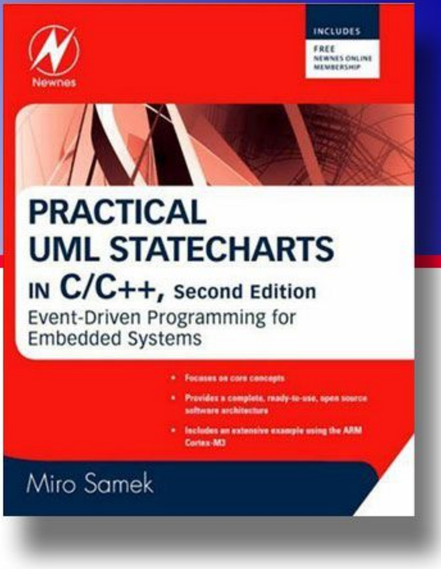
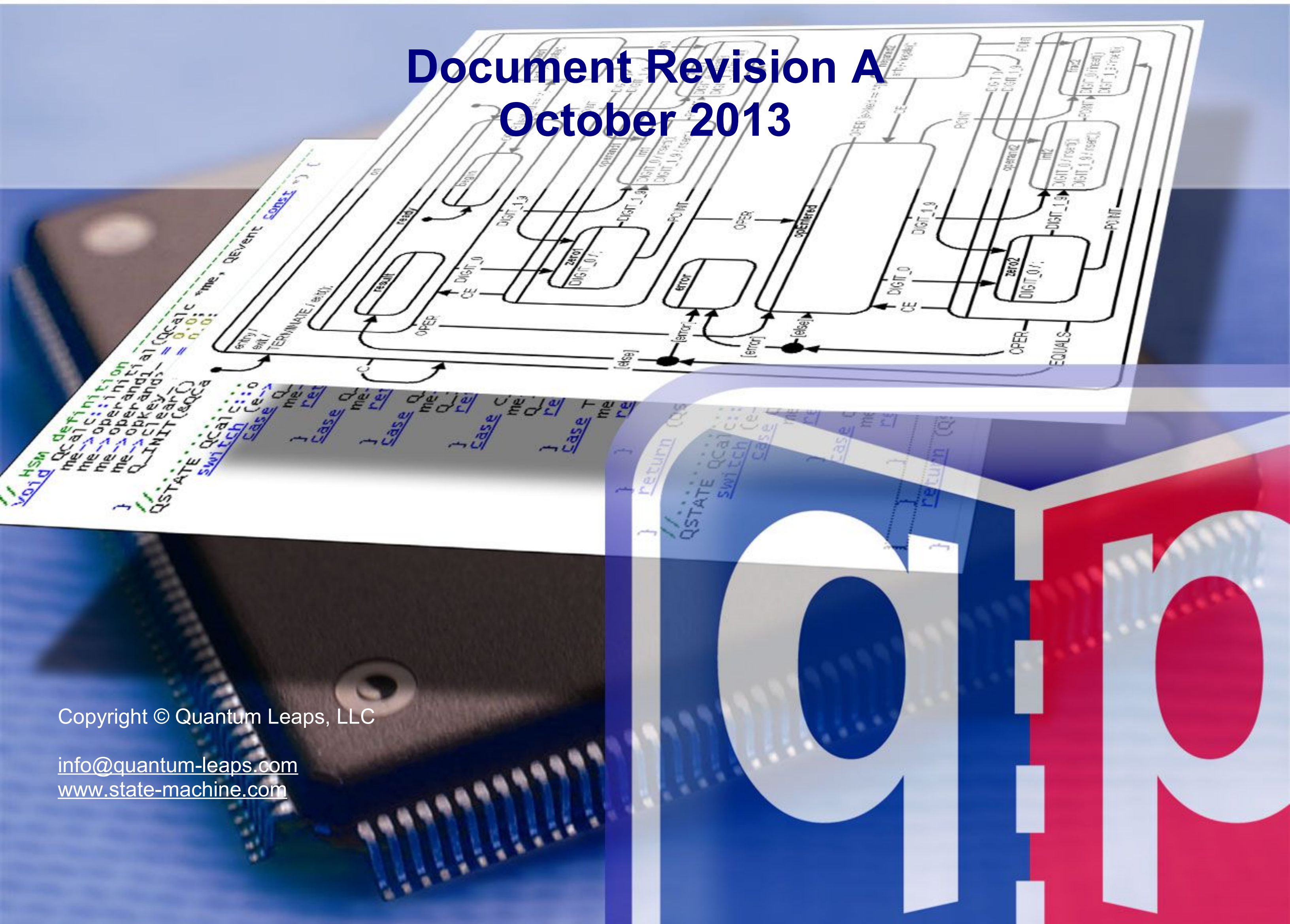


Getting Started with QP-nano™

Document Revision A
October 2013



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

Table of Contents

1 Introduction....... 1

2 Obtaining and Installing QP-nano and Qtools....... 2

- 2.1 Downloading QP-nano....... 2
- 2.2 Downloading Qtools....... 2
- 2.3 Installing QP-nano....... 3
- 2.4 Installing Qtools....... 4
- 2.5 Setting Up the QPN, 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 the eZ430-F2013 USB Stick with IAR....... 8
- 3.3 Blinky on Tiva LauchPad with GNU (Sourcery™ CodeBench)....... 9
- 3.4 Blinky on Tiva LauchPad with IAR (IAR EWARM)....... 12
- 3.5 Blinky on Tiva LauchPad with Keil/ARM (Keil uVision4)....... 13

4 The Blinky State Machine and Code....... 14

5 Creating your Own QP-nano Projects....... 16

6 Next Steps and Further Reading About QP™ and QM™....... 16

7 Contact Information....... 17

1 Introduction

This document explains how to install the QP-nano™ framework (version 5.1.1 or newer) and how to build and run a very simple “Blinky” QP-nano 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-nano active object framework as quickly as possible.

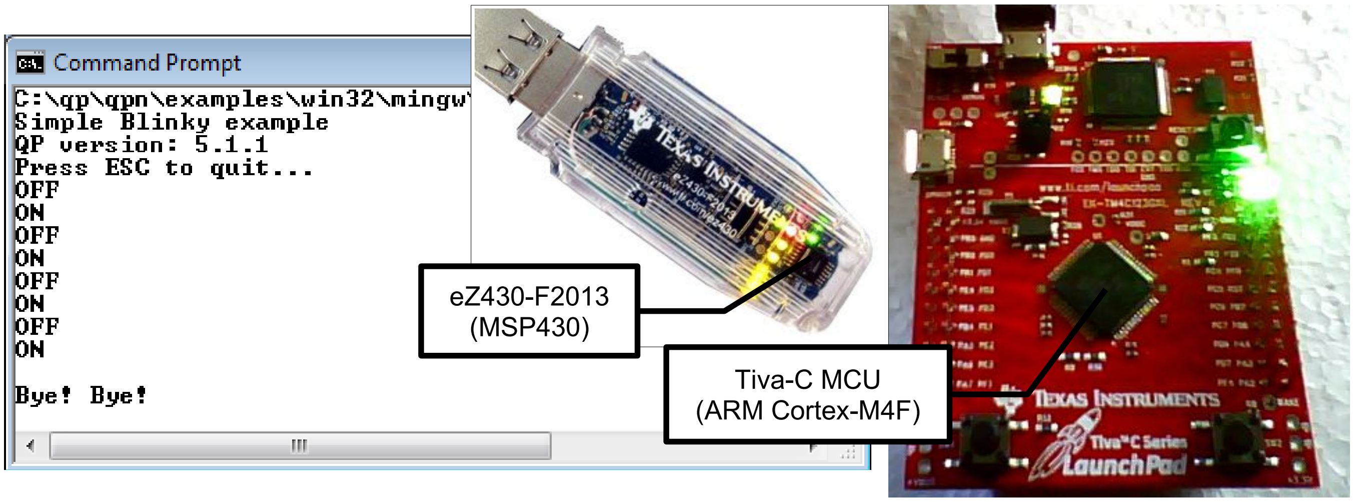
This document explains how to build and run the following versions of the “Blinky” example:

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

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

2. Version for the Texas Instruments eZ430-F2013 USB stick based on MSP430 MCU with IAR 430 Compiler (see Figure 1 middle); and
3. Versions for the Texas Instruments **Tiva™ C Series LaunchPad** board (EK-TM4C123GXL) based on the ARM Cortex-M4F core (see Figure 1 right) with the following embedded development toolsets:
   a) GNU toolset (Sourcery CodeBench)
   b) IAR EWARM toolset.
   c) Keil Microcontroller Development Kit (MDK).

Figure 1: Blinky on Windows (left), on eZ430-F2013 (middle), and on Tiva LaunchPad board (right)



```
C:\qp\qpn\examples\win32\mingw
Simple Blinky example
QP version: 5.1.1
Press ESC to quit...
OFF
ON
OFF
ON
OFF
ON
OFF
ON

Bye! Bye!
```

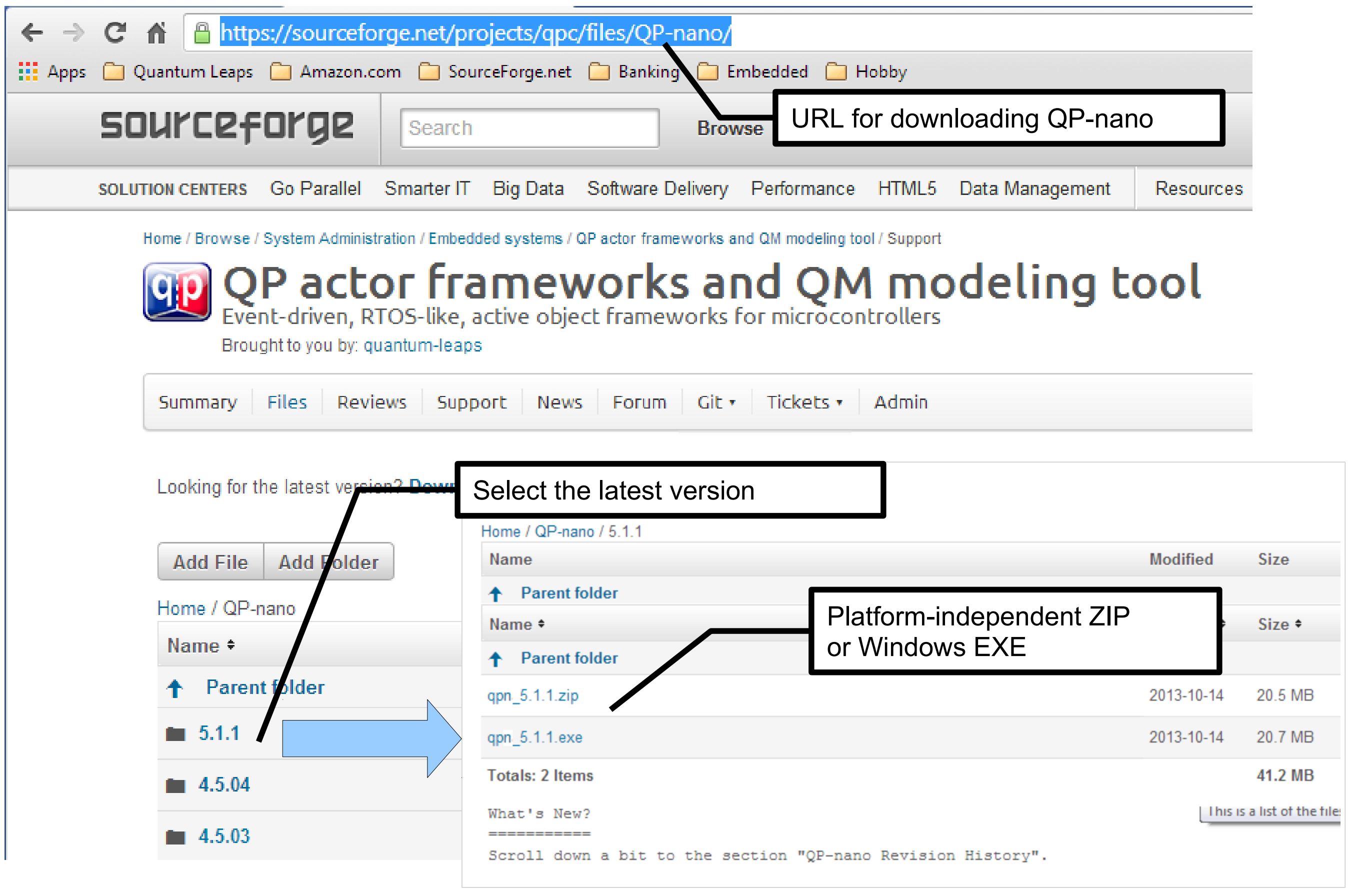
2 Obtaining and Installing QP-nano and Qtools

This section describes how to download and install the QP-nano framework and Qtools, the latter being a collection of various open source tools for working with QP-nano. The Qtools collection for Windows, contains the **GNU make** and the **GNU C/C++ compiler** (MinGW), which you can use to build the Blinky example.

2.1 Downloading QP-nano

The QP-nano framework is available for download from the SourceForge.net open source repository at the following URL http://sourceforge.net/projects/qpc/files/QP-nano/ . 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-nano framework from SourceForge.net


https://sourceforge.net/projects/qpc/files/QP-nano/
Apps
Quantum Leaps
Amazon.com
SourceForge.net
Banking
Embedded
Hobby
sourceforge
Search
Browse
URL for downloading QP-nano
SOLUTION CENTERS
Go Parallel
Smarter IT
Big Data
Software Delivery
Performance
HTML5
Data Management
Resources
Home / Browse / System Administration / Embedded systems / QP actor frameworks and QM modeling tool / Support
QP actor frameworks and QM modeling tool
Event-driven, RTOS-like, active object frameworks for microcontrollers
Brought to you by: quantum-leaps
Summary
Files
Reviews
Support
News
Forum
Git
Tickets
Admin
Looking for the latest version? Down
Select the latest version
Add File
Add Folder
Home / QP-nano
Name
Parent folder
5.1.1
4.5.04
4.5.03
Home / QP-nano / 5.1.1
Platform-independent ZIP or Windows EXE
This is a list of the file

Name	Modified	Size
Parent folder		
qpn_5.1.1.zip	2013-10-14	20.5 MB
qpn_5.1.1.exe	2013-10-14	20.7 MB
Totals: 2 Items		41.2 MB

```
What's New?
===========
Scroll down a bit to the section "QP-nano Revision History".
```

2.2 Downloading Qtools

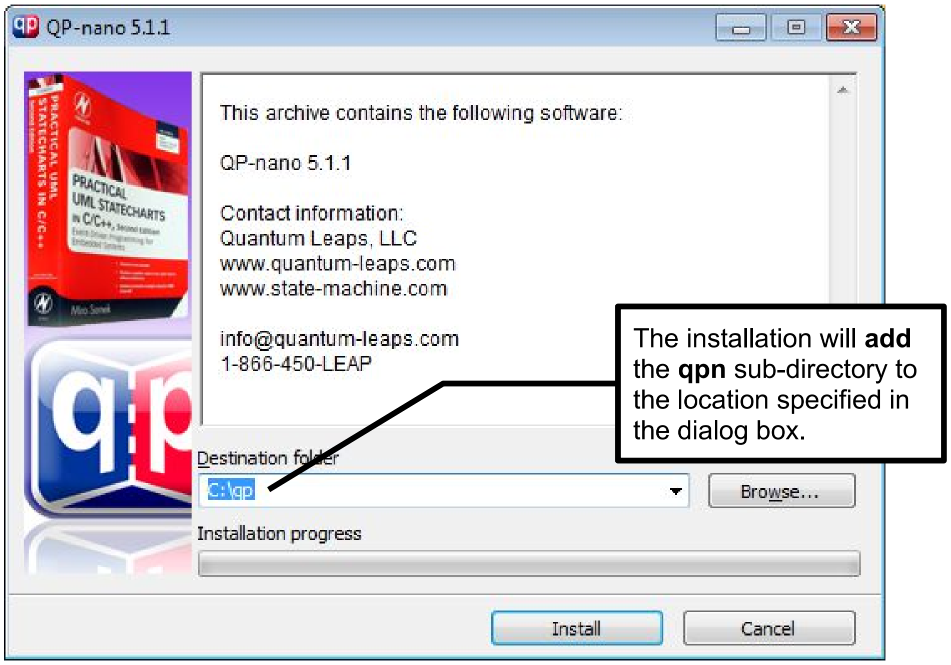
Once you are on SourceForge, you should also download the Qtools collection, which is available at the following URL http://sourceforge.net/projects/qpc/files/Qtools. From there, select the latest version and download the `qtools_win32_<ver>.exe` self-extracting Windows executable.

2.3 Installing QP-nano

The QP-nano 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-nano into the directory `C:\qp\qpn`.

NOTE: For your convenience of writing build scripts and make files, it is highly recommended to **avoid spaces** in the QP-nano 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-nano examples for 16-bit DOS, you need to extract QP-nano 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-nano from the Windows-EXE



QP-nano 5.1.1

This archive contains the following software:

QP-nano 5.1.1

Contact information:
Quantum Leaps, LLC
www.quantum-leaps.com
www.state-machine.com

info@quantum-leaps.com
1-866-450-LEAP

Destination folder: C:\qp

The installation will **add** the **qpn** sub-directory to the location specified in the dialog box.

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

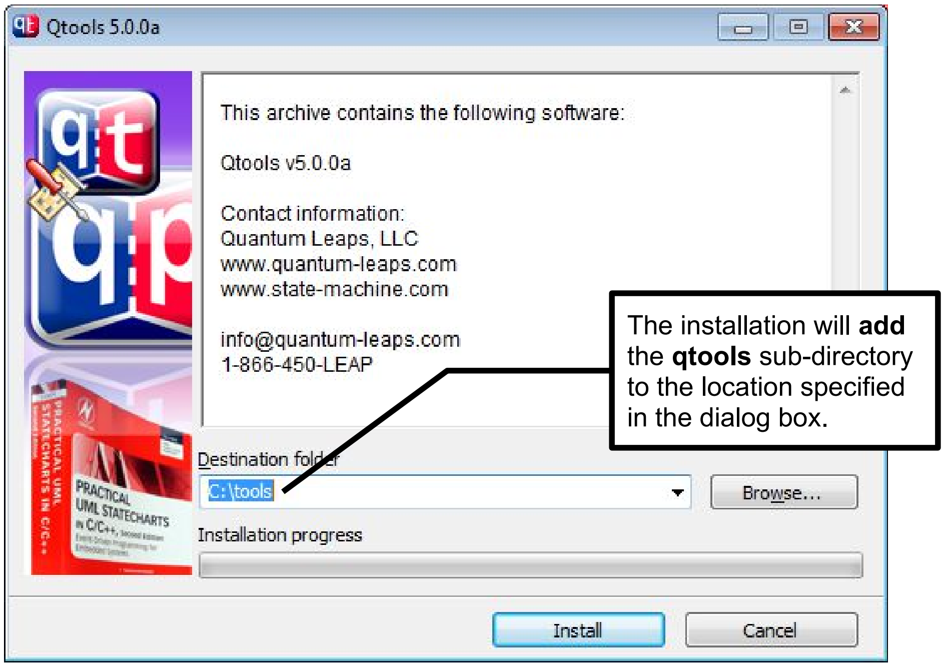
Listing 1: The main sub-directories of the QP-nano installation.

```
C:\
  +-qp\                 - QP directory
  | +-qpn\              - QP-nano directory
  | | +-doc\            - QP-nano documentation
  | | +-examples\       - QP-nano examples
  | | | +-arm-cm        - For ARM Cortex-M
  | | | +-msp430        - For MSP430
  | | | +-win32         - For Windows
  | | +-ports\          - QP-nano ports
  | | +-include\        - QP-nano platform-independent header files
  | | +-source\         - QP-nano platform-independent source files
```

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-nano framework. You simply run the 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 QPN, QTOOLS, and PATH Environment Variables

To complete the installation process you should define the `QPN` 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-nano framework. The following table shows the environment variables you need to define.

Environment variable	**Example setting (adjust to your system)**
QPN	`C:\qp\qpn`
QTOOLS	`C:\tools\qtools`
PATH	`…;%QTOOLS%`**`\bin`** (append to the PATH)

NOTE: The project files, make files, and build scripts provided in the QP-nano distribution assume that the environment variable **`QPN`** 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

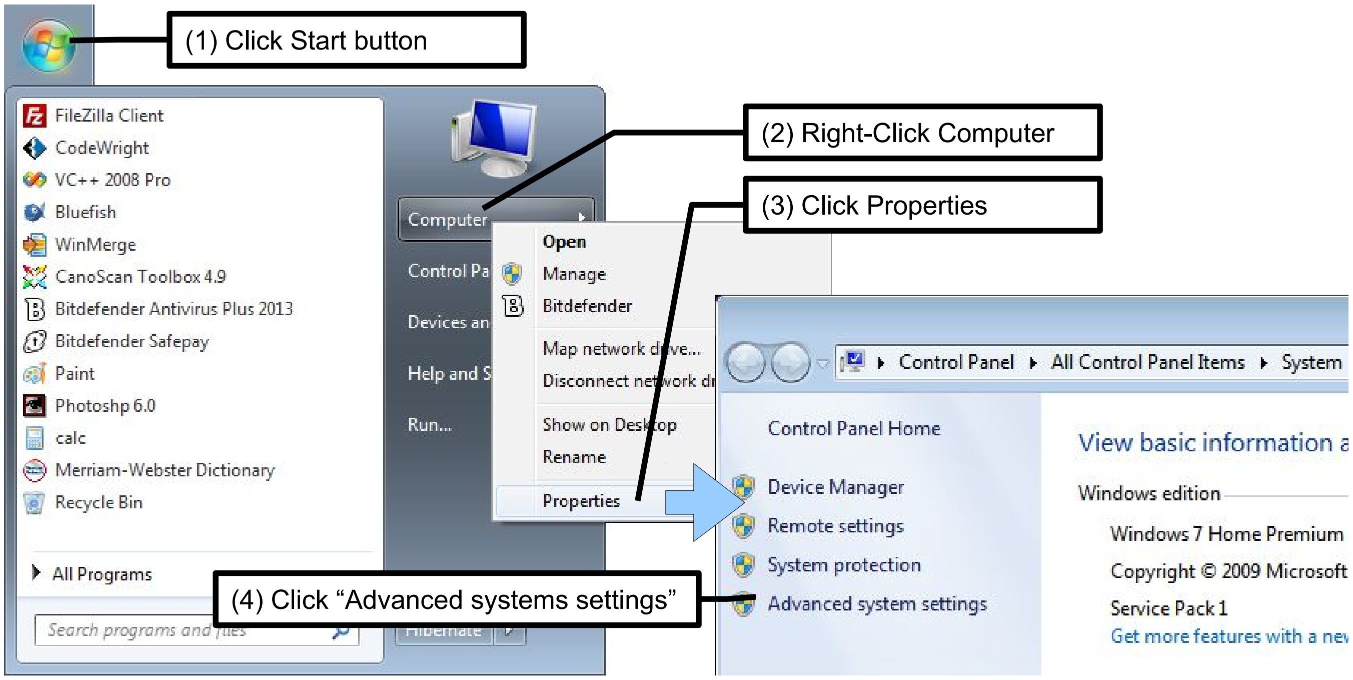
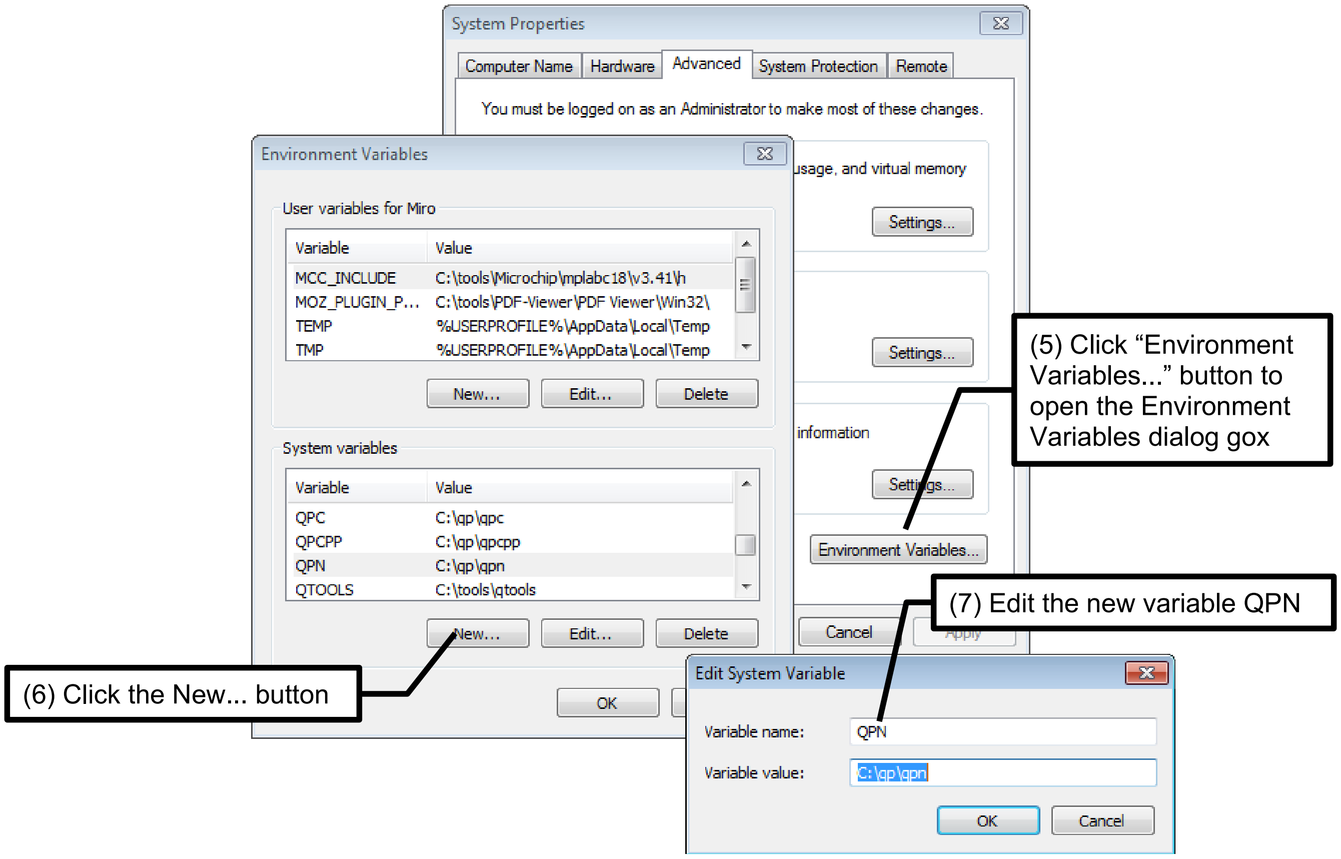


Figure 6: Adding/editing an environment variable in the "Environment Variables" dialog box



User variables for Miro

Variable	Value
MCC_INCLUDE	C:\tools\Microchip\mplabc18\v3.41\h
MOZ_PLUGIN_P...	C:\tools\PDF-Viewer\PDF Viewer\Win32\
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
QPC	C:\qp\qpc
QPCPP	C:\qp\qpcpp
QPN	C:\qp\qpn
QTOOLS	C:\tools\qtools

Variable name: QPN

Variable value: C:\qp\qpn

3 Building and Running the Blinky Example

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

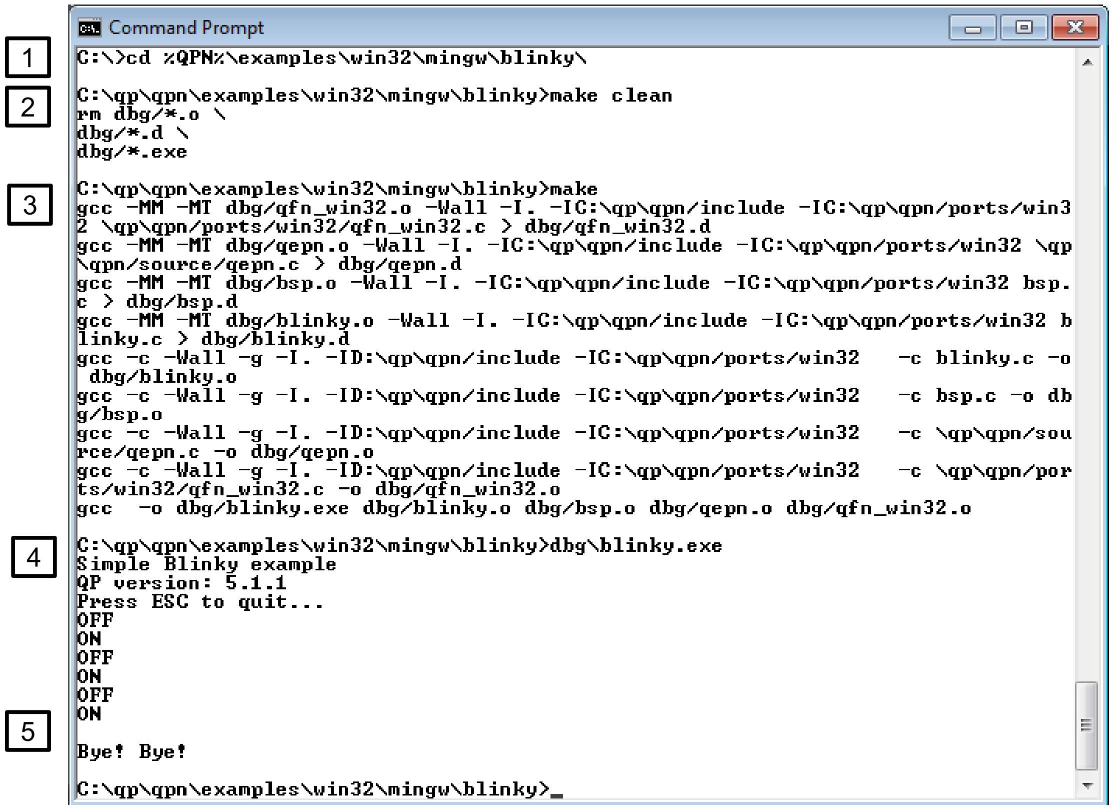
3.1 Blinky on Windows with MinGW (GNU C/C++ for Windows)

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 Qtools. The example is located in the `%QPN%\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-nano.

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.



```
C:\>cd %QPN%\examples\win32\mingw\blinky\

C:\qp\qpn\examples\win32\mingw\blinky>make clean
rm dbg/*.o \
dbg/*.d \
dbg/*.exe

C:\qp\qpn\examples\win32\mingw\blinky>make
gcc -MM -MT dbg/qfn_win32.o -Wall -I. -IC:\qp\qpn/include -IC:\qp\qpn/ports/win3
2 \qp\qpn/ports/win32/qfn_win32.c > dbg/qfn_win32.d
gcc -MM -MT dbg/qepn.o -Wall -I. -IC:\qp\qpn/include -IC:\qp\qpn/ports/win32 \qp
\qpn/source/qepn.c > dbg/qepn.d
gcc -MM -MT dbg/bsp.o -Wall -I. -IC:\qp\qpn/include -IC:\qp\qpn/ports/win32 bsp.
c > dbg/bsp.d
gcc -MM -MT dbg/blinky.o -Wall -I. -IC:\qp\qpn/include -IC:\qp\qpn/ports/win32 b
linky.c > dbg/blinky.d
gcc -c -Wall -g -I. -ID:\qp\qpn/include -IC:\qp\qpn/ports/win32   -c blinky.c -o
 dbg/blinky.o
gcc -c -Wall -g -I. -ID:\qp\qpn/include -IC:\qp\qpn/ports/win32   -c bsp.c -o db
g/bsp.o
gcc -c -Wall -g -I. -ID:\qp\qpn/include -IC:\qp\qpn/ports/win32   -c \qp\qpn/sou
rce/qepn.c -o dbg/qepn.o
gcc -c -Wall -g -I. -ID:\qp\qpn/include -IC:\qp\qpn/ports/win32   -c \qp\qpn/por
ts/win32/qfn_win32.c -o dbg/qfn_win32.o
gcc  -o dbg/blinky.exe dbg/blinky.o dbg/bsp.o dbg/qepn.o dbg/qfn_win32.o

C:\qp\qpn\examples\win32\mingw\blinky>dbg\blinky.exe
Simple Blinky example
QP version: 5.1.1
Press ESC to quit...
OFF
ON
OFF
ON
OFF
ON

Bye! Bye!

C:\qp\qpn\examples\win32\mingw\blinky>_
```

(1) Change directory to the Blinky example for Windows with the MinGW compiler. Please note that the command “`cd %QPN%\examples\win32\mingw\blinky`” tests the definition of the `QPN` environment variable. (NOTE: you can also verify the definition by typing “`echo %QPN%`”)

(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).

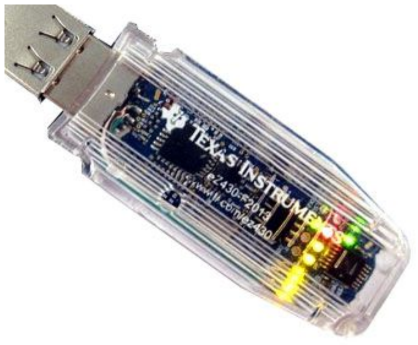
NOTE: The Blinky application is built from the **QP-nano source code and QP-nano port for Windows**. The location of these elements is determined by the `QPN` environment variable.

---

(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**.

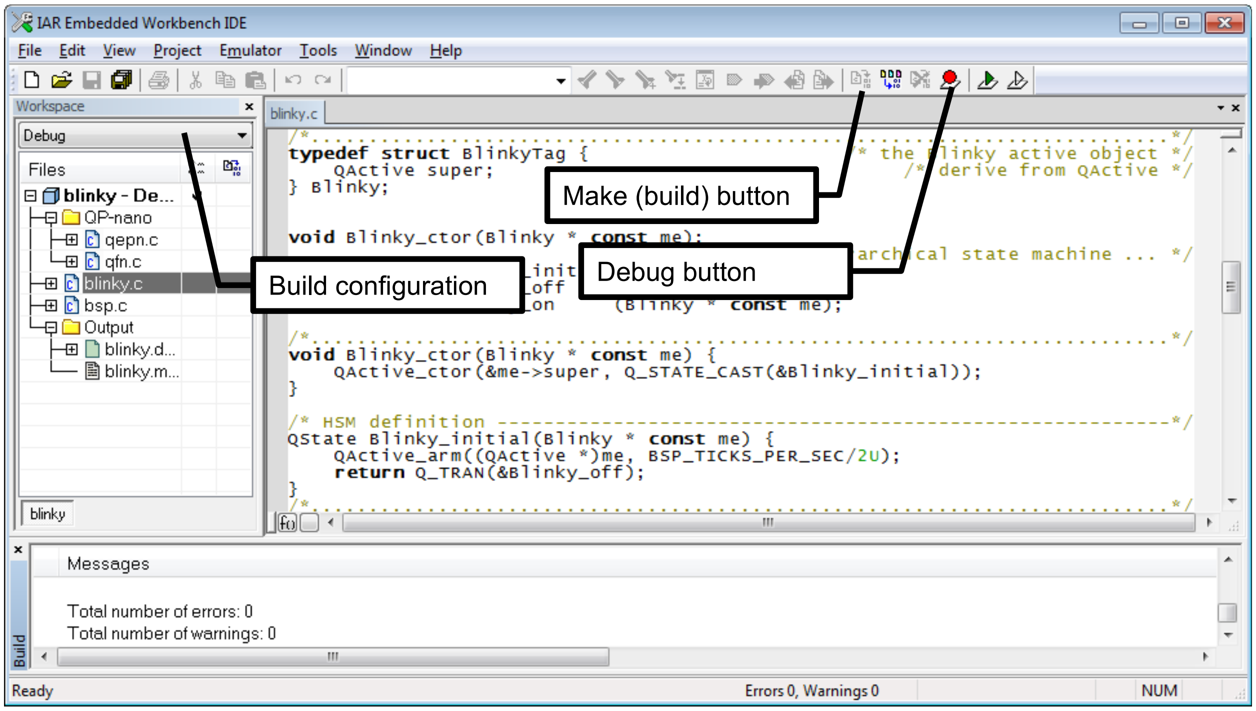
3.2 Blinky on the eZ430-F2013 USB Stick with IAR



The eZ430-F2013 is a nice little with all the hardware to evaluate the MSP430 microcontroller and develop a complete project in a convenient USB stick form factor. The MSP430F2013 on board has only 2KB of flash ROM for program and 128 bytes (1) of RAM for data and stack. Still, it is enough for non-trivial QP-nano applications, such as PEdestrian LIght CONtrolled (PELICAN) crossing controller (included in the standard QP-nano code). Of course, MSP430F2013 is more than capable to run the simple Blinky code, which has been compiled with the IAR EW430 toolset.

NOTE: IAR offers a **free** size-limited KickStart version of EW430 as well as time-limited evaluation options. The Blinky example described here has been built with the free KickStart EWARM edition limited to 4KB of generated code (which is twice as much as on-board flash ROM of MSP430F2013).

Figure 8: Blinky workspace in IAR EW430



```
/*..........................................................................*/
typedef struct BlinkyTag {                           /* the Blinky active object */
    QActive super;                                     /* derive from QActive */
} Blinky;

void Blinky_ctor(Blinky * const me);
                                             archical state machine ... */
                 init
                 off
                 on          (Blinky * const me);

/*..........................................................................*/
void Blinky_ctor(Blinky * const me) {
    QActive_ctor(&me->super, Q_STATE_CAST(&Blinky_initial));
}

/* HSM definition ----------------------------------------------------------*/
QState Blinky_initial(Blinky * const me) {
    QActive_arm((QActive *)me, BSP_TICKS_PER_SEC/2U);
    return Q_TRAN(&Blinky_off);
}
/*..........................................................................*/
```

```
Total number of errors: 0
Total number of warnings: 0
```

The Blinky example for IAR EW430 is located in the directory `%QPN%\examples\arm-cm\iar\blinky_eZ430\`. To open the Blinky project in IAR EW430, 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 eZ430 USB stick and debug it by pressing the Debug button (see Figure 8).

NOTE: The Blinky application uses to the **QP-nano source code** (files qepn.c and qfn.c).

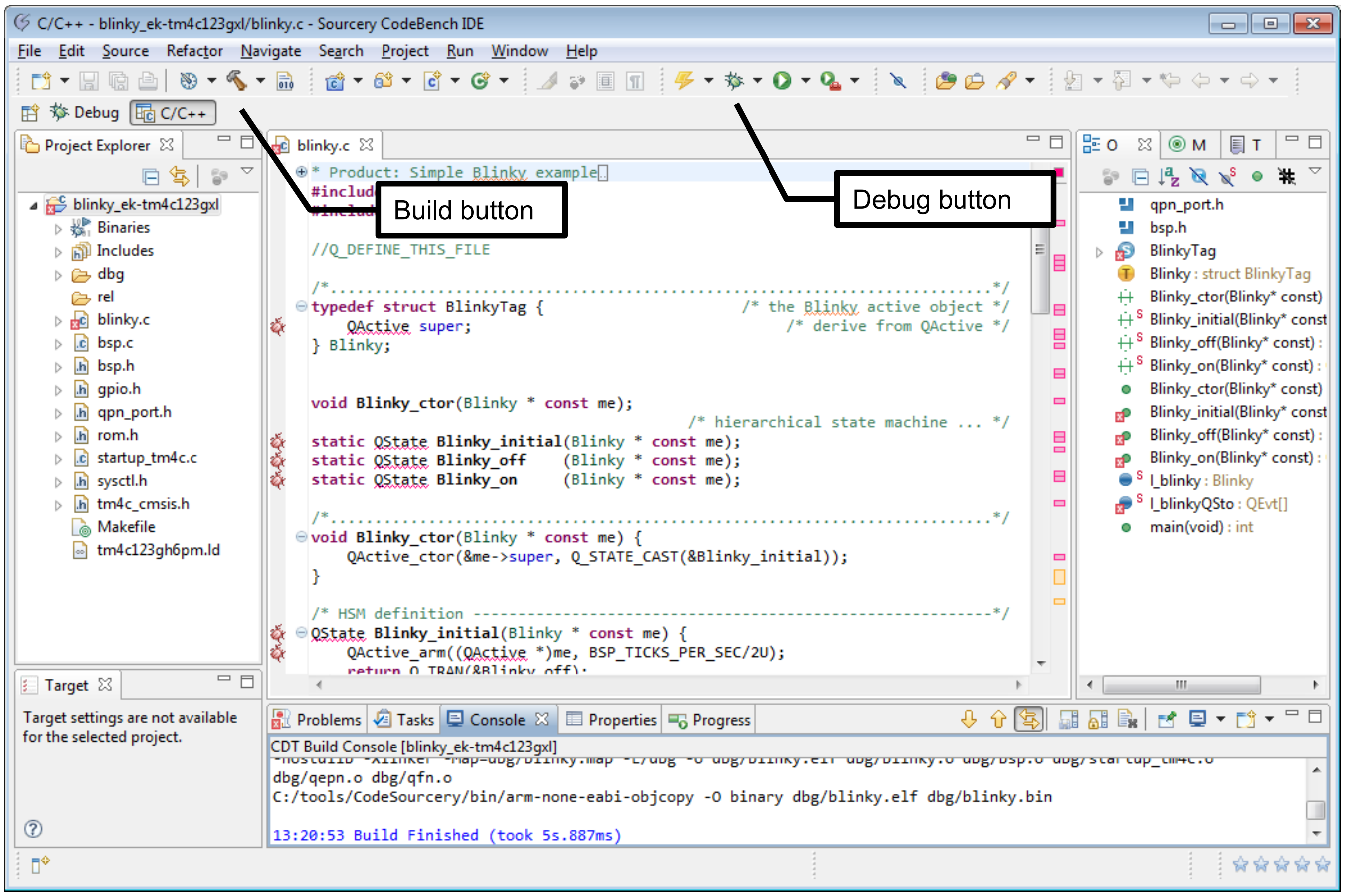
3.3 Blinky on Tiva LauchPad with GNU (Sourcery™ CodeBench)

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 9: Building and running Blinky in Sourcery CodeBench IDE


C/C++ - blinky_ek-tm4c123gxl/blinky.c - Sourcery CodeBench IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
C/C++
Project Explorer
blinky_ek-tm4c123gxl
Binaries
Includes
dbg
rel
blinky.c
bsp.c
bsp.h
gpio.h
qpn_port.h
rom.h
startup_tm4c.c
sysctl.h
tm4c_cmsis.h
Makefile
tm4c123gh6pm.ld
blinky.c
Build button
Debug button
qpn_port.h
bsp.h
BlinkyTag
Blinky : struct BlinkyTag
Blinky_ctor(Blinky* const)
Blinky_initial(Blinky* const
Blinky_off(Blinky* const) :
Blinky_on(Blinky* const) :
Blinky_ctor(Blinky* const)
Blinky_initial(Blinky* const
Blinky_off(Blinky* const) :
Blinky_on(Blinky* const) :
l_blinky : Blinky
l_blinkyQSto : QEvt[]
main(void) : int
Target
Target settings are not available for the selected project.
Problems
Tasks
Console
Properties
Progress
CDT Build Console [blinky_ek-tm4c123gxl]

```
* Product: Simple Blinky example
#includ
#includ

//Q_DEFINE_THIS_FILE

/*..........................................................................*/
typedef struct BlinkyTag {                         /* the Blinky active object */
    QActive super;                                      /* derive from QActive */
} Blinky;


void Blinky_ctor(Blinky * const me);
                                          /* hierarchical state machine ... */
static QState Blinky_initial(Blinky * const me);
static QState Blinky_off    (Blinky * const me);
static QState Blinky_on     (Blinky * const me);

/*..........................................................................*/
void Blinky_ctor(Blinky * const me) {
    QActive_ctor(&me->super, Q_STATE_CAST(&Blinky_initial));
}

/* HSM definition ----------------------------------------------------------*/
QState Blinky_initial(Blinky * const me) {
    QActive_arm((QActive *)me, BSP_TICKS_PER_SEC/2U);
    return Q_TRAN(&Blinky_off);
```

```
-nostdlib -Xlinker -Map=dbg/blinky.map -L/dbg -o dbg/blinky.elf dbg/blinky.o dbg/bsp.o dbg/startup_tm4c.o
dbg/qepn.o dbg/qfn.o
C:/tools/CodeSourcery/bin/arm-none-eabi-objcopy -O binary dbg/blinky.elf dbg/blinky.bin

13:20:53 Build Finished (took 5s.887ms)
```

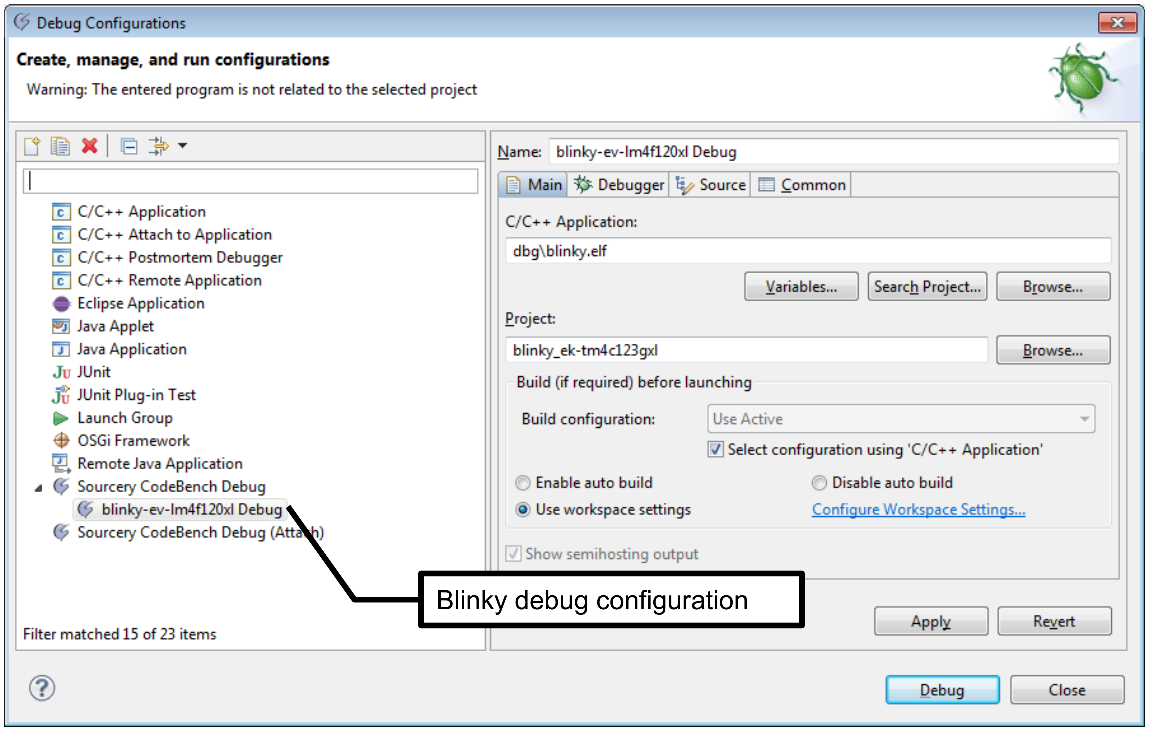
NOTE: The Blinky application uses to the **QP-nano source code for GNU-ARM.**

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 **%QPN%**

`examples\arm-cm\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 9).

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 10).

Figure 10: Editing the Debug Configurations in Sourcery CodeBench IDE


Debug Configurations

Create, manage, and run configurations

Warning: The entered program is not related to the selected project

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- Eclipse Application
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Launch Group
- OSGi Framework
- Remote Java Application
- Sourcery CodeBench Debug
  - blinky-ev-lm4f120xl Debug
- Sourcery CodeBench Debug (Attach)

Filter matched 15 of 23 items

Name: blinky-ev-lm4f120xl Debug

Main | Debugger | Source | Common

C/C++ Application: dbg\blinky.elf

Project: blinky_ek-tm4c123gxl

Build (if required) before launching

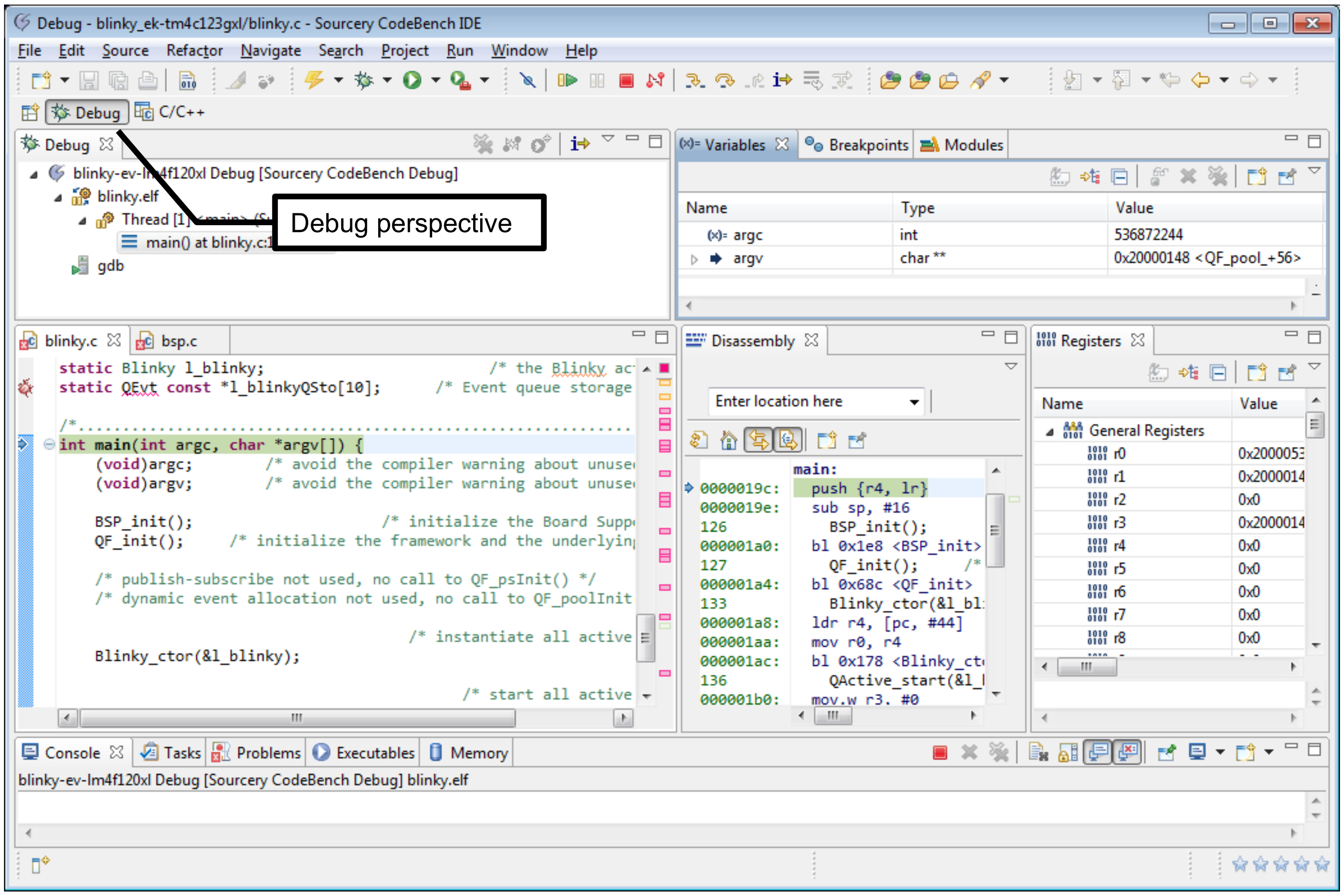
Build configuration: Use Active

[✓] Select configuration using 'C/C++ Application'

Enable auto build ( ) Disable auto build ( ) [● Use workspace settings] Configure Workspace Settings...

[✓] Show semihosting output

Figure 11: Debugging Blinky in Sourcery CodeBench IDE



Name	Type	Value
argc	int	536872244
argv	char **	0x20000148 <QF_pool_+56>

```
static Blinky l_blinky;                      /* the Blinky ac
static QEvt const *l_blinkyQSto[10];       /* Event queue storage

/*.............................................................
int main(int argc, char *argv[]) {
    (void)argc;         /* avoid the compiler warning about unuse
    (void)argv;         /* avoid the compiler warning about unuse

    BSP_init();                          /* initialize the Board Supp
    QF_init();     /* initialize the framework and the underlyin

    /* publish-subscribe not used, no call to QF_psInit() */
    /* dynamic event allocation not used, no call to QF_poolInit

                                       /* instantiate all active
    Blinky_ctor(&l_blinky);

                                             /* start all active
```

```
                main:
0000019c:         push {r4, lr}
0000019e:         sub sp, #16
126                 BSP_init();
000001a0:         bl 0x1e8 <BSP_init>
127                 QF_init();      /*
000001a4:         bl 0x68c <QF_init>
133                 Blinky_ctor(&l_bl
000001a8:         ldr r4, [pc, #44]
000001aa:         mov r0, r4
000001ac:         bl 0x178 <Blinky_ct
136                 QActive_start(&l_
000001b0:         mov.w r3, #0
```

Name	Value
General Registers	
r0	0x2000053
r1	0x2000014
r2	0x0
r3	0x2000014
r4	0x0
r5	0x0
r6	0x0
r7	0x0
r8	0x0

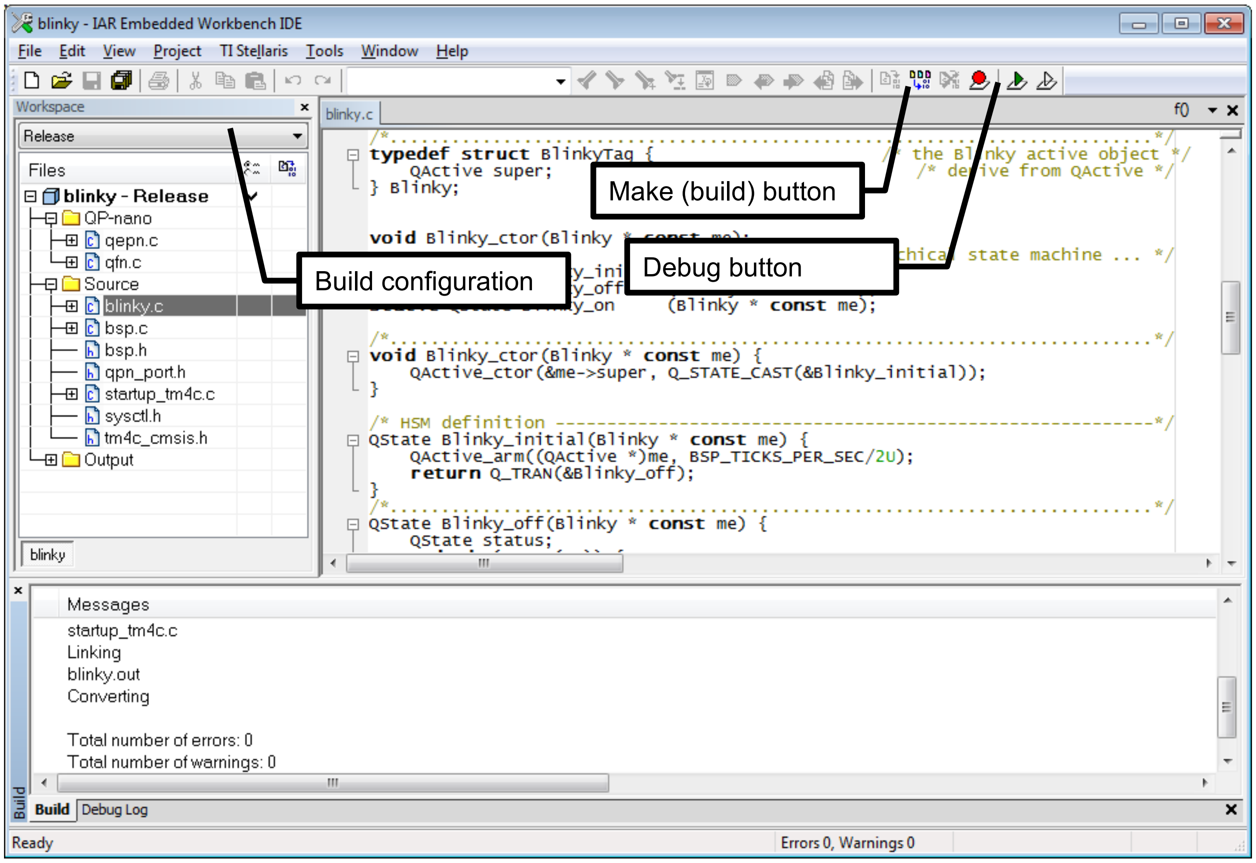
```
blinky-ev-lm4f120xl Debug [Sourcery CodeBench Debug] blinky.elf
```

3.4 Blinky on Tiva LauchPad 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 12: Blinky workspace in IAR EWARM



```
/*..........................................................................*/
typedef struct BlinkyTag {                          /* the Blinky active object */
    QActive super;                                    /* derive from QActive */
} Blinky;

void Blinky_ctor(Blinky * const me);
                                                  ...chical state machine ... */
                 y_ini
                 y_off
                 y_on       (Blinky * const me);

/*..........................................................................*/
void Blinky_ctor(Blinky * const me) {
    QActive_ctor(&me->super, Q_STATE_CAST(&Blinky_initial));
}

/* HSM definition ---------------------------------------------------------*/
QState Blinky_initial(Blinky * const me) {
    QActive_arm((QActive *)me, BSP_TICKS_PER_SEC/2U);
    return Q_TRAN(&Blinky_off);
}
/*..........................................................................*/
QState Blinky_off(Blinky * const me) {
    QState status;
```

```
Messages
startup_tm4c.c
Linking
blinky.out
Converting

Total number of errors: 0
Total number of warnings: 0
```

The Blinky example for IAR EWARM is located in the directory `%QPN%\examples\arm-cm\iar\blinky_ek-tm4c123gxl\`. 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 12).

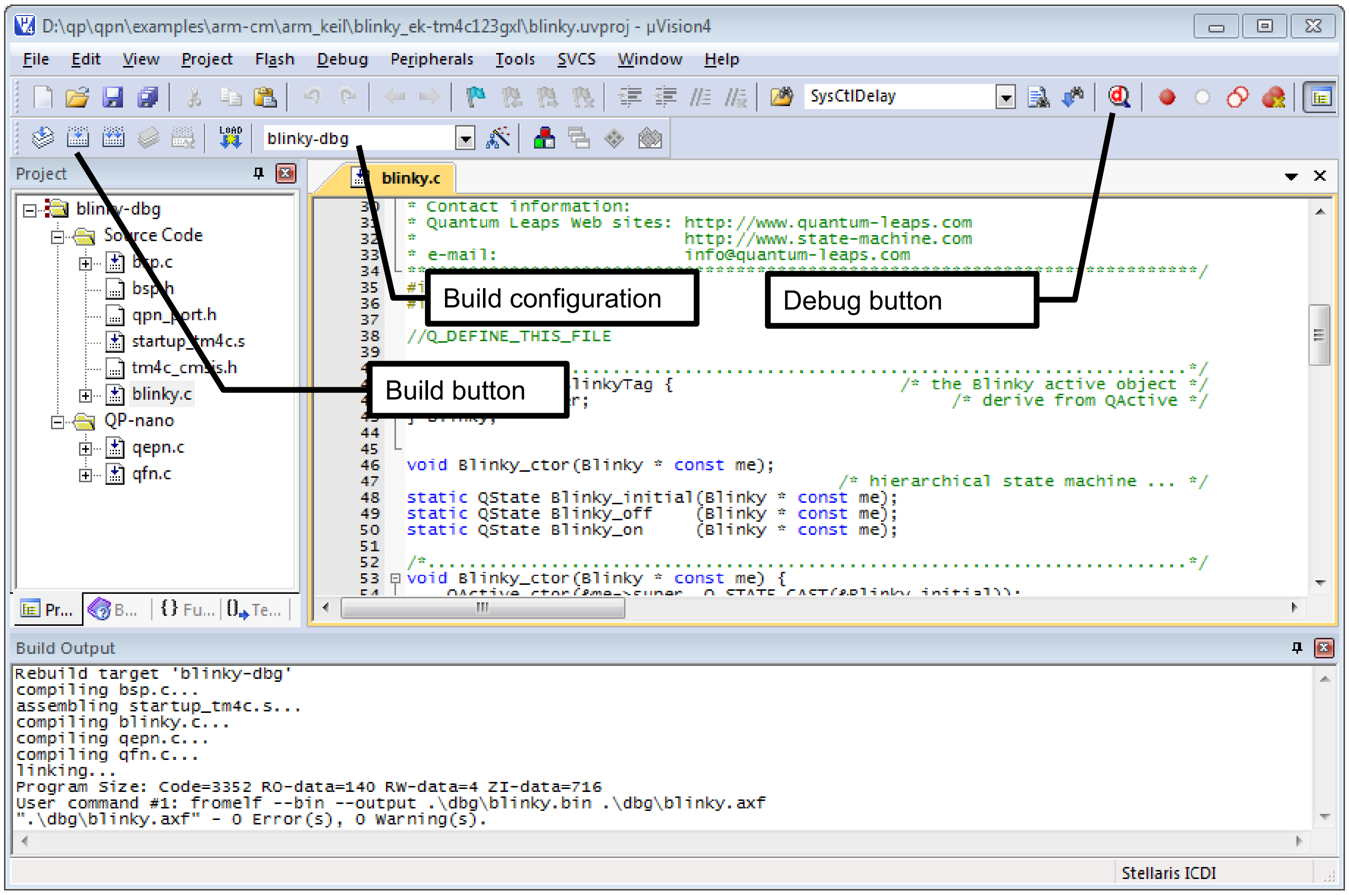
NOTE: The Blinky application uses to the **QP-nano source code for IAR EWARM**.

3.5 Blinky on Tiva LauchPad 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: Keail/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 13: Blinky workspace in Keil uVision4 IDE



```
 * Contact information:
 * Quantum Leaps Web sites: http://www.quantum-leaps.com
 *                          http://www.state-machine.com
 * e-mail:                  info@quantum-leaps.com
 *****************************************************************************/
#i
#i

//Q_DEFINE_THIS_FILE

/*..........................................................................*/
            linkyTag {                                /* the Blinky active object */
            r;                                           /* derive from QActive */
} Blinky;


void Blinky_ctor(Blinky * const me);
                                           /* hierarchical state machine ... */
static QState Blinky_initial(Blinky * const me);
static QState Blinky_off    (Blinky * const me);
static QState Blinky_on     (Blinky * const me);

/*..........................................................................*/
void Blinky_ctor(Blinky * const me) {
    QActive_ctor(&me->super, Q_STATE_CAST(&Blinky_initial));
```

```
Rebuild target 'blinky-dbg'
compiling bsp.c...
assembling startup_tm4c.s...
compiling blinky.c...
compiling qepn.c...
compiling qfn.c...
linking...
Program Size: Code=3352 RO-data=140 RW-data=4 ZI-data=716
User command #1: fromelf --bin --output .\dbg\blinky.bin .\dbg\blinky.axf
".\dbg\blinky.axf" - 0 Error(s), 0 Warning(s).
```

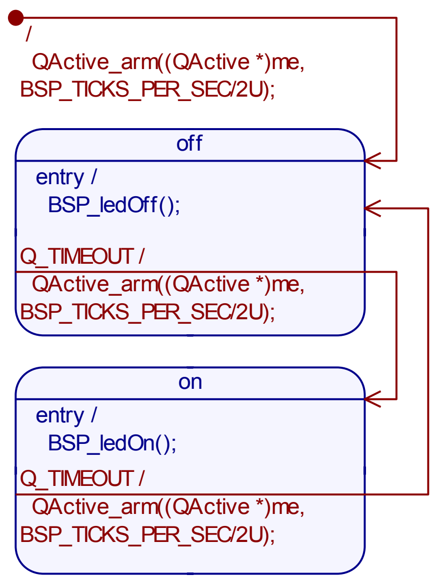
The Blinky example for Keil/ARM is located in the directory `%QPN%\examples\arm-cm\arm_keil\blinky_ek-tm4c123gxl\`. To open the Blinky project in Keil uVision4, you can double-click on `blinky.uvproj` project file located in this directory. Once the project opens, you can build it by pressing the Build button. You can also very easily download it to the LaunchPad board and debug it by pressing the Debug button (see Figure 13).

NOTE: The Blinky application links to the **QP-nano library for Keil/ARM,** which is pre-compiled and provided in the standard QP-nano distribution. The upcoming Section Error: Reference source not found describes how you can re-compile the QP-nano library yourself.

4 The Blinky State Machine and Code

The behavior of the Blinky example is modeled by a very simple state machine (see Error: Reference source not found). The top-most initial transition in this state machine arms a QP-nano time event to deliver the Q_TIMEOUT signal in half a second. Subsequently, every time the Q_TIMEOUT signal arrives, the time event is armed again for half a second, so that the LED can stay on for a 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 Q_TIMEOUT event arrives, the “off” state transitions to the “on” state, which turns the LED on in the entry action. When the Q_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 14: Blinky state machine



The Blinky state machine shown in Error: Reference source not found is implemented in the `blinky.c` source file, as shown in the following listing:

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

```
typedef struct BlinkyTag {                              /* the Blinky active object */
    QActive super;                                          /* derive from QActive */
} Blinky;


void Blinky_ctor(Blinky * const me);
                                                  /* hierarchical state machine ... */
static QState Blinky_initial(Blinky * const me);
static QState Blinky_off    (Blinky * const me);
static QState Blinky_on     (Blinky * const me);

/*..........................................................................*/
void Blinky_ctor(Blinky * const me) {
    QActive_ctor(&me->super, Q_STATE_CAST(&Blinky_initial));
}

/* HSM definition -----------------------------------------------------------*/
QState Blinky_initial(Blinky * const me) {
    QActive_arm((QActive *)me, BSP_TICKS_PER_SEC/2U);
    return Q_TRAN(&Blinky_off);
}
/*..........................................................................*/
QState Blinky_off(Blinky * const me) {
    QState status;
    switch (Q_SIG(me)) {
        case Q_ENTRY_SIG: {
            BSP_ledOff();
            status = Q_HANDLED();
            break;
        }
        case Q_TIMEOUT_SIG: {
            QActive_arm((QActive *)me, BSP_TICKS_PER_SEC/2U);
            status = Q_TRAN(&Blinky_on);
            break;
        }
        default: {
            status = Q_SUPER(&QHsm_top);
            break;
        }
    }
    return status;
}
/*.......................................................................*/
QState Blinky_on(Blinky * const me) {
    QState status;
    switch (Q_SIG(me)) {
        case Q_ENTRY_SIG: {
            BSP_ledOn();
            status = Q_HANDLED();
            break;
        }
        case Q_TIMEOUT_SIG: {
            QActive_arm((QActive *)me, BSP_TICKS_PER_SEC/2U);
            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.c`) is actually **the same** on all platforms, including Windows and the embedded boards. The only difference is in the Board Support Package (`bsp.c`), which is specific for the target.

5 Creating your Own QP-nano 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-nano application examples, such as the super-simple Blinky, or a bit more advanced PELICAN crossing, 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 an other project settings, which will help you get started much faster.

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

This quick-start guide is intended to get the QP-nano installed and running on your system as quickly as possible, but to work with QP-nano 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:

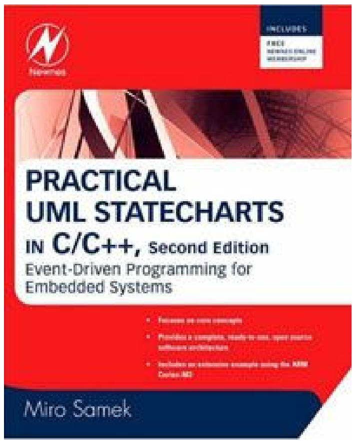
- The book “Practical UML Statecharts in C/C++, 2nd Edition” [PSiCC2] and the companion web-page to the book (http://www.state-machine.com/psicc2/)
- Free Support Forum for QP/QM (https://sourceforge.net/p/qpc/discussion/668726)
- QP Code Downloads summary (http://www.state-machine.com/downloads)
- QP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

7 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
Miro Samek
Newnes

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