Infineon XMC1200 Boot Kit: Cortex-M0 Lab

ARM Keil MDK 5 Toolkit Summer 2014

For the Infineon XMC1200 Boot Kit Version 2.0 Robert Boys bob.boys@arm.com

ARM KEIL Microcontroller Tools

Introduction:

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_263.asp

The purpose of this lab is to introduce you to the Infineon Cortex™-M0 processor family using the ARM® Keil® MDK™ toolkit featuring the IDE µVision®. This tutorial will use the on-board J-Link Lite debug adapter.

Keil provides a free MDK license for use with Infineon XMC1000 processors. See www.keil.com/infineon/mdk to obtain this complimentary license. If you do not install a license, you can still use this lab. MDK will then have a 32K compile limit. MDK includes a full version of Keil RTX™ RTOS. RTX source code is included with MDK.

Middleware: Keil has a suite of commercial grade middleware for Infineon processors. This includes TCP/IP, USB, CAN and a Flash File system. It is a component of MDK-Professional. See www.keil.com/arm/mdk for more information.

ARM Compiler Qualification Kit: For safety certification applications: www.keil.com/pr/article/1262.htm

Why Use Keil MDK ?

MDK provides these features particularly suited for Cortex-M processor users:

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM Compiler/Assembler and Linker toolchain. MDK is a turn-key product with included examples and is easy to get running.
2. A full feature RTOS called RTX is included with MDK with source code. It has a BSD license and was created and is maintained by Keil.
3. An RTX Kernel Awareness window that is updated in real-time.
4. **DAVE compatible.** See www.keil.com/appnotes/files/apnt_258.pdf
5. Choice of adapters: ULINK™2, ULINK-ME, ULINK*pro* or Segger J-Link and J-Link Lite. µVision also supports CMSIS-DAP.
6. All CoreSight™ debugger features are supported in the µVision debugger.
7. Keil Technical Support is included for one year and is easily renewable. This helps you get your project completed faster and more reliably.
8. Keil also supports Infineon 8051 and C166 processors. See www.keil.com/dd for the complete list.

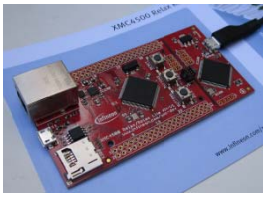
This document details these features:

1. Real-time Read update for Watch, Memory and RTX Threads windows. Memory and SVD also have Write access. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added.
2. Four Hardware Breakpoints (can be set/unset on-the-fly) and four Watchpoints (also called Access Breaks).
3. System and Thread Viewer: a kernel awareness program for RTX RTOS that updates while the program is running.
4. A DSP example with variables updated in real-time as the program is running using a Watch window.
5. Three examples using DAVE to configure and create µVision Blinky, RTX_Blinky and PWM_Blinky projects.

Other Infineon boards supported:

MDK supports, with examples, other evaluation boards from Infineon using ARM processors. These include XMC4500 Relax, XMC4500 Application Kit, XMC1100 2Go Kit and XMC4500 HiLight. See www.keil.com/infineon.

MDK supports Serial Wire Viewer (data trace) and ETM (instruction trace) on those Cortex-M4 processors that include these advanced debugging features. These features are described in their respective labs.



Index:

Part A: Obtaining and Installing MDK and the Examples:

Part B: Stand-alone Project Examples:

Part C: Projects created with Infineon DAVE:

Appendix:

Part A: Obtaining and Installing MDK and the Examples:

1) Keil Software Download and Installation:

1. Download MDK 5.11a or later from the Keil website. www.keil.com/mdk5/install
2. Install MDK into the default directory. You can install into any directory, but this lab uses the default C:\Keil_v5
3. We recommend you use the default examples directories for this tutorial. We will use C:\MDK\ for the examples.
4. If you install MDK or the examples into a different directory, you will have to adjust for the directory differences.
5. The examples Blinky5, RTX_Blinky5, DSP5 and PWM_Blinky are available where you got this document.
6. You do not need any external debug adapters: just the Boot Kit board, a USB cable and MDK installed on your PC.

2) Complimentary MDK License:

Without a license, MDK functions but is limited to a 32K compilation size and a few other limitations. In this mode, MDK is referred to MDK Lite. You can use MDK Lite for this tutorial. Adding a license removes the limitations.

Keil offers a complimentary MDK license for Infineon XMC1100 processors. Go to www2.keil.com/infineon/mdk/ to obtain a license. Installation instructions are also available at this URL.

3) The on-board J-Link Lite Debug Adapter:

The on-board J-Link Lite adapter is used exclusively in this lab. Instructions on configuring the J-Link are given. J-Link USB drivers come with DAVE download and are also included with MDK. Page 5 contains a test for the J-Link Lite drivers as well as instructions on installing the USB drivers. This document uses DAVE 3.1.10.

4) Example Programs:

The example programs are available where you obtained this document at www.keil.com/appnotes/docs/apnt_263.asp. The next page describes where to install these examples.

5) Getting Started MDK 5: Obtain this useful book here: www.keil.com/mdk5/.

More Information and Keil Contacts:

For more information about Keil support for Infineon products please visit www.keil.com/infineon

ARM Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content

Infineon Community Forum: www.infineonforums.com/

Concept: Why Use DAVE ?

In order to successfully configure modern microcontrollers peripherals the study of minute details such as register bits in complicated data sheets is required. DAVE is a graphical configuration program that avoids these manual tasks rather easily. DAVE takes care of the many details that you do need to understand completely. Using DAVE for even mildly complicated development can save a great deal of time and avoid many errors. You can develop and debug your programs either with MDK by itself or by using DAVE to create the initial project with the peripheral configurations that you select.

DAVE Apps are programs that are selected and configured to your specifications. DAVE uses them to create software that configures elements such as peripherals and other components of the Infineon processor. These become C source files that are compiled by DAVE and/or MDK. They become the base to the source files you add that are specific to your project.

Once DAVE creates a project, you can easily import this into µVision by double-clicking on the gpdsc file. No plug-ins for µVision or DAVE are needed. Do not modify any DAVE created files: do this as necessary within DAVE and the new files will be imported into µVision. Create, add and modify your own source files and DAVE will not modify or delete them.

Your projects can use RTX RTOS or not. Using RTX simplifies your development and provides useful benefits.

Documentation and Help:

DAVE tutorials are available on www.infineon.com/dave.

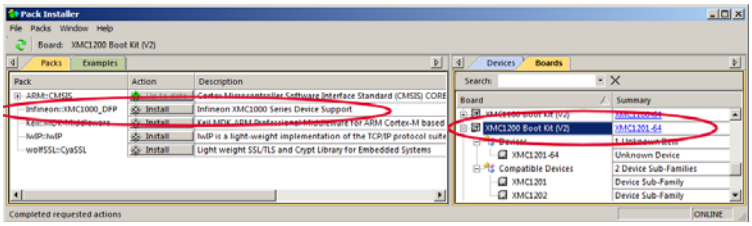
DAVE Help is available by selecting Help/Help Contents from inside DAVE. Note the content list on the left.

Using DAVE with MDK 5: www.keil.com/appnotes/files/apnt_258.pdf

6) µVision Software Packs Download and Install Process:

1) Start µVision and open Pack Installer:

1. Connect your computer to the internet. This is normally needed to download the Software Packs.
2. Start µVision by clicking on its desktop icon.
3. Open the Pack Installer by clicking on its icon: A Pack Installer Welcome screen will open. Read and close it.
4. This window opens up: Select the Boards tab: Select Infineon XMC1200 Boot Kit and then select the Packs tab as shown: The Packs tab is now filtered to list only the relevant Pack to the Boot Kit.



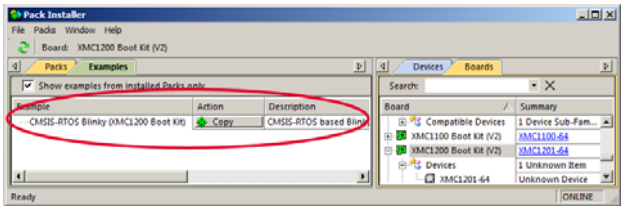
5. Note "ONLINE" is displayed at the bottom right. If "OFFLINE" is displayed, connect to the Internet before continuing.
6. If there are no entries shown because you were not connected to the Internet when Pack Installer opened, select Packs/Check for Updates or to refresh once you have connected to the Internet.

2) Install The XMC1000 Software Pack: The Packs tab should be selected by default.

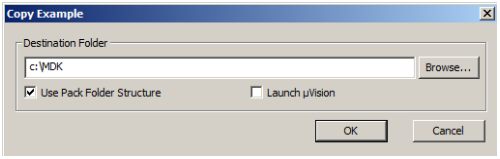
1. Initially, the Software Pack ARM::CMSIS is installed by default. These contain system headers and source files.
2. Select Infineon::XMC1000_DFP and click on Install. This Software Pack will download and install to C:\Keil_v5\ARM\Pack\Infineon\XMC1000\ by default. This download can take two to four minutes.
3. A Software Pack status is indicated by the "Up to date" icon: Up to date

TIP: If you click on the Devices tab, you can select the processor you are using and only the relevant Packs are displayed.

3) Install the CMSIS-RTOS Blinky MDK 5 Example:



1. Select the Examples tab.
2. Select CMSIS-RTOS Blinky as shown:
3. Select Copy Copy as shown here:
4. The next window opens up: Select Use Pack Folder Structure and unselect Launch µVision:



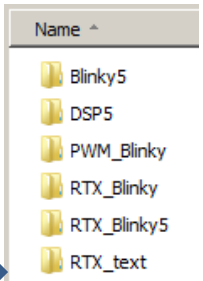
5. Type in C:\MDK. Click OK to copy the RTX_Blinky project.
6. This will copy to C:\MDK\Boards\Infineon\XMC1200 Boot Kit\. Pack Installer creates the rest of the directory tree after C:\MDK\.

TIP: The default directory for copied examples the first time you install MDK is C:\Users\<user>\Documents. For simplicity, we will use the default directory of C:\MDK\ in this tutorial. You can use any directory you prefer.

7. Close the Pack Installer. You can open it any time by clicking on its icon.
8. Select **Yes** when Reload Packs? box is displayed. This updates the µVision Pack database.

4) Install the Blinky5, RTX_Blinky5 and DSP5 examples from Keil.com:

1. Obtain the example software zip file from www.keil.com/appnotes/docs/apnt_263.asp.
2. Unzip this into the directory C:\MDK\Boards\Infineon\XMC1200 Boot Kit\.
3. Blinky5, DSP5, PWM_Blinky, RTX_text and RTX_Blinky5 folders will be created:

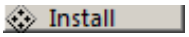


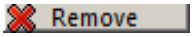
TIP: An Update icon means there is an updated Software Pack available for download. Update

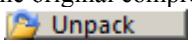
TIP: If you look in C:\Keil_v5\ARM\Pack\Infineon\XMC1000_DFP\1.0.2\Boards\Infineon\XMC1200 Boot Kit, you will find another RTX_ Blinky. This is the read-only version that you downloaded for backup purposes. Use only the projects you copied over from the Examples tab to the directory you chose: in this tutorial we have used C:\MDK.

The next page has a few notes on Software Packs Maintenance plus how to test the J-Link debug connection:

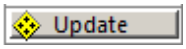
Software Packs Maintenance Notes:

1. Software Packs are downloaded and installed on your computer with the Install icon. 

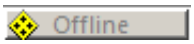
2. You can delete a Pack by double-clicking on it. A Remove icon appears. Click on this. 

3. The Pack is now removed but the original compressed file you downloaded is still present: You can reinstall the Pack by clicking on this icon: 

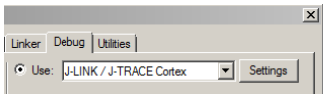
4. To completely remove it: right-click on the Pack name or Unpack icon and select Delete Pack. 

5. If an updated Pack is available, the Update icon appears. You can download this update (or not). 

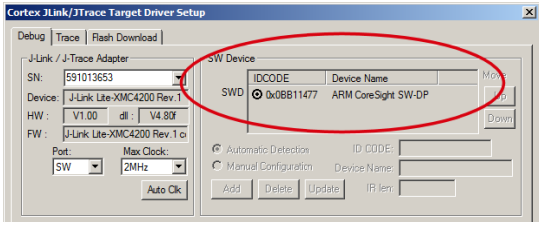
6. Example files, if provided by a Pack, can be copied into a directory of your choice. 

7. Pack file is not available on-line. It might be locally downloaded on your computer. 

7) Testing the J-Link Lite Connection:

1. Start µVision  if it is not already running. Connect the Boot Kit board to your PC with a USB cable.
2. Select Project/Open Project.
3. Select the Blinky project C:\MDK\Boards\Infineon\XMC1200 Boot Kit\Blinky5\Blinky.uvprojx.
4. Select Target Options  or ALT-F7 and select the Debug tab: 

5. Click on Settings: and the window below opens up: Select SW in the Port: box. If an ICODE and Device name is displayed, J-Link Lite is working. You can continue with this tutorial. Click on OK twice to return to the µVision main menu.
6. A number in the SN: box means µVision is successfully connected to the J-Link Lite adapter.
7. If nothing or an error is displayed in this SW Device box, this ***must*** be corrected before you can continue.
8. Make sure the DIP switches are all ON and the jumper is set to 3.3V. Confirm Port: is set to SW and not JTAG. If these are set correctly, see the next step: 8) Installing the J-Link USB Drivers:
9. If you see a proper display as shown, your J-Link USB drivers are installed properly and µVision is connected to the CoreSight debug module in the Infineon processor.



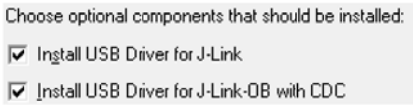
10. Click OK twice to close these windows. Continue to the next page since your J-Link drivers are correctly operating.

TIP: To refresh the SW Device box, in the Port: box select JTAG and then select SW again. You can also exit then re-enter this window. The XMC1200 will not work with JTAG selected, only SW. But, this is a useful way to refresh the SW setting.

8) Installing the J-Link Lite USB Drivers: (this is needed only if the above test fails)

The DAVE 3.1.10 download contains the correct J-Link USB driver software for the Boot Kit board. You must manually install the J-Link separately from DAVE. This is easy to do. This file is Setup_JLinkARM_V484f.exe for DAVE 3.1.10.

1. Unplug the Boot Kit board. Install the software in the usual fashion using the suggested defaults.
2. To use the UART feature make sure the J-Link-OB with CDC is selected:
3. Plug in the Boot Kit board and it will enumerate normally.
4. Run the test above again to confirm everything is operating properly.
5. If the installer asks you to replace the drivers in MDK or µVision, select or allow this.



TIP: MDK 5 also contains J-Link drivers here: C:\Keil_v5\ARM\Segger\USBDriver\CDC for the Boot Kit board. They are also available here: Go to www2.keil.com/infineon/mdk/ and select: 

We recommend you use the J-Link drivers that came with your version of DAVE. These should be V 4.84f for 3.1.10.

Part B: Stand-alone Project Examples:

9) Blinky Example Program using the XMC1200 Boot Kit board:

We will connect the Keil MDK development system using the XMC evaluation board. This project is pre-configured to use the on-board J-Link Lite. It is easy to configure µVision to use any other supported adapter.

1. Connect your Boot Kit board to your computer with a USB cable.
2. Start µVision by clicking on its desktop icon.
3. Select Project/Open Project. Open the MDK Blinky5 project file that you copied on page 4: C:\MDK\Boards\Infineon\XMC1200 Boot Kit\Blinky5\Blinky.uvprojx.
4. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
5. Program the XMC Flash by clicking on the Load icon: Progress will be indicated in the Output Window.
6. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
   Note: µVision is configured to program the Flash when entering Debug mode. You do not have to use Load.
7. Click on the RUN icon. Note: you stop the program with the STOP icon.

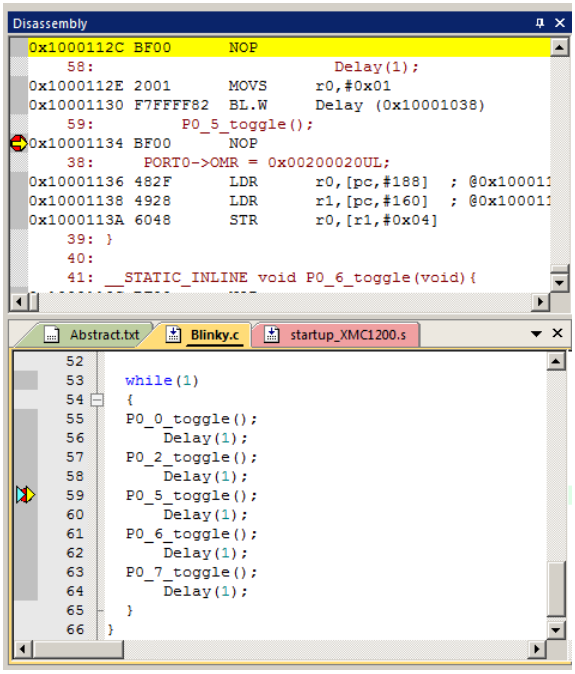
The five LEDs on the Boot Kit board will now blink in sequence.

Now you know how to compile a program, load it into the XMC processor Flash, run it and stop it.

Note: The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed.

10) Hardware Breakpoints:

1. With Blinky running, click in the left margin on a darker gray block somewhere appropriate in the while(1) loop in Blinky.c in the main() function as shown below:
2. A red circle is created and soon the program will stop at this point.
3. The yellow arrow is where the program counter is pointing to in both the disassembly and source windows.
4. The cyan arrow is a mouse selected pointer and is associated with the yellow band in the disassembly window. Click on a line in one window and this place will be indicated in the other window.
5. Each time you click on RUN, the program will execute until it reaches the next breakpoint.
6. Note you can set and unset hardware breakpoints while the program is running with ARM CoreSight technology.
7. The XMC1200 has 4 hardware breakpoints. A breakpoint does not execute the instruction it is set to. This is a very important feature for effective debugging. The XMC4000 series Cortex-M4 has 6 hardware breakpoints.
8. If you set too many breakpoints, µVision will notify you. Sometimes µVision will use one of the available breakpoints for one of its internal operation such as single stepping.
9. **Remove the hardware breakpoint(s).**



TIP: If you get multiple cyan arrows or have trouble understanding the relationship between the C source and assembly, try lowering the compiler optimization to Level 0 and rebuilding your project.

This level is set in Options for Target under the C/C++ tab.

TIP: Enable Use MicroLIB under the Target tab to create smaller programs. If your program is allowed to use MicroLIB, the compiler will not generate any errors.

11) Call Stack + Locals Window:

Local Variables:

The Call Stack and Local windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables belonging to the active function. If possible, the values of the local variables will be displayed and if not, the message <not in scope> will be displayed.

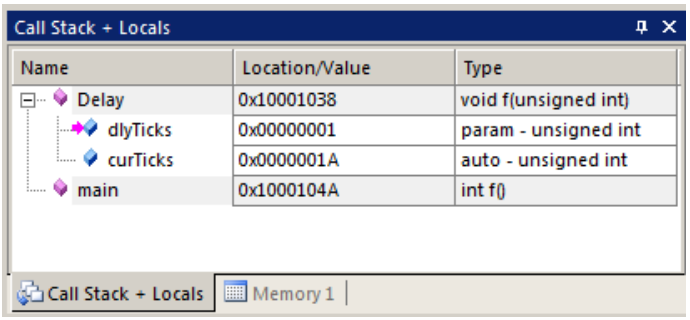
1. Stop the program and open the Call Stack and Locals window by clicking on its tab.
2. Shown below is the Locals window for the main function when the program counter is in the Delay function. This program spends most of its time in Delay() so it will probably nearly always stop there.
3. Click on Step Out (Ctrl-F11)  . Only the main() function will be displayed since the program has exited Delay().

TIP: The contents of the local variables are displayed as well as names of active functions. Each function name will be displayed as it is called from the function before it or from an interrupt or exception. Exactly which local variable that will be visible or not depends on precisely where you stop the program.

When a function exits, it is removed from the list.

The last called function is at the top of this table.

This table is active only when the program is stopped.



4. Click inside the body of the Disassembly window.
5. Click on the Step In icon or F11 until you re-enter Delay(): 
6. Variables will update as appropriate.
7. Click on the StepOut icon  Ctrl-F11 to exit the Delay function to return to main().
8. **Remove any breakpoints when you are done.** You can click on them individually or Ctrl-B and select Kill All.

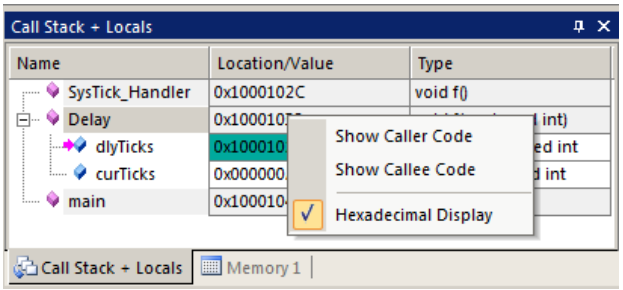
TIP: You can modify a variable value in the Call Stack & Locals window when the program is stopped.

TIP: If the Disassembly window is in focus, using Step  will cause steps at the assembly code level. If a source file is in focus, steps will be at the C or C++ level.

Call Stack:

The list of called functions is displayed when the program is stopped. This is very useful for debugging when you need to know which functions have been called and are stored on the stack.

9. Set a breakpoint in the SysTick Handler function near line 27 in Blinky.c. Click on RUN if necessary.
10. The program will very soon stop executing.
11. The window below will display. Note the three functions displayed:
12. Right click on a function name and try the Show Callee Code and Show Caller Code options as shown here: The appropriate code will be shown in the source and/or disassembly windows.
13. Click on Step and you will soon exit all functions except Main() and will soon be back in Delay.
14. Remove any breakpoints you set.



TIP: Use the Symbol window to locate and view components of your program including variables, structures and arrays. Select View/Symbol Window while in Debug mode.

TIP: For help finding Fault Exceptions see: www.keil.com/appnotes/docs/apnt_209.asp

12) Watch and Memory Windows and how to use them:

The Watch and Memory windows will display updated variable values in real-time. It does this using the ARM CoreSight debugging technology that is a component of Cortex-M processors. It is also possible to "put" or insert values into the Memory window in real-time. It is possible to "drag and drop" variable names into windows or enter them manually. You can also right click on a variable and select Add *varname* to.. and select the appropriate window.

Watch window:

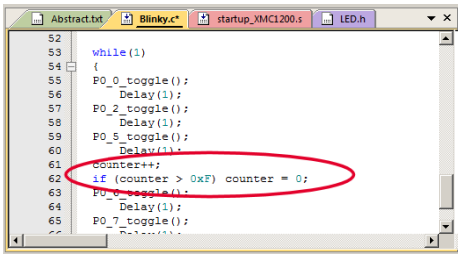
Add a global variable: Call Stack, Watch and Memory windows can't see local variables unless stopped in their function.

1. Stop the processor and exit Debug mode.
2. Declare a global variable is the usual manner (I called it **counter**) near line 20 in Blinky.c:

   unsigned int counter = 0;
3. Add the statements near line 61 just after Delay(1);

   counter++;

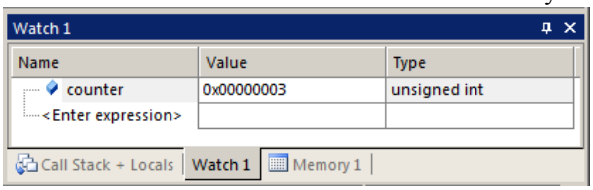
   if (counter > 0xF) counter = 0;



4. Select File/Save All or .
5. Click on Rebuild and program the Flash with Load .
6. Enter Debug mode. Click on RUN .

TIP: You can configure a Watch window while the program is running. You can also do this with a Memory window.

7. Select View and select Periodic Window Update if necessary: Periodic Window Update
8. In Blinky.c, right click on **counter** and select Add counter to … and select Watch 1. Watch 1 will automatically open. **counter** will be displayed as shown here:
9. **counter** will update in real time.



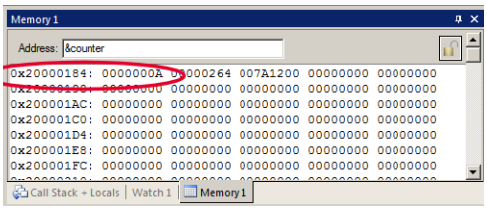
TIP: You can also block a variable name drag and drop it into Watch or Memory windows. You can also enter a variable manually by double-clicking <Enter expression> or press F2 and use copy and paste or typing the variable name.

TIP: To view variables and their location use the Symbol window. Select View/Symbol Window while in Debug mode. If you copy variables from here they will be entered fully qualified.

TIP: To Drag 'n Drop into a tab that is not active, pick up the variable and hold it over the tab you want to open; when it opens, move your mouse into the appropriate window and release the variable.

Memory window:

1. Right click on **counter** and select Add counter to … and select the Memory 1 window.
2. Note the value of **counter** is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to: but this not what we want to see at this time.
3. Add an ampersand "&" in front of the variable name and press Enter. The physical address is (0x2000_0184).
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **counter** is displayed as shown here:
6. Both the Watch and Memory windows are updated in real-time.
7. Right-click with the mouse cursor over the desired data field and select Modify Memory. You can change a memory location or variable on-the-fly while the program is still running.

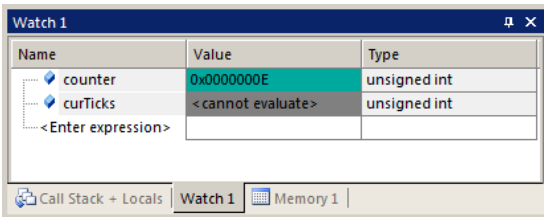


TIP: No CPU cycles are used to perform these operations. You can have more than one variable displayed. Structures can also be displayed and expanded.

These Read and Write accesses are handled by the Serial Wire Debug (SWD) connection via the CoreSight Debug Access Port (DAP), which provides on-the-fly memory accesses. The next page describes how this works.

13) How to view Local Variables in the Watch or Memory windows:

1. With the program still running:
2. There is a local variable **`curTicks`** declared near line 32 in Blinky.c in the Delay() function: **uint32_t curTicks;**
3. Enter **`curTicks`** into Watch 1 window by right clicking on it and selecting Add **`curTicks`** to.... Watch 1.
4. Note it says <cannot evaluate> or "not in scope".
5. Stop the program and the value of local will display. This is because this program spends most of its time in the Delay function and therefore is very likely to stop in it.
6. Click on Step Out Ctrl-F11 a few times to exit Delay.



7. Stop the program if it is running . will now read "Not in Scope"
8. The only time a local variable value will be displayed is if the program happens to stop while it is in scope.
9. µVision is unable to determine the value of **`curTicks`** when the program is running because it exists only when the function Delay is running. It disappears in functions and handlers outside of Delay.
10. Stop the program if it is running . Exit Debug mode.

How to view local variables updated in real-time:

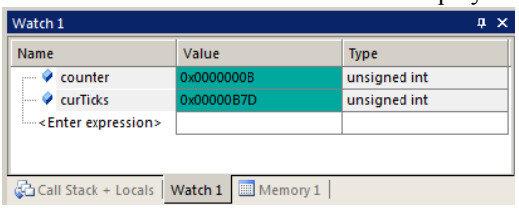
All you need to do is to make **`curTicks`** static where it is declared in Blinky.c !

1. In the declaration for **`curTicks`**, add the **`static`** keyword like this:

```
32      static uint32_t curTicks;
```

TIP: You can also make a variable global or have it as part of a structure so it will update in real-time.

2. Compile the source files by clicking on the Rebuild icon . Select File/Save All or .
3. To program the Flash, click on the Load icon. . Enter Debug mode. Click on RUN.
4. Note **`curTicks`** is still not updated in real-time. You must first show it to µVision by stopping when it is in scope.
5. Stop the program. . The program will probably stop in Delay() and a value for **`curTicks`** will now be displayed.
6. Click on RUN and **`curTicks`** will now update in real-time.



TIP: You must fully qualify a variable in order for it to update without initially stopping the program while it is in scope. To do this, you can open the View/Symbols window and copy the variable from there. This automatically fully qualifies the variable. In this case, **`curTicks`** fully qualified is \\Blinky\Blinky.c\Delay\curTicks. You also can enter this text line directly into the Watch or Memory windows from the Symbol window.

7. You can also enter a variable into a Memory window. Remember to prefix it with an &.
8. Stop the CPU for the next step. Select File/Save All or .

TIP: View/Periodic Window Update must be selected. Otherwise variables update only when the program is stopped.

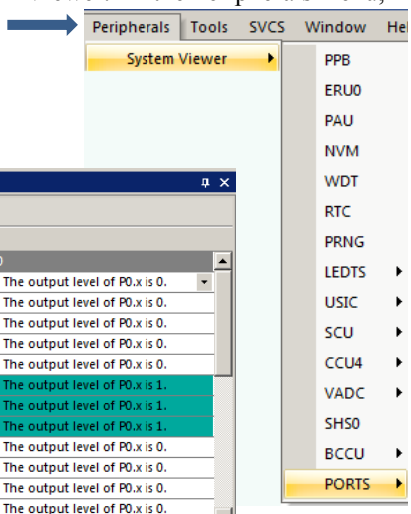
TIP: To program the Flash automatically when you enter Debug mode select Target Options , select the Utilities tab and select the "Update Target before Debugging" box. This is set by default in the two Blinky projects. This means you can skip using the LOAD icon. Programming the Flash will be automatically done when you enter Debug mode. .

How It Works:

µVision uses ARM CoreSight technology to read or write memory locations without stealing any CPU cycles. This is nearly always non-intrusive. While the CPU is fetching instructions at full speed, the CoreSight debug module can read or write values without stealing any CPU cycles. This can be slightly intrusive in the unlikely event the CPU and µVision reads or writes to the same memory location at exactly the same time. Then, the CPU will be stalled to allow this access to occur.

14) System Viewer (SV):

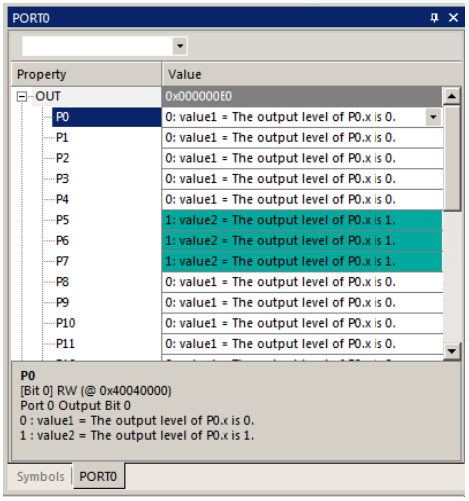
The System Viewer provides the ability to view and modify registers in the CPU core and in peripherals. In most cases, these views are updated in real-time while your program is running. These Views are available only while in Debug mode. There are two ways to access these Views: **a)** View/System Viewer and **b)** Peripherals/System Viewer. In the Peripherals menu, the Core Peripherals are also available: Note the various peripherals available:



1. Click on RUN. You can open SV windows when your program is running.

GPIO Port 1:

2. Select Peripherals/System Viewer and then PORTS and select PORT0.
3. This window opens up. Expand OUT:
4. You can now see the pins update:



TIP: If you click on a register in the properties column, a description about this register will appear at the bottom of the window.

SysTick Timer: This program uses the Cortex-M0 SysTick timer to determine the period of the LEDs.

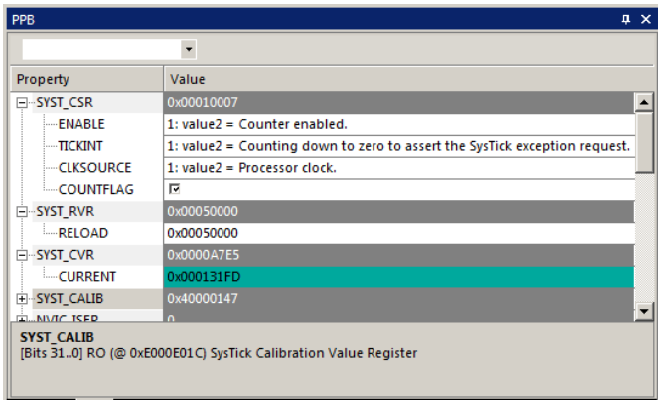
Select Peripherals/System Viewer and then select PPB.

1. The PPB window shown below opens:
2. Note this window updates in real-time while your program runs. These windows use the same CoreSight DAP technology as the Watch and Memory windows.
3. Expand the SYST_RVR register. This is the reload register value. This is set during the SysTick configuration by the SysTick_Config(SystemCoreClock /10); line in main().
4. Note that it is set to 0xC34FF (dec 799,999). This results from the hex value of (8,000,000/10)-1 that is programmed into SysTick_Config() in main() in Blinky.c. Changing the variable passed to this function is how you can change how often the SysTick timer creates its interrupt 15.
5. In the RELOAD register in the SysTick window, *while the program is running,* type in 0x50000 and press Enter !
6. The blinking LEDs will speed up. This will convince you of the power of ARM CoreSight debugging.
7. Replace RELOAD with 0xC34FF. A CPU RESET 

will also do this. The program will restart after RESET.
8. You can look at other Peripherals contained in the System View windows.
9. When you are done, stop the program  and close all the System Viewer windows that are open.

TIP: It is true: you can modify values in the SV while the program is running. This is very useful for making slight timing value changes instead of the usual modify, compile, program, run cycle.

You must make sure a given peripheral register allows and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.



15) Access Breakpoints: *Conditional Breakpoints*

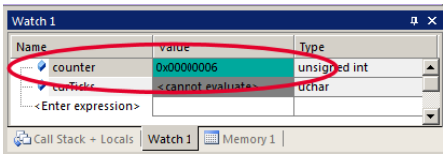
The XMC1100 Cortex-M0 processor has two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are referred to as Access Breaks in Keil documents. Cortex-M0 Watchpoints are slightly intrusive. When the Watchpoint is hit, µVision must test the memory location. Cortex-M3/M4 Watchpoints are not intrusive for the equality test.

1. Use the same Blinky configuration as the previous page. Stop the program if necessary. Stay in debug mode.
2. We will use the global variable **`counter`** you created in Blinky.c to explore Watchpoints.
3. Select Debug in the main µVision window and then select Breakpoints or press Ctrl-B.
4. Select Access to Read. (or write or both if you prefer)
5. In the Expression box enter: "**`counter > 0x5`**" without the quotes. This window will display:

TIP: An Access Breakpoint that does not use a data value (i.e. counter) is not intrusive. It is the test that is intrusive.

You can use <, > and ==. Currently, in MDK 5.10, counter == 0x5 works with any Keil ULINK. It does not work with the J-Link Lite. This feature will be added in a future version of MDK.

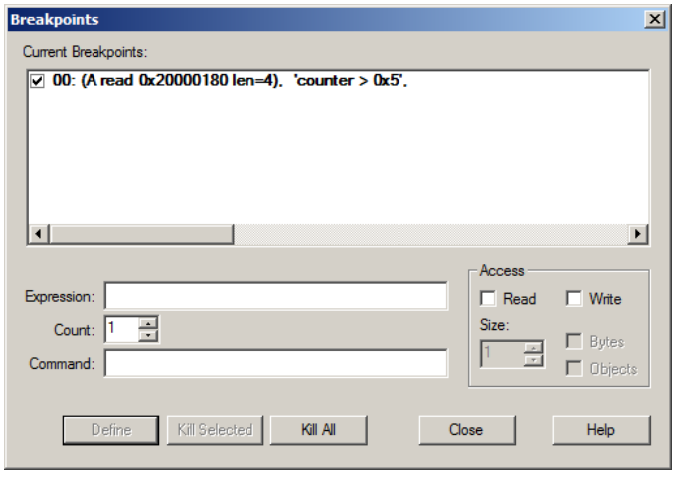
6. Click on Define or press Enter and the expression will be moved into the Current Breakpoints box as shown below:
7. Click on Close.
8. Enter the variable **`counter`** in Watch 1 if it is not already there.
9. Set counter to zero in the Watch window. This puts counter to less than 0x5 which is the test value.
10. Click on RUN.
11. When the program detects a read access of > 0x5 to **`counter`** as you selected, the program will stop. See Watch 1 shown below: In this case the value is 6.
12. Click on RUN twice and the program will run to the next read or write to counter. It will skip values 0 through 5.
13. **`counter`** might not update in the Watch window depending on how fast the variable is changed. This feature is turned off in µVision for speed considerations during the test. You will also notice the program slows down. This is because µVision must test the condition by stopping the program when the write or read occurs to `counter`. Minimize this by selecting only Read or Write Access and not both.
14. Stop the CPU if it is running.
15. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All and then select Close.
16. Exit Debug mode.



TIP: You cannot configure Access Breakpoints on-the-fly while the program is running like you can with hardware breakpoints.

TIP: To edit an Access Breakpoint: double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Make your modifications now. Clicking on Define will create another Watchpoint. You should delete the old one by highlighting it and click on Kill Selected or try the next **TIP:**

TIP: The checkbox beside the expression allows you to temporarily unselect or disable an Access Breakpoint without deleting it.



TIP: Raw addresses can be used with a Access Breakpoint. An example is: *((unsigned long *)0x20000004)

16) *RTX_Blinky* Example Program with Keil RTX RTOS:

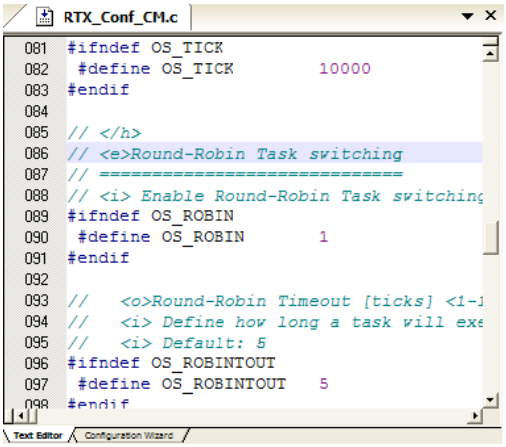
Keil provides RTX, a full feature RTOS. RTX is included as part of the Keil MDK tool suite. It can have up to 255 tasks and no royalty payments are required. RTX has a BSD type license. RTX with source code is provided with all MDK versions. See www.arm.com/cmsis and C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.1.0\CMSIS_RTX. This example explores RTX. Keil will work with any RTOS. A real-time awareness viewer for RTX is provided with µVision.

1. Start µVision by clicking on its icon if not already running. You must have installed the RTX_Blinky5 example previously. Instructions are provided on the bottom of page 4.
2. Exit Debug mode if necessary.
3. Select Project/Open Project and open C:\MDK\Boards\Infineon\XMC1200 Boot Kit\RTX_Blinky5\Blinky.uvprojx.
4. Compile the source files by clicking on the Rebuild icon. They will compile with no errors or warnings.
5. To program the Flash manually, click on the Load icon. A progress bar will be at the bottom left.
6. Enter Debug mode by clicking on the debug icon and click on the RUN icon.
7. The LEDs will blink according to the four threads in this program.
8. Click on STOP.

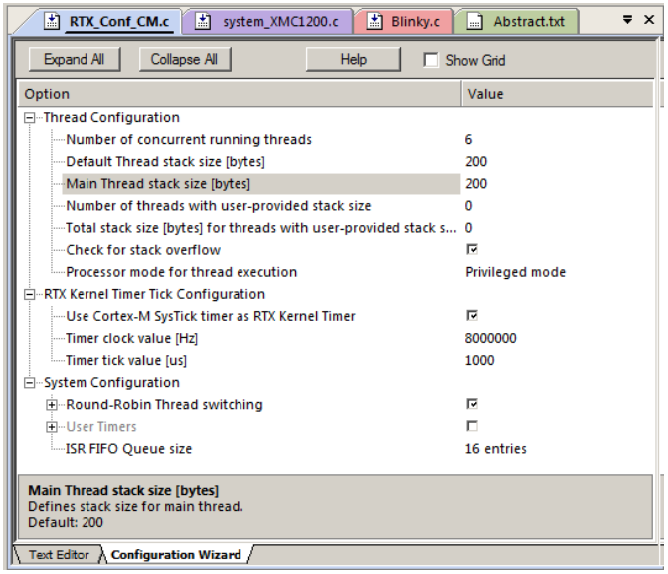
The Configuration Wizard for RTX:

1. In the Project window, double click on RTX_Conf_CM.c to open it in a Source window or with File/Open.
2. Click on the RTX_Conf_CM.c source file tab as shown below on the left below if it is not in focus.
3. Click on Configuration Wizard at the bottom and your view will change to the Configuration Wizard.
4. Open up the individual directories to show the various configuration items available.
5. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
6. This is a great feature as it is much easier changing items here than in the source code.
7. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
8. This scripting language is shown below in the Text Editor as comments starting such as a </h> or <i>. See www.keil.com/support/docs/2735.htm for instructions to add this feature to your own source code.

Getting Started MDK 5: Obtain this book here: www.keil.com/mdk5/. It has very useful information on implementing and managing RTX.



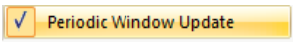
Text Editor: Source Code



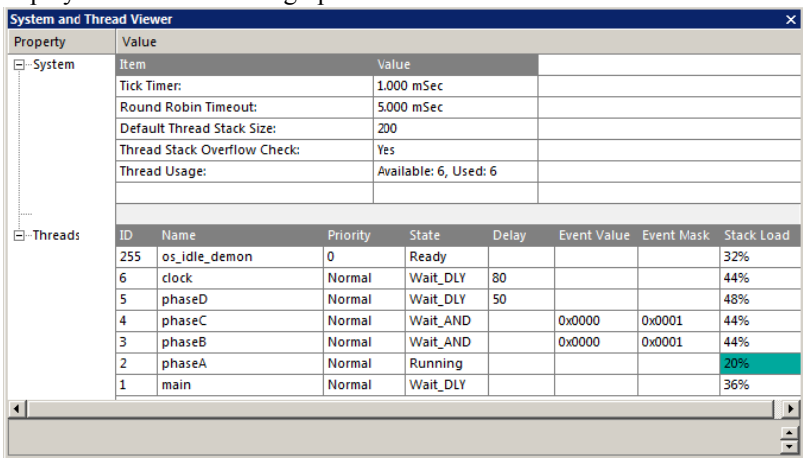
Configuration Wizard

17) RTX Kernel Awareness:

Users often want to know the number of the current operating task and the status of it and the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides a Task Aware window for RTX. Other RTOS companies also provide awareness plug-ins for µVision.

1. Run RTX_Blinky by clicking on the Run icon.
2. Open Debug/OS Support and select System and Thread Viewer. The window below opens up. You might have to grab the window and move it into the center of the screen. These values are updated in real-time using the same read write technology as used in the Watch and Memory windows.
3. Select View and select Periodic Window Update if these values do not change: 

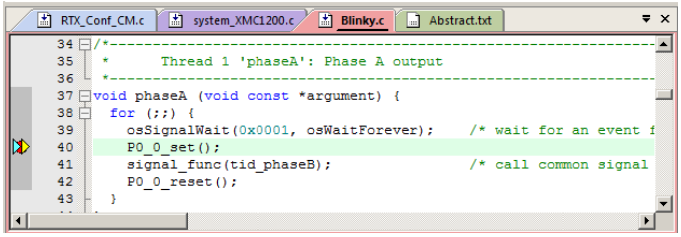
1. You will not have to stop the program to view this data. No CPU cycles are used. Your program runs at full speed. No instrumentation code needs to be inserted into your source. In this example, most of the time the CPU is executing the os_idle_demon. The processor spends relatively little time in each task. You can change this.
2. µVision also has an Event Viewer which displays RTX threads in a graphical format. The XMC1000 Cortex-M0 series does not have the Serial Wire Viewer (SWV) component of CoreSight that is needed for this window. The XMC4000 Cortex-M4 does have SWV and the Event Viewer can be displayed as well as other Serial Wire Viewer features.



Demonstrating States: (note: Tasks and Threads are used interchangeably in Keil MDK documentation**)**

Blinky.c contains four threads that blink the LEDs. Thread 1 (phaseA) is shown below:

1. The gray areas opposite the line numbers indicate there is valid assembly code located here.
2. Set a breakpoint on one of these in Thread 1 as shown: (but not on the void phase line)
3. Set a breakpoint in one or two other threads.
4. Click on RUN .
5. When the program stops, this information will be updated in the RTX Tasks window. The Task that is running when the program stopped will be indicated with a "Running" state. The window above shows the program stopped and phaseA is in the run state. The states of the other tasks are displayed as well.



6. Click on RUN . The other thread will show as "Running". Each time you click RUN, the next thread will run.
7. Look in the Call Stack and Locals window to display the stack.
8. Remove the breakpoints and close the RTX Tasks window. Exit Debug mode:

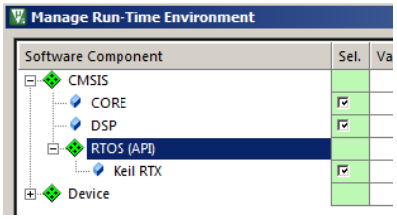
More Information of obtaining and using RTX:

It is very beneficial to use an RTOS. RTX is a good choice. It is small, efficient and easy to use yet it is full featured. RTX source, various ports and documentation are here: C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.1.0\CMSIS_RTX. index.html is the entry point into the documentation.
There are two versions of RTX: The first comes with MDK 4.7x and earlier. The second comes with MDK 5.10 and later. This second one is CMSIS-RTOS compliant and this is the one you want to use.

18) DSP SINE example using ARM CMSIS-DSP Libraries:

ARM CMSIS-DSP libraries are offered for Cortex-M0, Cortex-M3 and Cortex-M4 processors. DSP libraries are provided in MDK in C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.1.0\CMSIS\DSP_Lib. Documentation can be accessed here: Select the DSP tab: C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.1.0\CMSIS \index.html

The appropriate DSP library is selected in the RTE window:

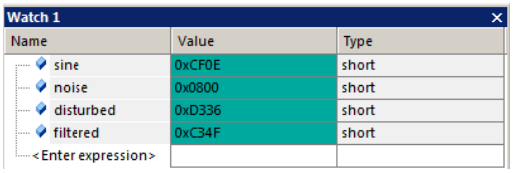


This example creates a sine wave to which another sine wave (to represent noise) is added, and then the noise is filtered out leaving the original sine wave. Four threads contain each task.

This example incorporates Keil RTX RTOS. RTX is available free with a BSD type license. Source code is provided.

To obtain this example file, go to **www.keil.com/appnotes/docs/apnt_263.asp** and copy the MDK 5 version into C:\MDK\Boards\Infineon\XMC1200 Boot Kit\. A \DSP5 directory will be created. You might have already done this.

1. Stop the program and exit Debug mode if necessary.
2. Select Project/Open Project and open: C:\MDK\Boards\Infineon\XMC1200 Boot Kit\DSP5\sine.uvprojx.
3. Build the files. There will be no errors or warnings.
4. Program the XMC1200 Flash by clicking on the Load icon: Progress will be indicated in the Output Window.
5. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
6. Click on the RUN icon.
7. Open Watch 1 by selecting View/Watch/Watch 1 if necessary.
8. Four global variables will be displayed in Watch 1 as shown here: If these variables are changing the program is most likely working properly.



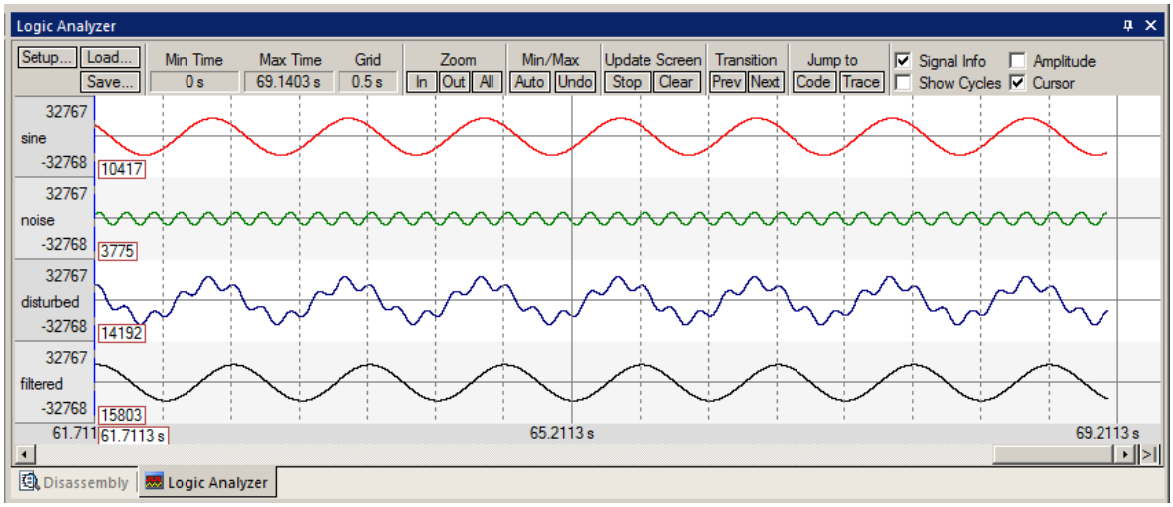
9. Each of these variables represent the value of one of four waveforms created by the DSP program.

Serial Wire Viewer (SWV) and ETM Instruction Trace: (only for reference)

The Infineon XMC4000 Cortex-M4 processors have Serial Wire Viewer (SWV). The four waveforms of the global variables mentioned above will be displayed in the Logic Analyzer as shown below. The Infineon XMC1000 series does not have SWV so this screen is shown for reference only to give you an idea on what is happening in this example.

If you use an Infineon XMC4000 Cortex-M4 and with any Keil ULINK or a J-Link, you can use this Logic Analyzer windows plus many other Serial Wire Viewer (SWV) features. Infineon Cortex-M4 processors also have ETM trace.

See www.keil.com/appnotes/docs/apnt_231.asp. This document shows how SWV and ETM trace works on the XMC4000.



RTX Tasks and System:

3. Click on the RUN icon if the program is not running. 
4. Open Debug/OS Support and select System and Thread Viewer. A window similar to below opens up.
5. This window updates in real-time. Note nearly all the processor time is spent in the idle daemon os_idle_demon. The processor spends relatively little time in each thread. You can change this if you need to.
6. Set a breakpoint in one of the threads in DirtyFilter.c by clicking in the left margin on a grey area.

   Here are the four threads with their approximate starting line numbers:

   1) sine_gen (88) **2)** noise_gen (106) **3)** disturb_gen (126) **4)** filter_tsk (146) **5)** sync_tsk (166)
7. The program will stop here and the Task window will be updated accordingly. Here, I set a breakpoint in the noise_gen thread: You can see that noise_gen was running when the breakpoint was activated.
8. The os_idle_demon is Ready to run when noise_gen is finished. No other task is Ready.

System and Thread Viewer

Property	Value
System	

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 6, Used: 6

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Ready				32%
6	sync_tsk	Normal	Wait_AND		0x0000	0x0001	44%
5	filter_tsk	Normal	Wait_AND	65384	0x0000	0x0001	44%
4	disturb_gen	Normal	Wait_AND	65284	0x0000	0x0001	44%
3	noise_gen	Normal	Running	65532	0x0000	0x0001	8%
2	sine_gen	Normal	Wait_DLY	50	0x0000	0x0001	48%
1	main	Normal	Wait_DLY				36%

TIP: os_idle_demon has a Priority of 0 which is the lowest priority possible. Every other task has a higher priority.

9. Set another breakpoint in a different task. Click on the RUN icon. 
10. Each time you click on RUN, the program will stop at one of the two tasks and this is indicated by the Running state.

TIP: Remember you have only 4 hardware breakpoints and sometimes µVision might commandeer one for its use, usually for run to main() and single-stepping. If you select too many breakpoints, µVision will notify you.

TIP: Recall this window uses the CoreSight DAP read and write technology to update this window in real-time.

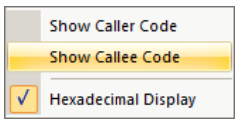
TIP: You might have noticed the Event Viewer in Debug/OS Support. This uses Serial Wire Viewer to get its information and this feature is available on the Infineon XMC4000 Cortex-M4 processors with any Keil ULINK or Segger J-Link debug adapter.

Call Stack and Locals:

1. Click on the Call Stack + Locals tab. This window opens up:

2. Each time you click on RUN and the program stops on one of the breakpoints you set, this information is updated depending on which thread is running.

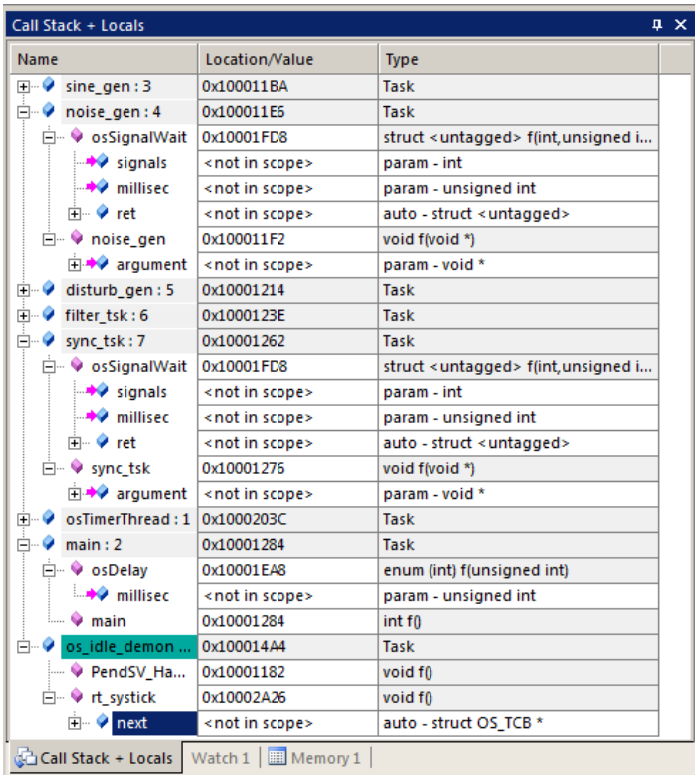
3. Right click on an element and select Callee or Caller Code to go there:



4. Stop the program.

5. Remove all breakpoints.

6. Exit Debug mode.



TIP: Recall the Call Stack and Locals window updates only when the program is stopped by one of the two breakpoints that were set on the previous page.

This is the end of the stand-alone examples.

The next section describes how to create projects from scratch using MDK 5.

The section after that describes how to use Infineon DAVE to create projects that you can easily import into µVision.

19) Creating your own MDK 5 project from scratch: (without DAVE)

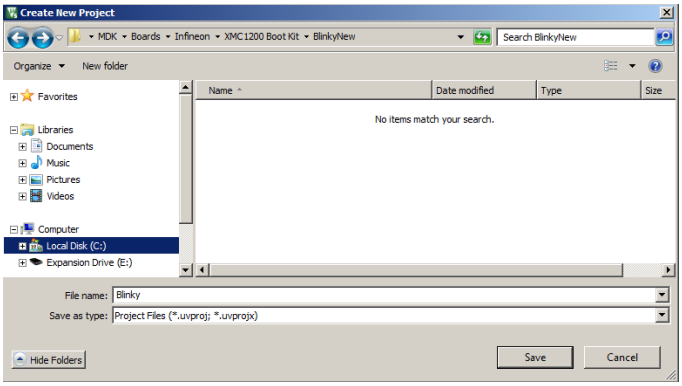
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a template for your own projects. However, we will start an example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load and run a bare Blinky example. It will have an empty main() function so it does not do much. However, the processor startup sequences are present and you can easily add your own source code and/or files. You can use this process to create any new project, including one using RTX.

Install the Infineon Software Pack for your processor:

1. Start µVision and leave in Edit mode. Do not be in Debug mode.
2. **Pack Installer:** The XMC1000 processor series Pack must be installed. This has already been done on page 4.
3. You do not need to copy any examples over.

Create a new Directory and a New Project:

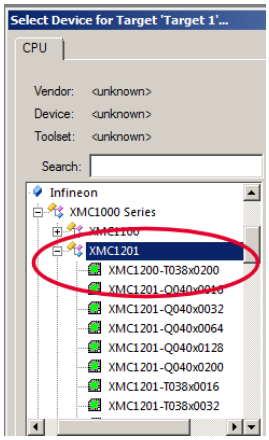
1. Click on Project/New µVision Project…
2. In the window that opens, go to the folder C:\MDK\Boards\Infineon\XMC1200 Boot Kit:
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or highlight it and select Open.
5. In the File name: box enter Blinky. Click on Save.



6. This creates the project Blinky.uvproj in C:\MDK\Boards\Infineon\XMC1200 Boot Kit\BlinkyNEW.
7. As soon as you click on Save, the next window opens:

Select the Device you are using:

1. Expand Infineon and then XMC1000 Series, then expand XMC1201 and then select XMC1200-TO38x0200 as shown here:



TIP: Processor icons in green are from the Software Packs. Grey icons are from MDK 4.7x.

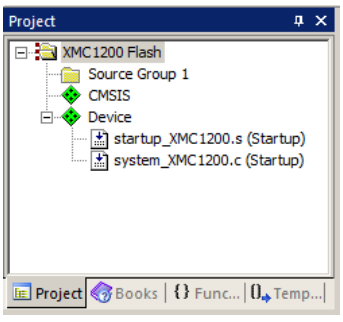
2. Click OK and the Manage Run Time window shown below bottom right opens.

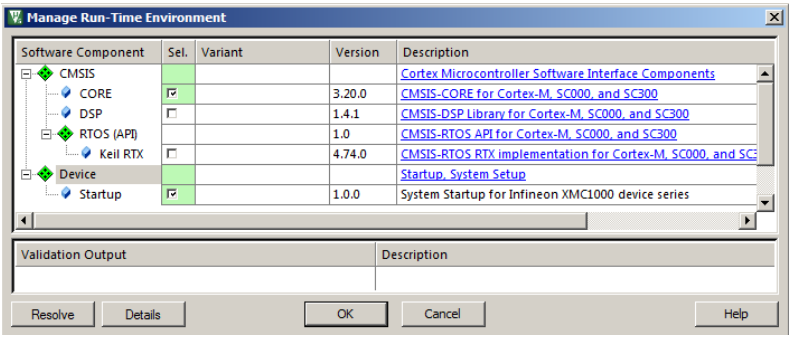
Select the CMSIS components you want:

1. Expand all the items and select CMSIS/CORE and Device/Startup as shown below. They will be highlighted in Green indicating there are no other files needed. Click OK.
2. Click on File/Save All or select the Save All icon: 
3. The project Blinky.uvproj will now be copied to Blinky.uvprojx.
4. You now have a new project list as shown on the bottom left: The appropriate CMSIS files you selected have been automatically entered and configured for the processor you chose. There are no user source files yet.
5. Note the Target Selector says Target 1. Highlight Target 1 in the Project window.
6. Click once on it and change its name to **XMC1200 Flash** and press Enter. The Target selector name will change.

What has happened to this point:

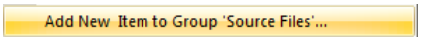
You have created a blank µVision project using MDK 5 Software Packs. All you need to do now is add your own source files.



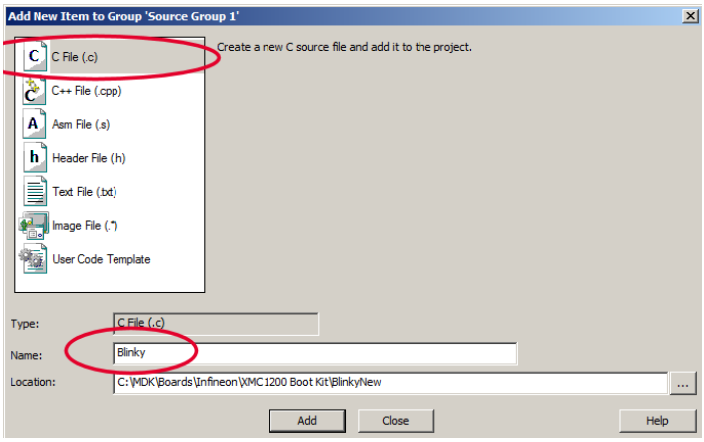


TIP: Other components such as File System, Graphics, Network and USB are part of Keil Middleware in MDK*pro*. Consult Keil tech support.

Create a blank C Source File:

1. Right click on Source Group 1 in the Project window and select 

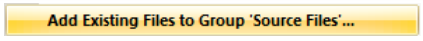
.
2. This window opens up:
3. Highlight the upper left icon: C file (.c):
4. In the Name: field, enter Blinky.
5. Click on Add to close this window.
6. Click on File/Save All or 
7. Expand Source Group 1 in the Project window and Blinky.c will now display.
8. It will also open in the Source window.



Add Some Code to Blinky.c:

1. In the blank Blinky.c, add the C code below:
2. Click on File/Save All or 
3. Build the files.  There will be no errors or warnings if all source code was entered correctly.

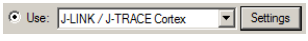
```
#include "XMC1200.h"
unsigned int counter = 0;
/*-------------------------------------------
  MAIN function
 *-------------------------------------------*/
int main (void) {

  while(1) {
        counter++;
        if (counter > 0x0F) counter = 0;
        }
}
```

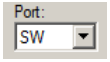
TIP: You could also add existing source files: 

We wil not do this in this tutorial.

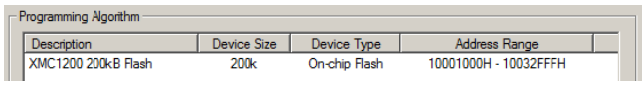
Configure the Target XMC1200 Flash: ***Please complete these instructions carefully to prevent unusual problems…***

1. Select the Target Options icon  . Select the **Target** tab.
2. Enter 32 in Xtal (MHz). This is used for timing calculations.
3. Select Use MicroLIB to compile smaller code size. 

4. Select the **Output** tab. Click on Select Folder for Objects…:
5. In the Browse for Folder window that opens: right click and create a new folder called Flash.
6. Double click on Flash to enter this folder and click OK. Compilation files will now be stored in this Flash folder.
7. Click on the **Listings** tab. Click on Select Folder for Objects…: Double click on Flash and click OK to close.
8. Click on the **Debug** tab. Select J-Link/J-Trace Cortex in the Use: box: 

9. Select the Settings: icon.
10. Select SW as shown here in the Port: box: 

A JTAG selection will not work. If your Boot Kit board is connected to your PC, you ***must*** see a valid IDCODE and Device Name in the SW Device box at this time.
11. Click on OK ***once*** to go back to the Target Configuration window. Or, fix the connection problem if you have one.
12. Click on the **Utilities** tab. Select Settings and confirm the correct Flash algorithm is present: Shown is the correct one for the Infineon XMC1200 board:



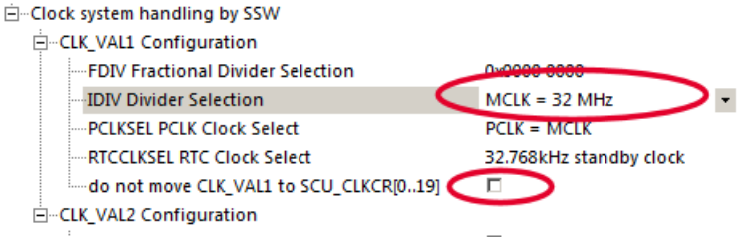
13. Click on OK twice to return to the main menu.
14. Click on File/Save All or 
15. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !

The Next Step ? Let us run your program and see what happens ! But first the clock ! Please turn the page….

Changing the Processor Clock Speed:

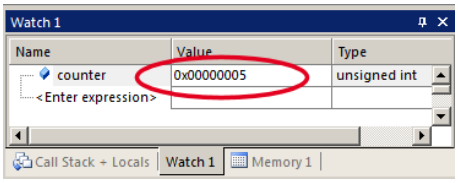
This is really easy to do in µVision. By default to processor is set to 8 MHz. We will change it to 32 MHz.

1. Open the file startup_XMC1200.s by double-clicking on it in the Project window or selecting its tab.
2. Click on the Configuration Wizard at the bottom. You can easily change values in our source code here.
3. Expand the elements so you can see these:
4. Unselect "do not move CLK_Val1 to SCU…:
5. Select 32 MHz in the drop down box:
6. To Blinky.c, just before the while(1) loop, add the line SystemCoreClockUpdate();
7. Click on File/Save All or 
8. Build the files.  The line SystemCoreClockUpdate is not needed to change the clock speed but provides the global variable SystemCoreClock that you can display in Watch 1 to display what the frequency is.



Running Your Program:

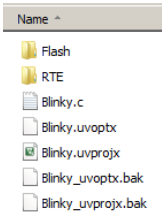
1. Connect your XMC1200 Boot Kit board to your PC with a USB cable.
2. Program the XMC1200 Flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
3. Enter Debug mode by clicking on the Debug icon . 
4. Click on the RUN icon.  Note: you stop the program with the STOP icon. 
5. No LEDs will blink since there is no source to accomplish this task. You would have to add such code yourself.
6. With the program running, right click on `counter` in Blinky.c and select Add counter to … and select Watch 1.
7. `counter` will be updating as shown here:
8. Double-click on <Enter expression> and type the global variable SystemCoreClock into Watch 1. Right click on the variable and unselect Hexadecimal Display.
9. The clock speed 32 MHz will display in Watch 1.
10. You can also set a breakpoint in Blinky.c and the program should stop at this point if it is running properly. If you do this, remove the breakpoint.
11. You could now be able to add your own source code to create a meaningful project. You can do this later.



TIP: The Watch 1 window is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip sequential values you know must exist.

Cleaning up your Project: (you only need to do this once: this is not a critical step)

We modified the folder where the output and listings files are stored. This was in Steps 3 through 7 on the preceding page. If you did a Build before this was done, there will be files in your project root directory. We want them only in .\Flash to keep the project more organized and neat.



1. Exit µVision. Otherwise, you can't delete files that it still has open.
2. Open Microsoft Explorer and navigate to:
   C:\MDK\Boards\Infineon\XMC1200 Boot Kit\BlinkyNEW\.
3. Delete all files and folders except these: (you can delete Flash – a Build will recreate it.)
4. You can also leave any backup or µVision files that use your computer or user name to retain your settings.
5. Restart µVision. Storing all compilation files stored in the .\Flash folder makes it cleaner.

TIP: If you want to save or send the project files to someone, delete the folder Flash to reduce file size. This folder and its contents are easily reconstructed with a Build.

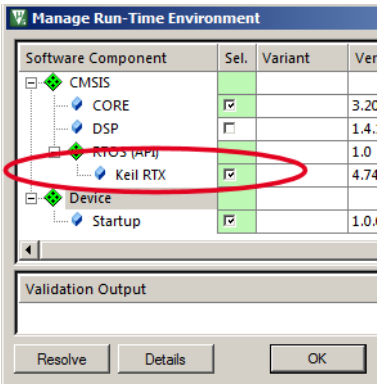
If they need the .axf executable and intermediary files and are unable to compile them, do not delete them.

20) Creating your own RTX MDK 5 project from scratch: (without DAVE)

The MDK Software Packs makes it easy to configure an RTX project. There are two versions of RTX: The first comes with MDK 4.7x and earlier. The second comes with MDK 5.10 and later. This second one is CMSIS-RTOS compliant.

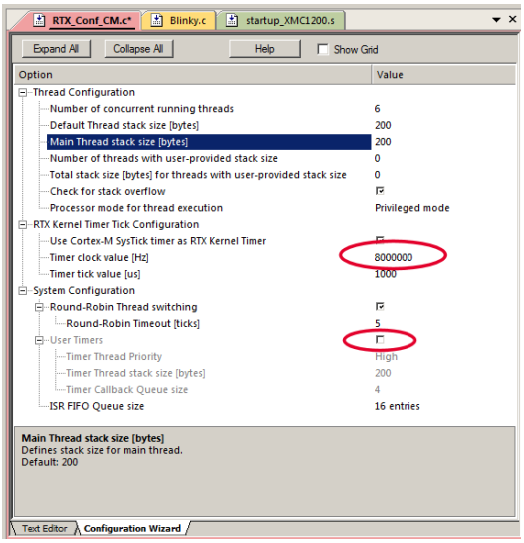
Configuring RTX is easy in MDK 5.10 and later. These steps use the same configuration as in the preceding Blinky example. You can use these procedures to convert an existing project to operate using RTX.

1. Using the same example from the preceding pages, Stop the program and Exit Debug mode.
2. In Blinky.c, at the top, add this line: #include "cmsis_os.h" The error is OK.
3. Open the Manage Run-Time Environment window:
4. Expand all the elements as shown here:
5. Select Keil RTX as shown and click OK.
6. Appropriate RTX files will be added to your project. See the Project window.
7. Click on File/Save All or



Configure RTX:

1. In the Project window, expand the CMSIS group.
2. Double click on RTX_Conf_CM.c to open it.
3. Select the Configuration Wizard tab: Select Expand All.
4. The window is displayed here:
5. Set Timer clock value: to 32000000 as shown: (32 MHz) or to the setting you selected (or not) on the previous page.
6. Unselect User Timers. Use defaults for the other settings.
7. Click on File/Save All or



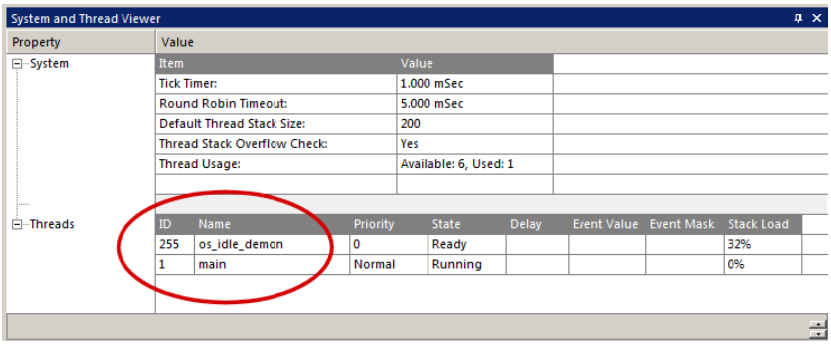
Build and Run Your RTX Program:

1. Build the files. Program the Flash: .
2. Enter Debug mode: Click on the RUN icon.
3. Select Debug/OS Support/System and Thread Viewer. The window below opens up.
4. You can see two thread listed: os_idle_demon and main. The main thread is the only one running. As you add more threads to create a real RTX program, these will automatically be added to this window.

What you have to do now:

1. You must add the RTX framework into your code and create your threads to make this into a real RTX project configured to your needs.
2. Progress to the next page.

Getting Started MDK 5: Obtain this book here: www.keil.com/mdk5/. It includes information on implementing and maintaining RTX.



Add RTX Configuration source lines to Blinky.c: We will create two threads in Blinky.c: phaseA and phaseB.
A copy of Blinky.c is provided in C:\MDK\Boards\Infineon\XMC1200 Boot Kit\RTX_text. You can either replace the existing Blinky.c with this file or cut and paste to modify it. This file has a few extra items but the source below will run correctly.

1. Stop the program:  Exit Debug mode:  **In Blinky.c add these lines:**
2. **Add this header files: You now should have three header files.**

```
#include "RTE_Components.h"
```

3. **Declare Two Global Variables: (starting near line 6)**

```
unsigned int countA = 0;
unsigned int countB = 0;
```

4. **Add a Thread ID for the two Threads: (starting near line 9 before main())**

```
osThreadId tid_phaseA;                  /* Thread id of thread: phase_a      */
osThreadId tid_phaseB;                  /* Thread id of thread: phase_b      */
```

5. **Add Thread 1: (starting at near line 12 before main())**

```
//  Thread 1 'phaseA': Phase A output
    void phaseA (void const *argument) {
      for (;;) {
        osDelay(250);                                   /* delay 250ms                 */
               osSignalWait(0x0001, osWaitForever);     /* wait for an event flag 0x0001 */
               countA++;
               if (countA > 0x10) countA = 0;
               osDelay(500);                          /* delay 250ms  */
          osSignalSet(tid_phaseB, 0x0001);            /* send signal to phaseB       */
      }}
```

6. **Add Thread 2: (starting at near line 24 before main())**

```
//  Thread 2 'phaseB': Phase B output
    void phaseB (void const *argument) {
      for (;;) {
        osDelay(250);                                   /* delay 250ms                 */
               osSignalWait(0x0001, osWaitForever);     /* wait for an event flag 0x0001 */
               countB++;
               if (countB > 0x10) countB = 0;
               osDelay(500);                          /* delay 250ms                */
               osSignalSet(tid_phaseA, 0x0001);       /* send signal to phaseA       */
      }}
```

7. **Define each thread: (starting near line 36 before main() after)**

```
osThreadDef(phaseA, osPriorityNormal, 1, 0);
osThreadDef(phaseB, osPriorityNormal, 1, 0);
```

8. **int main(void) is already located at this point: (starting near line 39)**
9. **Create the two Threads phaseA and phaseB: (starting near line 43 in main() before the while(1))**

```
tid_phaseA = osThreadCreate(osThread(phaseA), NULL);
tid_phaseB = osThreadCreate(osThread(phaseB), NULL);
```

10. **Send a signal to Thread 1 to start it: (starting near line 45 in main() before the while(1))**

```
osSignalSet(tid_phaseA, 0x0001);          /* send signal to phaseA thread   */
osDelay(osWaitForever);
```

11. **While(1){ } is already located at this point: (near line 49)**
12. Select File/Save All or  .

On the next page we will compile and run your new RTX Blinky project.

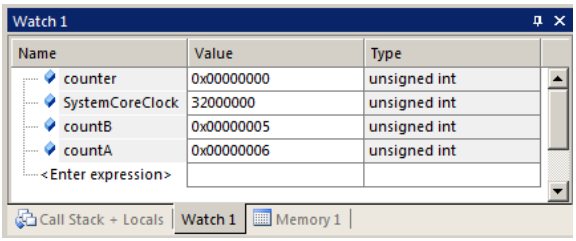
Compile and Run Your RTX BlinkyNEW Project:

1. Build the files.  If there are errors or warnings please fix these before continuing.
2. Program the Flash: 
3. Enter Debug mode:  Click on the RUN icon. 
4. No LEDS will blink. This program does not use them.

Add the two global variables countA and countB to the Watch 1 window.

5. Right click on countA and select Add countA to… and select Watch 1.
6. Right click on countB and select Add countB to… and select Watch 1.
7. These two variables will be displayed in Watch 1 and will be incrementing as shown here:

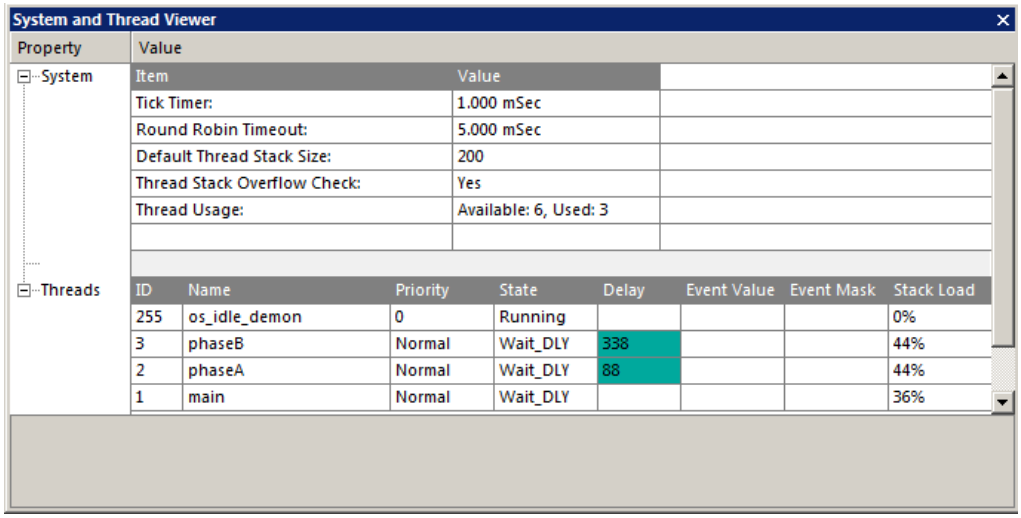
Congratulations: Your First RTX program is running correctly !



TIP: counter and SystemCoreClock are left over from previous steps.

Open the System and Threads Viewer:

8. Select Debug/OS Support and select System and Threads Viewer. the following window opens up: You probably have to drag into the middle of your screen.
9. The various elements will be updating in real-time as your RTX program is running.
10. Note the idle daemon is running most of the time. You can change this ratio if you want to.
11. Set a breakpoint in each thread.
12. Each time you click on RUN the program will advance to the next thread repeatedly.
13. The active thread will display Running in the State column.



This ends the exercise creating your own RTX Blinky project. You can easily add more threads and modify the timing. When you are finished, stop the program and leave Debug mode.

Obtain the Keil Getting Started MDK 5 manual: www.keil.com/mdk.
It contains valuable information of using RTX. Extensive help files are included with MDK 5.
So far, we have created projects from scratch or a fundamental beginning.
The next section deals with creating projects using Infineon DAVE.

PART C: Creating projects with Infineon DAVE and MDK 5:

This document uses DAVE 3.1.10 to create both a Blinky and an RTX_Blinky project. Please install DAVE to your computer in the usual manner. This action will install the correct J-Link USB drivers. Select Help/Check for Updates to make sure you have the latest software.

You must have downloaded the XMC1000 Packs into µVision as shown on page 4.

21) Blinky using DAVE:

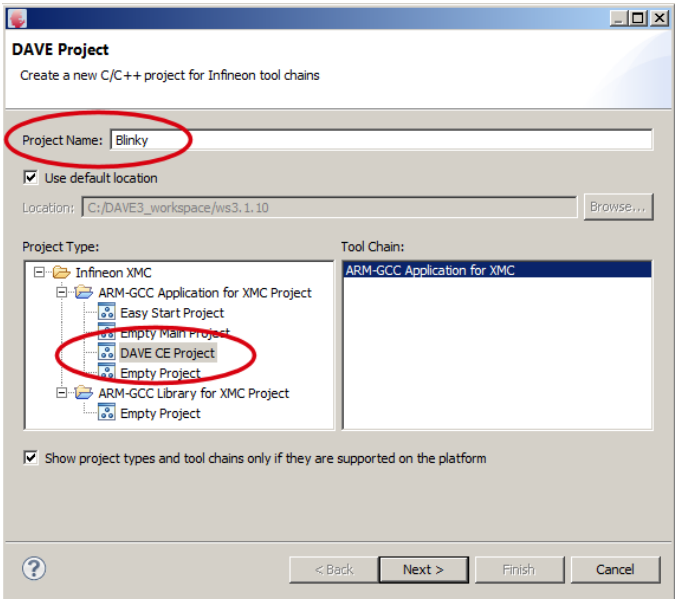
We will create a µVision project using DAVE to blink the two LEDs on the Boot Kit board. Two PWMs will be created and configured: each one blinks one LED. DAVE will then create a µVision project and µVision will then import it.

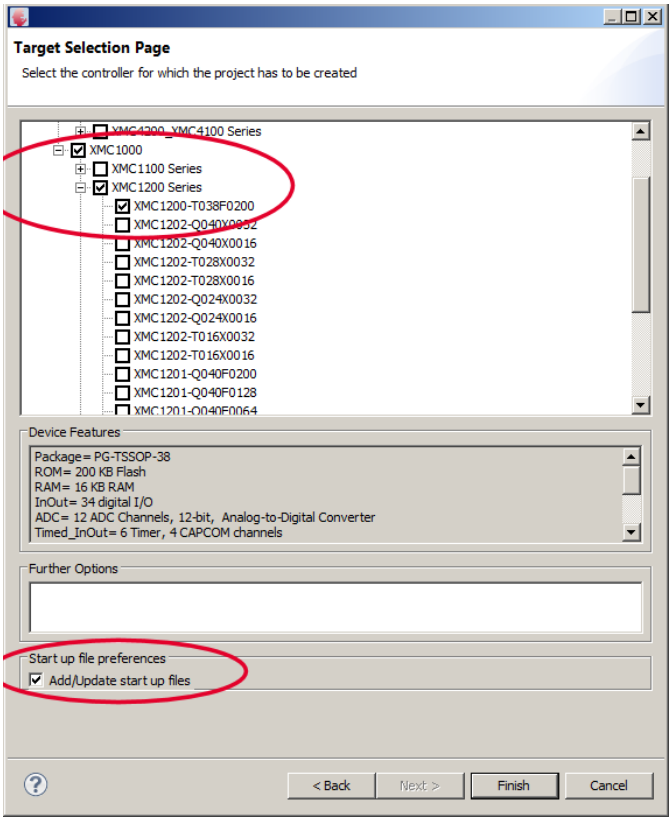
Start DAVE and create a DAVE project:

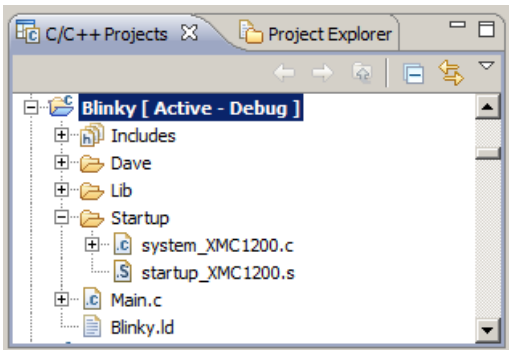
1. Start DAVE by clicking on its icon:
2. Select File/New/Dave Project and this window will open:
3. Select DAVE CE Project: Always select CE.
4. In Project Name box enter Blinky as shown here:
5. Select Next > and the Target Selection Page opens:
6. Select XMC1200-TO38F200 or the processor you are using as shown in the Target Selection window:
7. Confirm Add/Update startup files is enabled.
8. Click on Finish. DAVE will create your project.
9. The Blinky project will be displayed in the DAVE C/C++ Projects window as shown below.
10. Expand some of the elements.
11. You will see a Main.c and two CMSIS startup files.
12. Double click on Main.c, it will open in a window.
13. You can see DAVE has created a template Main.c with an int main(void) function but not much else.
14. If you open the startup files found in the Startup folder, you can see startup_XMC1200.s which contains processor initialization code and system_XMC1200.c contains mostly clock settings.

TIP: DAVE dynamically saves all your work as you progress into your default workspace: C:\DAVE3_workspace. You can manually save your work with File/Save or Ctrl-S.

15. We will next create two PWMs to blink the LEDs.

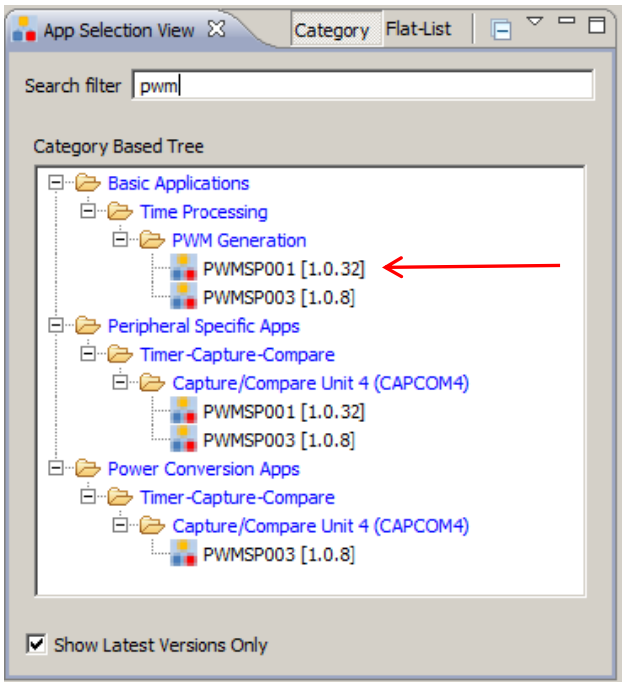






DAVE Apps:
DAVE includes Apps that you can graphically configure to create programs to suit your needs. These are programs DAVE uses to create the source code files that will become part of your µVision project. Select Help/Check for DAVE App Updates to make sure you have the latest software.

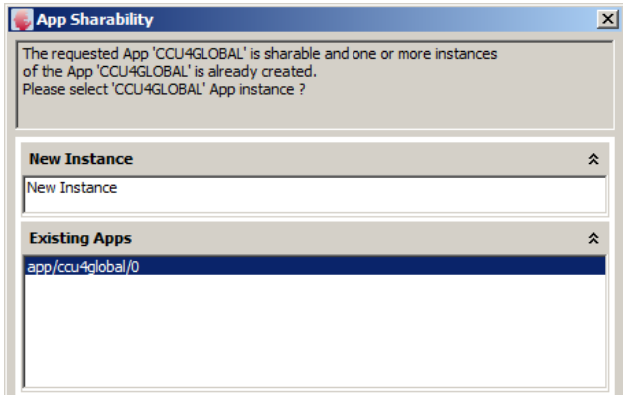
1. Apps are listed in the App Selection View.
2. Scroll through them to see your options.
3. Enter **pwm** in the Search filter as shown here:
4. If you hover your mouse over an App, a description of it will display. See the **TIP:** below for more details.
5. We will use two instances of the App PWMSP001 to blink each LED on and off.

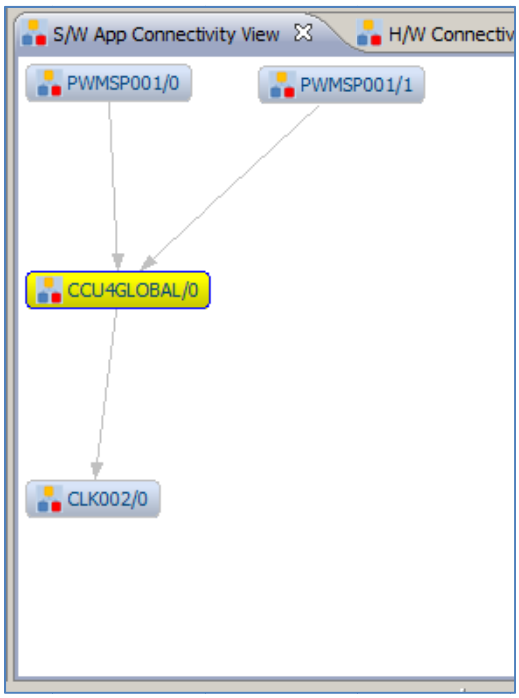


Create Two PWMs to Blink the LEDs:

1. Under the heading PWM Generation: double click on the PWMSP001 This will be added in the S/W App Connectivity View.
2. Double click again on the same PWMSP001 App.
3. The Apps Shareability window will open. Select app/ccu4global/0 as shown below:
4. Click on OK.
5. The second instance will now be added:
6. Your S/W App Connectivity View will be similar to the one shown below: You can move your Apps around.
7. Note there are two instances of PWMSP001 (denoted by /0 and /1), a Capture/CompareUnit 4 and a CLK002/0 clock App.
   These last two Apps are automatically inserted by Dave.

TIP: App Help: Select Help and click on Help Contents. On the left side, click on DAVE Apps. A listing of the Apps will open.



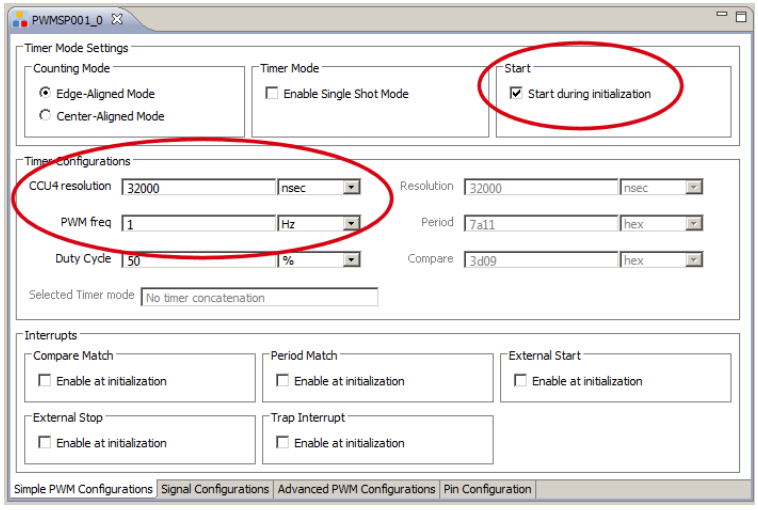


Concept: Manual Pin Assignment:

The resource solver integrated into DAVE assigns resources that are required by the DAVE Apps to the available chip resources. The resources solver ensures that the resources assignments are conflict free and all connectivity requirements are fulfilled. In particular for pad / pin assignments, it is often required that specific constraints of the HW PCB design are also considered. With the manual pin assignment functionality in DAVE such pin constraints for the resource solver can be defined.

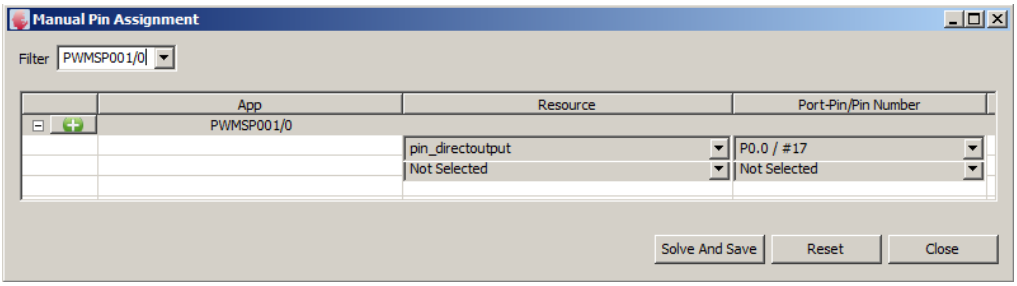
Configure PWM /0:

1. Right click on PWMSP001/0 and select UIEditor. This window opens up:
2. Set Start to Start during initialization:
3. Set CCU4 Resolution to 32000 nsec.
4. Set PWM freq to 1 Hz:
5. Select the Signal Configurations tab and set the Output Level to Passive High.
6. Leave all else at their defaults.



Connect PWM /0 output to GPIO Port Pin P0.0:

1. Right click on PWMSP001/0 again and select Manual Pin Assignment and the next window opens up.
2. Select Resource to pin_directoutput and Port Pin number P0.0/ #17 as shown below:
3. Select Solve And Save and then Close to assign this pin.



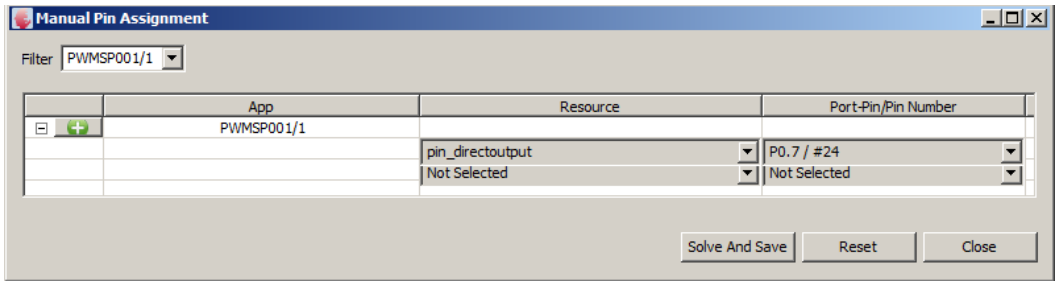
Configure PWM /1:

1. Right click on PWMSP001/1 and select UIEditor and a similar window opens up.
2. Set Start to Start during initialization:
3. Set CCU4 Resolution to 32000 nsec.
4. Set PWM freq to 1 Hz:
5. Select the Signal Configurations tab and set the Output Level to Passive Low. This puts PWMMSP001/1 180 º out of phase with PWMMSP001/0. When one LED is on, the other will be off.

Connect PWM /1 output to GPIO Port Pin P0.7:

1. Right click on PWMSP001/1 again and select Manual Pin Assignment and the next window opens up.
2. Select Resource to pin_directoutput and Port Pin number P0.7/ #24 as shown below:

Note: P0.0 is no longer listed. DAVE knows you have already assigned Pin 0.0 so it removes it from the list.



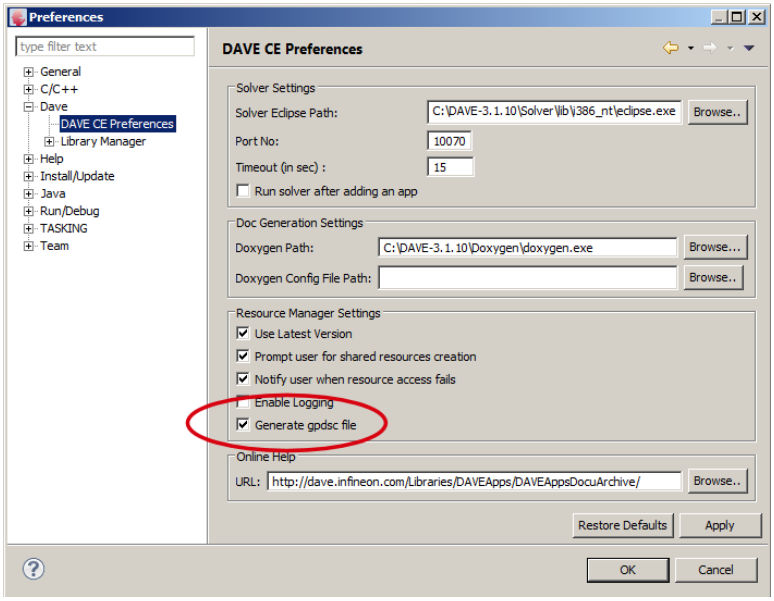
3. Select Solve And Save and then Close to assign this pin.
4. The PWMs are now completely configured. All we need do is Build the project and export it into µVision 5.

TIP: If your entries do not seem to work: ensure they are accepted by pressing Enter or clicking on another element.

Generator Package Description (gpdsc) File:
DAVE produces a complete µVision project and in addition a gpdsc file. This file is used to start, import and configure the appropriate files into µVision.

Configure DAVE to create a gpdsc file: (you only have to do this once in DAVE for all your projects)

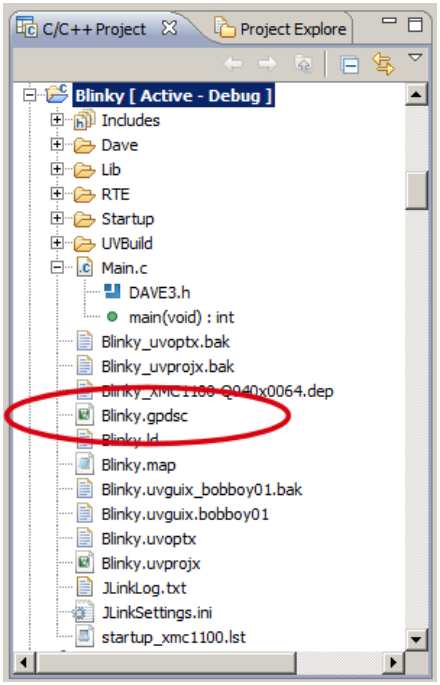
1. In the DAVE main window, select Window/Preferences and this window opens up:
2. Expand Dave and select DAVE CE Preferences as shown here:
3. Select Generate gpdsc file as shown:
4. Click on OK.



Generate the Code:

5. Click on the Generate Code icon or select DAVE/Generate Code.
6. The appropriate files will be created and a progress bar will be displayed.
7. The C/C++ Projects window will display the files in this project.
8. Blinky.gpdsc will be visible in the window as shown below right:

TIP: If you do not want another instance of µVision running, close the running instance(s) before double clicking on the gpdsc file.
Running multiple instances of µVision is permitted but a conflict with the J-Link Lite might arise in this case.



Import into µVision:

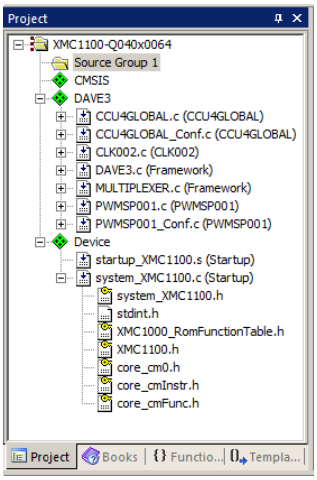
1. Close µVision running. See the TIP: above.
2. Double click on Blinky.gpdsc and a new instance of µVision will be started.

TIP: At this time you can shut DAVE down if you prefer. It is not needed again unless you want to add or modify any settings that must be done in DAVE. Otherwise, µVision can work stand alone with the files DAVE created for it.

On the next page, we will configure µVision, build the project, program it to Flash and run it.

Configure µVision:

After double clicking on Blinky.gpdsc, µVision will be started. The source files DAVE created are listed in the Project window as shown here:



This is almost complete: you must now add a file containing a main() function. You can add your own, or use the one DAVE created. We will use the one DAVE created.

1. Right click on Source Group 1 and select Add Existing Files to Group ….:

.

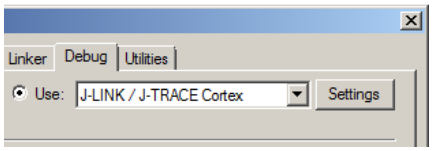
2. In the Add Files window, highlight main.c and select Add and then Close.
3. Main.c will now be added to Source Group 1. Expand Source Group 1 to see it.

TIP: If you create and add your own new file, you can select Add New Item to Group… and some templates will be offered that you can choose from.

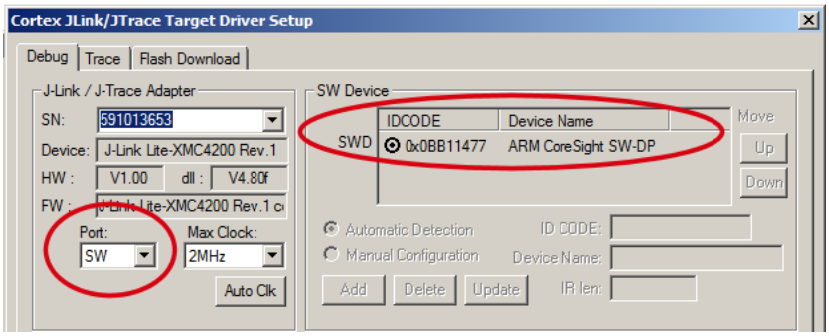
Configure the Debug Adapter and Flash programming:

1. In µVision, select the Options for Target Options icon: Options or ALT-F7.
2. Select the Target tab and select Use MicroLIB to compile for smaller program size:

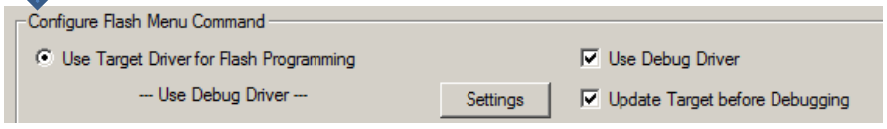
3. Select the Debug tab.
4. Select J-Link / J-Trace Cortex as shown here:



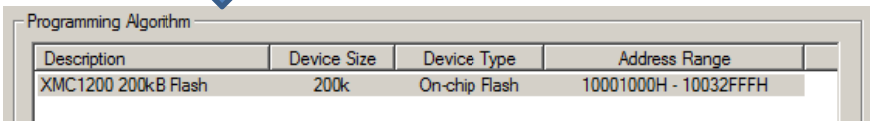
5. Select Settings: on the right side of this window.
6. Select SW and not JTAG as shown here in the Port: box:
7. Confirm you see an IDCODE and Device Name in SW Device box:
   If this box is blank, or an error is displayed, ***you must fix*** this before you can continue. Check your cables and make your Boot Kit board is plugged in.
   Ensure you have the correct J-Link USB drivers installed.



8. The SN: box indicates µVision is connected to the J-Link Lite. The SW Device box indicates µVision is also connected to the XMC1200 processor CoreSight debug module.
9. Click on OK to leave this window.
10. Select the Utilities tab: Select Use Debug Driver as shown:



11. Select Settings. A list appears. Select XMC 1200 XMC 200 kB Flash and then Add again. The Flash algorithm will display as shown here.



12. Click on OK to close and return to the main µVision window.
13. Select File/Save All or .

At this point, you have now configured µVision to be able to build and run the project that DAVE created.

Build the µVision project created by DAVE:

1. Compile the source files by clicking on the Rebuild icon.  . You can also use the Build icon beside it.

TIP: You will probably get a warning about the line Return() in Main.c not reachable. You can ignore this warning.

2. Program the XMC1200 Flash by clicking on the Load icon:  Progress will be indicated in the Output Window.

TIP: Since Update Target before Debugging is selected (see the previous page), you can skip step 2. Entering Debug mode will automatically program the Flash if there is a new executable .axf file available.

3. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.

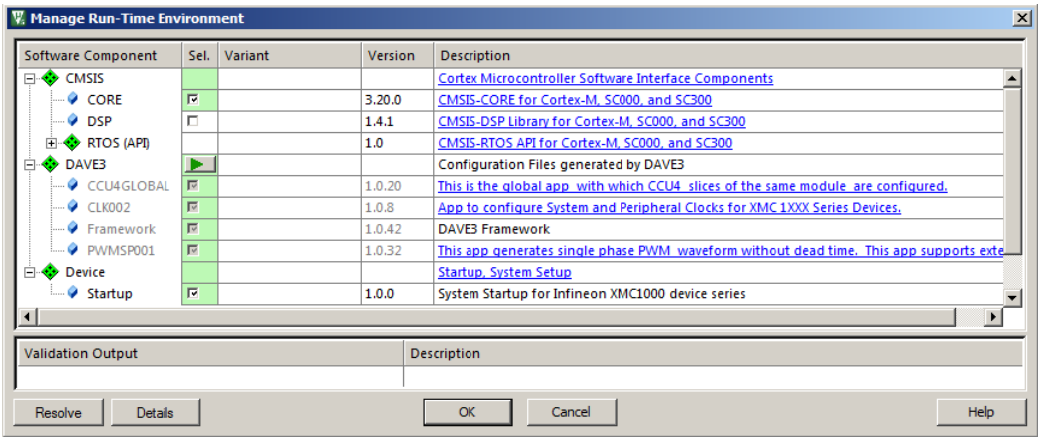
Concept: Software Development Information Notes:

1. During your development cycle, you will add your own source files to the µVision project after it is created by DAVE. Right click on a Group name (such as Source Group 1) and select either Add Existing files or New Files as appropriate. Modify these files to your needs. DAVE will not modify or replace the main.c file it initially produced and you subsequently modified.
2. You should not modify any files with µVision that were created by DAVE. These are located under the DAVE3 group name. Use DAVE to make any modifications to these files.
3. To change anything you created in DAVE, such as a peripheral configuration, make modifications or additions with DAVE and the updated files will be imported into µVision. You must select YES when notified files have changed outside out of µVision. If you had changed any DAVE created files inside µVision, your changes will be lost.
4. At this point, if you do not yet to change anything in DAVE, you can close it and select your µVision project directly from your DAVE workspace rather than double-clicking on the gpdsc file.

4. Click on the RUN icon.  **Note:** you stop the program with the STOP icon. 
5. Two LEDs will blink at 1 second intervals. Each LED is controlled by its own PWM you created in DAVE.

Run Time Environment:

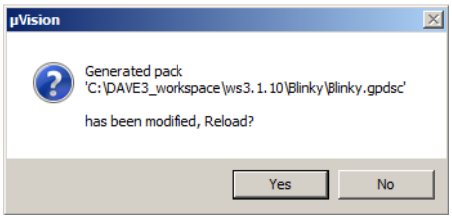
1. In µVision, stop the program.  and exit Debug mode. 
2. Click on the Manage Run-Time Environment icon:  The window below opens up.
3. This window displays the components of your project from µVision.



4. The components created by DAVE are listed under DAVE3. These cannot be changed in this menu.
5. The green arrow can be used to start DAVE. If DAVE is already running, an error will be generated.  Two instances of DAVE cannot share the same workspace.
6. The CMSIS CORE and Startup files were also created by DAVE.
7. Clicking on any blue line will display information from the web about this selection using your default browser.

How to Update Configurations inside DAVE:
We will change the frequency of one of the PWMs to demonstrate changing DAVE configuration files. Modify DAVE created files this way rather than modifying these files directly in µVision.

1. In µVision, stop the program. and exit Debug mode. Do not exit µVision.
2. In the DAVE S/W Connectivity View, right click on one of the PWMSP001 Apps and select UIEditor.
3. Change PWM freq from 1 to 5 Hz. Press Enter or click in another box to enter this new setting.
4. Click on the Generate Code icon or select DAVE/Generate Code. A new project will be generated.
5. When it has completed, bring µVision back into focus.
6. There will one or more indications that has been changed: Click on Yes for all of these. Files modified by DAVE are imported into µVision.
7. Select File/Save All or .



Compile, Load into Flash and RUN:

1. Compile the source files by clicking on the Rebuild icon. .
2. Enter Debug mode. The Flash will automatically be programmed.
3. Click on the RUN icon.
4. One of the LEDs will blink at 5 times per second reflecting the change you made it DAVE.
5. The source files you added are not replaced or modified by DAVE.

TIP: Do not select RTX in the µVision RTE window if your project is using RTX. DAVE supplies RTX using the RTOS001 App. A future version of MDK will allow the selection of RTX from the Software Packs. This allows better control and ensures the latest version of RTX is always available from the Software packs on the web.
See the next page that shows how to create an RTX_Blinky project using DAVE.

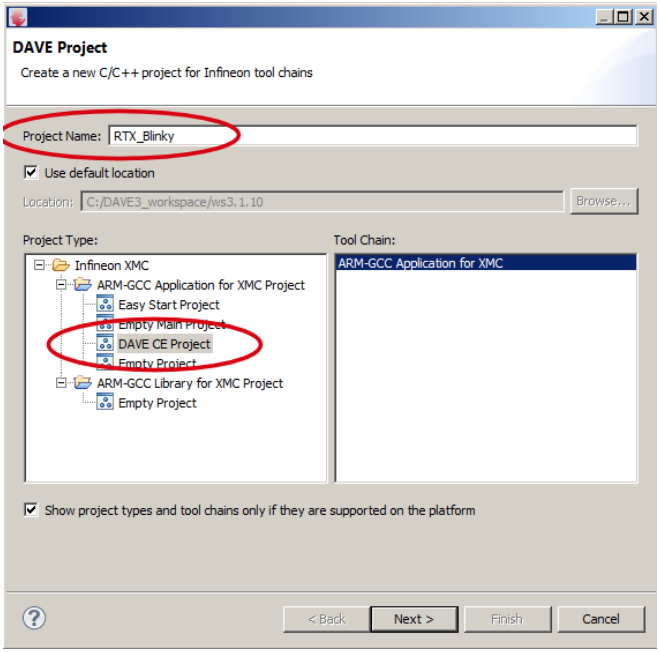
22) Create an RTX_Blinky project using DAVE and MDK 5:

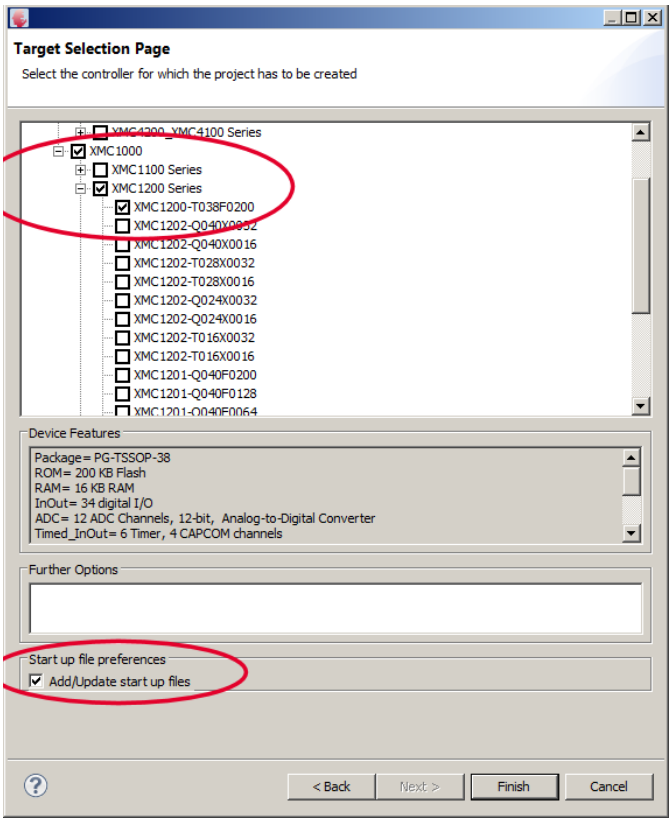
We will create a µVision project using DAVE to blink the two LEDs on the Boot Kit board. RTX is used. We will use direct software control of Ports 0.0 and 0.7 to blink the LEDS using two simple threads. DAVE will then create a µVision project and µVision will then import it. It is easy to add additional threads.

TIP: It is useful in DAVE to click on Help/Check for Updates and also DAVE App Updates.

Start DAVE and create a DAVE project:

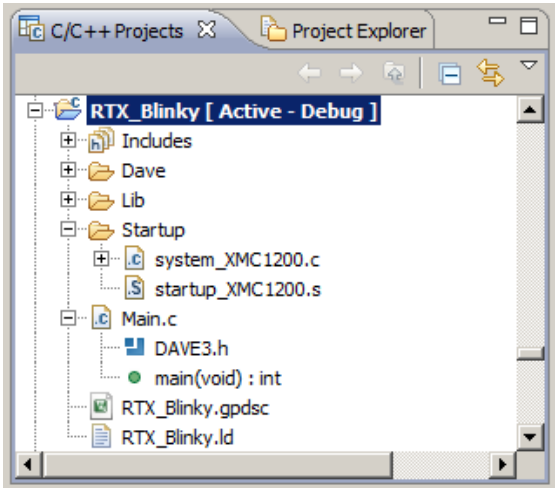
1. Start DAVE by clicking on its icon:
2. Select File/New/Dave Project and this window will open:
3. Select DAVE CE Project: Always use DAVE CE.
4. In Project name box enter RTX_Blinky as shown here:
5. Select Next > and the Target Selection Page opens:
6. Select XMC1200-TO38F200 or the processor you are using as shown in the Target Selection window:
7. Confirm Add/Update startup files is enabled.
8. Click on Finish. DAVE will create your project.
9. The Blinky project will be displayed in the C/C++ Projects window as shown below.
10. Expand some of the elements.
11. You will see a Main.c and two CMSIS startup files.
12. Double click on Main.c, it will open in a window.
13. You can see DAVE has created a template main.c with an int main(void) function but not containing much else.
14. If you open the startup files, you can see startup_XMC1200.s which contains processor initialization code and system_XMC1200.c which contains mostly clock settings.





TIP: DAVE dynamically saves all your work as you progress with your workspace. You can manually save your work with File/Save or Ctrl-S.

15. On the next page, we will create and configure the two IO ports to blink the LEDs.



DAVE Apps:

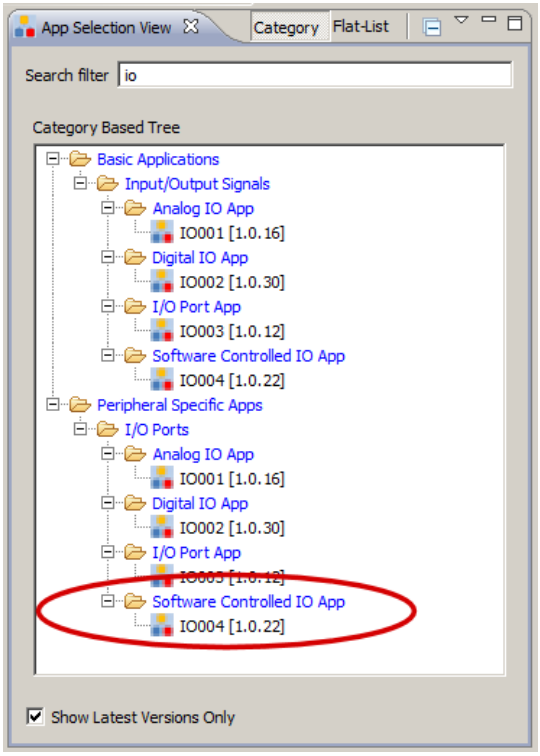
Dave includes Apps that you can graphically configure to create programs to suit your needs. We will use the RTOS001 App and two instances of the Software Controlled IO App IO004 to create our demonstration program.
TIP: You can also create a project with the RTOS App and implement RTX using µVision as described on page 20.

TIP: Your version numbers listed for each App might be different from the ones shown in the screens here. Always use the versions shown or later only.

In this project, RTX RTOS files are supplied by DAVE.

Apps available are listed in the App Selection View.

1. Scroll through them to see your options.
2. If you hover your mouse over an App, a description of it will display as shown in this window.
3. We will use two instances of the App IO004 to blink each LED on and off directly through the GPIO ports.
4. Enter **io** in the Search filter as shown here: Press Enter to make it active.

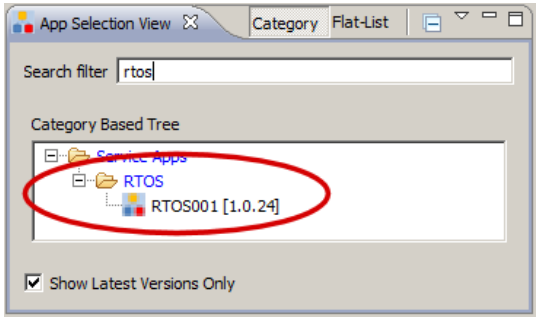


Create Two IO ports to Control the LEDs:

6. Under the heading I/O Ports/Software Controlled I/O App, double click on IO04 as shown here:
7. This will be added in the S/W App Connectivity View.
8. Double click again on the same IO04 App.
9. The second instance will now be added:
10. Your S/W App Connectivity View will contain the two IO Apps. There are two instances of IO04 (denoted by /0 and /1)

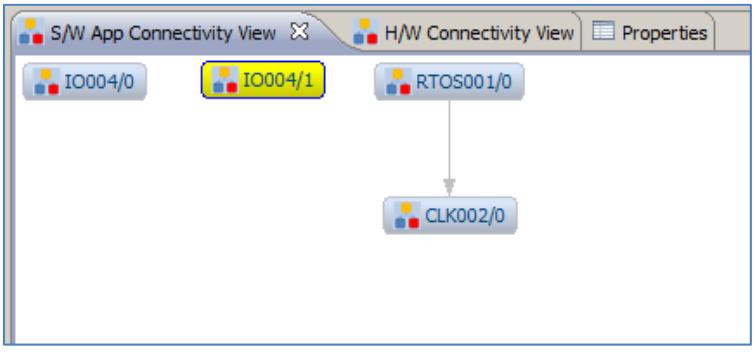
Create the RTX RTOS001 Instance:

11. Enter rtos in the Search filter as shown here: Press Enter to make it active.
12. Double-click on RTOS001. This will be entered in your project.
13. The clock CLK002/0 is inserted automatically by DAVE.
14. Your completed App Connectivity View will look like the one below. You can move the icons around.
15. The next steps will configure the IO and RTOS Apps.



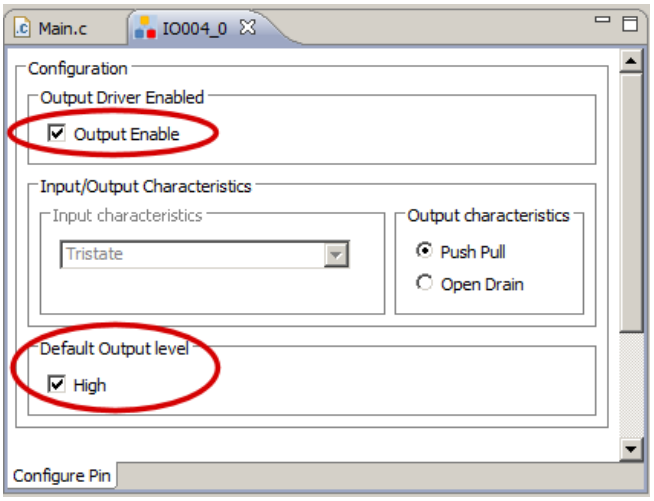
TIP: To use the RTX that comes with µVision, do not create an RTOS001 above. In the Manage Run-time Environment select RTX. See page 35 under Manage RTE in step 6.

TIP: App Help: Select Help and click on Help Contents. On the left click on DAVE Apps. A listing of the Apps will open.



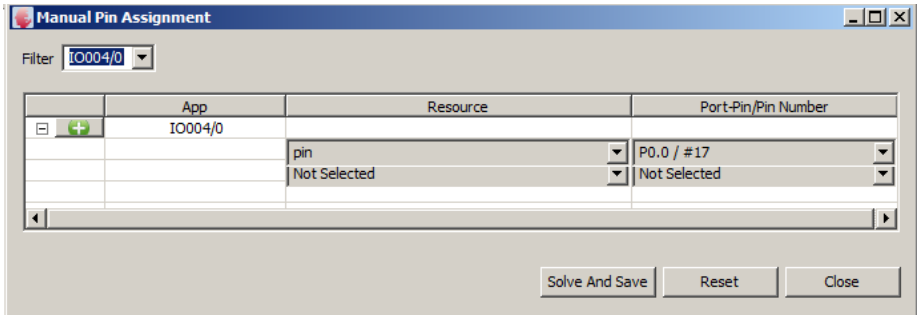
Configure IO004/0:

1. Right click on IO004/0 and select UIEditor. This window opens up:
2. Enable Output Enable as shown here:
3. Set Default Output level to High: This will cause the LEDs to blink alternatively.
4. Leave all else at their defaults.



Connect IO004/0 output to GPIO Port Pin P0.0:

5. Right click on IO004/0 again and select Manual Pin Assignment and the next window opens up.
6. Select pin in the Resource box. Select P0.0/ #17 in the Port Pin Number box as shown below:
7. Select Solve And Save and then Close to assign this pin.



Configure IO004/1:

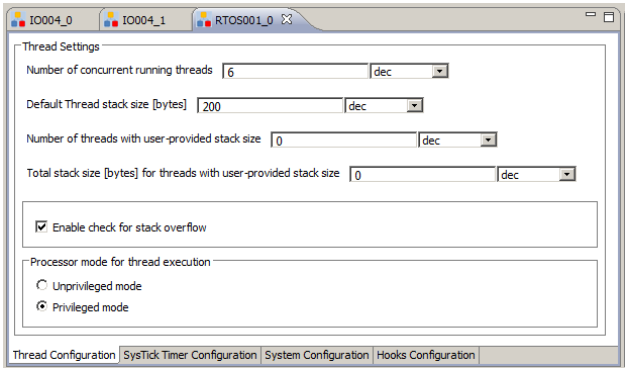
1. Right click on IO004/1 and select UIEditor and a similar window to the one above opens up.
2. Enable Output Enable as shown above the same way you configured IO004/0.
3. Unselect Default Output level/ High. This will cause the LEDs to blink alternatively.

Connect IO004/1 output to GPIO Port Pin P0.7:

4. Right click on IO004/1 again and select Manual Pin Assignment and the Manual Pin Assignment window opens.
5. Select Resource to pin and Port Pin number P0.7/ #24 similar to the above screen.
6. Select Solve And Save and then Close to assign this pin.
7. The IO ports are now configured.

Configure RTOS001/0:

1. Right click on RTOS001/0 and select UIEditor, The window that opens has four tabs:
2. We can use the default values.
3. Click on each tab to view the selections available. Leave everything at their default values.
8. It is possible to modify the same settings in the file in µVision. It is best to make any modifications inside DAVE rather than from µVision.

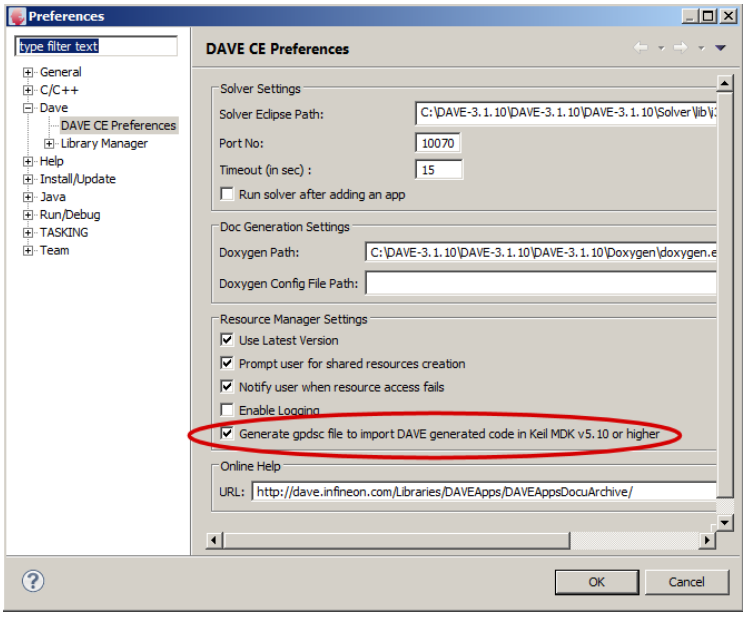


9. RTX is now completely configured. All we need do is Build the project and export it into µVision 5. We will need to add an application source code to Main.c.

Generator Package Description (gpdsc) File:
DAVE produces a complete µVision project and in addition a gpdsc file. This file is used to start, import and configure the appropriate files into µVision.

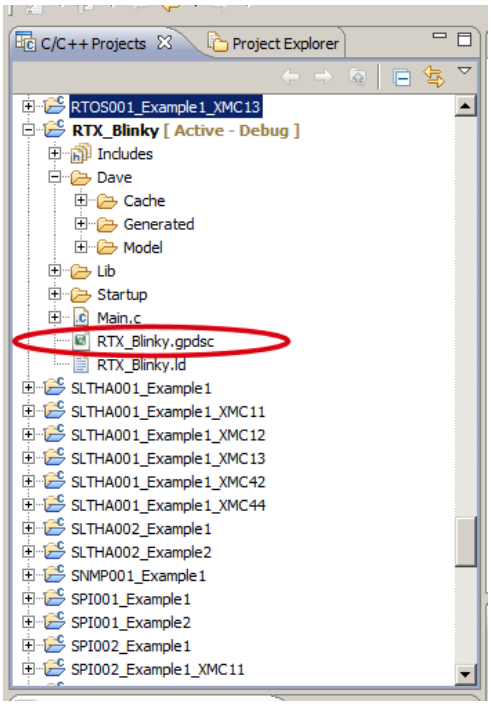
Configure DAVE to create a gpdsc file:
TIP: You need set the Generate gpdsc bit only once for all your DAVE projects.

1. In the DAVE main window, select Window/Preferences and this window opens up:
2. Expand Dave and select DAVE CE Preferences as shown here:
3. Select Generate gpdsc file as shown:
4. Click on OK.



Generate Code.

1. Click on the Generate Code icon or select DAVE/Generate Code.
2. The appropriate files will be created and a progress bar will be displayed.
3. The C/C++ Projects window will display the files in this project.
4. Blinky.gpdsc will be visible as shown in the screen below right:



.

Import into µVision:

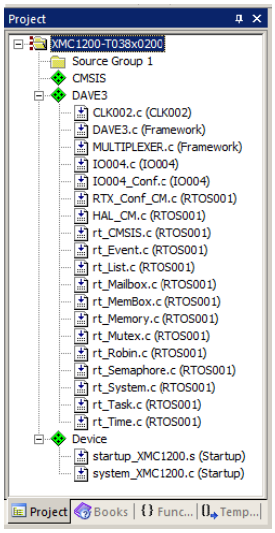
1. Close µVision. This prevents multiple instances from opening.
2. Locate the file Blinky.gpdsc as shown here:
3. Double click on Blinky.gpdsc and a new instance of µVision will be started.

TIP: If you do not want another instance of µVision running, close the running instance(s) before double clicking on the gpdsc file.
Running multiple instances of µVision is permitted but a conflict concerning the J-Link Lite might arise in this case.

On the next page, we will configure µVision, add source files, build the project, program the Flash and run it.

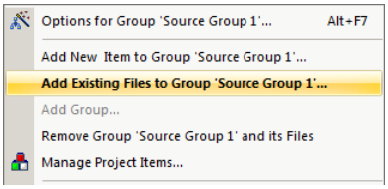
4. You can optionally close DAVE now. We will not need it.

Configure µVision:
After double clicking on Blinky.gpdsc, µVision will be started. The source files DAVE created are listed in the project window as shown here:



Enter Main.c: You must add a file containing a main() function. You can add your own, or use the one DAVE created. We will use the one DAVE created.

1. Right click on Source Group 1 and select Add Existing Files to Group ….:



.

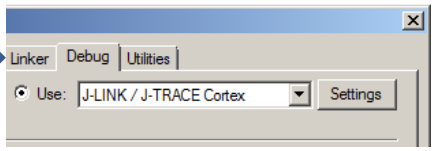
2. In the Add Files window, highlight Main.c and select Add. Select Close.
3. Select File/Save All or  .

TIP: If you create and add your own file, you can select Add New Item to Group… and some templates will be offered that you can choose from.

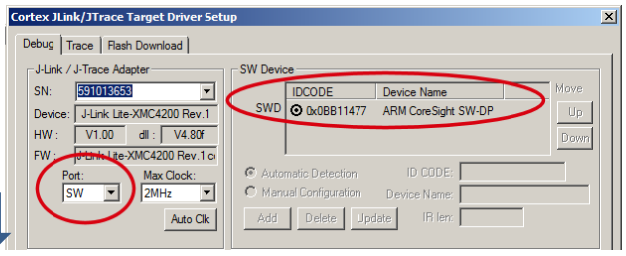
Configure the Debug Adapter and Flash programming:

1. Plug in your Boot Kit to your PC with a USB cable.
2. In µVision, select the Options for Target Options icon: Options  or ALT-F7 and select the Target tab.
3. Select Use MicroLIB for smaller compiled executable: 

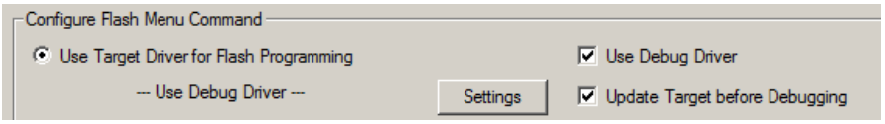
4. Enter 32 MHz in the Xtal (MHz) box.
5. Select the Debug tab. Select J-Link / J-Trace Cortex as shown here:



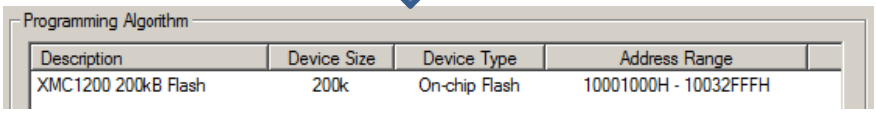
6. Select Settings: on the right side of this window.
7. Select SW and not JTAG as shown here in the Port: box:
8. Confirm you see an IDCODE and Device Name in SW Device box:
If this box is blank, or an error is displayed, ***you must fix*** this before you can continue. Check your cables and make your Boot Kit board is plugged in. Ensure the correct J-Link USB drivers are installed.
9. The SN: box indicates µVision is connected to the J-Link Lite. The SW Device box indicates µVision is connected to the XMC1200 processor CoreSight debug module.



10. Click on OK to close this window.
11. Select the Utilities tab: Select Use Debug Driver:



12. Select Settings and then select Add:
13. A list appears. Select XMC 1200 XMC 200 kB Flash and then Add again. The Flash algorithm will display as shown here:



14. Click OK twice to close and return to the main µVision window.
15. Select File/Save All or  .

At this point, you have configured µVision to be able to build and run the RTX_Blinky project that DAVE created. Main.c does not yet have the necessary RTX components to make a complete RTOS solution.

Build the µVision project created with DAVE:

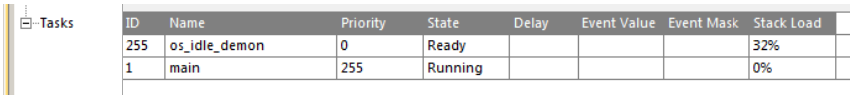
1. Compile the source files by clicking on the Rebuild icon.  . You can also use the Build icon beside it.

TIP: You will probably get an inconsequential warning about the line Return() in Main.c not reachable. You can comment this line out or ignore the warning.

2. Program the XMC1200 Flash by clicking on the Load icon:  Progress will be indicated in the Output Window.

TIP: Since Update Target before Debugging is selected (see the previous page), you can skip step 2. Entering Debug mode will automatically program the Flash if there is a new file available.

3. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
4. Click on the RUN icon.  **Note:** you stop the program with the STOP icon. 
5. One LED will come on indicating the two IO Apps are activated. Recall the main() function created by DAVE is almost blank. This will be rectified on the next page.
6. Select Debug/OS Support and select System and Thread Viewer. This window will open:
7. If you see the os_idle_demon as ID 255 and main as ID 1 and Running, everything is correct to this point.

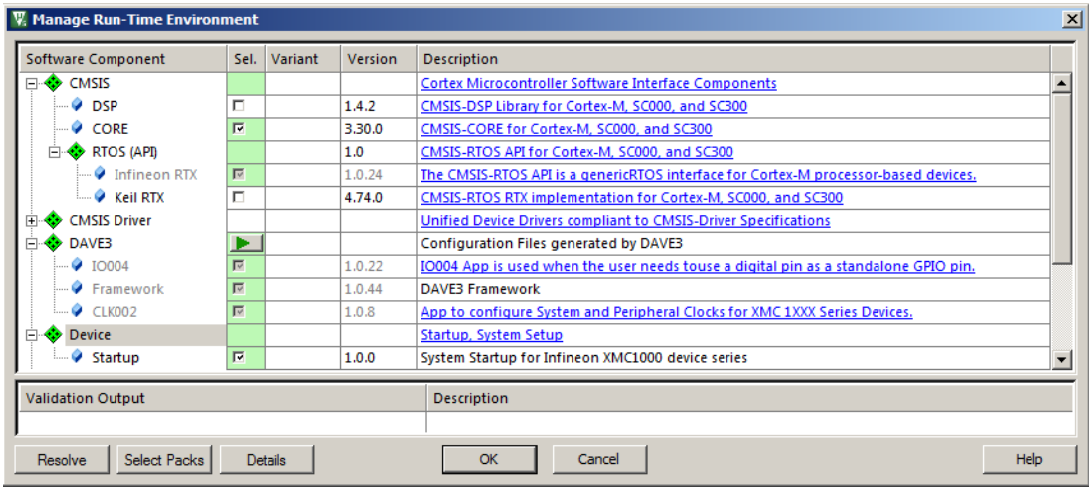


NOTE: At this writing, continuing development of the DAVE RTOS App is causing this window to be blank.

8. We need to add some source code to create the RTX environment and to blink the LEDs. This is on the next page.

Manage Run Time Environment:

1. Stop the program  and exit Debug mode  .
2. Click on the Manage Run-Time Environment (RTE) icon:  The window below opens up:
3. This window displays the components of your project.



The components created by DAVE are listed under DAVE3. These cannot be changed in this menu.

4. The green arrow can be used to start DAVE. If DAVE is already running, an error will be generated.  Two instances of DAVE cannot share the same workspace.
5. The CMSIS CORE and Startup files were also created by DAVE.
6. Note that Infineon RTX is selected and not Keil RTX. DAVE automatically made this selection.
7. Clicking on any blue line will display information from the web about this app using your default browser.
8. The components (not shown above) are the Keil Middleware. Contact Keil sales or tech support for information.
9. Click on OK to close the RTE window.

Add RTX Configuration source lines to Main.c: We will create two threads in main.c: phaseA and phaseB.
A copy of Main.c is provided in the software package that accompanies this document for your convenience. It is found in C:\MDK\Boards\Infineon\XMC Boot Kit\RTX_text. You can copy the entire contents into Main.c or type in the source.

1. Stop the program:  Exit Debug mode: 
2. Open Main.c by double-clicking on it in the Project window in µVision.
3. **Add the RTX header file:** Add these two lines near line 13 just after #include <DAVE3.h>:

```
#include "cmsis_os.h"
#include "RTE_Components.h"
```

4. **Note:** All other necessary header files are provided by #include <DAVE3.h> which is already in main.c.
5. **Add a Thread ID for the two Threads: (starting at near line 15 int main(void))**

```
osThreadId tid_phaseA;                  /* Thread id of thread: phase_a      */
osThreadId tid_phaseB;                  /* Thread id of thread: phase_b      */
```

6. **Add Thread 1: (starting at near line 18)**

```
/*----------------------------------------------------------------------------
 *      Thread 1 'phaseA': Phase A output:  Controls Port 0.0 LED
 *---------------------------------------------------------------------------*/
void phaseA (void const *arg) {
  for (;;) {
    IO004_TogglePin(IO004_Handle0);
    osDelay(250);                                /* delay 250ms  */
    }}
```

7. **Add Thread 2: (starting at near line 28 int main(void))**

```
/*----------------------------------------------------------------------------
 *      Thread 2 'phaseB': Phase B output       Controls Port 0.7 LED
 *---------------------------------------------------------------------------*/
void phaseB (void const *arg) {
  for (;;) {
    IO004_TogglePin(IO004_Handle1);
    osDelay(250);                                /* delay 250ms                */
    }}
```

8. **Define each thread: (starting at near line 37) before int main(void))**

```
osThreadDef(phaseA, osPriorityNormal, 1, 0);
osThreadDef(phaseB, osPriorityNormal, 1, 0);
```

9. **int main(void) is located at this point: (starting near line 40)**
10. **Initialize RTX: (starting near line 44) just after DAVE_Init(); inside int main(void))**

```
osKernelInitialize ();
```

11. **Create the two Threads phaseA and phaseB: (starting near line 47) before while(1)**

```
tid_phaseA = osThreadCreate(osThread(phaseA), NULL);
tid_phaseB = osThreadCreate(osThread(phaseB), NULL);
```

12. **Start the RTX Kernel: (starting near line 49) before while(1)**

```
osKernelStart ();
```

13. **while(1){ } is already located at this point: (near line 50)**
14. **Add inside the while loop near line 52:**

```
osThreadYield ();
```

15. Comment out return 0; and add a carriage return (Enter) at the very end of the file.
16. Select File/Save All or  .

Compile, Load and RUN your Program:

1. Compile the source files by clicking on the Rebuild icon.  . You can also use the Build icon beside it.

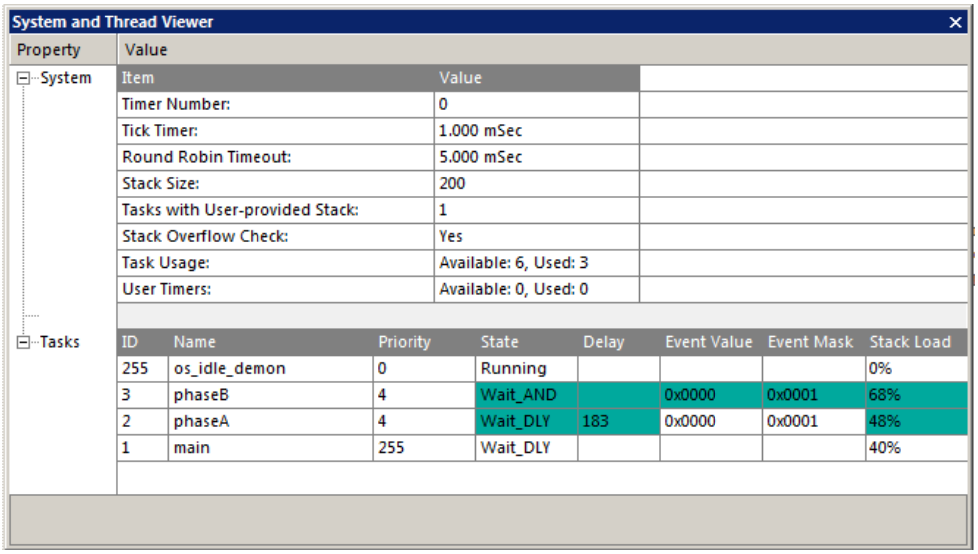
TIP: You will probably get a warning about the line Return() in main.c not reachable. You can comment this line out.

2. Program the XMC1200 Flash by clicking on the Load icon:  Progress will be indicated in the Output Window.

TIP: Since Update Target before Debugging is selected (see page 34), you can skip step 2. Entering Debug mode will automatically program the Flash if there is a new file available.

3. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
4. Click on the RUN icon.  **Note:** you stop the program with the STOP icon. 
5. The two LEDs will blink.
6. Open Debug/System and Threads Viewer and this window opens up:
7. phaseA and phaseB will take turns running proving the program is working correctly.

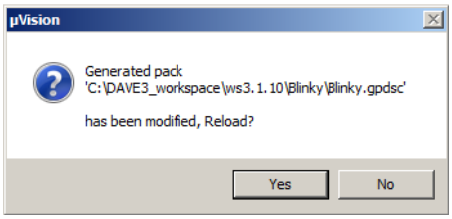
NOTE: At this writing, continuing development of the DAVE RTOS App is causing this window to be blank.



How to Update Configurations inside DAVE:

We will not actually demonstrate this. When files are modified in DAVE they can be imported into µVision as follows:

1. In µVision, stop the program.  and exit Debug mode. 
2. In DAVE, Make your modifications. You can try enabling Default Output level to High in IO004/0. Make sure you click on Enter or click in another box to enter this new setting.
3. Click on the Generate Code icon  or select DAVE/Generate Code. A new project will be generated.
4. When done, bring µVision back into focus.
5. There will one or more indications that has been changed: Click on Yes for all of these. Files modified by DAVE are imported into µVision.
6. Select File/Save All or  .



Compile, Load into Flash and RUN: 

1. Compile the source files by clicking on the Rebuild icon in µVision.
2. Enter Debug mode.  The Flash will automatically be programmed.
3. Click on the RUN icon. 
4. The changes you made in DAVE will be reflected in your updated project.
5. The source files you added are not replaced or modified.

23) PWM Controlled Blinky using DAVE

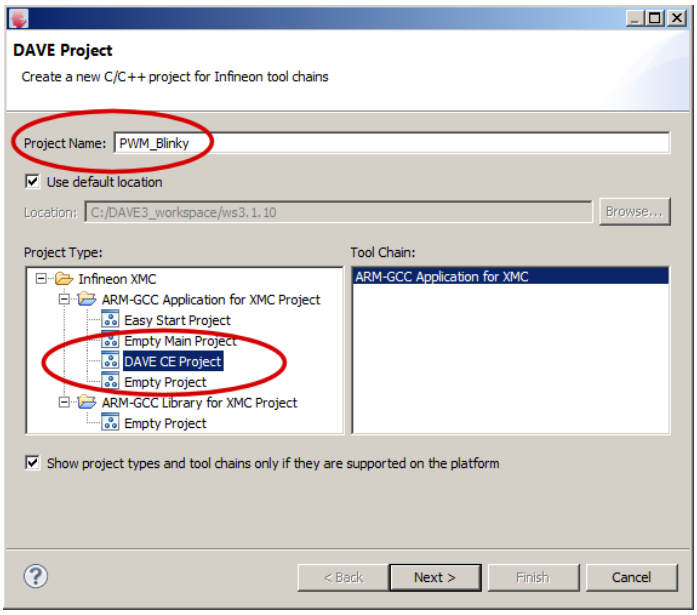
We will create a µVision project using DAVE to blink one LED on the Boot Kit board. A PWM App will be used and configured: it blinks the LED (slowly changing the brightness). Also, a Counter App will be used to count the PWM pulses. An interrupt on the count match toggles between increasing and decreasing the brightness intensity of the LED. DAVE will then create a µVision project and µVision will then import it.

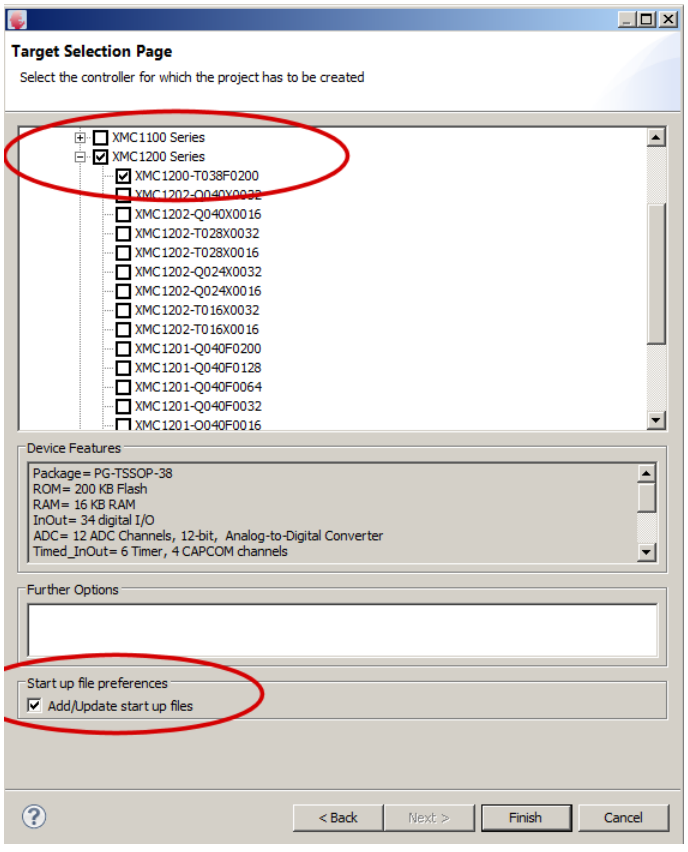
Start DAVE and create a DAVE project:

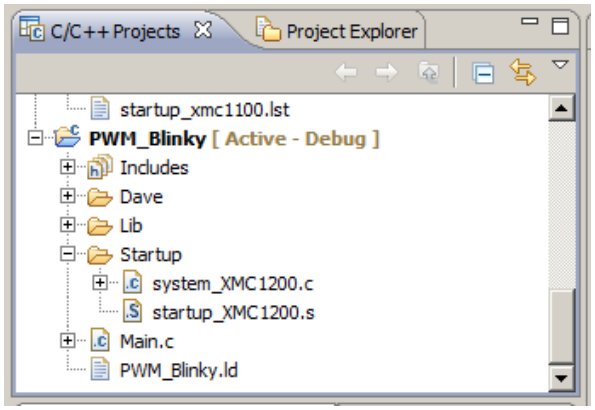
1. Start DAVE by clicking on its icon:
2. Select File/New/Dave Project and this window will open:
3. Select DAVE CE Project: Always select CE.
4. In Project Name box enter PWM_Blinky as shown here:
5. Select Next > and the Target Selection Page opens:
6. Select XMC1200-TO38F200 or the processor you are using as shown in the Target Selection window:
7. Confirm Add/Update startup files is enabled.
8. Click on Finish. DAVE will create your project.
9. The Blinky project will be displayed in the DAVE C/C++ Projects window as shown below.
10. Expand some of the elements.
11. You will see a Main.c and two CMSIS startup files.
12. Double click on Main.c, it will open in a window.
13. You can see DAVE has created a template Main.c with an int main(void) function but not much else.
14. If you open the startup files found in the Startup folder, you can see startup_XMC1200.s which contains processor initialization code and system_XMC1200.c which contains mostly clock settings.

TIP: DAVE dynamically saves all your work into the default workspace: C:\DAVE3_workspace. You can manually save your work with File/Save or Ctrl-S.

15. We will next create two PWMs to blink the LEDs.







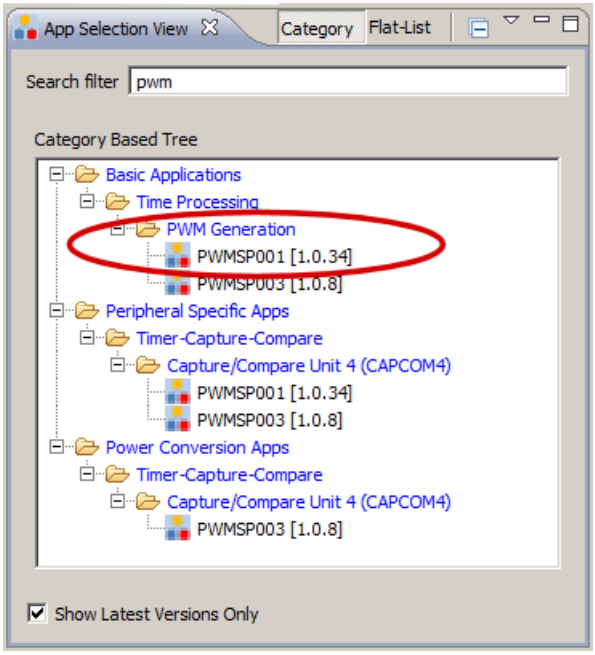
DAVE Apps:

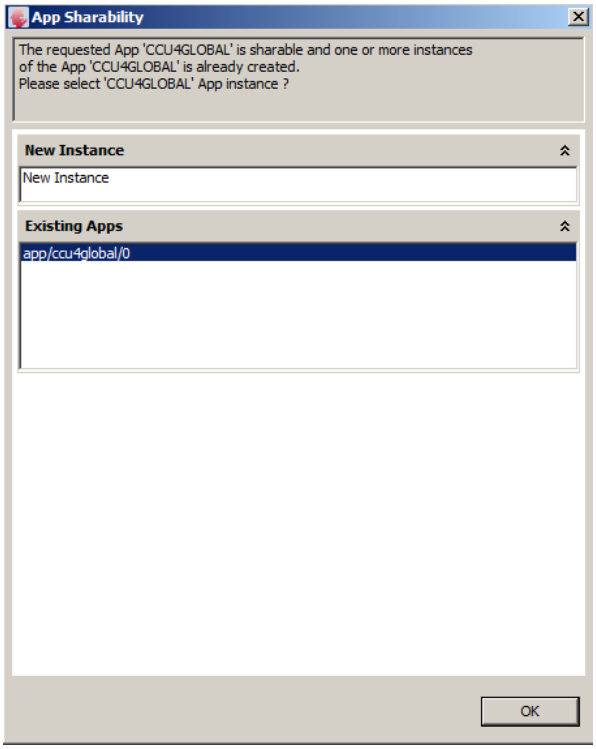
DAVE includes Apps that you can graphically configure to create programs to suit your needs. These are programs DAVE uses to create the source code files that will become part of your µVision project.

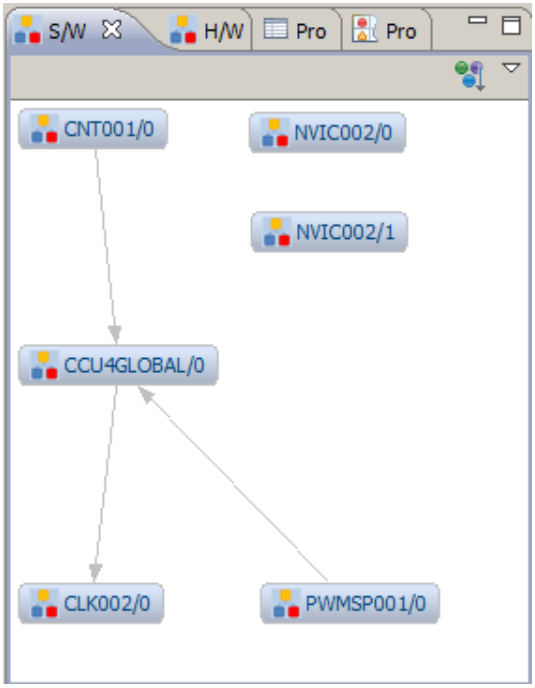
1. Apps are listed in the App Selection View.
2. Scroll through them to see your options.
3. If you hover your mouse over an App, a description of it will display.
4. We will use one instance of the App PWMSP001 to blink the LED on and off.
5. We will use one instance of the App CNT001 to count the number of PWM pulses.
6. We will use two instances of the App NVIC002 to react on interrupts on the PWMSP001 and CNT001.

Create the Apps:

1. Enter **pwm** in the Search filter as shown here:
2. Under the heading PWM Generation, double click on the App **PWMSP001** as shown here:
3. This will be added in the S/W App Connectivity View.
4. Enter **cnt** in the Search filter.
5. Under the heading Signal Capture, double click on the App **CNT001.**
6. The Apps Shareability window will open. Select app/ccu4global/0 as shown below on the right:
7. Click on OK.
8. Enter **nvic** in the Search filter.
9. Under the heading NVIC (Nested Vectored Interrupt Controller), double click on the App **NVIC002.**
10. It will be added to the S/W App Connectivity View.
11. Double click again on the same **NVIC002** App. A second instance of it will be added.
12. Your S/W App Connectivity View will be similar to the one shown below: You can move your Apps around.
13. Note there are two instances of NVIC002 (denoted by /0 and /1), a Capture/CompareUnit 4 and a CLK002/0 clock App.
14. These last two Apps are automatically inserted by DAVE.

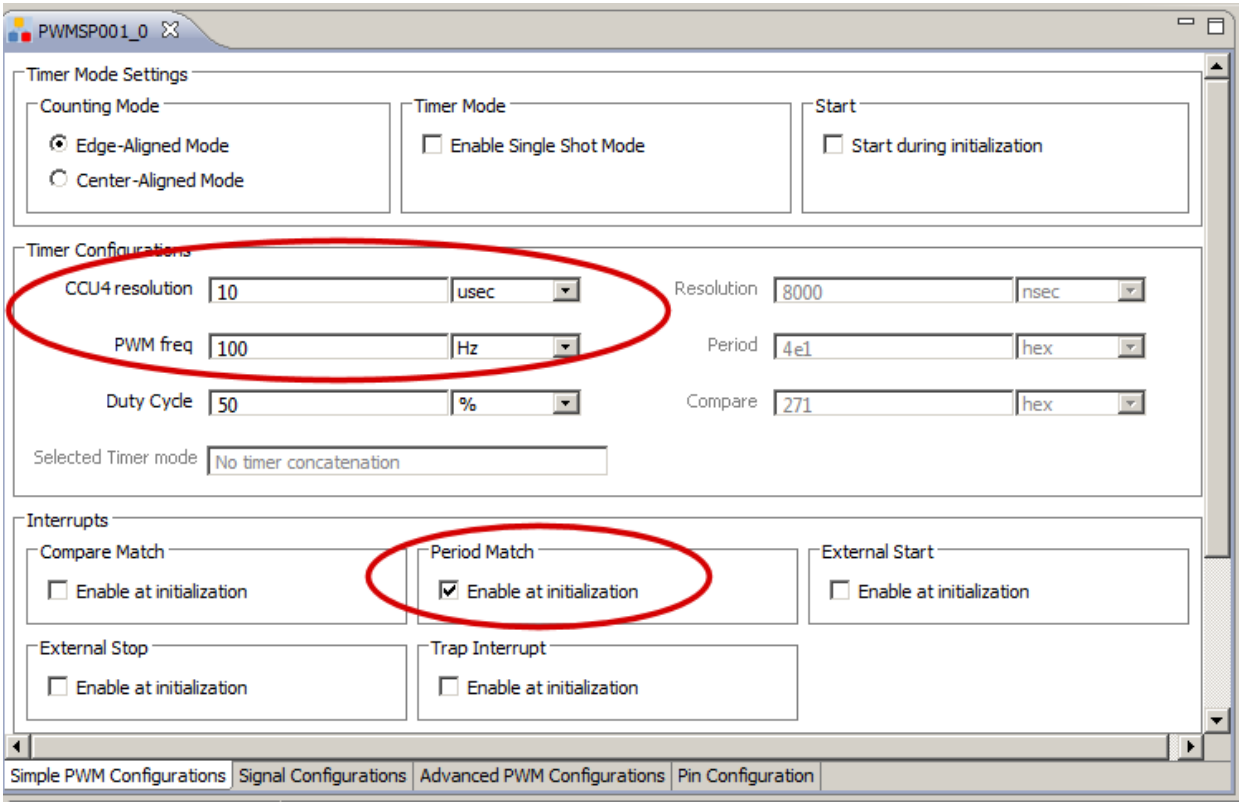






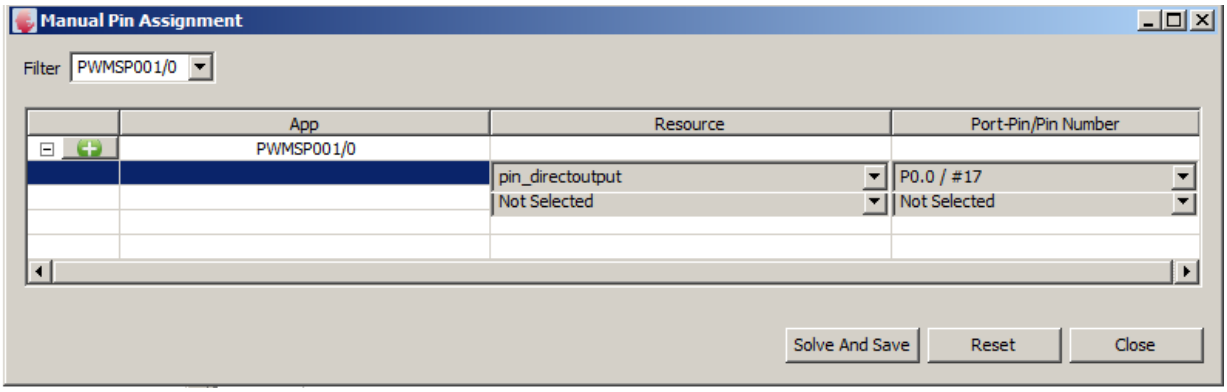
Configure PWMSP001/0:

1. Right click on PWMSP001/0 and select UIEditor. The window below opens up:
2. First, in the Timer Configuration, set CCU resolution units to usec.
3. Then set CCU4 Resolution to 10.
4. Set PWM freq to 100 Hz
5. In the Interrupts area, enable Period Match: Enable at initialization.
6. Select the Signal Configurations tab and set the Output Level to Passive High.
7. Leave all other settings at their defaults.



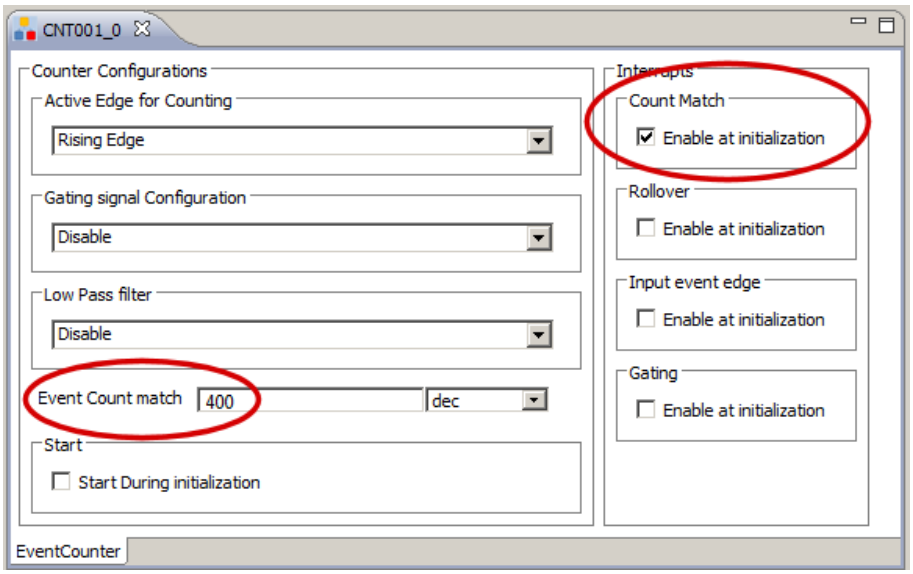
Connect PWMSP001/0 output to GPIO Port Pin P0.0:

1. Right click on PWMSP001/0 again and select Manual Pin Assignment and the next window opens up.
2. In the Resource column, select pin_directoutput and Port Pin number P0.0/ #17 as shown below:
3. Select Solve And Save and then Close to assign this pin.



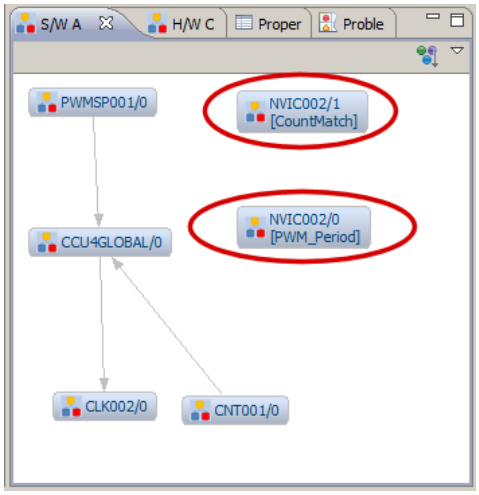
Configure CNT001:

1. Right click on the App CNT001/0 and select UIEditor. The window below opens up.
2. Set Event Count Match to 400.
3. In the Interrupts area, select Count Match to Enable at initialization as shown below:



Add User Labels to the NVIC002:

1. Right click NVIC002/0 and select Add User Label.
2. Enter PWM_Period and click OK.
3. Right click NVIC002/1 and select Add User Label.
4. Enter CountMatch and click OK.
5. Your apps will now look like this with labels attached:
   They are probably in different physical locations.

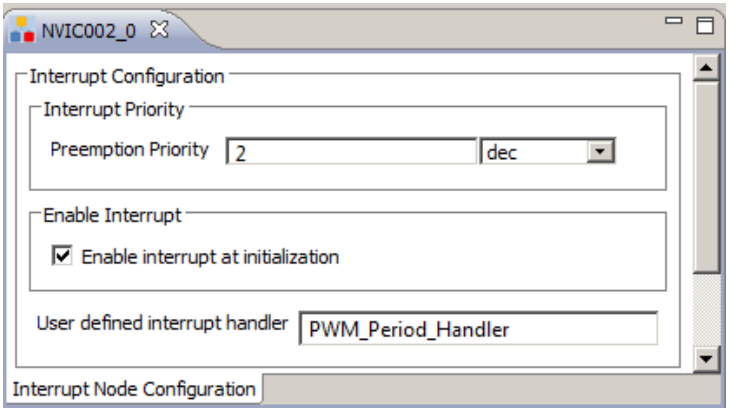


Configure NVIC002/0 (PWM_Period):

1. Right click NVIC002/0 (PWM_Period) and select UIEditor and the window below opens:
2. Set the Preemption Priority to 2.
3. Select Enable interrupt at initialization.
4. Set the User defined interrupt handler to PWM_Period_Handler.
5. Press Enter or click away to ensure this handler is accepted.

Configure NVIC002/1 (CountMatch):

1. Right click NVIC002/1 (CountMatch) and select UIEditor.
2. A similar screen opens.
3. Set the Preemption Priority to 3.
4. Select Enable interrupt at initialization
5. Set the User defined interrupt handler to CountMatch_Handler.
6. Press Enter to ensure this handler is accepted.



Configure Signal Connections:
One of the advanced features of DAVE3 is the option for the user to connect signals between different DAVE Apps. An important use for this feature is the connection of an event signal with the interrupt signal of an NVIC (interrupt) App.

PWMSP001/0 Signal Connections:

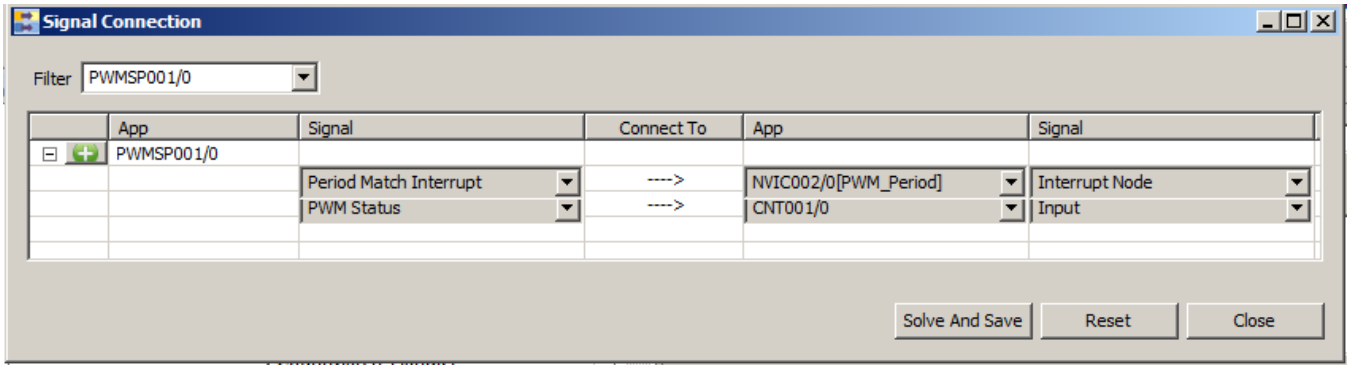
1. In the S/W App Connectivity View Right click on PWMSP001/0 and select Signal Connection. The window below opens:

Connect NVIC002/0

2. In the first Signal column, select Period Match Interrupt.
3. In the App column, select the NVIC002/0 App to connect to.
4. In the second Signal column, select Interrupt Node.
5. Click Solve and Save. Do not select Close yet. We are not quite done.

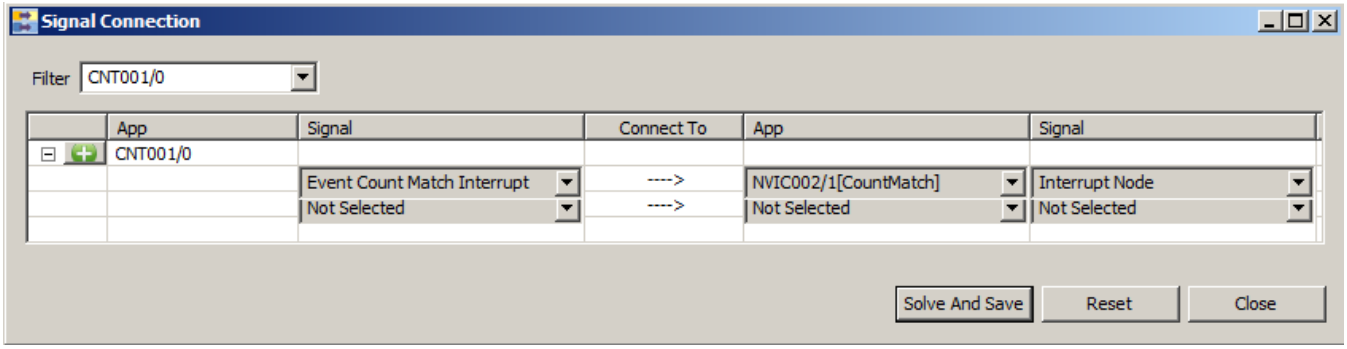
Connect CNT001/0

6. A new signal row will be created.
7. In the first Signal column, select PWM Status.
8. In the App column, select CNT001/0 to connect to.
9. In the second Signal column, select Input as the Signal.
10. Click Solve and Save. Your window will like the one below. If not, please make any changes and Solve and Save.
11. Click Close.



CNT001/0 Signal Connections:

1. In the S/W App Connectivity View, right click on CNT001/0 App and select Signal Connection. The window below opens.
2. In the first Signal column, select Event Count Match Interrupt.
3. In the App column, select NVIC002/1 (Count Match) to connect to.
4. In the second Signal column, select Interrupt Node as the Signal.
5. Click Solve and Save. Your window will look like the one below. If not, please edit and Solve and Save.
6. Click Close.



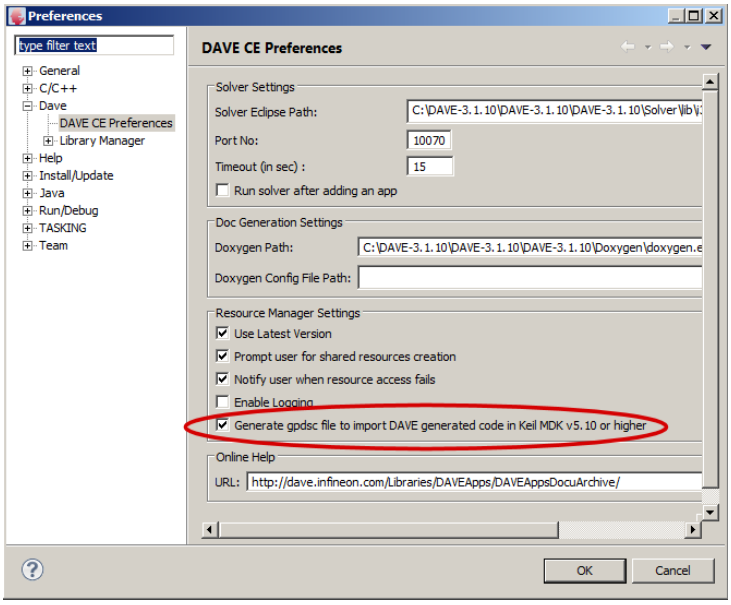
At this point, all the Apps we need for the PWM_Blinky project are now configured.

Generator Package Description (gpdsc) File:
DAVE produces a complete µVision project and in addition a gpdsc file. This file is used to start, import and configure the appropriate files into µVision. DAVE must be configured to generate this gpdsc file.

If you did this task on page 26, you do not need to repeat it. Jump to **Generate the Code** below:

Configure DAVE to create a gpdsc file: (you only have to do this once in DAVE for all your projects)

1. In the DAVE main window, select Window/Preferences and this window opens up:
2. Expand Dave and select DAVE CE Preferences as shown here:
3. Select Generate gpdsc file… as shown:
4. Click on OK.

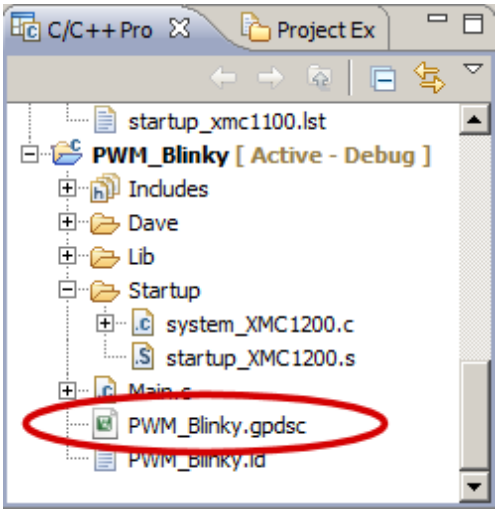


Generate the Code:

5. Click on the Generate Code icon  or select DAVE/Generate Code.
6. The appropriate files will be created and a progress bar will be displayed.
7. The C/C++ Projects window will display the files in this project.
8. PWM_Blinky.gpdsc will be visible in the window as shown below right:
9. You are ready to import this into µVision.

Import into µVision:

1. Close any instances of µVision. See the TIP below.
2. Double click on PWM_Blinky.gpdsc and a new instance of µVision will be started.



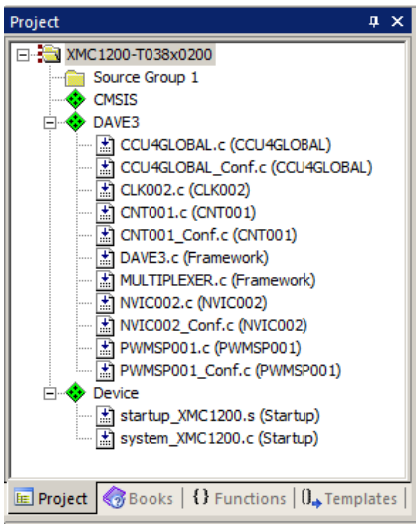
TIP: If you do not want another instance of µVision running, close any running instance(s) of µVision before double clicking on the gpdsc file. Running multiple instances of µVision is permitted but a conflict with the J-Link Lite might arise in this case.

TIP: At this time you can shut DAVE down if you prefer. It is not needed again unless you want to add or modify any settings that must be done in DAVE.
Otherwise, µVision can work stand alone with the files DAVE created for it.

On the next page, we will configure µVision, add some source code to PWM_Blinky.c, build the project, program it to Flash and run it.

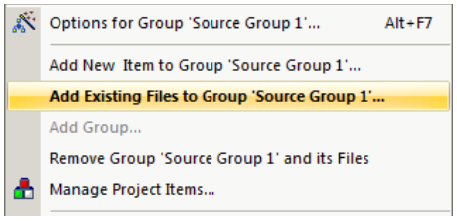
Configure µVision:

After double clicking on PWM_Blinky.gpdsc, µVision will be started. The source files DAVE created are listed in the Project window in µVision as shown here:



This is almost complete: you must now add a file containing a main() function. You can add your own, or use the one DAVE created. We will use the one DAVE created. It is called Main.c.

4. Right click on Source Group 1 and select Add Existing Files to Group ….:



.

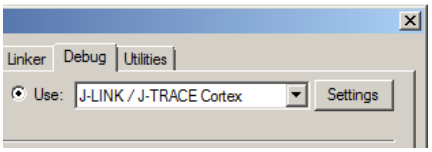
5. In the Add Files window, highlight Main.c and select Add and then Close.
6. Main.c will now be added to Source Group 1.

TIP: If you create and add your own new file, you can select Add New Item to Group… and some templates will be offered that you can choose from.

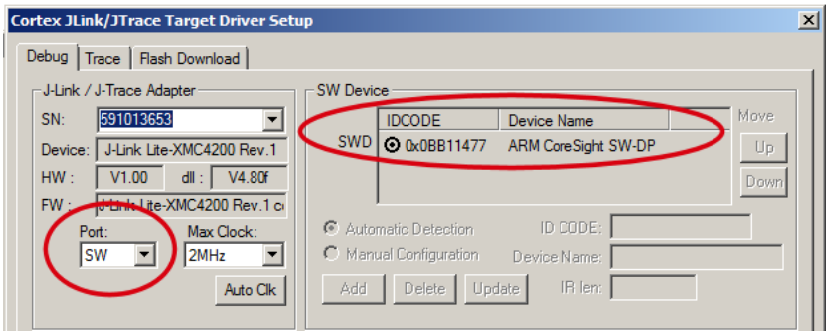
Configure the Debug Adapter and Flash programming:

14. In µVision, select the Options for Target Options icon: Options  or ALT-F7 and select the Target tab.
15. Select Use MicroLIB to compile for smaller program size: 

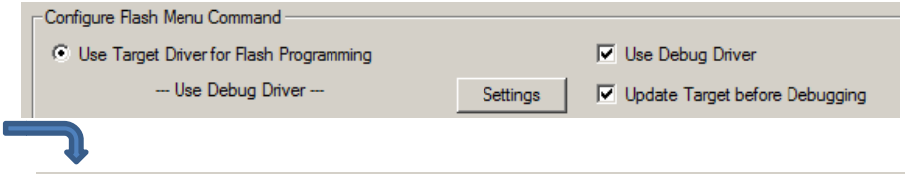
16. Select the Debug tab.
17. Select J-Link / J-Trace Cortex as shown here:
18. Select Settings: on the right side of this window.
19. Select SW and not JTAG as shown here in the Port: box:



20. Confirm you see an IDCODE and Device Name in SW Device box:
If this box is blank, or an error is displayed, ***you must fix*** this before you can continue. Check your cables and make your Boot Kit board is plugged in. Ensure you have the correct J-Link USB drivers installed.



21. The SN: box indicates µVision is connected to the J-Link Lite. The SW Device box indicates µVision is also connected to the XMC1200 processor CoreSight debug module.
22. Click on OK to leave this window.
23. Select the Utilities tab: Select Use Debug Driver as shown here:
24. Select Settings and then select Add:
25. A list appears. Select XMC 1200 XMC 200 kB Flash and then Add again. The Flash algorithm will display as shown here:



26. Click OK twice to close and return to µVision main window.
27. Select File/Save All or  .

At this point, you have now configured µVision. You now must add some files to main.c on the next page.

Add Application Code to Main.c:

Under Source Group 1, double click on Main.c in the Project window to display it. The code below is contained in the file Main.c or PWM.txt in C:\MDK\Boards\Infineon\XMC1200 Boot Kit\PWM_Blinky.

1. **Add the following variable declarations below line 13 after #include <DAVE3.h>:**

```
#include "XMC1200.h"
float LED_duty = 0;

enum bright {up, down};
enum bright LED_bright = up;

uint32_t PWMerrorcount = 0;
uint32_t CNTerrorcount = 0;
uint32_t CountVal;

PWMSP001_TimerRegsType TimerVal;
```

2. **Add these two handler functions that have been declared for the DAVE Apps near line 27 before main(void):**

```
void PWM_Period_Handler(void) {

    status_t status;

    switch (LED_bright) {
        case up: {if (LED_duty > 99)
                    LED_duty = 0;
                    LED_duty++;
                 }
        break;
        case down: {if (LED_duty < 1)
                    LED_duty = 100;
                    LED_duty--;
                 }
    }
    status = PWMSP001_SetDutyCycle(&PWMSP001_Handle0, LED_duty);
    if (status != DAVEApp_SUCCESS) PWMerrorcount++;
}

void CountMatch_Handler(void) {

    status_t status;

    switch (LED_bright) {
        case up: LED_bright = down;
                 LED_duty = 100;
        break;
        case down: LED_bright = up;
                   LED_duty = 0;
    }
    status = CNT001_Stop(&CNT001_Handle0);
    if (status != DAVEApp_SUCCESS) CNTerrorcount++;
    status = CNT001_ResetCounter(&CNT001_Handle0);
    if (status != DAVEApp_SUCCESS) CNTerrorcount++;
    status = CNT001_Start(&CNT001_Handle0);
    if (status != DAVEApp_SUCCESS) CNTerrorcount++;
}
```

3. **Uncomment the declaration of the variable `status` in main.**
4. **Add these lines after DAVE_Init(); which is near line 70:**

```
    status = PWMSP001_Start(&PWMSP001_Handle0);
    if (status != DAVEApp_SUCCESS) PWMerrorcount++;
    status = CNT001_Start(&CNT001_Handle0);
    if (status != DAVEApp_SUCCESS) CNTerrorcount++;
```

5. **Add these lines inside the while loop:**

```
status = PWMSP001_GetTimerRegsVal((PWMSP001_HandleType*)&PWMSP001_Handle0, &TimerVal);
if (status != DAVEApp_SUCCESS) PWMerrorcount++;
status = CNT001_GetEvtCountValue((CNT001_HandleType*)&CNT001_Handle0, &CountVal);
if (status != DAVEApp_SUCCESS) CNTerrorcount++;
```

Configure µVision Build Options:

1. Select Target Options or ALT-F7. Select the Target tab.
2. Select Use MicroLIB – this will result in smaller programs. Enter 32 in Xtal (MHz).
3. Select the C/C++ tab and in the Misc Controls box enter: **--diag_suppress=188**

   This entry has a double dash preceding diag: - - diag. This will prevent some immaterial warnings being displayed.

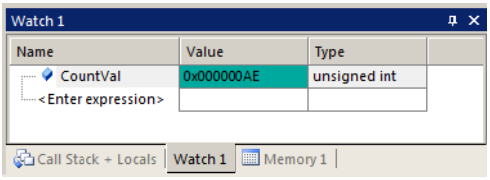
7. Click on OK. Select File/Save All or .
4. In Main.c, the last line return 0; will cause an immaterial warning. Either ignore this or comment this line out.

Build, Flash and Run the program !

1. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
2. Program the XMC Flash by clicking on the Load icon: Progress will be indicated in the Output Window.
3. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
   Note: µVision is configured to program the Flash when entering Debug mode. You do not have to use Load.
4. Click on the RUN icon. The LED at P0.0 will change in brightness..
   Note: you stop the program with the STOP icon.
5. You have successfully created a program using DAVE and µVision to modify the brightness of a LED.

Examining Some Variables:

1. With program till running, in Main.c, find the global variable CountVal near line 21.
2. Right clink on CountVal and select Add CountVal to… and select Watch 1.
3. CountVal will appear in Watch 1 and will be updating (if the program is running) as shown here:
4. Add LED_bright (near line 17) and TimerVal (near line 23) in the same way to Watch 1.
5. These variables will be now displayed and updated as shown here:

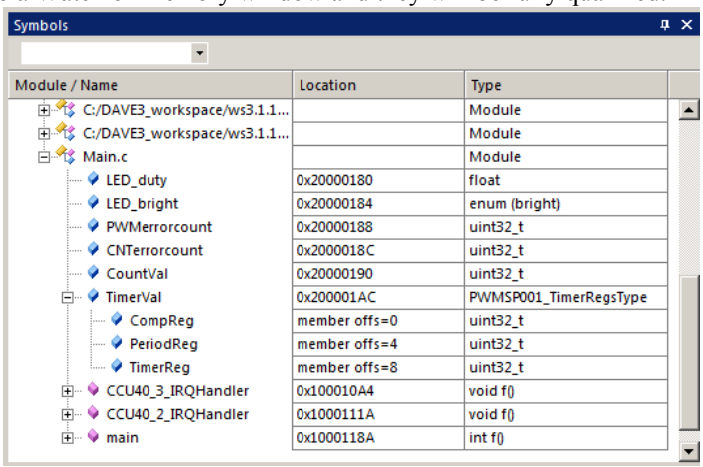


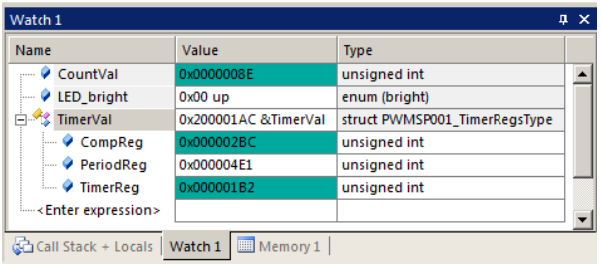
TIP: These variables are updated in real time. Nearly always, no CPU cycles are stolen to read these memory locations. You are not able to view local variables while the program is running.

Symbol Table:

1. Select View/Symbols Window in the main menu.

This window below opens. Expand Main.c and its variables are displayed with their locations. You can copy variables from here to a Watch or Memory window and they will be fully qualified.





This is the end of the PWM_Blinky tutorial.

23) Document Resources: See www.keil.com/infineon

Books:

1. ***NEW!*** **Getting Started MDK 5:** Obtain this free book here: www.keil.com/mdk5/.
2. There is a good selection of books available on ARM processors. A good list of books on ARM processors is found at www.arm.com/university by selecting "Teaching Resources". You can also select ARM Related Books but make sure to also select the "Books suited for Academia" tab to see the full selection.
3. µVision contains a window titled Books. Many documents including data sheets are located there.
4. **A list of resources is located at:** www.arm.com/products/processors/cortex-m/index.php
   Click on the Resources tab. Or search for "Cortex-M3" on www.arm.com and click on the Resources tab.
5. The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
6. The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
7. Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano.

Application Notes:

1. ***NEW!*** ARM Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/pr/article/1262.htm
2. Using DAVE with µVision: **µVision is DAVE compatible.** www.keil.com/appnotes/files/apnt_258.pdf
8. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
9. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
10. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
11. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
12. Using µVision with CodeSourcery GNU www.keil.com/appnotes/docs/apnt_199.asp
13. RTX CMSIS-RTOS in MDK 5 C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.1.0\CMSIS_RTX
14. Download RTX CMSIS-RTX www.keil.com/demo/eval/rtx.htm ***and*** www.arm.com/cmsis
15. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
16. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
17. Cortex Debug Connectors: www.arm.com and search for cortex_debug_connectors.pdf **or** www.keil.com/coresight/coresight-connectors
18. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp

Keil Tutorials for Infineon Boards: www.keil.com/infineon

1. XMC1200 Boot Kit (this document) www.keil.com/appnotes/docs/apnt_263.asp
2. XMC1100 2Go Kit: www.keil.com/appnotes/docs/apnt_260.asp
3. XMC4500 Relax Kit: *coming…* See www.keil.com/infineon
4. XMC4500 Application Board Kit: *coming…* See www.keil.com/infineon
5. Hitex XMC-HiLight XMC4500 board: www.keil.com/appnotes/docs/apnt_231.asp

ARM Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content

Infineon Community Forum: www.infineonforums.com/

ARM University program: www.arm.com/university. Email: university@arm.com

ARM Accredited Engineer Program: www.arm.com/aae

mbed™: http://mbed.org

For comments or corrections on this document please email bob.boys@arm.com

For more information on the ARM CMSIS standard: www.arm.com/cmsis

24) Keil Products and Contact Information: See www.keil.com/infineon

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite (Evaluation version) - $0
- MDK XMC1000 - $0
- ***NEW !!*** MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit)
- MDK-Standard (unlimited compile and debug code and data size)
- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries and Graphic User Interface (GUI))
- ***NEW !!*** ARM Compiler Qualification Kit: for Safety Certification Applications

USB-JTAG adapter (for Flash programming too)

- ULINK2 - *(ULINK2 and ME - SWV only – no ETM)*
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINK*pro* – Faster operation and Flash programming, Cortex-M*x* SWV & ETM trace.
- ***NEW !!*** ULINK*pro* D – Faster operation and Flash programming, Cortex-M*x* SWV, no ETM trace.

Contact sales.us@keil.com 800-348-8051 for USA prices.

Contact sales.intl@keil.com +49 89/456040-20 for pricing in other countries.

- ***For special promotional or quantity pricing and offers, please contact Keil Sales.***

The Keil RTX RTOS is now provided under a BSD type license. This makes it free.

All versions, including MDK-Lite, includes Keil RTX RTOS *with source code !*

Keil includes free DSP libraries for the Cortex-M0, Cortex-M3 and Cortex-M4.

Call Keil Sales for details on current pricing, specials and quantity discounts. Sales can also provide advice about the various tools options available to you. They will help you find various labs and appnotes that are useful.

All products are available from stock.

All products include Technical Support for 1 year. This is easily renewed.

Call Keil Sales for special university pricing. Go to www.arm.com/university to view various programs and resources.

Keil supports many other Infineon processors including 8051 and C166 series processors. See the Keil Device Database® on www.keil.com/dd for the complete list of Infineon support. This information is also included in MDK.

For more information:

Keil Sales In USA: sales.us@keil.com or 800-348-8051. Outside the US: sales.intl@keil.com or +49 89/456040-20

Keil Technical Support in USA: support.us@keil.com or 800-348-8051. Outside the US: support.intl@keil.com.

For comments or corrections please email bob.boys@arm.com.

For the latest version of this document, go to www.keil.com/infineon and for more Infineon specific information.

CMSIS documentation: www.arm.com/cmsis

CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

Cortex™
Intelligent Processors by ARM®

ARM® KEIL®
Microcontroller Tools