

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_265.asp

Introduction:

The purpose of this lab is to introduce you to the Maxim Zeus Cortex™-M3 processor family using the ARM® Keil® MDK™ 5 toolkit featuring the IDE µVision®. This tutorial will use a Keil ULINK®2 debug adapter.

Keil provides a free 32K toolchain called MDK Lite. This document uses MDK 5.11. MDK includes a full version of Keil RTX™ RTOS. RTX source code is included with all versions of MDK.

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

Why Use Keil MDK ?

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

1. µVision IDE with Integrated Debugger, Flash programmer and the ARM Compiler/Assembler and Linker toolchain. MDK is a turn-key product with included examples and is easy to get running.
2. **RTOS:** A full feature RTOS called RTX is included with MDK and includes source code. It has a BSD type license.
3. An RTX Kernel Awareness window is updated in real-time. This displays Kernel and thread status information and more.
4. **DSP:** Keil provides free DSP libraries with source code. They are CMSIS-DSP compliant.
5. Choice of adapters: ULINK™2, ULINK-ME, ULINK_{pro} or Segger J-Link and J-Link Lite.
6. All Zeus CoreSight™ debugger features are supported in the µVision debugger.
7. Keil Technical Support is included for one year and is easily renewable. This helps you get your project completed faster and more reliably.
8. Keil supports many Maxim 8051 processors. See www.keil.com/dd.
9. MDK is compatible with FreeRTOS and all other RTOSs.
10. MDK is completely CMSIS compliant. See www.arm.com/cmsis for information about this ARM standard.



This document details these features:

1. Real-time Read update for Watch, Memory and RTX Tasks windows. Memory and SVD windows also have Write access while the program is running. These reads and writes are nearly always 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. System and Thread Viewer: a kernel awareness program for RTX RTOS that updates while the program is running.
4. MDK 5 Software Packs which offer easy selection of software components.
5. Creating MDK 5 projects from scratch including one using RTX.

Other Maxim processors supported:

MDK supports or can support other Maxim ARM processors. Contact Keil sales or tech support for more information.

Keil Sales In USA: sales.us@keil.com or 1-800-348-8051. Outside the US: sales.intl@keil.com or +49 89/456040-20

Keil Technical Support in USA: support.us@keil.com or 1-800-348-8051. Outside the US: support.intl@keil.com.

Index:

Part A: Obtaining and Installing MDK and the Examples:

1. Keil Software Download and Installation::	3
2. Complimentary μ Vision License:	3
3. Example Programs:	3
4. Getting Started MDK 5 book:	3
5. μ Vision Software Packs Download and Install Process:	4
6. Testing the ULINK2 Connection to the Maxim board:	5

Part B: Project Examples:

7. Hello Example Program:	6
8. Hardware Breakpoints:	6
9. Call Stack + Locals Window	7
10. Watch and Memory Windows:	8
11. How to View Local Variables in Watch or Memory Windows:	9
12. Access Breakpoints (Watchpoints): Conditional Breakpoints	10
13. RTX_Blinky example program with Keil RTX RTOS:	11
14. RTX Kernel Awareness:	12
15. Call Stack for RTX_Blinky:	13
16. Creating your own MDK 5 project from scratch:	14
17. Creating your own MDK 5 RTX project from scratch:	17

Appendix:

18. Document Resources:	18
19. Keil Products and contact information:	19

Part A: Obtaining and Installing MDK and the Examples:

1) Keil Software Download and Installation:

1. Download MDK 5.11 or later from the Keil website. www.keil.com/mdk5/install
2. Install MDK into the default directory. You can install into any directory, but this lab uses the default C:\Keil_v5
3. We recommend you use the default examples directories for this tutorial. We will use C:\MDK\ for the examples.
4. If you install MDK into a different directory, you will have to adjust for the directory differences.
5. You need an external debug adapter such as the Keil ULINK2 or a ULINKpro. A Segger J-Link will also work.

Download MDK-Core Version 5

MDK 5 vs MDK 4: MDK 5 does not include processor support files such as headers, peripheral, Flash programming and example files. These are downloaded and installed for each processor family using Software Packs. This provides many advantages.

If you need to run MDK 4 projects, download and install the Legacy Support: www2.keil.com/mdk5/legacy.

With Legacy Support installed, MDK 5 will run both MDK 4 and MDK 5 programs.

2) The ULINK2 Debug Adapter:

The ULINK2 adapter is used exclusively in this lab. You must install the provided 10 pin Coresight cable in the ULINK as shown on the first page. You can use a ULINKpro. Page 5 contains a test for a debug adapter.

The Signum ADM-51 will not work with a Cortex-M processor, only with 8051 devices.

3) Example Programs:

The example programs are provided in the MDK 5.11 Software Packs. The next page describes where to install these examples.

4) Getting Started MDK 5:

Obtain this useful book here: www.keil.com/mdk5/.

More Information and Keil Contacts:

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



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

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

5) µVision Software Packs Download and Install Process:

The entire project and required files are contained in a Software Pack. A Pack is normally distributed on a website and accessed with the Pack Installer utility in µVision. A .pack file can also be imported manually into µVision.. A Pack is a zip file renamed with a .pack file extension. This enables µVision to recognize it. If you double-click on a pack file, µVision will install it. You can also select File/Import from the Pack Installer main menu.

1) Start µVision, open Pack Installer:


1. Install MDK 5.11 or later. www.keil.com/mdk You do not need a license for this tutorial.
2. For this tutorial, you need to be connected to the internet (it will display ONLINE in the bottom right corner in the Pack Installer utility in µVision. See the first screen below).
3. Start µVision by clicking on its desktop icon. 
4. The Pack Installer window shown below will open: If not, open it with its icon: 
5. If the Pack Installer Welcome window opens, please read it and close it.

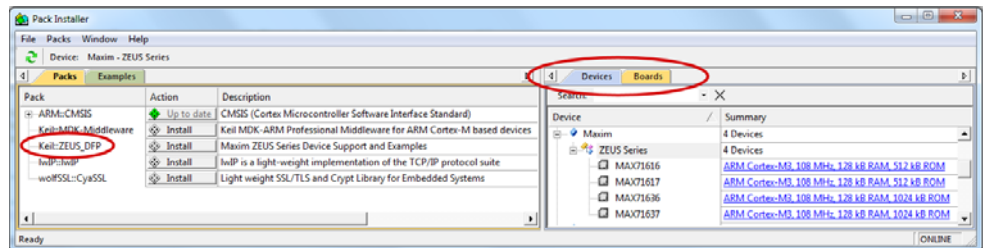
2) Download the Maxim Zeus Software Pack: *Note: you can also locally import a .pack file. Select File/Import.*

1. In the Devices tab, select Maxim/Zeus **or** in the Boards tab select DB-MAX71637. This filters the Packs tab.
2. In the Packs tab, select Install for Keil::Zeus_DFP. This pack will be downloaded and installed from the web.
3. A successful install is indicated by the Up to date icon:




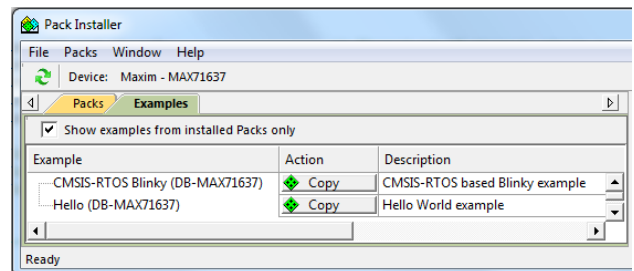
TIP: You can refresh the Packs

list from the web with  or local files with File/Refresh.




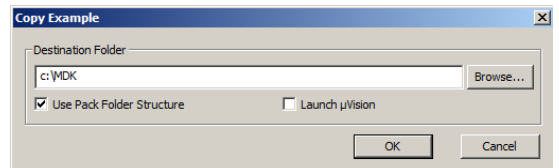
3) Copy the Hello and CMSIS RTOS Examples:

1. Select the Examples tab. There are two examples provided in this Software Pack as shown here:
2. Select CMSIS-RTOS Copy  as shown:
3. The Copy Example window opens up: Unselect Launch µVision. Select Use Pack Folder Structure as shown:
4. Type in C:\MDK for the Destination Folder.
5. Click OK to copy the RTX_Blinky project to C:\MDK\Boards\Maxim\DB-MAX71637.
6. Repeat for the Hello example.



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

7. Close the Packs Installer. You can open it any time by clicking on its icon. 


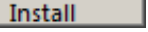

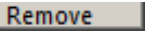

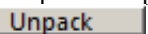
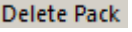

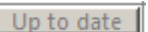

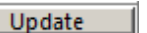

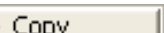


TIP: An Update icon means there is an updated Software Pack available for download. 



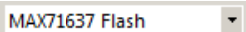

TIP: If you look in the directory C:\Keil_v5\ARM\Pack\Keil\ZEUS_DFP\1.0.0\Boards\Maxim\DB-MAX71637, you will find the RTX_Blinky and Hello projects. This is the read-only version you downloaded used for backup purposes. Use only the projects you copied over from the Examples tab to the directory you chose: in this tutorial we have used C:\MDK.

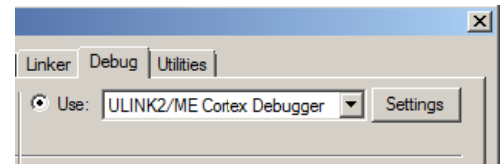
The next page has a few notes on Software Packs Maintenance:

Software Packs Maintenance Notes:

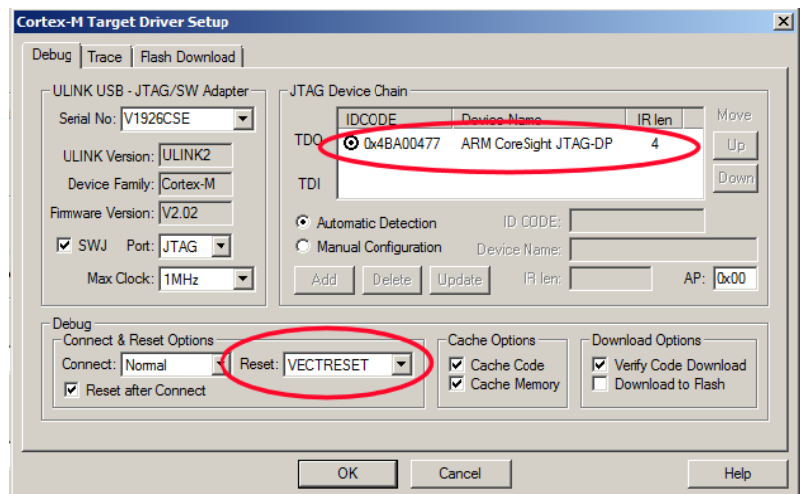
1. Software Packs can be downloaded and installed on your computer with the Install icon.  
2. You can delete a Pack by double-clicking on it. A Remove icon appears. Click on this.  
3. The Pack is now removed but the original .pack compressed file you downloaded is still present. You can reinstall the 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 a Pack is current, the Up to date icon will display:  
6. If an updated Pack is available, the Update icon appears. You can download this update (or not).  
7. Example files, if provided by a Pack, can be copied into a directory of your choice.  

6) Testing the ULINK Connection:

1. Start   if it is not already running. Select Project/Open Project.
2. Connect a ULINK2 to your Maxim board as shown on the first page of this tutorial.
3. Select the project C:\MDK\Boards\Maxim\DB-MAX71637\Hello\Hello.uvprojx. Any valid Maxim project will work for this test.
4. Select "MAX71637 Flash" in the Select Target menu: 
5. Select Target Options  or ALT-F7 and select the Debug tab:
6. Choose your debug adapter: here we have chosen the ULINK2: You can also choose ULINKpro or J-Link.
7. Click on Settings: and the window below opens up: If an ICODE and Device name is displayed, ULINK2 is working. Skip to Step 11 to continue with this tutorial. If not, see the next step.
8. Select the SWJ box. In the Port: box select JTAG.
9. If nothing or an error is displayed in this JTAG Device Chain box, this **must** be corrected before you can continue. Usual problems are no power connected to the board or ULINK. If you see a proper display as shown, your ULINK2 is installed properly and μ Vision is connected to the CoreSight debug module in the Maxim processor. ULINK2 uses the USB HID interface. The ULINKpro use USB2.
10. A number in the SN: box means μ Vision is successfully connected to the ULINK2 adapter and not necessarily to the Cortex-M3 core.
11. Select VECTRESET in the Reset: box. If you get weird problems with your projects, check this setting.
12. Click on OK twice to return to the μ Vision main menu.









TIP: To refresh the JTAG Device Chain box: in the Port: box select SW and then select JTAG again. You can also exit then re-enter this window. The Maxim Cortex-M3 currently works only with JTAG and not SW. But, this is a useful and quick way to refresh the JTAG setting.



7) Hello Example Program:

We will connect a Keil MDK development system using the Maxim MAX71637 evaluation board. This project is pre-configured to use a ULINK2. You can configure μ Vision to use a ULINK pro or Segger J-Link.

1. Connect a ULINK2 to your Maxim board as shown on page one.
2. Start μ Vision by clicking on its desktop icon. 
3. Select Project/Open Project. Open the project file: C:\MDK\Boards\Maxim\DB-MAX71637\Hello\Hello.uvprojx
4. Compile the source files by clicking on the Rebuild icon.  You can also use the Build icon beside it.
5. Program the Flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
6. Connect a USB cable to CN1 to your PC.
7. Configure a terminal program such as PuTTY to the appropriate virtual COM port and messages will be displayed. Specifications are 115,200 baud, 8,1. Select Serial Port in the Windows Device Manager under Ports (COM).
8. Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
Note: μ Vision can be configured to program the Flash when entering Debug mode. You do not have to use Load.
9. Click on the RUN icon.  Note: you stop the program with the STOP icon. 

The program will run and Hello World is printed on the COM port.

Now you know how to compile a program, load it into the Maxim 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.

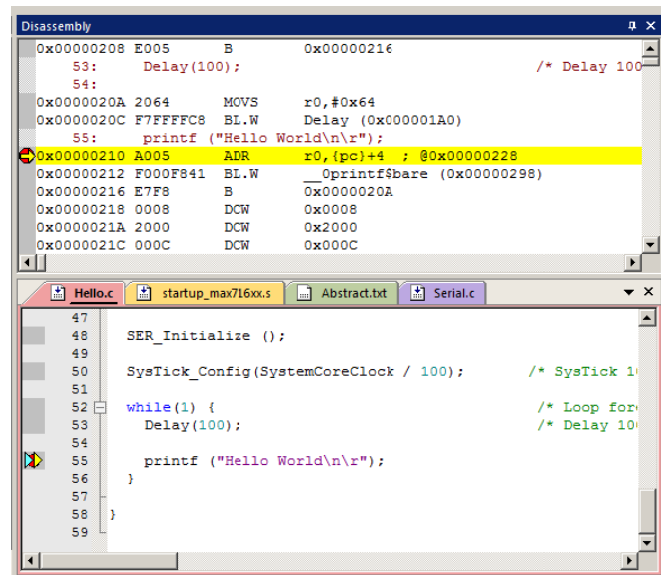
8) Hardware Breakpoints:

1. With Blinky running, click in the left margin on a darker gray block inside the while(1) loop in Hello.c.
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 with ARM CoreSight technology.
6. The MAX71600 series has 6 hardware breakpoints. A breakpoint does not execute the instruction it is set to. This is a very important feature for effective debugging.
7. If you set too many breakpoints, μ Vision will notify you. Sometimes μ Vision will use one of the available breakpoints for one of its internal operation such as single stepping or Run to main..
8. Remove any breakpoints you have set. You can select Ctrl-B and select Kill All or click on the breakpoint red circle.

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: For smaller programs, enable Use MicroLIB under the Target tab in Options for Target. If your program is allowed to use MicroLIB, the compiler will not generate any errors.



9) Call Stack + Locals Window:

Local Variables:

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

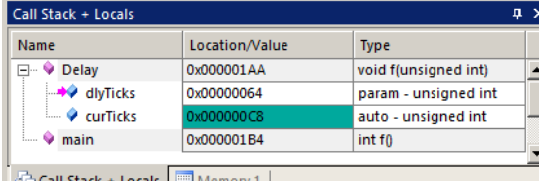
1. Open the Call Stack + Locals window by clicking on its tab.
2. Click on the RUN icon  Then, stop the program. 
3. main and the Delay function will be displayed as shown here with any local variables and their values. The program spends most of its time in the Delay function so it is likely to be running when you stop the program.

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 first called function is at the bottom of this table.

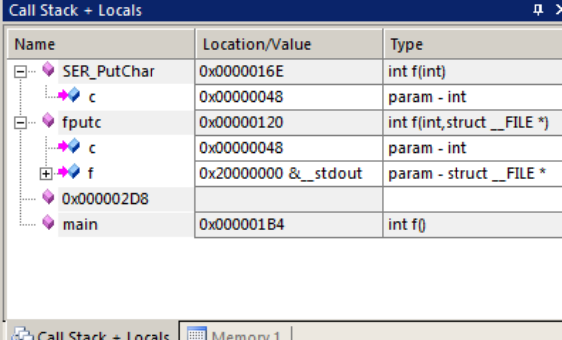
This table is active only when the program is stopped.



Name	Location/Value	Type
Delay	0x000001AA	void f(unsigned int)
dlyTicks	0x00000064	param - unsigned int
curTicks	0x000000C8	auto - unsigned int
main	0x000001B4	int f()

4. Click on the StepOut icon  (Ctrl-F11) to exit the Delay function to return to main().
5. Set a breakpoint in Serial.c in the SER_PutChar function near line 61.
6. Click on the RUN icon  The program will soon stop here and the window below opens.
7. Note the various functions that are now active and their local variable values.
8. Each time you click on RUN, these variables are updated as appropriate.

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



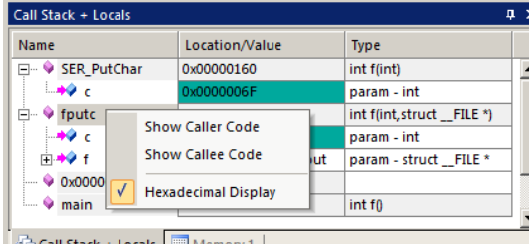
Name	Location/Value	Type
SER_PutChar	0x0000016E	int f(int)
c	0x00000048	param - int
fputc	0x00000120	int f(int, struct __FILE *)
c	0x00000048	param - int
f	0x20000000 & _stdout	param - struct __FILE *
0x000002D8		
main	0x000001B4	int f()

Call Stack:

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

9. 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.
10. **Remove all breakpoints when you are done.** You can click on them individually or Ctrl-B and select Kill All.

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.



Name	Location/Value	Type
SER_PutChar	0x00000160	int f(int)
c	0x0000006F	param - int
fputc		int f(int, struct __FILE *)
c		param - int
f		param - struct __FILE *
0x0000		
main		int f()

Context menu options:



- Show Caller Code
- Show Callee Code
- Hexadecimal Display (checked)

10) 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 20 in Hello.c:



```
unsigned int counter = 0;
```



3. Add the statements near line 54 just Delay(100);

```
counter++;
```

```
if (counter > 0xF) counter = 0;
```

4. Select File/Save All.

5. Click on Rebuild  and program the Flash with Load .

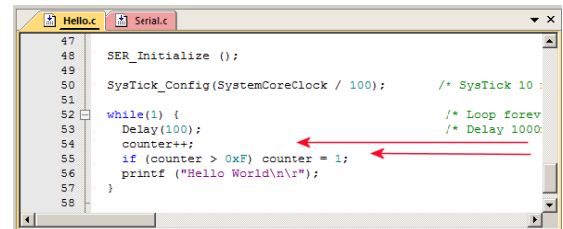
6. Enter Debug mode.  Click on RUN . You can configure a Watch window while the program is running. You can also do this with a Memory window.

7. Select View and select Periodic Window Update if necessary:

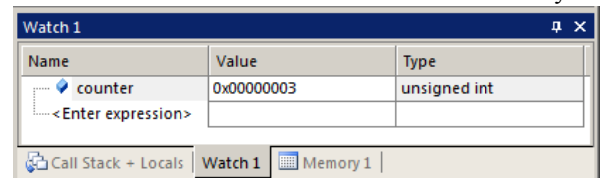
☒ Periodic Window Update

8. In Blinky.c, right click on **counter** and select Add counter to ... and select Watch 1. Watch 1 will automatically open. **counter** will be displayed as shown here:

9. **counter** will update in real time.



```
47
48 SER_Initialize ();
49
50 SysTick_Config(SystemCoreClock / 100); /* SysTick 10
51
52 while(1) {
53     Delay(100);
54     counter++;
55     if (counter > 0xF) counter = 1;
56     printf ("Hello World\n\r");
57 }
58
```



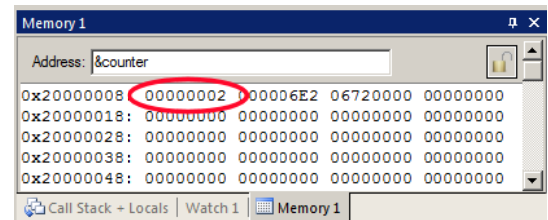
Name	Value	Type
counter	0x00000003	unsigned int
<Enter expression>		

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

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

Memory window:

1. Right click on **counter** and select Add counter to ... and select the Memory 1 window.
2. Note the value of **counter** is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to: but this not what we want to see at this time.
3. Add an ampersand “&” in front of the variable name and press Enter. The physical address is (0x2000_0008).
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.







Address	Value
0x20000008	00000002
0x20000018	00000000
0x20000028	00000000
0x20000038	00000000
0x20000048	00000000

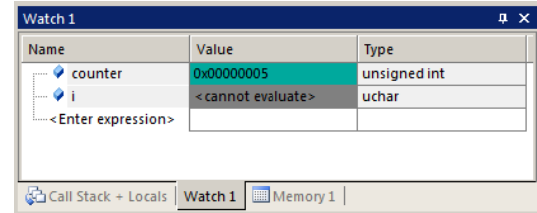
TIP: No CPU cycles are used to perform these operations. You can have more than one variable displayed. Structures can also be displayed and expanded.

TIP: To view variables and their location use the Symbol window. Select View/Symbol Window while in Debug mode.

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

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

1. Keep the program running: Start the program if it is not running. 
2. There is a local (automatic) variable `i` declared near line 34 in `Hello.c`: `uint32_t curTicks;`
3. Enter `curTicks` into Watch 1 window by right clicking on it and selecting Add `curTicks` to.... Watch 1.
4. Note it says <cannot evaluate> or “not in scope”.
5. Stop the program  and the value of `curTicks` will now display. This is because it is now in scope.
6. μ Vision is unable to determine the value of `curTicks` when the program is running because it exists only when the function Delay is running. It disappears in other functions and handlers.
7. Stop the program.  Exit Debug mode. 



Name	Value	Type
counter	0x00000005	unsigned int
i	<cannot evaluate>	uchar
<Enter expression>		







How to view local variables updated in real-time:

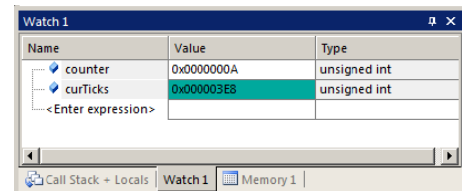
All you need to do is to make `curTicks` static where it is declared in `Hello.c` !

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

```
34      static uint32_t curTicks;
```


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

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






Name	Value	Type
counter	0x0000000A	unsigned int
curTicks	0x000003E8	unsigned int
<Enter expression>		

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

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

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


TIP: To program the Flash automatically when you enter Debug mode select Target Options , select the Utilities tab and select the “Update Target before Debugging” box. This means you can skip using the LOAD icon.  Programming the Flash will be automatically done when you enter Debug mode. .

How It Works:

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





12) Access Breakpoints: Conditional Breakpoints

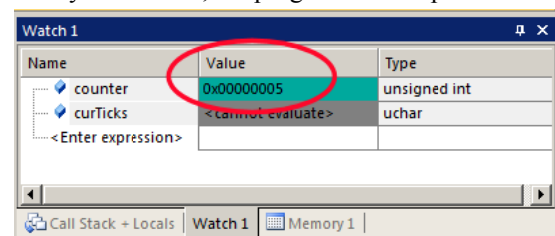
The Maxim Cortex-M3 processor has two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are referred to as Access Breaks in Keil documents.

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 μ Vision window and then select Breakpoints or press Ctrl-B.
4. Select Read in the Access box. (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.



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 (so things are interesting) in the Watch window.
10. Click on RUN. .
11. When the program detects a read or write access of 0x5 to `counter` as you selected, the program will stop. See Watch 1 shown here: 
12. Click on RUN and the program will run to the next read and/or write of 0x5 to counter.
13. Stop the CPU if it is running. 
14. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All and select Close.
15. Exit Debug mode. .



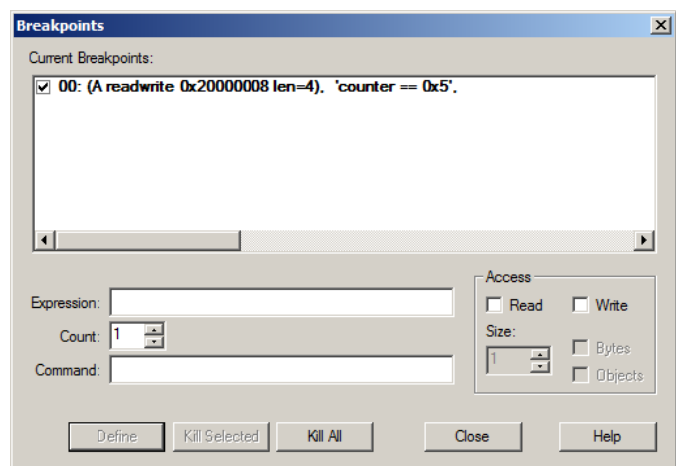
Name	Value	Type
counter	0x00000005	unsigned int
curTicks	<cannot evaluate>	uchar
<Enter expression>		

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

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

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

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



Current Breakpoints:
<input checked="" type="checkbox"/> 00: (A readwrite 0x20000008 len=4). 'counter == 0x5'.

Expression:

Count:

Command:







Access: ☐ Read ☐ Write

Size: ☐ Bytes ☐ Objects

Define Kill Selected Kill All Close Help

13) RTX_Blinky Example Program with Keil RTX RTOS: A Stepper Motor example

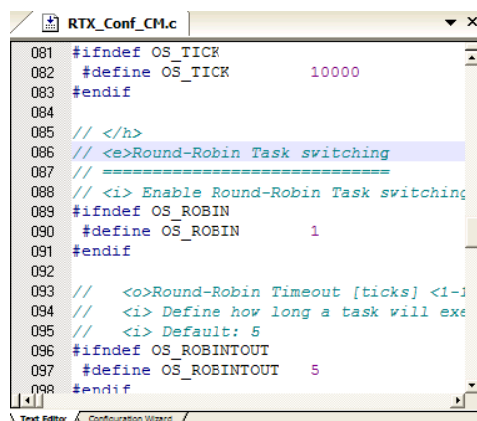
Keil provides RTX, a full feature RTOS. RTX is included as part of the Keil MDK tool suite. No royalty payments are required. RTX has a BSD type license. RTX with source code is provided with all MDK versions. See www.arm.com/cmsis and C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.1.0\CMSIS_RTX. Click on index.html. This example explores RTX. Keil will work with any RTOS. A RTOS is just a set of C functions that gets compiled with your project. A real-time awareness viewer for RTX is provided inside μ Vision.

1. Start μ Vision by clicking on its icon if not already running.  You must have installed the RTX_Blinky5 example previously. Instructions are provided on the bottom of page 4.
2. Select Project/Open Project and open C:\MDK\Boards\Maxim\DB-MAX71637\RTX_Blinky\Blinky.uvprojx.
3. Compile the source files by clicking on the Rebuild icon. . They will compile with no errors or warnings.
4. To program the Flash manually, click on the Load icon. . A progress bar will be at the bottom left.
5. Enter the Debug mode by clicking on the debug icon  and click on the RUN icon. 
6. The program is now running under the RTX operating system and on the next page we will illustrate this fact.
7. Click on STOP .

The Configuration Wizard for RTX:

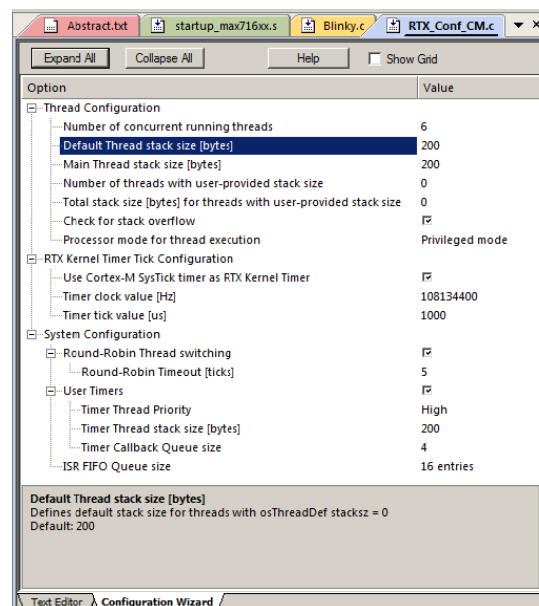
1. Click on the RTX_Conf_CM.c source file tab as shown below on the left below. You can open it with File/Open.
2. Click on Configuration Wizard at the bottom and your view will change to the Configuration Wizard.
3. Open up the individual directories to show the various configuration items available.
4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
5. This is a great feature as it is much easier changing items here than in the source code.
6. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
7. This scripting language is shown below in the Text Editor as comments starting such as a `</h>` or `<i>`. See www.keil.com/support/docs/2735.htm for instructions to add this feature to your own source code.

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



```
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
```


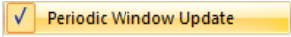
Text Editor: Source Code

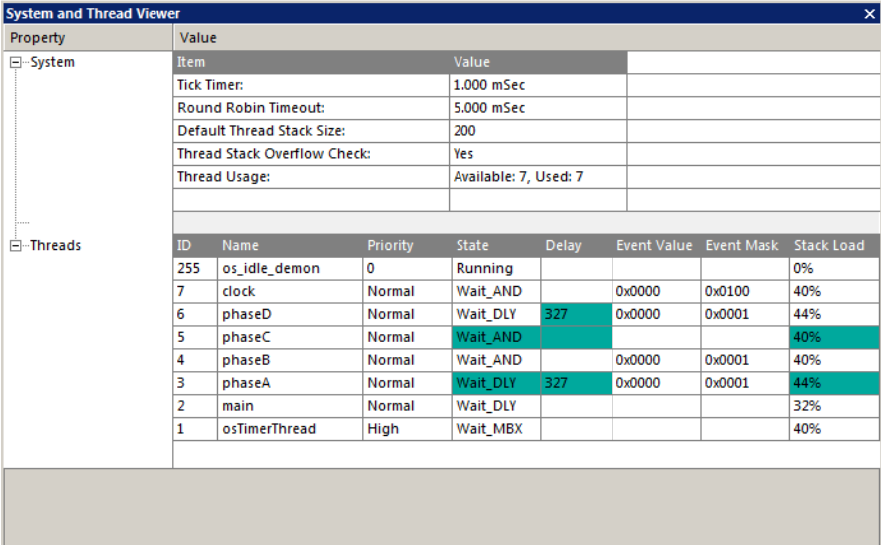


Configuration Wizard

14) 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 to display this information in real time, while the program is running. Other RTOS companies also provide awareness plug-ins for μ Vision.

1. Run RTX_Blinky by clicking on the Run icon. 
2. Open Debug/OS Support and select RTX 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. μ Vision also has an Event Viewer which displays RTX threads in a graphical format. The Maxim processors at this time do not support this feature.





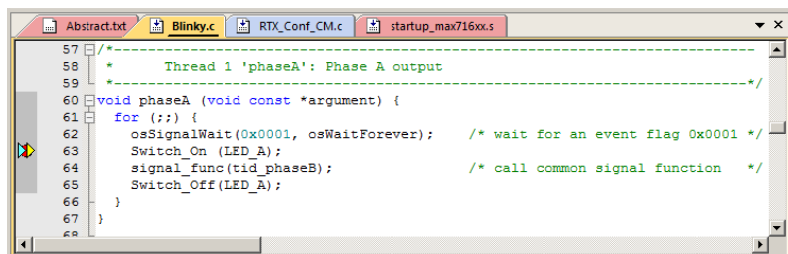
Property	Value
System	
Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 7

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
255	os_idle_demon	0	Running				0%
7	clock	Normal	Wait_AND		0x0000	0x0100	40%
6	phaseD	Normal	Wait_DLY	327	0x0000	0x0001	44%
5	phaseC	Normal	Wait_AND				40%
4	phaseB	Normal	Wait_AND		0x0000	0x0001	40%
3	phaseA	Normal	Wait_DLY	327	0x0000	0x0001	44%
2	main	Normal	Wait_DLY				32%
1	osTimerThread	High	Wait_MBX				40%

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


Blinky.c contains four threads that represent the stages of a stepping motor. Thread 1 (phaseA) is shown below:

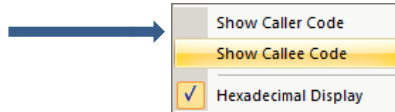
1. The gray areas opposite the line numbers indicate there is valid assembly code located here.
2. Set a breakpoint on one of these in Thread 1 as shown: (but not on the `for (;;)` line)
3. Set a breakpoint in one or two other similar threads.
4. Click on RUN .
5. When the program stops, this information will be updated in the RTX System and Thread Viewer window. The Task running when the program stopped will be indicated with a "Running" state.
6. Click on RUN . The other thread will show as "Running". Each time you click RUN, the next thread will run.




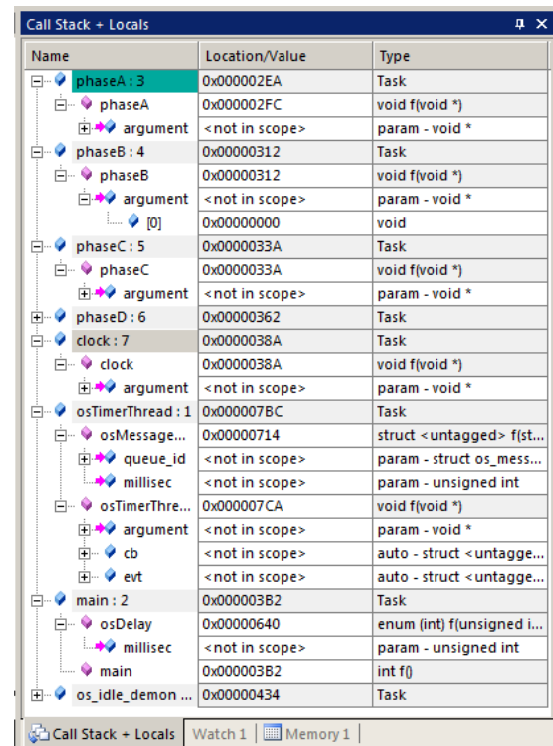
```
57  /*
58  *   Thread 1 'phaseA': Phase A output
59  */
60  void phaseA (void const *argument) {
61  for (;;) {
62  osSignalWait(0x0001, osWaitForever); /* wait for an event flag 0x0001 */
63  Switch_On (LED_A);
64  signal_func(tid_phaseB); /* call common signal function */
65  Switch_Off(LED_A);
66  }
67  }
```

15) Call Stack and Locals with RTX_Blinky:

1. Click on the Call Stack + Locals tab. This window opens up:
2. Each time you click on RUN  the 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.

More Information of obtaining and using RTX:

It is very beneficial to use an RTOS. RTX is a good choice. It is small, efficient and easy to use yet it is full featured. RTX source, various ports and documentation are here: C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.1.0\CMSIS_RTX.index.html is the entry point into the documentation.

There are two versions of RTX: The first comes with MDK 4.7x and earlier. The second comes with MDK 5.11 and later. This second one is CMSIS-RTOS compliant and this is the one you want to use.

Ports are available for ARM and GCC compilers and others.

This is the end of the stand-alone examples.

16) Creating your own MDK 5 project from scratch:

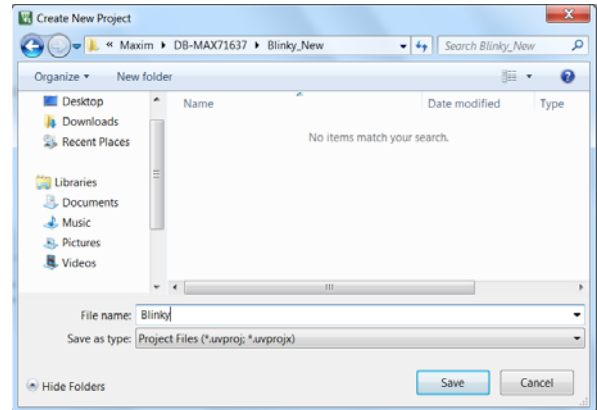
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a template for your own projects. However, we will start an example project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load 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 an RTOS.

Install the Maxim Zeus Software Pack for your processor:

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

Create a new Directory and a New Project:


1. Click on Project/New μ Vision Project...
2. In the window that opens, shown below, go to the folder C:\MDK\Boards\Maxim\DB-MAX71637\
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or highlight it and select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj in C:\MDK\Boards\Maxim\DB-MAX71637\BlinkyNEW.
7. As soon as you click on Save, the next window opens:

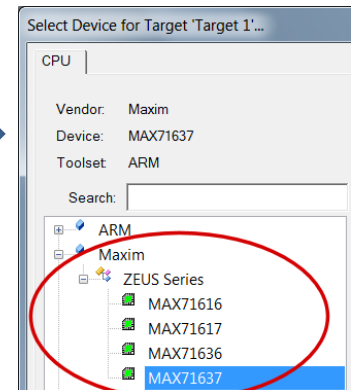


Select the Device you are using:

1. Expand Maxim, then Zeus Series, and then select MAX71637 as shown:
2. Click OK and the Manage Run Time window shown below bottom right opens.

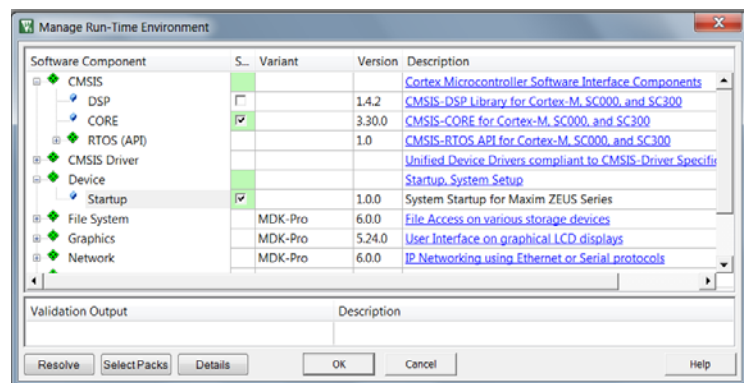
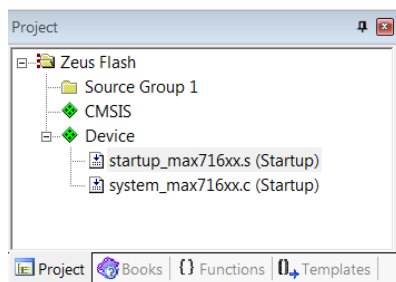
Select the CMSIS components you want:

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



What has happened to this point:


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

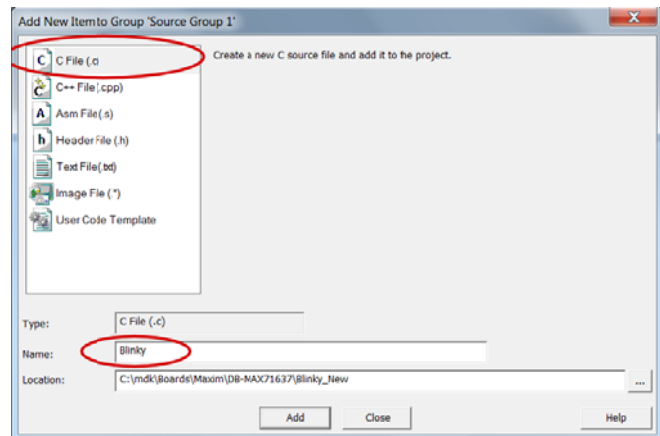


Create a blank C Source File:


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


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

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




Add Some Code to Blinky.c:


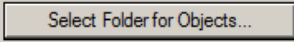
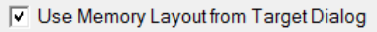
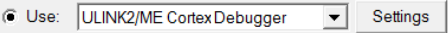



9. In the blank Blinky.c, add the C code below:
10. Click on File/Save All or 

11. Build the files.  There will be no errors or warnings if all was entered correctly.


```
#include "max716xx.h"
unsigned int counter = 0;
/*-----
  MAIN function
  -----*/
int main (void) {
    while(1) {
        counter++;
        if (counter > 0x0F) counter = 0;
    }
}
```

TIP: You can also add existing source files:  No need to at this time.

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






1. Select the Target Options icon . Select the **Target** tab.
2. Confirm 108 in Xtal (MHz). This is used for timing calculations. Select Use MicroLIB to optimize your code size.
3. Select the **Output** tab. Click on Select Folder for Objects...: 
4. In the Browse for Folder window that opens: right click and create a new folder called Flash.
5. Double click on Flash to enter this folder and click OK. Compilation files will now be stored in this Flash folder.
6. Click on the **Listings** tab. Click on Select Folder for Objects...: Double click on Flash and click OK to close.
7. Click on the **Linker** tab. Select Use memory Layout...: 
8. Click on the **Debug** tab. Select the ULINK2/ME Cortex Debugger: 
9. Select the Settings: icon.
10. Select JTAG as shown here in the Port: box:  If your board is connected to your PC, you **must** now see a valid IDCODE and Device Name in the JTAG Device Chain box.
11. Click on OK **once** to go back to the Target Configuration window. Otherwise, fix the connection problem.
12. Click on the **Utilities** tab. Select Settings and confirm the correct Flash algorithm: Shown is the correct one for the MAX716xx series processors: 
13. Click on OK twice to return to the main menu.
14. Click on File/Save All or 

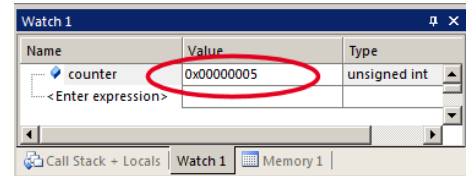
Programming Algorithm			
Description	Device Size	Device Type	Address Range
MAX716xx 1MB Flash	1M	On-chip Flash	00000000H - 000FFFFFH

15. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !

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

Running Your Program:


1. Program the Zeus Flash by clicking on the Load icon:  Progress will be indicated in the Output Window.
2. Enter Debug mode by clicking on the Debug icon. 
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.
5. Right click on counter in Blinky.c and select Add counter to ... and select Watch 1.
6. counter should 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 should now be able to add your own source code to create a meaningful project.

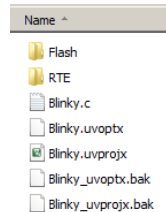


TIP: The Watch 1 is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip the sequential values you know must exist.

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

We modified the folder where the output and listings files are stored. This was in Steps 3 through 6 on the preceding page. If you did a Build before this was done, there will be files in your project root directory. We want them only in \Flash to keep things more organized.

1. Exit μ Vision. Otherwise, you can't delete files that it still has open.
2. Open Microsoft Explorer and navigate to:
C:\MDK\Boards\Maxim\DB-MAX71637\BlinkyNEW\.
3. Delete all files and folders except these: (you can delete Flash – a Build will recreate it.) 
4. You can also leave any backup or μ Vision files that identify your computer to retain your settings.
5. Restart μ Vision. Having all compilation files stored in the \Flash folder makes it cleaner.








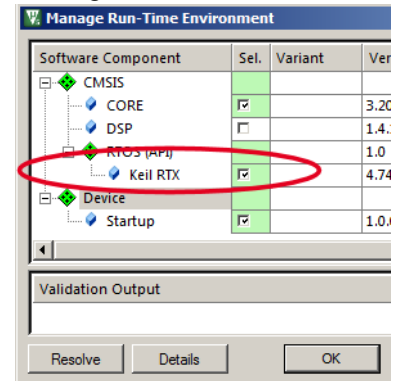
TIP: If you want to save or send the project files to someone, you can delete the folder Flash to reduce file size. This folder and its contents are easily reconstructed with a Build.

17) Creating your own RTX MDK 5 project from scratch:


The MDK Software Packs makes it easy to configure an RTX project. There are two versions of RTX: The first comes with MDK 4.7x and earlier. The second comes with MDK 5.11 and later. This second one is CMSIS-RTOS compliant.

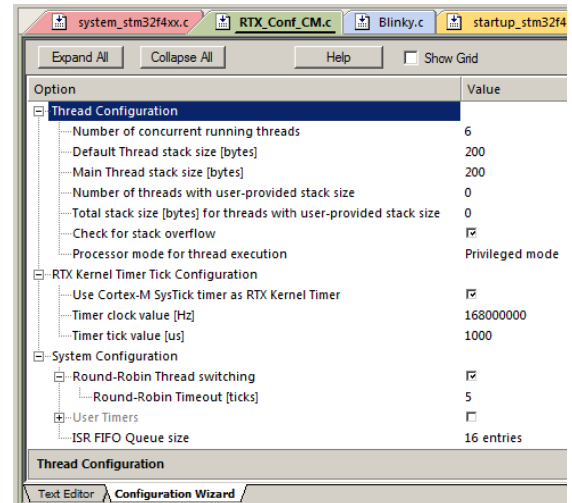
Configuring RTX is easy in MDK 5.11 and later. These steps use the configuration from the preceding Blinky example.

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







Configure RTX:

1. In the Project window, expand the CMSIS group.
2. Double click on RTX_Conf_CM.c to open it.
3. Select the Configuration Wizard tab: Select Expand All.
4. The window is displayed here: .
5. Set Timer clock value: to 108000000 as shown: (108 MHz)
6. Unselect User Timers. Use the defaults for the other settings.



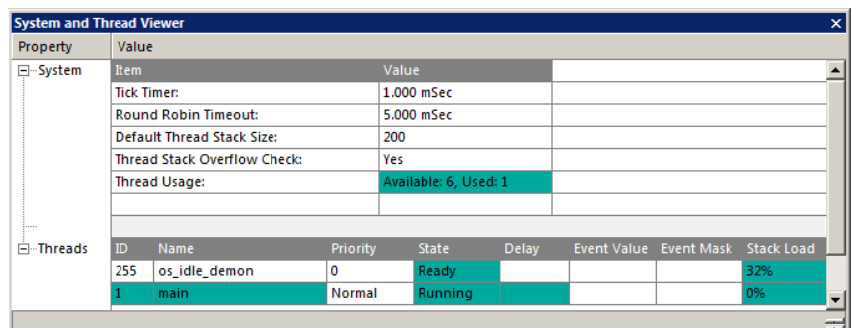
Build and Run Your RTX Program:

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

What you have to do now:

1. You must add the RTX framework into your code and create your threads to make this into a real RTX project configured to your needs.
2. See the RTX_Blinky example to use as templates and hints.
3. If you copy Blinky.c from the RTX_Blinky project, it will blink the LEDs. It has the RTX code incorporated into it.
4. **Getting Started MDK 5:** Obtain this useful book here: www.keil.com/mdk5/. It has very useful information on implementing and maintaining RTX.

This completes the exercise of creating your own RTX project from scratch.



18) Document Resources:

Books:

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

Application Notes:

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

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

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

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

mbed™: <http://mbed.org>

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

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

19) Keil Products and Contact Information:

Keil Microcontroller Development Kit (MDK-ARM™)

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

USB-JTAG adapter (for Flash programming too)

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

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

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

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

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

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

Keil includes free DSP libraries for all Cortex-M processors.

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

All products are available from stock.

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

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



For more information:

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

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

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

CMSIS documentation: www.arm.com/cmsis

