Tutorial

Version 1.8.0

ARM KEIL Microcontroller Tools

Creating a Middleware Application using CMSIS Components

The latest version of this document is here: developer.arm.com/docs/kan268

Abstract

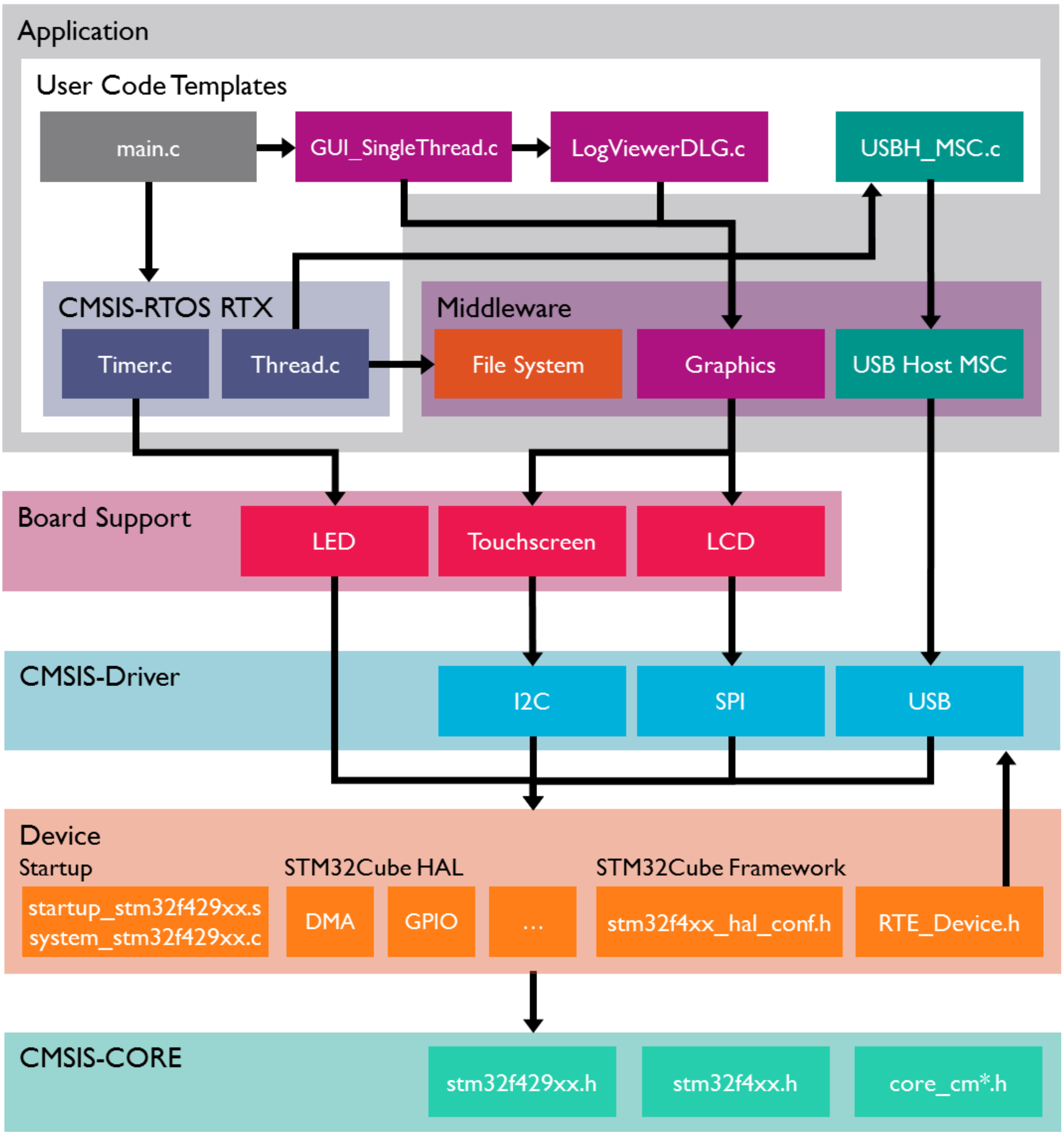
This tutorial shows how to read the contents of a text file from a USB memory stick attached to a development board. After pressing an update button on the touch screen, the content is shown on the LCD. The tutorial explains the required steps to create the application on an STM32F429I-Discovery board. Still, it can be easily ported to other underlying hardware using MDK-Professional Middleware, Keil RTX5 and CMSIS, the Cortex Microcontroller Software Interface Standard.

Contents

Introduction

This workshop explains creating a software framework for a sophisticated microcontroller application using CMSIS and Middleware components. During this workshop, a demo application is created that implements the following functions:

- Read the content of a *Test.txt* file from a USB memory stick.
- Provide an update button on a touch screen.
- Show the content on a graphical display.



Software Stack

The application is created by using user code templates. These templates are part of software components such as the Middleware, CMSIS-RTOS or the STM32F4xx Device Family Pack (DFP). Some other source files are automatically generated, such as the code that creates the graphical user interface (GUI) on the external display.

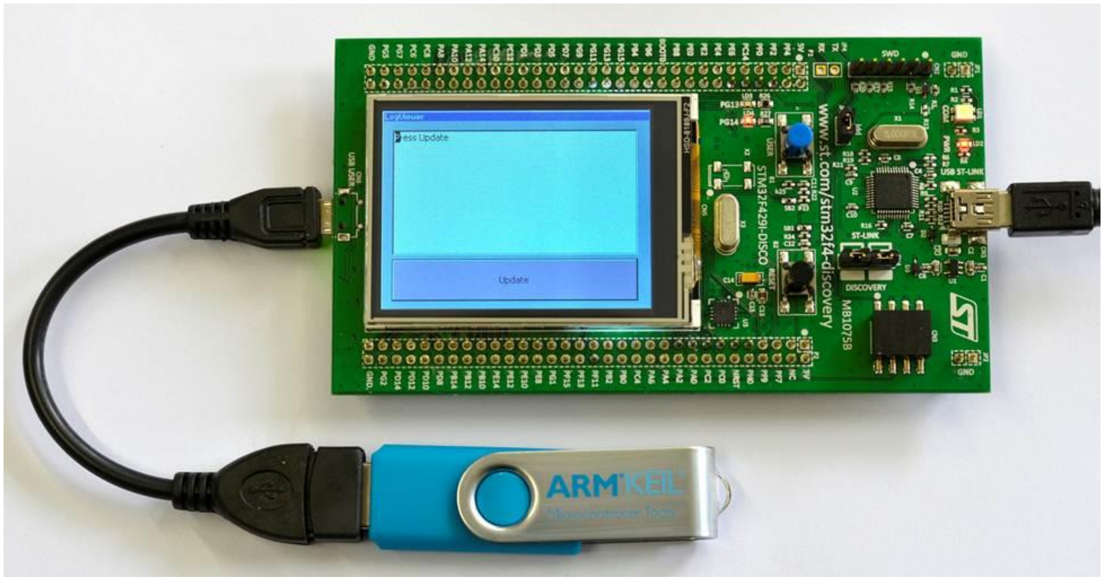
CMSIS-RTOS RTX is a real-time operating system that is part of Keil MDK and adheres to the CMSIS specification. It is used to control the application.

The **Board support** files enable the user to quickly develop code for the hardware used here. It provides a simple API to control LEDs, the touch screen and the LCD interface. Other components support push buttons, joysticks, A/D converters or other external devices.

Middleware provides TCP/IP networking stacks, USB communication, Graphics, and File access. The Middleware used in this application is part of MDK-Professional and uses several CMSIS-Driver components.

CMSIS-Driver is an API that defines generic peripheral driver interfaces for Middleware, making it reusable across compliant devices. It connects microcontroller peripherals with Middleware that implements, e.g. communication stacks, file systems, or graphic user interfaces. CMSIS drivers are available for several microcontroller families and are part of the DFPs. The DFP contains the support for the **Device** in terms of startup and system code, a configuration file for the CMSIS-Driver and a device family-specific software framework with a hardware abstraction layer (HAL).

The basis for the software framework is **CMSIS-Core** which implements the basic run-time system for a Cortex-M device and gives the user access to the processor core and the device peripherals. The device header files adhere to the CMSIS-Core standard and help the user to access the underlying hardware.



The STM32F32F429IDiscovery Kit with the USB Stick connected to USB User OTG Connector.

The LCD displays the screen created in the Graphical Display section in Steps 4, 5 and 6. In our example in this tutorial, the display will be rotated by 90 º from that shown above.

Prerequisites

To run through the workshop, you need to install the following software. Directions are given below:

- MDK-ARM Version 5.38a or later (https://www.keil.com/demo/eval/arm.htm, http://www2.keil.com/mdk5).
- A valid MDK-Professional license, e.g. provided by
  - a free of charge 30 days limited license key (https://www.keil.com/MDKEvaluationRequest/)
  - or the MDK Community Edition (https://www2.keil.com/mdk5/editions/community).
- Keil::MDK-Middleware 7.16.0 or higher, ARM::CMSIS 5.9.0 or higher, Keil::ARM_Compiler 1.7.2 or higher, Keil::MDK-Middleware_Graphics 1.2.0
- Keil::STM32F4_DFP 2.17.0 (or later), which includes the STM32F429I-Discovery Board Support Package (BSP). We will download this from the Internet using Pack Installer.
- STM32F429I-Discovery Kit (www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF259090).

  Note: The Solder bridge SB9 ***must*** be bridged for the Serial Wire Viewer (SWV) to work. A soldering iron is needed.

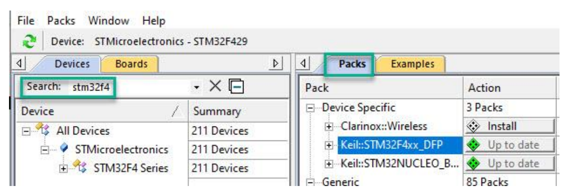
Set up the Workshop Environment

Install MDK:

1. Install ***Keil MDK Version 5.38a*** or later. Use the default folder *C:\Keil_v5*
2. After the initial Keil MDK installation, the Pack Installer utility opens up. Read the Welcome message and close it.

Install the STM32F4xx Software Pack:

1. If Pack Installer is not open, first open µVision®: . Then open Pack Installer by clicking on its icon:
2. The bottom right corner should display ONLINE: If it shows OFFLINE, connect your PC to the Internet.
3. Locate the Pack by entering *stm32f4* at the *Search* tab on the left. Next, find the **Keil::STM32F4xx_DFP** at the *Pack* tab on the right. Click *Install* at the latest Pack version offered. The installation will commence.
4. The Pack Installer confirms the successful installation.



Note that these other required Software Packs in this list are pre-installed:

- Keil::MDK-Middleware
- ARM::CMSIS
- Keil::ARM_Compiler
- Keil::MDK-Middleware_Graphics

Install your MDK-Professional license.

1. Request a free 30-day trial of MDK-Professional https://www.keil.com/MDKEvaluationRequest/
2. With the Product Serial Number (PSN) received, activate your license: https://developer.arm.com/documentation/101454/0110/License-Management/Single-User-License/Installing-a-LIC
3. For more information and license installation instructions, see: www.keil.com/download/license/

Install the ST-Link USB Drivers:

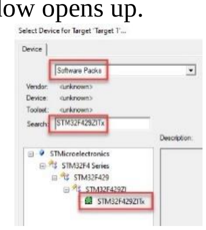
1. Open the Windows Explorer as administrator and navigate to *C:\Keil_v5\ARM\STLink\USBDriver*
2. Double-click on **stlink_winusb_install.bat** to install the required USB drivers for the onboard ST-Link debug adapter. The drivers will install it in the usual fashion.
3. Update the ST-Link firmware by executing *C:\Keil_v5\ARM\STLink**ST-LinkUpgrade.exe***. The best ST-Link firmware to use is *V3J11M3* or later. You can identify the version installed on your board with this Upgrade utility. The Discovery board must be connected to your PC with a USB-Mini cable to change its firmware.

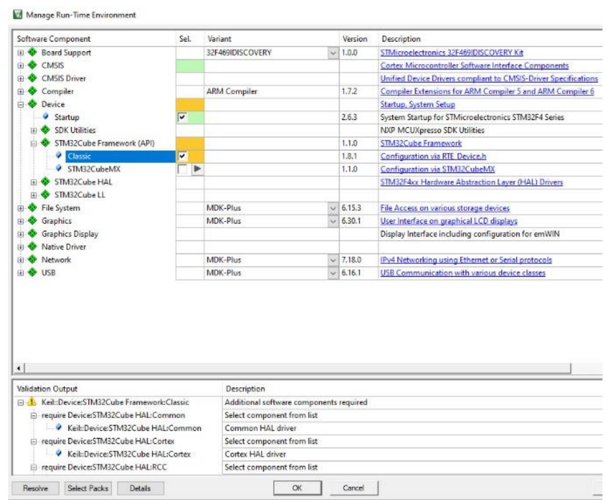
Step 1: Create a Project

Create a New Project for the Evaluation Board

Create a project with initialization files and the main module:

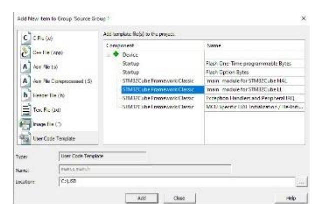
1. In the main µVision menu, select **Project → New µVision Project**. The Create New Project window opens up.
2. Create a suitable folder in the typical fashion and name your project. We will use *C:\USB*, and the project name will be ***USB***. When you save the project, the project file name will be *USB.uvprojx*.
3. The **Select Device for Target** window opens. Select **STM32F429ZITx**:
4. Click on **OK,** and the **Manage Run-Time Environment (RTE)** window opens:
5. Expand the various options as shown and select **CMSIS:Core**, **Device:Startup**. Select OK and open *Options For Target – C/C++(AC6).* Select at *Language C* **C99**. Return to the RTE.





6. Most devices provide additional hardware abstraction layers listed under the *Device* component. The *STM32Cube HAL* is a list of available drivers for the STM32F429. It requires a framework. Select **STM32Cube Framework (API):Classic**. For more information, click on the link *STM32Cube Framework* which opens the documentation.
7. In the *Sel.* column, you see some orange blocks. Click on **Resolve** in the Validation Output window, and these will turn green.
8. Click **OK** to close this window
9. In the Project window, expand all the items and have a look at the files that µVision has added to your project:

Add the main.c file:

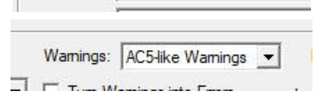


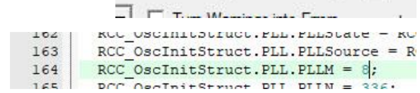
1. Right-click on Source Group 1 -select **Add New Item to Group 'Source Group 1'…**
2. In the window that opens, select **User Code Template**. Select the **'main' module for STM32Cube HAL**. It initializes the STM32Cube HAL and configures the clock system. Click on **Add**.

Set the CPU Clock Speed:

The external crystal oscillator on the development kit has a frequency of 8 MHz.

1. Open the **Options for Target** tab. Select **C/C++**.
2. Enter **HSE_VALUE=8000000** in the **Define** box. The HSE_VALUE represents the crystal frequency. This will set the CPU clock to 168 MHz in system_stmf4xx.c.
3. Also, select at *Warnings:* **AC5 Like Warnings**.
4. Click on **OK** to close this window.





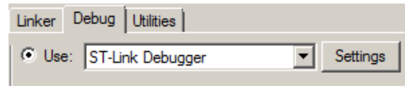
5. Open main.c, search for *SystemClock_Config*() and set **8** as **RCC_OscInitStruct.PLL.PLLM**
6. Select **File → Save All** or press
7. Compile the project source files: There will be no errors or warnings displayed in the Build Output window. If you get any errors or warnings, please correct this before configuring the ST-Link V2 Debug Adapter.

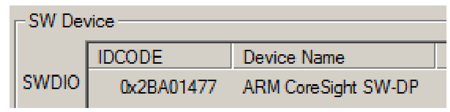
At this point: We have created a new MDK 5 project called USB.uvprojx. We have set the CPU clock speed and added the CMSIS environment, a main.c file and compiled the source files to test everything.

Setup the Debug Adapter

Select the ST-Link V2 Debug Adapter:

1. Select Target Options or ALT-F7. Select the **Debug** tab.
2. In the **Use** box, select “ST-Link Debugger”.
3. Click on **Settings**. In the **Port** box, select **SW** (for Serial-Wire Debug SWD).
4. In the **SWDIO** box, you must see a valid IDCODE and **ARM CoreSight SW-DP**. This indicates that µVision is connected to the STM32's debug module.

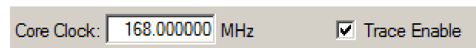




If you see an error or nothing in the SWDIO box, you must fix this before continuing. Make sure the board is connected.

Configure the Serial Wire Viewer (SWV):

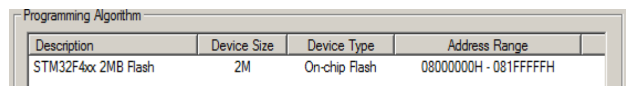
1. Select the **Trace** tab. In the **Core Clock** box, enter **168 MHz** and select **Trace Enable**. This sets the speed of the SWV UART signal and debugger timing displays. Unselect **EXCTRC** (Exception Tracing). Leave all other settings at their defaults.



Note: Solder Bridge SB9 must be bridged for SWV to function.

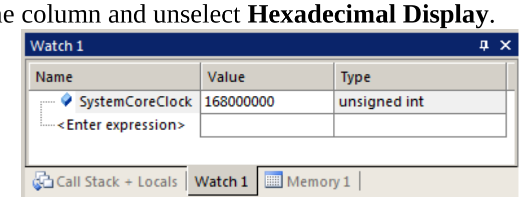
Select the Flash programming algorithm:

2. Select the **Flash Download** tab.
3. Confirm the *STM32F4xx 2 MB* Flash programming algorithm is selected as shown here:
   If not, click on **Add** to choose it.
4. Click on **OK** twice to return to the main menu.
5. Next, enter µVision's **Debug mode**:
6. Click on the RUN icon .
7. The program is now running.

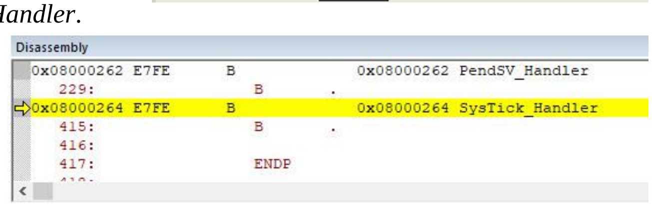


Insert a global variable in the Watch window:

1. In the Project tab under **Device**, double-click on **system_stm32f4xx.c** to open it up.
2. Find the variable *SystemCoreClock*. It is declared near line 137.
3. Right-click on it and select **Add SystemCoreClock to…** and select **Watch 1**. *Watch 1* will automatically open if it is not already open and display this variable.
4. In the Watch 1 window, right-click on SystemCoreClock in the Name column and unselect **Hexadecimal Display**. SystemCoreClock will now be displayed with the correct frequency of 168 MHz.
   Note: You can add variables to the Watch and Memory windows while running your program.



5. Stop the program. See the ***Disassembly*** window. The program counter (R15) will be at a B instruction in the *SysTick_Handler*. The B instruction is a branch of itself. Stopping in the SysTick Handler can be avoided by adding the user code template "Exception Handlers and Peripheral IRQ". As we will use CMSIS-RTOS RTX, this is not required here.
6. The yellow arrow is the program Counter (PC).
7. Exit Debug mode.

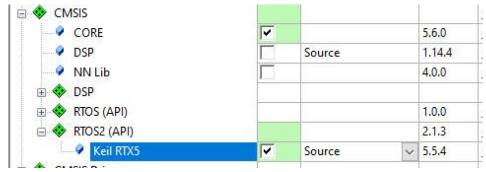


At this point: We have selected the debug adapter, enabled the Serial Wire Viewer trace (SWV) and selected the Flash programmer. We also demonstrated how to display the CPU clock in a Watch window.

Step 2: Add CMSIS-RTOS

Add and configure CMSIS-RTOS RTX for a simple Blinky application

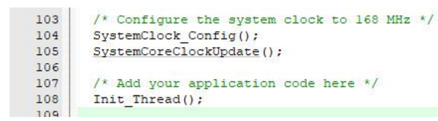
Select and Configure RTX RTOS:



1. Open the Manage Run-Time Environment (RTE) window:
2. Under **CMSIS:RTOS2 (API)**, select **Keil RTX5** with Variant **Source** as shown here
3. Press OK
4. In the *Project* window, note that new files are added under the *CMSIS* heading – e.g. *RTX_CM4F.lib, rtx_lib.c, RTX_Config.c* and *RTX_Config.h*

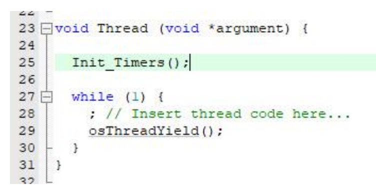
Add a Thread Template

1. In the Project window under Target 1, right-click **Source Group 1** and select **Add New Item Group 'Source Group 1'**…
2. In the window that opens, select **User Code Template**.
3. Select at the CMSIS component the **CMSIS-RTOS2 Thread**.
4. Click on **Add**. Note that *Thread.c* is added to the Source Group 1 in the Project window.
5. In *main.c* near line 79, enter `extern int Init_Thread(void);`
6. and after */* Add your application code here */* enter `Init_Thread();`



Add the Timer.c source file and add Timer Initialization Function Call:

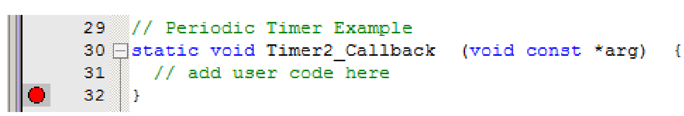
1. In the Project window under Target 1, right-click **Source Group 1** and select **Add New Item Group 'Source Group 1'**…
2. In the window that opens, select **User Code Template**. Select **CMSIS-RTOS2 Timer**.
3. Click on **Add**. Note *Timer.c* is added to the Source Group 1 in the Project window.
4. In **Thread.c** near line 10, add this line: `extern int Init_Timers(void);`
5. …and in function *Thread()* add `Init_Timers();`
   Init_Timers() creates two timers: *Timer1* (a one-shot) and *Timer2*, which is a 1-second periodic timer. Timer2 calls a callback function.



6. Select **File → Save All** or .
7. Compile the project source files by clicking on the Rebuild icon . There will be no errors or warnings in the Build Output window. If there are any errors or warnings, please correct them before continuing.

Demonstrating the Timer is Working:

1. Program the Flash and enter Debug mode: Click on the RUN icon.
2. The program is running.
3. In *Timer.c*, at the *Timer2_Callback()* function, near line 32, set a breakpoint by clicking on the grey box. A red circle will appear. The grey box indicates that assembly language instructions are present, and a hardware breakpoint will be legal.

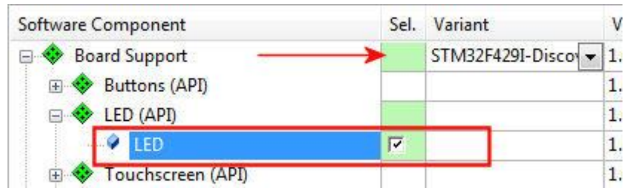


4. The program will soon stop here.
5. Click on RUN , and in 1 second, and it will stop here again when the Timer2 is activated.
6. Remove the breakpoint for the next step.

At this point: We added the RTX RTOS to your project. We enabled a periodic Timer and demonstrated that the program was running.

Blink the LED:

1. Exit Debug mode.
2. Open the Manage Run-Time Environment window:
3. Expand the Board Support (ensure that **STM32F429I-Discovery** is selected – see the red arrow)
4. Under **Board Support:LED (API)** select **LED**
5. Click **OK** to close this window.



In the Project window, the new header *Board Support* contains the file *LED_F429Discovery.c*., used to configure the I/O pins of the LEDs with a *LED_Initialize()* routine. The *LED_On()* and *LED_Off()* functions control the LEDs.

Add C Code to Blink LED LD3:

In **Thread.c** and **Timer.c**, add `#include "Board_LED.h"`

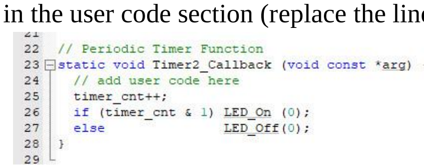
TIP: You can also select #includes from a list:

- Select a line in a source code file and right-click on it.
- Select **Insert '#include file'**. A menu opens up with provided #includes that you can select from.

1. In **Thread.c**, next to *Init_Timers()*, add `LED_Initialize();`

2. In Timer.c, near line 10, add this line:
   `static int timer_cnt = 0;`
3. In Timer.c inside the Timer2_Callback function near line 23, add this code in the user code section (replace the line `//add user code here`):

```
timer_cnt++;
if (timer_cnt & 1) LED_On (0);
else               LED_Off(0);
```



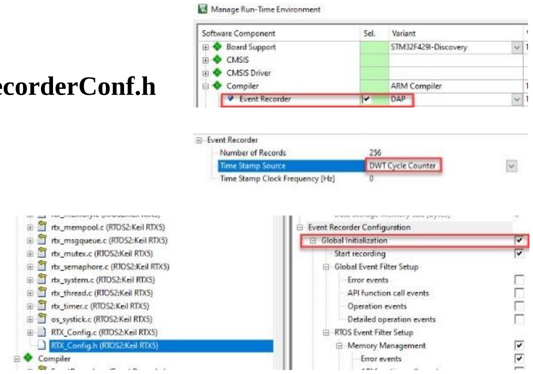
4. Select File/Save All or .
5. Compile the project: There will be no errors or warnings in the Build Output window.
6. Program the Flash and enter Debug mode:
7. Click on RUN.
8. LED PG13 (green) will now blink according to your created Timer.
9. Leave the program running for the next steps.

TIP: In the `LED_On` function call: (0) is the green LED. Using (1) will blink the red LED.

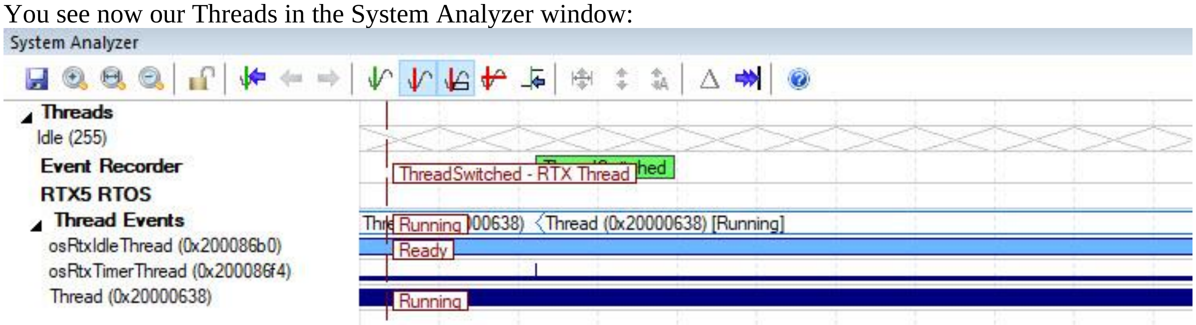
At this point: We have selected a LED driver from the CMSIS-Pack BSP to create a blinking LED. Using a timer, we have created a simple program that blinks this LED every 1 second.

RTX Kernel Awareness

System Analyzer:



1. Enable the **Event Recorder** in the **RTE-Compiler** component.
2. At the Project window in the **Compiler** Tab, open the file **EventRecorderConf.h** and select the Configuration Wizard.
3. Expand the Event Recorder group.
4. Ensure that the **DWT Cycle Counter** is set.
5. Open at the Project tree the **CMSIS** tab.
6. Open the file **RTX_Config.h** and select the **Configuration Wizard.**
7. At **Event Recorder Configuration,** enable the **Global Initialization.**
8. Save all files, build and flash the project.
9. Start the **Debugger** and open the **System Analyzer** from the toolbar or via the *View - Analysis Windows - System Analyzer* menu. Run your project for a few seconds and stop it.
10. You see now our Threads in the System Analyzer window:

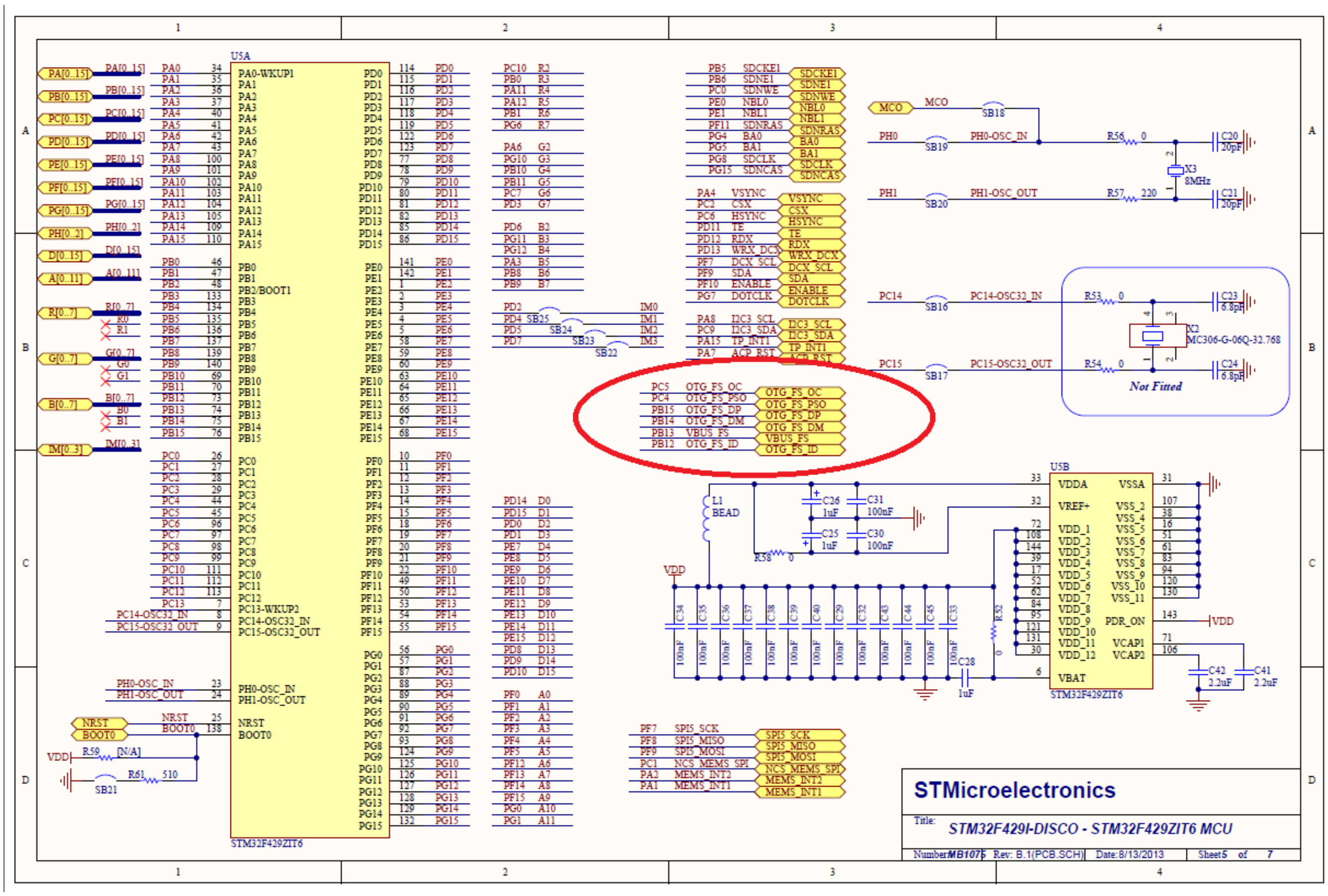


Exit the Debug mode. 

Step 3: Add USB Host with Mass Storage Support

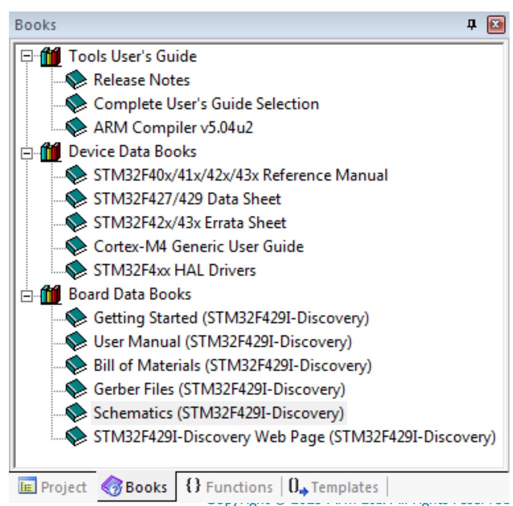
Configure the CMSIS-Driver for the USB component

To correctly configure the USB Host Middleware, it is necessary to understand the USB User connector available on the target hardware.



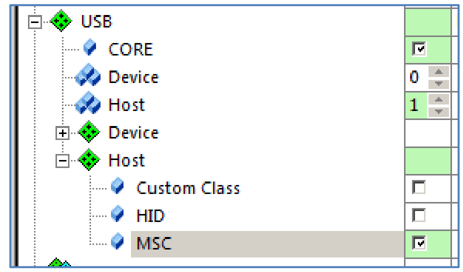
The STM32F429I Discovery Kit provides a **USB connector** that interfaces with the **USB OTG High-speed STM32F429 peripheral** via the **on-chip full-speed PHY** (GPIOB.14 and GPIOB.15). The **VBUS power on/off pin** is active low on GPIOC.4. The **Overcurrent Detection pin** is active low on GPIOC.5. Since we are only using the USB Host interface we can ignore the remaining OTG pins.

This schematic is part of the Software Pack for the STM32F4. You access these documents using the **Books** tab. Other documents found here are datasheets, STMicroelectronics Getting Started Guides, ARM compiler and µVision manuals and more.



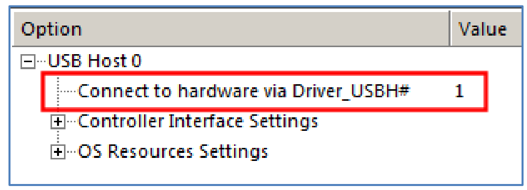
Add the USB Host middleware component to the project

As we want to connect a USB memory stick to the development board, we need to add support for the **USB Mass Storage Class (MSC)** to the project:



1. Open the RTE window:
2. Ensure the **MDK-Pro** Variant is set for the File System, Graphics, Network and USB components.
3. Under **USB:Host**, select **MSC** as shown here:
   Make sure you do not accidentally select MSC in the *Device* header. We are setting the STM32 up as a Host and not a Device.
4. Under **CMSIS Driver:USB Host (API),** select **High-speed**
5. Click **Resolve** to add other mandatory middleware components.
6. Click **OK** to close this window.

Connect USB Host 0 to the hardware and increase stack size:

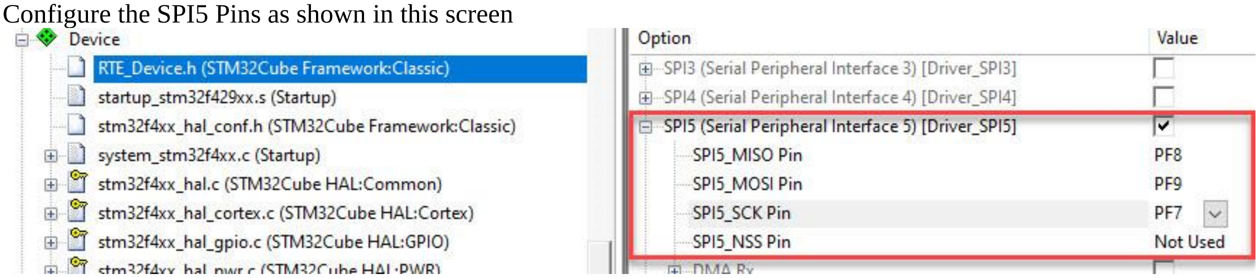


1. In the Project window, under the **USB** heading, double-click on **USBH_Config_0.c (Host)** to open it.
2. Click on its **Configuration Wizard** tab and then on **Expand All**.
3. Set **Connect to Hardware via Driver_USBH#** to **1**.
4. **Note:** **Driver_USBH1** represents the USB OTG High-speed interface. This is the CMSIS Driver that is configured in the previous step.
5. Keep the default value of **1024** bytes for the **Core Thread Stack Size**.

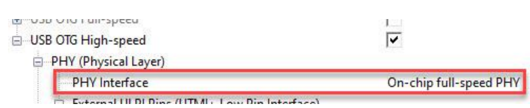
6. Select File/Save All or .

Configure the CMSIS-Driver for the USB Host

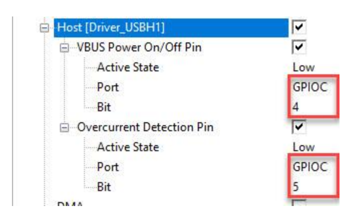
1. In the Project window, under the Device header, double-click on **RTE_Device.h** to open it for editing.
2. Open the **Configuration Wizzard**
3. Expand **SPI5** (Serial Peripheral Interface 5)
4. Configure the SPI5 Pins as shown in this screen



5. Enable **USB OTG High-Speed** and change the **PHY Interface** to **On-chip Full-speed PHY**.

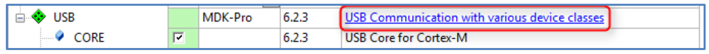


6. Enable **Host [Driver_USBH1]** as shown here:
7. Set the hardware parameters for the **USB OTG High-speed interface** precisely as shown here:
   - Both *Ports* must be **GPIOC;** the first *Bit* is **4**, and the second *Bit* is **5**.



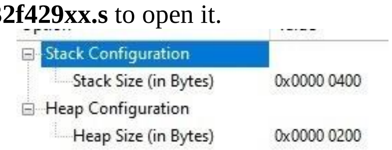
Configure the Stack, Heap and Thread memory resources

The resource requirements of the USB component can be found in the Middleware documentation that is accessible using the link next to the USB component in the Manage Run-Time Environment window:

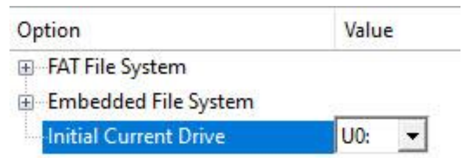


Configure Heap and Thread Stack USB sizes:

1. In the **Project** window under the **Device** heading, double-click on **startup_stm32f429xx.s** to open it.
2. Select its **Configuration Wizard** tab.
3. Confirm the Stack Size is set to **0x400** bytes and Heap Size is set to **0x200**.
4. Under the **CMSIS** heading, double-click on **RTX_Config.h** to open it.



Set the Default Drive Letter:



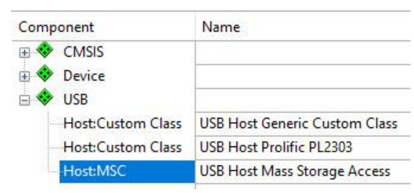
1. In the **Project** window under the **File System** heading, double-click on **FS_Config.c** to open it.
2. Select the **Configuration Wizard** tab.
3. For a USB mass storage drive, the File System component expects the drive letter to be **U0**. So change **Initial Current Drive** to **U0:**
4. Select File/Save All or .
5. Compile the project: 

No errors or warnings will be generated as shown in the Build Output window. Please correct any errors or warnings before you continue.

Next, we will add the user code to access a USB Device (the USB stick)

Add the user code that accesses the USB storage device

Add **USBH_MSC.c** and **USBH_MSC.h:**



1. Right-click on **Source Group 1** in the Project window again. Select **Add New item to Group 'Source Group1'**…
2. Select **User Code Template**.
3. Under the **USB** heading and in the **Name** column, select **USB Host Mass Storage Access** and click on **Add**.
4. The files **USBH_MSC.c** and **USBH_MSC.h** are now added to your project under the Source Group 1 heading.
5. These provide the relevant access functions for the USB storage device.
6. Select File/Save All or 

We will use a CMSIS-RTOS thread to implement access to a file on the USB stick.

Modify **Thread.c** (that was already included in Step2, “Add a Thread Template”):

To allow file access, we add and save the following application code in the module **Thread.c**:

```
#include <stdio.h>
#include "main.h"
#include "cmsis_os2.h"                         // CMSIS RTOS header file
#include "Board_LED.h"                         // Board Support:LED
#include "USBH_MSC.h"                          // Access storage via USB Host

char fbuf[200] = { 0 };
extern int Init_Timers (void);

/*----------------------------------------------------------------------------
-
 *      Thread 1 'Thread_Name': Sample thread
 *---------------------------------------------------------------------------
*/
osThreadId_t tid_Thread;                        // thread id
void Thread (void const *argument);             // thread function

int Init_Thread (void) {

  tid_Thread = osThreadNew((void *)(uint32_t)Thread, NULL, NULL);
  if (tid_Thread == NULL) {
    return(-1);
  }
  return(0);
}

void Thread (void const *argument) {
  static unsigned int result;
  static FILE *f;

  Init_Timers();
  LED_Initialize();
  USBH_Initialize (0);

  while (1) {
    result = USBH_MSC_DriveMount ("U0:");
    if (result == USBH_MSC_OK)  {
      f = fopen ("Test.txt", "r");
      if (f) {
        fread (fbuf, sizeof (fbuf), 1, f);
        fclose (f);
      }
    }
    osDelay (1000);
  }
}
```

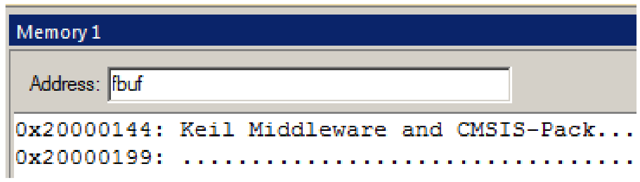
At this point: On this page, we added the code to open, read and close the data in file Test.txt located in a USB stick connected to USB User.

Prepare a USB memory stick:

1. Take a USB memory stick, label it **USB USER**, and create a file called **Test.txt** containing the message *Keil Middleware and CMSIS-Pack* using ASCII characters.
2. Plug this stick with an adapter cable into the STM32F429I-Discovery board.

Build and RUN:

1. Compile the project:  .
2. Enter Debug mode: 
3. Click on the **Memory 1** tab. Enter **fbuf** in this window:
4. Right-click anywhere in the data field area and select **Ascii**
5. Set a breakpoint in **Thread.c** on `fclose (f)` near line 35.



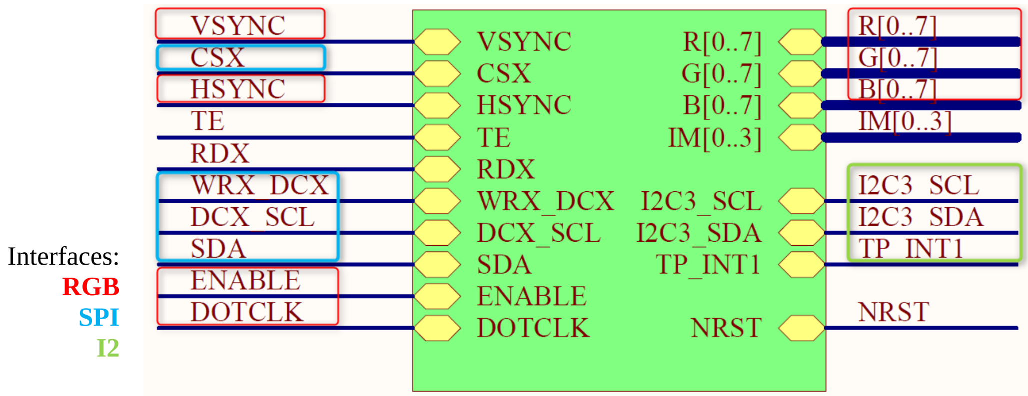
6. Click on RUN. 
7. The text will appear in the Memory 1 window in a few seconds.
8. The program will stop at the hardware breakpoint.
9. To repeat this sequence, click on the RESET icon  and then RUN .
10. Stop the program and leave the Debugger.

Step 4: Add the Graphical User Interface

Understanding the Hardware

To correctly configure the Graphic Interface it is necessary to understand the **schematics**. Here's another excerpt from the schematics showing the LCD connections.

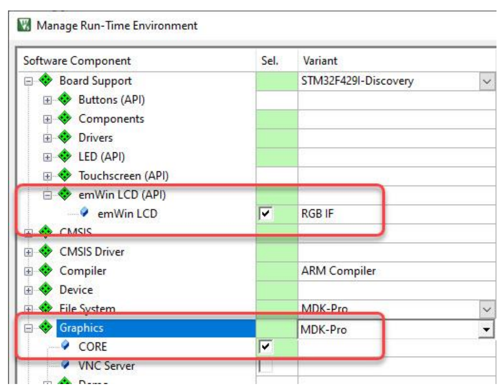
The STM32F429 has a **high-speed RGB interface** (red) connected to the LCD. SPI (blue) is connected to the Device's SPI5 interface to configure the display. The Touch Screen connects via I2C (green) to the microcontroller's I2C3 interface.



Add the Graphic Core and Graphics Display Interface

Select the **emWin graphics components**:

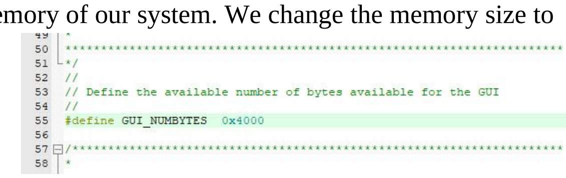
1. Open the Manage Run-Time Environment window:
2. Under **Board Support:emWin LCD (API)**, select **emWin LCD**. This component is the interface to the board LCD display.
3. Select **Graphics:Core**. This will be used for the User interface.
4. Click **Resolve** to add the missing CMSIS-Drivers.
5. Click **OK** to close this window.



Configure Memory for Graphics Core

The Graphics Core uses a dedicated memory for its features that needs configuration.

1. In the Project window under the **Graphics** heading, double-click on **GUIConf.c** to open it. *GUIConf.c* configures the Graphics Core. The default configuration exceeds the memory of our system. We change the memory size to 0x4000, which is sufficient for many applications (refer to the emWin User Manual).
2. Change the `GUI_NUMBYTES` define near line 55 to **0x4000**
3. Select File/Save All or .

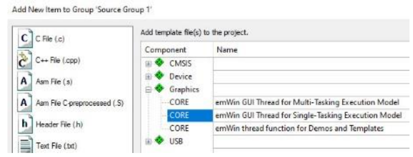


What we have at this point: The graphics hardware configuration is complete.

Add the code to output "Hello World" to the LCD

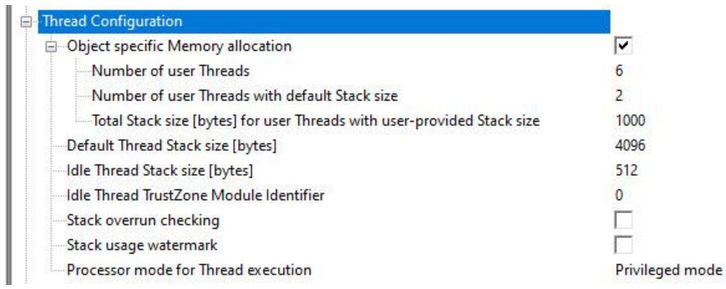
Add The Graphics Thread and start the thread in main.c:

1. In the Project window under Target 1, right-click **Source Group 1** and select **Add New Item to Group 'Source Group 1'...**
2. Select **User Code Template**.
3. From the **Graphics** heading, select **emWin GUI Thread for Single-Tasking Execution Model**.
   Note: Single-task execution is where one thread (task) calls the emWin functions. This reduces the memory footprint and is sufficient for many applications. Only one thread can call the GUI functions (refer to the Execution Model in the emWin User Manual).
4. Click on **Add**, and the file **GUI_Single_Thread.c** is part of your project.



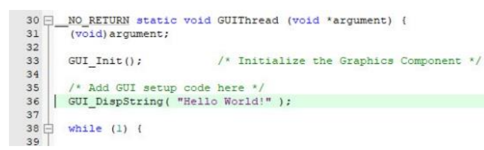
Modify the RTX for this new thread

1. Open at the CMSIS-tab **RTX_Config.h**
2. Modify the **Thread Configuration** as shown here:



Add the text that will display on the LCD:

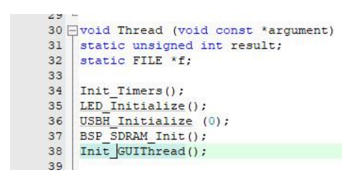
1. In the Project window under the **Source Group 1** heading, double-click on **GUI_SingleThread.c** to open it.
2. Near line 24, just before the `while(1)` loop, add:
   `GUI_DispString("Hello World!");`
3. Select File/Save All or .

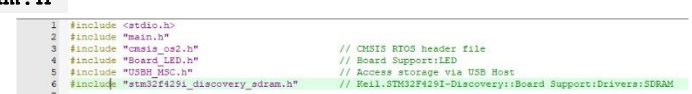


Modify Thread.c

You can now demonstrate the display of the string "Hello World!" on the LCD – in **Thread.c**

1. near line 10 add: **`BSP_SDRAM_Init();`**
2. and **`extern int Init_GUIThread (void);`**
3. Extend the includes with
   `#include "stm32f429i_discovery_SDram.h"`
4. Select File/Save All or .





Build and run your project:

1. Compile the project:
2. Program the Flash and enter Debug mode:
3. Click on RUN.
4. The LCD will display **Hello World!**
5. Stop the processor . Exit Debug mode.

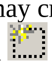
Step 5: Design and Add the Graphics to be displayed on the LCD

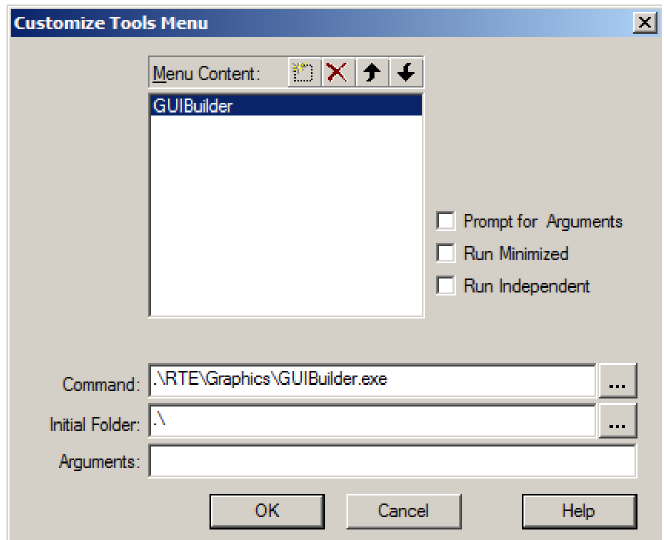
Configure GUIBuilder and Use it to Create the Graphics

emWin provides a tool called **GUIBuilder** to design the graphics that will display on the LCD screen. µVision allows you to execute GUIBuilder from within.

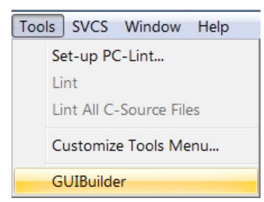
1. Open the Manage Run-Time Environment window: 
2. Under **Graphics:Tools** select **GUI Builder**
3. Click **OK**

Create a shortcut on the µVision Tools menu:

1. In the main µVision menu, select **Tools → Customize Tools Menu**. The window below opens up.
2. This will allow you to add a shortcut to your tools menu to launch GUIBuilder. This only needs to be done once for every installation of MDK-ARM and not every project you may create.
3. Click on the Insert icon  (or press the Insert key).
4. Enter the text **GUIBuilder** as shown and press Enter.
5. In the Command and Initial Folder boxes enter **.\RTE\Graphics\GUIBuilder.exe** and **.** .
6. Click on **OK** to close it.



7. Click on **Tools** in µVision, and the new GUIBuilder menu item will display like this:
8. Click on **GUIBuilder,** and it will start.



Create the Frame:

1. Click on the Framewin icon: A box will be created labelled Framewin.
2. With the FrameWin box selected, change the Property Name from FrameWin to **LogViewer**.
3. In the property column, enter xSize = **240** and ySize = **320**. This specifies the size of the LCD.
4. Press Enter.

Property	Value
Name	LogViewer
xPos	0
yPos	0
xSize	240
ySize	320
Extra bytes	0

Add the Multi Edit Widget

1. Click on the **Multiedit** icon:
2. Click and drag to fill the **LogViewer** area, as shown below. Leave a space at the bottom for the button.

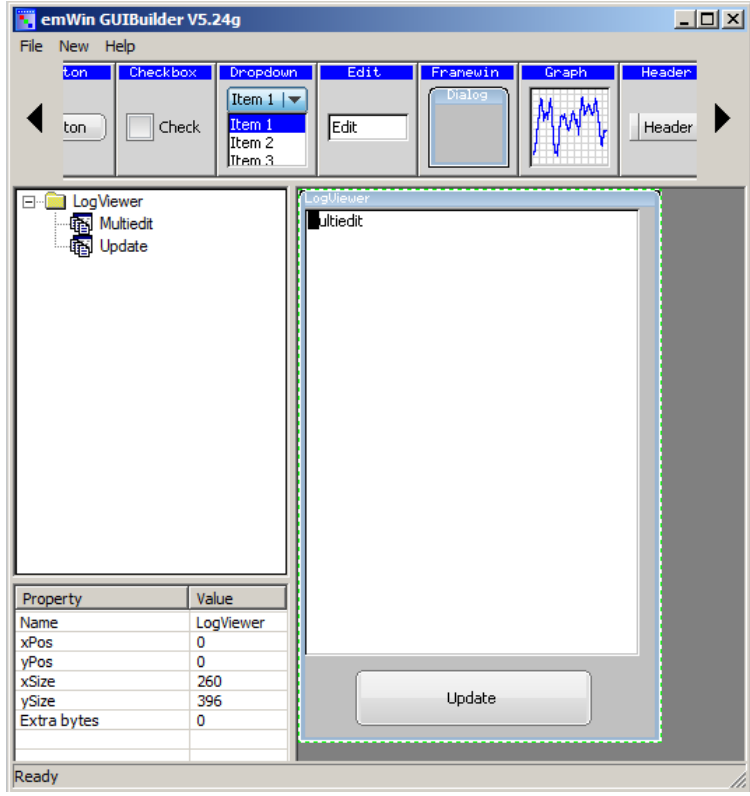
Add the Button:

1. Click on the Button icon:
2. Use your mouse to size and position as shown below:
3. With the Button selected, change the Property Name to **Update**.
4. Click Enter to finish.

Property	Value
Name	Update
xPos	9
yPos	245
xSize	210
ySize	50
Extra bytes	0

Save and Export your GUI:

1. Select **File → Save**. A C source file with your GUI design is created and saved into your µVision project root folder. The file name is derived from your parent GUI element; in this case, the name is **LogViewerDLG.c**.
2. You will need to add this to your project.
3. Close GUIBuilder.



Add LogViewerDLG.c to the Project and Run the GUI

Adding your GUI design file LogViewerDLG.c to Your Project:

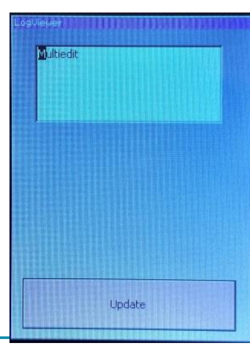
1. In the µVision Project window, right-click on “Source Group 1”.
2. Select **Add Existing Files to Group 'Source Group 1'**…
   Note: Choose ***Existing*** rather than ***New*** as previously.
3. In the window that opens up, select the file **LogViewerDLG.c**. Click on **Add** once and then **Close**.
4. LogViewerDLG.c is now added to your project.
5. In the Project window, under **Source Group 1**, double-click **LogViewerDLG.c** to open it for editing.
6. Near line 70, add this line to reference the file buffer fbuf: `extern char fbuf[200];`

Create the GUI Design:

1. In the µVision Project window under **Source Group 1**, double-click on **GUI_SingleThread.c** to edit it.
2. In GUI_SingleThread.c, near line 4 add this line: `#include "dialog.h"`
3. In GUI_SingleThread.c, near line 5 add this line: `extern WM_HWIN CreateLogViewer(void);`
4. Comment out: `//GUI_DispString("Hello World!");`
5. Near line 26 add this line: `CreateLogViewer();`

Build and RUN:

1. Select File/Save All or .
2. Compile the project: 
3. Enter Debug mode:  and click on RUN. 
4. The GUI we have just created appears on the screen:



Step 6: Add the Touchscreen Interface

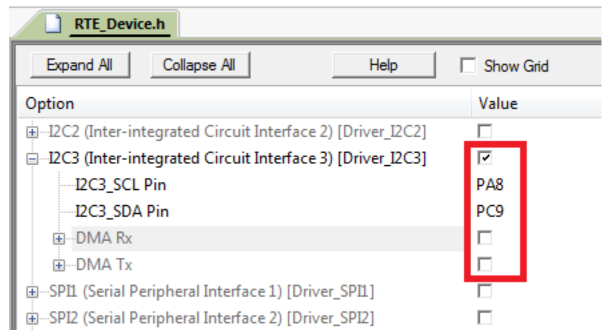
An implementation for the touchscreen interface is provided as a Software Component under **Board Support**. The touchscreen hardware connects via the I2C peripheral (I2C3); therefore, we will use the standard CMSIS-Driver for I2C.

Add Software Components for Touchscreen

1. Open the Manage Run-Time Environment window:
2. Under **Graphics:Input Device**, select **Touchscreen**
3. Click **Resolve** to select other required components. This adds from the Board Support the Touchscreen Interface and from the CMSIS Driver the I2C driver.
4. Click **OK** to close this window.

Configure the CMSIS-Driver for the I2C Interface

1. In the Project window, under the **Device** group, double-click on **RTE_Device.h** to open it for editing.
2. Click on its **Configuration Wizard** tab.
3. Enable **I2C3** and configure the parameters for this driver instance, as shown in the picture. Select **PA8** and **PC9** since these pins provide the interface to the touchscreen hardware.
4. Touchscreen is a low-bandwidth interface, so we can disable the DMA channels. This avoids DMA conflicts with other drivers.

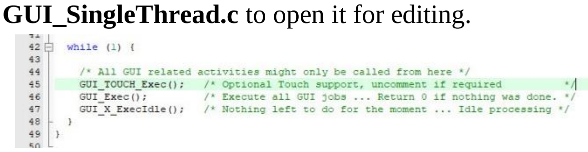


Enable Touch support in GUI_SingleThread.c

1. In the Project window, under **Source Group 1**, double-click **LogViewerDLG.c** to open it for editing.
2. Near line 118 is case `WM_NOTIFICATION_CLICKED` for the Update button; add this code:

```
hItem = WM_GetDialogItem(pMsg->hWin, ID_MULTIEDIT_0);
MULTIEDIT_SetTextColor (hItem, 1, GUI_BLACK);
MULTIEDIT_SetText(hItem, fbuf);
```

3. In the Project window, under **Source Group 1**, double-click **GUI_SingleThread.c** to open it for editing.
4. Extend GUIThread with the call of `GUI_TOUCH_Exec();`



Build and RUN:

5. Select File/Save All or .
6. Compile the project:
7. Enter Debug mode: and click on RUN.
8. Press the **Update** button on the LCD. The content of the file Test.txt appears on the screen, which completes the task of our project:

The Component Viewer

Keil RTX5 supports the **Component Viewer**, which shows static information and helps to analyze the operation of software components. For a detailed description and its configuration, refer to the [Component Viewer] documentation. One part of the Component Viewer is the **RTX RTOS** window.

Our project still is running in the Debugger.
Select **View → Watch Windows → RTX RTOS**

The RTX RTOS window opens.

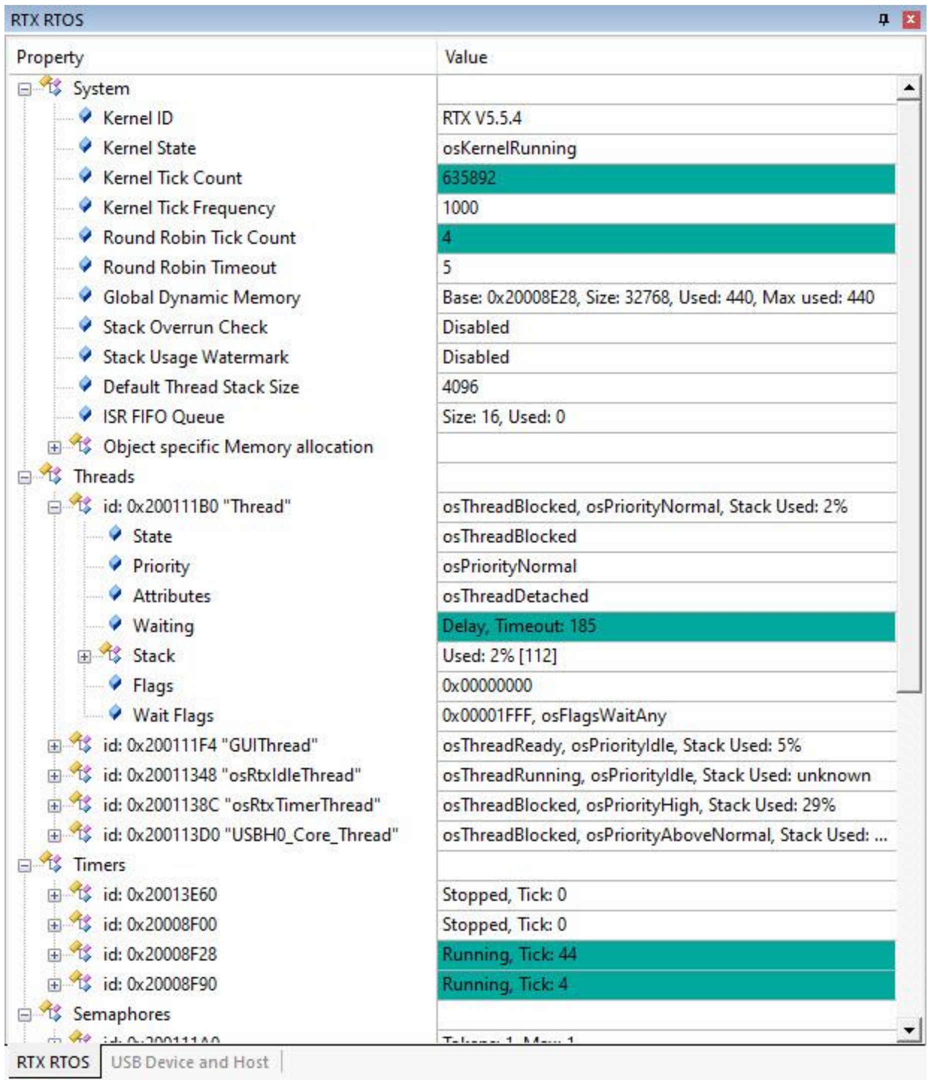
Arrange the window to see all content collected.

The content is continuously getting updated.

Stop the program execution .

The **System** tab confirms, e.g. the *RTX version* in use or the *Default Stack Size* that we have configured in Step 4.

You see also all the **Threads** created with their current statuses.



Serial Wire Viewer Summary

Serial Wire Viewer (SWV) is a 1-bit data-trace. It is output on the SWO pin, which is shared with the JTAG TDO pin. This means you cannot use JTAG and SWV together. Instead, use Serial Wire Debug (SWD or SW), a two-pin alternative to JTAG with about the same capabilities. SWD is selected inside the µVision IDE and is easy to use.

1. The STM329F429I Disco board ***must*** have the Solder Bridge SB9 bridged. SB9 is located on the bottom of the board close to jumper ldd. If SB9 is open, SWV will not work. The board is shipped with SB9, ***not*** bridged.
2. The Core Clock: is the CPU frequency and must be set accurately. In this tutorial, 168 MHz is used. The clock frequency is probably wrong if you see ITM frames in the Trace Records window of a number other than 0 or 31 or no frames at all.
3. SWV is configured in the Cortex-M Target Setup in the Trace tab. **In Edit mode:** Select Target Options  or ALT-F7 and select the Debug tab. Select Settings: Then select the Trace tab. **In Debug mode:** Select Debug/Debug Settings.. and then select the Trace tab.
4. Many STM32 processors need a particular initialization file to get SWV and/or ETM trace to function. This file is not required for this board as µVision accomplishes this during entry into Debug mode. Contact Keil tech support if you use a different STM32 processor and cannot get SWV working. SWO*xx*.ini files are provided in many µVision example projects that you can use. Insert it just below where you choose the debug adapter.
5. If SWV stops working, you can get it working by exiting and re-entering Debug mode. In rare cases, you might also have to cycle the board power. Constant improvements to the ST-Link V2 firmware are helping in this regard.
6. SWV outputs its data over a 1-bit SWO pin. Overloading can be typical depending on how much information you have selected to be displayed. Reducing the information to only what you really need helps limit the activity of variables. Using a ULINK*pro* on boards equipped with a 20 CoreSight ETM connector enables the SWV information to be output on the 4-bit ETM trace port.
7. For more information on STM32F429I-Discovery board see: www.keil.com/appnotes/docs/apnt_253.asp

---

Watch, Memory windows and Serial Wire Viewer can display:

- Global and Static variables. Raw addresses: i.e. *((unsigned long *)0x20000004)
- Structures.
- Peripheral registers – just read or write to them.
- Can't see local variables. (just make them global or static).
- Cannot see DMA transfers – DMA bypasses CPU and CoreSight and CPU by definition.
- You might have to qualify or copy your variables from the Symbol window fully.

Serial Wire Viewer (SWV) displays in various ways:

- PC Samples.
- A printf facility that does not use a UART.
- Data reads. Graphical format display in the Logic Analyzer: Up to 4 variables can be graphed.
- Exception and interrupt events.
- All these are Timestamped.
- CPU counters.

Instruction Trace (ETM):

- ETM Trace records where the program has been. Assembly instructions are all recorded.
- Assembly is linked to C source when available (this is up to your program).
- A recorded history of the program execution *in the order it happened.*
- Provides Performance Analysis and Code Coverage. Higher SWV performance.
- ETM needs a Keil ULINK*pro* to provide the connection to the 4-bit Trace Port found on many STM32 processors.

Document Resources

Books

- **Getting Started MDK 5:** www.keil.com/mdk5/.
- **Keil MDK ...resources that help you to get started** https://community.arm.com/support-forums/f/keil-forum/49652/keil-mdk-resources-that-help-you-to-get-started
- A good list of books on ARM processors: www.arm.com/support/resources/arm-books/index.php
- µVision contains a window titled **Books**. Many documents, including data sheets, are located there.
- A list of resources is located at: www.arm.com/products/processors/cortex-m/index.php (Resources tab).
- The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
- The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
- Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano.
- MOOC: Massive Open Online Class: University of Texas: http://users.ece.utexas.edu/~valvano/

Application Notes

1. Overview of application notes: www.keil.com/appnotes
2. Keil MDK for Functional Safety Applications: www.keil.com/safety
3. Using DAVE with µVision: www.keil.com/appnotes/files/apnt_258.pdf
1. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
2. CAN Primer using NXP LPC1700: www.keil.com/appnotes/files/apnt_247.pdf
3. CAN Primer using the STM32F Discovery Kit www.keil.com/appnotes/docs/apnt_236.asp
4. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
5. Porting a mbed project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
6. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
7. Using µVision with CodeSourcery GNU www.keil.com/appnotes/docs/apnt_199.asp
8. RTX CMSIS-RTOS in MDK 5 http://www.keil.com/pack/doc/cmsis_rtx/index.html
9. Lazy Stacking on the Cortex-M4 www.arm.com and search for DAI0298A
10. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
11. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
12. Cortex Debug Connectors: http://www.keil.com/support/man/docs/ulinkpro/ulinkpro_cs_connectors.htm

Useful ARM Websites

1. ARM Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content
2. ARM University Program: www.arm.com/university. Email: university@arm.com
3. ARM Accredited Engineer Program: www.arm.com/aae
4. mbed™: http://mbed.org
5. CMSIS standard: www.arm.com/cmsis
6. CMSIS documentation: www.keil.com/cmsis

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

Keil Products and Contact Information

Keil Microcontroller Development Kit (MDK-ARM™)

https://developer.arm.com/Tools%20and%20Software/Keil%20MDK#Editions

- MDK-Lite (Evaluation version) - $0
- MDK Community Edition – $0, full-featured, for non-commercial use
- MDK-Essential (unlimited compile and debug code and data size Cortex-M, ARM7 and ARM9)
- MDK-Plus, like MDK Professional, but no USB Host and IPv6 support
- MDK-Professional (includes File System, IPv4, IPv6, USB Device, USB Host and Graphic User Interface)
- ARM Compiler Qualification Kit: for Safety Certification Applications

USB-JTAG adapter (for Flash programming too)

- ULINK2 – Programming and Debug adapter, https://developer.arm.com/Tools%20and%20Software/ULINK2
- ULINK Plus - isolated debug connection, power measurement, and I/O for test automation https://developer.arm.com/Tools%20and%20Software/ULINKplus
- ULINKpro – Faster operation and Flash programming, Cortex-Mx SWV & ETM trace https://developer.arm.com/Tools%20and%20Software/ULINKpro
- ULINKpro D – Faster operation and Flash programming, Cortex-Mx SWV, no ETM trace.

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

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

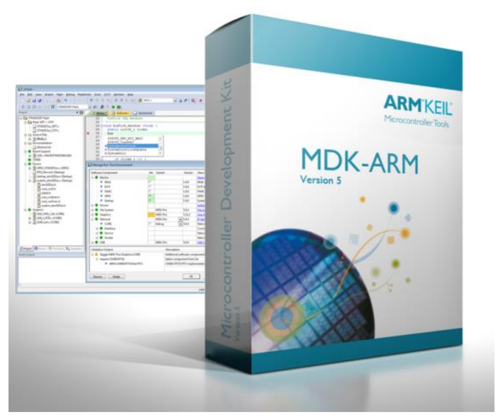
Contact sales.intl@keil.com +49 89/456040-20 for pricing in other countries.

CMSIS-RTOS RTX is now provided under a BSD license.

All versions, including MDK-Lite, include CMSIS-RTOS RTX *with source code!*

Keil includes free DSP libraries for the Cortex-M family.

Call your distributor 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.



http://www.keil.com/distis/

All products are available from stock.

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

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

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

For more information:

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

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

For the latest version of this document, go to www.keil.com/appnotes/docs/apnt_268.asp

CMSIS documentation: www.arm.com/cmsis

ARM CORTEX Processor Technology

CMSIS COMPLIANT ARM® Cortex™ Microcontroller Software Interface Standard

ARM KEIL Microcontroller Tools