

The latest version of this document is here: [www.keil.com/appnotes/docs/apnt\\_260.asp](http://www.keil.com/appnotes/docs/apnt_260.asp)

## Introduction:

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.

You can use MDK Lite with this tutorial. No license is needed. You can compile and link up to 32K code and data. MDK includes the full version of Keil RTX™ RTOS. RTX source code is included with a BSD or Apache 2.0 license.

**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](http://www.keil.com/arm/mdk) for more information.

## 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 toolchain. MDK is turn-key "out-of-the-box".
2. µVision IDE provides all Arm CoreSight® debug features to improve your productivity. MDK quickly pays for itself. MDK is a good investment.
3. Arm Compiler 5 and Arm Compiler 6 (LLVM) are included. GCC is supported. See <https://developer.arm.com>
4. **DAVE compatible.** See [www.keil.com/appnotes/files/apnt\\_258.pdf](http://www.keil.com/appnotes/files/apnt_258.pdf)
5. RTX RTOS available with sources. Apache 2.0 license. [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)
6. Dynamic Syntax checking on C/C++ source lines.
7. **Compiler Safety Certification Kit:** [www.keil.com/safety/](http://www.keil.com/safety/)
8. TÜV certified. SIL3 (IEC 61508) and ASILD (ISO 26262).
9. MISRA C/C++ support using PC-Lint. [www.gimpel.com](http://www.gimpel.com)
10. **Keil Middleware:** Network, USB, Flash File and Graphics.
11. **NEW! Event Recorder for Keil Middleware, RTOS and User programs.** See page 14 for printf.
12. Adapters: Keil ULINK™2, ULINK-ME, ULINKplus, ULINKpro, CMSIS-DAP and Segger J-Link.
13. Affordable perpetual and term licensing with support. Contact Keil sales for options. [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com)
14. Keil Technical Support is included for one year and is renewable. This helps you get your project completed faster.
15. ULINKplus Power Analysis: [www.keil.com/mdk5/ulink/ulinkplus/](http://www.keil.com/mdk5/ulink/ulinkplus/) Contact Keil sales for details.
16. Keil also supports Infineon 8051 and C166 processors. See [www.keil.com/Infineon](http://www.keil.com/Infineon) for the complete list.
17. Kernel Awareness windows for RTX is updated in real-time. FreeRTOS is now supported by µVision.

### This document details these features:

1. Real-time Read update for Watch, Memory, Peripherals and RTX Threads windows. 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 two Watchpoints (also called Access Breaks).
3. printf without using any hardware resources such as a UART.
4. System and Thread Viewer: a kernel awareness program for RTX RTOS that updates while the program is running.
5. A DSP example with variables updated in real-time as the program is running using a Watch window.
6. How to create your own µVision project without DAVE. How to easily add RTX RTOS to any project.
7. A Blinky example created using Infineon DAVE.



## Index:

### Part A: Obtaining and Installing MDK and the Examples:

1. Infineon Boards and Arm Processors Supported by Keil MDK 5:	3
2. Installing Keil MDK and Examples:	4
3. $\mu$ Vision Software Packs Download and Install Process:	5
4. Other Features of Software Packs:	6
5. Software Pack Maintenance Notes:	7
6. Testing the J-Link Lite Connection to the XMC1100:	7
7. Installing the J-Link Lite USB Drivers (if necessary)	7
8. CoreSight™ Definitions:	8

### Part B: Stand-alone Project Examples:

1. Blinky Example Program using the XMC 2Go:	9
2. Hardware Breakpoints:	9
3. Call Stack + Locals Window	10
4. Watch and Memory Windows:	11
5. System Viewer (SV):	12
6. Watchpoints Breakpoints (Access): Conditional Breakpoints	13
7. <b>NEW !</b> <i>printf</i> using Event Recorder:	14
8. RTX_Blinky example program with Keil RTX RTOS:	15
9. RTX Kernel Awareness:	16
10. DSP SINE example using ARM CMSIS-DSP Libraries:	16
11. Creating Your Own MDK 5 project from scratch:	20
12. Creating Your Own <b>RTX</b> MDK 5 project from scratch:	23

### Part C: Projects created with Infineon DAVE:

1. Blinky:	26
------------	----


### Appendix:

1. Document Resources:	33
2. Keil Products and contact information:	34

## 1) Infineon boards supported:

MDK supports, with examples, other evaluation boards from Infineon using ARM processors. These include XMC4500 Relax, XMC4500 Application Kit, XMC1200 Boot Kit and XMC4500 HiLight. See [www.keil.com/Infineon](http://www.keil.com/Infineon). Custom boards are easily supported with Software Packs.

Pack Installer is used to download Software Packs that contain header, examples, Flash programming files, documents and much more. You can create your own Pack files for confidential projects as they are XML based.

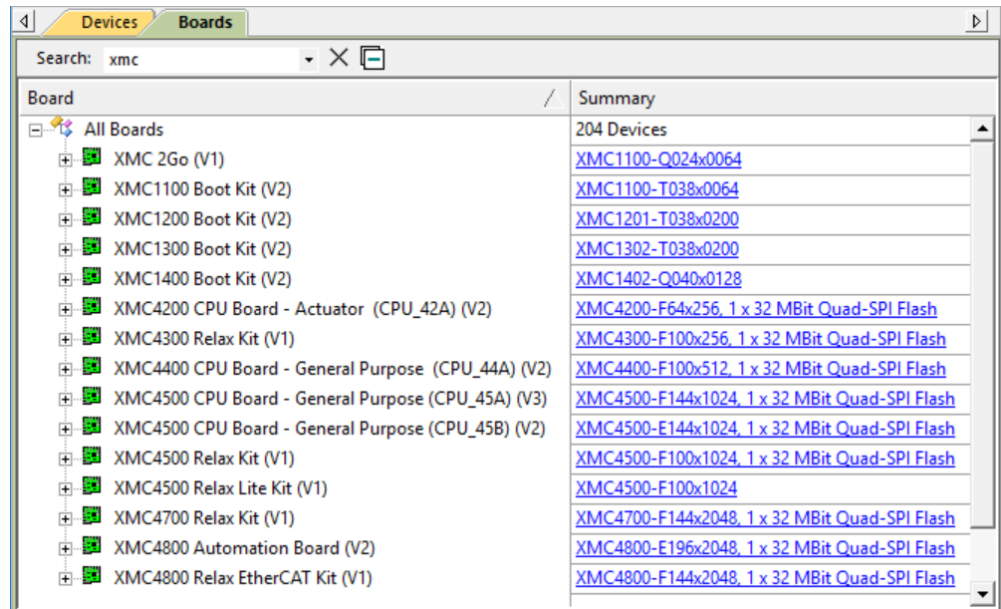
Pack Installer is a utility inside µVision and is invoked by clicking on this icon: 

A Pack is an ordinary .zip file with the filename extension changed to .pack.

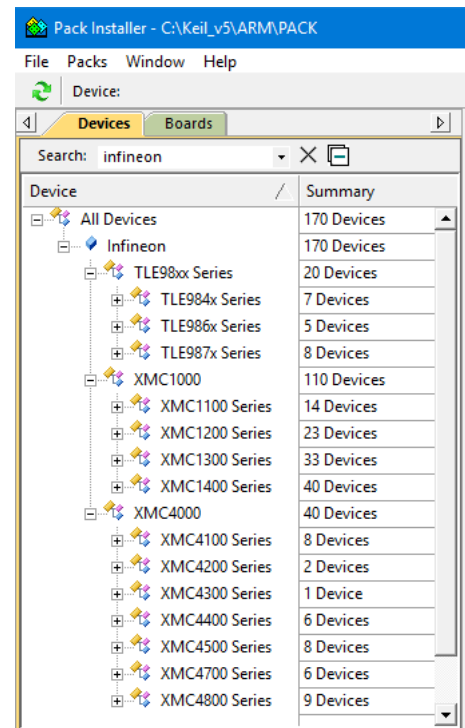
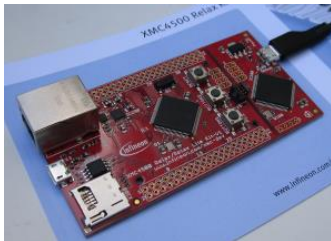
This is the current list of Infineon Arm boards supported:

Click on the Devices tab and enter Infineon. You will see all the supported list as shown on the bottom right.

You do not have to wait for the latest release of MDK to get the latest software for your processor. This is available separately via Pack Installer.



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



## Part A: Obtaining and Installing MDK and the Examples:

### 2) Keil Software Download and Installation:

1. Download MDK 5.25 or later from the Keil website. [www.keil.com/mdk5/install](http://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:\00MDK\ 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, DSP5 and RTX\_Blinky5 are available on the web where you got this document.
6. You do not need any external debug adapters: just the 2Go board, a USB cable and MDK installed on your PC.



### 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. You can obtain a one-time 7 day license in File/License Management. Contact Keil Sales if you need an extended license for evaluation.

### The on-board J-Link Lite Debug Adapter:

The on-board J-Link Lite adapter is used exclusively in this lab. J-Link USB drivers must be manually installed. Page 5 contains a test for the J-Link Lite drivers as well as instructions on installing the USB drivers.

### Example Programs:

The Blinky and RTX\_Blinky example programs are provided in the Infineon Software Pack. The other examples needed are found with this document at [www.keil.com/appnotes/docs/apnt\\_260.asp](http://www.keil.com/appnotes/docs/apnt_260.asp). The next page describes where to install these examples.

**Getting Started MDK 5:** Obtain this useful book here: [www.keil.com/mdk5/](http://www.keil.com/mdk5/).

---

### More Information and Keil Contacts:

For more information about Keil support for Infineon products please visit [www.keil.com/infineon](http://www.keil.com/infineon)

**ARM Community Forums:** [www.keil.com/forum](http://www.keil.com/forum) and <https://developer.arm.com>

**Infineon Community Forum:** [www.infineonforums.com/](http://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  $\mu$ Vision by double-clicking on the gpdsc file. No plug-ins for  $\mu$ Vision 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  $\mu$ Vision. 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. RTX has either a BSD or Apache 2.0 license.  $\mu$ Vision now supports FreeRTOS.




#### **Documentation and Help:**

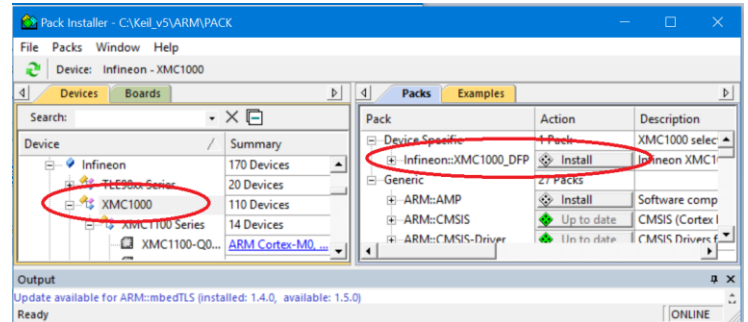
DAVE tutorials are available on [www.infineon.com/dave](http://www.infineon.com/dave).

Using DAVE with MDK 5: [www.keil.com/appnotes/files/apnt\\_258.pdf](http://www.keil.com/appnotes/files/apnt_258.pdf)

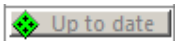
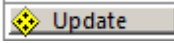
### 3) µVision Software Packs Download and Install Process:

#### 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 Devices tab: Under the Devices tab, Select Infineon XMC1100 as shown below: This will filter the list under the Packs tab.
5. Note “ONLINE” is displayed at the bottom right. If “OFFLINE” is displayed, connect to the Internet.
6. If there are no entries shown because you were not connected to the Internet when Pack Installer was opened, select Packs/Check for Updates or to refresh once you have connected to the Internet. 




#### 2) Install The XMC1000 Software Pack:

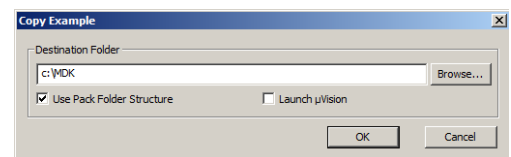
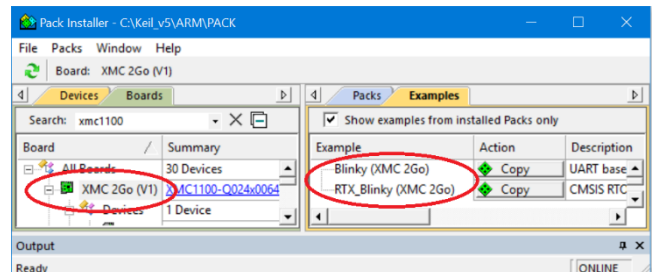
1. Select the Packs tab.
2. Select Infineon::XMC1000\_DFP and click on Install. This Software Pack will download and install to C:\Keil\_v5\ARM\Pack\Infineon\XMC1000\ by default. This download can take two to four minutes.
3. A Software Pack status is indicated by one of these icons:  or 


**TIP:** If you click on the Boards tab, you can select the board you are using and only relevant Packs are displayed.

#### 3) Install the Blinky and RTX\_Blinky Examples:


1. Select the Boards tab. Select XMC 2Go (V1):
2. Select the Examples tab.
3. Select Blinky and RTX\_Blinky (XMC 2Go):
4. Select Copy  as shown here:
5. This window opens up: Select Use Pack Folder Structure. Unselect Launch µVision as shown below
6. Type in C:\00MDK. Click OK to copy the examples.
7. They are copied C:\00MDK\Boards\Infineon\XMC\_2Go\.. Pack Installer creates the rest of the directory tree after C:\00MDK\.

**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:\00MDK\ in this tutorial. You can use any directory you prefer.

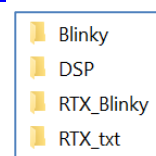


8. Close the Packs Installer. You can open it any time by clicking on its icon. 
9. Select Yes when Reload Packs box is displayed. This updates the µVision Software Pack database.

#### 4) Install the DSP5 examples from Keil.com:

1. Obtain the example software zip file from [www.keil.com/appnotes/docs/apnt\\_260.asp](http://www.keil.com/appnotes/docs/apnt_260.asp).
2. Unzip this file into the directory C:\00MDK\Boards\Infineon\XMC\_2Go\.
3. The DSP folder will be added as shown: 

You now have installed µVision, XMC1000 support files including examples.




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

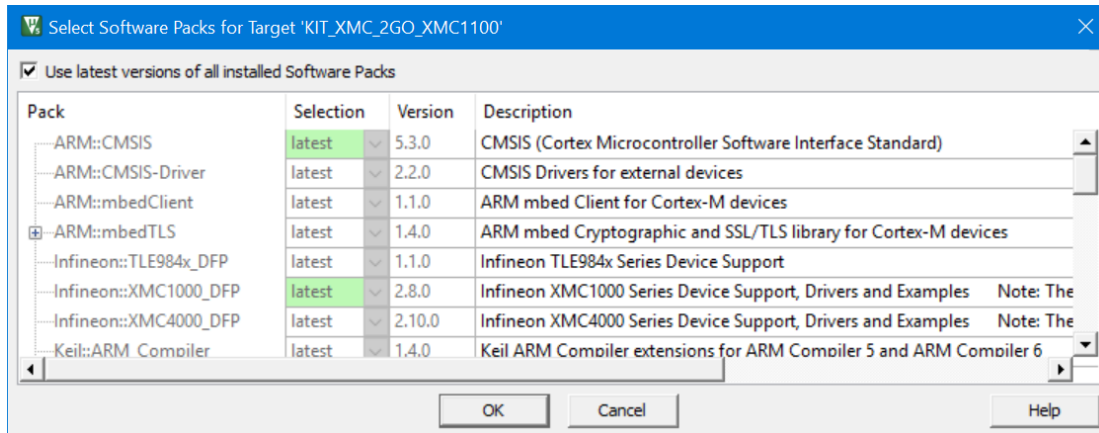


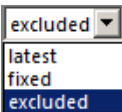
## 4) Other features of Software Packs:

### 1) Select Software Pack Version:


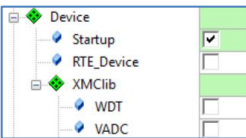
This µVision utility provides the ability to choose among the various software pack versions installed in your computer.

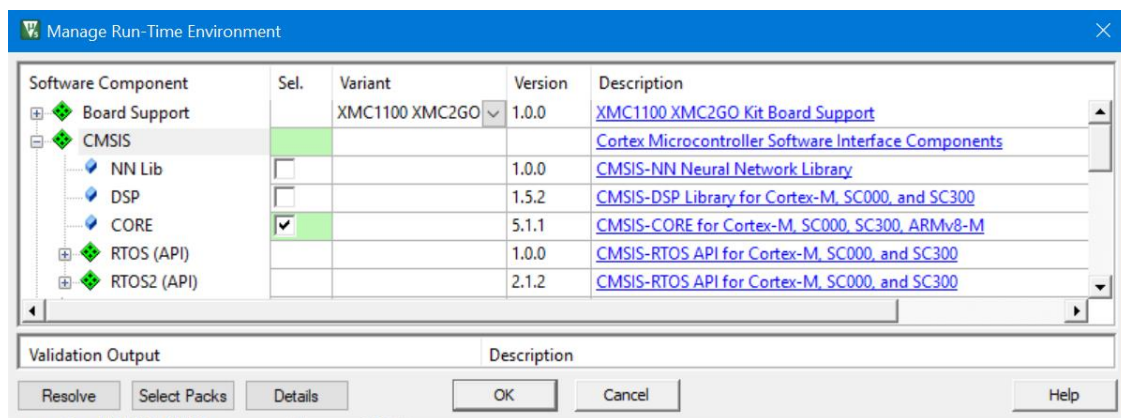
1. Open the Select Software Pack by clicking on its icon: 
2. This window opens up. Note **Use latest versions ...** is selected. The latest version of the Pack will be used.
3. Unselect this setting and the window changes to allow various versions of Packs to be selected.



4. Note various options are visible as shown here: 
5. Select excluded and see the options as shown:
6. Select Use latest versions... Do not make any changes.
7. Click Cancel to close this window to make sure no changes are made.

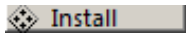
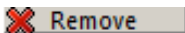
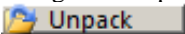
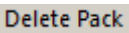
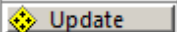
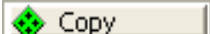
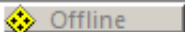
### 2) Manage Run-Time Environment (MRTE):

1. Select Project/Open Project.
2. Open the project: C:\00MDK\Boards\Infineon\XMC\_2Go\Blinky.uvprojx.
3. Click on the Manage Run-Time Environment (MRTE) icon:  The window below opens:
4. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project. You can select the XMCLib files under Device: 
5. Do not make any changes. Click Cancel to close this window.



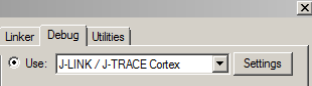


**TIP:** µVision icon meanings are found here: [www.keil.com/support/man/docs/uv4/uv4\\_ca\\_filegrp\\_att.htm](http://www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm)

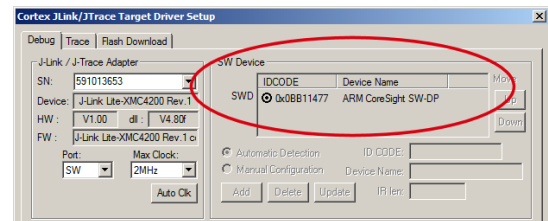
## 5) Software Pack Maintenance Notes:

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 can be locally downloaded on your computer. 

## 6) Testing the J-Link Lite Connection:

1. Start  if it is not already running. Connect the 2Go board to your PC with a USB cable.
2. Select Project/Open Project.
3. Select the Blinky project C:\00MDK\Boards\Infineon\XMC\_2Go\Blinky.uvprojx.
4. Select Target Options  or ALT-F7 and select the Debug tab:
5. Confirm J-LINK / J-TRACE Cortex is displayed as shown: 
6. Click on Settings: and the window below opens up: If an ICODE and Device name is displayed, J-Link Lite is working. Your J-Link USB drivers are installed properly and µVision is connected to the CoreSight debug module.
7. ***You can continue with this tutorial on the next page.*** Click on OK twice to return to the µVision main menu.
8. If nothing or an error is displayed in this SW Device box, this **must** be corrected before you can continue. See the next step: **8) Installing the J-Link USB Drivers:**
9. A number in the SN: box means µVision is successfully connected to the J-Link Lite adapter.
10. When the problem is corrected, click OK twice to continue.

**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 XMC1100 will not work with JTAG selected, only SW. But, this is a useful way to refresh the SW setting.



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

You must manually install the J-Link USB drivers for µVision. MDK contains the correct drivers.

1. Unplug the 2Go board.
2. With Windows Explorer, navigate to C:\Keil\_v5\ARM\Segger\USBDriver:
3. Click the appropriate executable to install the J-Link drivers. Try InstDrivers.exe
4. No indication is returned indicating a successful install.
5. Plug in the 2Go board and it will enumerate normally.
6. Run the test above again to confirm everything is operating properly.
7. If µVision suggests to upload the latest firmware to the J-Link, select or allow this.


To use DaVE: [www.keil.com/appnotes/files/apnt\\_258.pdf](http://www.keil.com/appnotes/files/apnt_258.pdf)

## 8) CoreSight™ Definitions: It is useful to have a basic understanding of these terms:

Cortex-M0 and Cortex-M0+ may have only features 2) and 4) plus 11), 12) and 13) implemented. Cortex-M3, Cortex-M4 and Cortex-M7 can have all features listed implemented. MTB is normally found on Cortex-M0+. It is possible some processors have all features except ETM Instruction trace and the trace port. Consult your specific datasheet.

1. **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
2. **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except Boundary Scan is not possible. SWD is referenced as SW in the  $\mu$ Vision Cortex-M Target Driver Setup. The SWJ box must be selected in ULINK2/ME or ULINKpro. Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDO shares the same pin as SWO. The SWV data normally comes out the SWO pin or Trace Port.
3. JTAG and SWD are functionally equivalent. The signals and protocols are not directly compatible.
4. **DAP:** Debug Access Port. This is a component of the Arm CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention.  $\mu$ Vision uses the DAP to update Memory, Watch, Peripheral and RTOS kernel awareness windows while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no code stubs are needed. You do not need to configure or activate DAP.  $\mu$ Vision configures DAP when you select a function that uses it. Do not confuse this with CMSIS\_DAP which is an Arm on-board debug adapter standard.
5. **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
6. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDO.
7. **Trace Port:** A 4 bit port that ULINKpro uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ITM:** Instrumentation Trace Macrocell: As used by  $\mu$ Vision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations.  $\mu$ Vision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
9. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. The ULINKpro provides ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis. ETM is output on the Trace Port or accessible in the ETB (ETB has no Code Coverage or Performance Analysis).
10. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINKpro. ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific datasheet.
11. **MTB:** Micro Trace Buffer. A portion of the device internal user RAM is used for an instruction trace buffer. Only on Cortex-M0+ processors. Cortex-M3/M4 and Cortex-M7 processors provide ETM trace instead.
12. **Hardware Breakpoints:** The Cortex-M0+ has 2 breakpoints. The Cortex-M3, M4 and M7 usually have 6. These can be set/unset on-the-fly without stopping the processor. They are no skid: they do not execute the instruction they are set on when a match occurs. The CPU is halted before the instruction is executed.
13. **Watchpoints:** Both the Cortex-M0, M0+, Cortex-M3, Cortex-M4 and Cortex-M7 can have 2 Watchpoints. These are conditional breakpoints. They stop the program when a specified value is read and/or written to a specified address or variable. There also referred to as Access Breaks in Keil documentation.

### Read-Only Source Files:

Some source files in the Project window will have a yellow key on them:  This means they are read-only. This is to help unintentional changes to these files. This can cause difficult to solve problems. These files normally need no modification.  $\mu$ Vision icon meanings are found here:

[www.keil.com/support/man/docs/uv4/uv4\\_ca\\_filegrp\\_att.htm](http://www.keil.com/support/man/docs/uv4/uv4_ca_filegrp_att.htm)

If you need to modify one, you can use Windows Explorer to modify its permission.






1. In the Projects window, double click on the file to open it in the Sources window.
2. Right click on its source tab and select Open Containing folder.
3. Explorer will open with the file selected.
4. Right click on the file and select Properties.
5. Unselect Read-only and click OK. You are now able to change the file in the  $\mu$ Vision editor.
6. It is a good idea to make the file read-only when you are finished modifications.



## Part B: Stand-alone Project Examples:

### 1) Blinky Example Program using the XMC 2Go 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  $\mu$ Vision to use any other supported adapter.

1. Connect your 2Go board to your computer with a USB cable.
2. Start  $\mu$ Vision by clicking on its desktop icon. 
3. Select Project/Open Project. Open the MDK project file:  
C:\00MDK\Boards\Infineon\XMC\_2Go\Blinky\Blinky.uvprojx.
4. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
5. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.  $\mu$ Vision programs the Flash memory. Progress is shown.
6. Click on the RUN icon.  Note: you stop the program with the STOP icon. 

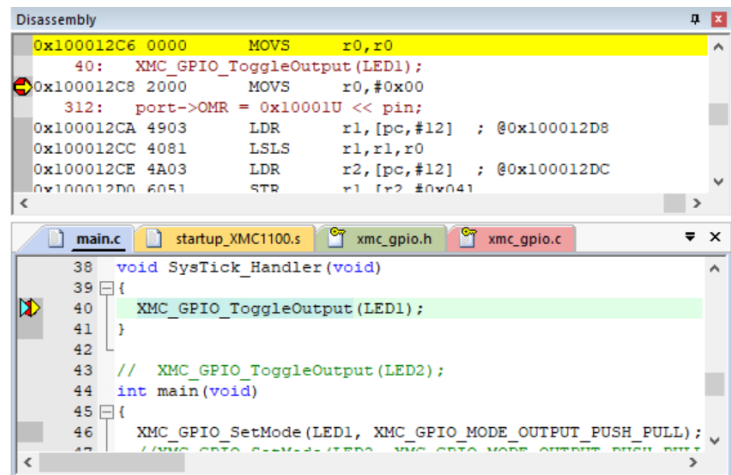
**LED 1 on the 2Go board will now blink. This is connected to P1.0.**

**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.

### 2) Hardware Breakpoints:

1. With Blinky running, click in the left margin on a darker gray block somewhere in the SysTick\_Handler function which starts at near line 38.
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. Note you can set and unset hardware breakpoints while the program is running or stopped with ARM CoreSight™ technology.
6. The XMC1100 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.
7. If you set too many breakpoints,  $\mu$ Vision will notify you. Sometimes  $\mu$ Vision will use one of the available breakpoints for one of its internal operation such as single stepping.
8. Leave the breakpoint in the SysTick handler.



**TIP:** The J-Link has the capability of modifying your code in Flash by replacing an instruction. Segger calls these Flash Breakpoints. This is effective but intrusive. Heisenbugs are a possibility with this technique.

**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.

### 3) 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.

This simple example will illustrate how this window operates. It can handle much more complicated stacks.

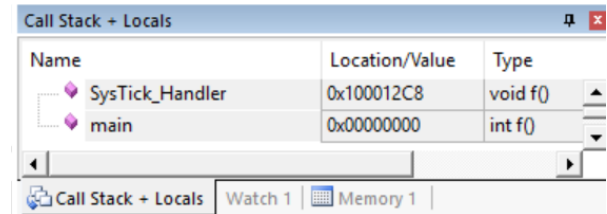
1. Open the Call Stack + Locals window by clicking on its tab.
2. Shown is the Locals window for the main function with the hardware breakpoint active from the previous page.
3. The two functions on the Stack are shown: main and SysTick\_Handler.

**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.

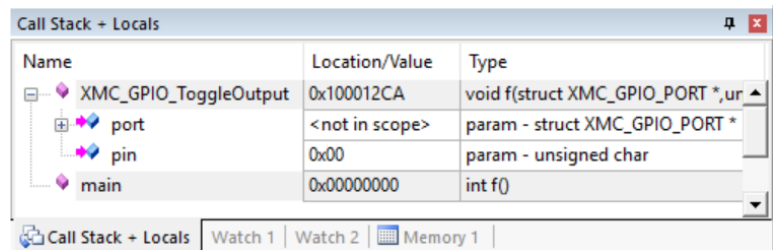
When a function exits, it is removed from the list.

The most recent called function is at the top of this table.

This table is active only when the program is stopped.



4. Click on the Step icon or F11:
5. The function XMC\_GPIO\_ToggleOutput is displayed as shown below:



6. Variables will update as appropriate.
7. Click on the Step icon or F11 a few times.
8. You will eventually exit SysTick\_Handler and then back into main().
9. Click on RUN to repeat this process.
10. **TIP:** To exit a function immediately, click on the StepOut icon (Ctrl-F11).

11. **Remove any breakpoints when you are done.** You can click on them individually or Ctrl-B and select Kill All.

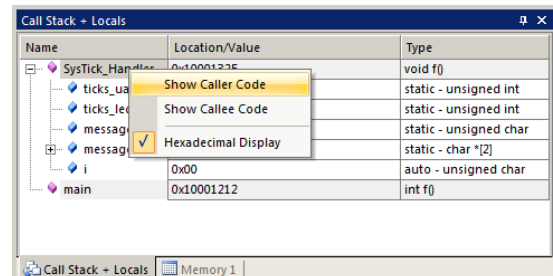
**TIP:** You can modify a variable value in the Call Stack & Locals window when the program is stopped.

#### 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.

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.

**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](http://www.keil.com/appnotes/docs/apnt_209.asp)

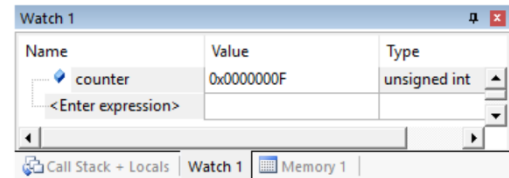
## 4) Watch and Memory Windows and how to use them:

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:

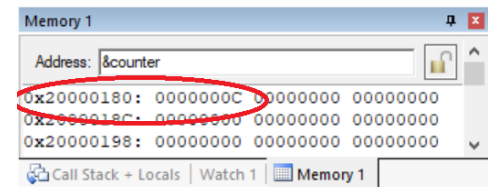
**Add a global variable:** Call Stack, Watch and Memory windows can't see local variables unless stopped in their function.

1. Stop the processor  and exit Debug mode. 
2. Declare a global variable in the usual manner (I called it **counter**) near line 37 in main.c:  
**unsigned int counter = 0;**
3. Add the statements near line 42 just in SysTick\_Handler:  
**counter++;**  
**if (counter > 0xF) counter = 0;**  

4. Select File/Save All.
5. Click on Rebuild .
6. Enter Debug mode.  The Flash will be programmed.
7. Click on RUN . **Note:** You can configure a Watch window while the program is running or not. You can also do this with a Memory window.
8. In main.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 and is not intrusive.
10. You can change a variable in a Watch window if the variable is not changing too fast by clicking on it and entering the new value..



### 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\_0180).
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.

**TIP:** Watch and Memory windows cannot display a local variable. Make it static or a global variable or in a structure.

**TIP:** To view variables and their location use the Symbol window. Select View/Symbol Window while in Debug mode. If you select a variable from the Symbols window, it will be entered fully qualified into a Memory or Watch window.

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.

**TIP:** If these windows do not update, select View and select Periodic Window Update: 

## 5) 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:

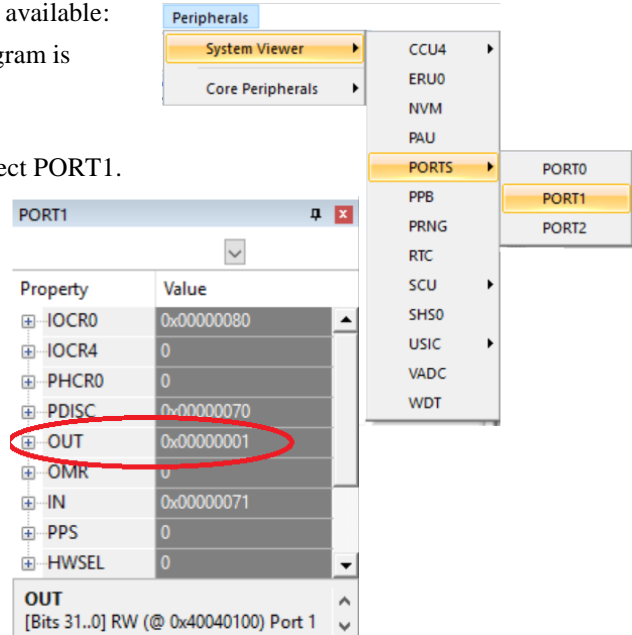
1. Click on RUN. You can open SV windows when your program is running.

### GPIO Port 1:

2. Select Peripherals/System Viewer and then PORTS and select PORT1.
3. This window opens. Expand OUT:
4. You can now see OUT P1.0 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. This makes it easy to find these physical addresses.

**TIP:** CPU Clock: enter SystemCoreClock into a Watch window and the CPU speed is displayed. Right click to change it to a decimal value. This is a global variable created by the CMSIS file system\_XMC1000.c.



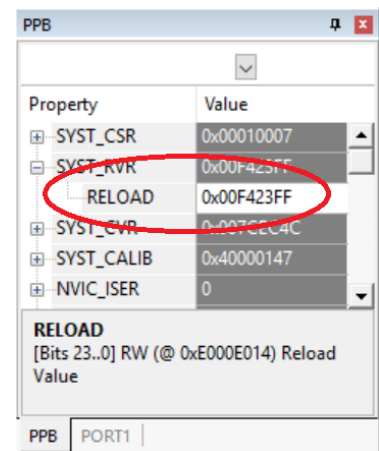
**SysTick Timer:** This program uses the Cortex-M0 SysTick timer to determine the period of the LED blinking.

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 >> 1); function call in main().
4. Note that it is set to 0x00F4 23FF (decimal 15,999,999). This results from the hex value of (36,000,000/2)-1 that is programmed into SysTick\_Config() in main() in main.c. Changing the variable passed to this function is how you can change how often the SysTick timer creates its interrupt 15. In this case, it is every 0.5 second.
5. In the RELOAD register in the SysTick window, *while the program is running*, block the first 'F' with your cursor and change it to 2 (0x0024 23FF) and press Enter.
6. The blinking LED will speed up. This will convince you of the power of ARM CoreSight debugging.
7. Replace RELOAD with 0x00F4 23FF. A CPU RESET  will also do this. The program will restart and run.
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.



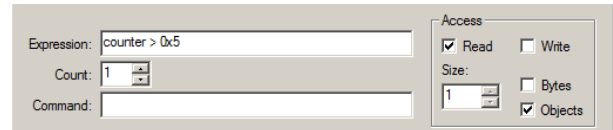
## 6) Watchpoints: Conditional Breakpoints




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,  $\mu$ Vision stops to test the memory location. Cortex-M3/M4 Watchpoints are not intrusive for the equality test.

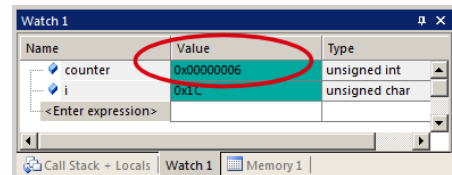
1. Use the same Blinky configuration as the previous page. Stop the program if necessary.  Stay in debug mode.
2. We will use the global variable **counter** you created in Blinky.c to explore Watchpoints.
3. Select Debug in the main  $\mu$ Vision window and then select Breakpoints or press Ctrl-B.
4. 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 ==.



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: counter will equal 6.
12. Click on RUN and the program will run to the next read and/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  $\mu$ Vision for speed considerations during the test. You will also notice the program slows down. This is because  $\mu$ Vision must test the condition by momentarily 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. .

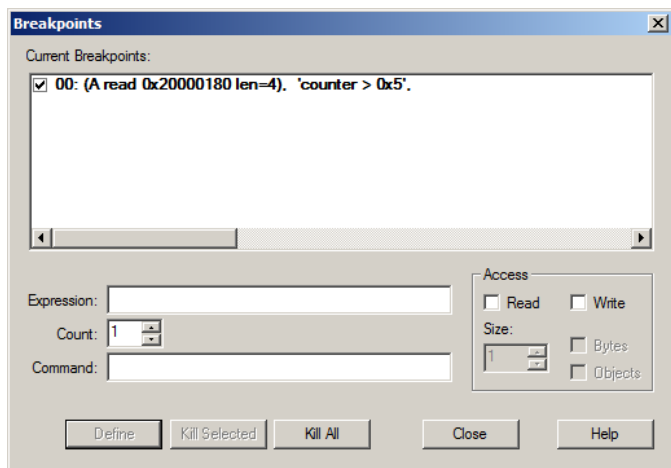


**TIP:** You cannot configure Access Breakpoints on-the-fly while the program is running like you can with hardware breakpoints.

**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 an Access Breakpoint. An example is: \*((unsigned long \*)0x20000004)







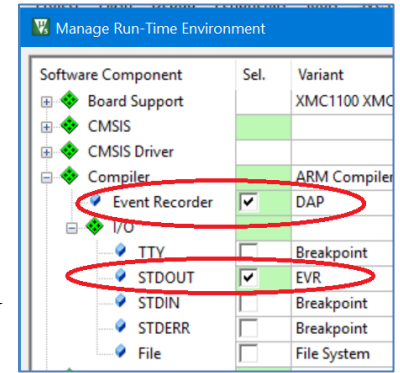
## 7) printf using Event Recorder:

printf is provided by Event Recorder instrumentation that requires minimal user code. printf data will be displayed in the Debug (printf) Viewer and Event Recorder windows. If you are using a Cortex-M3/M4/M7/M33 you can send this data out the ITM port 0 using SWV (Serial Wire Viewer). Cortex-M0/M0+ and M23 do not have SWV. They use DAP.

1. Stop the program  and exit Debug mode .

### Add STDOUT File (retarget\_io.c):

1. Open the Manage Run-Time Environment window (MRTE) .
2. Expand Compiler and I/O as shown here: .
3. Select Event Recorder and then DAP as shown:
4. Select STDOUT and then EVR. This adds the file retarget\_io.c to the project.
5. Ensure all blocks are green and click OK to close the MRTE. Click Resolve if there are red or orange blocks.








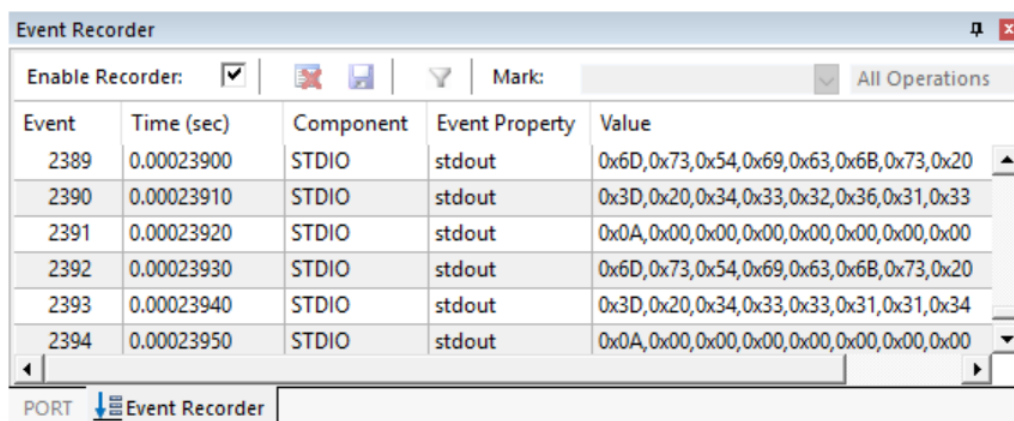
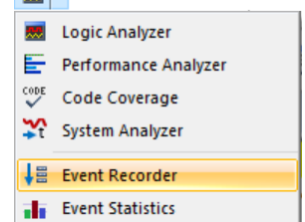
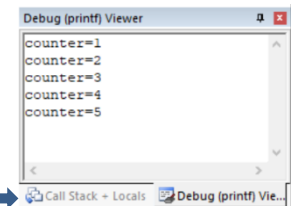
**TIP:** If you selected ITM instead of EVR, printf will require SWV. Cortex-M0 does not have SWV, only DAP.

### Add printf and ER statements to Blinky.c:

1. At the top of main.c near line 35, right-click and select 'Insert #include file' and then select EventRecorder.h".
2. At the next line, add #include "stdio.h".
3. At the beginning of main() add this line: EventRecorderInitialize (EventRecordAll, 1);
4. Inside the SysTick\_Handler, near line 45, add this line: printf("Hello\n");

### Compile and Run the Project:








1. Select File/Save All or click .
2. Rebuild the source files  and enter Debug mode .
3. Click on View/Serial Windows and select Debug (printf) Viewer and click on RUN.
4. In the Debug (printf) Viewer you will see the printf statements appear as shown here: .
5. Right click on the Debug window and select Mixed Hex ASCII mode. Note other useful settings that are available.
6. Select the small arrow beside the Analysis window icon and select Event Recorder. .
7. Event Recorder opens and the ASCII printf frames are displayed as shown below:
8. It is possible to annotate your code with Event Recorder with your own messages.
9. See [www.keil.com/pack/doc/compiler/EventRecorder/html/](http://www.keil.com/pack/doc/compiler/EventRecorder/html/)
10. RTX 5 and Keil middleware is configured with annotations. They can be compiled out.



Event	Time (sec)	Component	Event Property	Value
2389	0.00023900	STDIO	stdout	0x6D,0x73,0x54,0x69,0x63,0x6B,0x73,0x20
2390	0.00023910	STDIO	stdout	0x3D,0x20,0x34,0x33,0x32,0x36,0x31,0x33
2391	0.00023920	STDIO	stdout	0x0A,0x00,0x00,0x00,0x00,0x00,0x00,0x00
2392	0.00023930	STDIO	stdout	0x6D,0x73,0x54,0x69,0x63,0x6B,0x73,0x20
2393	0.00023940	STDIO	stdout	0x3D,0x20,0x34,0x33,0x33,0x31,0x31,0x34
2394	0.00023950	STDIO	stdout	0x0A,0x00,0x00,0x00,0x00,0x00,0x00,0x00

## 8) RTX\_Blinky Example Program with Keil RTX RTOS: A Stepper Motor example

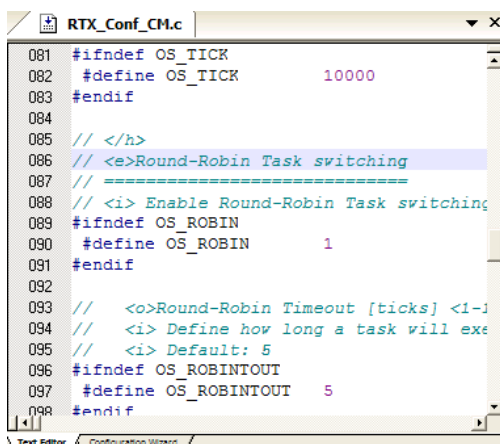
Keil provides two versions of RTX, a full feature RTOS. See [www.keil.com/rtx](http://www.keil.com/rtx). RTX is included as part of the Keil MDK tool suite. RTX 4 has a BSD license and RTX 5 is Apache 2.0. RTX with source code is provided with all MDK versions. See [www.arm.com/cmsis](http://www.arm.com/cmsis) for documentation. This example explores RTX 4. Keil will work with any RTOS. A real-time awareness viewer for RTX is provided with  $\mu$ Vision. FreeRTOS is now supported with  $\mu$ Vision.

1. Start  $\mu$ Vision by clicking on its icon if not already running.  You must have installed RTX\_Blinky.
2. Exit Debug mode if necessary. 
3. Select Project/Open Project and open C:\00MDK\Boards\Infineon\XMC\_2Go\RTX\_Blinky\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 two LEDs will blink alternatively.
8. Click on STOP .

### The Configuration Wizard for RTX:

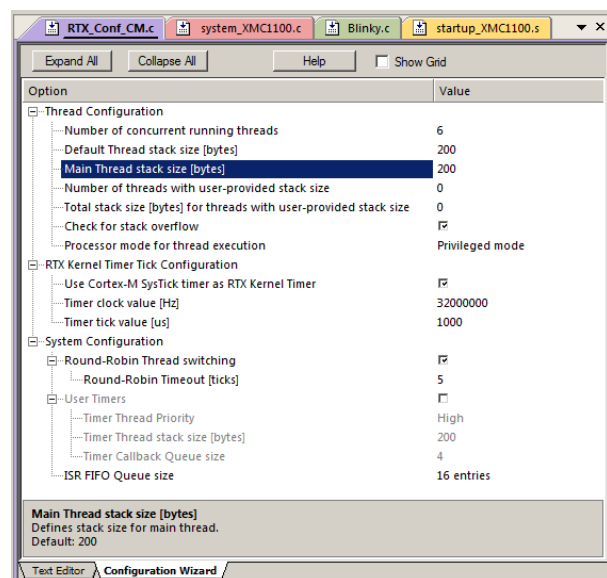
1. In the Project window, double click on RTX\_Conf\_CM.c to open it in a Source window or with File/Open.
  2. Click on the RTX\_Conf\_CM.c source file tab as shown below on the left below if it is not in focus.
  3. Click on Configuration Wizard at the bottom and your view will change to the Configuration 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](http://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/](http://www.keil.com/mdk5/). It has very useful information on implementing and managing RTX.



```
081 #ifndef OS_TICK
082 #define OS_TICK 10000
083 #endif
084
085 // </h>
086 // <e>Round-Robin Task switching
087 // =====
088 // <i> Enable Round-Robin Task switching
089 #ifndef OS_ROBIN
090 #define OS_ROBIN 1
091 #endif
092
093 // <o>Round-Robin Timeout [ticks] <1-1
094 // <i> Define how long a task will exe
095 // <i> Default: 5
096 #ifndef OS_ROBINTOUT
097 #define OS_ROBINTOUT 5
098 #endif
```


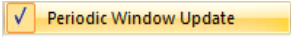
Text Editor: Source Code

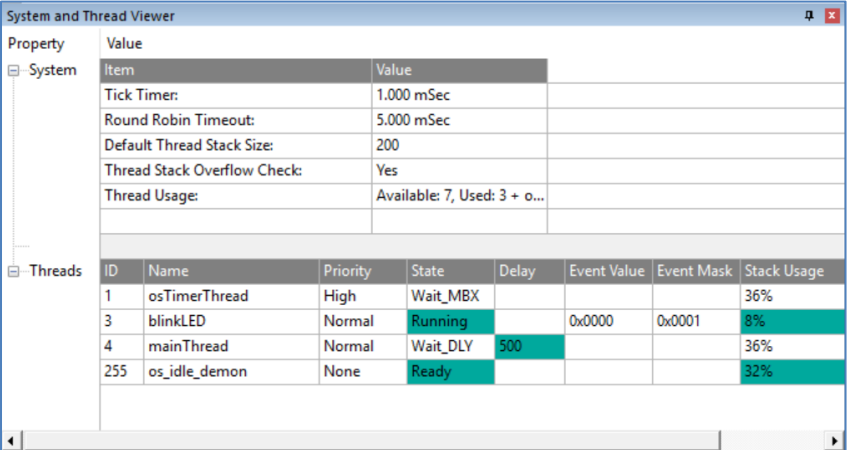


Configuration Wizard

## 9) 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. 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 to suit your needs.
2. µVision also has an Event Viewer which displays RTX threads in a graphical format. The XMC1000 Cortex-M0 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.




Property	Value
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: 7, Used: 3 + o...

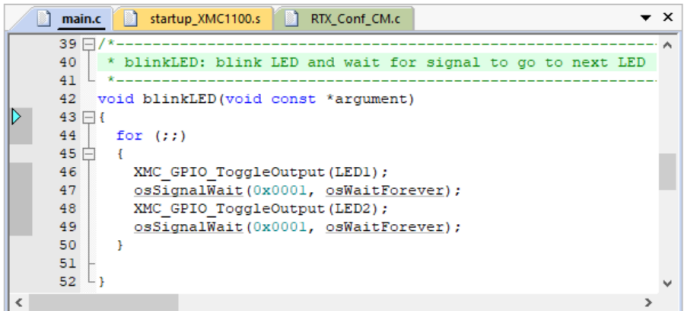
  

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				36%
3	blinkLED	Normal	Running		0x0000	0x0001	8%
4	mainThread	Normal	Wait_DLY	500			36%
255	os_idle_demon	None	Ready				32%



**Demonstrating Threads:** (note: Keil MDK documentation uses the term Threads rather than Tasks for consistency.)

Blinky.c contains three threads. The blinkLED is the main thread. It is shown below: It has a similar format to a function.

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 blinkLED as shown: (but not on the for (;;) line)
3. Click on RUN .
4. 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 blinkLED is in the run state. The states of the other tasks are displayed as well.



```
39 /*
40  * blinkLED: blink LED and wait for signal to go to next LED
41  */
42 void blinkLED(void const *argument)
43 {
44     for (;;)
45     {
46         XMC_GPIO_ToggleOutput(LED1);
47         osSignalWait(0x0001, osWaitForever);
48         XMC_GPIO_ToggleOutput(LED2);
49         osSignalWait(0x0001, osWaitForever);
50     }
51 }
52
```

5. Click on RUN . The other thread will show as "Running". Each time you click RUN, the thread will run.
6. Remove the breakpoints and close the RTX Tasks window. Exit Debug mode. 

### More Information of obtaining and using RTX:

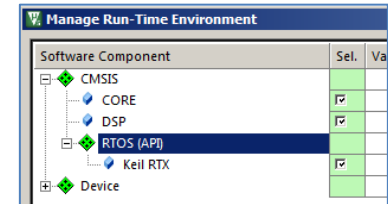
It is very beneficial to use an RTOS. RTX is a good choice. It is small, efficient and easy to use yet it is full featured. RTX 4 and RTX 5 is contained in MDK. RTX 5 is also located here: [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)

## 10) 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 and on GitHub: [https://github.com/ARM-software/CMSIS\\_5/tree/develop/CMSIS/DSP](https://github.com/ARM-software/CMSIS_5/tree/develop/CMSIS/DSP).

Documentation can be accessed here: Select the DSP tab: [www.keil.com/pack/doc/cmsis/General/html/](http://www.keil.com/pack/doc/cmsis/General/html/)




The appropriate DSP library is selected in the RTE window:

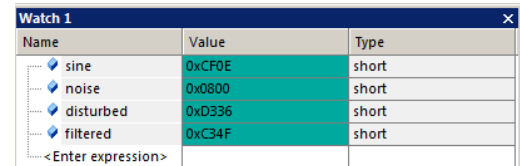


This example creates a sine wave to which another sine wave (to represent noise) is added, and then the noise is filtered out leaving the original sine wave. Four threads contain each task.

This example incorporates Keil RTX RTOS. RTX is available free. Source code is provided.

To obtain this example file, go to [www.keil.com/appnotes/docs/apnt\\_260.asp](http://www.keil.com/appnotes/docs/apnt_260.asp) and copy the MDK 5 version into C:\00MDK\Boards\Infineon\XMC\_2Go. A \DSP directory will be created. You probably have already done this on page 5.

1. Stop the program and exit Debug mode if necessary.
2. Select Project/OpenProject and open the project file: C:\00MDK\Boards\Infineon\XMC\_2Go\DSP\sine.uvprojx.
3. Build the files.  There will be no errors or warnings.
4. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
5. Click on the RUN icon. 
6. Open Watch 1 by selecting View/Watch/Watch 1 if necessary.
7. Four global variables will be displayed in Watch 1 as shown here:  
If these variables are changing the program is most likely working properly.
8. Each of these variables represent the value of one of four waveforms created by the DSP program.



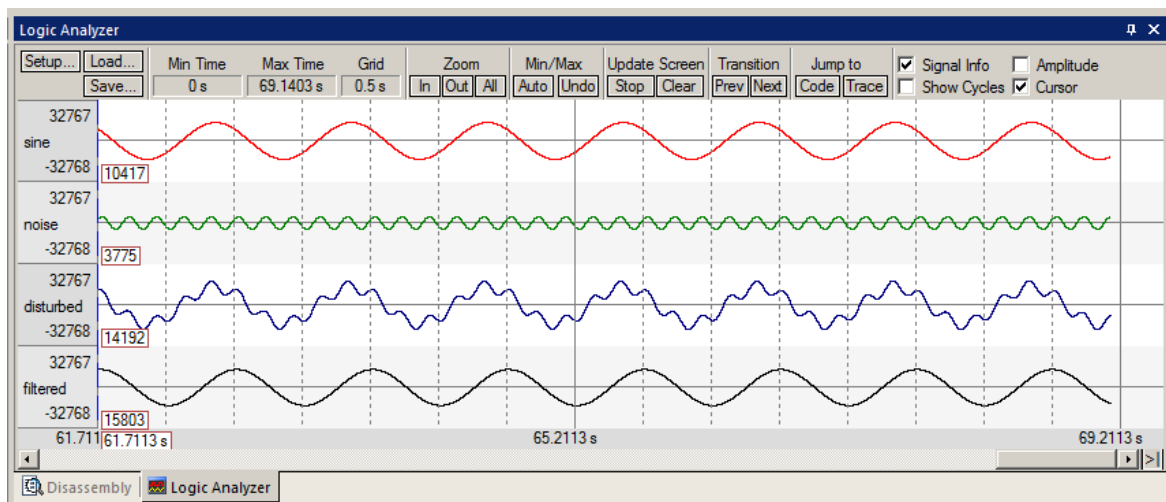
Name	Value	Type
sine	0xCF0E	short
noise	0x0800	short
disturbed	0xD336	short
filtered	0xC34F	short

### 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](http://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

System


Threads

Value

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 + o...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	main	Normal	Wait_DLY				36%
2	sine_gen	Normal	Wait_AND	65500	0x0000	0x0001	44%
3	noise_gen	Normal	Wait_DLY	16	0x0000	0x0001	48%
4	disturb_gen	Normal	Wait_DLY	16	0x0000	0x0001	48%
5	filter_tsk	Normal	Wait_AND	65400	0x0000	0x0001	44%
6	sync_tsk	Normal	Wait_AND		0x0000	0x0001	44%
255	os_idle_demon	None	Running				

**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  $\mu$ Vision might commandeer one for its use, usually for run to main() and single-stepping. If you select too many breakpoints,  $\mu$ Vision 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 data trace 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.



## printf:

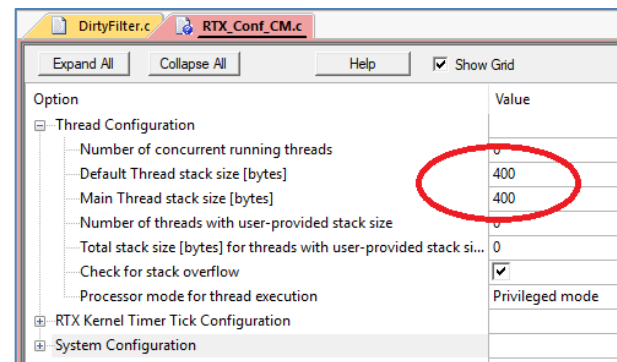
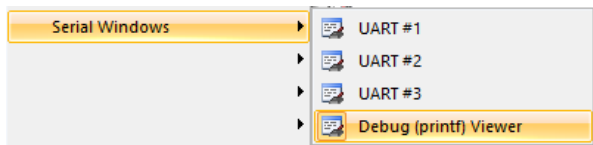
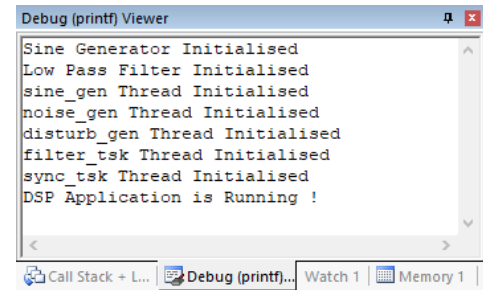
In DirtyFilter.c there are several printf statements. These will be displayed on the Debug (printf) Viewer as shown here:

You can open this window if needed by selecting from the  $\mu$ Vision menu: View/Serial Windows/Debug (printf) Viewer as shown below left:




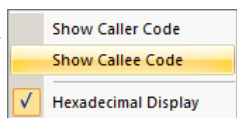
This uses the same technique as described on page 14.

To use this feature the stack sizes of RTX had to be increased to 400 bytes each from 200 bytes: See RTX\_Config\_CM.c below right:

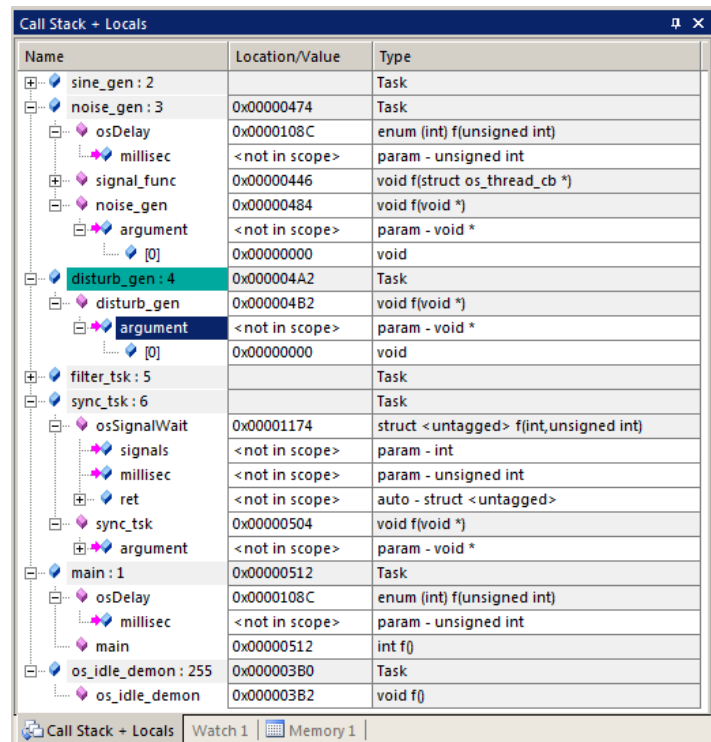
The  $\mu$ Vision Event Recorder was utilized to output printf statements since the Arm Cortex-M0 does not have Serial Wire Viewer. Cortex-M4 does.



## 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 a breakpoint, 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  $\mu$ Vision.

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

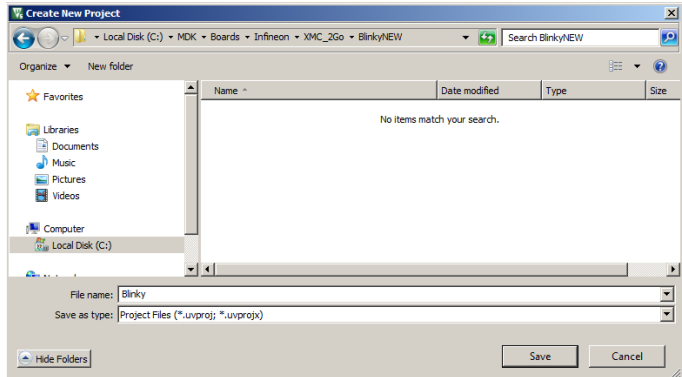
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a template for your own projects. However, we will start an example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load 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  $\mu$ Vision 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 5.
3. You do not need to copy any examples over.

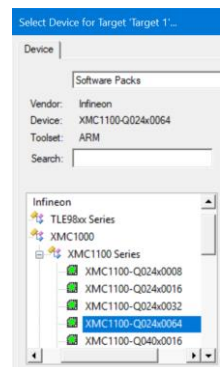
### Create a new Directory and a New Project:

1. Click on Project/New  $\mu$  Vision Project...
2. In the window that opens, shown here, go to the folder C:\00MDK\Boards\Infineon\XMC\_2Go\
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 Blinky.uvproj in C:\00MDK\Boards\Infineon\XMC\_2Go\BlinkyNEW.
7. When you clicked on Save, the Select Device for Target ... window opens:



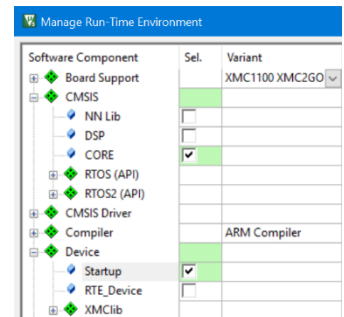
### Select the Device you are using:

1. Expand Infineon and then XMC1000 Series, then expand XMC1100 and then select XMC1100-Q024x0064 as shown here:
2. Click OK. The Manage Run-Time Environment window below bottom right opens:



### Select the CMSIS components you want:

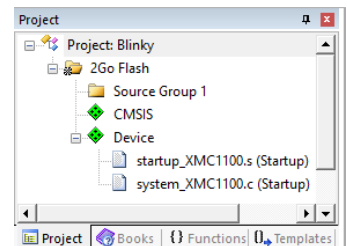
1. In Board Support, select XMC1100 XMC2GO as shown below right:
2. Select CMSIS/CORE and Device/Startup as shown below right. They will be highlighted in Green indicating there are no other files needed. Click OK.
3. Click on File/Save All or select the Save All icon:
4. The project Blinky.uvproj (MDK 4) will now become Blinky.uvprojx (MDK 5).
5. Expand all the folders in the Project window.
6. You now have a new project list as shown on the bottom right: The appropriate CMSIS files you selected have been automatically entered and configured for the processor you chose. There are no user source files yet.
7. Note the Target Selector says Target 1. Highlight Target 1 in the Project window.
8. Click once on it and change its name to **2Go Flash** and press Enter. The Target selector name will also change.




### What has happened to this point:

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

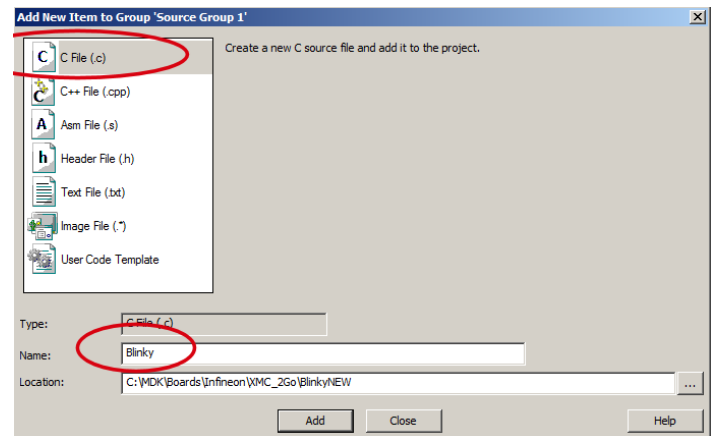
**TIP:** Other components such as File System, Graphics, Network and USB are part of Keil Middleware in MDKpro. Consult Keil Sales or Technical support. [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com)





### 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 New Item to Group 'Source Files'...



### Add Some Code to Blinky.c:

1. In the blank Blinky.c, right click at the top of the file and select Insert '#include file'.
2. Select XMC1100.h.
3. Add the C code below in the blank Blinky.c,:
4. Click on File/Save All or 
5. Build the files.  There will be no errors or warnings if all source code was entered correctly.





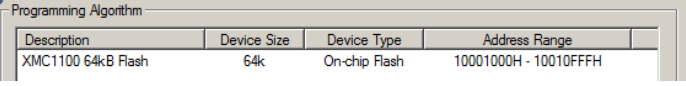


```
#include "XMC1100.h" //already added in step 2 above
unsigned int counter = 0;

/*-----
MAIN function
-----*/
int main (void) {

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





**TIP:** You could also add existing source files:  We will not do this in this tutorial.

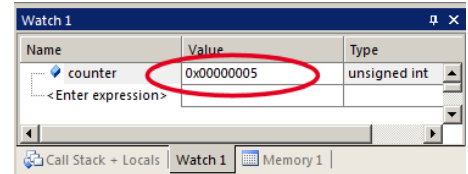
### Configure the Target 2Go Flash: *Please complete these instructions carefully to prevent unusual problems...*

1. Select the Target Options icon . Select the **Target** tab.
2. Enter 32 in Xtal (MHz). This is used only for timing calculations and not in your source code.
3. Select Use MicroLIB to compile smaller code size.
4. Click on the **Debug** tab. Select J-Link/J-Trace Cortex in the Use: box:  Use: J-LINK / J-TRACE Cortex
5. Plug your 2Go board to your PC. Select the Settings: icon. 
6. Select SW as shown here in the Port: box:  Port: SW A JTAG selection will not work. If your 2Go is connected to your PC, you **must** see a valid IDCODE and Device Name (ARM CoreSight SW-DP) in the SW Device box.
7. Click on OK **once** to go back to the Target Configuration window. Or, fix the connection problem if you have one.
8. Click on the **Utilities** tab. Select Settings and confirm the correct Flash algorithm is present: Shown is the correct one for the Infineon XMC 2Go board: 
9. Click on OK twice to return to the main menu.
10. Click on File/Save All or 
11. 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 ! Please turn the page....**

## Running Your Program:

1. Ensure your 2Go board is connected and powered to your PC with a USB cable.
2. Enter Debug mode by clicking on the Debug icon .  Flash progress will be indicated in the Output Window.
3. Click on the RUN icon.  Note: you stop the program with the STOP icon. 
4. No LEDs will blink since there is no source to accomplish this task. You would have to add such code yourself.
5. With the program running, right click on `counter` in Blinky.c and select Add counter to ... and select Watch 1.
6. `counter` will be updating as shown here: 
7. 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.
8. 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.


## Displaying the CPU Clock Frequency:

1. In Watch 1 window, double-click on <Enter expression> and type the global variable `SystemCoreClock`.
2. Right click on the data field and unselect Hexadecimal Display.
3. The clock speed of 32 MHz will display in Watch 1.

## Sending Your Project Files To Someone Else:

1. 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.
2. If they need the .axf executable and intermediary files and are unable to compile them, do not delete them.
3. In the root of your project will be a file `YourProjectName.uvguix.username`. This file contains GUI settings specifically of the windows settings. To transfer this to another person, delete the computer ID name.






In this case, leave only Blinky.uvguix Delete .bobby01

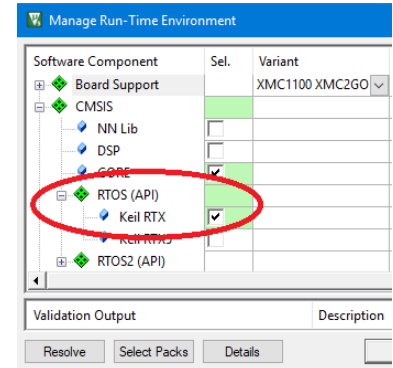
 Blinky.uvguix.bobby01

## 12) 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: See [www.keil.com/rtx](http://www.keil.com/rtx). MDK contains both RTX 4 and RTX 5. This page uses RTX 4.

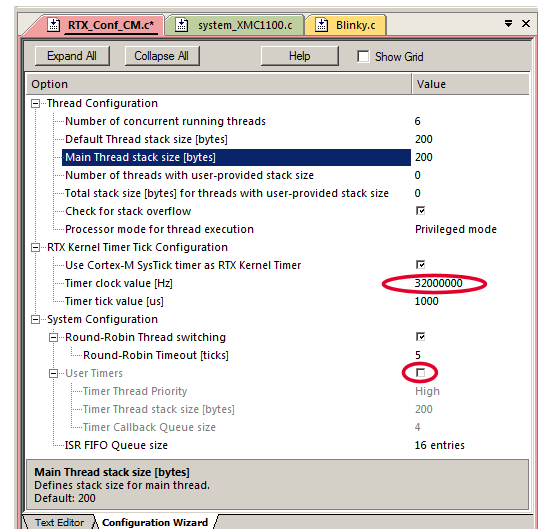
Configuring RTX is easy in MDK. These steps use the Blinky example created on the previous pages. You can use these procedures to convert an existing project to operate using RTX.

1. Using the same example from the preceding pages, Stop the program  and Exit Debug mode. .
2. Open the Manage Run-Time Environment window: .
3. Expand all the elements as shown here: .
4. Select Board Support as XMC1100 XMC2GO:
5. Select Keil RTX as shown and click OK.
6. Appropriate RTX files will be added to your project. See the Project window.
7. In Blinky.c, at the top, add this line: `#include "cmsis_os.h"`
8. 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 at the bottom of the window: Select Expand All.
4. The window is displayed here: .
5. Set Timer clock value: to 32000000 as shown: (32 MHz).
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. .
2. Enter Debug mode:  The Flash will be programmed.
3. Click on the RUN icon. .
4. Select Debug/OS Support/System and Thread Viewer. The window below opens up.
5. You can see two threads: `os_idle_demon` and `main`. The main thread is the only one running. As you add more threads to create a real RTX program, they 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 to add RTX components.

**Getting Started MDK 5:** Obtain this book here: [www.keil.com/mdk5/](http://www.keil.com/mdk5/). It includes information on implementing and maintaining RTX.

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: 1					
Threads	ID	Name	Priority	State	Delay	Event Value	Event Mask
	255	os_idle_demon	0	Ready			
	1	main	Normal	Running			



**Add the 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:\00MDK\Boards\Infineon\XMC\_2Go\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 file:** You should now have three #include lines.  

```
#include "RTE_Components.h"
```
3. **Declare Two Global Variables: (starting near line 6 just after the declaration of counter )**  

```
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 23 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 at near line 34 before main() )**  




```
osThreadDef(phaseA, osPriorityNormal, 1, 0);
osThreadDef(phaseB, osPriorityNormal, 1, 0);
```
8. **int main(void) is already located at this point: (starting near line 37 with appropriate line feeds)**
9. **Create the two Threads phaseA and phaseB: (starting near line 42 in main() before the while(1) loop)**  

```
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) loop)**  

```
osSignalSet(tid_phaseA, 0x0001);      /* send signal to phaseA thread */
osDelay(osWaitForever);
```
11. **While(1){ } is already located at this point: (near line 48)**
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. Enter Debug mode:  The Flash will be programmed.
3. 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.

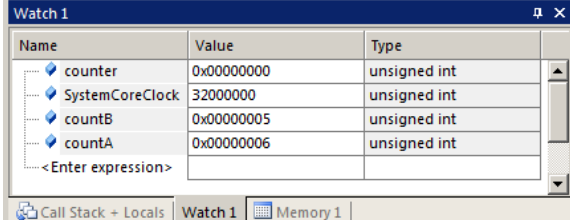
5. Right click on countA and select Add countA to... and select Watch 1.
6. Right click on countB and select Add countB to... and select Watch 1.
7. These two variables will be displayed in Watch 1 and will be incrementing as shown here:

**Congratulations:** Your First RTX program is running correctly !

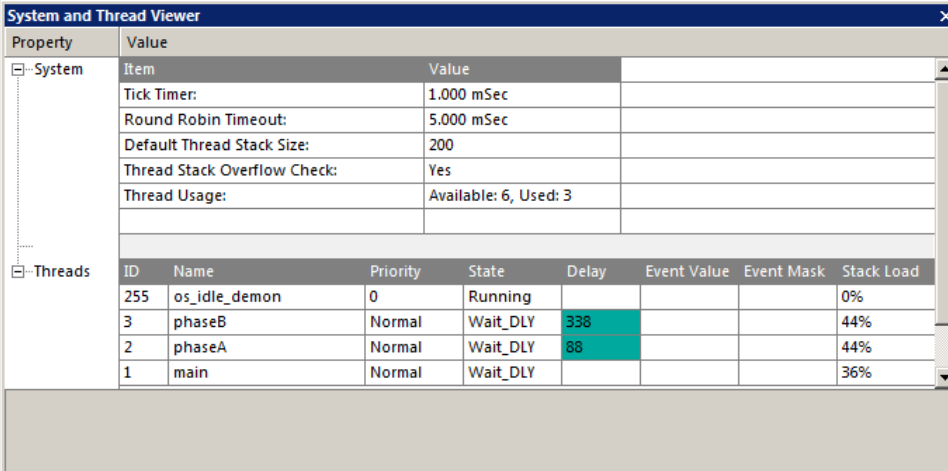
**TIP:** The counter and SystemCoreClock Watch 1 entries are left over from previous steps.

### 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 resize it or drag it 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.
14. **TIP:** counter no longer increments because the statement `osDelay(osWaitForever);` stops the main thread.
15. This is shown in the System and Thread Viewer as the main thread is shown as always in the wait\_DLY state. If you comment this line out and rebuild and run the program, the main thread will run and counter will increment.
16. Note the idle demon is mostly Running or in Wait\_DLY respectively.



Name	Value	Type
counter	0x00000000	unsigned int
SystemCoreClock	32000000	unsigned int
countB	0x00000005	unsigned int
countA	0x00000006	unsigned int
<Enter expression>		



Property	Value
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: 3

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
3	phaseB	Normal	Wait_DLY	338			44%
2	phaseA	Normal	Wait_DLY	88			44%
1	main	Normal	Wait_DLY				36%

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](http://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 4.4.2 or later to create both a Blinky and an RTX\_Blinky project. Please install DAVE to your computer in the usual manner. See installing the J-Link drivers on page 7. They must be correctly installed.

This example is adapted from the Infineon DAVE v4.1.2 Quick Start 1 to debug with MDK.

[http://dave.infineon.com/Libraries/Tutorials/DAVE-Quick\\_Start\\_1v2.pdf](http://dave.infineon.com/Libraries/Tutorials/DAVE-Quick_Start_1v2.pdf)

You must have downloaded and installed the XMC1100 Packs into µVision as shown on page 5 and J-Link on page 8.

### 1) Blinky using DAVE:

We will create a µVision project using DAVE to blink one LED on the 2Go board. A PWM will be created and configured to blink the LED. DAVE will then create a µVision project using a .gpdsc file. µVision will then import it.

#### Start DAVE and create a DAVE project:

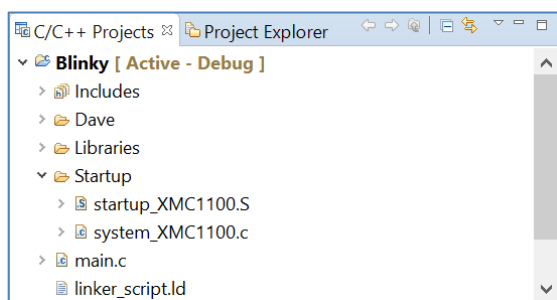
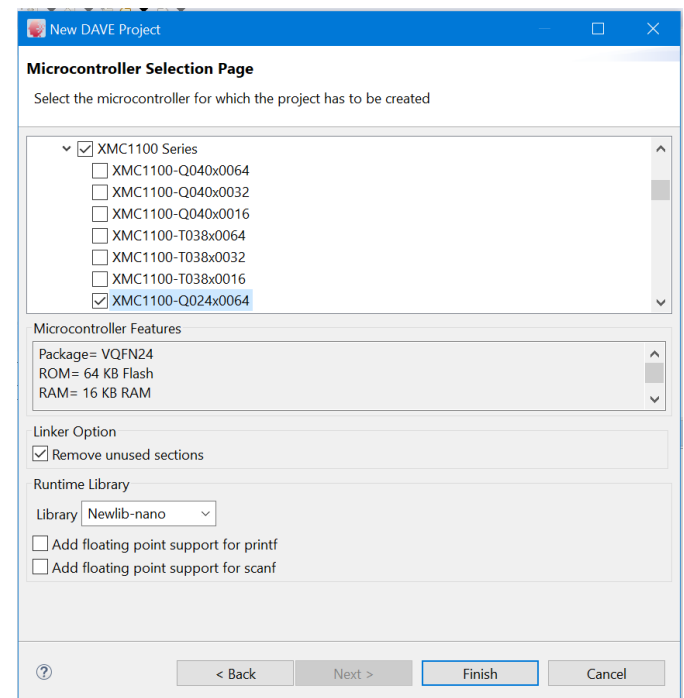
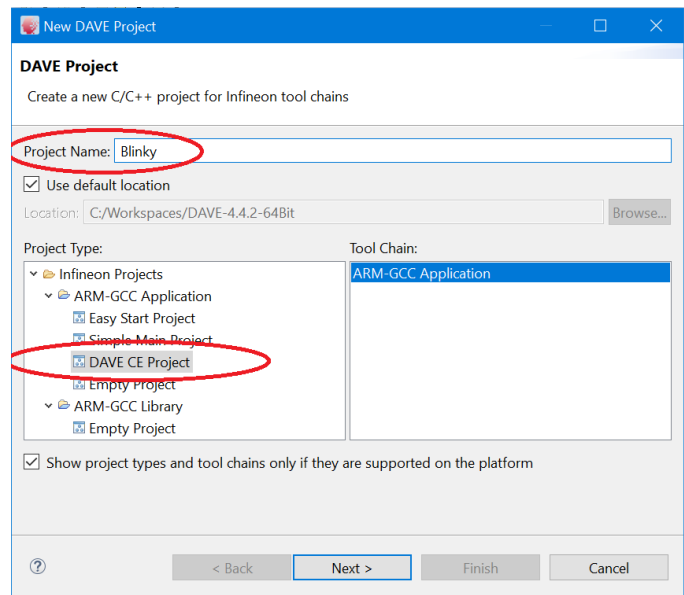
1. Start DAVE by clicking on its icon:
2. Use the default workspace.
3. Select File/New/Dave Project - this window opens:
4. Select DAVE CE Project: Always select Dave CE.
5. In Project Name box enter Blinky as shown here:
6. Select Next > and the Target Selection Page opens:
7. Select XMC1100-Q024F0064 or the processor you are using as shown below right:
8. Click on Finish. DAVE will create your project.

#### Examine the Project Contents:

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\_XMC1100.s which contains processor initialization code and system\_XMC1100.c contains mostly clock settings.

**TIP:** DAVE dynamically saves your work into the default workspace: C:\Workspaces\DAVE-4.4.2-64Bit or where you chose to save your project files. You can manually save your work with File/Save or Ctrl-S.


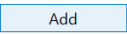
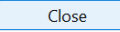
15. We will next create one PWMs to blink the LED.

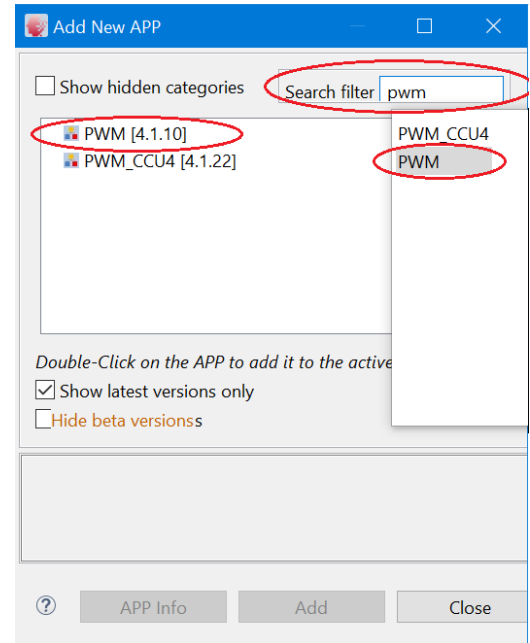


### Add the DAVE PWM Apps:

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  $\mu$ Vision project. Select Help/Check for DAVE App Updates to make sure you have the latest software.

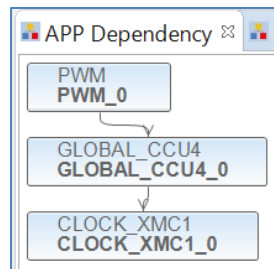
You will be in the DAVE CE perspective as shown on the upper right side of the DAVE GUI:

1. Select Add App  in the Tool Panel, or in the APP Dependency Tree pane or in the main menu: Select DAVE/Add New APP....
2. Enter **pwm** in the Search filter:
3. Highlight PWM in the box and press Enter or double-click.
4. Then highlight PWM [4.1.22] as shown here:
5. Click Add: 
6. Close the Add New APP window: 
7. We will use the PWM to blink the LED.



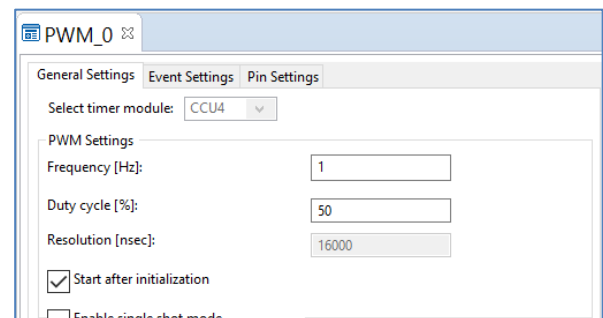
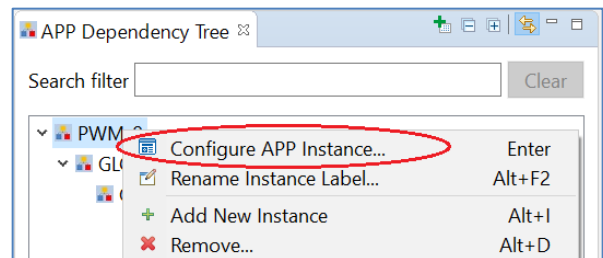
### View the Dependencies:

1. PWM\_0 is shown as you created it:
2. If you hover your mouse over one an information box will open.
3. The other two APPs are included automatically by DAVE.



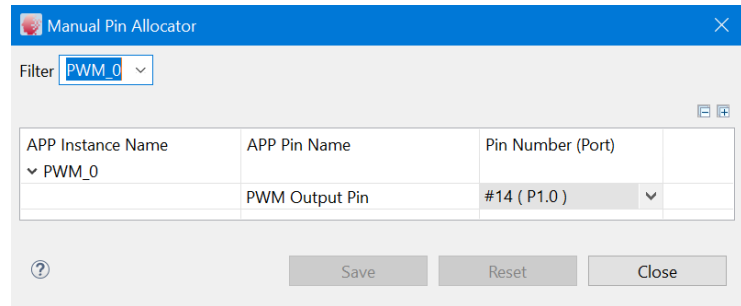
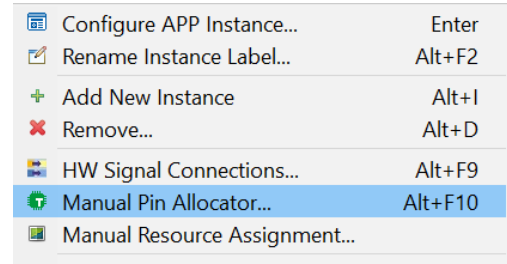
### Configure the PWM APP:

1. Right-click on PWM 0 in the App Dependency Tree view:
2. Select Configure APP Instance as shown here:
3. The PWM 0 View is created and opened:
4. Set Frequency to 1 Hz.
5. Select Start after Initialization:
6. Leave everything else at the default values.



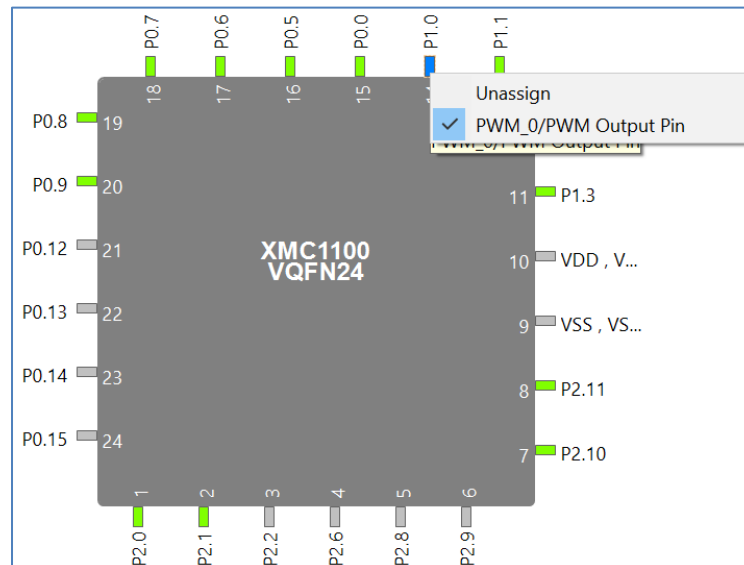
### Connect PWM 0 output to GPIO Port Pin P1.0:

1. Right click on PWM 0 again and select Manual Pin Allocator:
2. The Manual Pin Allocator view opens up:
3. Select Pin Number # 14 P1.0 as shown here:
4. Select Save.
5. Select Close to assign this pin.



### Verify Pin Selection:

1. Select the PIN perspective in the upper right side of the DAVE GUI.
- OR
2. Select Open Perspective: Select Pin Mapping and select Close
3. The Package View opens as shown below and you can confirm Pin P1.0 is selected by its blue colour.
4. You can select or unselect this pin by right clicking on a pin and select your choice.
1. The PWMs are now completely configured. All we need do is Build the project and export it into µVision 5.
2. Select the DAVE CE perspective to continue:



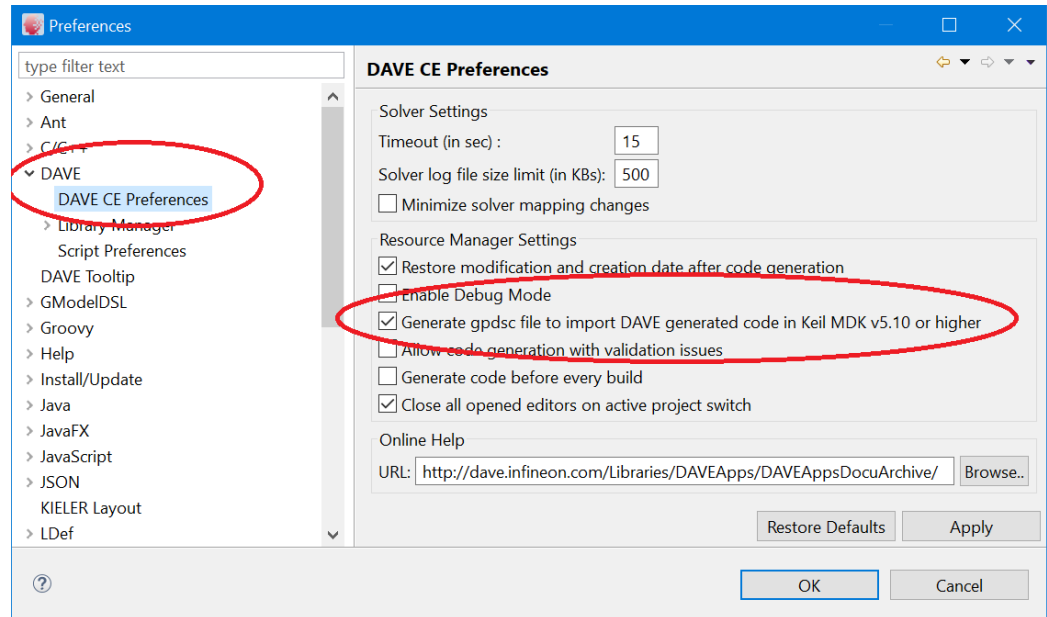


### Generator Package Description (gpdsc) File:


DAVE produces a complete  $\mu$ Vision project and in addition a gpdsc file. This file is used to start, import and configure the appropriate files into  $\mu$ Vision.

### 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.



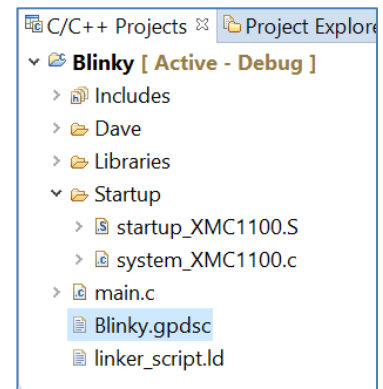
### Generate the Code:

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 now be visible in the window as shown here:

This is the file that  $\mu$ Vision will import and create a standard uvprojx project file.

It will be located in C:\Workspaces\DAVE-4.4.2-64Bit\Blinky\.

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



### Import into $\mu$ Vision:

1. Close  $\mu$ Vision if running. Otherwise a second instance will start.
2. Double click on Blinky.gpdsc either inside DAVE or with Microsoft Explorer.
3.  $\mu$ Vision will start and Blinky.gpdsc will be imported and Blinky.uvprojx is created.

- **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,  $\mu$ Vision can work stand alone with the files DAVE created for it.

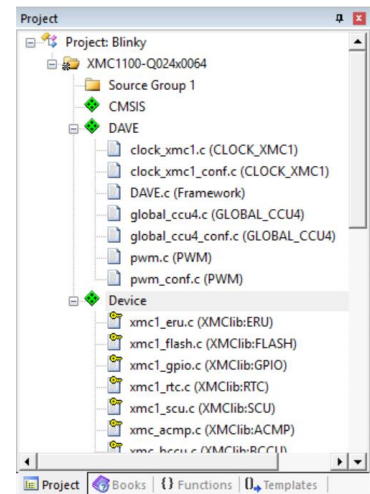
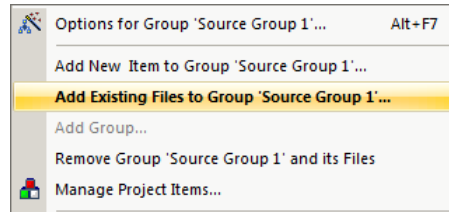
**On the next page, we will configure  $\mu$ Vision, build the project, program it to Flash and run it.**

## 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 ....:

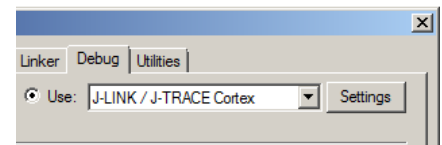


2. In the Add Files window, highlight main.c and select Add and then Close.
3. main.c will now be added to Source Group 1.
4. Select File/Save All or

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

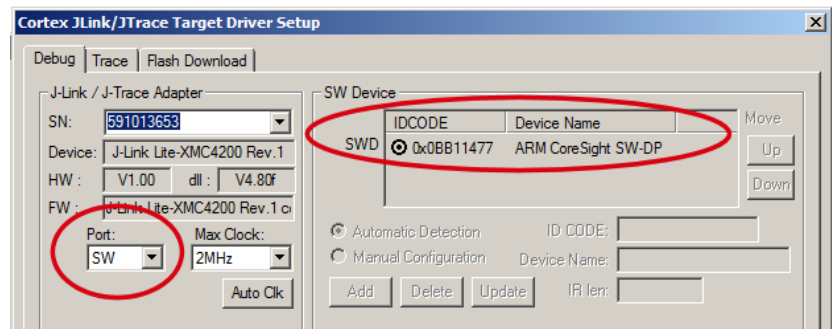
## Configure the Debug Adapter and Flash programming:

1. Plug your 2Go board to your PC with a USB cable.
2. In µVision, select the Options for Target Options icon: or ALT-F7.
3. Select the Target tab. Select Use MicroLIB to compile for smaller program size: ☒ Use MicroLIB
4. Select the Debug tab.
5. Select J-Link / J-Trace Cortex as shown here:
6. Select Run to Main: ☒ Run to main()
7. Select Settings: on the right side of this window.
8. Select SW and not JTAG as shown here in the Port: box:
9. 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 2Go board is plugged in. Ensure you have the correct J-Link USB drivers installed.


10. The SN: box indicates µVision is connected to the J-Link Lite. The SW Device box indicates µVision is also connected to the XMC1100 processor CoreSight debug module.
11. Click on OK to leave this window.
12. Select Settings. If the correct Flash algorithm is not displayed, select Add and highlight the correct Flash algorithm as shown below and select Add again to select it:
13. Click on OK twice to close and return to the main µVision window.
14. Select File/Save All or





Programming Algorithm			
Description	Device Size	Device Type	Address Range
XMC1100 64kB Flash	64k	On-chip Flash	10001000H - 10010FFFH

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

## Build the $\mu$ Vision 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 two warnings that you can ignore.

2. Enter Debug mode by clicking on the Debug icon.  The Flash will be programmed.
3. Click on the RUN icon.  The LED will blink at 1 second intervals.

### Concept: Software Development Information Notes:

During your development cycle, you will add your own source files to the  $\mu$ Vision 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.





You should not modify any files with  $\mu$ Vision that were created by DAVE. These are located under the DAVE3 group name in  $\mu$ Vision. Use DAVE to make any modifications to these files.

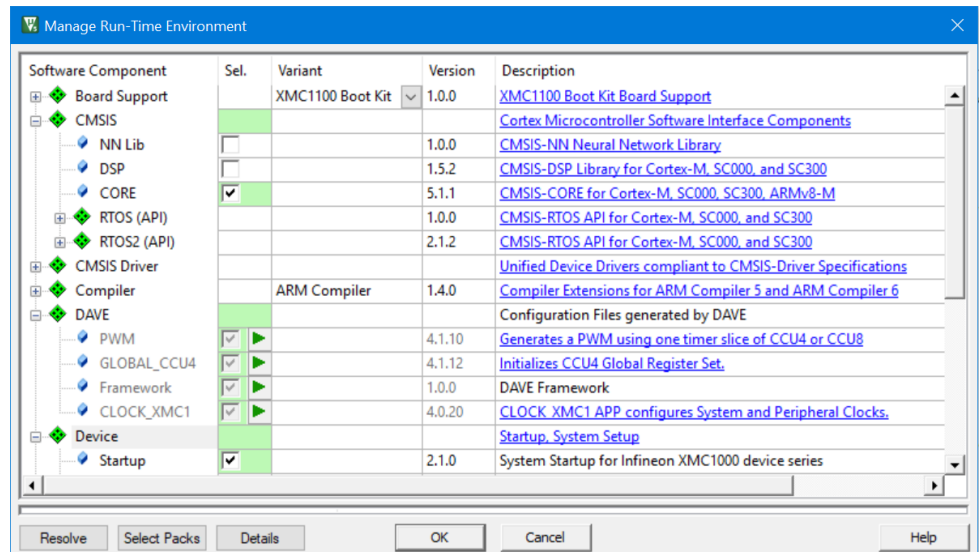
To change anything you created in DAVE, such as a peripheral configuration, make modifications or additions with DAVE and the updated files will be imported into  $\mu$ Vision. You must select YES when notified files have changed outside out of  $\mu$ Vision. If you had changed any DAVE created files inside  $\mu$ Vision, your changes will be lost.

At this point, if you do not yet to change anything in DAVE, you can close it and select your  $\mu$ Vision project directly from your DAVE workspace rather than double-clicking on the gpds.c file.

The LED will blink at 1 second intervals. The LED is controlled by the PWM you created in DAVE.





## Manage Run Time Environment:

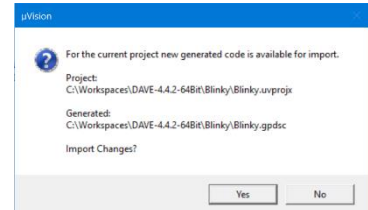
2. Stop the program.  Exit Debug mode. 
3. Click on the Manage Run-Time Environment icon:  The window below opens up.
  1. This window displays the components of your project from  $\mu$ Vision.
  2. The components created by DAVE are listed under DAVE3. These cannot be changed in this menu.
  3. The 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.
  4. The CMSIS CORE and Startup files were also created by DAVE.
  5. 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  $\mu$ Vision.

1. In  $\mu$ Vision, stop the program.  and exit Debug mode.  Do not exit  $\mu$ Vision.
2. In the APP Dependency view, right click on one of the PWM\_0 and select Configure APP Instance.
3. Change the 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  $\mu$ Vision back into focus.
6. There will one or more indications that files have changed: Click on Yes to import these changes. Files modified by DAVE are imported into  $\mu$ Vision.
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. 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.

### Create an RTX\_Blinky project using DAVE:

To add RTX to a DAVE project, please see the directions on page 22 Creating Your Own RTX MDK 5 project: This is not a hard task to accomplish.

## 1) Document Resources:

See [www.keil.com/infineon](http://www.keil.com/infineon)

### Books:

1. **NEW!** Getting Started MDK 5: Obtain this free book here: [www.keil.com/gsg/](http://www.keil.com/gsg/).
2. There is a good selection of books available on ARM: [www.arm.com/support/resources/arm-books/index.php](http://www.arm.com/support/resources/arm-books/index.php)
3.  $\mu$ Vision contains a window titled Books. Many documents including data sheets are located there.
4. The Definitive Guide to the Arm Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
5. The Definitive Guide to the Arm Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
6. **Embedded Systems: Introduction to Arm Cortex-M Microcontrollers** (3 volumes) by Jonathan Valvano
7. MOOC: Massive Open Online Class: University of Texas: <http://users.ece.utexas.edu/~valvano/>

### Application Notes:

1. **NEW!** Arm Compiler Qualification Kit: Compiler Safety Certification: [www.keil.com/safety](http://www.keil.com/safety)
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions [www.keil.com/appnotes/files/apnt209.pdf](http://www.keil.com/appnotes/files/apnt209.pdf)
3. CAN Primer using Keil MCB1700: [www.keil.com/appnotes/files/apnt\\_247.pdf](http://www.keil.com/appnotes/files/apnt_247.pdf)
4. Segger emWin GUIBuilder with  $\mu$ Vision™ [www.keil.com/appnotes/files/apnt\\_234.pdf](http://www.keil.com/appnotes/files/apnt_234.pdf)
5. Porting mbed Project to Keil MDK™ [www.keil.com/appnotes/docs/apnt\\_207.asp](http://www.keil.com/appnotes/docs/apnt_207.asp)
6. MDK-ARM™ Compiler Optimizations [www.keil.com/appnotes/docs/apnt\\_202.asp](http://www.keil.com/appnotes/docs/apnt_202.asp)
7. GNU tools (GCC) for use with  $\mu$ Vision <https://launchpad.net/gcc-arm-embedded>
8. GCC Arm Developer: <https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads>
9. Barrier Instructions <http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html>
10. Cortex-M Processors for Beginners: <http://community.arm.com/docs/DOC-8587>
11. Lazy Stacking on the Cortex-M4: [www.arm.com](http://www.arm.com) and search for DAI0298A
12. Cortex Debug Connectors: [www.keil.com/coresight/coresight-connectors](http://www.keil.com/coresight/coresight-connectors)
13. Sending ITM printf to external Windows applications: [www.keil.com/appnotes/docs/apnt\\_240.asp](http://www.keil.com/appnotes/docs/apnt_240.asp)
14. **NEW!** Migrating Cortex-M3/M4 to Cortex-M7 processors: [www.keil.com/appnotes/docs/apnt\\_270.asp](http://www.keil.com/appnotes/docs/apnt_270.asp)
15. **NEW!** ARMv8-M Architecture Technical Overview [www.keil.com/appnotes/files/apnt\\_291.pdf](http://www.keil.com/appnotes/files/apnt_291.pdf)
16. **NEW!** Determining Cortex-M CPU Frequency using SWV [www.keil.com/appnotes/docs/apnt\\_297.asp](http://www.keil.com/appnotes/docs/apnt_297.asp)

### Keil Tutorials for Infineon Boards:

[www.keil.com/infineon](http://www.keil.com/infineon)

1. XMC 2Go (this document) [www.keil.com/appnotes/docs/apnt\\_260.asp](http://www.keil.com/appnotes/docs/apnt_260.asp)
2. 1200 Boot Kit: *coming...* See [www.keil.com/infineon](http://www.keil.com/infineon)
3. XMC4500 Relax Kit: *coming...* See [www.keil.com/infineon](http://www.keil.com/infineon)
4. XMC4500 Application Board Kit: *coming...* See [www.keil.com/infineon](http://www.keil.com/infineon)
5. Hitex XMC-HiLight XMC4500 board: [www.keil.com/appnotes/docs/apnt\\_231.asp](http://www.keil.com/appnotes/docs/apnt_231.asp)

**Sales In Americas:** [sales.us@keil.com](mailto:sales.us@keil.com) or 800-348-8051. **Europe/Asia:** [sales.intl@keil.com](mailto:sales.intl@keil.com) +49 89/456040-20

**Keil Distributors:** See [www.keil.com/distis/](http://www.keil.com/distis/) **DS-5 Direct Sales Worldwide:** [orders@arm.com](mailto:orders@arm.com)

**Keil Technical Support** in USA: [support.us@keil.com](mailto:support.us@keil.com) or 800-348-8051. Outside the US: [support.intl@keil.com](mailto:support.intl@keil.com)

**Global Inside Sales Contact Point:** [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com)

**ARM Community Forums:** [www.keil.com/forum](http://www.keil.com/forum) and <http://community.arm.com/groups/tools/content>

**Infineon Community Forum:** [www.infineonforums.com/](http://www.infineonforums.com/)

**ARM University program:** [www.arm.com/university](http://www.arm.com/university). Email: [university@arm.com](mailto:university@arm.com)

**mbed™:** <http://mbed.org> ARM CMSIS standard: [www.arm.com/cmsis](http://www.arm.com/cmsis)



## 2) Keil Products and contact information: See [www.keil.com/Infineon](http://www.keil.com/Infineon)

### Keil Microcontroller Development Kit (MDK-ARM™) for NXP processors:

- **MDK-Lite™** (Evaluation version) up to 32K Code and Data Limit - \$0
- **New MDK-ARM-Essential™** For all Cortex-M series processors – unlimited code limit
- **New MDK-Plus™** MiddleWare Level 1. ARM7™, ARM9™, Cortex-M, SecureCore®.
- **New MDK-Professional™** MiddleWare Level 2. For details: [www.keil.com/mdk5/version520](http://www.keil.com/mdk5/version520).

For the latest MDK details see: [www.keil.com/mdk5/selector/](http://www.keil.com/mdk5/selector/)

Keil Middleware includes Network, USB, Graphics and File System. [www.keil.com/mdk5/middleware/](http://www.keil.com/mdk5/middleware/)

### USB-JTAG/SWD Debug Adapters (for Flash programming too)

- **ULINK2** - (ULINK2 and ME - SWV only – no ETM) **ULINK-ME** is equivalent to a ULINK2.
- **New ULINKplus-** Cortex-Mx High performance SWV & power measurement.
- **ULINKpro** - Cortex-Mx SWV & ETM instruction trace. Code Coverage and Performance Analysis.
- **ULINKpro D** - Cortex-Mx SWV no ETM trace ULINKpro also works with Arm DS-5.

For Serial Wire Viewer (SWV), a ULINK2, ULINK-ME, ULINKplus or a J-Link is needed. For ETM support, a ULINKpro is needed which also provides SWV.

Call Keil Sales for more details on current pricing. All products are available.

All software products include Technical Support and Updates for 1 year. This can easily be renewed.

### Keil RTX™ Real Time Operating System: [www.keil.com/rtx](http://www.keil.com/rtx)

- RTX is provided free as part of Keil MDK. It is the full version of RTX.
- No royalties are required. It has a BSD or Apache 2.0 license.
- [https://github.com/ARM-software/CMSIS\\_5](https://github.com/ARM-software/CMSIS_5)
- RTX source code is included with all versions of MDK.
- Kernel Awareness visibility windows are integral to µVision.

For the entire Keil catalog see [www.keil.com](http://www.keil.com) or contact Keil or your local distributor. For Infineon support: [www.keil.com/Infineon](http://www.keil.com/Infineon)

Getting Started with DS-MDK: [www.keil.com/gsg/](http://www.keil.com/gsg/)



### For more information:

Sales In Americas: [sales.us@keil.com](mailto:sales.us@keil.com) or 800-348-8051. Europe/Asia: [sales.intl@keil.com](mailto:sales.intl@keil.com) +49 89/456040-20

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

Global Inside Sales Contact Point: [Inside-Sales@arm.com](mailto:Inside-Sales@arm.com) **Arm Keil World Distributors:** [www.keil.com/distis](http://www.keil.com/distis)

Forums: [www.keil.com/forum](http://www.keil.com/forum) <http://community.arm.com/groups/tools/content> <https://developer.arm.com/>



**Cortex**  
Intelligent Processors by ARM®

**arm KEIL**