XMC1000, XMC4000

32-bit Microcontroller Series for Industrial Applications

Using DAVE™ with Rowley CrossWorks for ARM®

AP32220

Application Note

Scope and purpose

This application note describes how to use the Infineon DAVE™ in conjuction with the tool chain CrossWorks™ for ARM from Rowley Associates Limited, UK. While DAVE is used for configuring the micro controller and the DAVE Apps are applied to provide peripheral drivers and other, CrossWorks is used for the build process and debugging.

The following is covered within the Application Note:

- Installation requirements for both tools
- How to use DAVE™ in parallel with CrossWorks™ for ARM tools
- Get started with project development based on evaluation kits
- Tips and Tricks

Applicable Products

- XMC1000 Microcontroller Family
- XMC4000 Microcontroller Family

References

Infineon: DAVE™, http://www.infineon.com/DAVE

Rowley: CrossWorks™ for ARM, http://www.rowley.co.uk

Table of Contents

1 Introduction

1.1 Audience

It is assumed that the reader either already has some basic knowledge of how to work with the free DAVE™ code generation tool from Infineon, and/or the Rowley CrossWorks™ for ARM tool, or that they have sufficient technical knowledge to learn how to use these tools with the setup and integration guidance provided in this document.

1.2 Overview

The XMC microcontroller family based on ARM® Cortex™-M cores, is dedicated to applications in the field of renewable energy, factory and building automation, transportation, logistics and medical equipment, lighting and home appliances.

With the highest quality standards and long product family life times, the XMCs combine Infineon's powerful peripheral set, configurable to specific application requirements, with a fast embedded Flash, and the ability to support high ambient temperature ranges.

The XMC microcontroller portfolio features a wide range of products from low-end, low-pin-count devices up to advanced solutions for industrial applications. The XMC1000 integrates the ARM® Cortex™-M0 to a leading-edge 65nm manufacturing process to overcome the limitations of today's 8-bit designs. [2] The XMC4000 combines Infineon's leading-edge peripheral set with an industry-standard ARM® Cortex™-M4 core. [1]

With the XMC family being based on the ARM® Cortex™-M microcontroller core, it is possible to choose from any of the wide variety of development tools and software solutions available on the market today.

The traditional approach to enable support for a specific microcontroller family is to integrate a driver library into the respective tool chain. Infineon have taken a different approach by providing the free, flexible code generator DAVE™.

DAVE™ provides a graphical user interface to access application descriptions – 'Apps'. These Apps can be included into the project, individually configured, combined and linked to each other in order to create a system of peripheral drivers and/or middle-ware type communication stacks. DAVE™ will map the application requirements to the resources of the chosen microcontroller and generate standard C code with a well-documented API. [3]

With the current release of DAVE™ the code generation is integrated with a complete Eclipse based IDE and a tool chain that uses the GNU Compiler Collection GCC. A debugger and flash loader are also part of the package. Although this gives a complete set of development tools, customers are still free to use more sophisticated development environments. One such software development tool is CrossWorks™ for ARM offered by Rowley Associates Ltd.

CrossWorks™ for ARM is a complete C/C++ and assembly code development system for ARM7, ARM9, XScale, and Cortex™ microcontrollers. Because it is based on the GNU Compiler Collection, and offers a board support package for the XMC family of microcontrollers from Infineon, it can, as we shall describe, easily work together with the DAVE™ code generation tool.[4]

2 Tools, Installation and Setup

The descriptions in this document are based on the following tools and revisions being installed and working correctly:

- **DAVE™, rev. 3.1.10 [3]**
- **CrossWorks™ for ARM, rev. 3.2 [4]**
- **General Purpose Application Kit [5]**

Note: A more detailed overview about revisions is given in the Appendix.

DAVE™ is integrated into the Eclipse environment, and the full package is available for download for free from the Infineon website. The user has the possibility to either integrate the code generation tool directly into an existing Eclipse IDE, or to add professional tools such as a compiler, debugger or code management in to the DAVE™ environment.

CrossWorks™ for ARM is not based on Eclipse so it cannot be directly integrated into the DAVE™ environment.

The setup described here will show how to create a common project and work with both development tools in parallel by combining:

- the Apps library and code generator part of the DAVE™ package with
- CrossWorks™ for ARM for the project management, adding application code and for the compile, link & debug steps.

Attention: For additional information on using DAVE™ or the CrossWorks™ for ARM tools, please read the respective User Manuals or refer to the online help. In case of any inconsistency, the User Manuals take precedence.

Installation

There are no specific requirements regarding the installation. Both tools can be setup to use independent directories, and the files generated by the DAVE™ code generator will only be included by reference into the respective CrossWorks project.

Licensing

While DAVE™ is free and does not need any activation to begin using it, CrossWorks™ requires a license for compiling and debugging. Rowley Associates Ltd. offers a 30-day evaluation license.

3 Auto Code Generation with DAVE™

The first step is to create a new project in DAVE™. We will choose one of the example projects, PWMSP001_CNT001_Example1 which is available for all XMC microcontrollers.

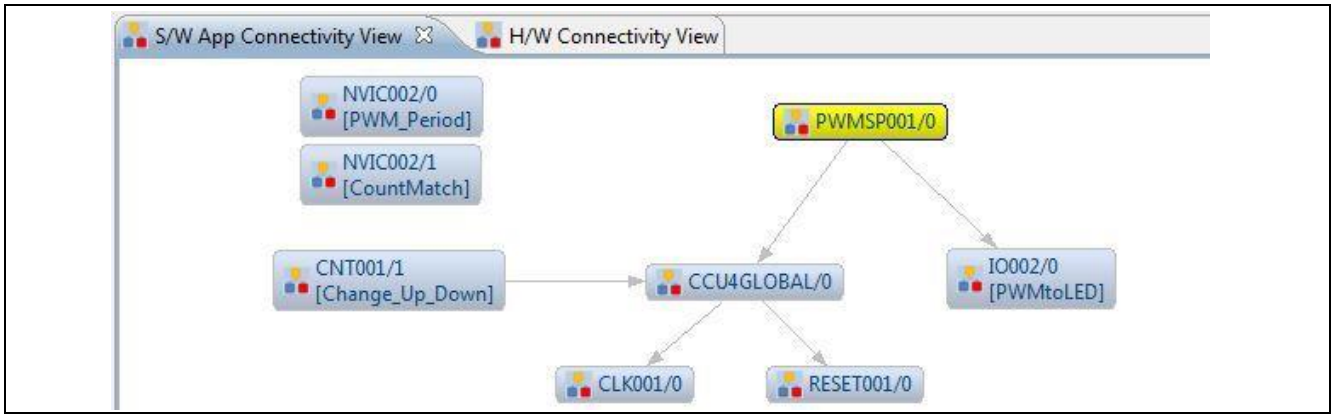

Figure 1 PWMSP001_CNT001_Example1

This example uses a CCU4 timer in PWM output mode to drive the on-board LED and control its brightness.

NVIC handlers are used to continuously change the brightness by modifying the PWM pulse width and also to change the dimming direction.

The project is setup and configured in DAVE™, and then the Apps are chosen, connected and configured. In the last stage the App requirements are mapped on to the microcontroller resources and the C code is generated.

Note: DAVE 3.1.10 and newer provide migration support for both XMC microcontroller and DAVE Apps making it possible to update projects.

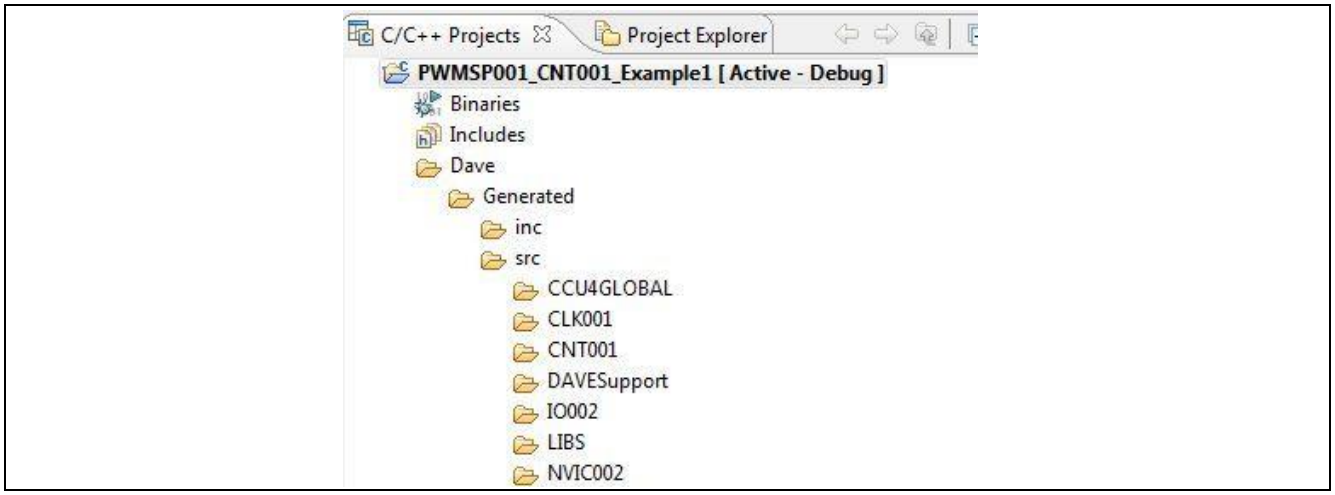

Figure 2 DAVE™ Code Generation

Attention: Generated code is placed into the sub-directories inc and src under .|Dave|Generated. This code is always overwritten, so modifications or additional code must never be placed there.

4 Rowley CrossWorks™ for ARM

CrossWorks™ for ARM provides support for individual microcontrollers via Board Support Packages (BSP) which can be individually installed.

The BSPs for the XMC series contain the definition for the memory maps of the individual devices and the flash loader. Instead of using a traditional driver library for BSP integration, we describe instead how to work with the DAVE™ generated code.

Global Macros

It is recommended to define the path to the DAVE™ workspace as well as include directories as global macros instead of using absolute paths. This only has to be done once for all future solutions and projects, and makes it easier to move the DAVE™ workspace or installation directory to a different location later on.

Use `Project => Macros`, as shown in the Figure 3.

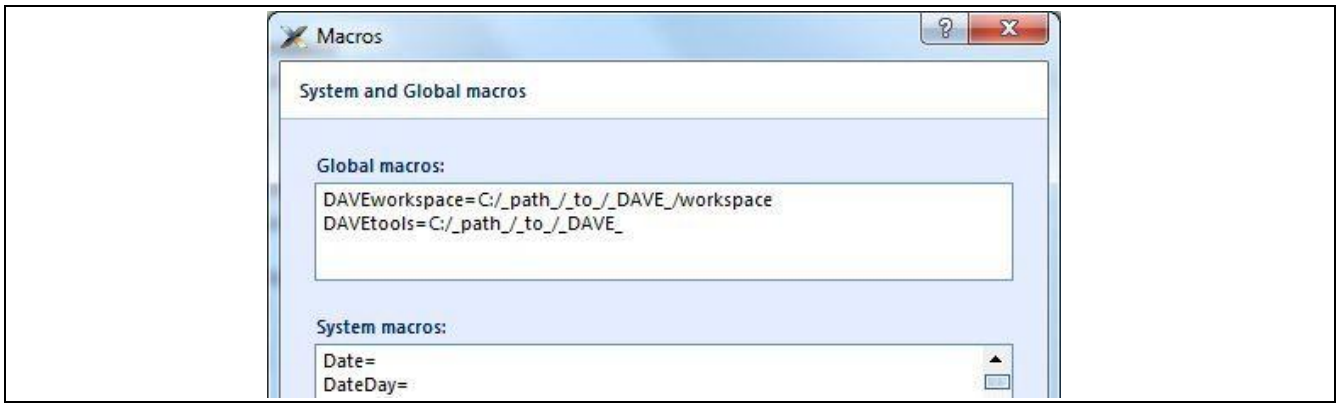

Figure 3 DAVE™ Workspace macro

Include and Header Files

The search path for the include files within a project has to be extended as follows. (configuration 'Common')

First CMSIS does not define a way to differentiate between individual devices and/or device steps. Infineon has implemented this into DAVE™ via the definition of a unique type id `UC_ID`. A header file `uc_id.h` is used to define symbols for the specific type, package and step. These symbols are than in turn referred to by the code generated from the DAVE™ Apps.

A project has to define the `UC_ID` and set the path to the file `uc_id.h` by using the macro defined above:

`$(DAVEtools)/CMSIS/Infineon/Include`

The individual header files for the code generated from DAVE™ Apps are referenced from the central `DAVE3.h` with a relative path, so only the path to the top level `DAVE3.h` file has to be added to the user includes of the project/solution.

This is done using the predefined DAVE™ workspace macro.

```
$(DAVEworkspace)/PWMSP001_CNT001_Example1/Dave/Generated/inc/DAVESupport
```

Depending upon the Apps and libraries used it might be necessary to insert additional search paths.

For DAVE™ Apps and libraries this can easily be copied over from the `Include` directory in the DAVE™ project view. (ref. to Figure 4 and Figure 5)

In general it is recommended to use macros instead of absolute path names.

Figure 4 Include Search Path for DAVE3.h and uc_id.h

Preprocessor Definitions

As describe in the section above the `UC_ID` of the device used for a project has to be defined. This `UC_ID` can be taken from the file `uc_id.h` in the respective DAVE™ project. As DAVE™ color codes the respective section, it is easily found by opening `uc_id.h`.

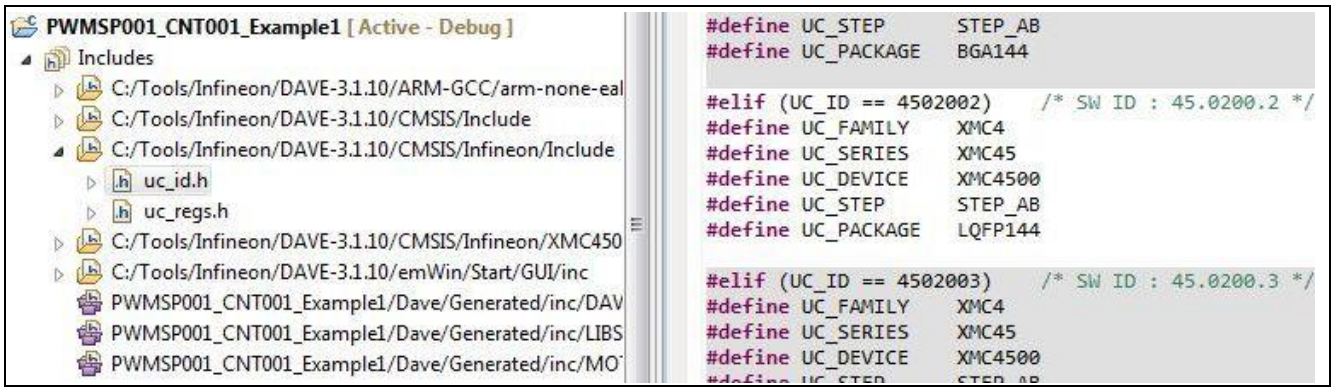

Figure 5 UC_ID in uc_id.h in DAVE™

In addition to the `UC_ID` the symbol `DAVE_CE` has to be defined as this is used within the DAVE™ code to decide on the inclusion of some additional header files.

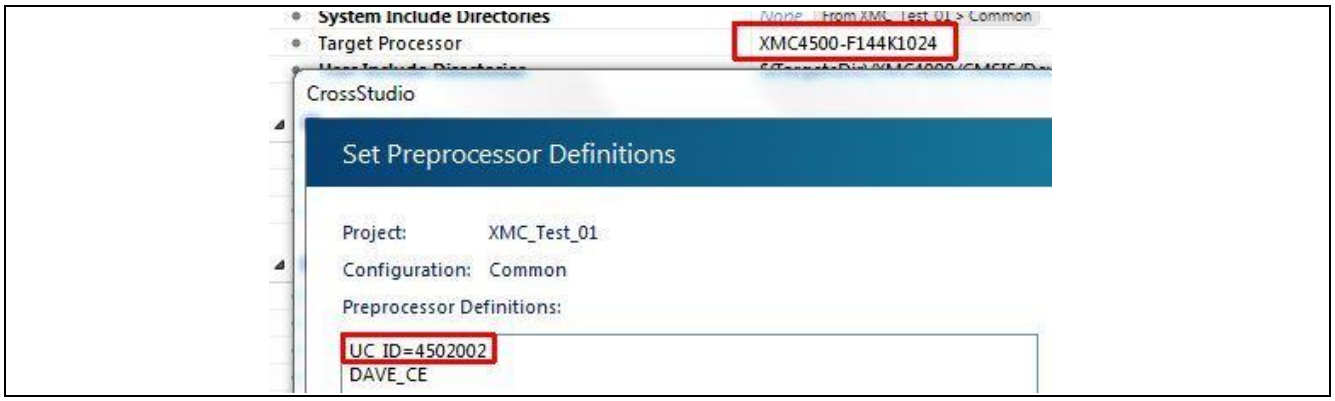

Figure 6 Preprocessor Definitions in CrossWorks

Dynamic Folder

A new project is created in CrossWorks with the same XMC device as chosen in the respective DAVE™ project.

In order to integrate the code generated by DAVE™, we use the `Dynamic Folder feature` provided by CrossWorks™. CrossWorks™ allows the folder to specify a dynamic folder upon creation. The macro defined above is used to point to the DAVE generated files, as shown below.

The file filter should be set for the *.c;*h files and '`Recurse into subdirectories`' activated.

Select 'OK' and CrossWorks™ will show the source files that were generated by DAVE™.

It is also possible to add a dynamic folder using drag & drop simply dropping the respective DAVE™ folder into the project. In this case CrossWorks™ will use the absolute path and only add the files in the root folder. With the `Setup` command the absolute path can be changed to the macro as described above and the option '`Recurse into subdirectories`' has to be set.

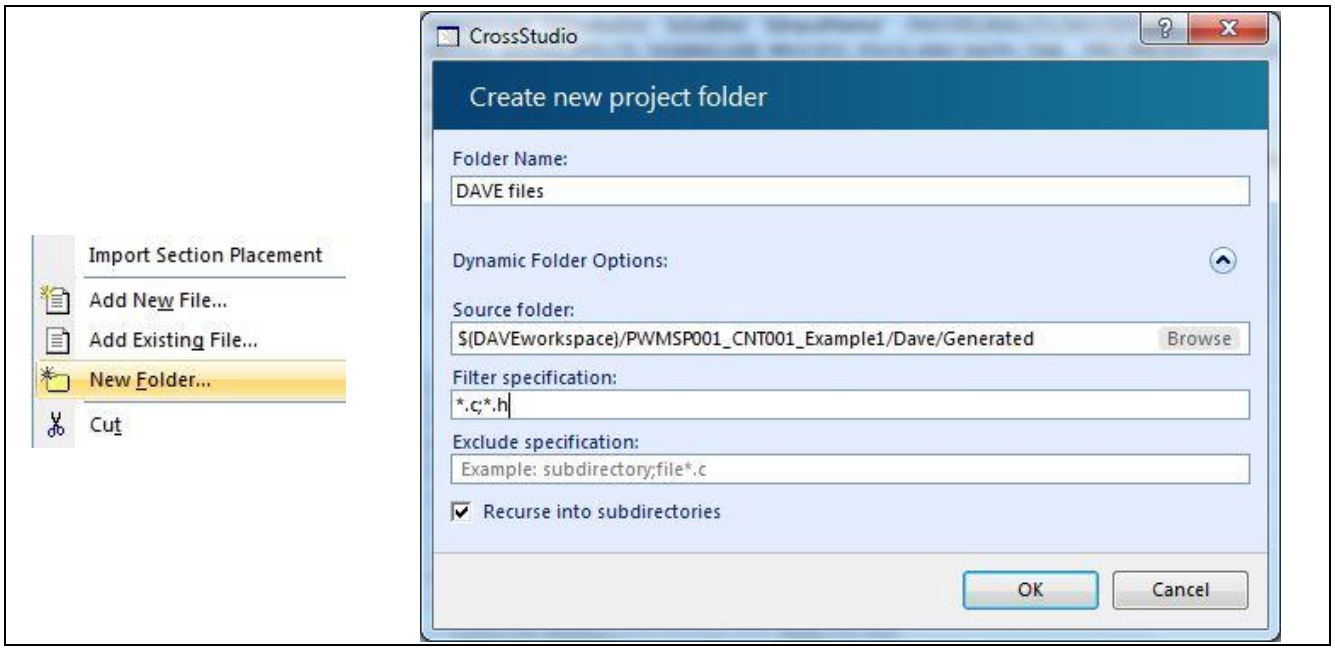

Figure 7 Setting up the Dynamic Folder

The project is now setup and ready for the addition of the actual application code.

As a starting point the `main.c` generated by DAVE™ can be imported into the project by using:

```
Add Existing File -> Import
```

At this point the project should build without errors and now can be modified and extended as needed.

CrossWorks™ monitors the 'Dynamic Folder' in the background, automatically adds newly generated files to the project and notifies the user if files have been changed. This also applies to files that are currently open in the IDE or the debugger.

It is also possible to manually refresh the folder content as shown below.

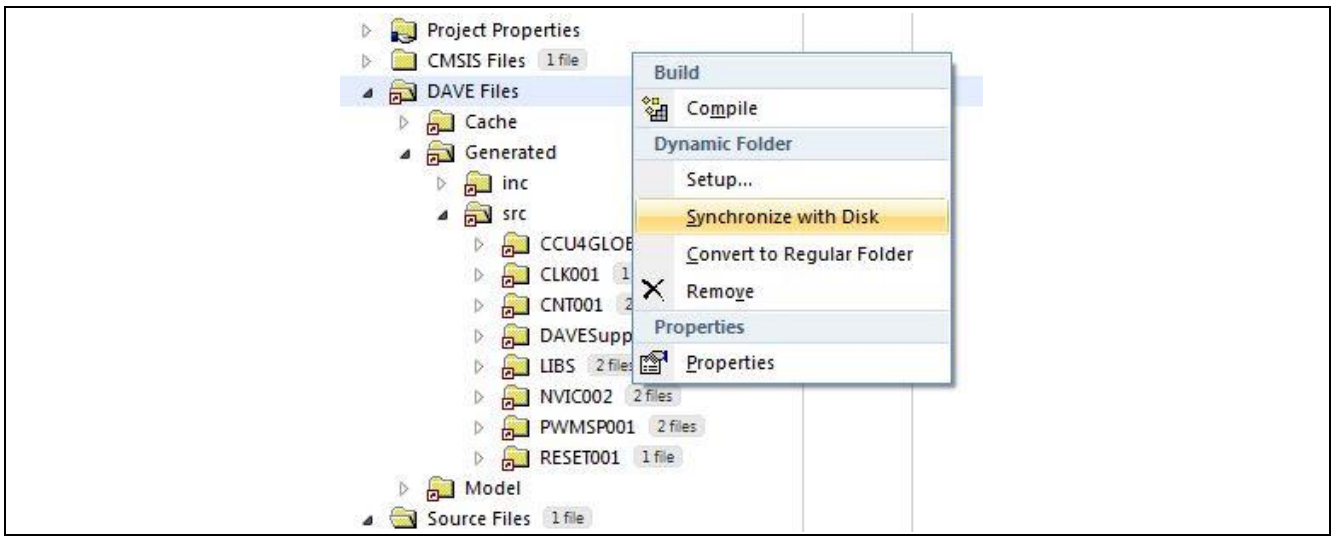

Figure 8 Reloading the Dynamic Folder

Interrupts

DAVE™ implements the standard method of weakly defined symbols to populate the NVIC interrupt table. If interrupts are given customized names in the GUI of the NVIC App like `PWM_Period_Handler` DAVE™ will redefine the symbol for the NVIC interrupt but it will not create the function body.

This has to be done by the user.

```
NVIC002_Conf.h:          #define PWM_Period_Handler    IRQ_Hdlr_49
XMC4500.h:               #define IRQ_Hdlr_49  CCU41_1_IRQHandler
Startup_XMC45600.s:   Entry   CCU41_1_IRQHandler
```

The following table summarizes the most important settings.

Table 1 Summary of Settings in CrossWorks

Global Macros	
	`DAVEworkspace=C:/_path_/_to_/_DAVE_/workspace`
	`DAVEtools=C:/_path_/_to_/_DAVE_/tools`
Include Search Paths	
	`$(DAVEtools)/CMSIS/Infineon/Include`
	`$(DAVEworkspace)/_project_/Dave/Generated/inc/DAVESupport`
Preprocessor definitions	
	`UC_ID=4502002` (UC_ID of target processor)
	`DAVE_CE`

5 Usage and Test

Once all of the described setup steps have been completed, both tools can be used in parallel and independently of each other.

Within DAVE™ the code generation is used to generate the *.c and *.h files as required. On the application level they can be seen to form of a project specific driver library.

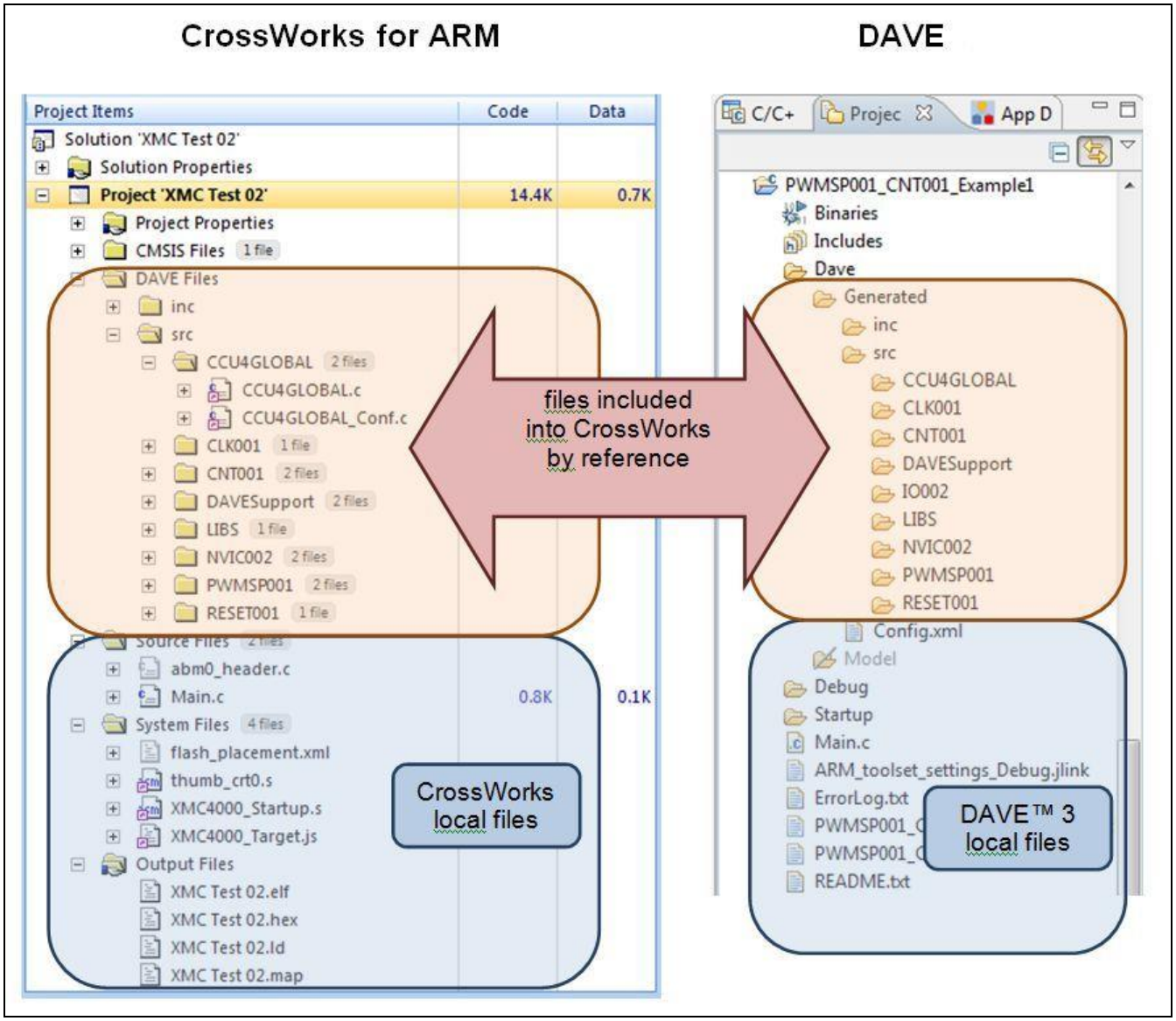

Figure 9 Tools and Project Setup

If the project is setup as described here, the CE perspective of DAVE™ is used to add Apps for additional peripherals, middleware and communication stacks, or to change the configuration settings for the Apps that are already in the project.

The integrated help for the individual Apps provides the API calls to control the Apps from the customer/application code. These API function calls itself can be copied from the help into the CrossWorks™ environment.

The CrossWorks™ tool can be used as normal; application code can be added, compiled and debugged.

The code completion feature available in CrossWorks™ can be used to assist in entering the API function calls for the Apps API.

Note: Keep in mind that the DAVE™ generated files in the Dynamic folder are included by reference and will be overwritten upon the next code generation.

By using both tools in parallel it is now possible to start with the basic 'kernel' of the application, bring it up, and then add modules and functionality by repetitively going through the following steps:

1. add and configure Apps for peripheral drivers or functions such as middle-ware in DAVE™
2. solve and generate the respective C code in DAVE™
3. switch to CrossWorks™ and add/modify the actual application code modules
4. download and debug the code in CrossWorks™

6 XMC4000 Evaluation Kits

The final step is to setup the debugger to the XMC4000 Evaluation Kit. [1]

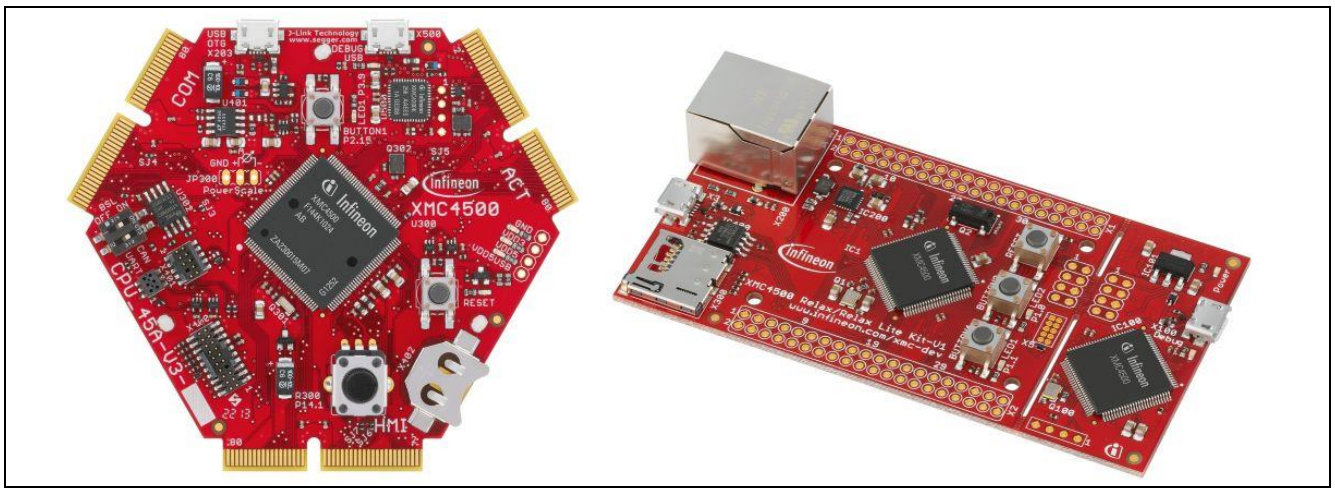

Figure 10 XMC4500 Application Kit and XMC4500 Relax Kit

All XMC4000 Application Kits (except the very first revision of the XMC4500 kit) have the debugger on board.

The debugger interface is a J-Link Lite CortexM and is supported by CrossWorks™ as SEGGER J-Link.

Because there are several versions of the respective driver DLL, we recommend selecting the debugger DLL supplied by Infineon and installed with DAVE™. To do this the respective target property called 'J-Link DLL File' in CrossWorks™ must be set pointing to the DLL.

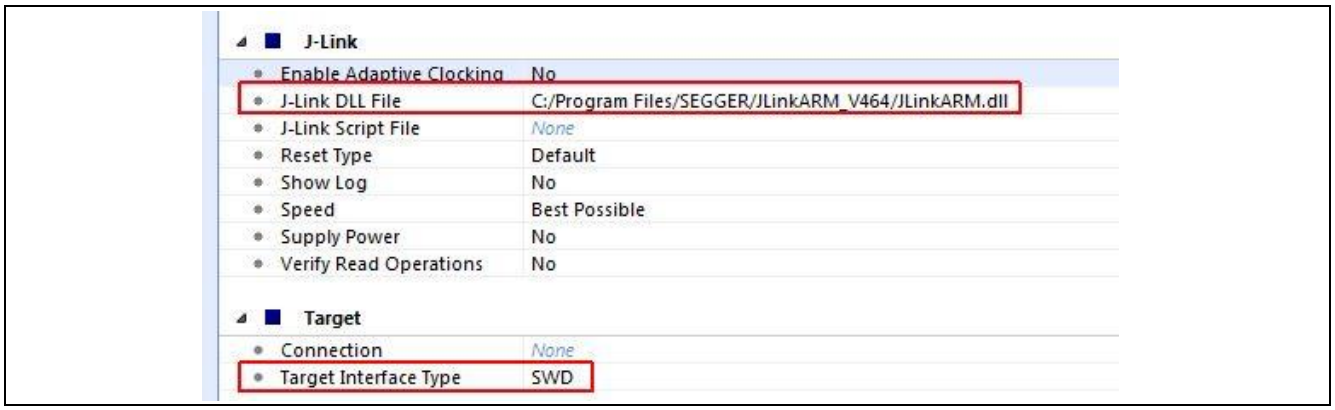

Figure 11 Segger J-Link settings

If the Application Kit is powered up and the debugger connected, it is possible to connect to the target, download the code, and start debugging.

Instead of the J-Link Lite CortexM, any JTAG Interface which is supported by CrossWorks™ can be used. The connector on the CPU Board complies with the standard pinning as defined by ARM®.

The XMC Application Boards are designed to use "Serial Wire Debug" as the debug interface. Please refer to the manual for more information. [5]

7 XMC1000 Evaluation Kits

For the XMC1000 family of ARM® Cortex™-M0 microcontrollers several development boards are available ranging from development boards with ARDUINO™ compatible footprint to boards with a dedicated extension interface for lighting or motor control applications.

The XMC 2Go kit is one of the worlds smallest, fully featured microcontroller evaluation kits.

All boards are supported by CrossWorks™ with a respective board support package.

Figure 12 XMC1000 Boot Kit and XMC2Go Kit

As with the XMC4000 Kits the development boards always have the debugger on board and the debugger interface is a J-Link Lite CortexM and is supported by CrossWorks™ as SEGGER J-Link.

As a starting point either the examples provided with CrossWorks™ can be used, or CrossWorks™ can be set up as described above to use DAVE™ projects.

The example for the XMC2Go that is provided on the Infineon web site does not use DAVE™ Apps and can be used ‘stand-alone’. The header file `GPIO.h` defines the IO pins required as well as inline functions to access the respective registers.

The example implements the `SysTick_Handler` to blink the LEDs and send out messages via the UART of USIC Channel 0. Since the UART is routed via the debugger to a virtual serial port on the PC the terminal window in CrossWorks can be used to show these messages. Initially the port setting is 115.2kbps/8N1.

Examples for the XMC1000 Boot Kits including the application kit boards are provided within DAVE™. All of them can be used within CrossWorks™ if the projects are setup as described above for the XMC4000.

Note: The XMC1000 family supports several different boot modes. It should be noted that the devices used on the kits are programmed to ‘User Mode Debug’ while productive devices are programmed to ‘UART-Bootstrap Loader’. Productive devices therefore have to be reprogrammed by changing the ‘Boot Mode Index - BMI’ before the SWD debugger can be used.

8 Tips and Tricks

8.1 Using Cached and Non-cached Flash (XMC4000)

The XMC4000 series of microcontrollers include instruction and data caches in order to allow code execution and data access with a minimum latency, even at higher core clock rates. As there might be applications which need fully deterministic timing behavior and therefore cannot use the cache, the implementation is such that the actual flash memory is accessible via two distinct memory regions:

- 0x08000000 for cached access
- 0x0C000000 for non-cached access

From revision 1.2 onwards, the board support packages for the XMC4000 implement full support for the cache. In the default build configuration the code and data sections are placed into the 0x08000000 region, resulting in cached accesses during run-time.

A special section `.uncached` is defined which can be used to place dedicated portions of code and/or data into the non-cached region.

Placing code or data into that section is done according to the standard CrossWorks™ methods by either setting the respective section properties on directories or files or by using the GCC section attribute on individual functions or data.

```
void foo_uncached ( void ) __attribute__ ((section (".uncached")))
unsigned int var_uncached  __attribute__ ((section (".uncached")))
```

Within the actual flash module the `.uncached` code and data is placed behind the cached code and data. The `.uncached` section can easily be aligned; e.g. at a 256 Byte boundary by setting the respective alignment value.

```
...  alignment="0x100" ...
```

In that way cached and non-cached code can be easily mixed within one project giving maximum flexibility.

Functions linked into the project from a library still remain within their original section (e.g. `.text`) which is per default placed into the cached flash area.

Note: Some early chip revisions of the XMC4000 family e.g. XMC4500 step A' or XMC4400 step AA, showed an errata where two consecutive instruction fetch accesses, the first to the non-cacheable and the second to the cacheable address space, may cause a corruption of the program flow (PMU_CM.001). This is corrected in present revisions but some kits may still contain the old chip revision. Please refer to the respective errata sheets for further details.

The full project can be placed into the 0x0C memory segment resulting in the application running without cache by changing the respective `Section Placement` property. This will result in using a dedicated section placement file which is part of the board support packages. With this method all code including library functions and the vector table are placed into the non-cached region.

Figure 13 Running from uncached flash

Note: By default the Section Placement property is set to 'Flash' in the Flash build configuration. Accordingly the setting to 'Uncached Flash' must be done in the 'Flash' build configuration. Alternatively it can be set back to 'Inherited' here and then the property can be defined in the configuration 'Common'.

8.2 PSRAM, DSRAM1, DSRAM2 (XMC4000)

The XMC4000 series provides three different SRAM blocks with the following usage recommendation:

- PSRAM => for use as fast program SRAM for code,
- DSRAM1 => data SRAM to be used by the system e.g. for variables,
- DSRAM2 => communication SRAM primarily for use by USB and Ethernet

It would be possible to place code and data differently, but this might cause delays; e.g. if code has to be fetched from the DSRAMs via the system bus.

Starting with revision 1.2 the board support packages for XMC4000 make full use of this concept and place the sections accordingly.

8.3 PSRAM vs. (non) cached Flash (XMC4000)

The XMC4000 Board Support Package includes some examples using the ARM® DSP code for the Cortex™-M4. Some of these examples have been used to compare the execution speed for code running from the different memory regions.

The system timer has been used to measure system ticks required for the main code. The main clock is set to 120MHz and flash wait states are set to 3. The examples use the internal FPU and the values are therefore not comparable to the first version of this Application Note.

The non-cached values are measured with the complete sections `.init`, `.text` and `.rodata` in the memory segment 0x0C by using the 'uncached Flash' setting as described above.

Table 2 Comparison of code execution speed

	non cached Flash	cached Flash	PSRAM
arm_class_marks_example	7.996	4.493	3.117
arm_convolution_example	80.362	39.135	35.736
arm_dotproduct_example	4.873	2.087	2.004
arm_linear_interp_example	26.397	10.962	n.a. see below
arm_fir_example	134.964	80.007	77.724

Executing code using the cached memory segment gives a clear advantage compared to non-cached flash, and almost reaches the execution speed of the program SRAM.

The linear interpolation example uses approximately 740 Kbyte of read only data which does not fit into the SRAM, and must be placed into flash.

8.4 SRAM vs. Flash (XMC1000)

The XMC1000 Board Support Package includes the same examples as used above for the XMC4000. Again some of these examples have been used to compare the execution speed for code running from the different memory regions.

The system timer has been used to measure system ticks required for the main code. The clock is set to 32MHz

Table 3 Comparison of code execution speed

	Flash	SRAM
arm_class_marks_example	37.384	27.051
arm_dotproduct_example	9.794	7.475
arm_fir_example	1.931.179	1.393.614

As the XMC1000 provides 16Kbytes of SRAM it can be used for timing critical section like interrupts, giving an advantage in execution speed compared to the flash.

8.5 Clang vs. GCC

The latest versions of CrossWorks™ not only come with the integrated GCC tool chain but additionally provide the clang Compiler from the LLVM project. [6]

Clang is a 'C only' compiler frontend and has been written from scratch. It therefore claims to "..deliver amazingly fast compiles .. and low memory use ".

The `Default Compiler Variant` to be used can be set in the Environment Settings. It should be noted that clang only provides the frontend compiler and the backend used is still GNU as.

While this is heavily depending on the concrete code of the project clang tends to generate slightly smaller code sizes compared to GCC. Since clang has been developed specifically for C/C++ it also gives better error descriptions and hints.

It should be noted that the examples in DAVE™ have only been tested with GCC.

8.6 Newlib vs. CrossWorks C Library

The C library provided within DAVE™ and used throughout the examples is the open source implementation newlib.

In contrast CrossWorks™ provides its "..own royalty-free ANSI and ISO C compliant C library that has been specifically designed for use within embedded systems".

While in theory both implementations should be fully compatible it should be kept in mind that there might be slight differences in the implementation. In addition CrossWorks™ provides ways to configure the content of the library linked into the individual project in order to further optimize code size (e.g. regarding type support in printf/scanf). Namely when using example projects from DAVE™ it should be assured that the functions used by the DAVE™ project are available and fully compatible.

8.7 Choice of RTOS

For more demanding solutions it might be advisable to use a RTOS. The tools described here offer three choices out of the box:

- CrossWorks™ Tasking Library CTL
- RTX-RTOS integrated in DAVE
- FreeRTOS™ [7]

Both XMC1000 and XMC4000 are officially supported by FreeRTOS™.

For the CTL as well as for FreeRTOS™ there are example solutions available within CrossWorks™. They can be found in the samples directory together with the ARM DSP library examples used above.

(defined as makro `SampleDir`)

In addition both RTOS are supported with a respective debug script to populate the thread view.

For the documentation of the RTX-RTOS please refer to the DAVE App RTOS001.

If DAVE Apps are used for peripheral functions, the RTOS aware version providing reentrant APIs should be used where available, e.g. UART002 instead of UART001.

Since the examples provided for DAVE are based on the RTX-RTOS from Keil some modifications may be required in order to combine the Apps with one of the other two RTOS.

8.8 Debugging Hard Fault & Other Exceptions

The ARM® Cortex™-M4 core implements a set of fault exceptions where each exception relates to an error condition in the hardware.

The standard implementation of the fault handlers do not provide any means to analyze the root cause of the fault, and since there is an escalation scheme implemented between the individual faults typically all of the exceptions end up in the `HardFault_Handler()`.

The FreeRTOS™ web site provides a good introduction and tips on implementing the individual handlers in order to be able to track down the root cause. [8]

8.9 Linker script generation

CrossWorks™ manages placement of code and data at dedicated memory locations via two xml files called `memory map` and `section placement`. The memory map defines the memory map of the device chosen and should normally not be changed.

The `section placement` file can be imported into the project and edited locally giving maximum flexibility also for more ‘exotic’ memory configurations. This might be necessary when integrating pre-compiled code which does use non-standard sections.

A linker script generator than creates a linker script for the GNU linker. As this is a standard GNU linker script it can also be used for the build process within DAVE™.

9 Appendix

9.1 Tool and Kit Information

The following tools and kits with the revisions given have been used.

Table 4 Tools and Revisions

DAVE™ rev. 3.1.10 [3]	
DAVEAppsLibrary001	v1.0.66
DAVEAppsMotorControlLibrary001	v1.0.28
DAVEApps_Lighting_Library_001	v1.0.18
DAVEApps_HMI_Library_001	v1.0.6
CrossWorks™ for ARM, rev. 3.2 [4]	
XMC1000 CPU Support Package	v1.1
XMC1100 Boot Kit BSP	v1.0
XMC1200 Boot Kit BSP	v1.0
XMC1300 Boot Kit BSP	v1.0
XMC4000 CPU Support Package	v1.4
XMC4200 Actuator Application Kit BSP	v1.2
XMC4400 General Purpose Application Kit BSP	v1.2
XMC4500 General Purpose Application Kit BSP	v1.3
XMC4500 Relax Kit BSP	v1.2
XMC4500 SDRAM Application Kit BSP	v1.0

Table 5 Evaluation & Application Kits [5]

XMC4500 Enterprise Kit	
XMC4500 Relax Kit	
XMC1300 Boot Kit	
XMC2Go Kit	

9.2 References

[1] Infineon: XMC4000 Microcontroller Family, http://www.infineon.com/xmc4000

[2] Infineon: XMC1000 Microcontroller Family, http://www.infineon.com/xmc1000

[3] Infineon: DAVE™, http://www.infineon.com/DAVE

[4] Rowley: CrossWorks™ for ARM, http://www.rowley.co.uk

[5] Infineon: XMC Development Kits, http://www.infineon.com/xmc-dev

[6] The LLVM Compiler Infrastructure, http://llvm.org/

[7] FreeRTOS™, http://www.freertos.org/

[8] FreeRTOS™: Debugging Hard Fault & Other Exceptions, http://www.freertos.org/Debugging-Hard-Faults-On-Cortex-M-Microcontrollers.html

9.3 Revision History

Major changes since the last revision

Page or Reference	Description of change
V1.0, 2012-10	
	Initial version
V1.1, 2013-04	
Tools	New tool revisions
V1.2, 2014-10	
Tools	New tool revisions, XMC1000 boards added
XMC1000	Added XMC1000 Evaluation Kits, runtime comparison
Tips & Tricks	Added sections Clang, Clib, RTOS, Hard Faults, Linker Script

www.infineon.com

Edition 2014-10

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

AP32220

Legal Disclaimer

THE INFORMATION GIVEN IN THIS APPLICATION NOTE (INCLUDING BUT NOT LIMITED TO CONTENTS OF REFERENCED WEBSITES) IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office. Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.