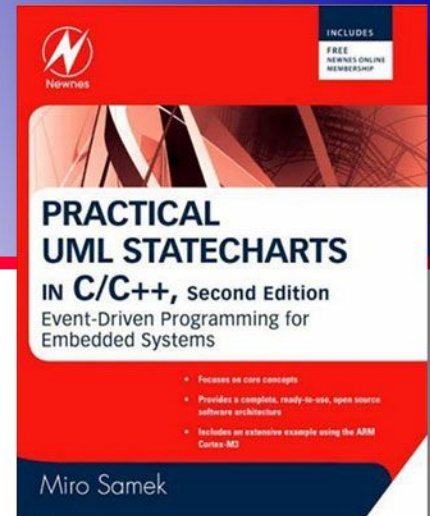


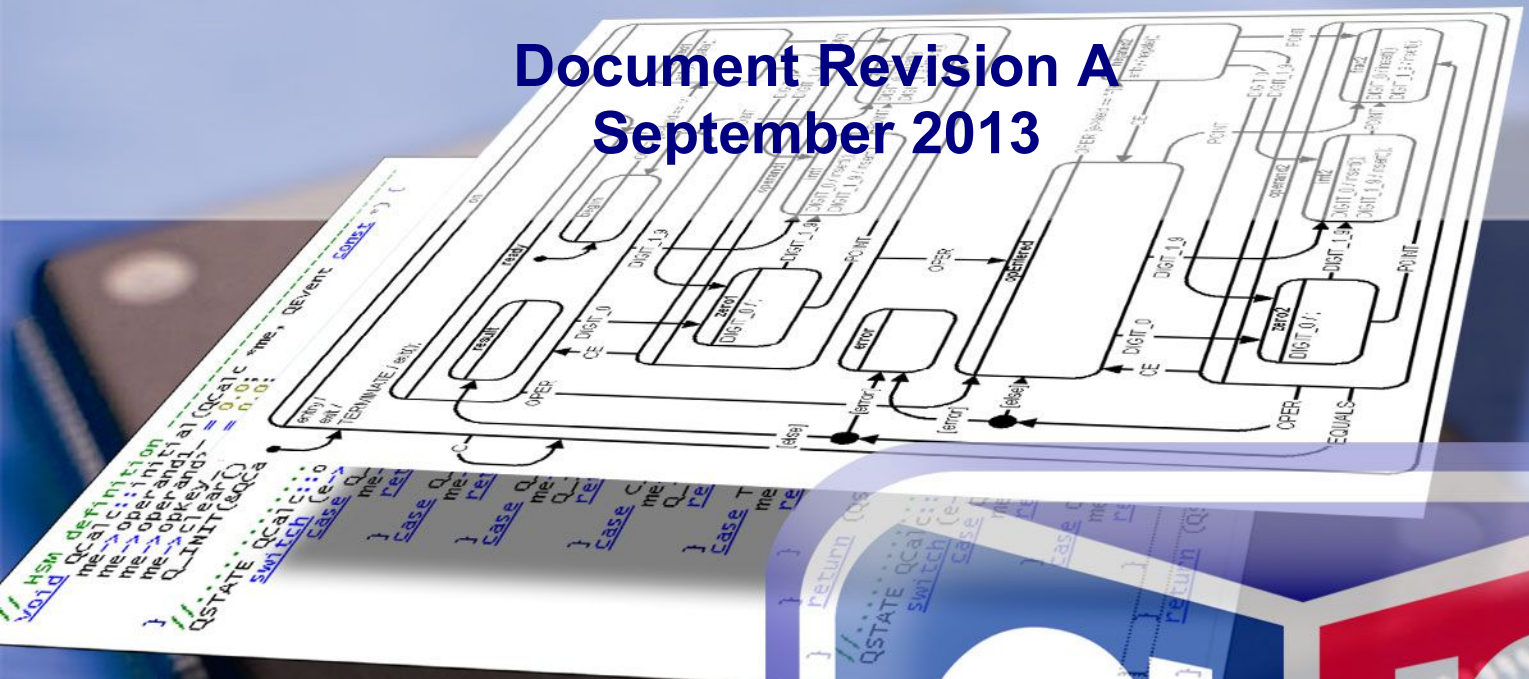


Quantum[®]Leaps
innovating embedded systems



Getting Started with QP/C++™

Document Revision A
September 2013



Copyright © Quantum Leaps, LLC

info@quantum-leaps.com
www.state-machine.com



Table of Contents

1 Introduction.....	1
2 Obtaining and Installing QP/C++ and Qtools.....	2
2.1 Downloading QP/C++.....	2
2.2 Downloading Qtools.....	2
2.3 Installing QP/C++.....	3
2.4 Installing Qtools.....	4
2.5 Setting Up the QPCPP, QTOOLS, and PATH Environment Variables.....	4
3 Building and Running the Blinky Example.....	6
3.1 Blinky on Windows with MinGW (GNU C/C++ for Windows).....	6
3.2 Blinky on Tiva LauchPad with GNU (Sourcery™ CodeBench).....	8
3.3 Blinky on Tiva LauchPad with IAR (IAR EWARM).....	11
3.4 Blinky on Tiva LauchPad with Keil/ARM (Keil uVision4).....	12
4 Re-building the QP/C++ Libraries.....	13
4.1 QP/C++ Library for Windows with MinGW.....	13
4.2 QP/C++ Library for ARM-Cortex-M with GNU (Sourcery™ CodeBench).....	14
4.3 QP/C++ Library for ARM-Cortex-M with IAR (IAR EWARM).....	16
4.4 QP/C++ Library for ARM-Cortex-M with Keil/ARM MDK.....	17
5 The Blinky State Machine and Code.....	18
6 Creating your Own QP/C++ Projects.....	20
7 Next Steps and Further Reading About QP™ and QM™.....	20
8 Contact Information.....	21

Legal Disclaimers

Information in this document is believed to be accurate and reliable. However, Quantum Leaps does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Quantum Leaps reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

All designated trademarks are the property of their respective owners.



1 Introduction

This document explains how to install the QP/C++™ framework (version 5.1.0 or newer) and how to build and run a very simple “Blinky” QP/C++ application, which blinks a light (such as LED on an embedded board) at a rate of 1Hz (once per second). The “Blinky” example is deliberately kept small and simple to help you get started with the QP/C++ active object framework as quickly as possible.

This document explains how to build and run the following versions of the “Blinky” example:

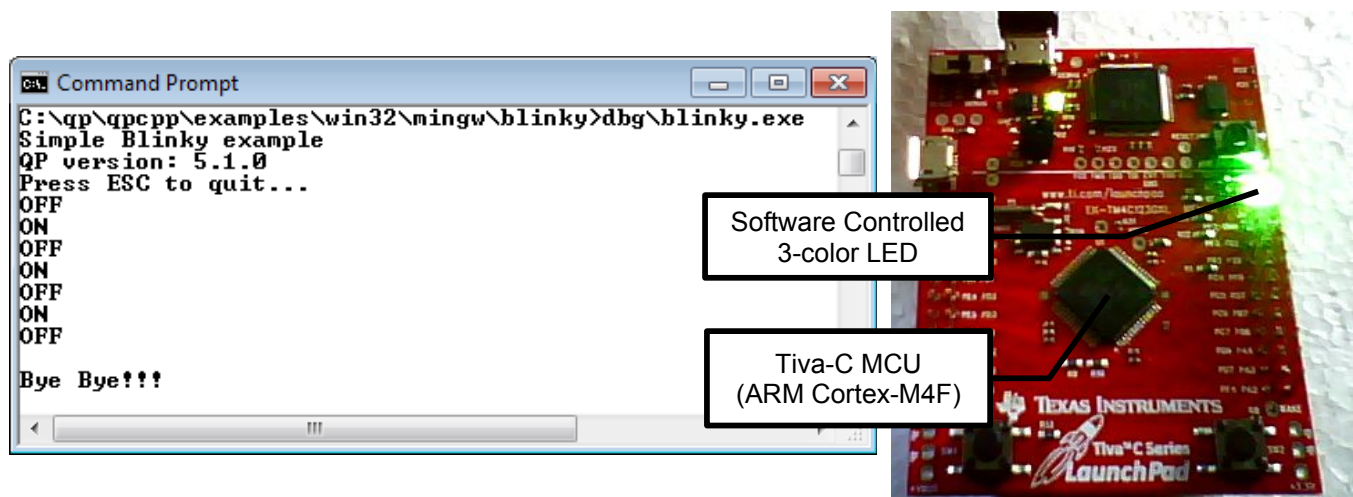
1. Version for **Windows** with the free MinGW C/C++ compiler for Windows (which is included in the Qtools collection).

NOTE: The Blinky example for Windows allows you to experiment with the QP/C++ framework on a standard Windows-based PC **without an embedded board and toolset.**

2. Versions for the Texas Instruments **Tiva™ C Series LaunchPad** board (EK-TM4C123GXL) based on the ARM Cortex-M4F core (see [Figure 1](#)) with the following embedded development toolsets:
 - a) GNU toolset (Sourcery CodeBench)
 - b) IAR EWARM toolset.
 - c) Keil Microcontroller Development Kit (MDK).

NOTE: The Tiva™ LaunchPad (EK-TM4C123GXL) is software-compatible with the slightly earlier Stellaris LaunchPad (EK-LM4F120XL).

Figure 1: Blinky on Windows (left) and on Tiva™ C Series LaunchPad board (right)



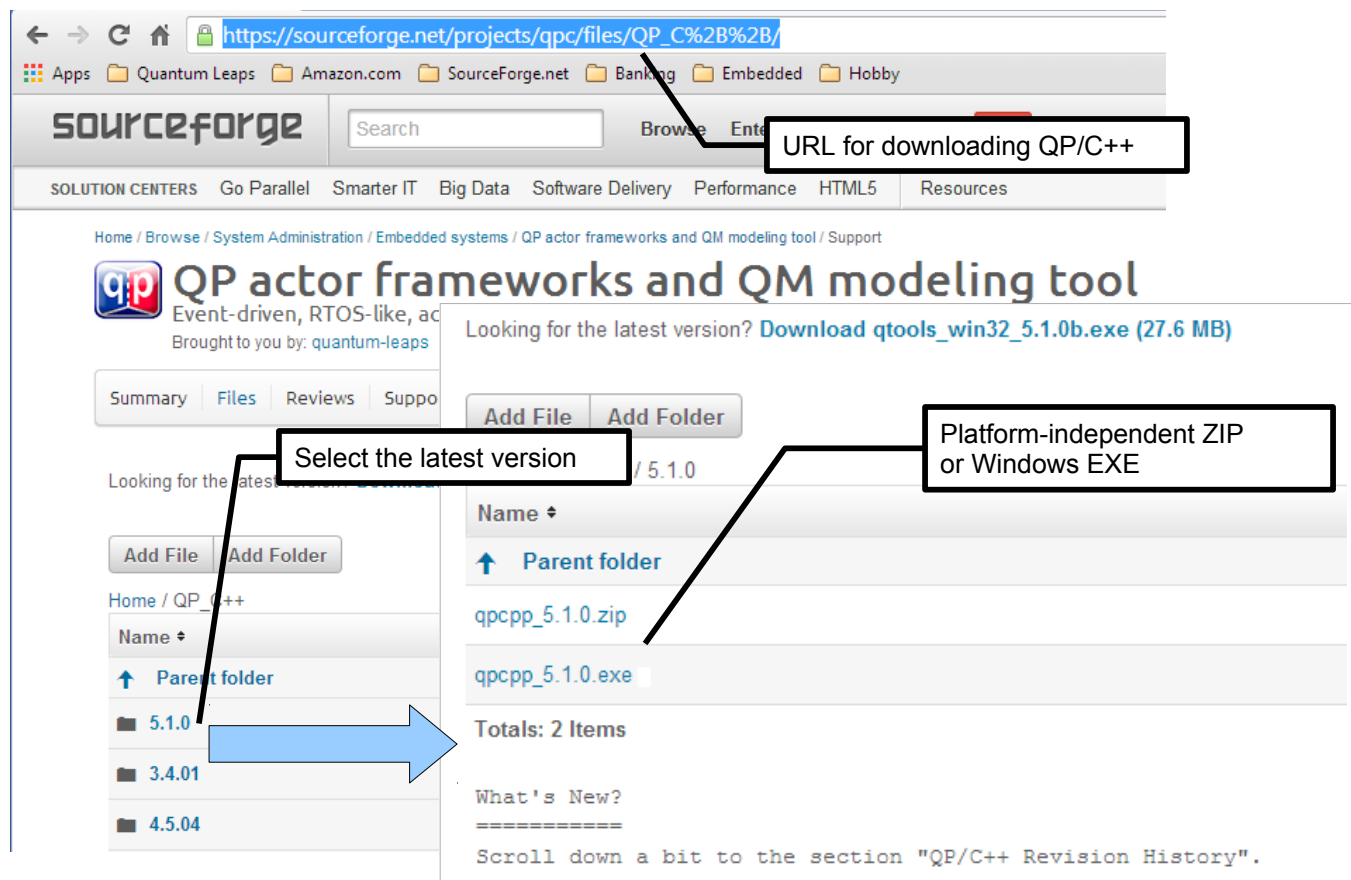
2 Obtaining and Installing QP/C++ and Qtools

This section describes how to download and install the QP/C++ framework and Qtools, the latter being a collection of various open source tools for working with QP, such as the QSPY host application for processing the QS software traces. The Qtools collection for Windows, contains additionally the **GNU make** and the **GNU C/C++ compiler** (MinGW), which you can use to build the Blinky example.

2.1 Downloading QP/C++

The QP/C++ framework is available for download from the [SourceForge.net open source repository](https://sourceforge.net/projects/qpc/files/QP_C%2B%2B/). On SourceForge, select the latest version and download either the platform-independent ZIP or the self-extracting Windows EXE (see Figure 2).

Figure 2: Downloading the QP/C++ framework from SourceForge.net



2.2 Downloading Qtools

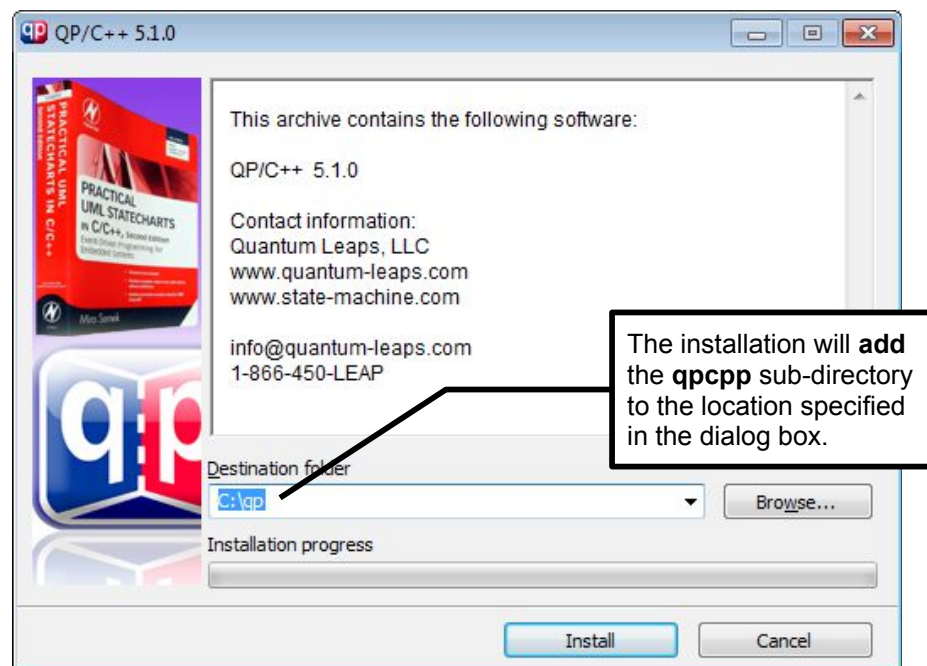
Once you are on SourceForge, you should also download the Qtools collection, which is available at the following URL <http://sourceforge.net/projects/qpcpp/files/Qtools>. From there, select the latest version and download the `qtools_win32_<ver>.exe` self-extracting Windows executable.

2.3 Installing QP/C++

The QP/C++ installation process consist of extracting the ZIP archive or the self-extracting Windows-EXE into a directory of your choice. For the rest of this document, it is assumed that you have installed QP/C++ into the directory `C:\qp\qpcpp`.

NOTE: For your convenience of writing build scripts and make files, it is highly recommended to **avoid spaces** in the QP/C++ installation directory (so, you should **avoid** the standard locations "C:\Program Files" or "C:\Program Files (x86)"). Also, if you wish to run the QP/C++ examples for 16-bit DOS, you need to extract QP/C++ into a directory close to the root of the drive, so that the absolute path to any file in the the example folder **does not exceed the 127-character limit**.

Figure 3: Extracting QP/C++ from the Windows-EXE



The QP/C++ installation copies the QP/C++ source code, ports, and examples to your hard-drive. The following listing shows the main sub-directories comprising the QP/C++ framework.

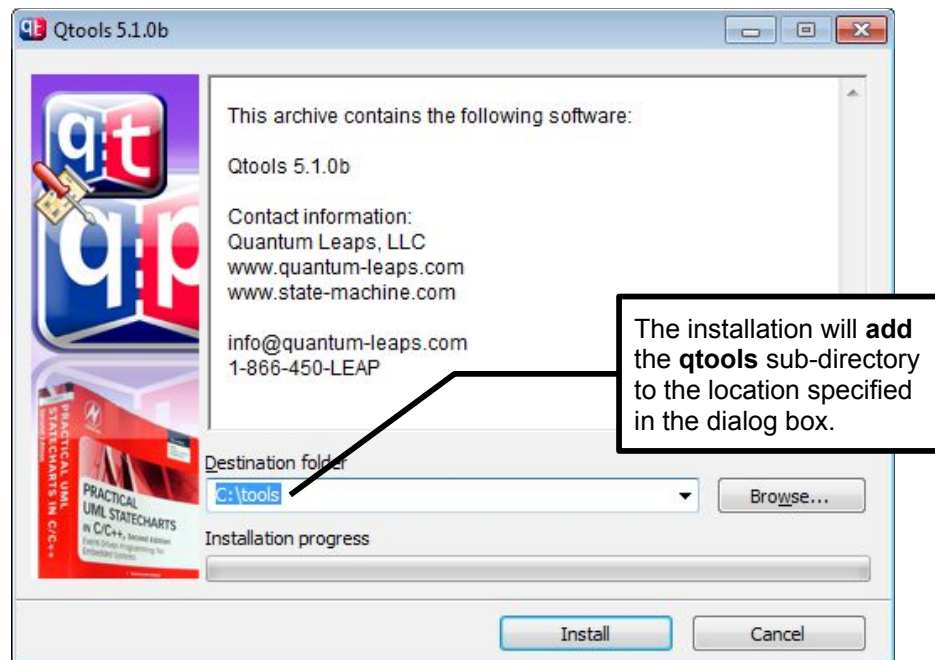
Listing 1: The main sub-directories of the QP/C++ installation.

```
C:\
+-qp\
| +-qpcpp\
| | +-doc\
| | +-examples\
| | +-ports\
| | +-include\
| | +-qep\
| | +-qf\
| | +-qk\
| | +-qs\
- QP directory
- QP/C++ directory
- QP/C++ documentation
- QP/C++ examples
- QP/C++ ports
- QP/C++ platform-independent header files
- QEP/C++ (event processor) source code
- QF/C++ (event-driven framework) source code
- QK/C++ (preemptive kernel) source code
- QS/C++ (software tracing) source code
```

2.4 Installing Qtools

As already mentioned, Qtools is a collection of various open source tools for working with QP. Qtools installation is very similar to installing the QP/C++ framework. You simply run the self-extracting Windows-EXE and choose the Qtools installation directory. For the rest of this document, it is assumed that you have installed Qtools into the directory `C:\tools\qtools`.

Figure 4: Extracting Qtools from the Windows-EXE



2.5 Setting Up the QPCPP, QTOOLS, and PATH Environment Variables

To complete the installation process you should define the `QPCPP` and `QTOOLS` environment variables and you need to append the `%QTOOLS%\bin` directory to the `PATH` variable on your machine. All this will enable you to conveniently run the Qtools utilities and place your QP projects anywhere in your file system, without any particular relation to the location of the QP/C++ framework. The following table shows the environment variables you need to define.

Environment variable	Example setting (adjust to your system)
QPCPP	C:\qp\qpcpp
QTOOLS	C:\tools\qtools
PATH	...;%QTOOLS%\bin (append to the PATH)

NOTE: The project files, make files, and build scripts provided in the QP/C++ distribution assume that the environment variable `QPC` has been defined and will not work if this variable is not provided.

The environment variable editor can be accessed in several ways on Windows. One way is to open the Control Panel and select the “Advanced systems settings” (see Figure 5). Next, you need to click on the “Environment Variables” button, which opens the “Environment Variables” dialog box (see Figure 6).

Figure 5: Getting to the “Advanced systems settings” on Windows 7

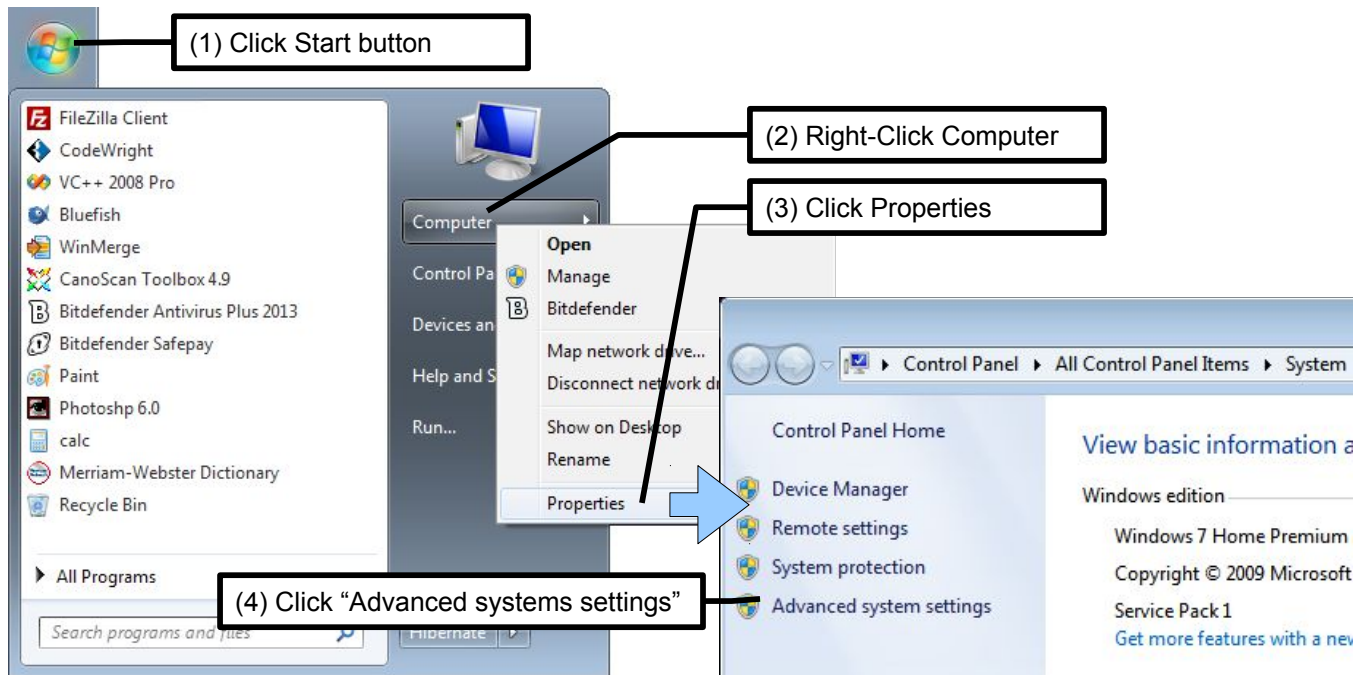
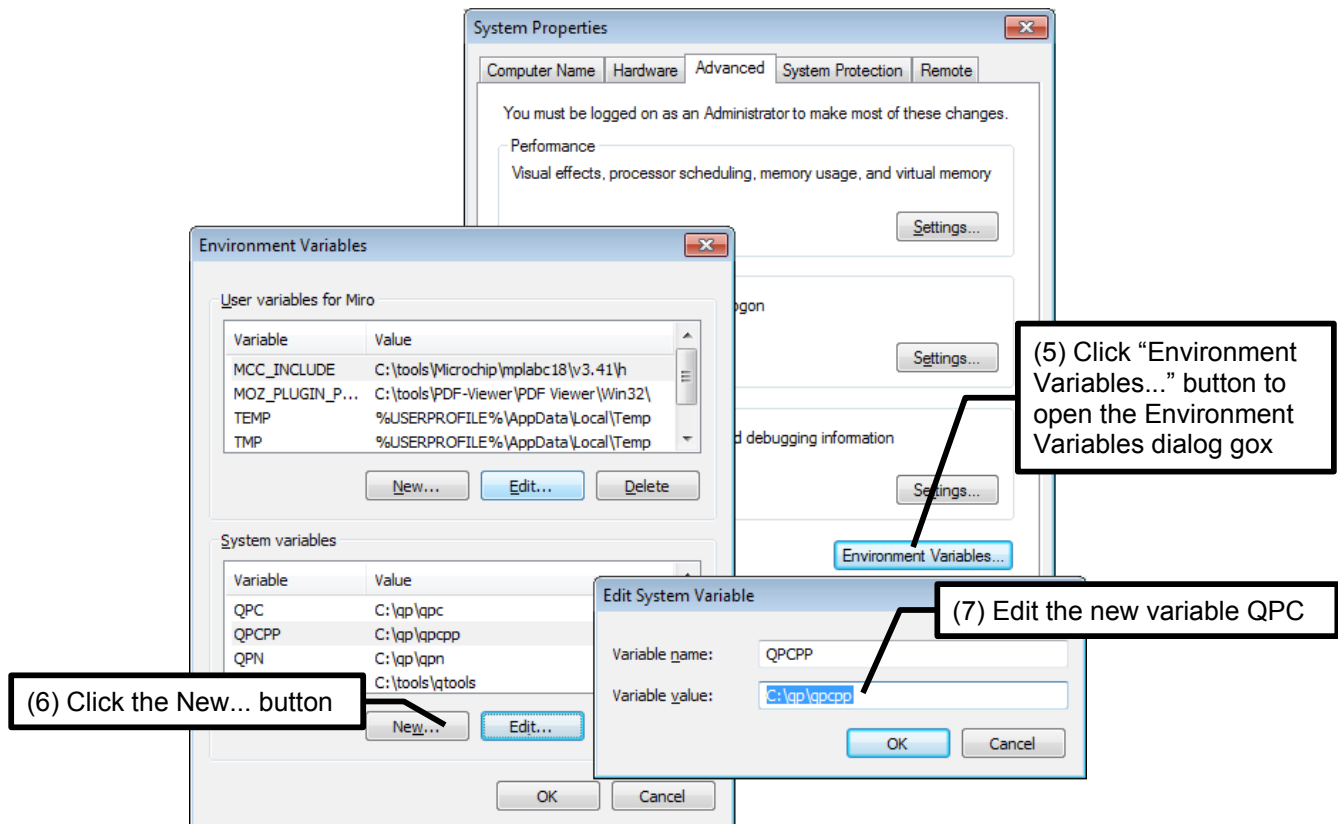


Figure 6: Adding/editing an environment variable in the “Environment Variables” dialog box



3 Building and Running the Blinky Example

This section explains how to build and run the Blinky QP/C++ example on various platforms.

3.1 Blinky on Windows with MinGW (GNU C/C++ for Windows)

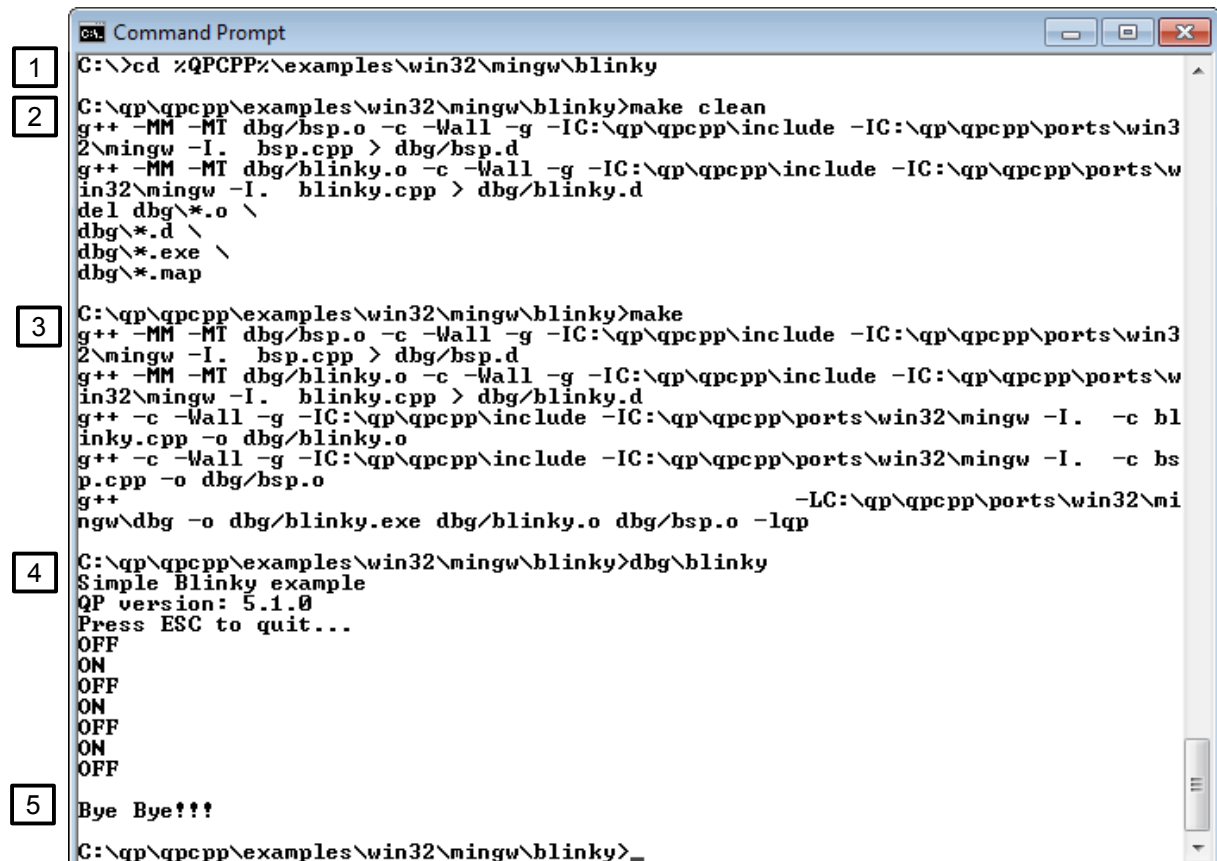
The Windows version of the Blinky example is a simple Windows console application. The example is built with the MinGW toolset and the make utility, which you have already installed with Qttools. The example is located in the `%QPCPP%\examples\win32\mingw\blinky\` directory and is specifically provided so that you don't need any special hardware board or an embedded development toolset to get started with QP/C++.



NOTE: The Blinky source code (`blinky.c`) is actually **the same** on Windows and the embedded boards. The only difference is in the Board Support Package (`bsp.c`), which is specific for the target.

Figure 7 shows the steps of building and running the Blinky example from a Windows command prompt. The explanation section immediately following the figure describes the steps.

Figure 7: Building and running Blinky in a Windows command prompt.



```

1 C:\>cd %QPCPP%\examples\win32\mingw\blinky
2 C:\qp\qpcpp\examples\win32\mingw\blinky>make clean
g++ -MM -MT dbg/bsp.o -c -Wall -g -IC:\qp\qpcpp\include -IC:\qp\qpcpp\ports\win32\mingw -I. bsp.cpp > dbg/bsp.d
g++ -MM -MT dbg/blinky.o -c -Wall -g -IC:\qp\qpcpp\include -IC:\qp\qpcpp\ports\win32\mingw -I. blinky.cpp > dbg/blinky.d
del dbg\*.o \
dbg\*.d \
dbg\*.exe \
dbg\*.map
3 C:\qp\qpcpp\examples\win32\mingw\blinky>make
g++ -MM -MT dbg/bsp.o -c -Wall -g -IC:\qp\qpcpp\include -IC:\qp\qpcpp\ports\win32\mingw -I. bsp.cpp > dbg/bsp.d
g++ -MM -MT dbg/blinky.o -c -Wall -g -IC:\qp\qpcpp\include -IC:\qp\qpcpp\ports\win32\mingw -I. blinky.cpp > dbg/blinky.d
g++ -c -Wall -g -IC:\qp\qpcpp\include -IC:\qp\qpcpp\ports\win32\mingw -I. -c blinky.cpp -o dbg/blinky.o
g++ -c -Wall -g -IC:\qp\qpcpp\include -IC:\qp\qpcpp\ports\win32\mingw -I. -c bsp.cpp -o dbg/bsp.o
g++ -LC:\qp\qpcpp\ports\win32\mingw\dbg -o dbg/blinky.exe dbg/blinky.o dbg/bsp.o -lqp
4 C:\qp\qpcpp\examples\win32\mingw\blinky>dbg\blinky
Simple Blinky example
QP version: 5.1.0
Press ESC to quit...
OFF
ON
OFF
ON
OFF
ON
OFF
5 Bye Bye!!!
C:\qp\qpcpp\examples\win32\mingw\blinky>

```




- (1) Change directory to the Blinky example for Windows with the MinGW compiler. Please note that the command `cd %QPCPP%\examples\win32\mingw\blinky` tests the definition of the `QPC` environment variable. (NOTE: you can also verify the definition by typing `echo %QPCPP%`)
- (2) The `make clean` command invokes the GNU make utility (from the `$QTOOLS$\bin\` directory) to clean the build.
- (3) The `make` command performs the build. The `make` command uses the `Makefile` from the Blinky directory. The printouts following the `make` command are produced by the `gcc` compiler (from the `$QTOOLS$\bin\` directory).

NOTE: The Blinky application links to the **QP/C++ library for Windows**, which is pre-compiled and provided in the standard QP/C++ distribution. The upcoming Section 4.1 describes how you can re-compile the QP/C++ library yourself.

- (4) The `dbg\blinky.exe` command runs the Blinky executable, which is produced in the `dbg\` directory. The output following this command is produced by the Blinky application.
- (5) The Blinky application is exited by pressing the **ESC key**.

3.2 Blinky on Tiva LaunchPad with GNU (Sourcery™ CodeBench)

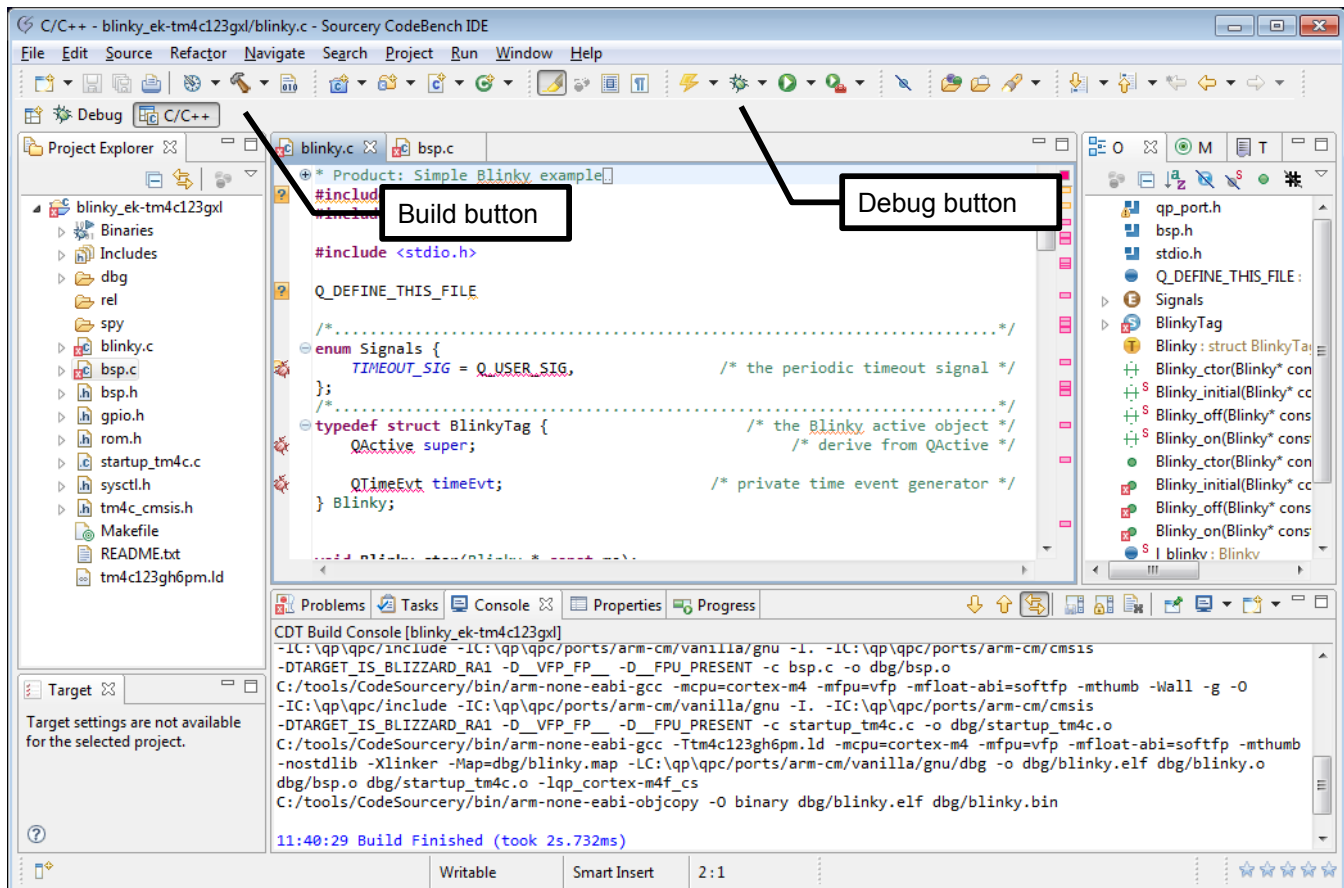
The Sourcery™ CodeBench toolset is an example of the embedded toolset based on the GNU ARM compiler suite. Sourcery for **ARM EABI** (suited for embedded development, sometimes called “free standing” or “bare metal”) can be downloaded from <http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/evaluations/arm-eabi>.



NOTE: The Sourcery™ CodeBench toolset comes in several editions, including the free, command-line based Sourcery Lite edition. The Lite edition can be used to build the Blinky example, but it does not provide any support for downloading the code into the target board and for debugging the code on the target.

The Blinky example described here has been built with the commercial Sourcery™ CodeBench Personal edition (version as of this writing). This version of Sourcery CodeBench offers a free 30-day trial period.

Figure 8: Building and running Blinky in Sourcery CodeBench IDE



NOTE: The Blinky application links to the QP/C++ library for GNU-ARM, which is pre-compiled and provided in the standard QP/C++ distribution. The upcoming Section 4.2 describes how you can re-compile the QP/C++ library yourself.

Once you download and install Sourcery™ CodeBench, you need to import the Blinky project into it. To do so, click on the **File | Import...** menu and chose “Existing Projects into Workspace”. Click Next and in the “Import” dialog box, click the Browse button next to the “Select root directory” box. Choose the `%\examples\arm-cm\vanilla\gnu\blinky_ek-tm4c123gxl\` directory. Once the project opens, you can build it by clicking on the Build button. You can choose the build configuration (Debug, Release, or Spy) from the drop-down menu next to the Build button (see [Figure 8](#)).

To download the code to the Tiva LaunchPad board you might need to edit the Debug Configurations, which you perform from the drop-down menu next to the Debug button (see [Figure 9](#)).

Figure 9: Editing the Debug Configurations in Sourcery CodeBench IDE

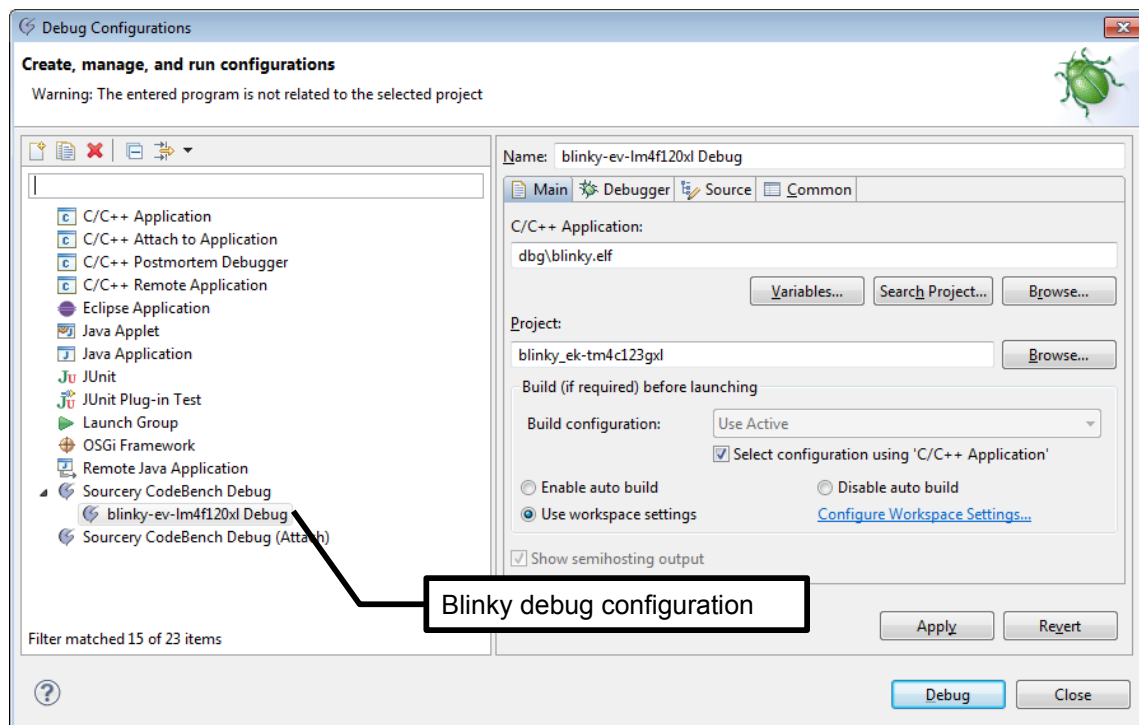
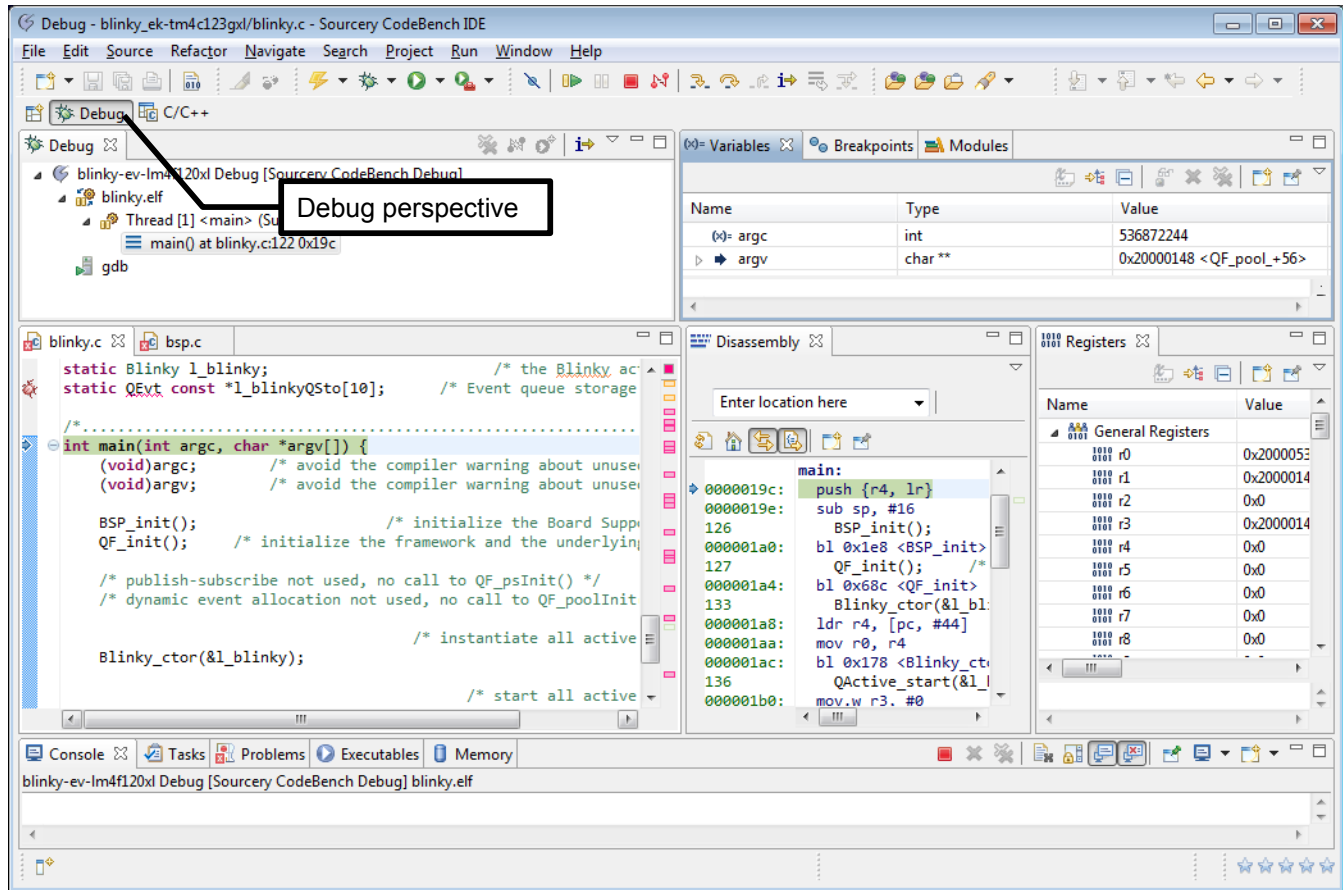


Figure 10: Debugging Blinky in Sourcery CodeBench IDE



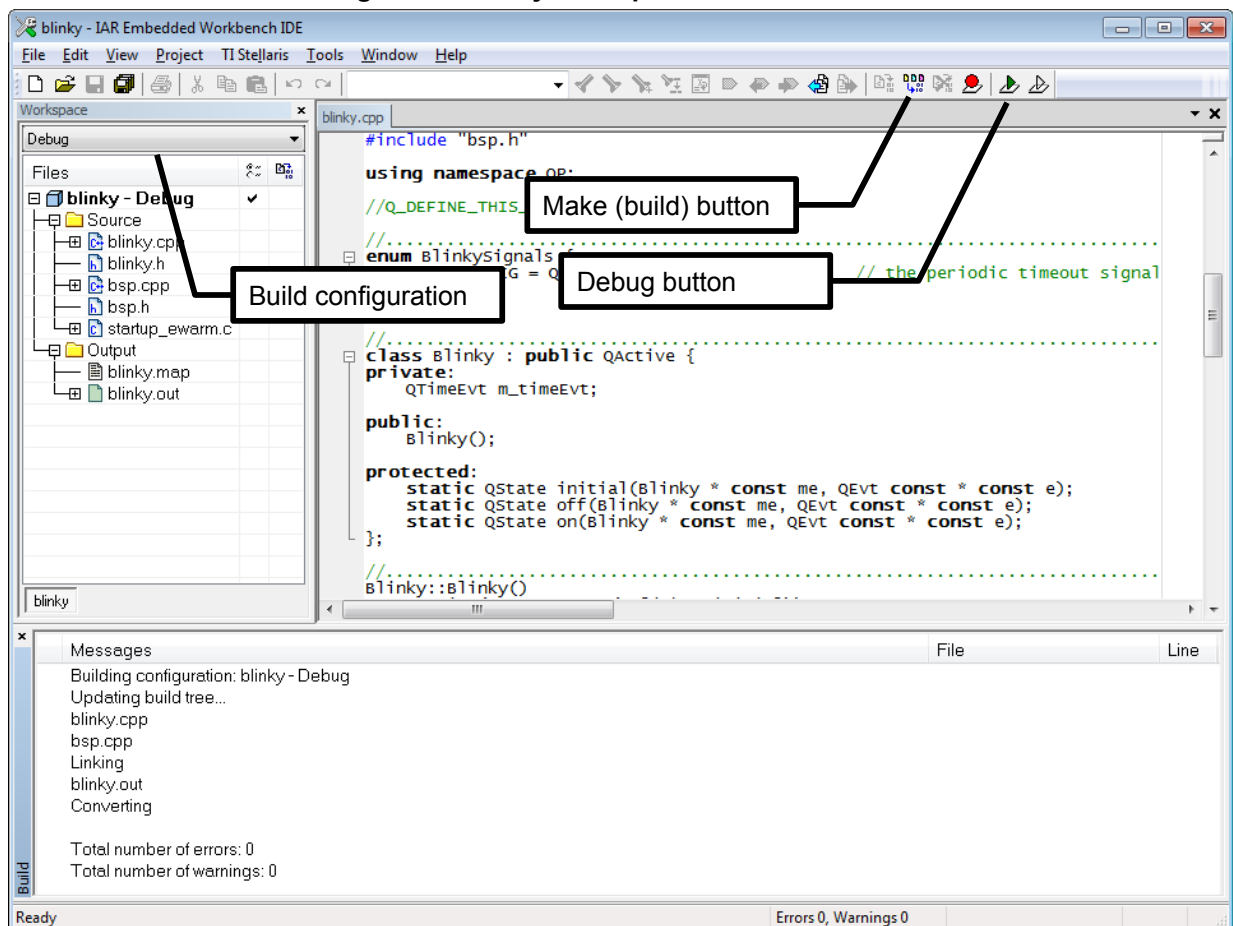
3.3 Blinky on Tiva LaunchPad with IAR (IAR EWARM)

IAR EWARM is an example of commercial toolset (<http://www.iar.com/en/Products/IAR-Embedded-Workbench/ARM/>), which offers superior code generation than GNU ARM and offers much faster code downloads and better debugging experience than Eclipse-based toolsets (such as Sourcery CodeBench).



NOTE: IAR offers a **free** size-limited KickStart version of EWARM as well as time-limited evaluation options. The Blinky example described here has been built with the free KickStart EWARM edition limited to 32KB of generated code.

Figure 11: Blinky workspace in IAR EWARM



The Blinky example for IAR EWARM is located in the directory `%QPCPP%\qcpp\examples\arm-cm\vanilla\iar\blinky_ek-tm4c123gx1\`. To open the Blinky project in EWARM, you can double-click on `blinky.eww` workspace file located in this directory. Once the project opens, you can build it by pressing the Make button. You can also very easily download it to the LaunchPad board and debug it by pressing the Debug button (see Figure 11).

NOTE: The Blinky application links to the **QP/C++ library for IAR EWARM**, which is pre-compiled and provided in the standard QP/C++ distribution. The upcoming Section 4.3 describes how you can re-compile the QP/C++ library yourself.

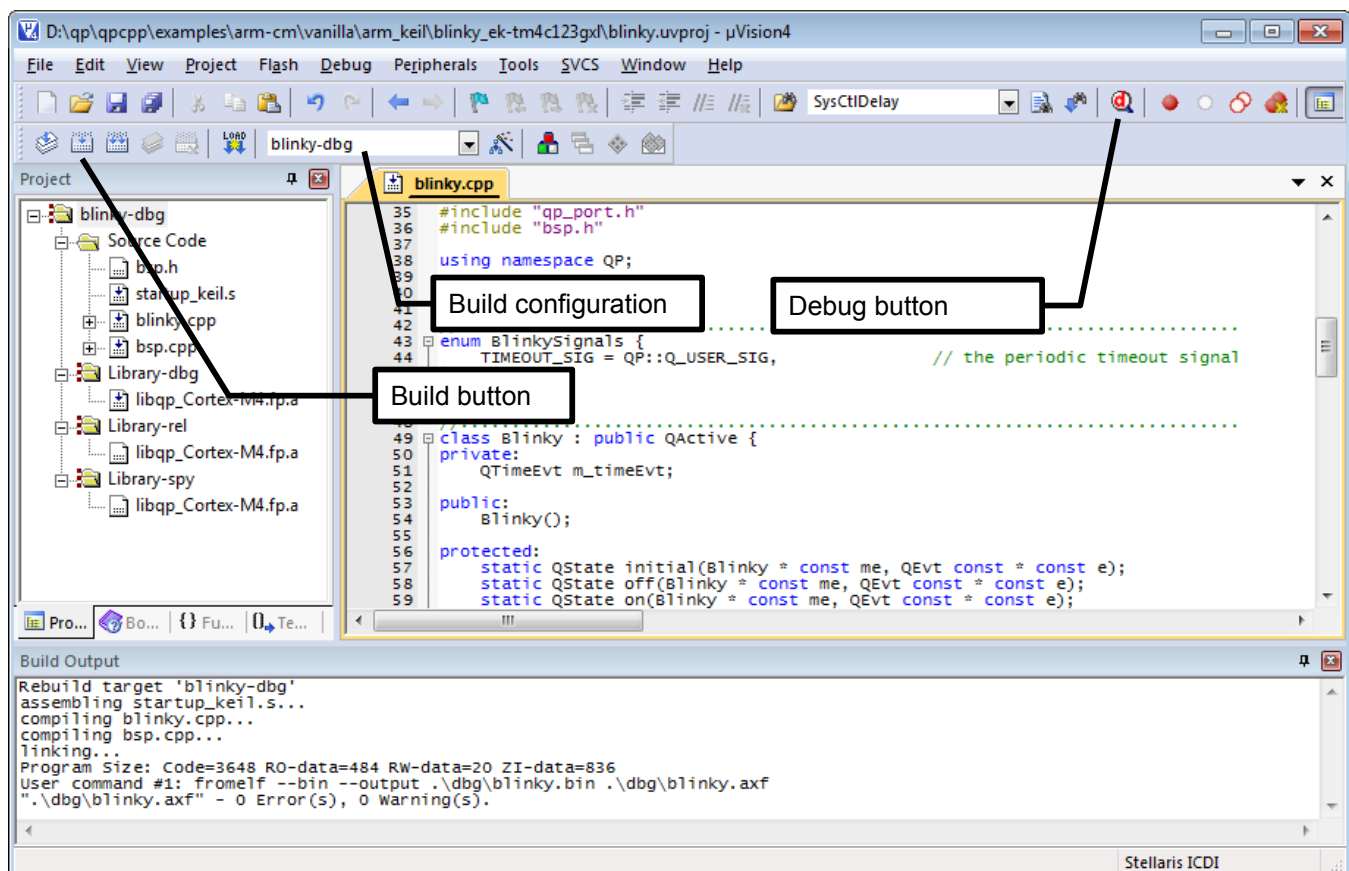
3.4 Blinky on Tiva LaunchPad with Keil/ARM (Keil uVision4)

Keil/ARM MDK is another example of commercial toolset (<http://www.keil.com/arm/mdk.asp>), which offers superior code generation than GNU ARM and offers much faster code downloads and better debugging experience than Eclipse-based toolsets.



NOTE: Keil/ARM offers a **free** size-limited version of Keil MDK as well as time-limited evaluation options. The Blinky example has been built with the free MDK edition limited to 32KB of code.

Figure 12: Blinky workspace in Keil uVision4 IDE



The Blinky example for Keil/ARM is located in the directory %QPCPP%\qpcpp\examples\arm-cm\vanilla\arm_keil\blinky_ek-tm4c123gx1\. To open the Blinky project in Keil uVision4, you can double-click on `blinky.uvproj` project file located in this directory. Once the project opens, you can build it by pressing the Build button. You can also very easily download it to the LaunchPad board and debug it by pressing the Debug button (see [Figure 12](#)).

NOTE: The Blinky application links to the **QP/C++ library for Keil/ARM**, which is pre-compiled and provided in the standard QP/C++ distribution. The upcoming Section 4.4 describes how you can re-compile the QP/C++ library yourself.

4 Re-building the QP/C++ Libraries

QP/C++ is deployed as a library that you statically link to your application. The pre-built QP libraries are provided inside the %QPCPP%\ports\ directory. Normally, you should have no need to re-build the QP libraries. However, if you want to modify QP code or you want to apply different settings, this section describes steps you need to take to rebuild the libraries yourself.

NOTE: To achieve commonality among different development tools, Quantum Leaps software does not use the vendor-specific IDEs, such as IAR, Keil, or Eclipse IDEs, for building the QP libraries. Instead, QP supports *command-line* build process based on simple batch scripts.

NOTE: The QP libraries and QP applications can be built in the following three **build configurations**:

Debug - this configuration is built with full debugging information and minimal optimization. When the QP framework finds no events to process, the framework busy-idles until there are new events to process.

Release - this configuration is built with no debugging information and high optimization. Single-stepping and debugging is effectively impossible due to the lack of debugging information and optimized code, but the debugger can be used to download and start the executable. When the QP framework finds no events to process, the framework puts the CPU to sleep until there are new events to process.

Spy - like the debug variant, this variant is built with full debugging information and minimal optimization. Additionally, it is built with the QP's Q-SPY trace functionality built in. The on-board serial port and the Q-Spy host application are used for sending and viewing trace data. Like the Debug configuration, the QP framework busy-idles until there are new events to process.

4.1 QP/C++ Library for Windows with MinGW

For the MinGW port, you perform a console build with the provided Makefile in %QPCPP%\ports\win32\mingw\ . This Makefile supports three build configurations: Debug, Release, and Spy.

You choose the build configuration by providing the `CONF` argument to the `make`. The default configuration is “dbg”. Other configurations are “rel”, and “spy”. The following table summarizes the commands to invoke `make`.



Table 1 Make commands for the Debug, Release, and Spy configurations

Software Version	Build command
Debug (default)	<code>make</code>
Release	<code>make CONF=rel</code>
Spy	<code>make CONF=spy</code>

NOTE: The provided Makefile assumes that the %QTOOLS%\bin directory is added to the `PATH`.



Figure 13: Building QP/C++ library for Windows with MinGW

```

C:\qp\qpcpp\ports\win32\mingw>make
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qep.o -I. -I.....\include
-I.....\qp\source ..\..\..\qp\source\qep.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qmsm_ini.o -I. -I.....\inc
lude -I.....\qp\source ..\..\..\qp\source\qmsm_ini.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qmsm_dis.o -I. -I.....\inc
lude -I.....\qp\source ..\..\..\qp\source\qmsm_dis.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qfsm_ini.o -I. -I.....\inc
lude -I.....\qp\source ..\..\..\qp\source\qfsm_ini.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qfsm_dis.o -I. -I.....\inc
lude -I.....\qp\source ..\..\..\qp\source\qfsm_dis.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qhsm_dis.o -I. -I.....\inc
lude -I.....\qp\source ..\..\..\qp\source\qhsm_dis.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qhsm_top.o -I. -I.....\inc
lude -I.....\qp\source ..\..\..\qp\source\qhsm_top.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qte_darm.o -I. -I.....\inc
lude -I.....\qp\source ..\..\..\qp\source\qte_darm.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qte_rarm.o -I. -I.....\inc
lude -I.....\qp\source ..\..\..\qp\source\qte_rarm.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qte_ctr.o -I. -I.....\incl
ude -I.....\qp\source ..\..\..\qp\source\qte_ctr.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qf_port.o -I. -I.....\incl
ude -I.....\qp\source qf_port.cpp
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\win32_gui.o -I. -I.....\incl
ude -I.....\qp\source win32_gui.cpp
del dbg\libqp.a
ar rs dbg\libqp.a \
dbg\qep.o dbg\qmsm_ini.o dbg\qmsm_dis.o dbg\qfsm_ini.o dbg\qfsm_dis.o dbg\qhsm_i
ni.o dbg\qhsm_dis.o dbg\qhsm_top.o dbg\qhsm_in.o \
dbg\qa_defer.o dbg\qa_fifo.o dbg\qa_lifo.o dbg\qa_get.o dbg\qa_sub.o dbg\qa_usu
b.o dbg\qa_usuba.o dbg\qeq_init.o dbg\qeq_fifo.o dbg\qeq_lifo.o dbg\qeq_get.o db
g\qf_act.o dbg\qf_gc.o dbg\qf_log2.o dbg\qf_new.o dbg\qf_pool.o dbg\qf_psini.o d
bg\qf_psub.o dbg\qf_pwr2.o dbg\qf_tick.o dbg\qmp_get.o dbg\qmp_init.o dbg\qmp_p
ut.o dbg\qte_ctr.o dbg\qte_arm.o dbg\qte_darm.o dbg\qte_rarm.o dbg\qte_ctr.o db
g\qf_port.o dbg\win32_gui.o \
ar: creating dbg\libqp.a
del dbg\*.o
C:\qp\qpcpp\ports\win32\mingw>

```

4.2 QP/C++ Library for ARM-Cortex-M with GNU (Sourcery™ CodeBench)



The QP/C++ Baseline Code contains the batch file `make_<core>.bat` for building all the libraries located in the `%QPCPP%\ports\arm-cm\vanilla\gnu\` directory. For example, to build the debug version of all the QP libraries for ARM Cortex-M4F, with the GNU ARM compiler, Vanilla kernel, you open a console window on a Windows PC, change directory to `%QPCPP%\ports\arm-cm\vanilla\`, and invoke the batch by typing at the command prompt the following command:

```
make_cortex-m4f_cs
```

The build process should produce the QP library in the location: `<qp>\ports\arm-cm\vanilla\gnu\dbg\`. The batch files assume that the Code Sourcery GNU toolset has been installed in the directory `C:\tools\CodeSourcery\`.

NOTE: You need to adjust the symbol `GNU_ARM` at the top of the batch scripts if you've installed the GNU ARM toolset into a different directory.

In order to take advantage of the QS (“spy”) instrumentation, you need to build the QS version of the QP libraries. You achieve this by invoking the `make cortex-m3 cs.bat` utility with the “spy” target, like this:

```
make_cortex-m4f_cs spy
```

Figure 14: Building QP/C++ library for ARM Cortex-M with the GNU Code Sourcery toolset

```
C:\Command Prompt
```

```
C:\qp\qpcpp\ports\arm-cm\vanilla\gnu>make_cortex-m3_cs.bat  
default selected  
A subdirectory or file .\dbg already exists.  
  
C:\qp\qpcpp\ports\arm-cm\vanilla\gnu>"C:\tools\CodeSourcery"\bin\arm-none-eabi-g  
++ -g -c -mcpu=cortex-m3 -mthumb -Wall -fno-rtti -fno-exceptions -O -I. -I..\..  
...\include -I..\..\..\..\qp\source .....\qp\source\qp.cpp -o.\d  
bg\qep.o  
  
      *   *   *   *   *   *  
  
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qmp_put.o -I. -I..\..\..\incl  
ude -I..\..\..\qf\source .....\qf\source\qmp_put.cpp  
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qte_ctor.o -I. -I..\..\..\inc  
lude -I..\..\..\qf\source .....\qf\source\qte_ctor.cpp  
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qte_arm.o -I. -I..\..\..\incl  
ude -I..\..\..\qf\source .....\qf\source\qte_arm.cpp  
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qte_darm.o -I. -I..\..\..\inc  
lude -I..\..\..\qf\source .....\qf\source\qte_darm.cpp  
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qte_rarm.o -I. -I..\..\..\inc  
lude -I..\..\..\qf\source .....\qf\source\qte_rarm.cpp  
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qte_ctr.o -I. -I..\..\..\incl  
ude -I..\..\..\qf\source .....\qf\source\qte_ctr.cpp  
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\qf_port.o -I. -I..\..\..\incl  
ude -I..\..\..\qf\source qf_port.cpp  
g++ -c -g -O -fno-rtti -fno-exceptions -Wall -odbg\win32_gui.o -I. -I..\..\..\in  
clude -I..\..\..\qf\source win32_gui.cpp  
del dbg\libqp.a  
ar rs dbg\libqp.a \  
qp\qp.o dbg\qsm_ini.o dbg\qsm_dis.o dbg\qfm_ini.o dbg\qfm_dis.o dbg\qhm_i  
ni.o dbg\ghsm_dis.o dbg\ghsm_top.o dbg\ghsm_in.o \  
dbg\qa_defer.o dbg\qa_fifo.o dbg\qa_lifo.o dbg\qa_get.o dbg\qa_sub.o dbg\qa_usu  
b.o dbg\qa_usuba.o dbg\qeq_init.o dbg\qeq_fifo.o dbg\qeq_lifo.o dbg\qeq_get.o d  
bg\qf_act.o dbg\qf_gc.o dbg\qf_log2.o dbg\qf_new.o dbg\qf_pool.o dbg\qf_psi.o d  
bg\qf_pub.o dbg\qf_pwr2.o dbg\qf_tick.o dbg\qmp_get.o dbg\qmp_init.o dbg\qmp_p  
ut.o dbg\qte_ctor.o dbg\qte_arm.o dbg\qte_darm.o dbg\qte_rarm.o dbg\qte_ctr.o db  
g\qf_port.o dbg\win32_gui.o \  
  
ar: creating dbg\libqp.a  
del dbg\*.o  
  
C:\qp\qpcpp\ports\win32\mingw>
```

The make process should produce the QP libraries in the directory: <qp>\ports\arm-cm\vanilla\gnu-\bpy\. You choose the build configuration by providing a target to the make_cortex-m3_cs.bat utility. The default target is “dbg”. Other targets are “rel”, and “spy” respectively. The following table summarizes the targets accepted by make_cortex-m4f_cs.bat.

Table 2 Make targets for the Debug, Release, and Spy software configurations

Software Version	Build command
Debug (default)	make_cortex-m3_cs make_cortex-m4f_cs
Release	make_cortex-m3_cs rel make_cortex-m4f_cs re
Spy	make_cortex-m3_cs spy make_cortex-m4f_cs spy


4.3 QP/C++ Library for ARM-Cortex-M with IAR (IAR EWARM)

The QP/C++ Baseline Code contains the batch file `make_<core>.bat` for building all the libraries located in the `%QPCPP%\ports\arm-cm\vanilla\iar\` directory. For example, to build the debug version of the QP library for Cortex-M4F, Vanilla kernel, with the IAR ARM compiler, you open a console window on a Windows PC, change directory to `%QPCPP%\ports\arm-cm\vanilla\iar\`, and invoke the batch by typing at the command prompt the following command:



```
make_cortex-m4f
```

Figure 15: Building QP/C++ library for ARM Cortex-M with the IAR EWARM toolset



```

C:\qp\qpcpp\ports\arm-cm\vanilla\iar>make_cortex-m4f.bat
default selected
A subdirectory or file .\dbg already exists.

C:\qp\qpcpp\ports\arm-cm\vanilla\iar>iccarm -D DEBUG --debug --endian little --c
pu=cortex-m4f --fpu VFPv4_sp --eem --e --dlib_config "C:\tools\IAR\ARM_6.60\AR
M\INC\c\DLib_Config_Normal.h --diag_suppress Pa050 --no_static_destruction -Ol -
I. -I..\.\.\.\.\.include -I..\.\.\.\.qp\source -o.\dbg\ ..\.\.\.\.qp\source
e\qp.cpp

. . . . .

C:\qp\qpcpp\ports\arm-cm\vanilla\iar>iccarm -D DEBUG --debug --endian little --c
pu=cortex-m4f --fpu VFPv4_sp --eem --e --dlib_config "C:\tools\IAR\ARM_6.60\AR
M\INC\c\DLib_Config_Normal.h --diag_suppress Pa050 --no_static_destruction -Ol -
I. -I..\.\.\.\.\.include -I..\.\.\.\.qp\source -o.\dbg\ ..\.\.\.\.qp\source\
qvanilla.cpp

IAR ANSI C/C++ Compiler V6.60.1.5097/W32 for ARM
Copyright 1999-2013 IAR Systems AB.

202 bytes of CODE memory (+ 158 bytes shared)
0 bytes of CONST memory (+ 48 bytes shared)
14 bytes of DATA memory

Errors: none
Warnings: none

C:\qp\qpcpp\ports\arm-cm\vanilla\iar>iarchive -r .\dbg\libqp_cortex-m4f.a .\dbg\
qa_defer.o .\dbg\qa_fifo.o .\dbg\qa_lifo.o .\dbg\qa_get.o .\dbg\qa_sub.o .\dbg\
qa_usub.o .\dbg\qa_usuba.o .\dbg\qeq_fifo.o .\dbg\qeq_get.o .\dbg\qeq_init.o .\d
bg\qeq_lifo.o .\dbg\qf_act.o .\dbg\qf_gc.o .\dbg\qf_log2.o .\dbg\qf_new.o .\dbg\
qf_pool.o .\dbg\qf_psin.o .\dbg\qf_pspub.o .\dbg\qf_pwr2.o .\dbg\qf_tick.o .\db
g\qmp_get.o .\dbg\qmp_init.o .\dbg\qmp_put.o .\dbg\qte_ctor.o .\dbg\qte_ctr.o .\
dbg\qte_arm.o .\dbg\qte_darm.o .\dbg\qte_rarm.o .\dbg\qvanilla.o
Could Not Find C:\qp\qpcpp\ports\arm-cm\vanilla\iar\dbg\tmp*.
C:\qp\qpcpp\ports\arm-cm\vanilla\iar>_
  
```

The build process should produce the QP library in the location: `<qp>\ports\arm-cm\vanilla\iar\-.dbg\`. The `make.bat` files assume that the ARM toolset has been installed in the directory `C:\tools\IAR\ARM_6.60`.

NOTE: You need to adjust the symbol `IAR_ARM` at the top of the batch scripts if you've installed the IAR ARM compiler into a different directory.

You choose the build configuration by providing a target to the `make_cortex-m4f.bat` utility. The default target is "dbg". Other targets are "rel", and "spy" respectively. The following table summarizes the targets accepted by `make_cortex-m4f.bat`.

4.4 QP/C++ Library for ARM-Cortex-M with Keil/ARM MDK

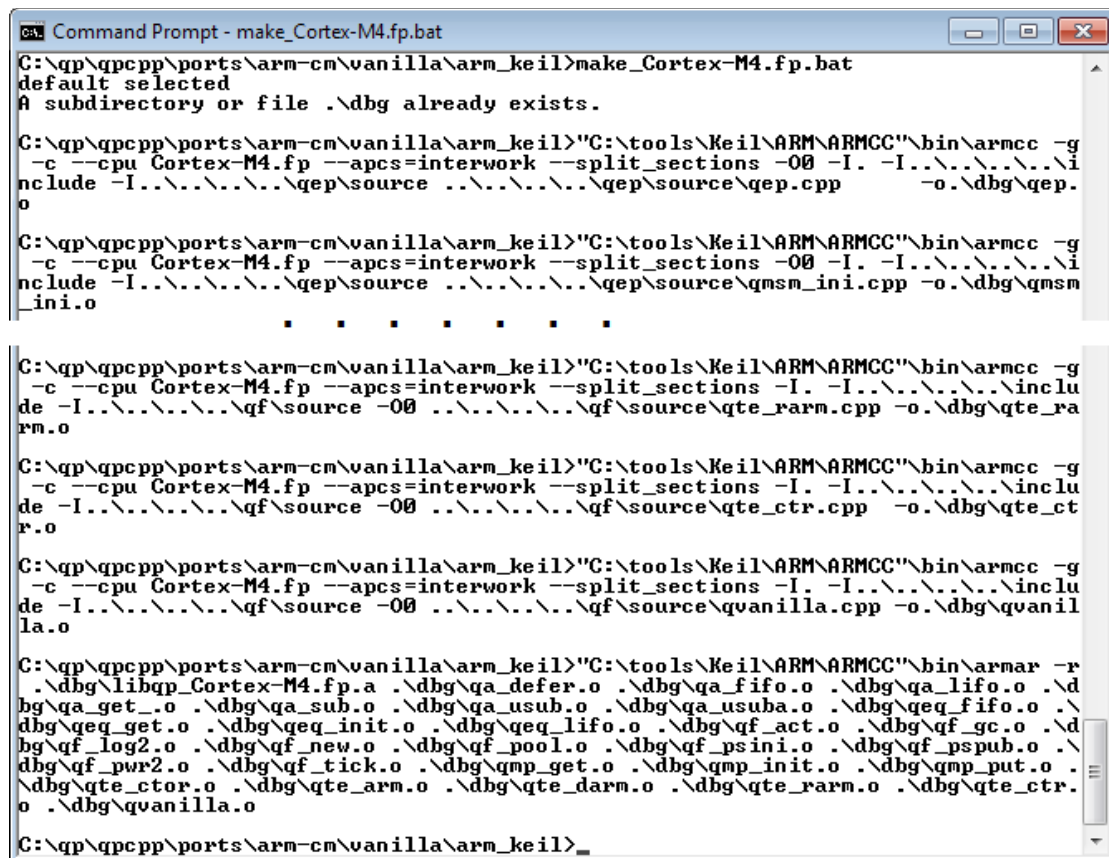
The QP/C++ Baseline Code contains the batch file `make_<core>.bat` for building all the libraries located in the `%QPCPP%\ports\arm-cm\vanilla\arm_keil\` directory.

For example, to build the debug version of the QP library for Cortex-M4F, Vanilla kernel, with the IAR ARM compiler, you open a console window on a Windows PC, change directory to `%QPCPP%\ports\arm-cm\vanilla\arm_keil\`, and invoke the batch by typing at the command prompt the following command:

```
make_Cortex-M4F.fp
```



Figure 16: Building QP/C++ library for ARM Cortex-M with the IAR EWARM toolset



```

C:\qp\qpcpp\ports\arm-cm\vanilla\arm_keil>make_Cortex-M4F.bat
default selected
A subdirectory or file .\dbg already exists.

C:\qp\qpcpp\ports\arm-cm\vanilla\arm_keil>"C:\tools\Keil\ARM\ARMCC"\bin\armcc -g
-c --cpu Cortex-M4.fp --apcs=interwork --split_sections -O0 -I. -I..\.\.\.\.\.\.\.\.\.
include -I..\.\.\.\.\.\.\.\.qp\source ..\.\.\.\.\.qp\source\qep.cpp -o.\dbg\qep.
o

C:\qp\qpcpp\ports\arm-cm\vanilla\arm_keil>"C:\tools\Keil\ARM\ARMCC"\bin\armcc -g
-c --cpu Cortex-M4.fp --apcs=interwork --split_sections -O0 -I. -I..\.\.\.\.\.\.\.
include -I..\.\.\.\.\.qp\source ..\.\.\.\.qp\source\qmsm_ini.cpp -o.\dbg\qmsm
_ini.o

. . . . .

C:\qp\qpcpp\ports\arm-cm\vanilla\arm_keil>"C:\tools\Keil\ARM\ARMCC"\bin\armcc -g
-c --cpu Cortex-M4.fp --apcs=interwork --split_sections -I. -I..\.\.\.\.\.\.\.includ
e -I..\.\.\.\.qp\source -O0 ..\.\.\.qp\source\qte_rarm.cpp -o.\dbg\qte_ra
rm.o

C:\qp\qpcpp\ports\arm-cm\vanilla\arm_keil>"C:\tools\Keil\ARM\ARMCC"\bin\armcc -g
-c --cpu Cortex-M4.fp --apcs=interwork --split_sections -I. -I..\.\.\.\.\.\.\.includ
e -I..\.\.\.qp\source -O0 ..\.\.qp\source\qte_ctr.cpp -o.\dbg\qte_ctr
r.o

C:\qp\qpcpp\ports\arm-cm\vanilla\arm_keil>"C:\tools\Keil\ARM\ARMCC"\bin\armcc -g
-c --cpu Cortex-M4.fp --apcs=interwork --split_sections -I. -I..\.\.\.\.\.\.\.includ
e -I..\.qp\source -O0 ..\.qp\source\qvanilla.cpp -o.\dbg\qvanil
la.o

C:\qp\qpcpp\ports\arm-cm\vanilla\arm_keil>"C:\tools\Keil\ARM\ARMCC"\bin\armar -r
.\dbg\libqp_Cortex-M4.fp.a .\dbg\qa_defer.o .\dbg\qa_fifo.o .\dbg\qa_lifo.o .\d
bg\qa_get.o .\dbg\qa_sub.o .\dbg\qa_usub.o .\dbg\qa_usuba.o .\dbg\qeq_fifo.o .\
dbg\qeq_get.o .\dbg\qeq_init.o .\dbg\qeq_lifo.o .\dbg\qf_act.o .\dbg\qf_gc.o .\d
bg\qf_log2.o .\dbg\qf_new.o .\dbg\qf_pool.o .\dbg\qf_psin.o .\dbg\qf_psub.o .\
dbg\qf_pwr2.o .\dbg\qf_tick.o .\dbg\qmp_get.o .\dbg\qmp_init.o .\dbg\qmp_put.o .
\dbg\qte_ctor.o .\dbg\qte_arm.o .\dbg\qte_darm.o .\dbg\qte_rarm.o .\dbg\qte_ctr
.o .\dbg\qvanilla.o

C:\qp\qpcpp\ports\arm-cm\vanilla\arm_keil>_

```

The build process should produce the QP library in the location: `<qp>\ports\arm-cm\vanilla\arm_keil\dbg\`. The `make_Cortex-M4F.fp.bat` file assumes that the ARM-KEIL toolset has been installed in the directory `C:\tools\Keil\ARM\ARMCC`.

NOTE: You need to adjust the symbol `ARM_KEIL` at the top of the batch scripts if you've installed the ARM-KEIL compiler into a different directory.

You choose the build configuration by providing a target to the `m make_Cortex-M4F.fp.bat` utility. The default target is "dbg". Other targets are "rel", and "spy" respectively. The following table summarizes the targets accepted by `make_Cortex-M4F.fp.bat`.

5 The Blinky State Machine and Code

The behavior of the Blinky example is modeled by a very simple state machine (see Figure 17). The top-most initial transition in this state machine arms a QP time event to deliver the TIMEOUT signal every half second, so that the LED can stay on for one half second and off for the other half. The initial transition leads to state “off”, which turns the LED off in the entry action. When the TIMEOUT event arrives, the “off” state transitions to the “on” state, which turns the LED on in the entry action. When the TIMEOUT event arrives in the “on” state, the “on” state transitions back to “off”, which causes execution of the entry action, in which the LED is turned off. From that point on the cycle repeats forever.

The Blinky state machine shown in Figure 17 is implemented in the `blinky.cpp` source file, as shown in the following listing:

Listing 2 Implementation of the Blinky state machine (file `blinky.cpp`)

```
class Blinky : public QActive {
private:
    QTimeEvt m_timeEvt;

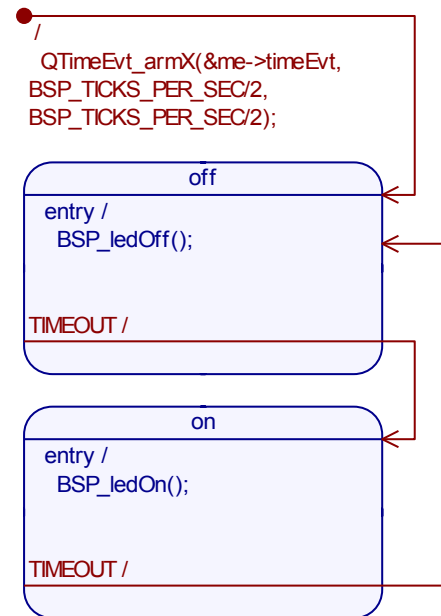
public:
    Blinky();

protected:
    static QState initial(Blinky * const me, QEvt const * const e);
    static QState off(Blinky * const me, QEvt const * const e);
    static QState on(Blinky * const me, QEvt const * const e);
};

//.....
Blinky::Blinky()
: QActive(Q_STATE_CAST(&Blinky::initial)),
  m_timeEvt(this, TIMEOUT_SIG, 0U)
{
    // empty
}

// HSM definition -----
QState Blinky::initial(Blinky * const me, QEvt const * const e) {
    (void)e; // avoid compiler warning about unused argument
    // arm the time event to expire in half a second and every half second
    me->m_timeEvt.armX(BSP_TICKS_PER_SEC/2U, BSP_TICKS_PER_SEC/2U);
    return Q_TRAN(&Blinky::off);
}
//.....
```

Figure 17: Blinky state machine





```
QState Blinky::off(Blinky * const me, QEvt const * const e) {
    QState status;
    switch (e->sig) {
        case Q_ENTRY_SIG: {
            BSP_ledOff();
            status = Q_HANDLED();
            break;
        }
        case TIMEOUT_SIG: {
            status = Q_TRAN(&Blinky::on);
            break;
        }
        default: {
            status = Q_SUPER(&QHsm::top);
            break;
        }
    }
    return status;
}

//.....
QState Blinky::on(Blinky * const me, QEvt const * const e) {
    QState status;
    switch (e->sig) {
        case Q_ENTRY_SIG: {
            BSP_ledOn();
            status = Q_HANDLED();
            break;
        }
        case TIMEOUT_SIG: {
            status = Q_TRAN(&Blinky::off);
            break;
        }
        default: {
            status = Q_SUPER(&QHsm::top);
            break;
        }
    }
    return status;
}
```

As you can see, the structure of the state machine is very clearly recognizable in this code. Please refer to the [PSiCC2] book for exact explanation of the state machine coding techniques.

NOTE: The Blinky source code (blinky.cpp) is actually **the same** on all platforms, including Windows and the embedded boards. The only difference is in the Board Support Package (bsp.cpp), which is specific for the target.

6 Creating your Own QP/C++ Projects

Perhaps the most important fact of life to remember is that in embedded systems nothing works until everything works. This means that you should always start with a **working** system and gradually evolve it, changing one thing at a time and making sure that it keeps working every step of the way.

Keeping this in mind, the provided QP/C++ application examples, such as the super-simple Blinky, or a bit more advanced DPP (dining philosophers) or a "Fly 'n' Shoot" game, allow you to get started with a working project rather than starting from scratch. You should also always try one of the provided example projects on the same evaluation board that it was designed for, before making any changes.

Only after convincing yourself that the example project works "as is", you can think about creating your own projects. At this point, the easiest and recommended way is to copy the existing working example project folder (such as the Blinky example) and rename it.

After copying the project folder, you still need to change the name of the project/workspace. The easiest and safest way to do this is to open the project/workspace in the corresponding IDE and use the Save As... option to save the project under a different name. You can do this also with the QM model file, which you can open in QM and "Save As" a different model.

NOTE: By copying and re-naming an **existing, working project**, as opposed to creating a new one from scratch, you inherit the correct compiler and linker options and other project settings, which will help you get started much faster.

7 Next Steps and Further Reading About QP™ and QM™

This quick-start guide is intended to get the QP/C++ installed and running on your system as quickly as possible, but to work with QP/C++ effectively, you need to learn a bit more about active objects and state machines. For example, this quick guide didn't cover posting events among multiple active objects, publishing events, QS software tracing, or modeling with the QM tool. Below is a list of links to enable you to further your knowledge:

- The book "Practical UML Statecharts in C/C++, 2nd Edition" [PSiCC2] and the companion web-page to the book (<http://www.state-machine.com/psicc2/>)
- Free Support Forum for QP/QM (<https://sourceforge.net/p/qpcpp/discussion/668726>)
- QP Code Downloads summary (<http://www.state-machine.com/downloads>)
- QP Application Notes (<http://www.state-machine.com/resources/appnotes.php>)
- QP Articles (<http://www.state-machine.com/resources/articles.php>)
- "State Space" Blog (<http://embeddedgurus.com/state-space/>)

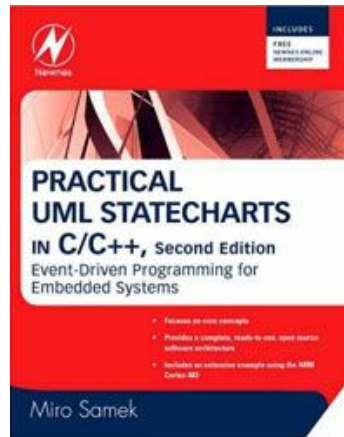
8 Contact Information

Quantum Leaps, LLC
103 Cobble Ridge Drive
Chapel Hill, NC 27516
USA

+1 866 450 LEAP (toll free, USA only)
+1 919 869-2998 (FAX)

e-mail: info@quantum-leaps.com

WEB : <http://www.quantum-leaps.com>
<http://www.state-machine.com>



“Practical UML Statecharts in C/C++, Second Edition: Event Driven Programming for Embedded Systems”, by Miro Samek, Newnes, 2008

