Quantum™ LeaPs
innovating embedded systems

# QDK™-nano
## Renesas RSKM32C87-NC308

**Document Revision C**
**November 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 ........ 13

**5 Board Support Package ........ 14**
5.1 Compiler Options Used ........ 14
5.2 Linker Options Used ........ 15
5.2.1 Specifying Program Sections ........ 15
5.2.2 Specifying Stack and Heap Sizes ........ 17
5.3 The BSP header file bsp.h ........ 17
5.4 BSP initialization ........ 17
5.5 Starting Interrupts in QF_onStartup() ........ 19
5.6 Assertion Handling Policy in Q_onAssert() ........ 19

**6 Related Documents and References ........ 20**

**7 Contact Information ........ 21**

# 1 Introduction

This **QP-nano™ Development Kit** (QDK-nano) describes how to use QP-nano™ event-driven platform with the Renesas M16C processors, the Renesas NC30 compiler, and the High Performance Embedded Workshop 4 (HEW4).

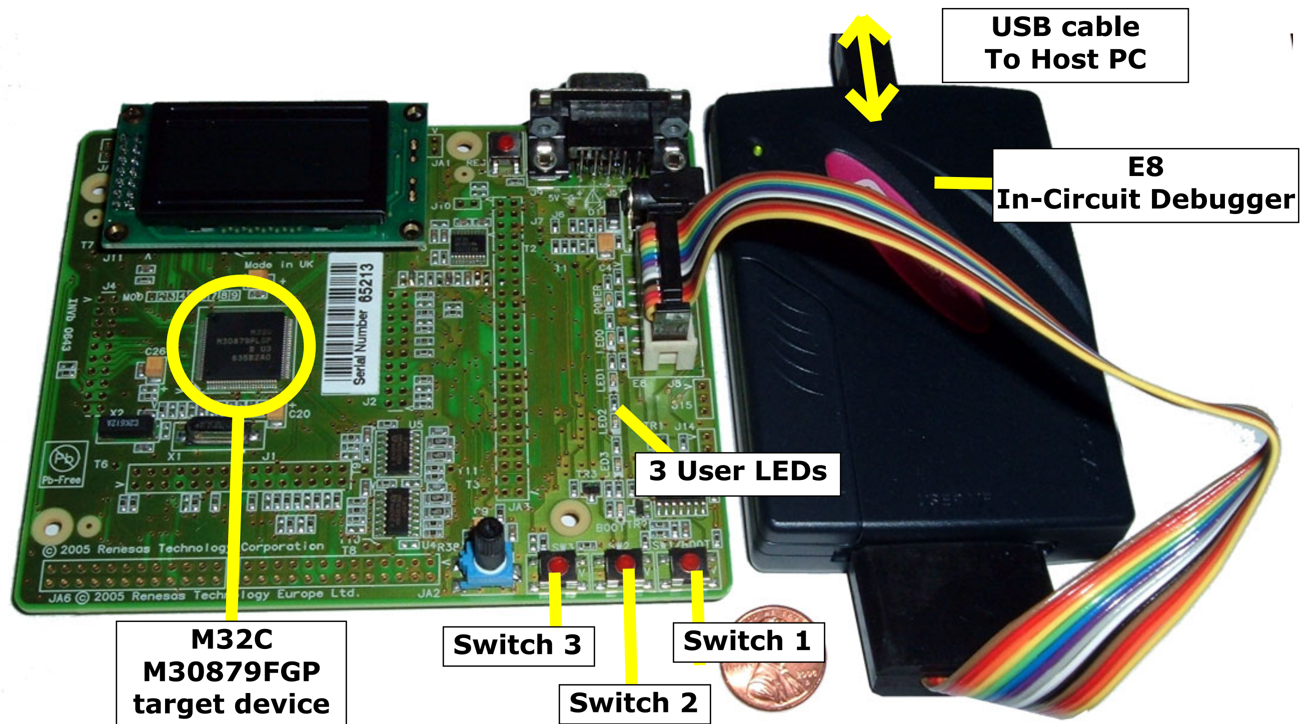



**Figure 1 Renesas RSKM32C87 Starter Kit with E8 In-Circuit Debugger.**

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

1. Renesas RSKM32C87 development board with E8 In-Circuit Debugger.
2. High Performance Embedded Workshop 4.04 (HEW4)

Copyright © Quantum Leaps, LLC. All Rights Reserved.

3. Renesas NC308 compiler for M32C/90,80,M16C/80,70 Series v5.41.01.
4. QP-nano v4.0.02 or higher.

As shown in Figure 1, the Renesas RSKM32C87 Starter Kit Plus contains the RSKM32C87 board and 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 has been tested with the M32C/87 M30879FGP device with 49KB of RAM and 1MB of on-board flash. However, the described port should be applicable to most M32C devices as well as R32C in the future.

## 1.1 What's Included in the QDK-nano?

This QDK-nano provides the QP-nano port to M32C with the Renesas NC308 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 (RSKM32C87 in this case), the most important parts of the QP-nano ports are applicable to all M32C-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 M16C 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.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

# 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_m32c-nc308_rskm32c87_<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>/                    - QP-nano Root Directory
  |
  +-examples\             - subdirectory containing the QP-nano example files
  | +-m16c\               - examples for M16C
  | | +-nc308\            - examples compiled with Renesas NC308 compiler
  | | | +-pelican_rskm32c87\ - the PELICAN example for RSKM32C87 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 M16C 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_rskm32c87.hwp - HEW project for this application
  | | | | +-pelican_rskm32c87.hws - HEW workspace for this application
  | | | | +-cregdef.h     - definitions of stack, heap, and standard I/O sizes
  | | | | +-firm.c        - definition of ROM section used by the ROM-Monitor
  | | | | +-firm_ram.c    - definition of RAM section used by the ROM-Monitor
  | | | | +-fvector.c     - fixed vector table for M32C/80
  | | | | +-heapdef.h     - heap size definition
  | | | | +-heap.c        - heap section definition
  | | | | +-initsct.c     - initialization of sections in C
  | | | | +-intprg.c      - dummy definitions of interrupt handlers
  | | | | +-resetprog.c   - reset handler
  | | | | +-rskm32c87def.h - Definitions for the RSKM32C87 board
  | | | | +-sfr32c87.h     - Special Function Registers for the M32C/87 device
  | | | | +-lcd.c         - LCD driver for the RSKM16C26A board
  | | | | +-lcd.h         - LCD driver for the RSKM16C26A board
  | | | | |
  | | +-nc308\            - examples compiled with Renesas NC308 compiler
  | | | +-pelican_qk_rskm32c87\ - the PELICAN example for RSKM32C87 board, QK 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 M16C MCU
  | | | | +-bsp.h         - BSP header file
  | | | | +-main.c        - the main function
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

```
| | | | +-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_qk_rskm32c87.hwp - HEW project for this application
| | | | +-pelican_qk_rskm32c87.hws - HEW workspace for this application
| | | | +-pelican_rskm32c87.hwp - HEW project for this application
| | | | +-pelican_rskm32c87.hws - HEW workspace for this application
| | | | +-cregdef.h      - definitions of stack, heap, and standard I/O sizes
| | | | +-firm.c         - definition of ROM section used by the ROM-Monitor
| | | | +-firm_ram.c     - definition of RAM section used by the ROM-Monitor
| | | | +-fvector.c      - fixed vector table for M32C/80
| | | | +-heapdef.h      - heap size definition
| | | | +-heap.c         - heap section definition
| | | | +-initsct.c      - initialization of sections in C
| | | | +-intprg.c       - dummy definitions of interrupt handlers
| | | | +-resetprog.c    - reset handler
| | | | +-rskm32c87def.h - Definitions for the RSKM32C87 board
| | | | +-sfr32c87.h       - Special Function Registers for the M32C/87 device
| | | | +-lcd.c          - LCD driver for the RSKM16C26A board
| | | | +-lcd.h          - LCD driver for the RSKM16C26A board
| | | | |
| |
+-include\             - subdirectory containing the QP-nano public interface
| +-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/              - QP-nano source files
| +-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-M32C-NC308_RSKM32C87 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_rskm32c87.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 RSKM32C87 board with the provided ribbon cable as described in the Quick Start Guide.
2. Connect the E8 debugger to the PC with the provided USB cable as described in the Quick Start Guide.
3. Launch HEW IDE and open the project `pelican_rskm32c87.hws` (located in `<qpn>\examples\m16c\-nc308\pelican_rskm32c87\`). Figure 2 shows the screen shot of the HEW IDE after opening the project.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

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.

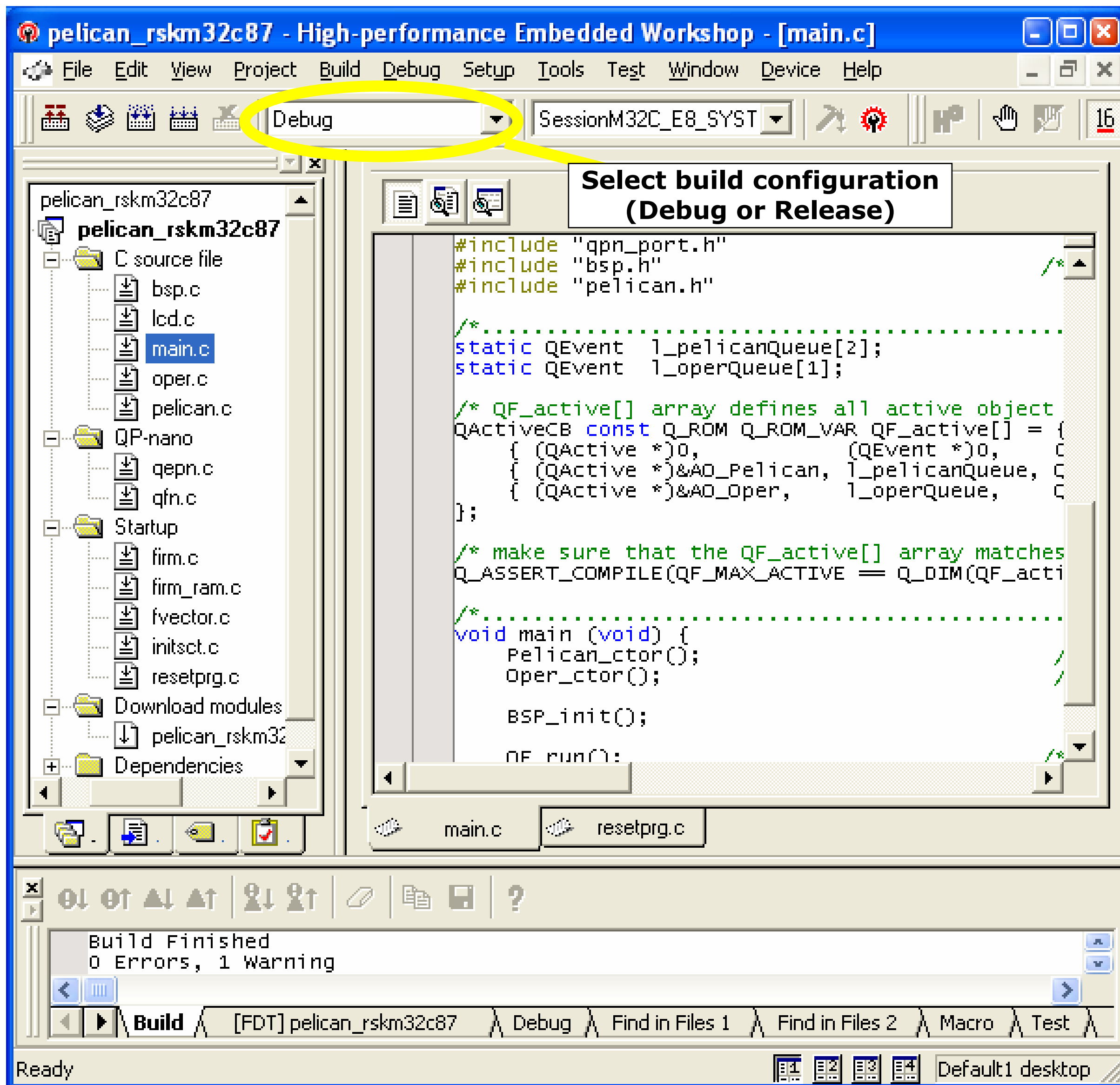



**Figure 2 HEW IDE with the pelican_rskm32c87 project**

Copyright © Quantum Leaps, LLC. All Rights Reserved. 

### 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.

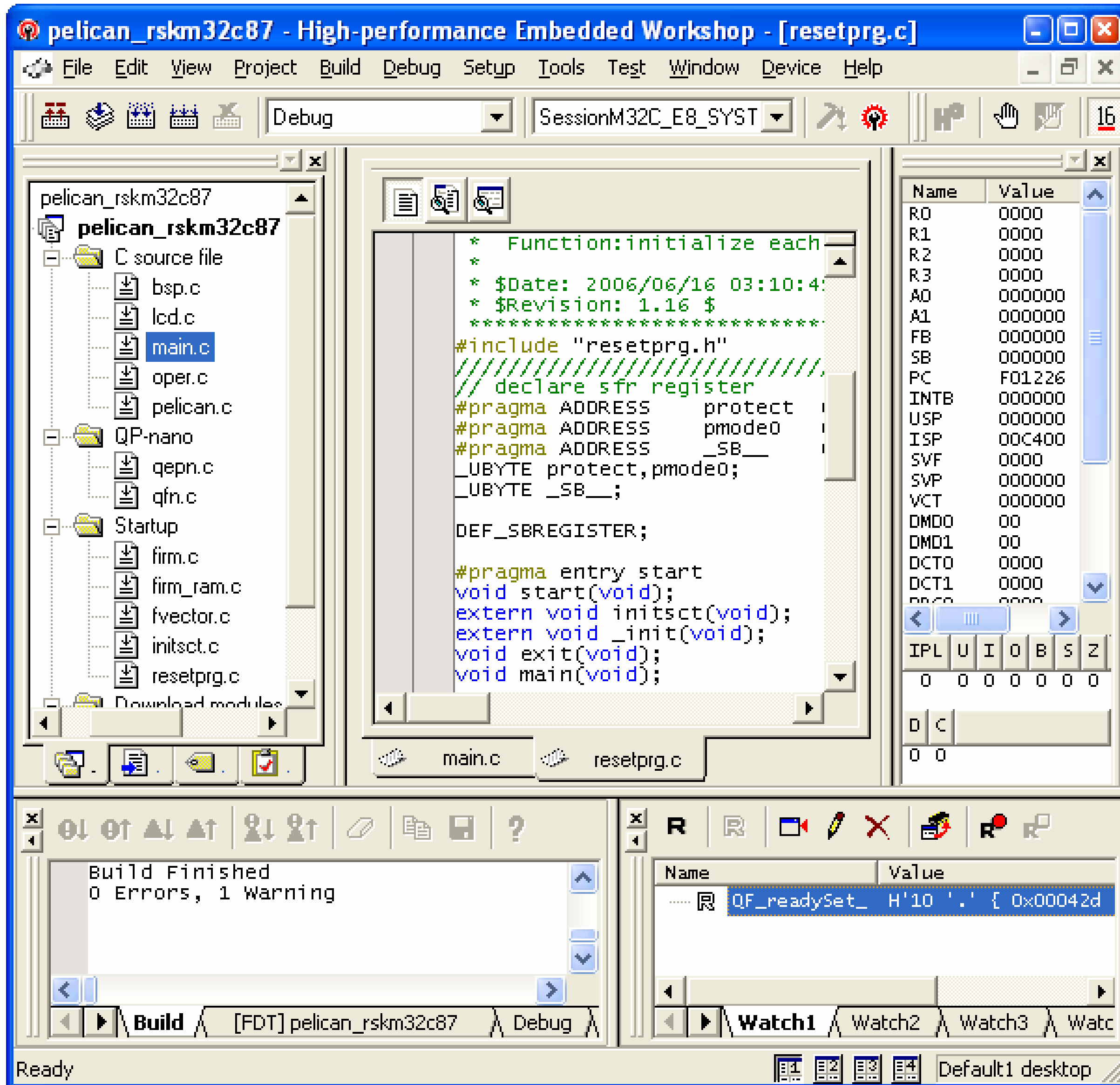



**Figure 3 HEW Debugger with various views of the executing PELICAN application.**

Figure 1 shows the PELICAN example running on the RSKM32C87 board. The operation of the PELICAN crossing is visualized by means of the LCD and 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 second line of the LCD is used to display the pedestrian signal ("DON'T WK" and "* WALK *" text). The first line of the LCD is used to display the name of the currently active state of the PELICAN state machine (see [QL AN-PELICAN 08]). **The traffic light is activated by pressing Switch SW1** (see Figure 1).

Copyright © Quantum Leaps, LLC. All Rights Reserved.

# 3 Non-Preemptive Configuration of QP-nano

The example of using QP-nano with the cooperative "Vanilla" kernel is located in the directory: `<qpn>\examples\m16c\nc308\pelican_rskm32c87\`. 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 M16C systems.

```
 (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()              _asm("FCLR I")
 (6) #define QF_INT_UNLOCK()            _asm("FSET I")

                         /* interrupt locking policy for interrupt level, see NOTE01 */
 (7) #define QF_ISR_NEST

        /* 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 "qepn.h"         /* QEP-nano platform-independent public interface */
(10) #include "qfn.h"           /* QF-nano platform-independent public interface */
```

**Listing 2 qpn_port.h header file for the non-preemptive QF-nano configuration and NC30 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.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

(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 M16C, the inline assembly instruction FCLR I clears the global interrupt enable flag. Conversely, the assembly instruction FSET I sets the global interrupt enable flag in the M16C flags register.

(7) Defining the macro QF_ISR_NEST configures QP-nano for interrupt nesting. This means that the QP-nano services for ISRs (QF_postISR() and QF_tick()) must be invoked with interrupts **unlocked** to avoid nesting of critical sections inside ISRs.

**NOTE:** The M16C microcontroller supports interrupt prioritization. Therefore it is safe to unlock interrupts inside ISRs without running the risk of priority inversions for interrupts. You have full control over interrupt nesting by configuring the priorities to interrupts. In particular, you can entirely prevent interrupt nesting by assigning the same priority level to all interrupts.

The simple policy of unconditional unlocking of interrupts upon exit from a critical section used in QP-nano precludes nesting of critical sections. This policy combined with QF_ISR_NEST macro means that you **must** unlock interrupts inside every ISR before invoking any QF-nano service, such as QF_postISR() or QF_tick().

(8) The Renesas NC30 compiler does not provide the C99-standard exact-width integer types, which are here defined using the typedef directives.

(9) The qpn_port.h must include the QEP-nano event processor interface qepn.h.

(10) The qpn_port.h 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) #pragma INTERRUPT ta0_isr (vect = 21)
(2) void ta0_isr(void) {                                        /* Timer A0 ISR */
(3)     QF_INT_UNLOCK();                                          /* see NOTE01 */

(4)     /* clear any level-sensitive interrupt sources, if necessary ... */

(5)     QF_tick();

        /* perform other work of the ISR, e.g., switch debouncing */

(6)     QF_INT_LOCK();                                            /* see NOTE01 */
    }
```

**Listing 3 Time tick interrupt for M16C 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 NC30 compiler for M16C interrupt must be declared with the #pragma INTERRUPT. 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.

Copyright © Quantum Leaps, LLC. All Rights Reserved. 

(3) The macro QF_INT_UNLOCK() **must** be called before invoking any QF services.

(4) The level-sensitive interrupt sources should be cleared right at the beginning of the ISR

**NOTE:** Because the M16C CPU prioritizes interrupts in hardware, it really does not matter at which point during the ISR processing the interrupt source is cleared. Even though the interrupts are unlocked at line (3), the CPU will not allow the same interrupt preempt itself.

(5) 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 M16C hardware performs interrupt prioritization and will not allow the same interrupt to preempt currently running interrupt.

(6) The macro QF_INT_UNLOCK() **must** be called before returning from the interrupt.

## 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.

M16C 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 M16C into the idle power-saving mode. Please note that M16C 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)     _asm("FSET I");              /* NOTE: the following WAIT instruction will */
(5)     _asm("WAIT");            /* execute before entering any pending interrupt */
    #else
(6)     QF_INT_UNLOCK();                                 /* just unlock interrupts */
    #endif
    }
```

**Listing 4 QF_onIdle() for the non-preemptive ("vanilla") QP-nano port to M32C.**

(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 FSET I instruction.

(5) The sleep mode is activated with the WAIT instruction.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

**NOTE:** As described in "M16C/60, M16C/20 Series Software Manual", Section 5.2.1 "Interrupt Enable Flag", the instruction immediately following the `FSET I` instruction, will be executed before any pending interrupt. This guarantees atomic transition to the WAIT mode.

**CAUTION:** The instruction pair (`FSET I, WAIT`) 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.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

# 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 `<qpn>\examples\m16c\-nc30\pelican_qk_rskm32c87`.

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)

```
    #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()           _asm("FCLR I")
    #define QF_INT_UNLOCK()         _asm("FSET I")

                    /* interrupt locking policy for interrupt level, see NOTE01 */
    #define QF_ISR_NEST

                                                       /* interrupt entry code */
(1) #define QK_ISR_ENTRY() do { \
(2)     ++QK_intNest_; \
(3)     QF_INT_UNLOCK(); \
    } while (0)

                                            /* interrupt exit code, see NOTE02 */
(4) #define QK_ISR_EXIT() do { \
(5)     asm("LDC #0,FLG"); \
(6)     --QK_intNest_; \
(7)     QK_SCHEDULE_(); \
    } while (0)


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

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

(1) The macro `QK_ISR_ENTRY()` is designed to be called upon the entry to an ISR programmed in C. The macro informs the QK-nano preemptive kernel about entering an interrupt.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

(2) The QK-nano interrupt nesting up-down counter QK_intNest_ is incremented to account for entering a new interrupt level. This informs the QK-nano scheduler about the interrupt context, so that if another interrupt preempts this one, the QK-nano scheduler will not be called.

(3) The interrupts are unlocked.

**NOTE:** The hardware interrupt entry sequence for M16C includes clearing the global interrupt enable flag in the FLAGS register, so that interrupts remain locked unless unlocked explicitly.

(4) The macro QK_ISR_EXIT() is designed to be called upon the exit from an ISR programmed in C. The macro informs the QK-nano preemptive kernel about exiting an interrupt.

(5) The flags register is set to zero, which accomplishes two things at the same time. First, the interrupts are locked again (the I flag is cleared). Second, the current interrupt priority level (IPL) of the M16C processor is forced to zero, which corresponds to the task-level priority. Setting the IPL to zero enables all interrupts and is effectively the End-Of-Interrupt instruction for the M16C architecture.

(6) The QK-nano interrupt nesting up-down counter QK_intNest_ is decremented to the level before entering the ISR.

(7) The QK-nano macro QK_SCHEDULE_() invokes the QK-nano scheduler to perform the asynchronous preemptions only if the interrupt level is zero and there are some events in the active object event queues.

(8) The QK-nano configuration is specified simply by including the "qkn.h" QK-nano interface in the qpn_port.h header file.

## 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.

```
#pragma INTERRUPT ta0_isr (vect = 12)              /* system clock tick ISR */
void ta0_isr(void) {
    static uint8_t btn_debounced  = 0;
    static uint8_t debounce_state = 0;
    uint8_t btn;

    QF_INT_UNLOCK();                                          /* see NOTE01 */

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

    btn = SW1;                                   /* read the user switch SW1 */
    switch (debounce_state) {
        case 0:
            if (btn != btn_debounced) {
                debounce_state = 1;         /* transition to the next state */
            }
            break;
        case 1:
            if (btn != btn_debounced) {
                debounce_state = 2;         /* transition to the next state */
            }
            else {
                debounce_state = 0;           /* transition back to state 0 */
            }
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

```
            break;
        case 2:
            if (btn != btn_debounced) {
                btn_debounced = btn;     /* save the debounced button value */

                if (btn == 0) {                  /* is the button depressed? */
                    QActive_postISR((QActive *)&AO_Pelican,
                                    PEDS_WAITING_SIG, 0);
                }
            }
            debounce_state = 0;               /* transition back to state 0 */
            break;
    }

    QF_INT_LOCK();                                           /* see NOTE01 */
}
```

**Listing 6 Time tick interrupt calling QF_tick() function and debouncing the SW1 switch.**

## 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 M16C.

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

**Listing 7 QK_onIdle() callback for M32C.**

Copyright © Quantum Leaps, LLC. All Rights Reserved.

# 5 Board Support Package

The Board Support Package (BSP) for M32C with the non-preemptive Vanilla kernel is located in the directory: `<qpn>\examples\m16c\nc308\pelican_rskm32c87\` and consists of the following files:

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

**NOTE:** This QDK uses the recommended by Renesas C-startup for M32C. The C-startup code is comprised of the following files (see also Listing 1):

1. `cregdef.h`
2. `firm.c`
3. `firm_ram.c`
4. `fvector.c`
5. `heap.c`
5. `heapdef.c`
6. `initsct.c`
7. `initsct.h`
8. `intprg.c`
9. `resetprg.c`
10. `resetprg.h`
11. `rskm32c87.h`
12. `sfr32c87.h`

## 5.1 Compiler Options Used

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

```
-c -g -Wall -silent
```

The –g 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 M32C Standard Toolchain..., or by right-clicking on any of the C modules in the Workspace window and choosing the Build Options/Renesas M32C Standard Toolchain... pop-up menu.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

## 5.2 Linker Options Used

### 5.2.1 Specifying Program Sections

The HEW IDE allows to specify very precisely all the program sections, as shown in the screen shot in Figure 4. For the RSKM32C87 board, the sections are configured according to the memory map included in Section 6.3 of the "RSKM32C87 User's Manual" [Renesas RSKM32C87].

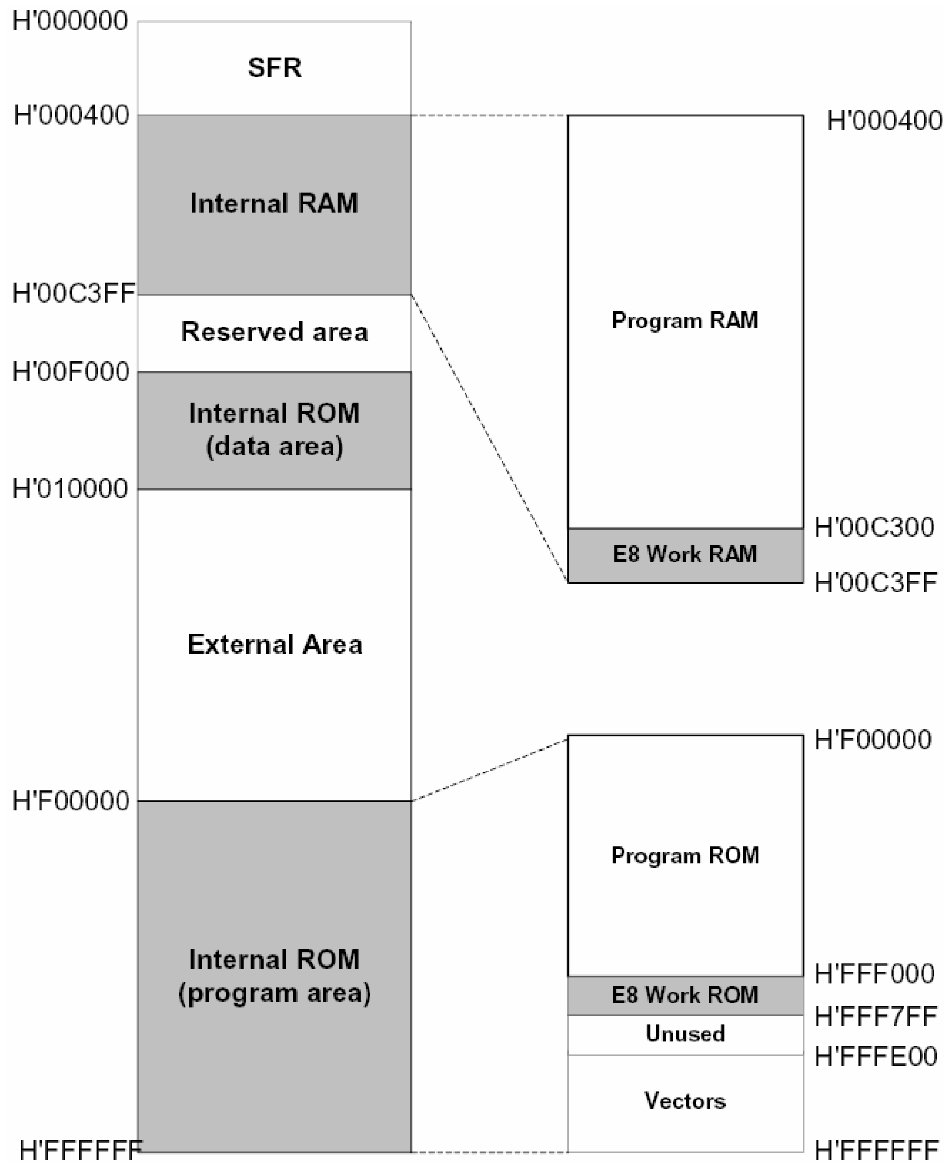



**Figure 4 Memory map of the M32C87 device with the "E8 Work RAM and ROM" (Renesas ROM monitor program).**

Copyright © Quantum Leaps, LLC. All Rights Reserved.

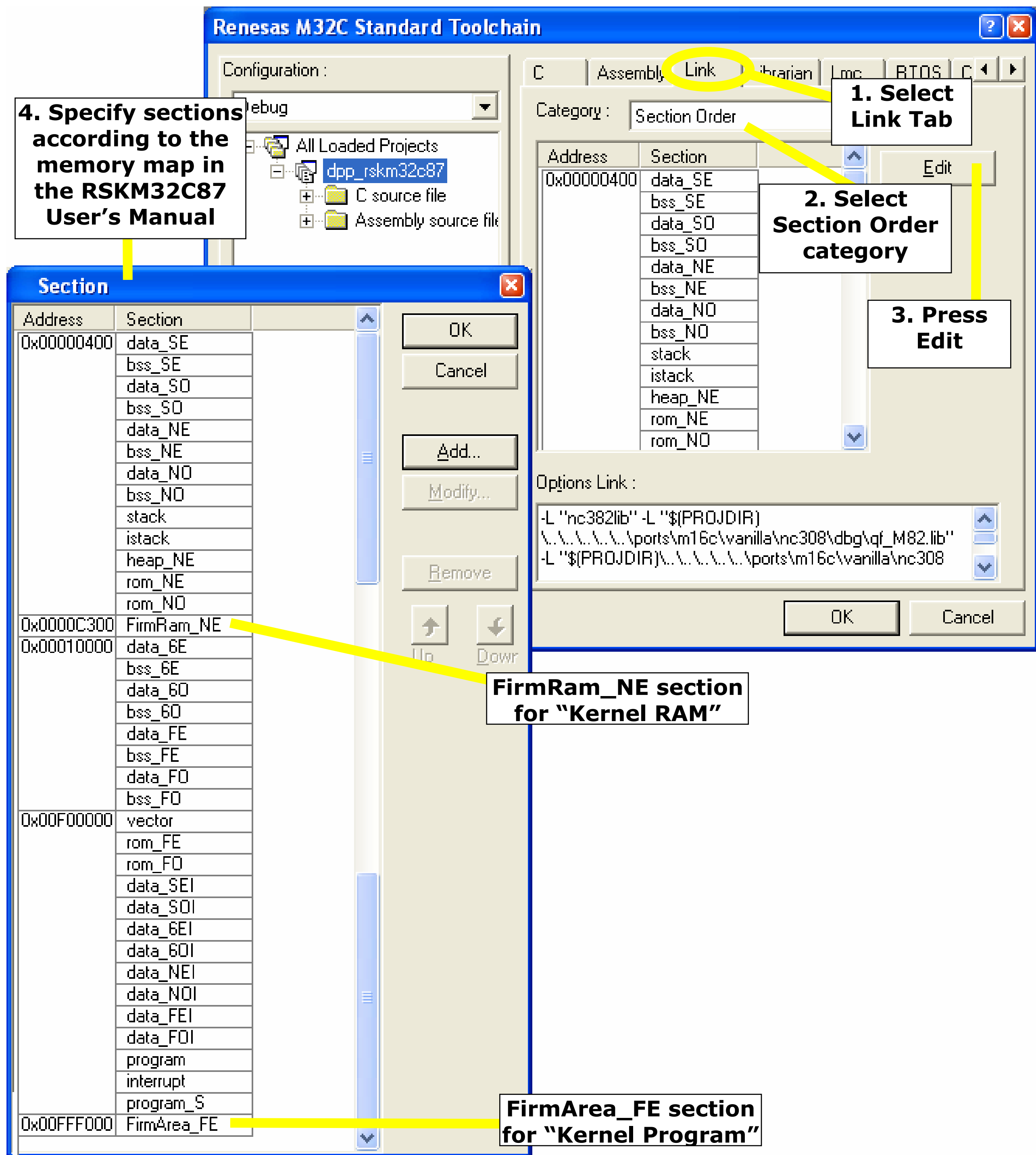



**Figure 5 Specifying the sections names and addresses in the HEW IDE.**

In particular, the Section Order dialog box shows the special RAM section **FirmRAM_NE** for the "E8 Work RAM" located at 0x0C300 and the ROM section **FirmArea_FE** for "E8 Work ROM" at 0xFFF000. "Kernel" in the Renesas terminology means ROM-monitor resident in the target and communicating with the HEW debugger via the E8 In-Circuit Debugger.

Copyright © Quantum Leaps, LLC. All Rights Reserved.

### 5.2.2 Specifying Stack and Heap Sizes

This QDK-nano-M32C uses only the **Interrupt Stack** both for task-level and interrupt-level processing. In particular, the User Stack is **not** used. You need to declare adequate stack size for the Interrupt Stack. You specify the interrupt stack size via the `__ISTACKSIZE__` preprocessor macro, which you define using the –D compiler option.

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

The provided example does not use the heap, because using `malloc()` and `free()` in a real-time, concurrent system is typically a **bad idea**. However, if you decide to use the heap, you need to define the macro `__HEAP__` using the –D compiler option. You also need to include the file `heap.c` in the build and define the size of the heap via the __HEAPSIZE__ macro (also defined by –D compiler option).

## 5.3 The BSP header file bsp.h

```
(1) #include "sfr32c87.h"             /* special function registers for M32C87 */
(2) #include "rskm32c87def.h"                        /* RSKM32C87 board interface */

    /*-----------------------------------------------------------------------*/
(3) #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
    };

    /* BSP services ...........................................................*/
    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);
```

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

(1) The header file "`sfr32c87.h`" provides the definitions of the M32C/87 special function registers.

(2) The header file "`rskm32c87def.h`" defines the interface to the RSKM32C87 board.

(3) The BSP defines the desired ticking rate. This constant is useful for defining timeouts, which are always specified in units of clock ticks.

## 5.4 BSP initialization

The following `BSP_init()` function from the PELICAN crossing application for the RSKM32C87 board configures the PIO lines for the User LEDs and the User switches, initializes the LCD, and the system clock tick:

Copyright © Quantum Leaps, LLC. All Rights Reserved.

```
void BSP_init(void) {
    uint16_t volatile delay;
                                                 /* initialize the clock... */

    prc0 = 1;                  /* allow writing to clock control registers */
    cm07 = 0;    /* clock selected by cm21 in CM2 (oscillation stop detect) */
    cm21 = 0;    /* clock selected by cm17 in CM1 (system clock select reg) */
    cm17 = 0;                                /* clock source is main clock */
    mcd  = 0x12;                       /* set divide for BCLK to divide by 1 */

                              /* configure and switch main clock to PLL... */
    /* set PLL to multiply by 6 and divide by 2 giving 30MHz when writing PLL
    * control registers write to both plc0 and plc1 (addresses 0026 and 0027)
    * using a word write command. Set divide ratio with the PLL off
    */
    *(uint16_t *)&plc0 = 0x0253;
    _asm("nop");
    *(uint16_t *)&plc0 = 0x02D3;                          /* turn the PLL on */
    for (delay = 0; delay < 0x4000; ++delay) {           /* delay for 20msec */
    }
    cm17 = 1;                          /*  make the PLL the CPU clock source */
    prc0 = 0;                             /* protect clock control registers */


    pd6 |= 0x0F;   /* port 6 is used by E8 do not modify upper half of port */

                                                   /* enable the User Button */
    SW1_DDR = 0;

                                                /* LED port configuration... */
    LED0_DDR = 1;
    LED1_DDR = 1;
    LED2_DDR = 1;
    LED3_DDR = 1;
    LED0     = LED_OFF;
    LED1     = LED_OFF;
    LED2     = LED_OFF;
    LED3     = LED_OFF;

                                              /* Configure the LCD module... */

    p2  = 0;
    pd2 = 0xFF;
    InitialiseDisplay();

    /* start the 32kHz crystal subclock (remove if 32kHz clock not used)... */
    prc0  = 1;    /* unlock CM0 and CM1 and set GPIO to inputs (XCin/XCout) */
    pd8_7 = 0;
    pd8_6 = 0;
    cm04  = 1;                                     /* start the 32kHz crystal */

                                       /* setup Timer A running from fc32... */
    ta0mr = 0xC0;                       /* Timer mode, fc32, no pulse output */
    ta0   = (int)((fc_CLK_SPEED/32 + BSP_TICKS_PER_SEC/2)
              / BSP_TICKS_PER_SEC) - 1;                              /* period */
    ta0ic = TICK_ISR_PRIO;   /* set the clock tick interrupt priority level */
}
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

## 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) {
    ta0s = 1;                                         /* Start timer A0 */
}
```

**Listing 9 Configuring and enabling interrupts in the `QF_onStartup()` callback.**

## 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 for M16C. 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) {
    (void)file;                                  /* avoid compiler warning */
    (void)line;                                  /* avoid compiler warning */
    QF_INT_LOCK();                                  /* lock the interrupts */
    for (;;) {                                /* hang in this for-ever loop */
    }
}
```

Copyright © Quantum Leaps, LLC. All Rights Reserved.

# 6 Related Documents and References

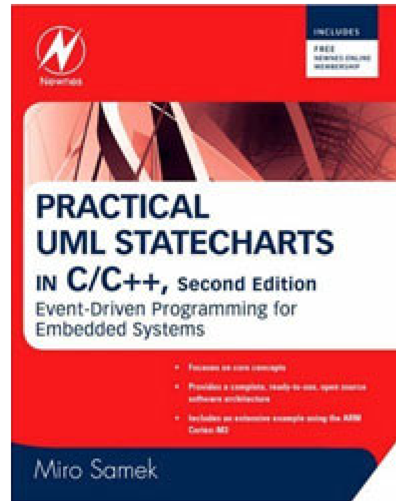

| Document | Location |
|---|---|
| [PSiCC2] "Practical UML Statecharts in C/C++, Second Edition", Miro Samek, Newnes, 2008 | Available from most online book retailers, such as amazon.com. See also: http://www.quantum-leaps.com/psicc2.htm |
| [QP-nano 08] "QP-nano Reference Manual", Quantum Leaps, LLC, 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 |
| [M32C/87 Hw] "M32C/87 Group (M32C/87, M32C/87A, M32C/87B) Hardware Manual", Renesas 2006 | Renesas document REJ09B0180-0100, available online at www.renesas.com |
| [M16C Sw] "M16C/60, M16C/20 Series Software Manual", Renesas, 2003. | Document included in PDF with the RSKM16C26A Plus kit. |
| [NC308] "M32C/90,80,M16C/80,70 Series C Compiler Package V.5.41 C Compiler User's Manual", Renesas 2006 | Renesas document REJ10J1451-0200, available online at www.renesas.com |
| [Renesas RSKM32C87] "Renesas Starter KitRSKM32C87 User's Manual", Reneas 2006 | Document REG10J0010 included in PDF with RSKM32C87 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 |


Copyright © Quantum Leaps, LLC. All Rights Reserved.

# 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*", by Miro Samek, Newnes, 2008

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

Copyright © Quantum Leaps, LLC. All Rights Reserved.