



Quantum[®]Leaps

Modern Embedded Software

Getting Started with QP™ Real-Time Embedded Frameworks

Document Revision H
October 2021

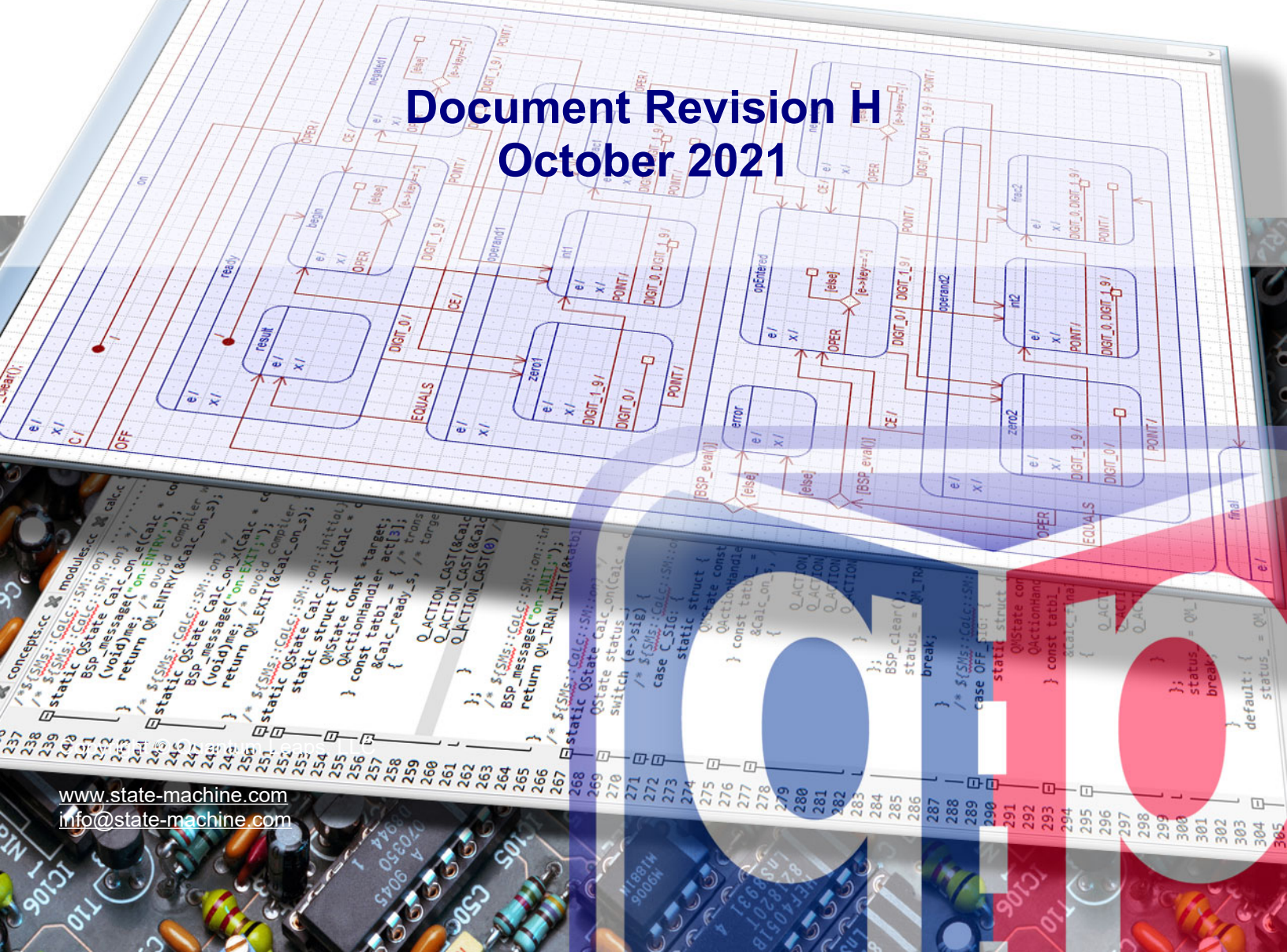


Table of Contents

1 Introduction.....	1
2 Obtaining and Installing QP-bundle.....	2
2.1 Downloading QP-bundle.....	2
2.2 Installing QP/C.....	3
3 Building and Running the Blinky Example.....	5
3.1 Blinky on Windows with MinGW (GNU C/C++ for Windows).....	6
3.2 Blinky on Tiva LaunchPad with GNU-ARM (Makefile Project).....	7
3.3 Blinky on Tiva LaunchPad with Keil/ARM (Keil uVision).....	10
3.4 Blinky on Tiva LaunchPad with IAR (IAR EWARM).....	11
4 The Blinky State Machine and Code.....	12
5 Re-building the QP/C Libraries for Windows.....	15
5.1 QP/C Library for Windows with MinGW.....	15
6 Creating Your Own QP/C Projects.....	16
7 Next Steps and Further Reading About QP™ and QM™.....	16
8 Contact Information.....	17

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

This document explains how to install the **QP/C** and **QP/C++** real-time embedded frameworks (version **6.x** or newer) and how to build and run a very simple “Blinky” QP 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:



NOTE: To focus the discussion, this document describes the process for the **QP/C** framework, but the process is almost identical for **QP/C++**.

1. Version for **Windows** with the free MinGW C/C+ compiler for Windows, which allows you to experiment with the QP/C framework on a standard Windows-based PC **without an embedded board and toolchain**.
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 toolchains:
 - a) GNU-ARM toolchain (Makefile Project and CCS Eclipse project)
 - b) ARM-KEIL toolchain (Microcontroller Development Kit MDK).
 - c) IAR EWARM toolchain.

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



2 Obtaining and Installing QP-bundle

This section describes how to download and install everything you need to get started: QP/C and QP/C++ frameworks, the QM modeling tool, and the QTools collection.

NOTE: The QTools collection for Windows, contains additionally the **GNU make, Python**, and the **GNU C/C++ compilers** for Windows (**MinGW**) and for ARM Cortex-M (**GNU-ARM**), which you can use to build the Blinky example. **It is highly recommend to install QTools along with the QP frameworks to get these basic command-line tools for Windows.**

2.1 Downloading QP-bundle

The QP-bundles for various host operating systems are available for download from the state-machine.com website. The website offers the latest downloads for the Windows, Linux, and macOS hosts (see Figure 2).

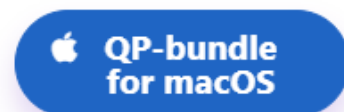
Figure 2: QP-bundle downloads from state-machine.com



Download & Try it!

It's easier than you think...

The following **free downloads** contain everything you need to get started: the QP™ frameworks, the QM™ modeling tool and the QTools™ collection bundled together in a single, streamlined **QP-bundle**:



NOTE: The QP frameworks can be also downloaded directly from [GitHub](https://github.com) repositories. In each case it is recommended to download the pre-packaged **releases**.

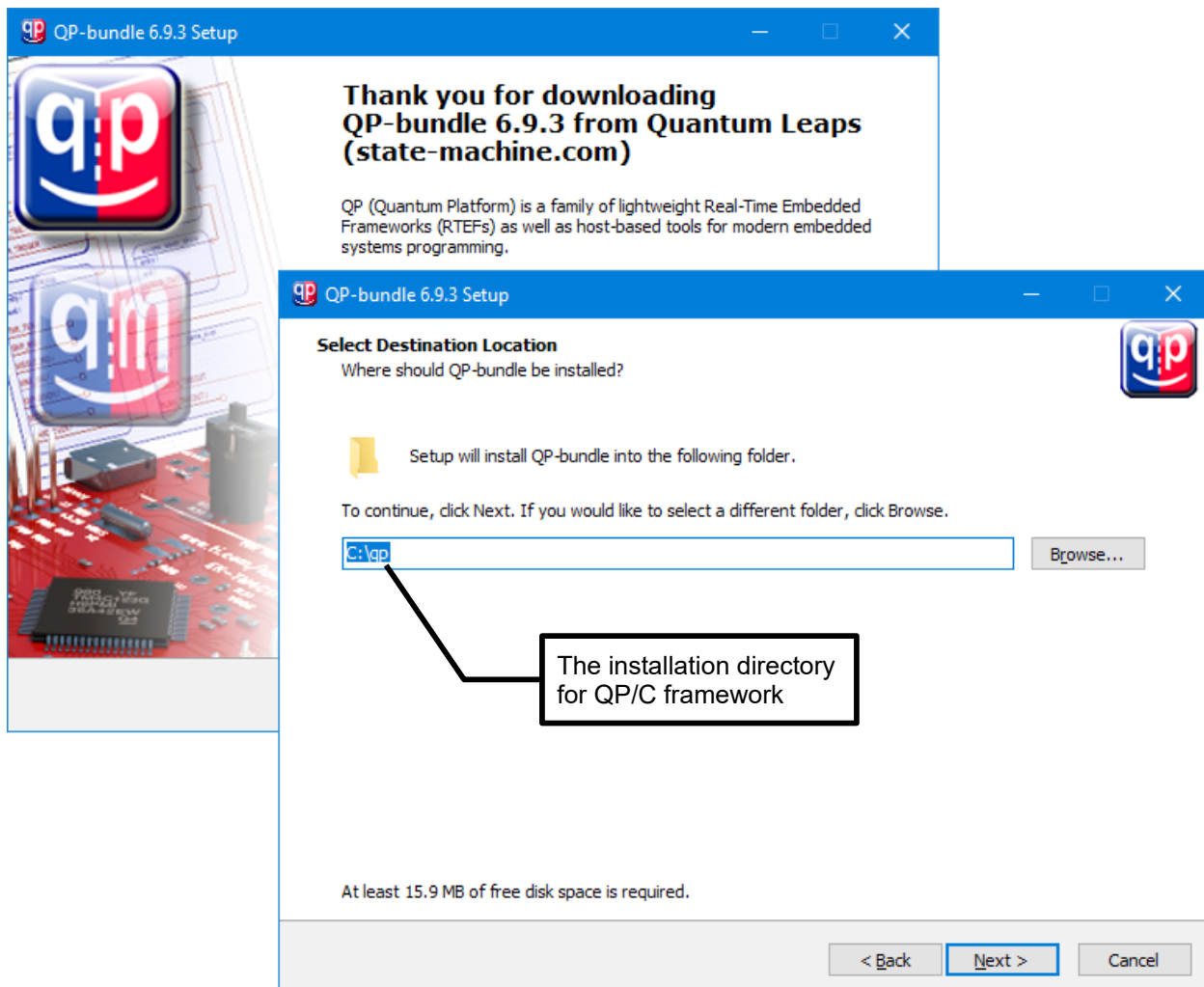
NOTE: The installer application is recommended for installing QP on Windows, because it is **digitally signed** by Quantum Leaps. However, if you are allergic to running installers, you can also install QP on Windows from the provided ZIP files (choose "Download QP on Linux/macOS").

2.2 Installing QP/C

The QP/C installation process on Windows consist of running the Windows installer, while installation on Linux/macOS consists of extracting the ZIP archive into a directory of your choice. For the rest of this document, it is assumed that you have chosen the default installation directory `C:\qp`.

NOTE: For your convenience of writing build scripts and make files, it is highly recommended to **avoid spaces** in the QP installation directory (so, you should **avoid** the standard locations “C:\Program Files” or “C:\Program Files (x86)”).

Figure 3: Windows installer for QP/C



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

NOTE: The main components like QP/C, QP/C++, QM and QTools can be also downloaded separately from [GitHub](https://github.com) repositories. In each case it is recommended to download the pre-packaged releases.

Listing 1: The main sub-directories of the QP-bundle installation (on Windows).

```
C:\qp\
| +-qm\          - QM directory (modeling tool)
|
| +-qpc\         - QP/C directory
| | +-3rd_party\ - 3rd-party software used in QP/C examples
| | +-examples\  - QP/C examples
| | +-html\      - QP/C offline documentation in HTML
| | +-include\   - QP/C platform-independent header files
| | +-ports\     - QP/C ports
| | +-src\       - QP/C platform-independent source code
|
| +-qpcpp\       - QP/C++ directory (similar structure to QP/C)
|
| +-qtools\      - QTools directory (Windows version)
| | +-bin\
| | | +-make.exe - the make utility for Windows
| | | +-qspy.exe - the QSPY host utility
| | | +-...
| | |
| | +-qspy\      - QSPY source code
| | +-qutest\    - QUTest unit testing (Python)
| | +-qview\     - QView monitoring (Python)
| | |
| | +-mingw32\   - MinGW GNU-C/C++ compiler for Windows
| | +-gnu_arm-none-eabi\ - GNU-C/C++ cross compiler for ARM Cortex-M/R
| | +-Python3\   - Python-3 for Windows
```

NOTE: It is highly recommended to watch the ["Getting Started with QP" Video](#)

Figure 4: Getting Started Video



3 Building and Running the Blinky Example

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

NOTE: The QP/C 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.

NOTE: All examples for *embedded boards* include the QP/C framework as **source code** within the projects, instead of statically linking with a QP/C library.

NOTE: Examples for desktop Windows still use QP/C as pre-built libraries.

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

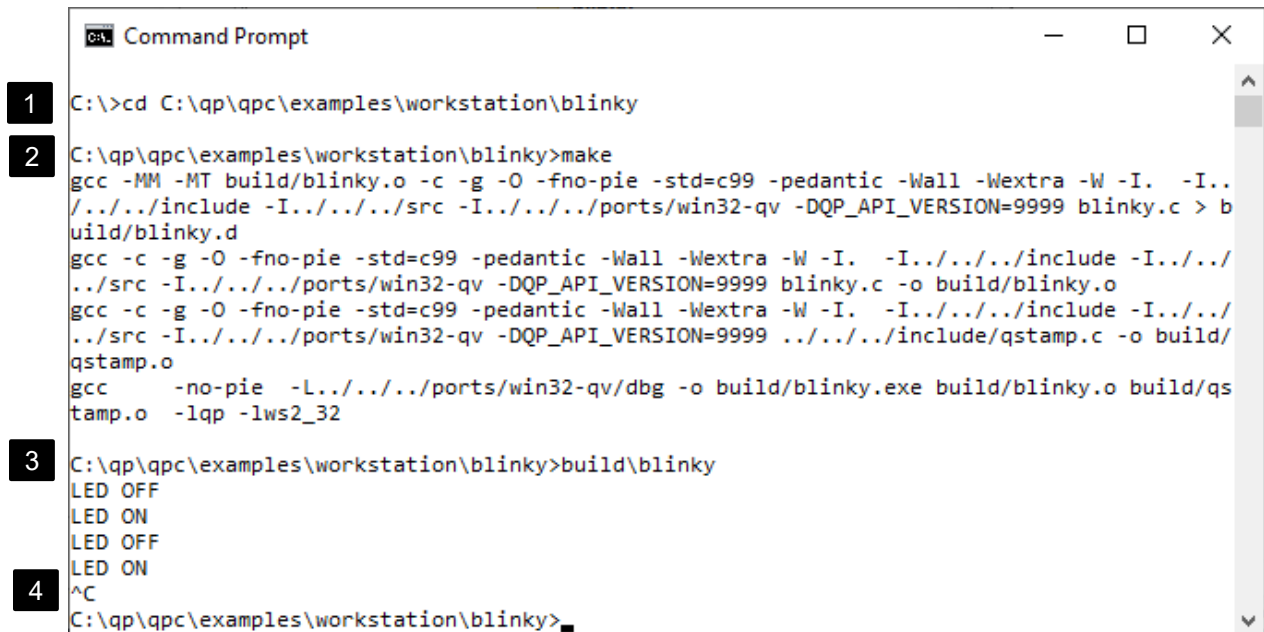
The Windows version of the Blinky example is a simple console application. The example is built with the MinGW toolchain and the `make` utility, which you have already installed with QTools. The example is located in the directory `qpc\examples\workstation\blinky` and is specifically provided so that you don't need an embedded board or a development toolchain 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 implemented differently on Windows than in for the embedded boards.

Figure 5 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 5: Building and running Blinky in a Windows command prompt.



```

C:\>cd C:\qp\qpc\examples\workstation\blinky

C:\qp\qpc\examples\workstation\blinky>make
gcc -MM -MT build/blinky.o -c -g -O -fno-pie -std=c99 -pedantic -Wall -Wextra -W -I. -I../.../include -I../.../src -I../.../ports/win32-qv -DQP_API_VERSION=9999 blinky.c > build/blinky.d
gcc -c -g -O -fno-pie -std=c99 -pedantic -Wall -Wextra -W -I. -I../.../include -I../.../src -I../.../ports/win32-qv -DQP_API_VERSION=9999 blinky.c -o build/blinky.o
gcc -c -g -O -fno-pie -std=c99 -pedantic -Wall -Wextra -W -I. -I../.../include -I../.../src -I../.../ports/win32-qv -DQP_API_VERSION=9999 ../.../include/qstamp.c -o build/qstamp.o
gcc -no-pie -L../.../ports/win32-qv/dbg -o build/blinky.exe build/blinky.o build/qstamp.o -lqp -lws2_32

C:\qp\qpc\examples\workstation\blinky>build\blinky
LED OFF
LED ON
LED OFF
LED ON
^C

C:\qp\qpc\examples\workstation\blinky>
  
```

- [1] Change directory to the Blinky example for Windows. The command “`cd C:\qp\qpc\examples\arm-cm\blinky_ek-tm4c123gx1\win32`” assumes that QP/C has been installed in the default directory `C:\qp\qpc`.
- [2] 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.

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 5.1 describes how you can re-compile the QP/C library yourself.

- [3] The “`build\blinky.exe`” command runs the Blinky executable, which is produced in the `build\` directory. The output following this command is produced by the Blinky application.
- [4] The Blinky application is exited by pressing the **Ctrl-C key**.

The **GNU-ARM** toolchain used in these `Makefiles` is now part of the QTools collection for Windows. It has been downloaded and adapted from (<http://gnutoolchains.com/arm-eabi/>). This pre-built toolchain is an example of an open-source toolchain, which offers acceptable code generation, but no support for code download or debugging. To get these features, you would need to use IDE's (typically based on Eclipse), such as TI Code Composer Studio (CCS), Atollic TrueSTUDIO, and many others.

NOTE: The `gnu\` sub-directory contains project files for TI Code Composer Studio (CCS) that you can immediately import into the CCS Eclipse-based IDE.

Figure 6: Building and Blinky in the Command Prompt Window

```
C:\>cd C:\qp\qpc/examples/arm-cm\blinky_ek-tm4c123gx1\qk\gnu
```

```
C:\qp\qpc/examples/arm-cm\blinky_ek-tm4c123gx1\qk\gnu>make clean  
rm dbg/*.o \\  
dbg/*.ld \\  
dbg/*.bin \\  
dbg/*.elf \\  
dbg/*.map
```

```
C:\qp\qpc/examples/arm-cm\blinky_ek-tm4c123gx1\qk\gnu>make  
C:/tools/gnu_arm-eabi/bin/arm-eabi-gcc -MM -MT dbg/qk_mutex.o -g -mcpu=cortex-m4  
-mfpu=vfp -mfloat-abi=softfp -mthumb -Wall -ffunction-sections -fdata-sections  
-O -I.....-I.....-include -I.....-source -I.....-por  
ts/arm-cm/qk.gnu -I.....3rd_party\CMSIS/include -I.....3rd_p  
arty/ek-tm4c123gx1 -D_ARM_ARCH=? -D_FPU_PRESENT .....source/qk_m  
utex.c > dbg/qk_mutex.a  
C:/tools/gnu_arm-eabi/bin/arm-eabi-gcc -MM -MT dbg/qk.o -g -mcpu=cortex-m4 -mfpu  
=vfp -mfloat-abi=softfp -mthumb -Wall -ffunction-sections -fdata-sections -O -I.  
...-I.....-include -I.....-source -I.....ports/arm  
-cm/qk.gnu -I.....3rd_party\CMSIS/include -I.....3rd_party/e  
k-tm4c123gx1 -D_ARM_ARCH=? -D_FPU_PRESENT .....source/qk.c > dbg  
/qk.a  
  
. . . . .  
  
C:/tools/gnu_arm-eabi/bin/arm-eabi-gcc -Tblinky-qk.ld -mcpu=cortex-m4 -mfpu=vfp  
-mfloat-abi=softfp -mthumb -nostdlib -Wl,-Map,dbg/blinky-qk.map,-cref,-gc-sect  
ions -O dbg/blinky-qk.elf dbg/qk_port.o dbg/blinky.o dbg/bsp.o dbg/main.o dbg/s  
ystem_TM4C123GH6PM.o dbg/startup_TM4C123GH6PM.o dbg/gep_hsm.o dbg/gep_msm.o dbg/  
gf_act.o dbg/gf_actq.o dbg/gf_defer.o dbg/gf_dyn.o dbg/gf_mem.o dbg/gf_ps.o dbg/  
gf_qact.o dbg/gf_req.o dbg/gf_qmact.o dbg/gf_time.o dbg/qk.o dbg/qk_mutex.o dbg/  
gstamp.o  
C:/tools/gnu_arm-eabi/bin/arm-eabi-objcopy -O binary dbg/blinky-qk.elf dbg/blink  
y-qk.bin  
C:\qp\qpc/examples/arm-cm\blinky_ek-tm4c123gx1\qk\gnu>
```

NOTE: For the Makefile to work, you need to adjust the Makefile to provide the location of the GNU-ARM toolchain on **your** machine.

- [1] Change directory to the Blinky example for EK-TM4C123GXL board with GNU. The command “`cd C:\qp\qpc\examples\arm-cm\blinky_ek-tm4c123gx1\qv\gnu`” assumes that QP/C has been installed in the default directory `C:\qp\qpc`.
- [2] The “`make clean`” command invokes the GNU `make` utility (from the QTools directory) to clean the build.
- [3] The “`make`” command performs the actual build. The `make` command uses the `Makefile` from the Blinky directory. The printouts following the “`make`” command are produced by the GNU-ARM compiler/linker.

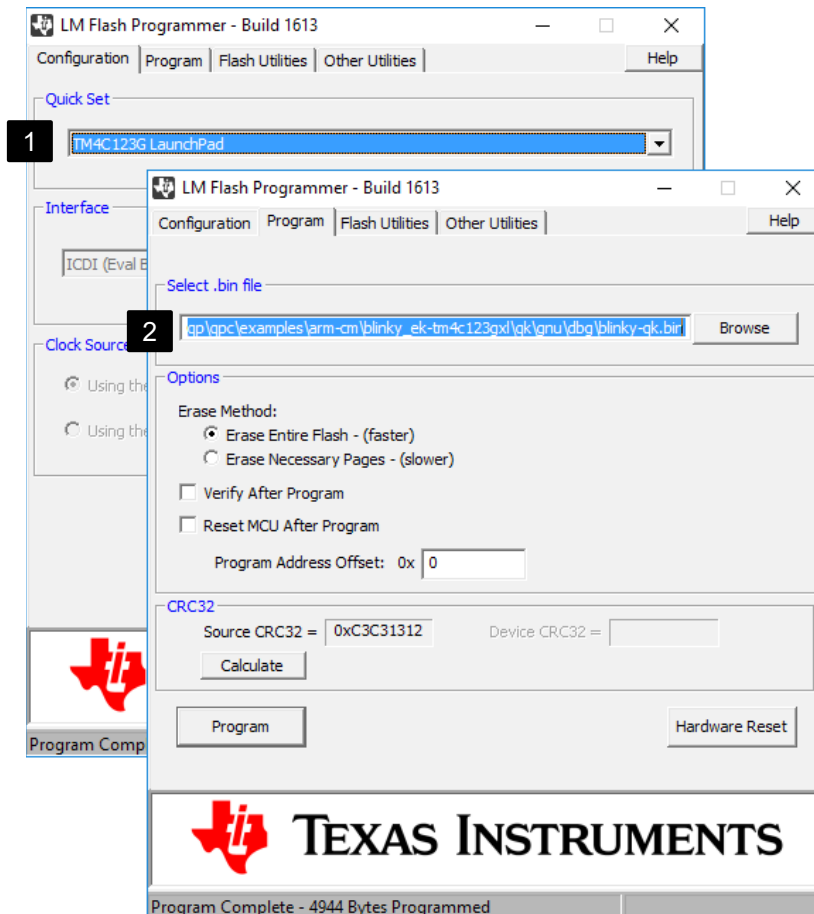
The provided Makefiles support the following build configurations:

Table 1 Make targets for the Debug, Release, and Spy build configurations

Build Configuration	Build/Clean command
Debug (default)	make / make clean
Release	make CONF=rel / make CONF=rel clean
Spy	make CONF=spy / make CONF=spy clean

Once you have successfully built the Blinky application (you can check for the file `blinky-qk.bin` in the `dbg` subdirectory) you can download it to the EK-TM4C123GXL board with the TI utility called `lmflash`.

Figure 7: Downloading the Blinky Application to the EK-TM4C123GXL Board with the LmFlash Utility

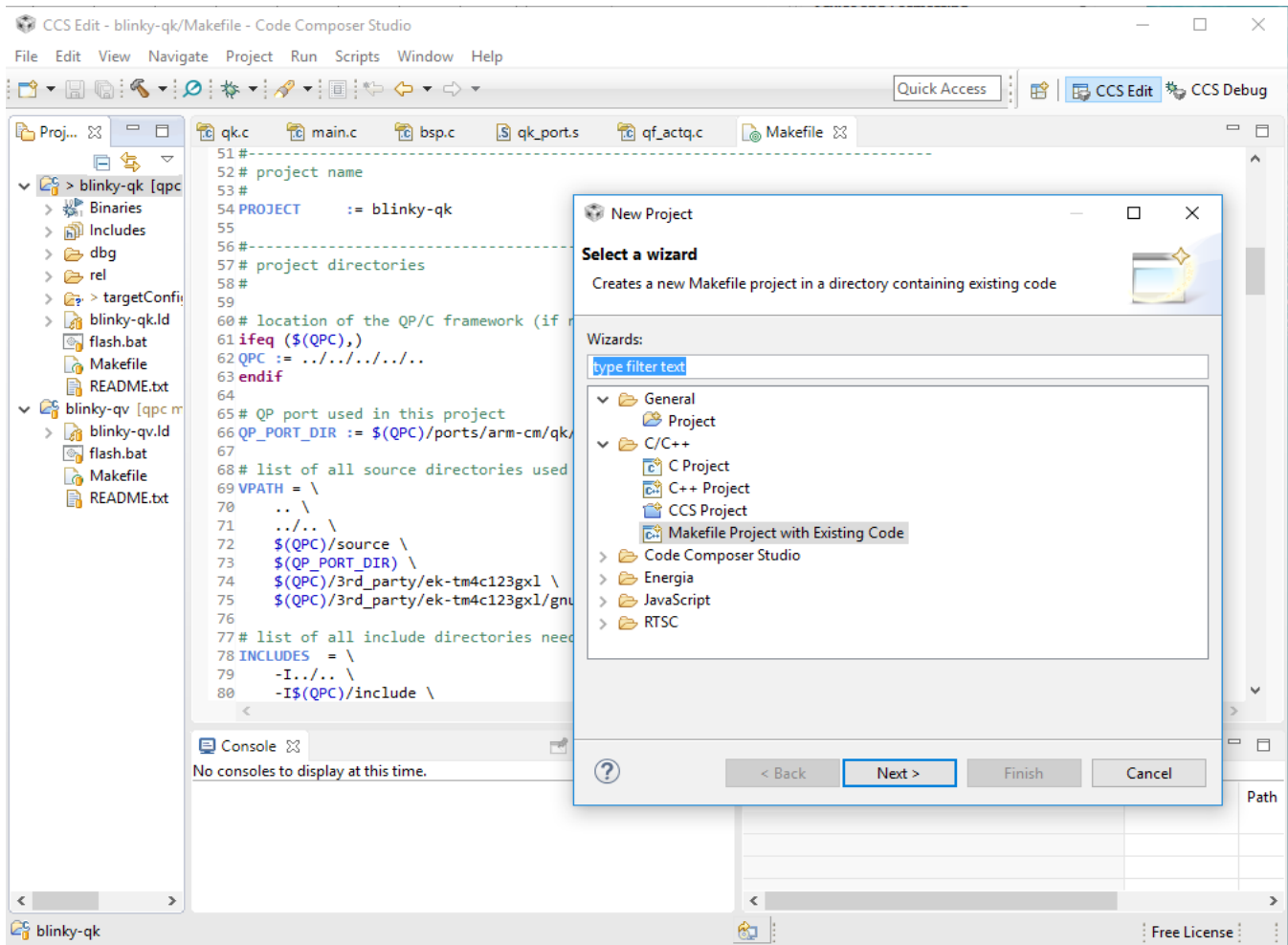


NOTE: You need to download the LmFlash utility from Texas Instruments (go to www.ti.com and search for "LmFlash")

- [1] In the Configuration tab, select the TM4C123GXL LaunchPad
- [2] In the Program tab, browse to the `blinky-qk.bin` file produced by the Makefile.

Finally, as mentioned before, you can import the Makefiles to an Eclipse-based IDE of your choice (the IDE should support the Stellaris-ICD debug interface of your TivaC LaunchPad board). The following screen shot shows the Makefile Project imported to the TI Code Composer Studio (CCS) IDE:

Figure 8: The Blinky Projects Imported into the TI CCS IDE



3.3 Blinky on Tiva LaunchPad with Keil/ARM (Keil uVision)

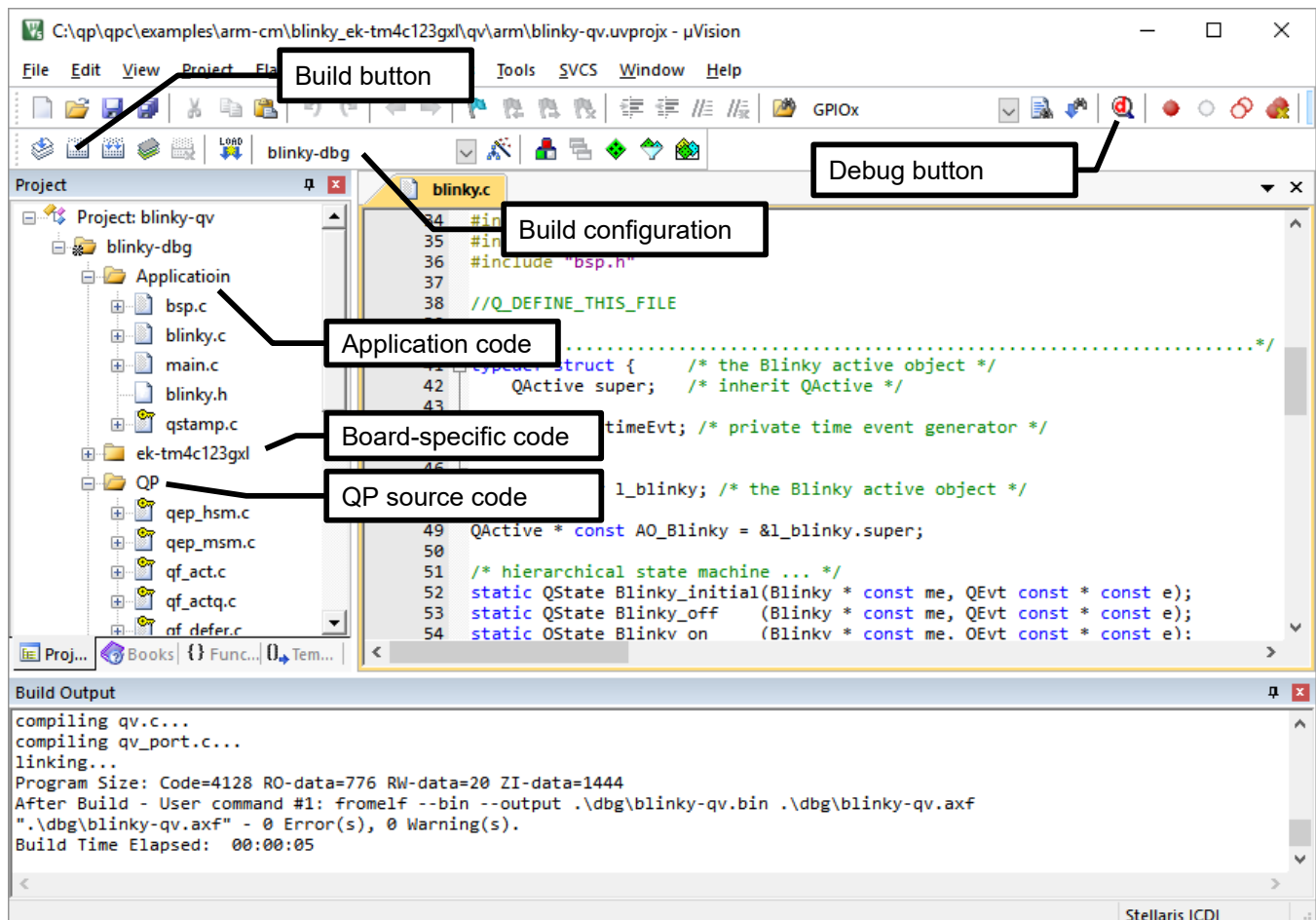
The Blinky example for the EK-TM4C123GXL board with ARM-KEIL uVision is located in the directories C:\qp\qpc\examples\arm-cm\blinky_ek-tm4c123gxl\qv\arm. (for the cooperative QV kernel) and C:\qp\qpc\examples\arm-cm\blinky_ek-tm4c123gxl\qk\arm (for the preemptive QK kernel). Each of these directories contains the uVision project file `blinky.uvproj`.



Keil/ARM MDK (<http://www.keil.com/arm/mdk.asp>) is an example of a commercial toolchain, which offers superior code generation, fast code download and good debugging experience.

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 9: Blinky workspace in Keil uVision5 IDE



To open the Blinky project in Keil uVision, 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 9).

3.4 Blinky on Tiva LaunchPad with IAR (IAR EWARM)

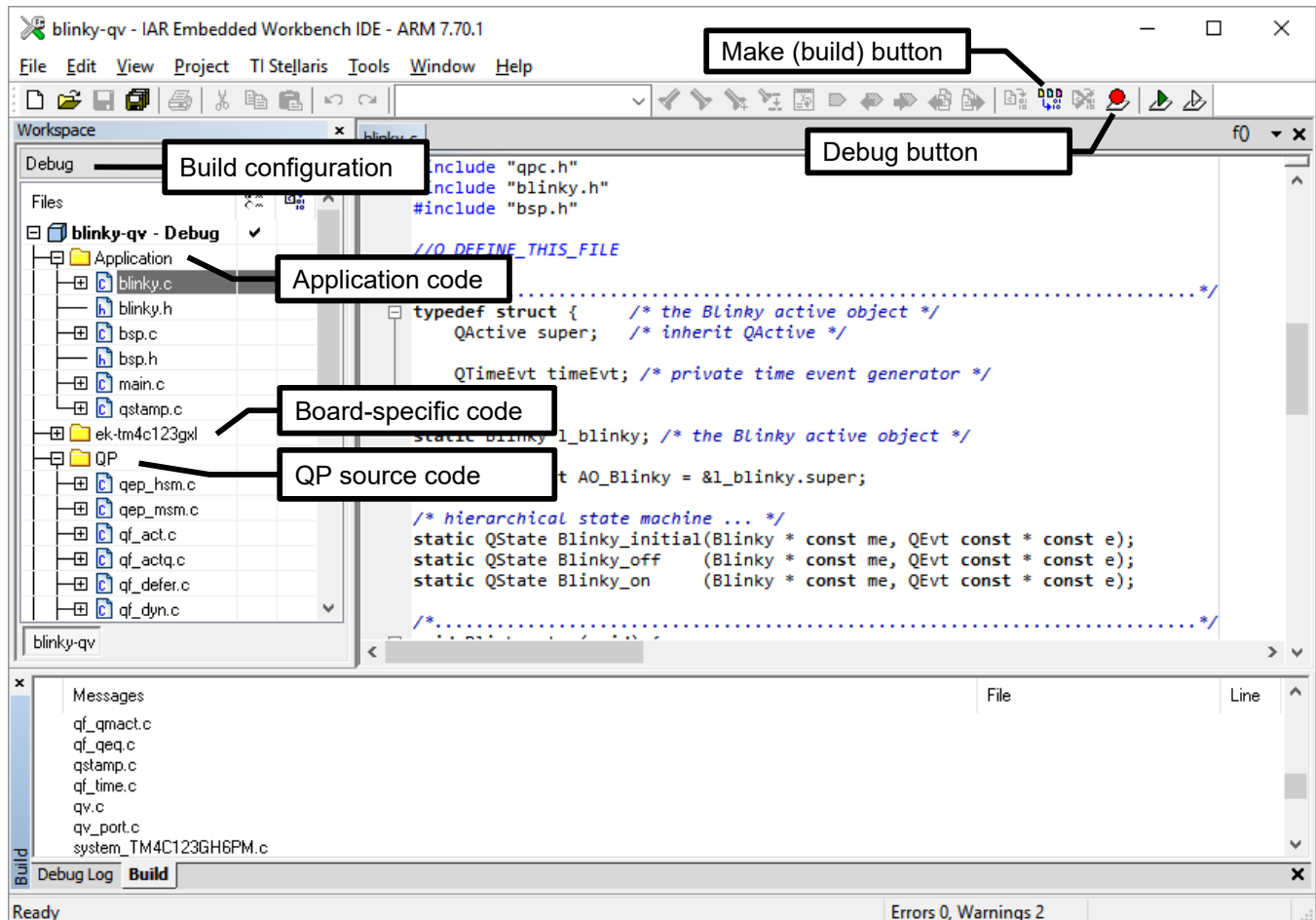
The Blinky example for the EK-TM4C123GXL board with IAR EWARM is located in the directories C:\qp\qpc\examples\arm-cm\blinky_ek-tm4c123gxl\qv\iar. (for the cooperative QV kernel) and C:\qp\qpc\examples\arm-cm\blinky_ek-tm4c123gxl\qk\iar (for the preemptive QK kernel). Each of these directories contains the IAR workspace file blinky.eww.



IAR EWARM is an example of commercial toolchain (<https://www.iar.com/iar-embedded-workbench/>), which offers superior code generation, fast code download and good debugging experience.

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 10: Blinky workspace in IAR EWARM

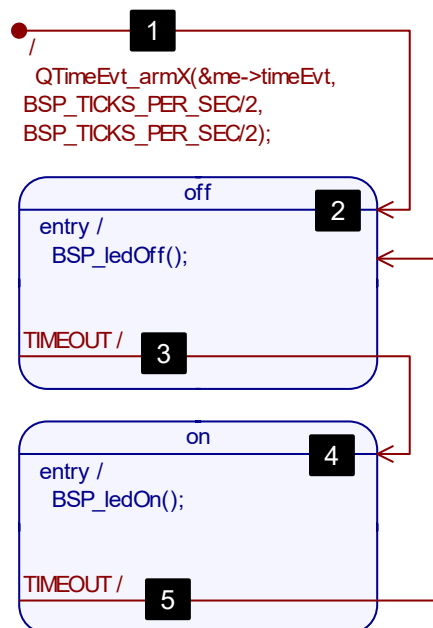


To open the Blinky workspace in EWARM, you can double-click on `blinky.eww` workspace file located in this directory. Once the project opens, you can build it by pressing the Make button. You can also very easily download it to the LaunchPad board and debug it by pressing the Debug button (see Figure 10).

4 The Blinky State Machine and Code

The behavior of the Blinky example is modeled by a very simple state machine (see Figure 11). 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 cases execution of the entry action, in which the LED is turned off. From that point on the cycle repeats forever.

Figure 11: Blinky state machine



- [1] The top-most initial transition in this state machine arms a QP time event (`QTimeEvt_armX()`) 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.
- [2] The initial transition leads to state "off", which turns the LED off in the entry action (`BSP_ledOff()`).
- [3] When the TIMEOUT event arrives in the "off" state, the "off" state transitions to the "on" state
- [4] The "on" state turns the LED on in the entry action (`BSP_ledOn()`).
- [5] When the TIMEOUT event arrives in the "on" state, the "on" state transitions back to "off", which cases execution of the entry action, in which the LED is turned off. From that point on the cycle repeats forever because the TIMEOUT events keep getting generated at the pre-determined rate.

The Blinky state machine shown in Figure 11 is implemented in the `blinky.c` source file, as shown in the following listing:

NOTE: The following code has been auto-generated by the [QM Model-Based Design Tool](#).



```
        case TIMEOUT_SIG: {
            status_ = Q_TRAN(&Blinky_on);
            break;
        }
        default: {
            status_ = Q_SUPER(&QHsm_top);
            break;
        }
    }
    return status_;
}

/*${Aos::Blinky::SM::on} .....*/
static QState Blinky_on(Blinky * const me, QEvt const * const e) {
    QState status_;
    switch (e->sig) {
        /*${Aos::Blinky::SM::on} */
        case Q_ENTRY_SIG: {
            BSP_ledOn();
            status_ = Q_HANDLED();
            break;
        }
        /*${Aos::Blinky::SM::on::TIMEOUT} */
        case TIMEOUT_SIG: {
            status_ = Q_TRAN(&Blinky_off);
            break;
        }
        default: {
            status_ = Q_SUPER(&QHsm_top);
            break;
        }
    }
    return status_;
}

/*$enddef${Aos::Blinky} ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^*/
```


5 Re-building the QP/C Libraries for Windows

On Windows, QP/C is deployed as a library that you statically link to your application. The pre-built QP libraries are provided inside the `C:\qp\qpc\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.

5.1 QP/C Library for Windows with MinGW

For the MinGW port, you perform a console build with the provided Makefile in `%QPC%\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 2 Make targets for the Debug, Release, and Spy build configurations

Build Configuration	Build/Clean command
Debug (default)	<code>make / make clean</code>
Release	<code>make CONF=rel / make CONF=rel clean</code>
Spy	<code>make CONF=spy / make CONF=spy clean</code>

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

Figure 12: Building QP/C library for Windows with MinGW

```

c:\qp>cd qpc\ports\win32
c:\qp\qpc\ports\win32>make
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_hsm.c -o dbg/qp_hsm.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_msm.c -o dbg/qp_msm.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_act.c -o dbg/qp_act.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_actq.c -o dbg/qp_actq.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_defer.c -o dbg/qp_defer.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_dyn.c -o dbg/qp_dyn.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_mem.c -o dbg/qp_mem.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_ps.c -o dbg/qp_ps.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_qact.c -o dbg/qp_qact.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_geq.c -o dbg/qp_geq.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_qmact.c -o dbg/qp_qmact.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c ../source/qp_time.c -o dbg/qp_time.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c win32_gui.c -o dbg/win32_gui.o
gcc -c -g -ffunction-sections -fdata-sections -O -I../include -I../source
-I.. -DWIN32_GUI -Wall -W -c qp_port.c -o dbg/qp_port.o
ar rs dbg/libqp.a dbg/qp_hsm.o dbg/qp_msm.o dbg/qp_act.o dbg/qp_actq.o dbg/qp_defer.o
dbg/qp_dyn.o dbg/qp_mem.o dbg/qp_ps.o dbg/qp_qact.o dbg/qp_geq.o dbg/qp_qmact.o
dbg/qp_time.o dbg/win32_gui.o dbg/qp_port.o
rm dbg/*.o
c:\qp\qpc\ports\win32>make CONF=rel

```

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 or "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. Below is a list of links to enable you to further your knowledge:

- Key Concepts behind QP frameworks and QM modeling tool (<https://www.state-machine.com/doc/concepts>)
- QP/C++ Reference Manual (<https://www.state-machine.com/qpcpp>)
- QM Reference Manual (<https://www.state-machine.com/qm>)
- QP Application Notes & Articles (<https://www.state-machine.com/doc/an>)
- Book "Practical UML Statecharts in C/C++, 2nd Edition" [PSiCC2] and the companion web-page to the book (<https://www.state-machine.com/psicc2>)
- Free Support Forum for QP/QM (<https://sourceforge.net/p/qpc/discussion/668726>)
- "State Space" Blog (<https://embeddedgurus.com/state-space>)

8 Contact Information

Quantum Leaps, LLC

info@state-machine.com
state-machine.com

+1 919 360-5668 (9AM-5PM US EST)
+1 919 869-2998 (FAX)

Quantum Leaps on GitHub



<https://github.com/QuantumLeaps>

