Clock Diagram | Pin Diagram | Pin Settings

MPLAB Harmony & Application Configuration
- Application Configuration
- Harmony Framework Configuration
- BSP Configuration
- Third Party Libraries
- Device & Project Configuration
  - PIC32MZ2048EFM100 Device Configuration
  - Project Configuration
    - Generate Standalone Project?
    - XC32 (Global Options)
    - ISA Mode

Source:
http://ww1.microchip.com/downloads/en/DeviceDoc/Migrating%20Standalone%20MPLAB%20Harmony%20Project%20to%20Standard%20MPLAB%20Harmony%20Project.pdf, page 6.

17. MPLAB X IDE generates specific application handler code and defines a specific element in the specific code to be handled by one of the generic application functions for that hardware module. For example, MPLAB X IDE uses MPLAB Harmony Configurator (MHC) as a plug-in for generating system-specific application handler code. System-specific application handler code comprises functions and data structures ("specific element") corresponding to specific hardware modules that extend or otherwise connect the system-specific application handler code and data structures made available by the generic application handler code. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



system_definitions.h

This topic describes the purpose of the system definitions header file.

Description

The system configuration source files (`system_init.c`, `system_tasks.c`, and `system_interrupt.c`) all require a definition of the system objects data structure and an `extern` declaration of it. The MHC generates these items in the `system_definitions.h` header file and the system source files all include that header file.

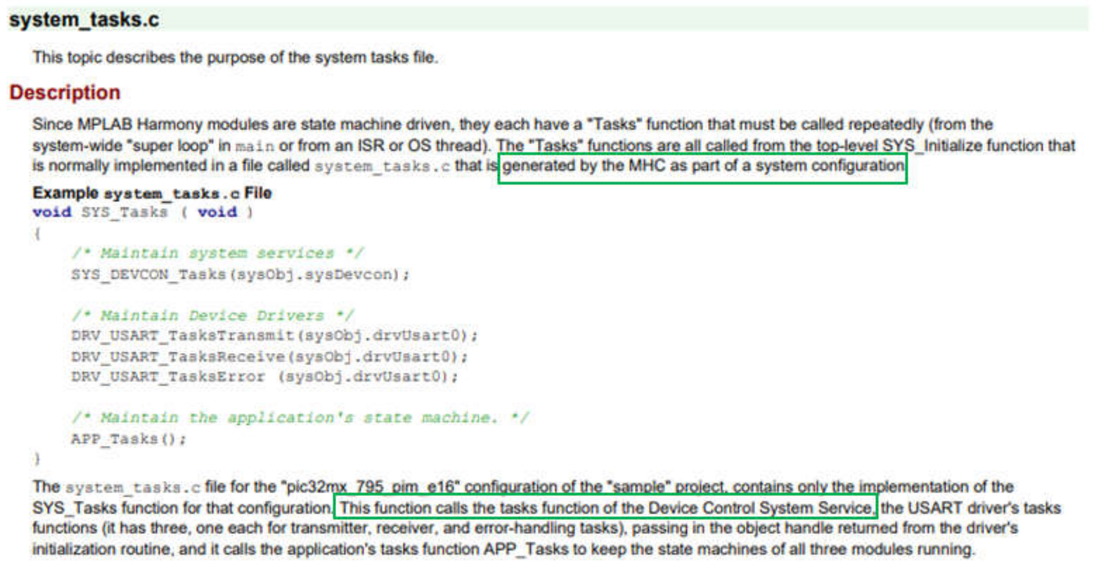
For example, the sample application defines the following structure definition and extern declaration.

```
typedef struct
{
SYS_MODULE_OBJ sysDevcon;
SYS_MODULE_OBJ drvUsart0;

} SYSTEM_OBJECTS;

extern SYSTEM_OBJECTS sysObj;
```

This structure holds the object handles returned by the Initialize functions for the device control and USART modules (in `system_init.c`) because they must be passed into the associated Tasks functions (called in `system_tasks.c`), which is why the system global sysObj structure requires an `extern` declaration. The MHC generates object handle variables in this structure for every instance of an active module in the system.

Additionally, the system configuration source files require the interface headers for all libraries and applications included in the system so that they have prototypes for their Initialize and Tasks functions. In the sample application, the `system_definitions.h` file includes the following interface headers (and standard C headers).

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Understanding%20MPLAB%20Harmony_v2.04.pdf, page 23.



system_tasks.c

This topic describes the purpose of the system tasks file.

Description

Since MPLAB Harmony modules are state machine driven, they each have a "Tasks" function that must be called repeatedly (from the system-wide "super loop" in `main` or from an ISR or OS thread). The "Tasks" functions are all called from the top-level SYS_Initialize function that is normally implemented in a file called `system_tasks.c` that is generated by the MHC as part of a system configuration.

Example `system_tasks.c` File

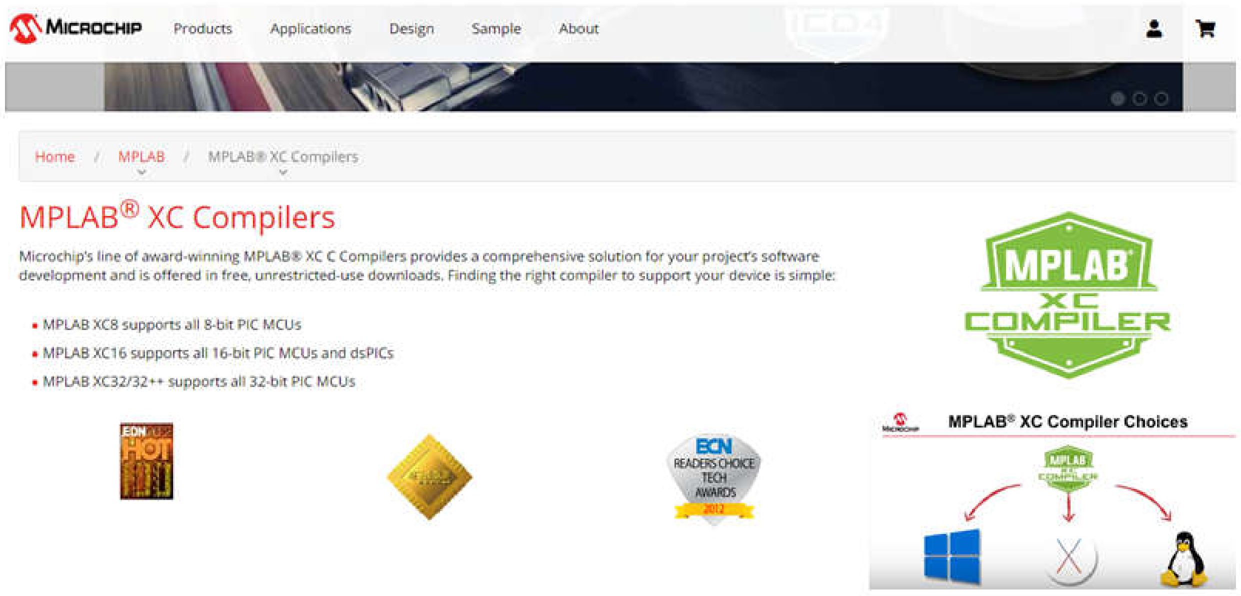
```
void SYS_Tasks ( void )
{
    /* Maintain system services */
    SYS_DEVCON_Tasks(sysObj.sysDevcon);

    /* Maintain Device Drivers */
    DRV_USART_TasksTransmit(sysObj.drvUsart0);
    DRV_USART_TasksReceive(sysObj.drvUsart0);
    DRV_USART_TasksError (sysObj.drvUsart0);

    /* Maintain the application's state machine. */
    APP_Tasks();
}
```

The `system_tasks.c` file for the "pic32mx_795_pim_e16" configuration of the "sample" project, contains only the implementation of the SYS_Tasks function for that configuration. This function calls the tasks function of the Device Control System Service, the USART driver's tasks functions (it has three, one each for transmitter, receiver, and error-handling tasks), passing in the object handle returned from the driver's initialization routine, and it calls the application's tasks function APP_Tasks to keep the state machines of all three modules running.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Understanding%20MPLAB%20Harmony_v2.04.pdf, page 23.

18. MPLAB X IDE compiles the generic application handler programs together with the specific application handler code to produce machine-readable code to be executed by an embedded processor in the at least one of the types of the hardware modules. For example, when a specific application is needed for a particular hardware, the generic functions and the specific functions are compiled together to yield a machine readable code. Microchip

Technology and/or its customers compile the generic functions and the specific functions using MPLAB X IDE and/or any other compiling SDK supported by Microchip Technology. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

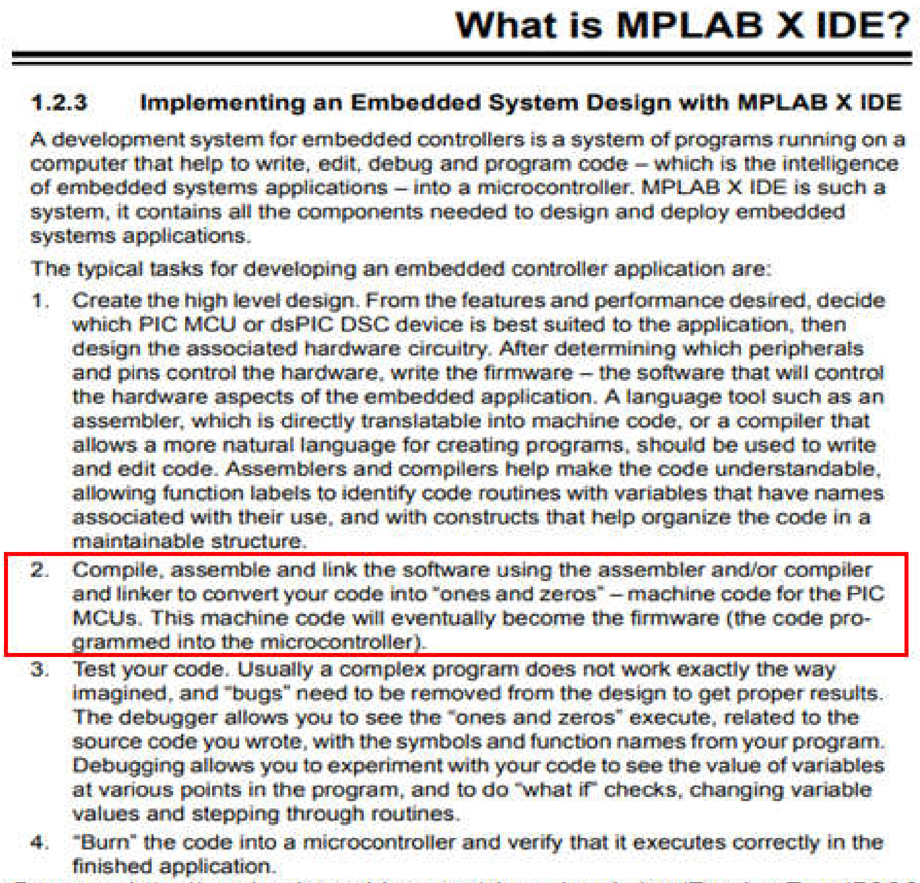

MICROCHIP
Products
Applications
Design
Sample
About
Home / MPLAB / MPLAB® XC Compilers
MPLAB XC COMPILER
MPLAB® XC Compiler Choices
EDN HOT 100
ECN READERS CHOICE TECH AWARDS 2012

MPLAB® XC Compilers

Microchip's line of award-winning MPLAB® XC C Compilers provides a comprehensive solution for your project's software development and is offered in free, unrestricted-use downloads. Finding the right compiler to support your device is simple:

- MPLAB XC8 supports all 8-bit PIC MCUs
- MPLAB XC16 supports all 16-bit PIC MCUs and dsPICs
- MPLAB XC32/32++ supports all 32-bit PIC MCUs

Source: http://www.microchip.com/mplab/compilers.



What is MPLAB X IDE?

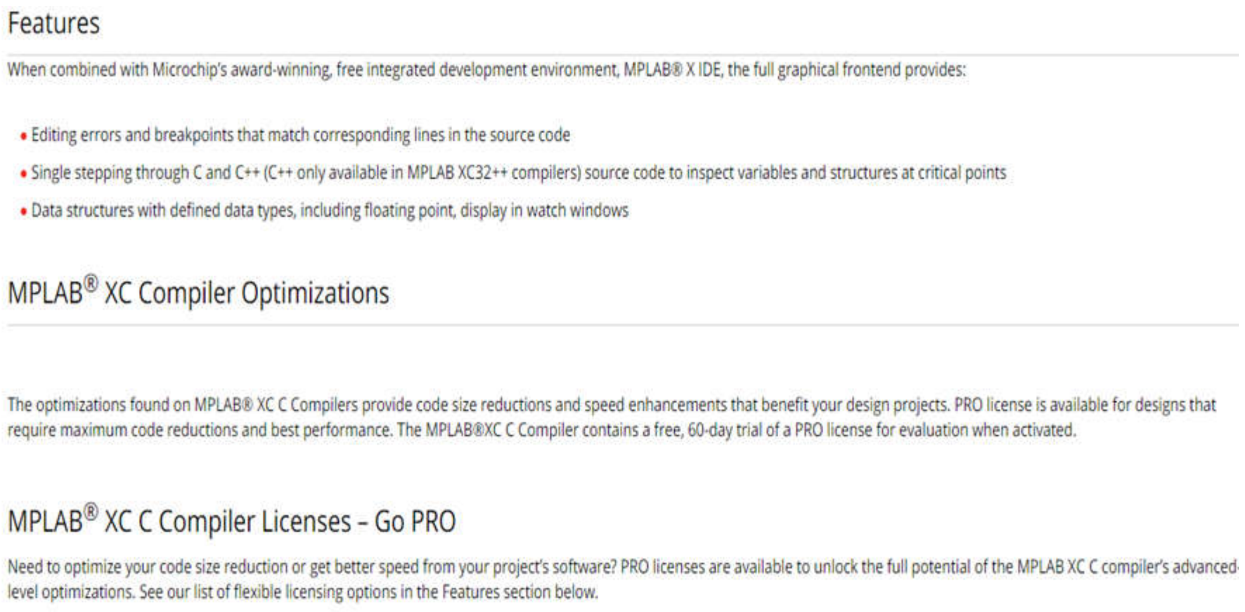
1.2.3 Implementing an Embedded System Design with MPLAB X IDE

A development system for embedded controllers is a system of programs running on a computer that help to write, edit, debug and program code – which is the intelligence of embedded systems applications – into a microcontroller. MPLAB X IDE is such a system, it contains all the components needed to design and deploy embedded systems applications.

The typical tasks for developing an embedded controller application are:

1. Create the high level design. From the features and performance desired, decide which PIC MCU or dsPIC DSC device is best suited to the application, then design the associated hardware circuitry. After determining which peripherals and pins control the hardware, write the firmware – the software that will control the hardware aspects of the embedded application. A language tool such as an assembler, which is directly translatable into machine code, or a compiler that allows a more natural language for creating programs, should be used to write and edit code. Assemblers and compilers help make the code understandable, allowing function labels to identify code routines with variables that have names associated with their use, and with constructs that help organize the code in a maintainable structure.
2. Compile, assemble and link the software using the assembler and/or compiler and linker to convert your code into "ones and zeros" – machine code for the PIC MCUs. This machine code will eventually become the firmware (the code programmed into the microcontroller).
3. Test your code. Usually a complex program does not work exactly the way imagined, and "bugs" need to be removed from the design to get proper results. The debugger allows you to see the "ones and zeros" execute, related to the source code you wrote, with the symbols and function names from your program. Debugging allows you to experiment with your code to see the value of variables at various points in the program, and to do "what if" checks, changing variable values and stepping through routines.
4. "Burn" the code into a microcontroller and verify that it executes correctly in the finished application.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/50002027D.pdf, page 19.



Features

When combined with Microchip's award-winning, free integrated development environment, MPLAB® X IDE, the full graphical frontend provides:

- Editing errors and breakpoints that match corresponding lines in the source code
- Single stepping through C and C++ (C++ only available in MPLAB XC32++ compilers) source code to inspect variables and structures at critical points
- Data structures with defined data types, including floating point, display in watch windows

MPLAB® XC Compiler Optimizations

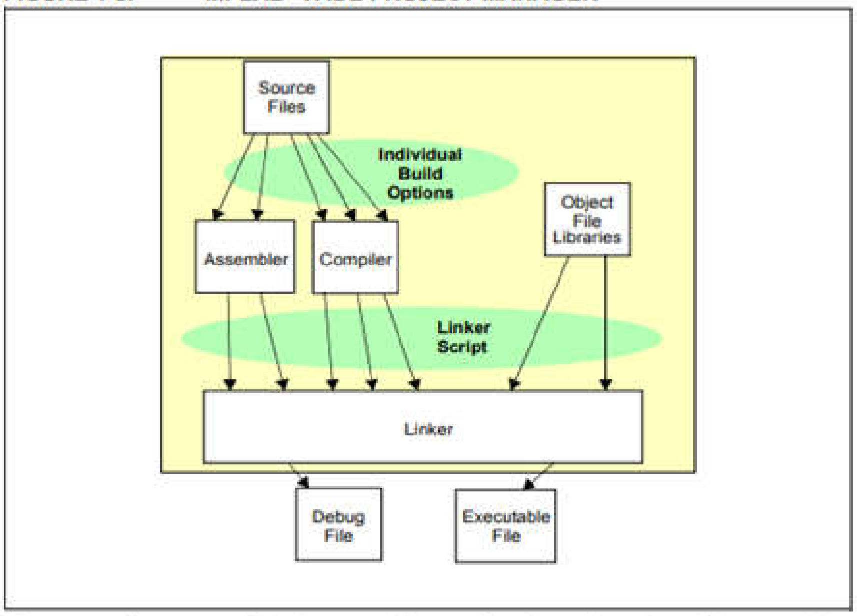
The optimizations found on MPLAB® XC C Compilers provide code size reductions and speed enhancements that benefit your design projects. PRO license is available for designs that require maximum code reductions and best performance. The MPLAB®XC C Compiler contains a free, 60-day trial of a PRO license for evaluation when activated.

MPLAB® XC C Compiler Licenses – Go PRO

Need to optimize your code size reduction or get better speed from your project's software? PRO licenses are available to unlock the full potential of the MPLAB XC C compiler's advanced-level optimizations. See our list of flexible licensing options in the Features section below.

Source: http://www.microchip.com/mplab/compilers.

FIGURE 1-8: MPLAB® X IDE PROJECT MANAGER



The source files are text files that are written conforming to the rules of the assembler or compiler. The assembler and compiler convert them into intermediate modules of machine code and placeholders for references to functions and data storage.

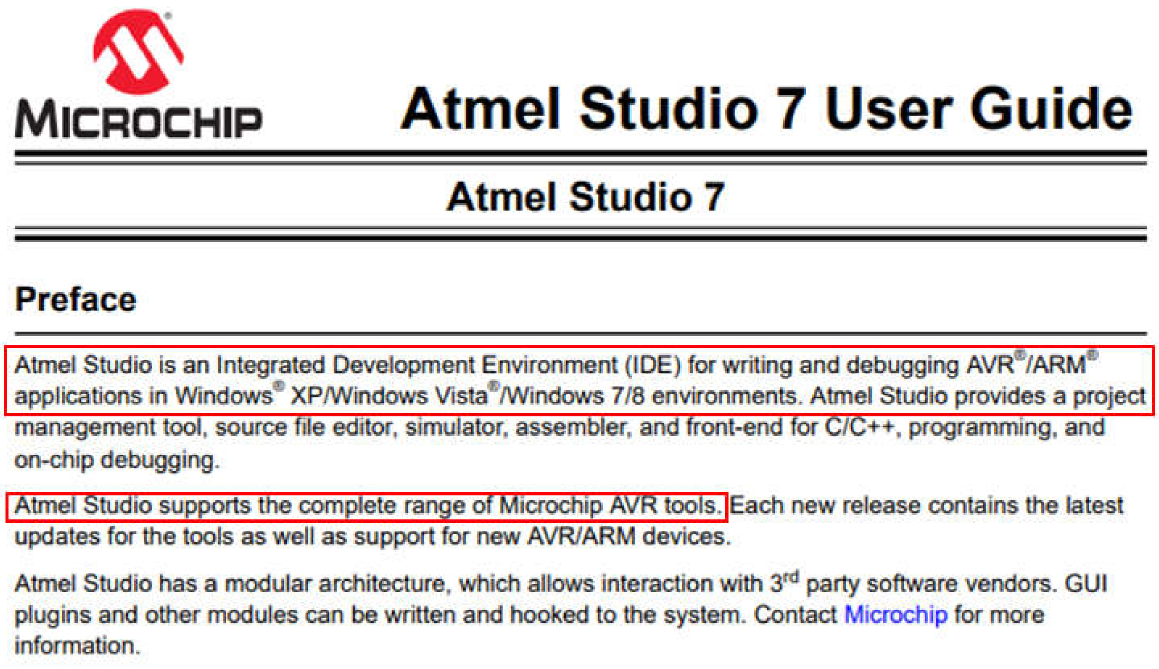
The linker resolves these placeholders and combines all the modules into a file of executable machine code. The linker also produces a debug file which allows MPLAB X IDE to relate the executing machine codes back to the source files.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/50002027D.pdf, page 22.

<u>**ATMEL STUDIO 7**</u>

19. Atmel Studio 7 is an Integrated Development Environment (IDE) for developing and debugging all AVR and SAM microcontroller applications. Atmel Studio 7 provides one or more generic application handler programs, each such program comprising computer program code for performing generic application functions common to multiple types of hardware modules used in a communication system. For example, Atmel Studio 7 comprises a Hardware Abstraction Layer (HAL) containing Hardware Access Functions for defined functions (“generic application handler”). Hardware Access Functions include source code comprising functions and data structure, modules and/or pre-built Libraries, which are common and uniform across all supported Atmel Studio 7 hardware modules. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in

connection with other elements herein.


MICROCHIP

Atmel Studio 7 User Guide

Atmel Studio 7

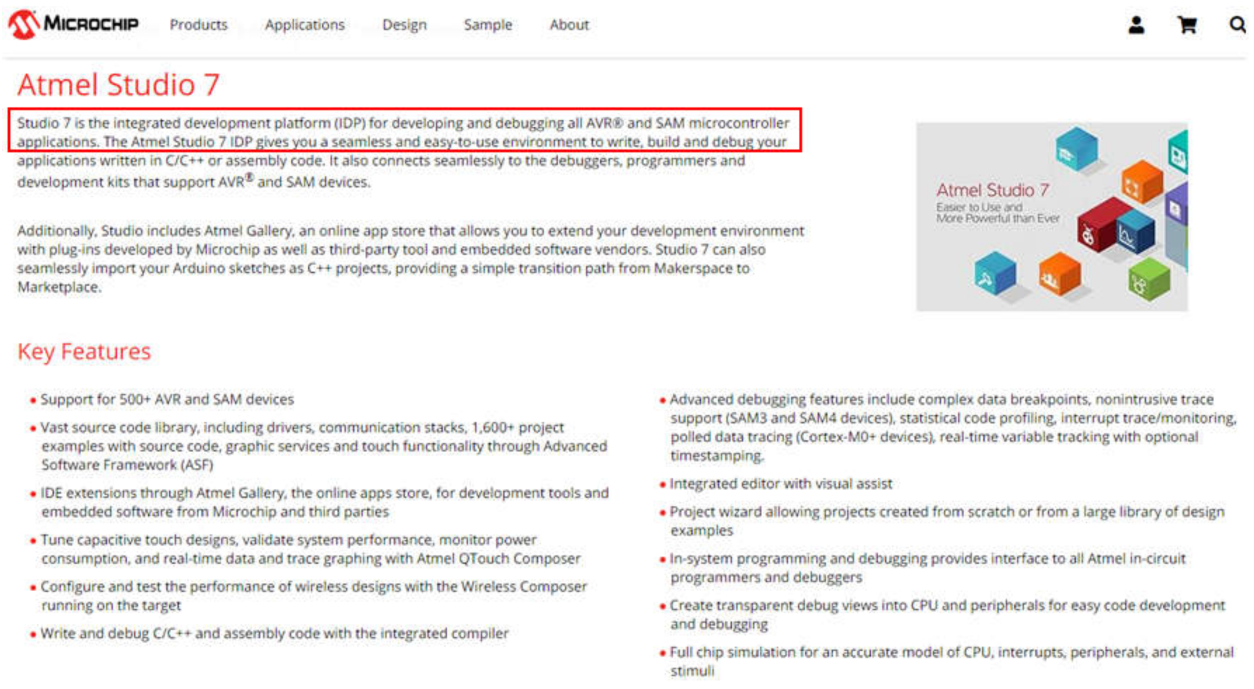
Preface

Atmel Studio is an Integrated Development Environment (IDE) for writing and debugging AVR®/ARM® applications in Windows® XP/Windows Vista®/Windows 7/8 environments. Atmel Studio provides a project management tool, source file editor, simulator, assembler, and front-end for C/C++, programming, and on-chip debugging.

Atmel Studio supports the complete range of Microchip AVR tools. Each new release contains the latest updates for the tools as well as support for new AVR/ARM devices.

Atmel Studio has a modular architecture, which allows interaction with 3rd party software vendors. GUI plugins and other modules can be written and hooked to the system. Contact Microchip for more information.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-Studio-7-User-Guide.pdf, page 1.


MICROCHIP Products Applications Design Sample About

Atmel Studio 7

Studio 7 is the integrated development platform (IDP) for developing and debugging all AVR® and SAM microcontroller applications. The Atmel Studio 7 IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. It also connects seamlessly to the debuggers, programmers and development kits that support AVR® and SAM devices.

Additionally, Studio includes Atmel Gallery, an online app store that allows you to extend your development environment with plug-ins developed by Microchip as well as third-party tool and embedded software vendors. Studio 7 can also seamlessly import your Arduino sketches as C++ projects, providing a simple transition path from Makerspace to Marketplace.

Key Features

- Support for 500+ AVR and SAM devices
- Vast source code library, including drivers, communication stacks, 1,600+ project examples with source code, graphic services and touch functionality through Advanced Software Framework (ASF)
- IDE extensions through Atmel Gallery, the online apps store, for development tools and embedded software from Microchip and third parties
- Tune capacitive touch designs, validate system performance, monitor power consumption, and real-time data and trace graphing with Atmel QTouch Composer
- Configure and test the performance of wireless designs with the Wireless Composer running on the target
- Write and debug C/C++ and assembly code with the integrated compiler
- Advanced debugging features include complex data breakpoints, nonintrusive trace support (SAM3 and SAM4 devices), statistical code profiling, interrupt trace/monitoring, polled data tracing (Cortex-M0+ devices), real-time variable tracking with optional timestamping.
- Integrated editor with visual assist
- Project wizard allowing projects created from scratch or from a large library of design examples
- In-system programming and debugging provides interface to all Atmel in-circuit programmers and debuggers
- Create transparent debug views into CPU and peripherals for easy code development and debugging
- Full chip simulation for an accurate model of CPU, interrupts, peripherals, and external stimuli

Source: http://www.microchip.com/mplab/avr-support/atmel-studio-7 6

1.2 Components of Hardware Abstraction Layer

Figure 1 shows a representative block diagram of the Hardware Abstraction Layer in relation to the MC Application Framework and the customer application.

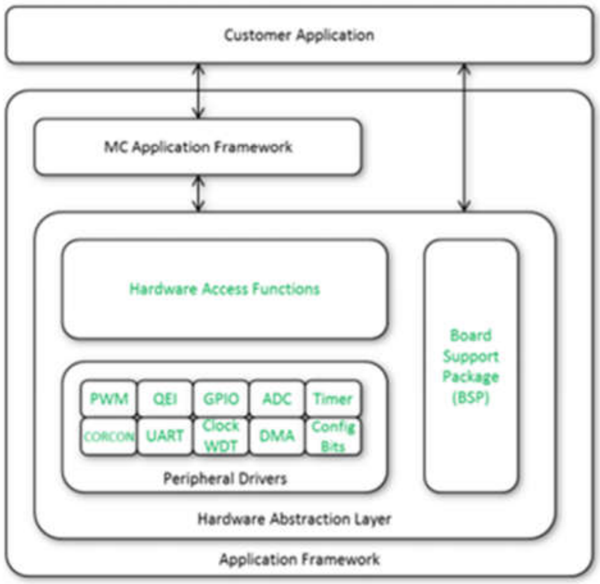
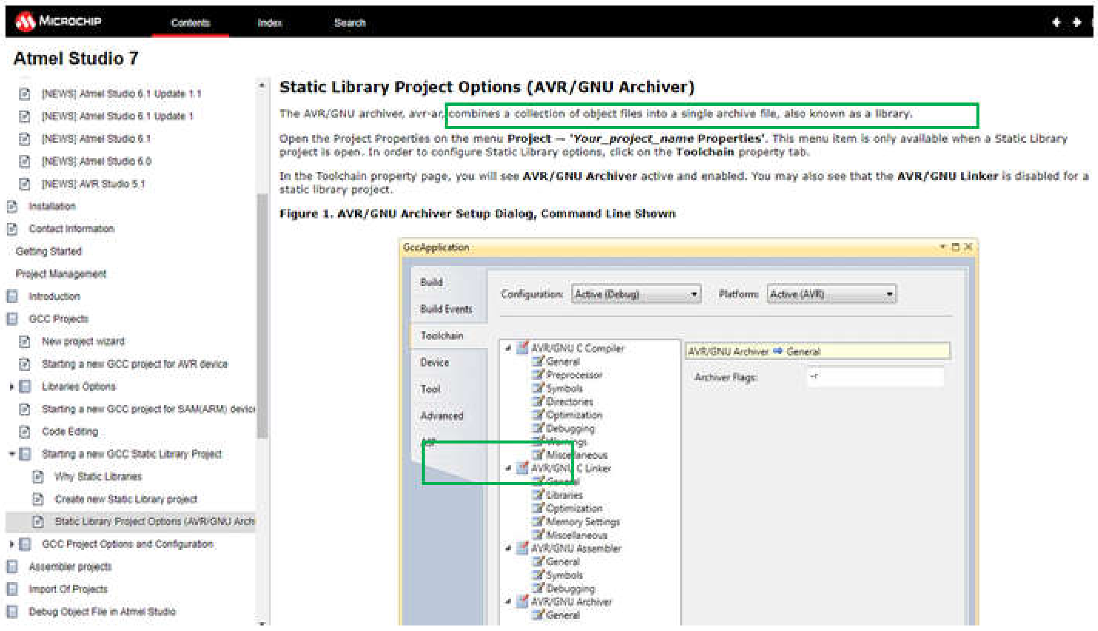


Figure 1: Block diagram of Hardware Abstraction Layer

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/hardware-abstraction-layer.pdf, page 5.



Atmel Studio 7

- [NEWS] Atmel Studio 6.1 Update 1.1
- [NEWS] Atmel Studio 6.1 Update 1
- [NEWS] Atmel Studio 6.1
- [NEWS] Atmel Studio 6.0
- [NEWS] AVR Studio 5.1
- Installation
- Contact Information
- Getting Started
- Project Management
- Introduction
- GCC Projects
  - New project wizard
  - Starting a new GCC project for AVR device
  - Libraries Options
  - Starting a new GCC project for SAM(ARM) devic
  - Code Editing
  - Starting a new GCC Static Library Project
    - Why Static Libraries
    - Create new Static Library project
    - Static Library Project Options (AVR/GNU Arch
  - GCC Project Options and Configuration
- Assembler projects
- Import Of Projects
- Debug Object File in Atmel Studio

Static Library Project Options (AVR/GNU Archiver)

The AVR/GNU archiver, avr-ar, combines a collection of object files into a single archive file, also known as a library.

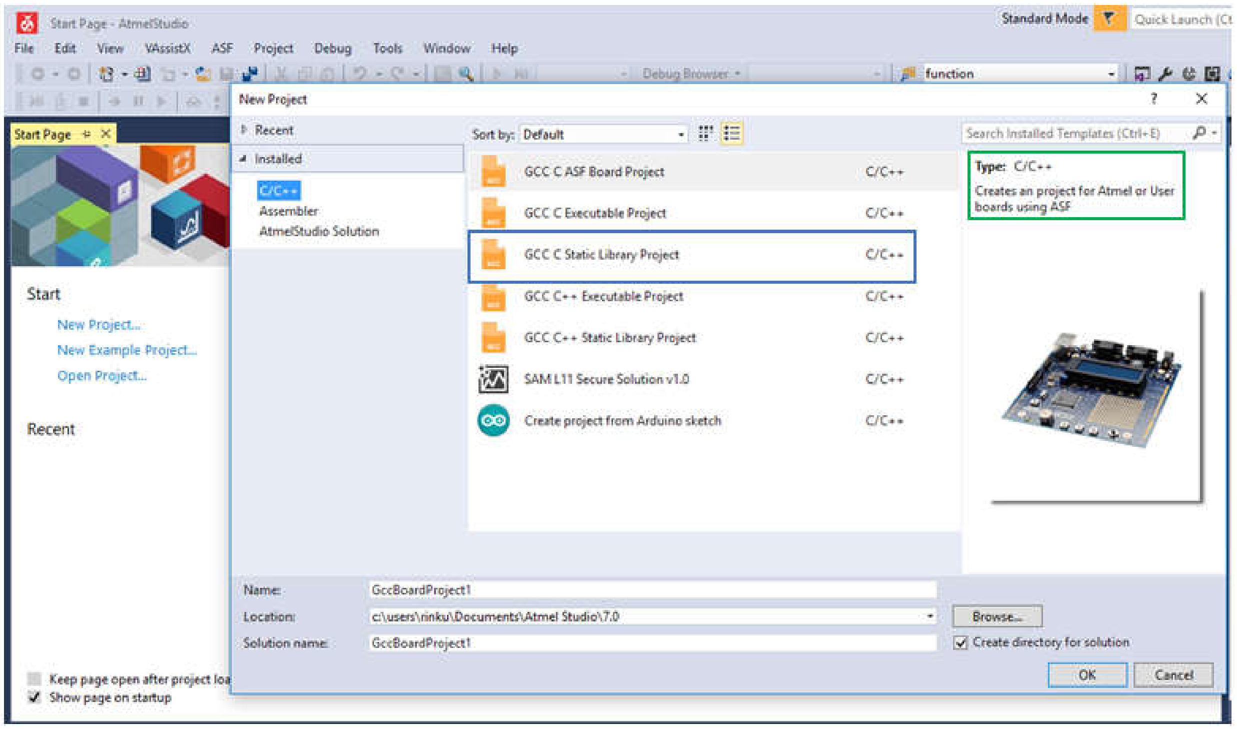
Open the Project Properties on the menu **Project → 'Your_project_name Properties'**. This menu item is only available when a Static Library project is open. In order to configure Static Library options, click on the **Toolchain** property tab.

In the Toolchain property page, you will see **AVR/GNU Archiver** active and enabled. You may also see that the **AVR/GNU Linker** is disabled for a static library project.

Figure 1. AVR/GNU Archiver Setup Dialog, Command Line Shown

Configuration: Active (Debug)
Platform: Active (AVR)
AVR/GNU Archiver ⇨ General
Archiver Flags: -r

Source: https://www.microchip.com/webdoc/GUID-ECD8A826-B1DA-44FC-BE0B-5A53418A47BD/index.html



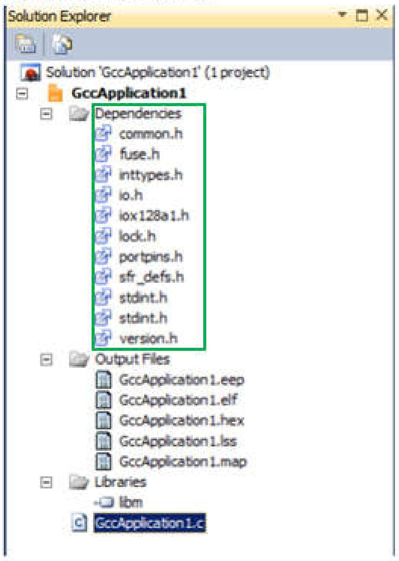
GCC C ASF Board Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++
SAM L11 Secure Solution v1.0	C/C++
Create project from Arduino sketch	C/C++

Type: C/C++
Creates an project for Atmel or User boards using ASF

Name: GccBoardProject1
Location: c:\users\rinku\Documents\Atmel Studio\7.0
Solution name: GccBoardProject1
☑ Create directory for solution

Source: Atmel Studio Software.

Figure 3-6. View of a GCC Project after Build Completed



```
Solution 'GccApplication1' (1 project)
⊟ GccApplication1
  ⊟ Dependencies
      common.h
      fuse.h
      inttypes.h
      io.h
      iox128a1.h
      lock.h
      portpins.h
      sfr_defs.h
      stdint.h
      stdint.h
      version.h
  ⊟ Output Files
      GccApplication1.eep
      GccApplication1.elf
      GccApplication1.hex
      GccApplication1.lss
      GccApplication1.map
  ⊟ Libraries
      libm
    GccApplication1.c
```

Dependencies

All the included files are listed here. Double click on any file to open it in the editor.

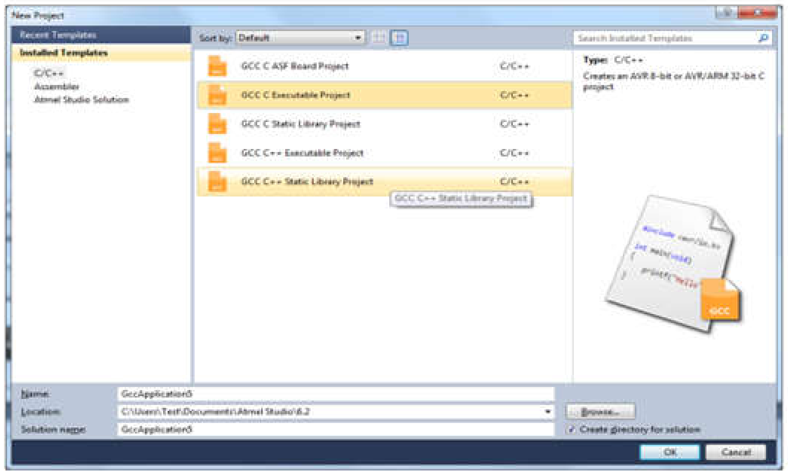
Source: http://ww1.microchip.com/downloads/en/devicedoc/atmel-42167-atmel-studio_user%20guide.pdf, page 29.

20. Atmel Studio 7 generates specific application handler code to associate the generic application functions with specific functions of a device driver for at least one of the

types of the hardware modules. For example, in addition to the generic drivers and HAL, Atmel Studio 7 also includes specific microcontroller handler code that is specific to the application and specific to particular microcontroller families such as SAM (ARM) device. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

3.2.4. Starting a New GCC Project for SAM (ARM) Device

1. Create a new project by selecting **New Project** from the **Project** menu. This will open the **Project Wizard**.

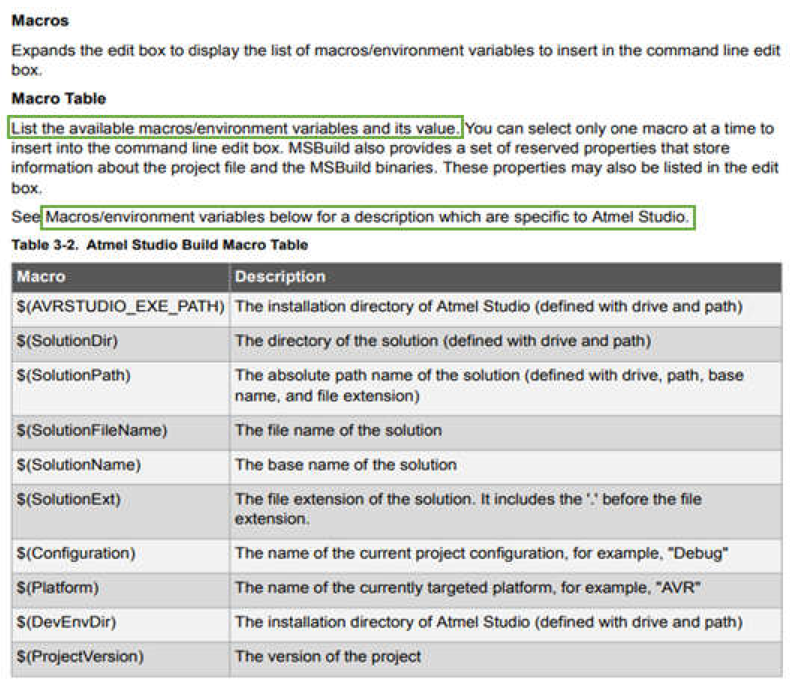

New Project
Recent Templates
Installed Templates
C/C++
Assembler
Atmel Studio Solution
Sort by: Default
Search Installed Templates
GCC C ASF Board Project C/C++
GCC C Executable Project C/C++
GCC C Static Library Project C/C++
GCC C++ Executable Project C/C++
GCC C++ Static Library Project C/C++
GCC C++ Static Library Project
Type: C/C++
Creates an AVR 8-bit or AVR/ARM 32-bit C project
GCC
Browse...
OK
Cancel

Name: GccApplication5
Location: C:\Users\Test\Documents\Atmel Studio\6.2
Solution name: GccApplication5
✓ Create directory for solution

2. Select **C/C++ → GCC C Executable Project** as a template, then specify a project name, select a location, and write a solution name for the project. Some start-up files will be added to the project by default, which will contain some device specific functions and libraries. Press **OK** when you are satisfied with the settings.
3. Select **C/C++ → GCC C Static Library Project** as a template, then specify a project name, select a location, and write a solution name for the project. This creates a Static Library (LIB) project, which is a good way to reuse code.

Source: http://ww1.microchip.com/downloads/en/devicedoc/atmel-42167-atmel-studio_user%20guide.pdf, page 33.

21. Atmel Studio 7 generates specific application handler code and defines a specific element in the specific code to be handled by one of the generic application functions for that hardware module. For example, Atmel Studio 7 generates system-specific application handler code by defining functions, data structures and/or Macros ("specific element") corresponding to specific hardware modules that extend or otherwise connect the system-specific application handler code and data structures made available by the generic application handler code of the Atmel Studio 7. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



Macros

Expands the edit box to display the list of macros/environment variables to insert in the command line edit box.

Macro Table

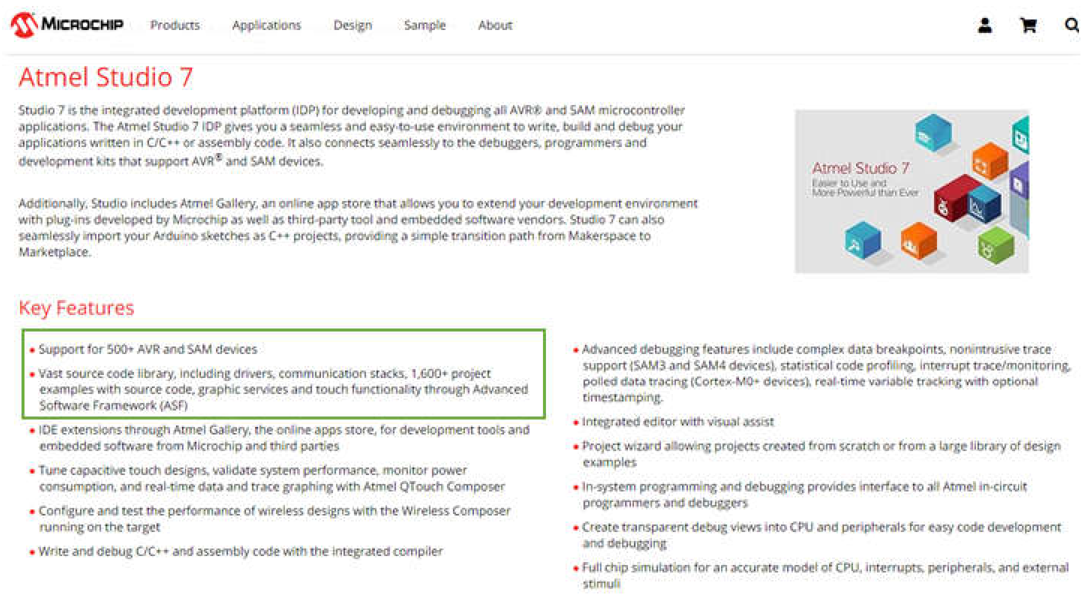
List the available macros/environment variables and its value. You can select only one macro at a time to insert into the command line edit box. MSBuild also provides a set of reserved properties that store information about the project file and the MSBuild binaries. These properties may also be listed in the edit box.

See Macros/environment variables below for a description which are specific to Atmel Studio.

Table 3-2. Atmel Studio Build Macro Table

Macro	Description
$(AVRSTUDIO_EXE_PATH)	The installation directory of Atmel Studio (defined with drive and path)
$(SolutionDir)	The directory of the solution (defined with drive and path)
$(SolutionPath)	The absolute path name of the solution (defined with drive, path, base name, and file extension)
$(SolutionFileName)	The file name of the solution
$(SolutionName)	The base name of the solution
$(SolutionExt)	The file extension of the solution. It includes the '.' before the file extension.
$(Configuration)	The name of the current project configuration, for example, "Debug"
$(Platform)	The name of the currently targeted platform, for example, "AVR"
$(DevEnvDir)	The installation directory of Atmel Studio (defined with drive and path)
$(ProjectVersion)	The version of the project

Source: http://ww1.microchip.com/downloads/en/devicedoc/atmel-42167-atmel-studio_user%20guide.pdf, page 43.



Atmel Studio 7

Studio 7 is the integrated development platform (IDP) for developing and debugging all AVR® and SAM microcontroller applications. The Atmel Studio 7 IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. It also connects seamlessly to the debuggers, programmers and development kits that support AVR® and SAM devices.

Additionally, Studio includes Atmel Gallery, an online app store that allows you to extend your development environment with plug-ins developed by Microchip as well as third-party tool and embedded software vendors. Studio 7 can also seamlessly import your Arduino sketches as C++ projects, providing a simple transition path from Makerspace to Marketplace.

Key Features

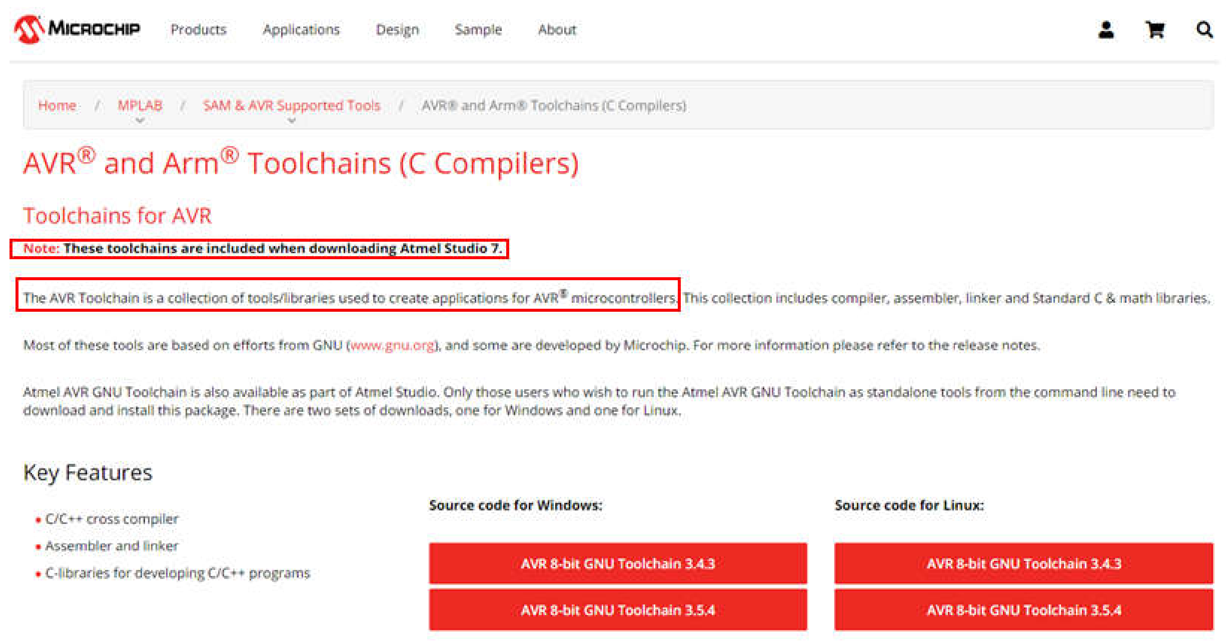
- Support for 500+ AVR and SAM devices
- Vast source code library, including drivers, communication stacks, 1,600+ project examples with source code, graphic services and touch functionality through Advanced Software Framework (ASF)
- IDE extensions through Atmel Gallery, the online apps store, for development tools and embedded software from Microchip and third parties
- Tune capacitive touch designs, validate system performance, monitor power consumption, and real-time data and trace graphing with Atmel QTouch Composer
- Configure and test the performance of wireless designs with the Wireless Composer running on the target
- Write and debug C/C++ and assembly code with the integrated compiler
- Advanced debugging features include complex data breakpoints, nonintrusive trace support (SAM3 and SAM4 devices), statistical code profiling, interrupt trace/monitoring, polled data tracing (Cortex-M0+ devices), real-time variable tracking with optional timestamping.
- Integrated editor with visual assist
- Project wizard allowing projects created from scratch or from a large library of design examples
- In-system programming and debugging provides interface to all Atmel in-circuit programmers and debuggers
- Create transparent debug views into CPU and peripherals for easy code development and debugging
- Full chip simulation for an accurate model of CPU, interrupts, peripherals, and external stimuli

13

Source: http://www.microchip.com/mplab/avr-support/atmel-studio-7

22. Atmel Studio 7 compiles the generic application handler programs together with the specific application handler code to produce machine-readable code to be executed by an

embedded processor in the at least one of the types of the hardware modules. For example, when a specific application is needed for a particular hardware, the generic functions and the specific functions are compiled together to yield a machine readable code. Microchip Technology and/or its customers compile the generic functions and the specific functions using Atmel Studio 7 and/or any other compiling SDK supported by Microchip Technology. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.


MICROCHIP
Products
Applications
Design
Sample
About
Home / MPLAB / SAM & AVR Supported Tools / AVR® and Arm® Toolchains (C Compilers)

AVR® and Arm® Toolchains (C Compilers)

Toolchains for AVR

Note: These toolchains are included when downloading Atmel Studio 7.

The AVR Toolchain is a collection of tools/libraries used to create applications for AVR® microcontrollers. This collection includes compiler, assembler, linker and Standard C & math libraries.

Most of these tools are based on efforts from GNU (www.gnu.org), and some are developed by Microchip. For more information please refer to the release notes.

Atmel AVR GNU Toolchain is also available as part of Atmel Studio. Only those users who wish to run the Atmel AVR GNU Toolchain as standalone tools from the command line need to download and install this package. There are two sets of downloads, one for Windows and one for Linux.

Key Features

- C/C++ cross compiler
- Assembler and linker
- C-libraries for developing C/C++ programs

Source code for Windows:
AVR 8-bit GNU Toolchain 3.4.3
AVR 8-bit GNU Toolchain 3.5.4
Source code for Linux:
AVR 8-bit GNU Toolchain 3.4.3
AVR 8-bit GNU Toolchain 3.5.4

Source: http://www.microchip.com/mplab/avr-support/avr-and-arm-toolchains-c-compilers

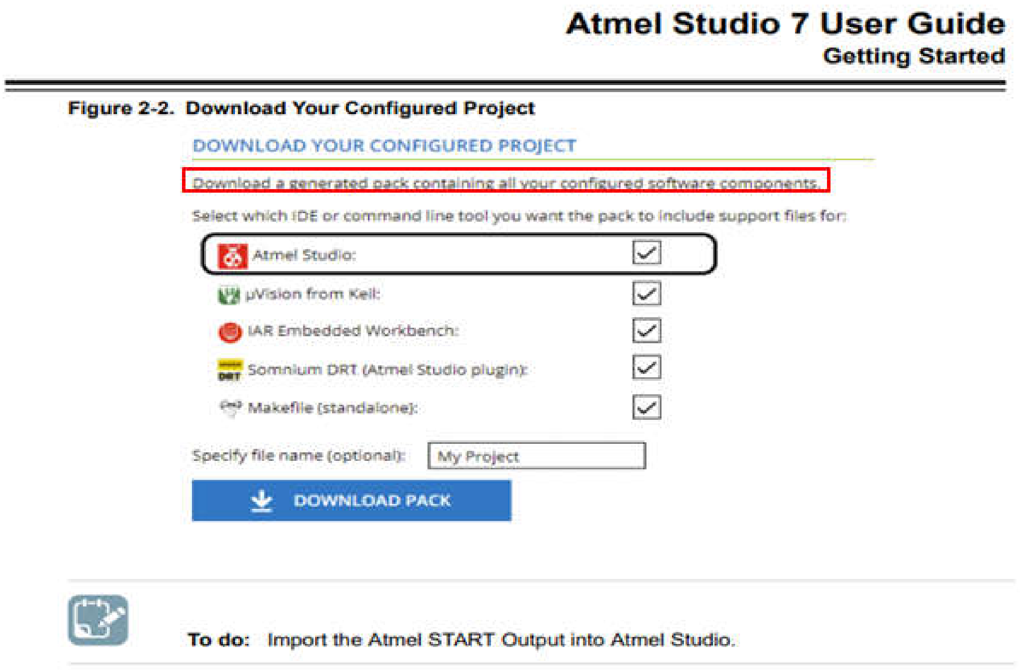

Atmel Studio 7 User Guide
Getting Started

Figure 2-2. Download Your Configured Project

DOWNLOAD YOUR CONFIGURED PROJECT

Download a generated pack containing all your configured software components.

Select which IDE or command line tool you want the pack to include support files for:

Atmel Studio:	✓
µVision from Keil:	✓
IAR Embedded Workbench:	✓
Somnium DRT (Atmel Studio plugin):	✓
Makefile (standalone):	✓

Specify file name (optional): My Project

DOWNLOAD PACK

To do: Import the Atmel START Output into Atmel Studio.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-Studio-7-User-Guide.pdf, page 28

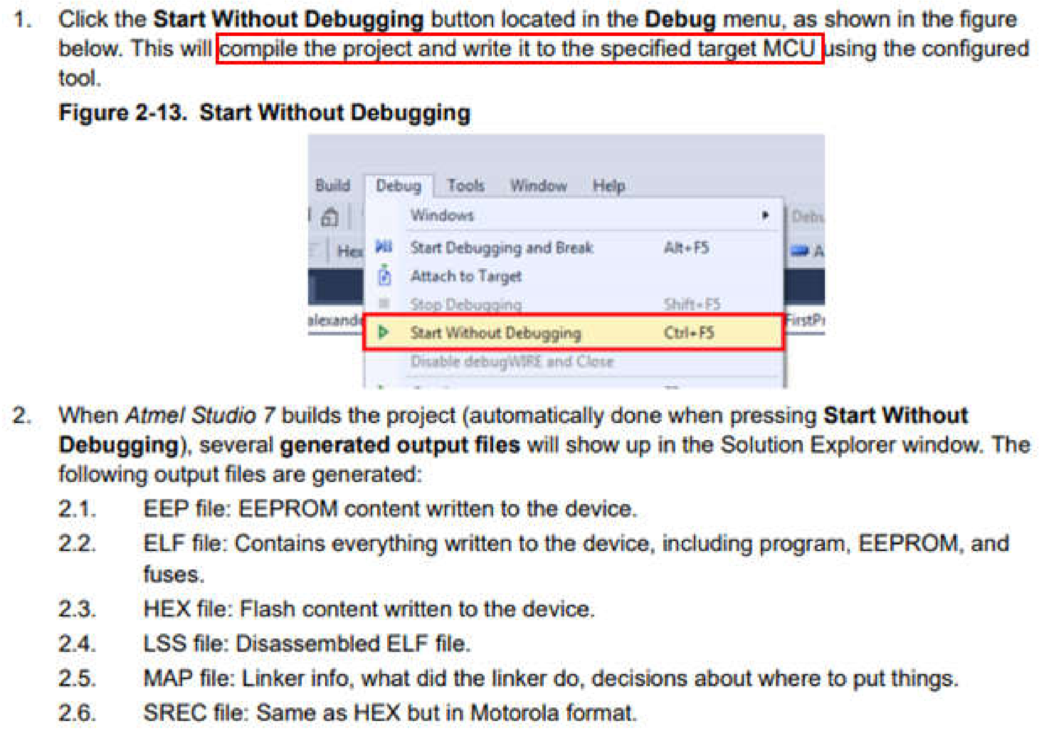

1. Click the **Start Without Debugging** button located in the **Debug** menu, as shown in the figure below. This will compile the project and write it to the specified target MCU using the configured tool.

Figure 2-13. Start Without Debugging

Build
Debug
Tools
Window
Help
Windows
Start Debugging and Break Alt+F5
Attach to Target
Stop Debugging Shift+F5
Start Without Debugging Ctrl+F5
Disable debugWIRE and Close

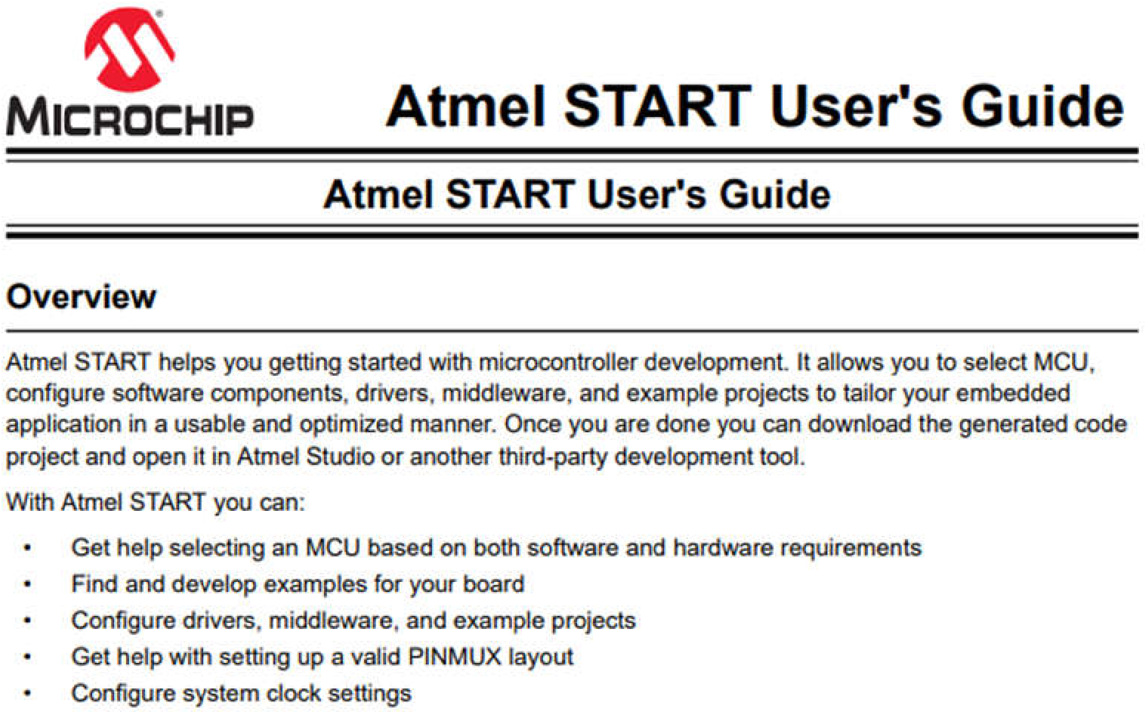
2. When *Atmel Studio 7* builds the project (automatically done when pressing **Start Without Debugging**), several **generated output files** will show up in the Solution Explorer window. The following output files are generated:
   2.1. EEP file: EEPROM content written to the device.
   2.2. ELF file: Contains everything written to the device, including program, EEPROM, and fuses.
   2.3. HEX file: Flash content written to the device.
   2.4. LSS file: Disassembled ELF file.
   2.5. MAP file: Linker info, what did the linker do, decisions about where to put things.
   2.6. SREC file: Same as HEX but in Motorola format.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-Studio-7-User-Guide.pdf, page 43.

ATMEL START

23. Atmel Start 7 is a web-based software configuration tool for various software

frameworks, which are used in development of embedded application ("producing embedded software") in Microcontroller Unit (MCU). Atmel Start provides one or more generic application handler programs, each such program comprising computer program code for performing generic application functions common to multiple types of hardware modules used in a communication system. For example, Atmel Start includes a Hardware Abstraction Layer (HAL) containing Hardware Access Functions for defined functions ("generic application handler"). Hardware Access Functions include source code comprising files such as prebuilt libraries and header files, functions, and data structures, which are common and uniform across all supported Atmel Start hardware modules. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



MICROCHIP Atmel START User's Guide

Atmel START User's Guide

Overview

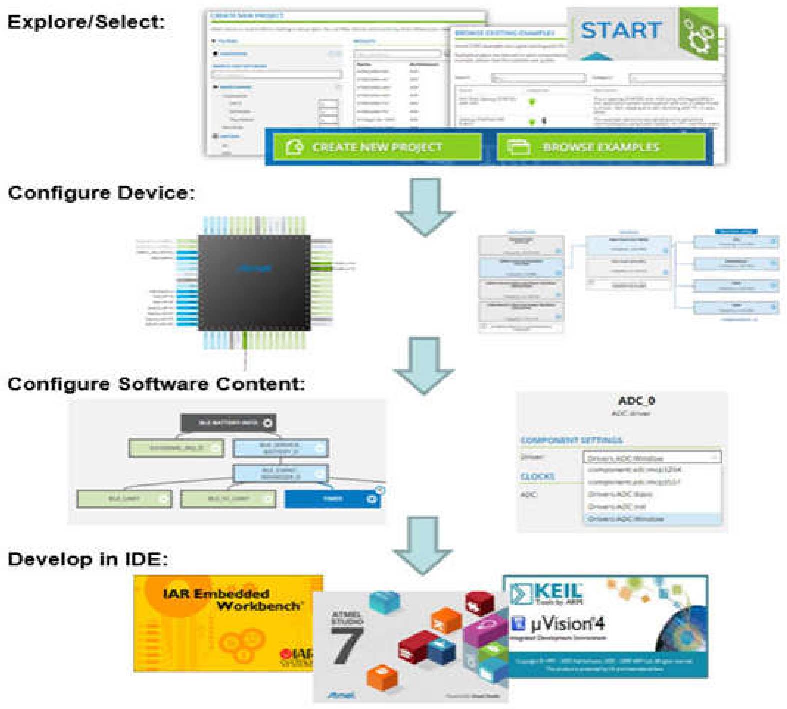
Atmel START helps you getting started with microcontroller development. It allows you to select MCU, configure software components, drivers, middleware, and example projects to tailor your embedded application in a usable and optimized manner. Once you are done you can download the generated code project and open it in Atmel Studio or another third-party development tool.

With Atmel START you can:

- Get help selecting an MCU based on both software and hardware requirements
- Find and develop examples for your board
- Configure drivers, middleware, and example projects
- Get help with setting up a valid PINMUX layout
- Configure system clock settings

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-START-User-Guide-DS50002793A.pdf, page 1.

Figure 1-1. Relation Between START, Software Content, and IDEs



Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-START-User-Guide-DS50002793A.pdf, page 5.

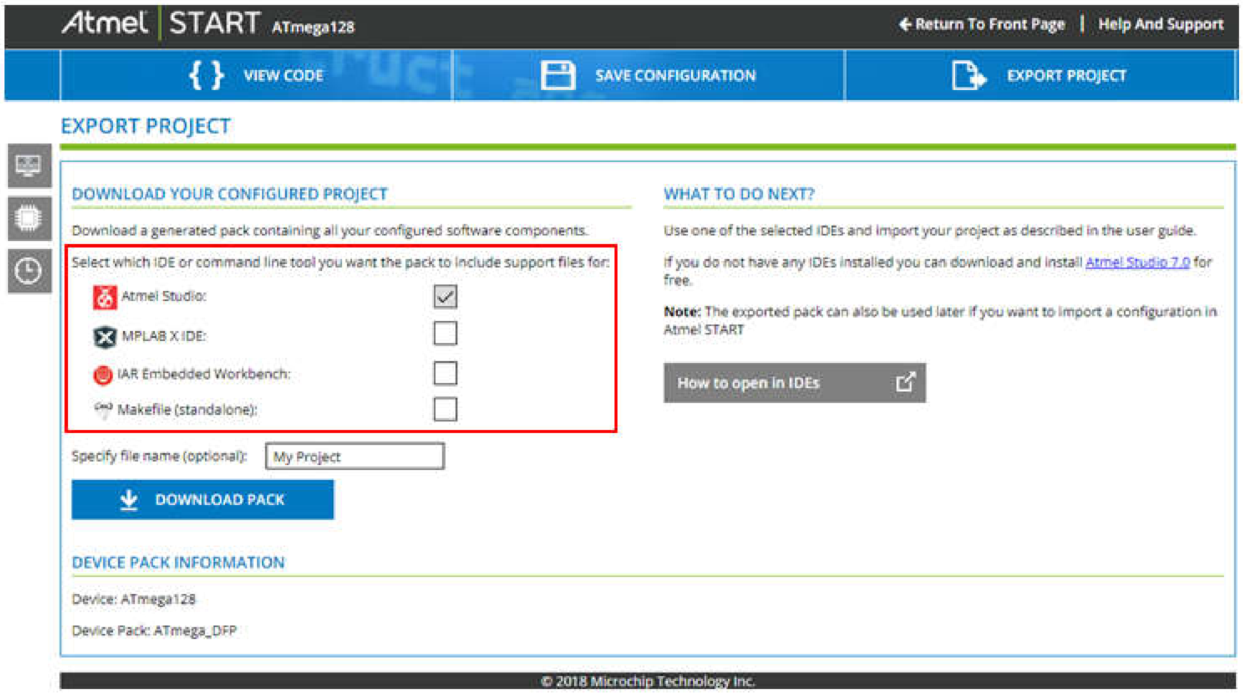
1.1.1 Atmel START

Atmel START is a web-based software configuration tool for various software frameworks, which helps you get started with MCU development. Starting from either a new project or an example project, Atmel START allows you to select and configure software components (from **ASF4** and **AVR Code**), such as drivers and middleware to tailor your embedded application in a usable and optimized manner. Once an optimized software configuration is done, you can download the generated code project and open it in the IDE of your choice, including Studio 7, IAR Embedded Workbench®, Keil® µVision®, or simply generate a makefile.

Atmel START enables you to:

- Get help with selecting an MCU, based on both software and hardware requirements
- Find and develop examples for your board
- Configure drivers, middleware, and example projects
- Get help with setting up a valid PINMUX layout
- Configure system clock settings

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-START-User-Guide-DS50002793A.pdf, page 4.



EXPORT PROJECT

DOWNLOAD YOUR CONFIGURED PROJECT

Download a generated pack containing all your configured software components.

Select which IDE or command line tool you want the pack to include support files for:

Atmel Studio:	✓
MPLAB X IDE:	
IAR Embedded Workbench:	
Makefile (standalone):	

Specify file name (optional): My Project

DOWNLOAD PACK

WHAT TO DO NEXT?

Use one of the selected IDEs and import your project as described in the user guide.

If you do not have any IDEs installed you can download and install Atmel Studio 7.0 for free.

Note: The exported pack can also be used later if you want to import a configuration in Atmel START

How to open in IDEs

DEVICE PACK INFORMATION

Device: ATmega128

Device Pack: ATmega_DFP

© 2018 Microchip Technology Inc.

Source: http://start.atmel.com/#dashboard/export.

1.2 Components of Hardware Abstraction Layer

Figure 1 shows a representative block diagram of the Hardware Abstraction Layer in relation to the MC Application Framework and the customer application.

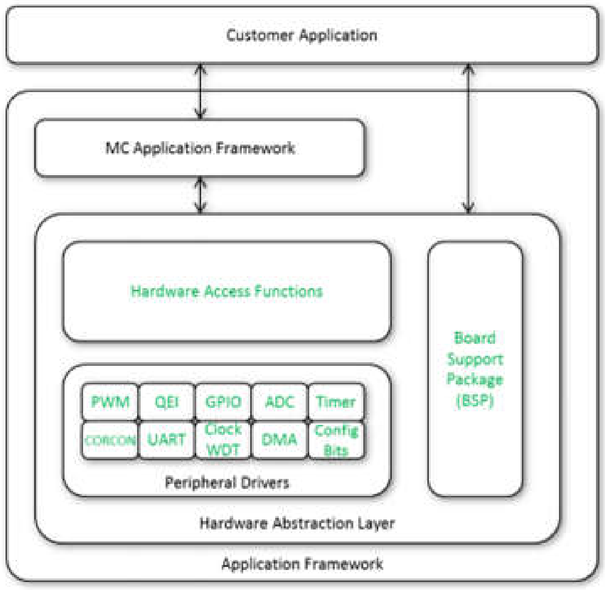
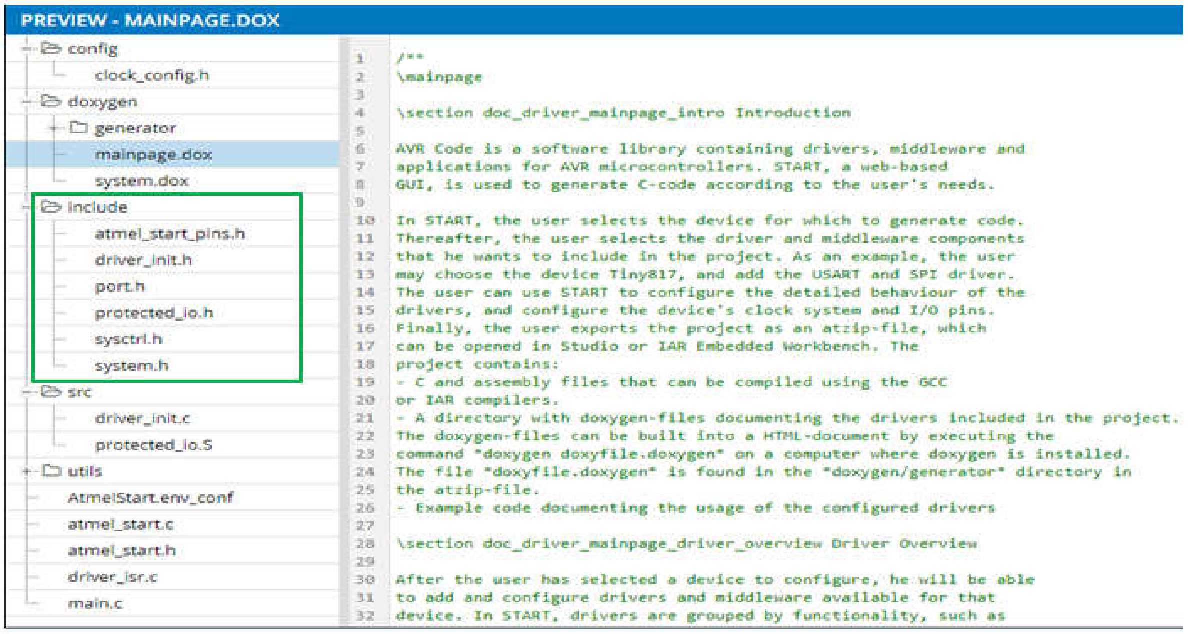


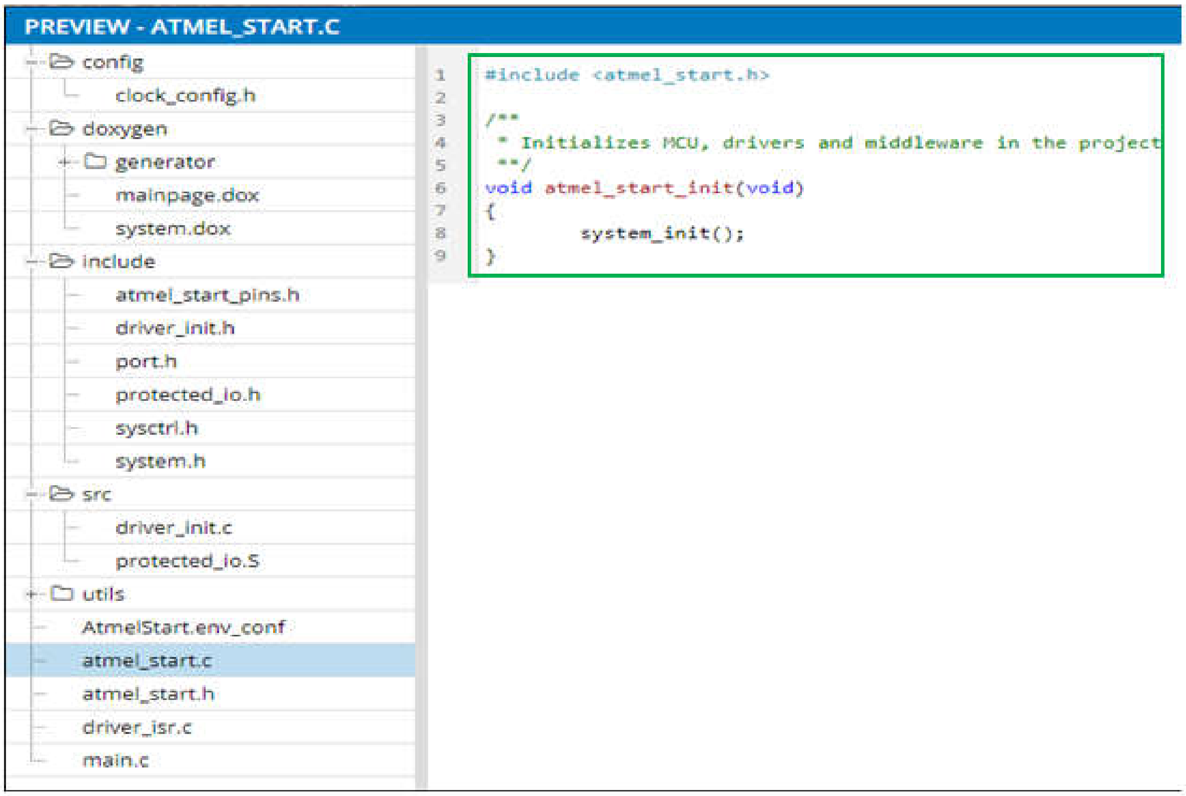
Figure 1: Block diagram of Hardware Abstraction Layer

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/hardware-abstraction-layer.pdf, page 5.



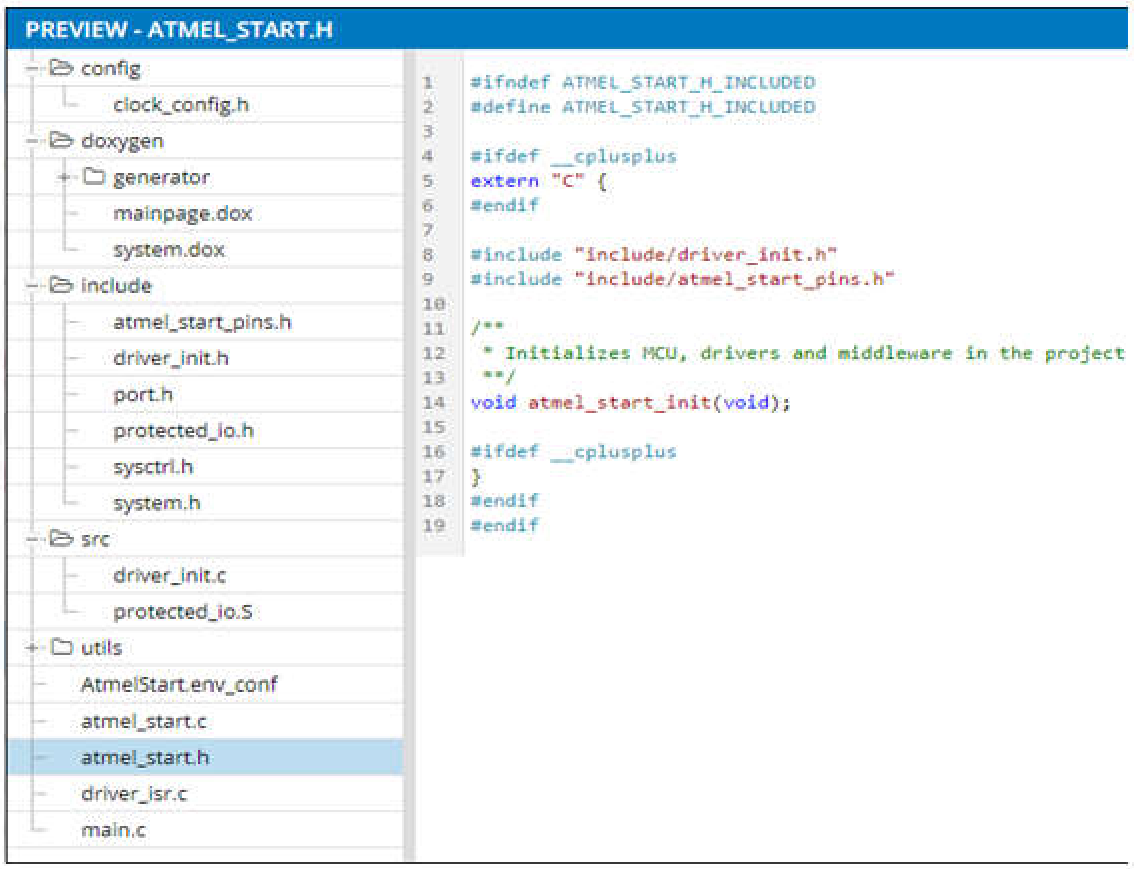
```
/**
\mainpage

\section doc_driver_mainpage_intro Introduction

AVR Code is a software library containing drivers, middleware and
applications for AVR microcontrollers. START, a web-based
GUI, is used to generate C-code according to the user's needs.

In START, the user selects the device for which to generate code.
Thereafter, the user selects the driver and middleware components
that he wants to include in the project. As an example, the user
may choose the device Tiny817, and add the USART and SPI driver.
The user can use START to configure the detailed behaviour of the
drivers, and configure the device's clock system and I/O pins.
Finally, the user exports the project as an atzip-file, which
can be opened in Studio or IAR Embedded Workbench. The
project contains:
- C and assembly files that can be compiled using the GCC
or IAR compilers.
- A directory with doxygen-files documenting the drivers included in the project.
The doxygen-files can be built into a HTML-document by executing the
command "doxygen doxyfile.doxygen" on a computer where doxygen is installed.
The file "doxyfile.doxygen" is found in the "doxygen/generator" directory in
the atzip-file.
- Example code documenting the usage of the configured drivers

\section doc_driver_mainpage_driver_overview Driver Overview

After the user has selected a device to configure, he will be able
to add and configure drivers and middleware available for that
device. In START, drivers are grouped by functionality, such as
```

Source: http://start.atmel.com/#dashboard/export.



```
#include <atmel_start.h>

/**
 * Initializes MCU, drivers and middleware in the project
 **/
void atmel_start_init(void)
{
        system_init();
}
```

Source: http://start.atmel.com/#dashboard/export.



```
#ifndef ATMEL_START_H_INCLUDED
#define ATMEL_START_H_INCLUDED

#ifdef __cplusplus
extern "C" {
#endif

#include "include/driver_init.h"
#include "include/atmel_start_pins.h"

/**
 * Initializes MCU, drivers and middleware in the project
 **/
void atmel_start_init(void);

#ifdef __cplusplus
}
#endif
#endif
```

Source: http://start.atmel.com/#dashboard/export.

24. Atmel Start generates specific application handler code to associate the generic application functions with specific functions of a device driver for at least one of the types of the hardware modules. For example, in addition to the generic drivers and HAL, Atmel Start also includes specific application handler code that is specific to the application and specific to particular microcontroller families. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

1.2 Components of Hardware Abstraction Layer

Figure 1 shows a representative block diagram of the Hardware Abstraction Layer in relation to the MC Application Framework and the customer application.

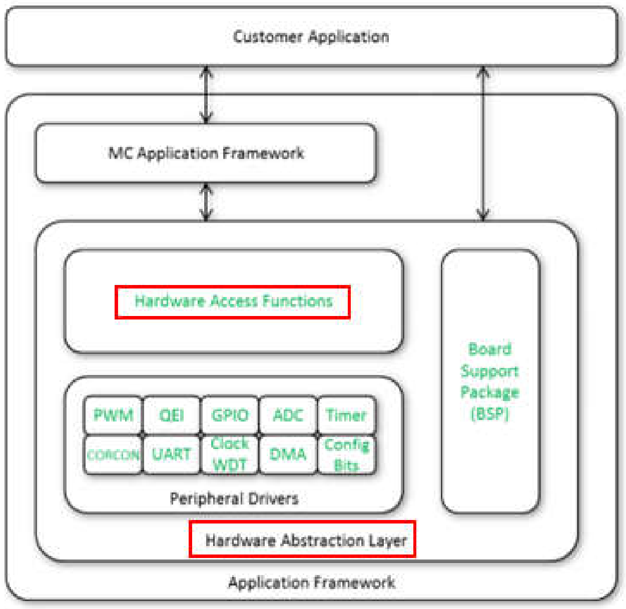
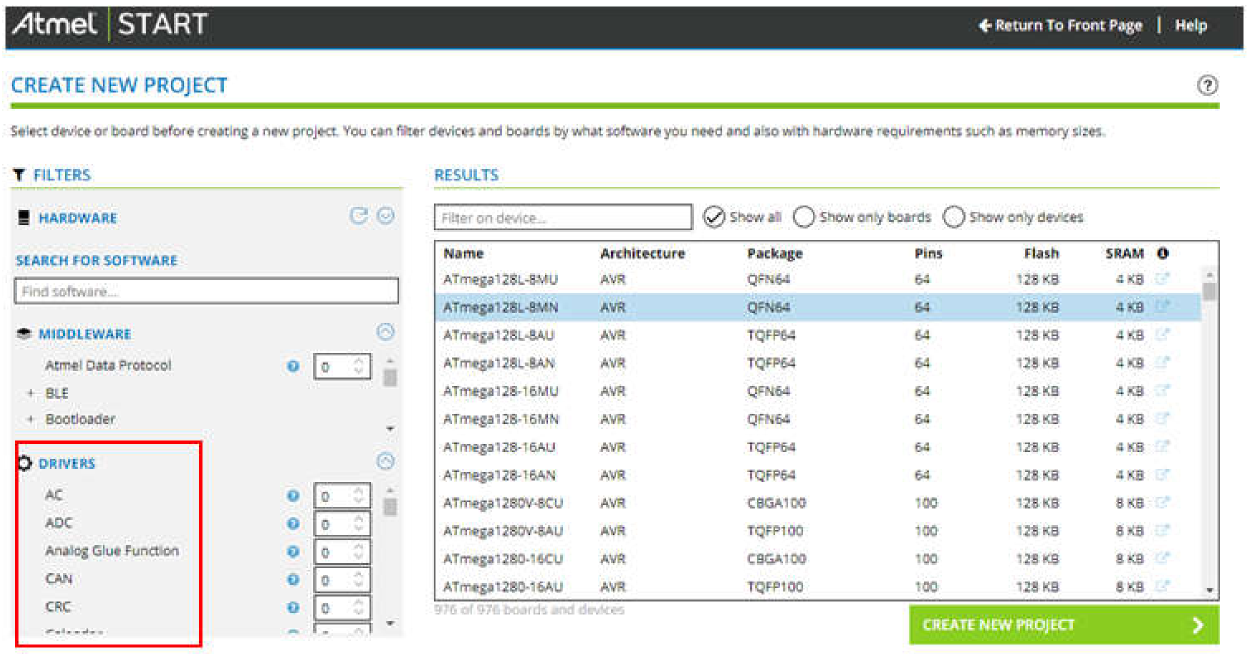


Figure 1: Block diagram of Hardware Abstraction Layer

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/hardware-abstraction-layer.pdf, page 5.



CREATE NEW PROJECT

Select device or board before creating a new project. You can filter devices and boards by what software you need and also with hardware requirements such as memory sizes.

FILTERS

HARDWARE

SEARCH FOR SOFTWARE

Find software...

MIDDLEWARE

Atmel Data Protocol	0
+ BLE	
+ Bootloader	

DRIVERS

AC	0
ADC	0
Analog Glue Function	0
CAN	0
CRC	0

RESULTS

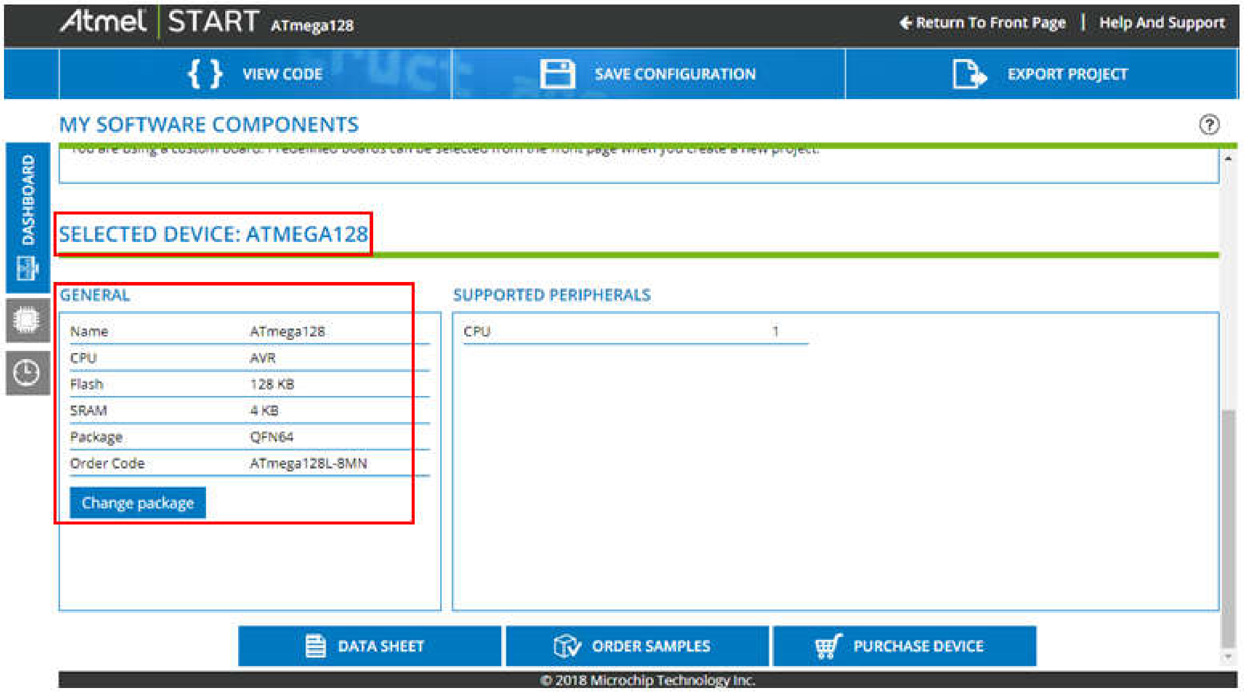
Filter on device... Show all | Show only boards | Show only devices

Name	Architecture	Package	Pins	Flash	SRAM
ATmega128L-8MU	AVR	QFN64	64	128 KB	4 KB
ATmega128L-8MN	AVR	QFN64	64	128 KB	4 KB
ATmega128L-8AU	AVR	TQFP64	64	128 KB	4 KB
ATmega128L-8AN	AVR	TQFP64	64	128 KB	4 KB
ATmega128-16MU	AVR	QFN64	64	128 KB	4 KB
ATmega128-16MN	AVR	QFN64	64	128 KB	4 KB
ATmega128-16AU	AVR	TQFP64	64	128 KB	4 KB
ATmega128-16AN	AVR	TQFP64	64	128 KB	4 KB
ATmega1280V-8CU	AVR	CBGA100	100	128 KB	8 KB
ATmega1280V-8AU	AVR	TQFP100	100	128 KB	8 KB
ATmega1280-16CU	AVR	CBGA100	100	128 KB	8 KB
ATmega1280-16AU	AVR	TQFP100	100	128 KB	8 KB

976 of 976 boards and devices

CREATE NEW PROJECT

Source: http://start.atmel.com/#project.



SELECTED DEVICE: ATMEGA128

GENERAL

Name	ATmega128
CPU	AVR
Flash	128 KB
SRAM	4 KB
Package	QFN64
Order Code	ATmega128L-8MN

Change package

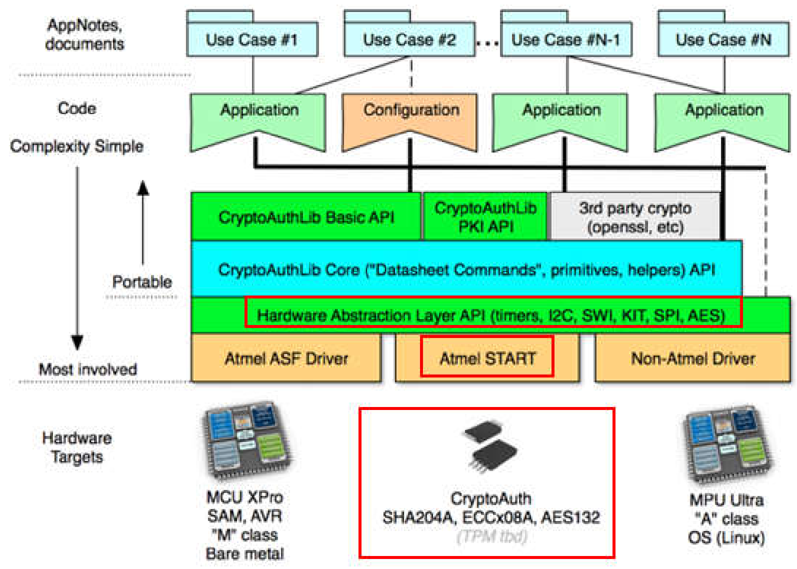
SUPPORTED PERIPHERALS

CPU	1

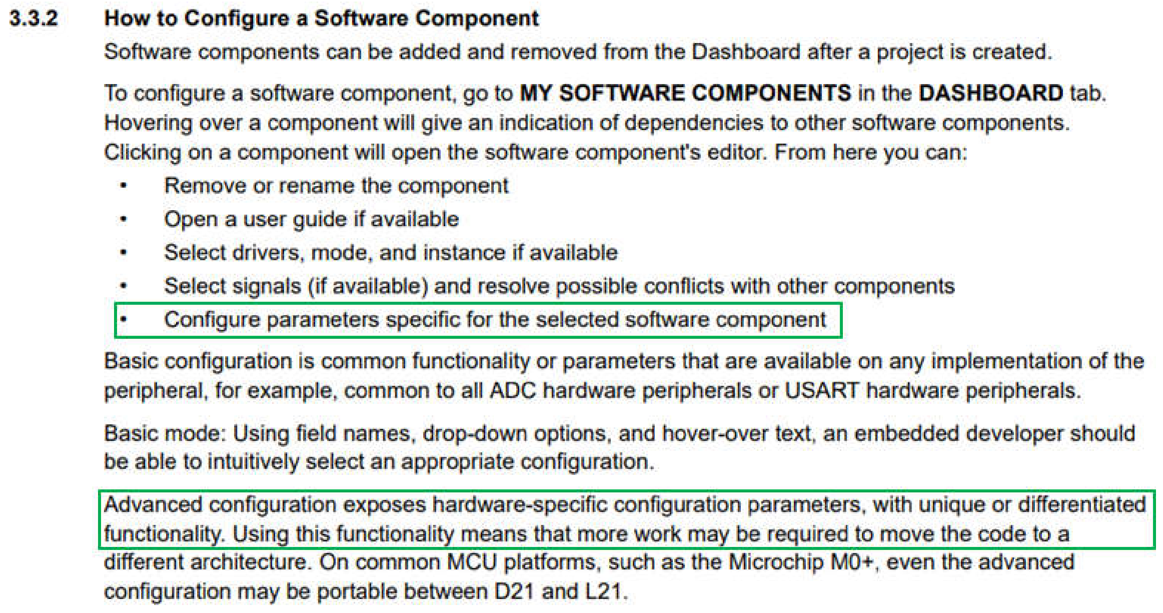
Source: http://start.atmel.com/#dashboard

25. Atmel Start generates specific application handler code and defines a specific element in the specific code to be handled by one of the generic application functions for that hardware module. For example, Atmel Start generates system-specific application handler code by defining hardware-specific configuration parameters that extend or otherwise connect the system-specific application handler code and data structures made available by the generic application handler code of Atmel Start. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

Figure 2-1. General Architecture


AppNotes, documents
Use Case #1
Use Case #2
Use Case #N-1
Use Case #N
Code
Complexity Simple
Application
Configuration
Application
Application
CryptoAuthLib Basic API
CryptoAuthLib PKI API
3rd party crypto (openssl, etc)
CryptoAuthLib Core ("Datasheet Commands", primitives, helpers) API
Portable
Hardware Abstraction Layer API (timers, I2C, SWI, KIT, SPI, AES)
Atmel ASF Driver
Atmel START
Non-Atmel Driver
Most involved
Hardware Targets
MCU XPro SAM, AVR "M" class Bare metal
CryptoAuth SHA204A, ECCx08A, AES132 (TPM tbd)
MPU Ultra "A" class OS (Linux)

Source: http://ww1.microchip.com/downloads/en/AppNotes/Atmel-8984-CryptoAuth-CryptoAuthLib-ApplicationNote.pdf, page 5.



3.3.2 How to Configure a Software Component

Software components can be added and removed from the Dashboard after a project is created.

To configure a software component, go to **MY SOFTWARE COMPONENTS** in the **DASHBOARD** tab. Hovering over a component will give an indication of dependencies to other software components. Clicking on a component will open the software component's editor. From here you can:

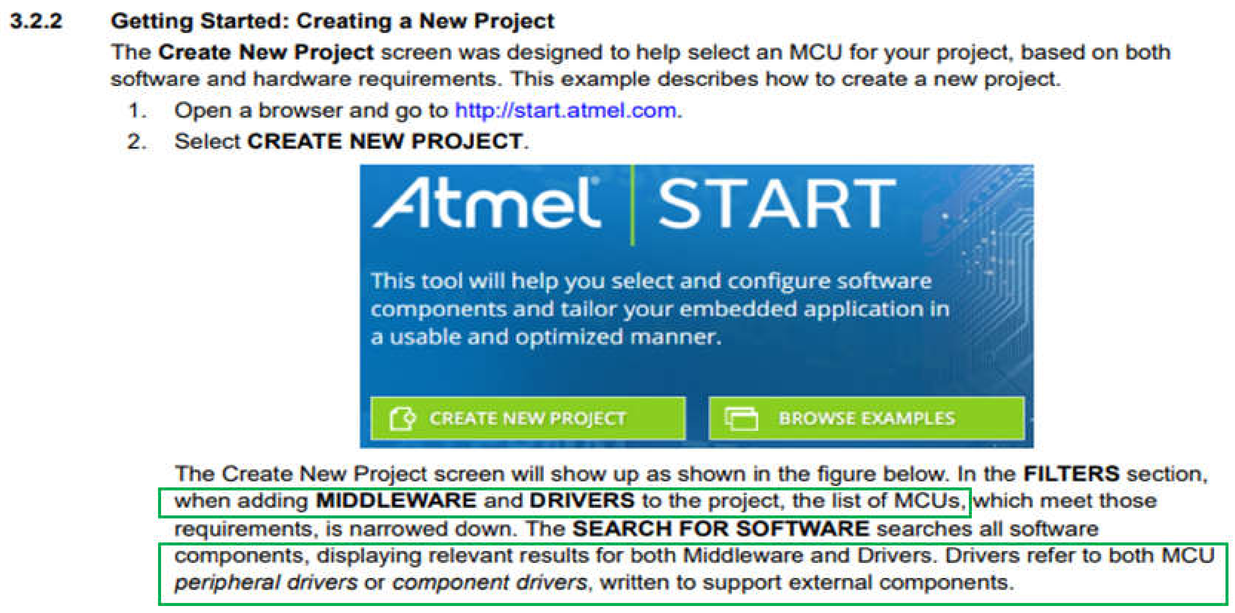
- Remove or rename the component
- Open a user guide if available
- Select drivers, mode, and instance if available
- Select signals (if available) and resolve possible conflicts with other components
- Configure parameters specific for the selected software component

Basic configuration is common functionality or parameters that are available on any implementation of the peripheral, for example, common to all ADC hardware peripherals or USART hardware peripherals.

Basic mode: Using field names, drop-down options, and hover-over text, an embedded developer should be able to intuitively select an appropriate configuration.

Advanced configuration exposes hardware-specific configuration parameters, with unique or differentiated functionality. Using this functionality means that more work may be required to move the code to a different architecture. On common MCU platforms, such as the Microchip M0+, even the advanced configuration may be portable between D21 and L21.

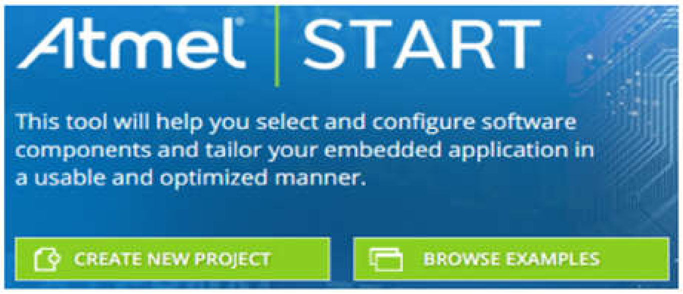
Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-START-User-Guide-DS50002793A.pdf, page 19.



3.2.2 Getting Started: Creating a New Project

The **Create New Project** screen was designed to help select an MCU for your project, based on both software and hardware requirements. This example describes how to create a new project.

1. Open a browser and go to http://start.atmel.com.
2. Select **CREATE NEW PROJECT**.

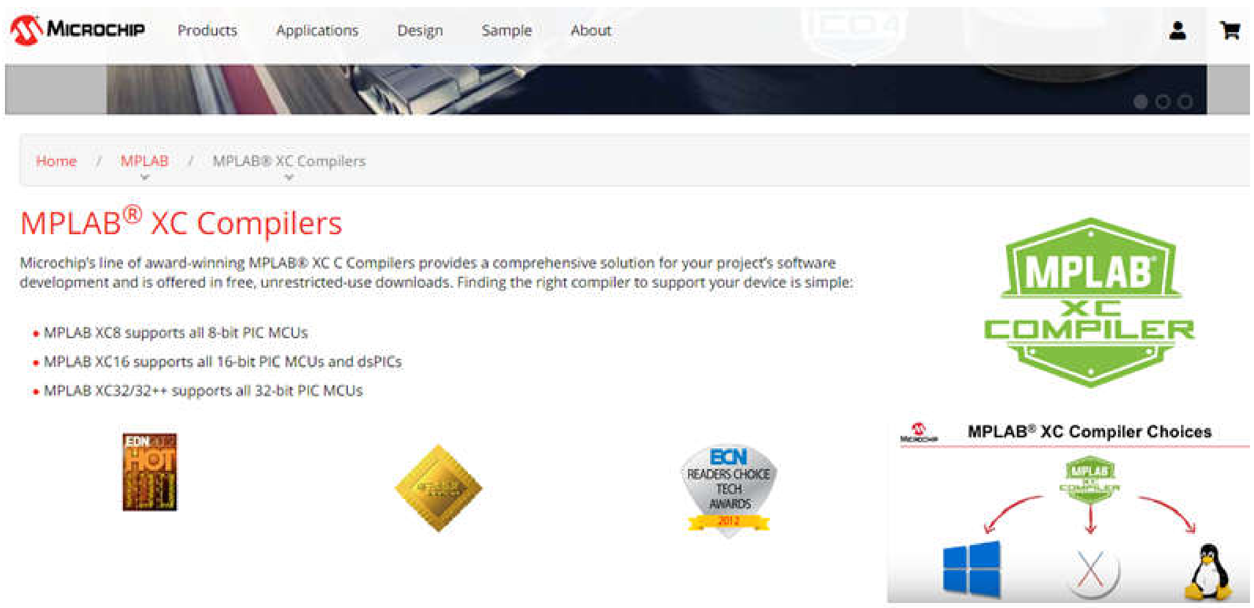


This tool will help you select and configure software components and tailor your embedded application in a usable and optimized manner.

The Create New Project screen will show up as shown in the figure below. In the **FILTERS** section, when adding **MIDDLEWARE** and **DRIVERS** to the project, the list of MCUs, which meet those requirements, is narrowed down. The **SEARCH FOR SOFTWARE** searches all software components, displaying relevant results for both Middleware and Drivers. Drivers refer to both MCU *peripheral drivers* or *component drivers*, written to support external components.

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-START-User-Guide-DS50002793A.pdf, page 12.

26. Atmel Start compiles the generic application handler programs together with the specific application handler code to produce machine-readable code to be executed by an embedded processor in the at least one of the types of the hardware modules. For example, when a specific application is needed for a particular hardware, the generic functions and the specific functions are compiled together to yield a machine readable code. Microchip Technology and/or its customers compile the generic functions and the specific functions using MPLAB XC Compilers and/or any other compiling SDK supported by Microchip Technology. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

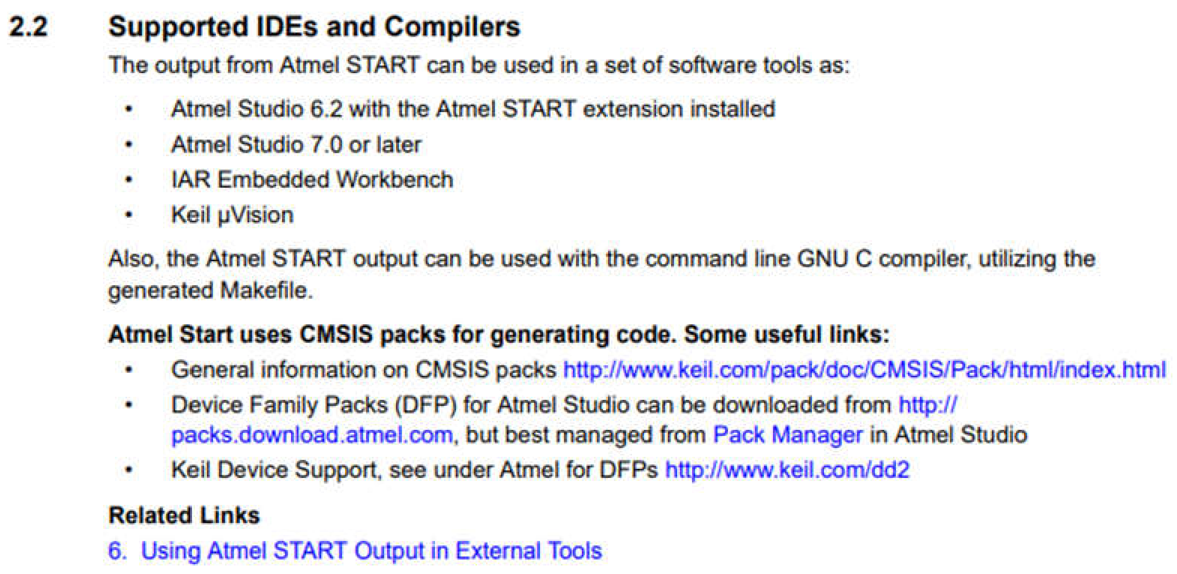

Products
Applications
Design
Sample
About
Home / MPLAB / MPLAB® XC Compilers
MPLAB XC COMPILER
READERS CHOICE TECH AWARDS
MPLAB® XC Compiler Choices

MPLAB® XC Compilers

Microchip's line of award-winning MPLAB® XC C Compilers provides a comprehensive solution for your project's software development and is offered in free, unrestricted-use downloads. Finding the right compiler to support your device is simple:

- MPLAB XC8 supports all 8-bit PIC MCUs
- MPLAB XC16 supports all 16-bit PIC MCUs and dsPICs
- MPLAB XC32/32++ supports all 32-bit PIC MCUs

Source: http://www.microchip.com/mplab/compilers.



2.2 Supported IDEs and Compilers

The output from Atmel START can be used in a set of software tools as:

- Atmel Studio 6.2 with the Atmel START extension installed
- Atmel Studio 7.0 or later
- IAR Embedded Workbench
- Keil µVision

Also, the Atmel START output can be used with the command line GNU C compiler, utilizing the generated Makefile.

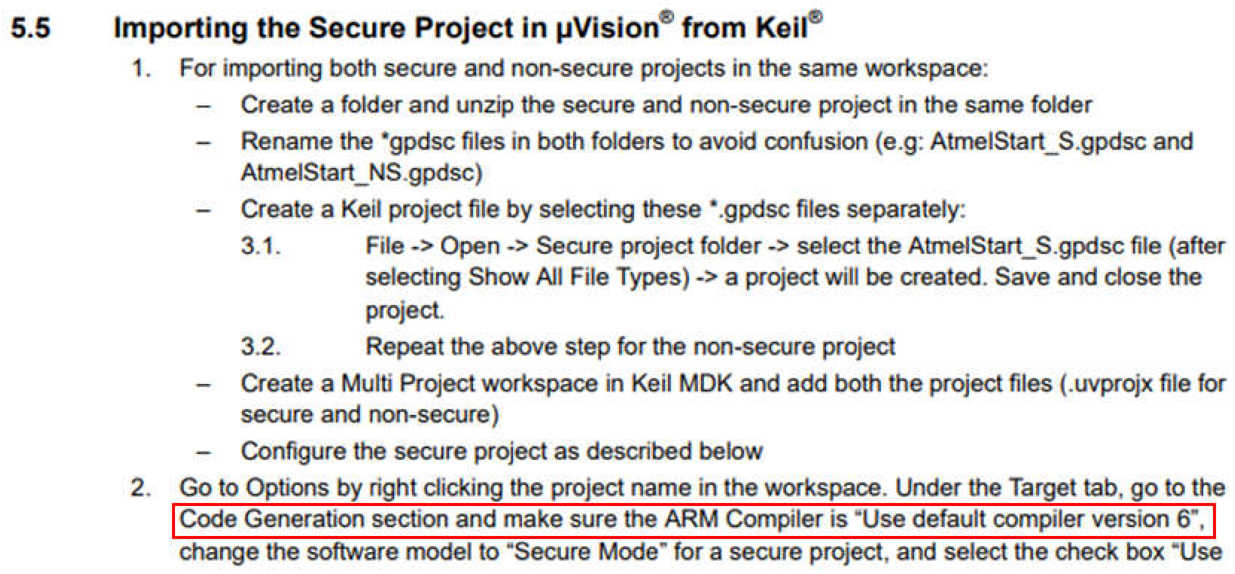
Atmel Start uses CMSIS packs for generating code. Some useful links:

- General information on CMSIS packs http://www.keil.com/pack/doc/CMSIS/Pack/html/index.html
- Device Family Packs (DFP) for Atmel Studio can be downloaded from http://packs.download.atmel.com, but best managed from Pack Manager in Atmel Studio
- Keil Device Support, see under Atmel for DFPs http://www.keil.com/dd2

Related Links

6. Using Atmel START Output in External Tools

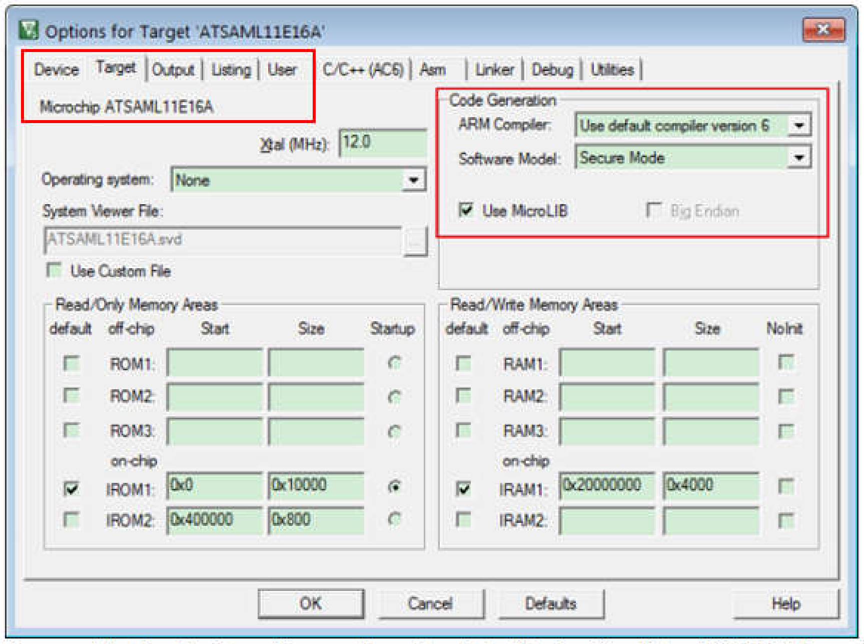
Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-START-User-Guide-DS50002793A.pdf, page 8.



5.5 Importing the Secure Project in µVision® from Keil®

1. For importing both secure and non-secure projects in the same workspace:
   - Create a folder and unzip the secure and non-secure project in the same folder
   - Rename the *gpdsc files in both folders to avoid confusion (e.g: AtmelStart_S.gpdsc and AtmelStart_NS.gpdsc)
   - Create a Keil project file by selecting these *.gpdsc files separately:
     - 3.1. File -> Open -> Secure project folder -> select the AtmelStart_S.gpdsc file (after selecting Show All File Types) -> a project will be created. Save and close the project.
     - 3.2. Repeat the above step for the non-secure project
   - Create a Multi Project workspace in Keil MDK and add both the project files (.uvprojx file for secure and non-secure)
   - Configure the secure project as described below
2. Go to Options by right clicking the project name in the workspace. Under the Target tab, go to the Code Generation section and make sure the ARM Compiler is "Use default compiler version 6", change the software model to "Secure Mode" for a secure project, and select the check box "Use

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-START-User-Guide-DS50002793A.pdf, page 50.



Options for Target 'ATSAML11E16A'

Microchip ATSAML11E16A

Xtal (MHz): 12.0

Operating system: None

System Viewer File: ATSAML11E16A.svd

Code Generation

ARM Compiler: Use default compiler version 6

Software Model: Secure Mode

[✓] Use MicroLIB

[ ] Big Endian

Read/Only Memory Areas default	off-chip	Start	Size	Startup
[ ]	ROM1:			○
[ ]	ROM2:			○
[ ]	ROM3:			○
	on-chip			
[✓]	IROM1:	0x0	0x10000	●
[ ]	IROM2:	0x400000	0x800	○

Read/Write Memory Areas default	off-chip	Start	Size	NoInit
[ ]	RAM1:			[ ]
[ ]	RAM2:			[ ]
[ ]	RAM3:			[ ]
	on-chip			
[✓]	IRAM1:	0x20000000	0x4000	[ ]
[ ]	IRAM2:			[ ]

Source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-START-User-Guide-DS50002793A.pdf, page 51.

27. Defendant's actions complained of herein will continue unless Defendant is enjoined by this court.

28. Defendant's actions complained of herein are causing irreparable harm and monetary damage to Plaintiff and will continue to do so unless and until Defendant is enjoined and restrained by this Court.

29. Plaintiff is in compliance with 35 U.S.C. § 287.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff asks the Court to:

(a) Enter judgment for Plaintiff on this Complaint on all causes of action asserted herein;

(b) Enter an Order enjoining Defendant, its agents, officers, servants, employees, attorneys, and all persons in active concert or participation with Defendant who receive notice of the order from further infringement of United States Patent No. 7,069,546 (or, in the alternative, awarding Plaintiff a running royalty from the time of judgment going forward);

(c) Award Plaintiff damages resulting from Defendant's infringement in accordance with 35 U.S.C. § 284;

(d) Award Plaintiff pre-judgment and post-judgment interest and costs; and

(e) Award Plaintiff such further relief to which the Court finds Plaintiff entitled under law or equity.

Dated: January 31, 2019

Respectfully submitted,

/s/ Jay Johnson

JAY JOHNSON
State Bar No. 24067322
D. BRADLEY KIZZIA
State Bar No. 11547550
KIZZIA JOHNSON, PLLC
1910 Pacific Ave., Suite 13000
Dallas, Texas 75201
(214) 451-0164
Fax: (214) 451-0165
jay@kjpllc.com
bkizzia@kjpllc.com

ATTORNEYS FOR PLAINTIFF

EXHIBIT A