



Quantum™ Leaps
innovating embedded systems



QDK™-nano

Renesas H8/Quark

Document Revision A
October 2008

Copyright © Quantum Leaps, LLC

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



Table of Contents

1	Introduction.....	1
1.1	What's Included in the QDK-nano?	2
1.2	Licensing QP-nano	2
2	Getting Started	3
2.1	Installation	3
2.2	Building and Running the Examples	4
2.2.1	Loading the Project into HEW.....	4
2.2.2	Running/Debugging the Examples	6
3	Non-Preemptive Configuration of QP-nano	7
3.1	The qpn_port.h Header File)	7
3.2	ISRs in the Non-preemptive "Vanilla" Configuration	8
3.3	QP Idle Loop Customization in QF_onIdle()	9
4	Preemptive Configuration with QK-nano.....	11
4.1	ISRs in the Preemptive Configuration with QK-nano	12
4.2	Idle Loop Customization in the QK Port	12
5	Board Support Package.....	13
5.1	Compiler Options Used	13
5.2	Linker Options Used	14
5.2.1	Specifying Program Sections	14
5.2.2	Specifying Stack and Heap Sizes	16
5.3	The BSP header file bsp.h	16
5.4	BSP initialization	17
5.5	Starting Interrupts in QF_onStartup()	17
5.6	Assertion Handling Policy in Q_onAssert()	17
6	Related Documents and References	18
7	Contact Information.....	19



1 Introduction

This **QP-nano™ Development Kit (QDK-nano)** describes how to use QP-nano™ event-driven platform with the Renesas H8 family of processors, the Renesas H8, H8S, H8/300 Standard C/C++ Toolchain, and the High Performance Embedded Workshop 4 (HEW4).

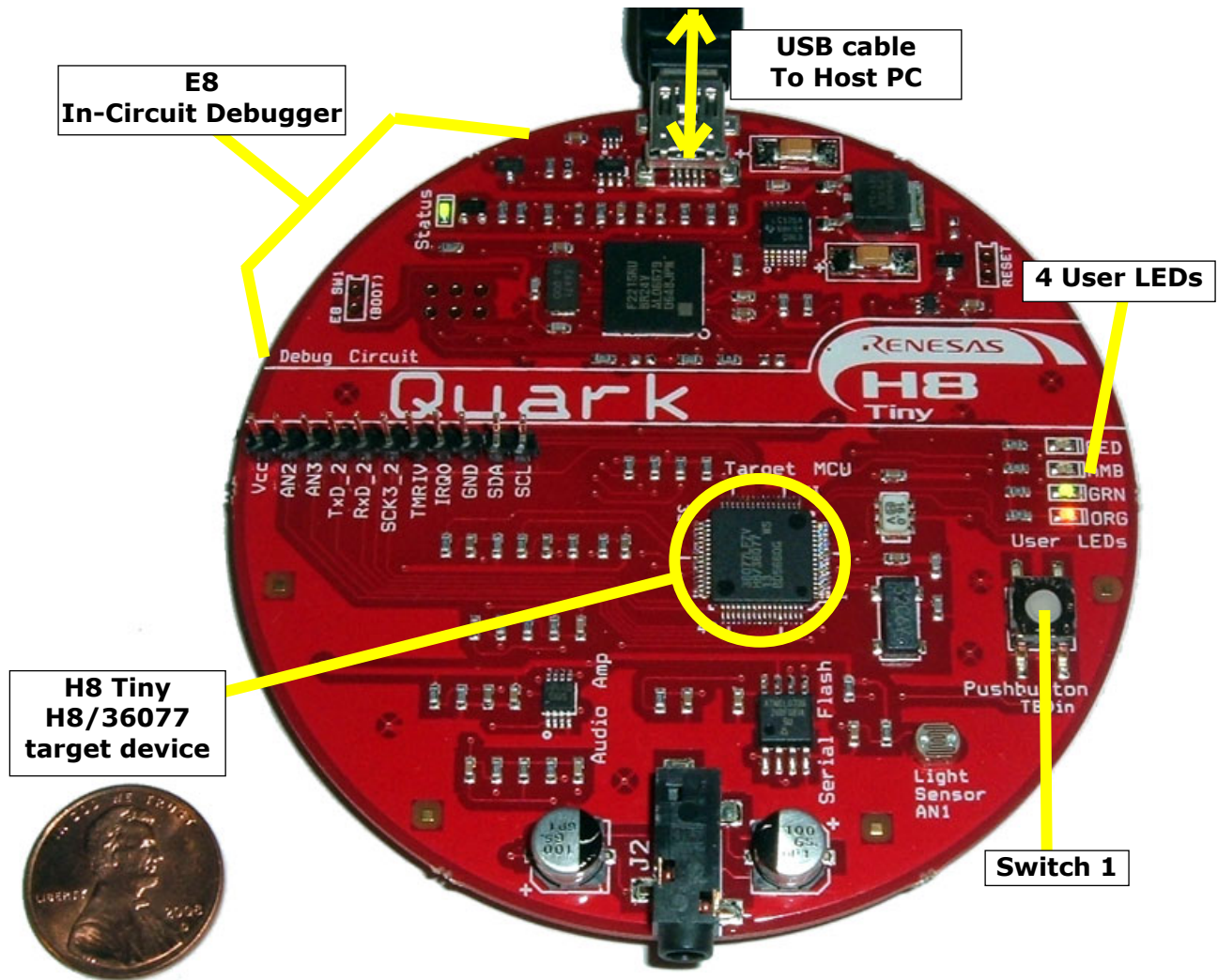


Figure 1 Renesas H8/Quark board with built-in E8 In-Circuit Debugger.

The actual hardware/software used to test this QDK-nano is described below (see Figure 1):

1. Renesas H8/QUARK evaluation board with E8 In-Circuit Debugger.
2. High Performance Embedded Workshop 4.04 (HEW4)
3. Renesas H8S, H8/300 Standard C/C++ Toolchain v6.1.2.0.
4. QP-nano v4.0.02 or higher.

As shown in Figure 1, the Renesas H8/Quark board contains the E8 In-Circuit Debugger that connects directly to the host development workstation via the provided USB cable. The USB connection also powers up the board, so no additional power is required. This QDK-nano has been tested with the H8/36077 device with 4KB of RAM and 56KB of onboard flash. However, the described port should be applicable to most H8, H8S, and H8/300 devices.

1.1 What's Included in the QDK-nano?

This QDK-nano provides the QP-nano port to H8 with the Renesas H8 compiler, the Board Support Package (BSP) and two versions of the PEdestrian LIght CONtrolled (PELICAN) crossing example application described in the Application Note "PELICAN Crossing Example" [QL AN-PELICAN 08].

1. PELICAN crossing with the cooperative "Vanilla" kernel; and
2. PELICAN crossing with the preemptive run-to-completion QK-nano kernel.

NOTE: Even though this QDK-nano is based on a specific development board (QUARK in this case), the most important parts of the QP-nano ports are applicable to all H8-based MCUs. In particular, the QP-nano port to the cooperative "Vanilla" kernel, as well as the QP-nano port to the preemptive QK-nano kernel are generic and should not need to change for other H8 systems.

1.2 Licensing QP-nano

The **Generally Available (GA)** distribution of QP-nano available for download from the www.state-machine.com/downloads website is offered with the following two licensing options:

- The GNU General Public License version 2 (GPL) as published by the Free Software Foundation and appearing in the file GPL.TXT included in the packaging of every Quantum Leaps software distribution. The GPL *open source* license allows you to use the software at no charge under the condition that if you redistribute the original software or applications derived from it, the complete source code for your application must be also available under the conditions of the GPL (GPL Section 2[b]).
- One of several Quantum Leaps commercial licenses, which are designed for customers who wish to retain the proprietary status of their code and therefore cannot use the GNU General Public License. The customers who license Quantum Leaps software under the commercial licenses do not use the software under the GPL and therefore are not subject to any of its terms.



For more information, please visit the licensing section of our website at: www.state-machine.com/licensing.

2 Getting Started

This section describes how to install, build, and use QDK-nano based on two examples. This information is intentionally included early in this document, so that you could start using QDK-nano™ as soon as possible.

NOTE: Every QDK-nano™ contains only example(s) pertaining to the specific MCU and compiler, but does not include the platform-independent baseline code of QP-nano™, which is available for a separate download. It is strongly recommended that you read Chapter 12 in [PSiCC2] before you start with this QDK-nano™.

2.1 Installation

The QDK-nano code is distributed in a ZIP archive (qdkn_h8-renesas_quark_<ver>.zip, where <ver> stands for a specific QDK-nano version, such as 4.0.02). You can uncompress the archive into any directory. The installation directory you choose will be referred henceforth as QP-nano Root Directory <qpn>. The following Listing 1 shows the directory structure and selected files included in the QP-nano distribution. (Please note that the QP directory structure is described in detail in a separate Quantum Leaps Application Note: "[QP Directory Structure](#)")

```

<qpn>/
+-examples\          - subdirectory containing the QP-nano example files
  +-h8\              - examples for H8
    +-renesas\       - examples compiled with Renesas H8 compiler
      +-pelican_quark\ - the PELICAN example for QUARK board, Vanilla kernel
        +-Debug\     - directory containing the binaries for the Debug build
        +-Release\   - directory containing the binaries for the Release build
        +-bsp.c       - Board Support Package for the H8 MCU
        +-bsp.h       - BSP header file
        +-main.c      - the main function
        +-oper.c      - the Operator state machine
        +-pelican.c   - the PELICAN crossing state machine
        +-pelican.h   - the PELICAN application header file
        +-qpn_port.h  - QP-nano configuration for this application
        +-pelican_quark.hwp - HEW project for this application
        +-pelican_quark.hws - HEW workspace for this application
        +-quarkdef.h  - Definitions for the QUARK board
        +-iodefine.h  - I/O definitions for H8/36077 (generated startup code)
        +-dbsct.c     - Definitions of program sections (generated startup code)
        +-intprg.c    - dummy definitions of interrupt handlers (generated code)
        +-resetprog.c - reset handler (generated startup code)
        +-sbrk.h      - heap section definition (generated startup code)
        +-sbrk.c      - heap section definition (generated startup code)
        +-stacksct.h  - stack section definition (generated startup code)
        +-typedef.h   - Exact-width types (generated startup code)
      +-renesas\     - examples compiled with Renesas H8 compiler
        +-pelican_qk_quark\ - the PELICAN example for QUARK board, QK-nano kernel
          +-Debug\   - directory containing the binaries for the Debug build
          +-Release\ - directory containing the binaries for the Release build
          +-bsp.c     - Board Support Package for the H8 MCU
          +-bsp.h     - BSP header file
          +-main.c    - the main function
          +-oper.c    - the Operator state machine
          +-pelican.c - the PELICAN crossing state machine
          +-pelican.h - the PELICAN application header file
  
```

```

|--+--qpnan_port.h - QP-nano configuration for this application
|--+--pelican_quark.hwp - HEW project for this application
|--+--pelican_quark.hws - HEW workspace for this application
|--+--quarkdef.h - Definitions for the QUARK board
|--+--iodefine.h - I/O definitions for H8/36077 (generated startup code)
|--+--dbsct.c - Definitions of program sections (generated startup code)
|--+--intprg.c - dummy definitions of interrupt handlers (generated code)
|--+--resetprog.c - reset handler (generated startup code)
|--+--sbrk.h - heap section definition (generated startup code)
|--+--sbrk.c - heap section definition (generated startup code)
|--+--stacksct.h - stack section definition (generated startup code)
|--+--typedef.h - Exact-width types (generated startup code)

+--include\
| +-qassert.h - embedded-systems-friendly assertions used in QP-nano
| +-qepn.h - The platform-independent QEP-nano header file
| +-qfn.h - The platform-independent QF-nano header file
| +-qkn.h - The platform-independent QK-nano header file
+--source/
| +-qepn.c - QEP-nano implementation
| +-qfn.c - QF-nano implementation
| +-qkn.c - QK-nano implementation

```

Listing 1 Selected QP directories and files after installing QDK-nano. The highlighted directories and files are provided in the QDK-nano-H8-Renesas_QUARK ZIP file.

2.2 Building and Running the Examples

The examples included in this QDK-nano are based on the standard PELICAN crossing application implemented with active objects (see Quantum Leaps Application Note: "PELICAN Crossing Application" [QL AN-PELICAN 08] includes in this QDK-nano).

The example directory contains the HEW workspace file `pelican_neutrino.hws` that you can load into the HEW IDE. The workspace contains two build configurations (Debug and Release) that you can select with the drop-down list, as shown in Figure 2.

2.2.1 Loading the Project into HEW

1. Connect the E8 debugger to the QUARK board with the provided USB cable
2. Launch HEW IDE and open the project `pelican_quark.hws` (located in `<qpnan>\examples\h8\renesas\pelican_quark\`). Figure 2 shows the screen shot of the HEW IDE after opening the project.
3. Establish a connection to the QUARK board by clicking OK in the "Connect" dialog box (see Figure 2).
4. Build the project by select Build | Build menu or by pressing F7.

NOTE: The PELICAN example workspace comes with two build configurations: Debug and Release. You select the build configuration through the drop-down box, as shown in Figure 2.

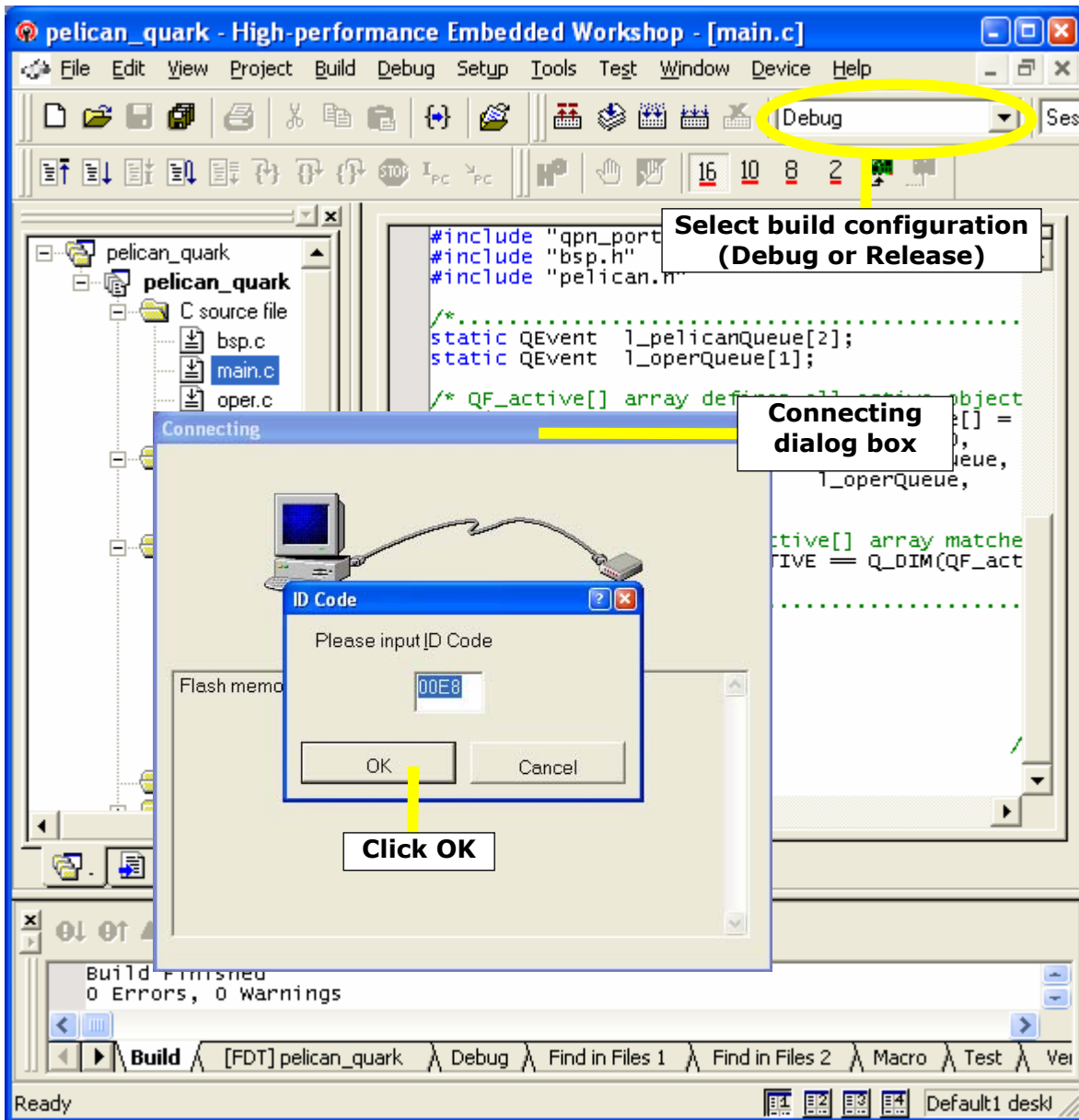


Figure 2 HEW IDE with the pelican_quark project

2.2.2 Running/Debugging the Examples

The HEW debugger can directly communicate with the E8 In-Circuit Debugger (see Figure 1). To load the code into the MCU's flash, select Debug | Download Modules menu (see Figure 2). This will launch the progress-bar message box. When the message box window goes away, the MCU is programmed.

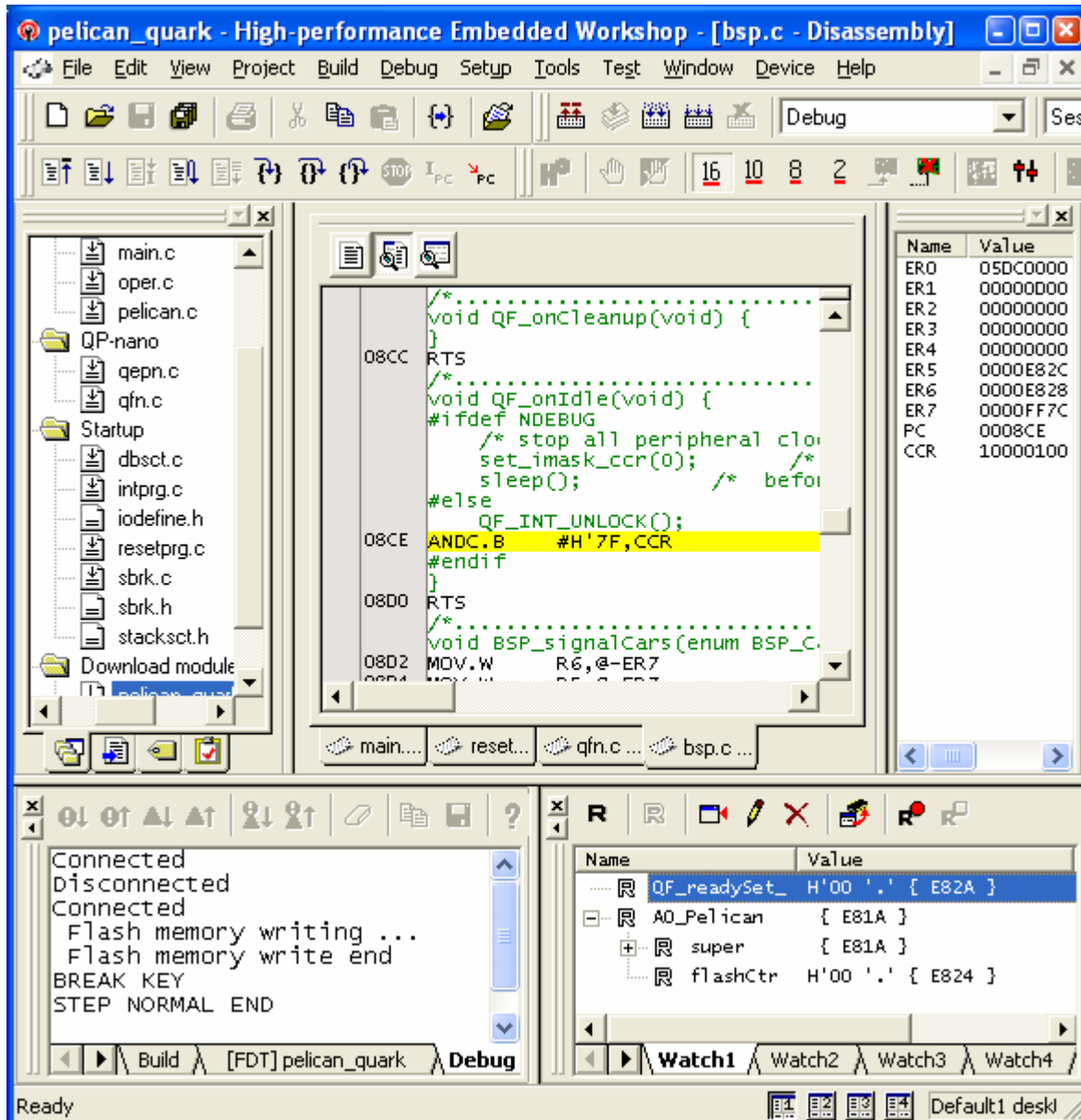


Figure 3 HEW Debugger with various views of the executing PELICAN application.

Figure 1 shows the PELICAN example running on the QUARK board. The operation of the PELICAN crossing is visualized by means of the User LEDs. The LEDs are used for the cars (red, yellow, and green LED as red, yellow, and green light for the cars). The bottom orange LED is used as signal for pedestrians (orange LED on means "DON'T WALK"). **The traffic light is activated by pressing Switch SW1** (see Figure 1).

3 Non-Preemptive Configuration of QP-nano

The example of using QP-nano with the cooperative “Vanilla” kernel is located in the directory: <qp>\examples\h8\renesas\pelican_quark\. This section describes the generic QP-nano configuration, which consist of the qpn_port.h header file. The board-specific elements are common for both the non-preemptive and preemptive (QK-nano) configuration and will be covered in Section 5.

3.1 The qpn_port.h Header File

You configure and customize QP-nano through the header file qpn_port.h, which is included by the QP-nano source files (qepn.c and qfn.c) as well as in all your application C modules.

NOTE: The QP-nano port to the cooperative “Vanilla” kernel qpn_port.h is generic and should not need to change (except for the QF_MAX_ACTIVE definition) for other H8 systems.

```
#ifndef qpn_port_h
#define qpn_port_h

(1) #define Q_NFSM
(2) #define Q_PARAM_SIZE      1
(3) #define QF_TIMEEVT_CTR_SIZE  2

/* maximum # active objects--must match EXACTLY the QF_active[] definition */
(4) #define QF_MAX_ACTIVE      2

/* interrupt locking policy for the task level */
(5) #define QF_INT_LOCK()      set_imask_ccr(1)
(6) #define QF_INT_UNLOCK()   set_imask_ccr(0)

/* interrupt locking policy for ISR level */
(7) /* #define QF_ISR_NEST */ /* nesting of ISRs not allowed */

/* exact-width integer types (NC30 compiler does NOT provide <stdint.h>) */
(8) typedef unsigned char  uint8_t;
    typedef signed   char  int8_t;
    typedef unsigned short uint16_t;
    typedef signed   short int16_t;
    typedef unsigned long  uint32_t;
    typedef signed   long  int32_t;

(9) #include <machine.h> /* prototype for set_imask_ccr() */
(10) #include "qepn.h" /* QEP-nano platform-independent public interface */
(11) #include "qfn.h" /* QF-nano platform-independent public interface */

#endif /* qpn_port_h */
```

Listing 2 qpn_port.h header file for the non-preemptive QP-nano configuration and Renesas H8 compiler

- (1) Defining the macro Q_NFSM eliminates the code for the simple non-hierarchical FSMs.
- (2) The macro Q_PARAM_SIZE defines the size (in bytes) of the scalar event parameter. The allowed values are 0 (no parameter), 1, 2, or 4 bytes. If you don't define this macro in qpn_port.h, the default of 0 (no parameter) will be assumed.

- (3) The macro `QF_TIMEEVT_CTR_SIZE` defines the size (in bytes) of the time event down-counter. The allowed values are 0 (no time events), 1, 2, or 4 bytes. If you don't define this macro in `qpn_port.h`, the default of 0 (no time events) will be assumed.
- (4) You must define the `QF_MAX_ACTIVE` macro as the exact number of active objects used in the application. The provided value must be between 1 and 8 and must be consistent with the definition of the `QF_active[]` array in `main.c`.
- (5-6) The macros `QF_INT_LOCK()/QF_INT_UNLOCK()` define the task-level interrupt locking policy for QP-nano (see Section 12.3.2 in Chapter 12 in [PSiCC2]). For the H8, the inline function `set_imask_ccr(1)` expands to a single assembly instruction `ORC.B #H'80,CCR`. Conversely, the inline function `set_imask_ccr(0)` expands to a single assembly instruction `ANDC.B #H'7F,CCR`.
- (7) This QP-nano port to H8 does **not** allow nesting of interrupts.

The H8 microcontrollers do not provide any advanced interrupt prioritization in hardware. In other words, if you enable interrupts at the CPU level (by clearing the I flag in the CCR), the H8 hardware will allow all interrupts, including the interrupt level currently being serviced. For that reason unlocking interrupts inside ISRs is not advisable, because ISRs are typically not reentrant. Consequently, QF services invoked from the ISRs (such as `QF_tick()`, `QF_publish()`, `QActive_postFIFO`, etc.) must not inadvertently unlock interrupts that are locked in hardware upon the entry to the interrupt processing. All this means that QF must use an interrupt locking/unlocking scheme that allows for nesting critical sections. QF provides such a policy called "saving and restoring interrupt status" and described in Chapter 7 of [PSiCC2].

NOTE: The H8 hardware "prioritizes" only those interrupts that arrive while the interrupts are disabled. If multiple interrupts arrive during that time, the H8 hardware picks the one with the highest hardware priority (see H8 Hardware Manual [Renesas H8 HW]). However, the H8 has no prioritized interrupt controller, which would keep track of the priority of the currently serviced interrupt and allow only interrupts of higher priority to preempt the currently running interrupt.

Therefore, unlocking interrupts inside ISRs (the H8 hardware automatically locks interrupts upon entry to an ISR) is **not** advisable. Allowing interrupts to nest can lead to all sorts of priority inversions, including the pathological case of an interrupt preempting itself.

NOTE: This QP-nano port assumes that you never unlock interrupts inside ISRs.

- (8) The Renesas H8 compiler does not provide the C99-standard exact-width integer types, which are here defined using the typedef directives.
- (9) The `machine.h` provides prototypes for intrinsic functions, such as `set_imask_ccr()`.
- (10) The `qpn_port.h` header file must include the QEP-nano event processor interface `qepn.h`.
- (11) The `qpn_port.h` header file must include the QF-nano real-time framework interface `qfn.h`.

3.2 ISRs in the Non-preemptive "Vanilla" Configuration

The NC30 compiler supports writing interrupts in C. In the non-preemptive case used in this QDK-nano, **all** ISRs must invoke the macro `QF_INT_UNLOCK()`. This is necessary to avoid nesting of critical sections, which the simple "unconditional interrupt locking and unlocking" policy does not support.

The following Listing 3 shows the ISR servicing Timer A0 system clock tick interrupt, which calls the `QF_tick()` function.

```
(1) __interrupt(vect = 26)
(2) void timer_z_isr(void) {
(3)     /* clear any level-sensitive interrupt sources, if necessary ... */
(4)     QF_tick();
        /* perform other work of the ISR, e.g., switch debouncing */
}
```

Listing 3 Time tick interrupt for H8 calling the `QF_tick()` function to generate `Q_TIMEOUT` events.

- (1) The ISR in C is always compiler-specific, as the C standard does not define how to specify ISRs. In the case of the Renesas H8 compiler for interrupt must be declared with the `__interrupt` extended keyword. You also need to specify the interrupt vector number in the parentheses as shown.
- (2) The ISR in C that follows the interrupt vector specification must have a `void (void)` signature.
- (3) The level-sensitive interrupt sources should be cleared somewhere in the ISR body
- (4) The time-tick ISR must invoke `QF_tick()`, and can also perform other things, if necessary. The function `QF_tick()` cannot be reentered, that is, it necessarily must run to completion and return before it can be called again. This requirement is automatically fulfilled, because the H8 hardware locks interrupts upon the interrupt entry and will not allow the same interrupt to preempt currently running interrupt.

NOTE: You should **not** enable interrupts inside ISRs, because the H8 CPU does not prioritize interrupts in hardware. Also, the `QF_tick()` and `QActive_postISR()` functions (the only QP-nano services callable from ISRs) do not lock interrupts internally, because QP-nano is not configured for nesting interrupts (see Listing 2(7)).

3.3 QP Idle Loop Customization in `QF_onIdle()`

The cooperative “vanilla” kernel can very easily detect the situation when no events are available, in which case `QF_run()` calls the `QF_onIdle()` callback. You can use `QF_onIdle()` to suspended the CPU to save power, if your CPU supports such a power-saving mode. Please note that `QF_onIdle()` is called repetitively from the event loop whenever the event loop has no more events to process, in which case only an interrupt can provide new events. The `QF_onIdle()` callback is called with interrupts **locked**, because the determination of the idle condition might change by any interrupt posting an event.

H8 microcontrollers support several power-saving levels (consult the specific data sheet for details). The following piece of code shows the `QF_onIdle()` callback that puts H8 core into the idle power-saving mode. Please note that H8 architecture allows for very **atomic** setting the low-power mode and enabling interrupts at the same time (see the article “Using Low-Power Modes in Foreground/Background Systems” [Samek 07a]).

```
(1) void QF_onIdle(void) {  
(2) #ifdef NDEBUG /* low-power mode interferes with debugging */  
(3) /* stop all peripheral clocks that you can in your applicaiton ... */  
(4) set_imask_ccr(0); /* the following SLEEP instruction will execute */  
(5) sleep(); /* before entering any pending interrupt, see NOTE01 */  
#else  
(6) QF_INT_UNLOCK(); /* just unlock interrupts */  
#endif  
}
```

Listing 4 QF_onIdle() for the non-preemptive (“vanilla”) QP-nano port to H8

- (1) The QF_onIdle() callback is always called with interrupts locked to prevent any race condition between posting events from ISRs and transitioning to the sleep mode.
- (2) The low-power mode is entered only in the Release (not DEBUG) configuration.
- (3) The clock management registers are setup the desired sleep mode. Please note that the sleep mode is not active until the SLEEP command.
- (4) The interrupts are unlocked with the set_imask_ccr(0) intrinsic function.
- (5) The sleep mode is activated with the sleep() intrinsic function.

NOTE: As described in “H8/300H Series Software Manual”, Section 2.2.5 “ANDC” [Renesas H8 Sw], **no** interrupt requests, including NMI, are accepted immediately after execution of the ANDC instruction. This means that the ANDC instruction and the immediately following SLEEP instruction are executed **atomically**.

CAUTION: The instruction pair (ANDC.B #H'7F,CCR, SLEEP) should never be separated by any other instruction.

- (6) In the DEBUG configuration the interrupts are simply unlocked.

NOTE: Every path through QF_onIdle() callback function must ultimately unlock interrupts.

4 Preemptive Configuration with QK-nano

The QP port with the preemptive kernel (QK) is remarkably simple and very similar to the “vanilla” port. In particular, the interrupt locking/unlocking policy is the same, and the BSP is identical, except some small additions to the ISRs.

The PELICAN example for the QK port is provided in the directory <qp>\examples\h8\renesas\pelican_qk_quark.

You configure and customize QP-nano through the header file `qpn_port.h`, which is included by the QP-nano source files (`qepn.c`, `qfn.c`, and `qkn.c`) as well as in all your application C modules. The following Listing 5 shows the `qpn_port.h` header file for the QK-nano port. Except for the highlighted fragments, the listing is identical as in the non-preemptive case (Section 3)

```

#ifndef qpn_port_h
#define qpn_port_h

#define Q_NFSM
#define Q_PARAM_SIZE      1
#define QF_TIMEEVT_CTR_SIZE  2

/* maximum # active objects--must match EXACTLY the QF_active[] definition */
#define QF_MAX_ACTIVE      2

/* interrupt locking policy for the task level */
#define QF_INT_LOCK()      set_imask_ccr(1)
#define QF_INT_UNLOCK()   set_imask_ccr(0)

/* interrupt locking policy for ISR level */
/* nesting of ISRs not allowed */
/* #define QF_ISR_NEST */

/* interrupt entry/exit for QK-nano */
(1) #define QK_ISR_ENTRY()  ((void)0)
(2) #define QK_ISR_EXIT()  QK_SCHEDULE_()

/* exact-width integer types (NC30 compiler does NOT provide <stdint.h>) */
typedef unsigned char  uint8_t;
typedef signed   char  int8_t;
typedef unsigned short uint16_t;
typedef signed   short int16_t;
typedef unsigned long  uint32_t;
typedef signed   long  int32_t;

#include <machine.h> /* prototype for set_imask_ccr() */

#include "qepn.h" /* QEP-nano platform-independent public interface */
#include "qfn.h" /* QF-nano platform-independent public interface */
(3) #include "qkn.h" /* QK-nano platform-independent public interface */

```

Listing 5 `qpn_port.h` header file for the preemptive QK-nano configuration

(1-2) The interrupt entry and exit macro for QK-nano are defined consistently with the interrupt nesting policy, which does not allow interrupt nesting.

(3) The preemptive configuration of QP-nano is selected by including the `qkn.h` header file.

When interrupt nesting is *not* allowed (the macro `QF_ISR_NEST` is *not* defined in `qpn_port.h`), QK-nano allows for a simpler interrupt handling compared to the full-version QK. Specifically, when in-

errupts cannot nest you don't need to increment the interrupt nesting counter (`QK_intNest_`) upon the ISR entry, and you don't need to decrement it upon the ISR exit. In fact, when the macro `QF_ISR_NEST` *not* defined, the `QK_intNest_` counter is not even available. This simplification is possible, because QP-nano uses special ISR-version of the event posting function `QActive_postISR()`, which does *not* call the QK-nano scheduler. Consequently, there is no need to prevent the synchronous preemption within ISRs (see Chapter 10 in [PSiCC2]).

4.1 ISRs in the Preemptive Configuration with QK-nano

As all preemptive kernels, QK-nano must be notified about interrupt entry and exit. You achieve this by means of the QK-nano macros `QK_ISR_ENTRY()` and `QK_ISR_EXIT()`, as shown in Listing 6.

```

__interrupt(vect = 26)
void timer_z_isr(void) {
    static uint8_t btn_debounced = 0;
    static uint8_t debounce_state = 0;
    uint8_t btn;

    QK_ISR_ENTRY();                /* inform QK-nano about ISR entry */

    TZ0.TSR.BIT.IMFA = 0;         /* clear compare match flag */

    QF_tick();                    /* process all armed time events */

    . . .                        /* debounce the switch SW1 */

    QK_ISR_EXIT();                /* inform QK-nano about ISR exit */
}

```

Listing 6 Time tick interrupt calling `QF_tick()` function to manage armed time events and QK-nano ISR entry/exit macros.

4.2 Idle Loop Customization in the QK Port

As described in Chapter 10 of [PSiCC2], the QK idle loop executes only when there are no events to process. The QK allows you to customize the idle loop processing by means of the callback `QK_onIdle()`, which is invoked by every pass through the QK idle loop. You can define the platform-specific callback function `QK_onIdle()` to save CPU power, or perform any other "idle" processing (such as Quantum Spy software trace output).

NOTE: The idle callback `QK_onIdle()` is invoked with interrupts unlocked (which is in contrast to `QF_onIdle()` that is invoked with interrupts locked, see Section).

The following Listing 7 shows an example implementation of `QK_onIdle()` for the H8.

```

void QK_onIdle(void) {
#ifdef NDEBUG                /* low-power mode interferes with debugging */
    /* stop all peripheral clocks that you can in your applicaiton ... */
    sleep();                 /* before entering any pending interrupt */
#endif
}

```

Listing 7 `QK_onIdle()` callback for H8.

5 Board Support Package

The Board Support Package (BSP) for H8 with the non-preemptive Vanilla kernel is located in the directory: <qpn>\examples\h8\renesas\pelican_quark\ and consists of the following files:

1. qpn_port.h contains the platform-specific customization of QP-nano (the QP-port)
2. bsp.h contains the Board Support Package interface (BSP)
3. bsp.c contains the implementation of the BSP, which includes all ISRs and all platform-specific QP-nano callbacks.

NOTE: This QDK-nano uses the recommended by Renesas C-startup for H8. The C-startup code is comprised of the following files (see also Listing 1):

1. stacksct.h – stack size definition (**customize for your application**)
2. sbrk.h – heap size definition (**customize for your application**)
3. sbrk.c
4. iodefne.c
5. dbsct.h
6. intrpg.c – default interrupt handlers (**customize for your application**)
7. resetprg.c
8. typedefine.h

Out of these 8 files only the indicated 3 files need customization for a specific application.

5.1 Compiler Options Used

You set the compiler and linker options through the HEW IDE. The compiler options are as follows:

```
-cpu=300HN -include="$(PROJDIR)\." "$ (PROJDIR)\..\..\..\include" -  
object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -nolist -chginpath -nologo
```

The `-debug` option denotes that the compiler should generate the Debug information. This option is only used in the Debug build. Both the Debug and Release builds use the generic CPU type "With no specification".

NOTE: You can access the compiler options by selecting the Build->Renesas H8S, H8/300 Standard Toolchain..., or by right-clicking on any of the C modules in the Workspace window and choosing the Build Options/Renesas H8S, H8/300 Standard Toolchain... pop-up menu.

5.2 Linker Options Used

5.2.1 Specifying Program Sections

The H8 devices have typically RAM divided into two separate blocks, as shown in the memory map of the H8/36077 device published in Section 6.3 of the "SKP36077 User's Manual" [Renesas SKP36077] and reproduced in Figure 5 below.

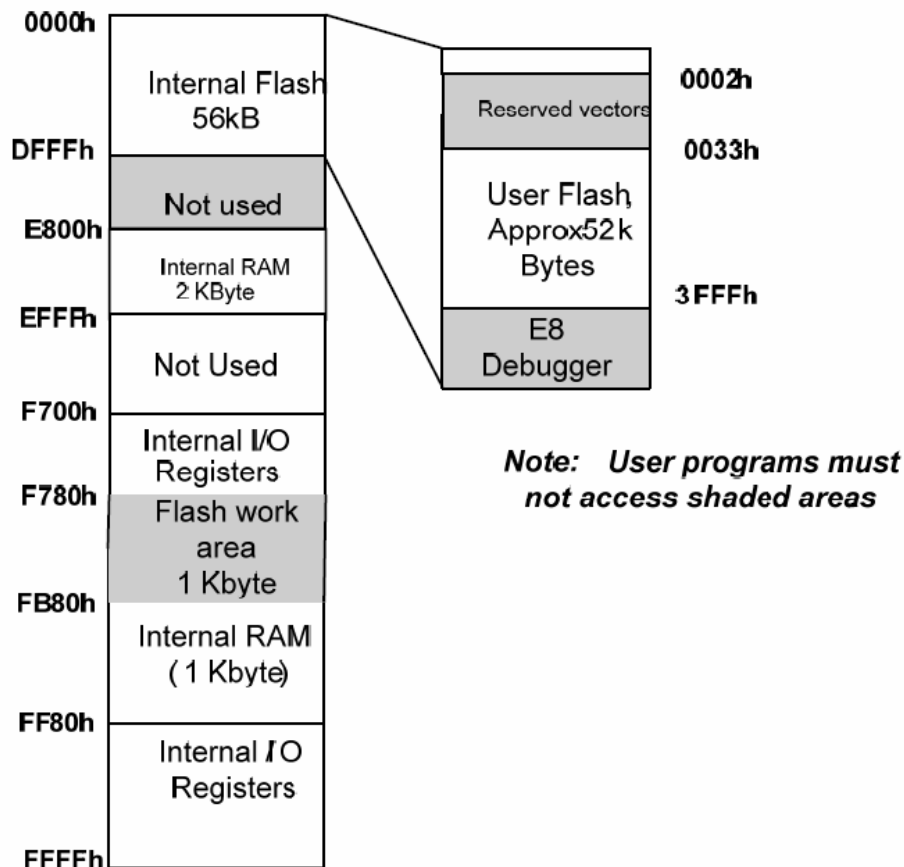


Figure 4 Memory map of the H8/36077 device with the "Kernel" program (Renesas ROM monitor program).

The HEW IDE allows to specify very precisely all the program sections, as shown in the screen shot in Figure 5. For the QUARK board, the sections are configured according to the memory map for the H8/36077 device.

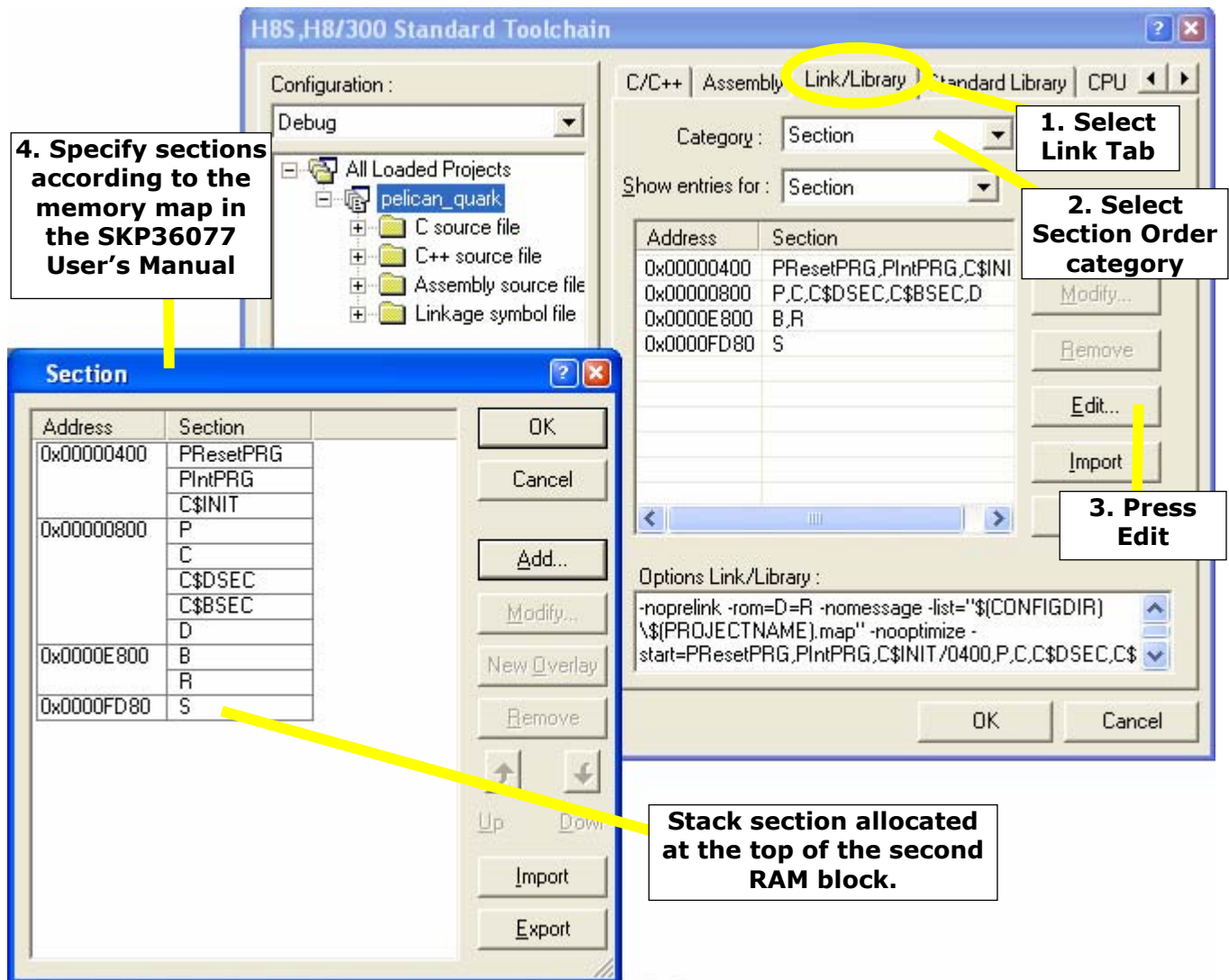


Figure 5 Specifying the sections names and addresses in the HEW IDE.

In particular, the Section dialog box shows the special RAM section **S** for the stack located at 0x0000FD80, that is, in the **second** RAM block. The stack does not use the whole 1KB of RAM available in that block, and you can define other RAM sections in the lower part of this RAM (addresses 0x0000FB80 .. 0x0000FD7F).

NOTE: The stack must entirely fit in the RAM block, so the start address for the stack section plus the stack size must be less than the end of the RAM block (at 0x0000FF80 in case of H8/36077). The stack size is configured in the file stacksct.h described below.

5.2.2 Specifying Stack and Heap Sizes

You specify the size of the stack in the file `stackst.h`, which contains the single `#pragma` directive:

```
#pragma stacksize 0x200
```

NOTE: The QK preemptive kernel generally requires more stack space than the cooperative "Vanilla" kernel. You need to adjust the `stacksize` value for your system.

You specify the size of the heap in the file `sbrk.h`, which is shown below:

```
#define HEAPSIZE 1
```

This QDK-nano-H8 does not use the heap¹, and consequently the `heapsize` is defined to the minimum value of 1.

5.3 The BSP header file `bsp.h`

```
#ifndef bsp_h
#define bsp_h

(1) #include "iodefine.h"          /* I/O register definitions for H8/36077 */
/*-----*/
(2) #define BSP_TICKS_PER_SEC    50

/* street signals .....*/
enum BSP_CarsSignal {
    CARS_RED, CARS_YELLOW, CARS_GREEN, CARS_OFF
};
enum BSP_PedsSignal {
    PEDS_DONT_WALK, PEDS_WALK, PEDS_BLANK
};
void BSP_init(void);
void BSP_signalCars(enum BSP_CarsSignal sig);
void BSP_signalPeds(enum BSP_PedsSignal sig);

void BSP_showState(uint8_t prio, char const *state);

#endif                               /* bsp_h */
```

Listing 8 The `bsp.h` for the PELICAN crossing example.

- (1) The header file "iodefine.h" provides the definitions of the H8 special function registers.
- (2) The BSP defines the desired ticking rate. This constant is useful for defining timeouts, which are always specified in units of clock ticks.

¹ Using the heap in real-time embedded devices can cause many problems, such as heap fragmentation, non-determinism, concurrency issues, etc.

5.4 BSP initialization

The following `BSP_init()` function from the PELICAN crossing application for the QUARK board configures the PIO lines for the User LEDs, the User switches, and the system clock tick:

```
void BSP_init(void) {
    WDT.TCSRWD.BYTE = 0x10;           /* disable watchdog */
    WDT.TCSRWD.BYTE = 0x00;

    MSTCR2.BIT.MSTTZ = 0;             /* turn on TimerZ */
    TZ0.TCR.BIT.TPSC = 3;             /* internal clock phi/8 */
    TZ0.TCR.BIT.CCLR = 1;
    TZ0.GRA = (uint16_t)((f1_CLK_SPEED/8 + BSP_TICKS_PER_SEC/2)
                        / BSP_TICKS_PER_SEC);
    TZ0.TIER.BIT.IMIEA = 1;          /* compare match interrupt enable */

    /* enable the User LEDs... */
    LED0_DDR_1();                    /* configure LED0 pin as output */
    LED1_DDR_1();                    /* configure LED1 pin as output */
    LED2_DDR_1();                    /* configure LED2 pin as output */
    LED3_DDR_1();                    /* configure LED3 pin as output */
    LED0 = LED_OFF;
    LED1 = LED_OFF;
    LED2 = LED_OFF;
    LED3 = LED_OFF;

    /* enable the Switch... */
    SW1_DDR = 0;
}
```

5.5 Starting Interrupts in QF_onStartup()

QP-nano invokes the `QF_onStartup()` callback just before starting the event loop inside `QF_run()`. The `QF_onStartup()` function must start the interrupts configured earlier. In this BSP only the system tick interrupt is started.

```
void QF_onStartup(void) {
    TZ.TSTR.BIT.STR0 = 1;             /* start TimerZ */
}
```

5.6 Assertion Handling Policy in Q_onAssert()

As described in Chapter 6 of [PSiCC2], all QP components use internally assertions to detect errors in the way application is using the QP services. You need to define how the application reacts in case of assertion failure by providing the callback function `Q_onAssert()`. Typically, you would put the system in fail-safe state and try to reset. It is also a good idea to log some information as to where the assertion failed. The following code fragment shows the `Q_onAssert()` callback. The function simply locks all interrupts and enters a for-ever loop. This policy is only adequate for testing, but probably is **not** adequate for production release.

```
void Q_onAssert(char const Q_ROM * const Q_ROM_VAR file, int line) {
    QF_INT_LOCK();                    /* lock the interrupts */
    for (;;) {                        /* hang in this for-ever loop */
    }
}
```

6 Related Documents and References

Document	Location
[PSiCC2] "Practical UML Statecharts in C/C++, Second Edition", Miro Samek, Newnes, 2008	Available from most online book retailers, such as amazon.com . See also: http://www.quantum-leaps.com/psicc2.htm
[QP-nano 08] "QP-nano Reference Manual", Quantum Leaps, LLC, 2008	http://www.quantum-leaps.com/doxygen/qpn/
[QL AN-Directory 07] "Application Note: QP Directory Structure", Quantum Leaps, LLC, 2007	http://www.quantum-leaps.com/doc/AN_QP_Directory_Structure.pdf
[QL AN-PELICAN 08] "Application Note: PELICAN Crossing Application", Quantum Leaps, LLC, 2008	http://www.quantum-leaps.com/doc/AN_PELICAN.pdf
[Renesas H8 HW] "H8/36077 Group Hardware Manual", Renesas 2005	Renesas document REJ09B0216-0100, available online at www.renesas.com
[Renesas H8 Sw] "H8/300H Series Software Manual", Renesas, 2004.	Renesas document REJ09B0213-0300, available online at www.renesas.com
[Renesas H8 compiler] "H8S, H8/300 Series C/C++ Compiler, Assembler, Optimizing Linkage Editor Compiler Package Ver.6.01 User's Manual", Renesas 2004	Renesas document REJ10B0161-0100, available online at www.renesas.com .
[Renesas SKP36077] "SKP36077 StarterKit Plus User's Manual", Renesas 2007	Document included in PDF with SKP36077 kit.
[Samek 06b] "Build a Super Simple Tasker", Embedded Systems Design, Miro Samek and Robert Ward, July 2006	http://www.embedded.com/shared/-printableArticle.jhtml?articleID=190302110
[Samek 07a] "Using Low-Power Modes in Foreground/Background Systems", Miro Samek, to be published in ESD, October 2007	www.embedded.com/mag.htm

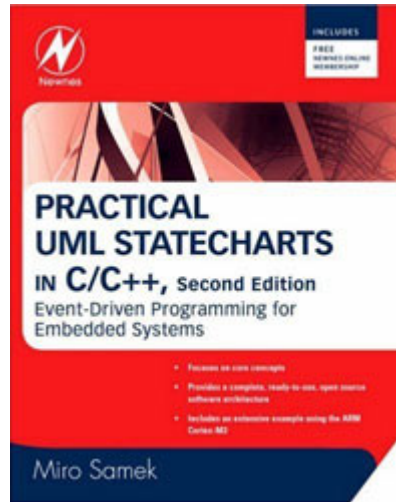
7 Contact Information

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>



Renesas Technology Corp.
2-6-2, Ote-machi Chiyoda-ku,
Tokyo,100-0004
Japan
WEB: www.renesas.com



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

