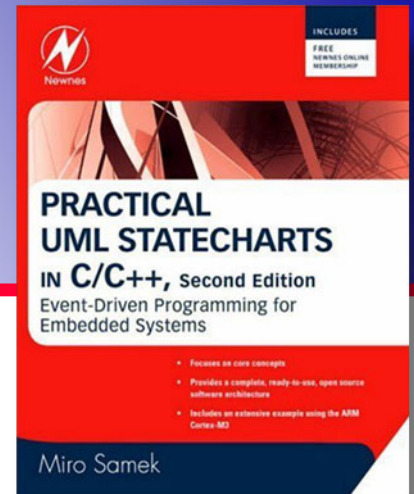


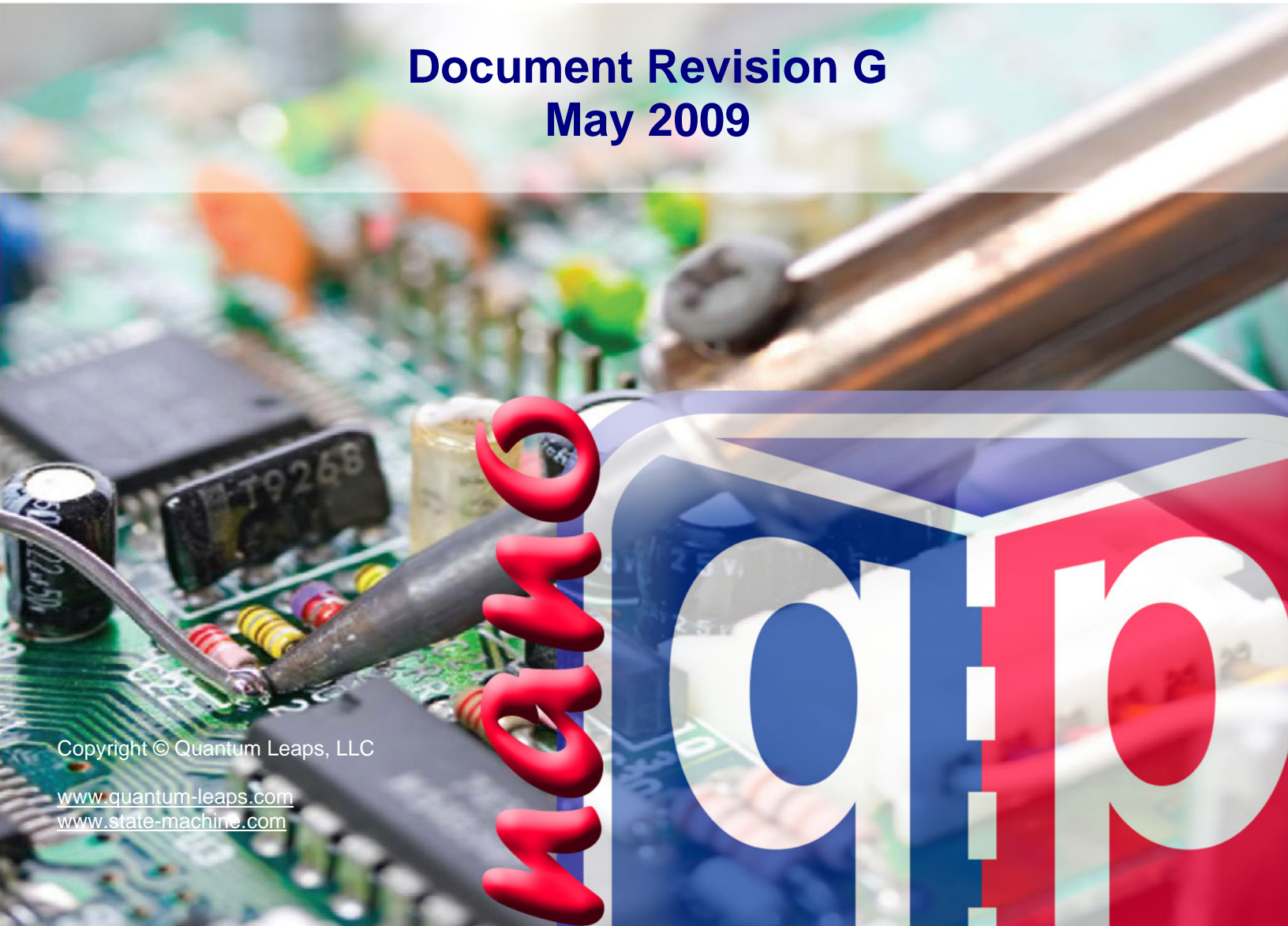


**Quantum™ Leaps**  
innovating embedded systems



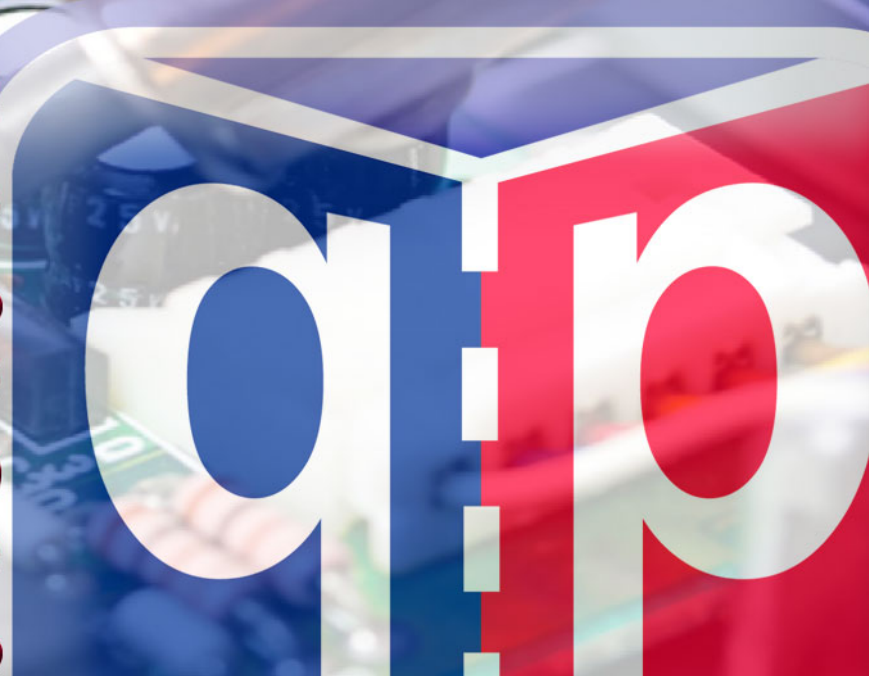
# QDK™-nano Cypress PSoC-ICCM8C

Document Revision G  
May 2009



Copyright © Quantum Leaps, LLC

[www.quantum-leaps.com](http://www.quantum-leaps.com)  
[www.state-machine.com](http://www.state-machine.com)



# Table of Contents

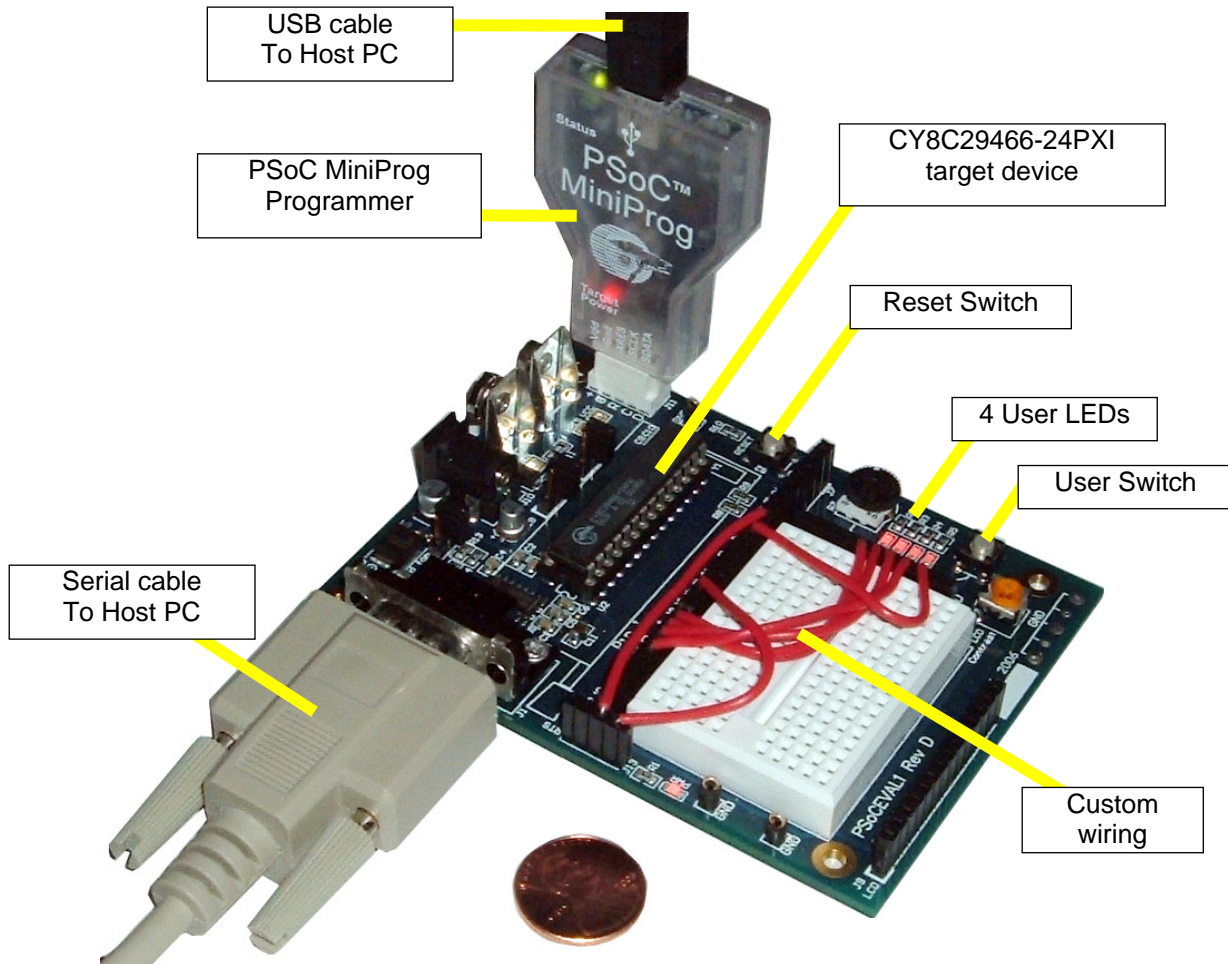
<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	About QP-nano™	2
1.2	What's Included in the QDKn-PSoC-ICCM8C?	3
1.3	Licensing QP-nano™	3
<b>2</b>	<b>Getting Started</b>	<b>4</b>
2.1	Software Installation	4
2.2	Building and Running the Examples	5
2.2.1	Custom Wiring of the PSoCEVAL1 Board	5
2.2.2	Building the Example Projects in the PSoC Designer	6
2.2.3	Programming the PSoC	6
2.2.4	Running the QHsmTst Example	7
2.2.5	Running the PELICAN Crossing Examples	8
<b>3</b>	<b>Non-Preemptive Configuration of QP-nano</b>	<b>9</b>
3.1	The qpn_port.h Header File)	9
3.2	ISRs in the Non-Preemptive "Vanilla" Configuration	10
3.3	QP Idle Loop Customization in QF_onIdle()	11
<b>4</b>	<b>Preemptive Configuration with QK-nano</b>	<b>13</b>
4.1	ISRs in the Preemptive Configuration with QK-nano	14
4.2	Idle Loop Customization in the QK Port	15
<b>5</b>	<b>BSP for PSoCEVAL1 Board</b>	<b>16</b>
5.1	Compiler Options Used	16
5.2	The BSP header file bsp.h	16
5.3	BSP initialization	17
5.4	Starting Interrupts in QF_onStart()	17
5.5	Placing the ISRs in the Vector Table (Boot.tpl Template)	17
5.6	Assertion Handling Policy in Q_onAssert()	18
<b>6</b>	<b>Related Documents and References</b>	<b>19</b>
<b>7</b>	<b>Contact Information</b>	<b>20</b>



# 1 Introduction

This **QP-nano Development Kit™** (QDK-nano) describes how to use QP-nano™ with the Cypress Programmable System on Chip (PSoC™) and the ImageCraft C M8C compiler (ICCM8C).

**Figure 1 Cypress PSoCEVAL1 Kit with the MiniProg USB programmer.**



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

1. Cypress PSoCEVAL1 evaluation kit
2. Cypress MiniProg USB programmer
3. Cypress PSoC Designer® version 4.4 with the ImageCraft M8C compiler ICCM8C v1.65
4. QP-nano 4.0.02 or higher

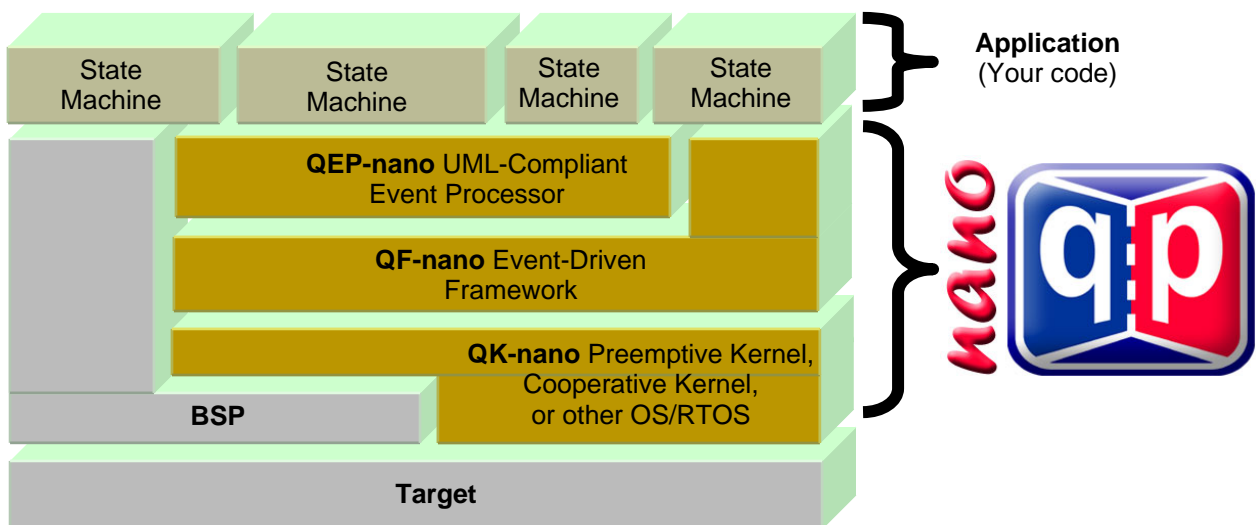
As shown in [Figure 1](#), the Cypress PSoCEVAL1 Kit contains the PSoCEVAL1 board, the 28-pin socket and the MiniProg programmer that connects directly to the host development workstation via the provided USB cable. The USB connection also powers up the board, so no additional power is required. This QDK has been tested with the with the CY8C29466-24PXI PSoC mixed-signal array device with 2KB of RAM and 32KB of onboard flash. However, the described QP-nano port should be applicable to almost all PSoC devices.

## 1.1 About QP-nano™

**QP-nano™** is an ultra-lightweight, open source, state machine framework and RTOS for developing real-time embedded applications. QP-nano has been specifically designed to enable event-driven programming with concurrent hierarchical state machines (UML statecharts) on low-end 8- and 16-bit single-chip MCUs and DSPs, such as **Cypress PSoC**, 8051, PICmicro, AVR, MSP430, 68HC08/11/12, R8C/Tiny, H8/S, TMS320C28x and others alike, with a few hundred bytes of RAM and a few kilobytes of ROM. With QP-nano, coding of modern state machines directly in C is a non-issue. No big design automation tools are needed.

As shown in [Figure 2](#), QP-nano consists of a universal UML-compliant event processor (QEP-nano), a highly portable event-driven framework (QF-nano), and a tiny run-to-completion kernel (QK-nano).

**Figure 2 QP-nano components and their relationship with the target hardware, board support package (BSP), and the application comprised of state machines**

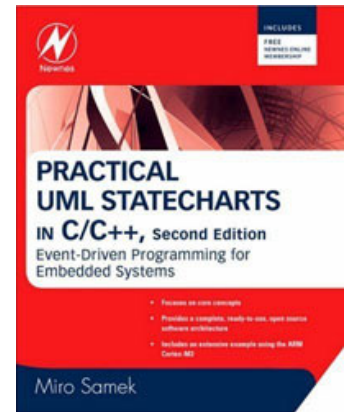


The QP-nano framework can manage up to 8 concurrently executing hierarchical state machines and requires only 1-2KB of code (ROM) and just several bytes of RAM. This tiny footprint, especially in RAM, makes QP-nano ideal for high volume, cost-sensitive applications, such as motor control, lighting control, capacitive touch sensing, remote access control, RFID, thermostats, small appliances, toys, power

supplies, battery chargers, or just about any **system-on-a-chip** (SoC or ASIC) that contains a small processor inside.

Also, because the event-driven paradigm naturally uses the CPU only when handling events and otherwise can very easily switch the CPU into a low-power sleep mode, QP-nano is particularly suitable for ultra-low power applications, such as wireless sensor networks or implantable medical devices.

All versions of QP, including QP-nano, are described in detail in the book "*Practical UML Statecharts in C/C++, 2nd Edition: Event-Driven Programming for Embedded Systems*" [PSiCC2] published by Newnes in 2008 (see [www.state-machine.com/psicc2](http://www.state-machine.com/psicc2)). QP-nano has a strong user community and has been applied worldwide in industries, such as: consumer electronics, telecommunications, equipment, industrial automation, transportation systems, medical devices, and many more. Please refer to the [www.state-machine.com](http://www.state-machine.com) website for more information.



## 1.2 What's Included in the QDKn-PSoC-ICCM8C?

This QDK-nano provides the QP-nano, a basic Board Support Package (BSP) for PSoC, and to example applications:

- The QHsmTst exhaustive test of the QEP-nano event processor described in Chapter 2 of [PSiCC2];
- The PEdestrian Light CONtrolled (PELICAN) crossing example with the cooperative “vanilla” kernel described in Chapter 12 of [PSiCC2] as well as in the Application Note “PELICAN Crossing Example” (included in the QDK distribution); and
- The PEdestrian Light CONtrolled (PELICAN) crossing example with the preemptive QK-nano kernel

## 1.3 Licensing QP-nano™

The **Generally Available (GA)** distribution of QP-nano™ available for download from the [www.state-machine.com/downloads](http://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](http://www.state-machine.com/licensing).

## 2 Getting Started

This section describes how to install, build, and use QDKn-PSoC-ICCM8C.

---

**NOTE:** Typically, a QDK-nano™ contains only example(s) pertaining to the specific CPU and compiler, but does not include the platform-independent baseline code of QP-nano™. However, the PSoC Designer® tool made it difficult to preserve the separation between the generic QP-nano baseline code and the target-specific code. The example project included in the QDKn-PSoC-ICCM8C contains the QP-nano source code (files `qepn.h`, `qfn.h`, `qassert.h`, `qepn.c`, `qfn.c`) in the project directory. You will need to manually replace all these files to upgrade to a newer version of QP-nano.

---

### 2.1 Software Installation

The QDK-nano code is distributed in a ZIP archive (`qdkn_psoc-iccm8c_<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 `<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](#)”).

**Listing 1 Selected QP directories and files after installing QDKn-PSoC-ICCM8C**

```

<qpn>/                                - QP-nano Root Directory
|
|--doc\
|  |--AN_PELICAN.pdf                  - Application Note "PELICAN crossing example"
|  |--QDKn_PSoC-ICCM8C.pdf           - This QDK Manual "QDK-nano PSoC-ICCM8C"
|
|--examples\                          - subdirectory containing the QP-nano example files
|  |--psoc\                           - examples for PSoC
|  |  |--iccm8c\                       - examples compiled with the ImageCraft ICCM8C compiler
|  |  |  |--qhsmtst_psoceval1\         - QHsmTst example for PSoCEVAL1 (non-preemptive)
|  |  |  |  |--output\                - directory containing the hex file
|  |  |  |  |  |--qhsmtst_psoceval1.hex - image of the application
|  |  |  |  |  |--qhsmtst_psoceval1.mp - map file of the application
|  |  |  |--bsp.c                     - Board Support Package for the Toolstick (non-preemptive)
|  |  |  |--bsp.h                     - BSP header file
|  |  |  |--main.c                    - the main function
|  |  |  |--qhsmtst.c                 - the QHsmTst state machine
|  |  |  |--qhsmtst.h                 - the QHsmTst application header file
|  |  |  |--qhsmtst_psoceval1.SOC     - PSoC Designer project for the application
|  |  |  |--boot.tpl                  - bootstrap code template
|  |  |  |--qpn_port.h                - QP-nano configuration for this application
|  |  |  |--qassert.h                 - embedded-systems-friendly assertions used in QP-nano
|  |  |  |--qepn.h                    - The platform-independent QEP-nano header file
|  |  |  |--qepn.c                    - QEP-nano implementation
|  |  |--pelican_psoceval1\          - PELICAN example for PSoCEVAL1 (non-preemptive)
|  |  |  |--output\                  - directory containing the hex file
|  |  |  |  |--pelican_psoceval1.hex - image of the application

```

```

| | | | +-pelican_psocevall.mp - map file of the application
| | | |
| | | | +-bsp.c - Board Support Package for the Toolstick (non-preemptive)
| | | | +-bsp.h - BSP header file
| | | | +-main.c - the main function
| | | | +-pelican.c - the Pelican state machine
| | | | +-pelican.h - the Pelican application header file
| | | | +-oper.c - the Operator state machine
| | | | +-pelican_psocevall.SOC - PSoC Designer project for the application
| | | | +-boot.tpl - bootstrap code template
| | | | +-qpn_port.h - QP-nano configuration for this application
| | | |
| | | | +-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
| | | | +-qepn.c - QEP-nano implementation
| | | | +-qfn.c - QF-nano implementation
| | | |
| | | | +-pelican_qk_psocevall\ - PELICAN example for PSoCEVAL1 with QK-nano
| | | | +-output\ - directory containing the hex file
| | | | | +-pelican_qk_psocevall.hex - image of the application
| | | | | +-pelican_qk_psocevall.mp - map file of the application
| | | |
| | | | +-bsp.c - Board Support Package for the Toolstick (non-preemptive)
| | | | +-bsp.h - BSP header file
| | | | +-main.c - the main function
| | | | +-pelican.c - the Pelican state machine
| | | | +-pelican.h - the Pelican application header file
| | | | +-oper.c - the Operator state machine
| | | | +-pelican_qk_psocevall.SOC - PSoC Designer project for the application
| | | | +-boot.tpl - bootstrap code template
| | | | +-qpn_port.h - QP-nano configuration for this application
| | | |
| | | | +-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
| | | | +-qepn.c - QEP-nano implementation
| | | | +-qfn.c - QF-nano implementation
| | | | +-qkn.c - QK-nano implementation

```

## 2.2 Building and Running the Examples

The examples included in this QDK-nano are based on the standard QHsmTst application for testing the hierarchical state machines as well as the PELICAN crossing application implemented with active objects (see Quantum Leaps Application Note: “PELICAN Crossing Application” [QL AN-PELICAN 08] included in this QDK-nano). The example directory contains the PSoC Designer workspaces that you can load into the Designer IDE.

### 2.2.1 Custom Wiring of the PSoCEVAL1 Board

The example applications require the following custom wiring of the PSoCEVAL1 board (see [Figure 1](#)).

```

P00 -> LED1
P01 -> LED2
P02 -> LED3

```

P03 -> LED4  
P10 -> SW  
P16 -> RX = Serial RX  
P27 -> TX = Serial TX

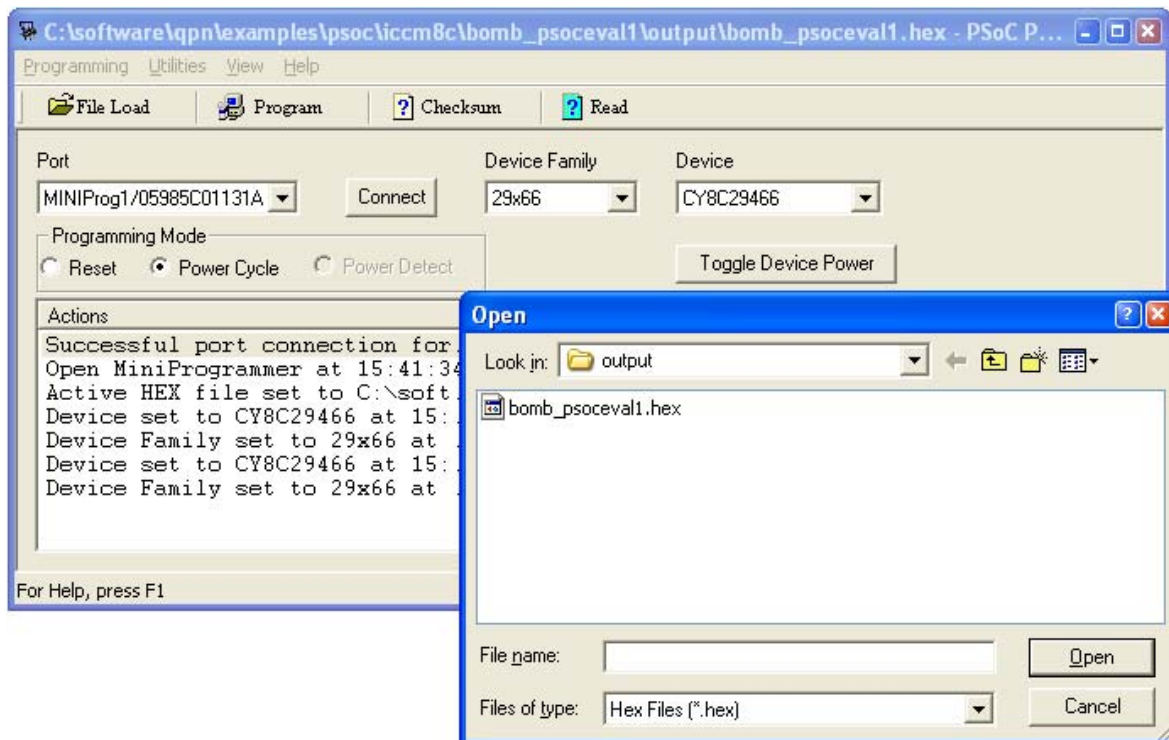
## 2.2.2 Building the Example Projects in the PSoC Designer

1. Connect the PSoC MiniProg programmer to the PSoCEVAL1 board to the J11 connector.
2. Connect the PSoC MiniProg programmer to the PC with the provided USB cable.
3. Launch PSoC Designer and open the project qhsmtst\_psoceval1.SOC (located in <qpn>\examples\psoc\iccm8c\qhsmtst\_psoceval1\).
4. Build the project by select Build->Build menu or by pressing F7.
5. Repeat steps 1-4 for the PELICAN crossing examples (see [Listing 1](#)).

## 2.2.3 Programming the PSoC

The PSoC Designer can launch the PSoC Programmer application, which can program the generated hex file to PSoC device via the PSoC MiniProg programmer. To launch the PSoC Programmer, select Program | Program part... menu. In the PSoC Programmer, you click the File Load button to browse for the hex file of the application (see [Figure 3](#)). After selecting the file, you click on Program button. You need to wait until the PSoC Programmer tool indicates that the process has completed, which may take a few minutes.

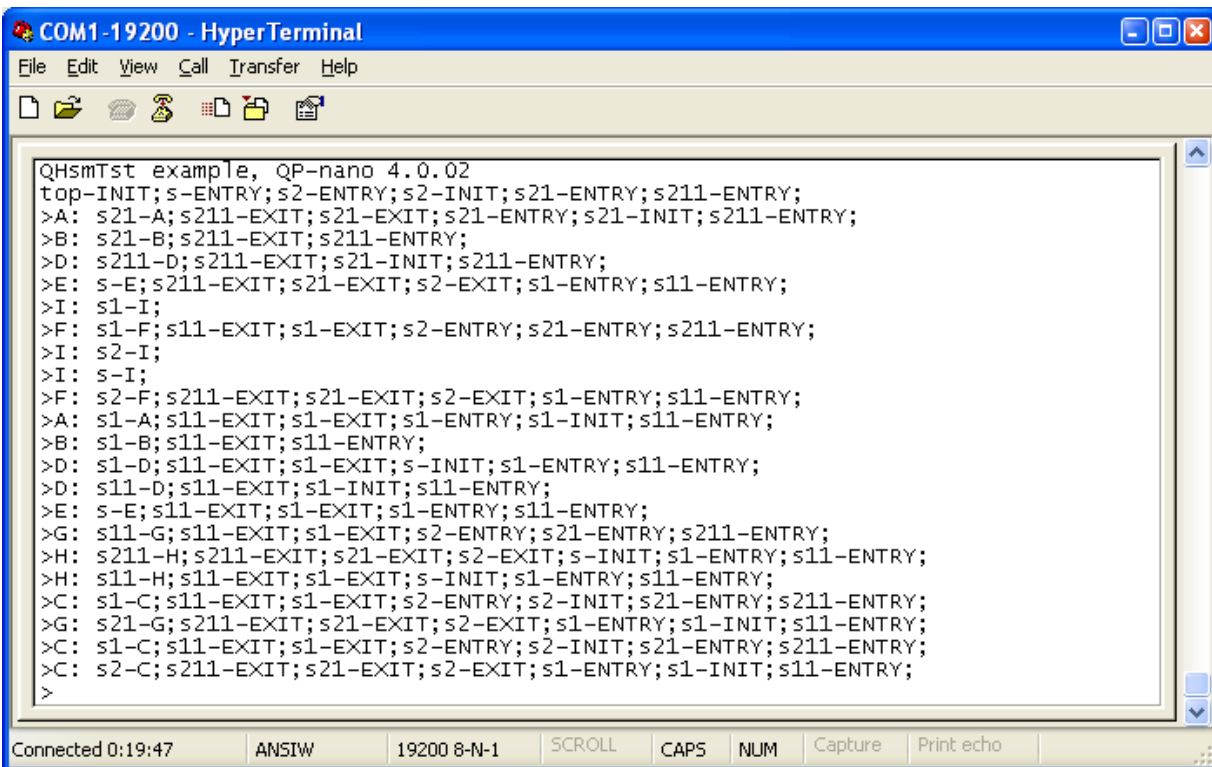
**Figure 3 Loading the Bomb example application with the PSoC Programmer.**





The QHsmTst application for PSoCEVAL1 board works exactly as described in Chapter 2 of [PSiCC2], except it uses the HyperTerminal to print to the screen and receive the events generated from the keyboard. The HyperTerminal is connected to the PSoCEVAL1 board via a straight serial cable, as shown in Figure 1. The HyperTerminal must be configured to use the COM port connected to the board, 19200 baud rate, no parity, 1 stop bit. Figure 6 shows an example HyperTerminal output generated by the QHsmTst application. This screen shot corresponds exactly to the expected output found in the file log\_pass.txt.

**Figure 6 HyperTerminal output from the QHsmTst application**



```

QHsmTst example, QP-nano 4.0.02
top-INIT; s-ENTRY; s2-ENTRY; s2-INIT; s21-ENTRY; s211-ENTRY;
>A: s21-A; s211-EXIT; s21-EXIT; s21-ENTRY; s21-INIT; s211-ENTRY;
>B: s21-B; s211-EXIT; s211-ENTRY;
>D: s211-D; s211-EXIT; s21-INIT; s211-ENTRY;
>E: s-E; s211-EXIT; s21-EXIT; s2-EXIT; s1-ENTRY; s11-ENTRY;
>I: s1-I;
>F: s1-F; s11-EXIT; s1-EXIT; s2-ENTRY; s21-ENTRY; s211-ENTRY;
>I: s2-I;
>I: s-I;
>F: s2-F; s211-EXIT; s21-EXIT; s2-EXIT; s1-ENTRY; s11-ENTRY;
>A: s1-A; s11-EXIT; s1-EXIT; s1-ENTRY; s1-INIT; s11-ENTRY;
>B: s1-B; s11-EXIT; s11-ENTRY;
>D: s1-D; s11-EXIT; s1-EXIT; s-INIT; s1-ENTRY; s11-ENTRY;
>D: s11-D; s11-EXIT; s1-INIT; s11-ENTRY;
>E: s-E; s11-EXIT; s1-EXIT; s1-ENTRY; s11-ENTRY;
>G: s11-G; s11-EXIT; s1-EXIT; s2-ENTRY; s21-ENTRY; s211-ENTRY;
>H: s211-H; s211-EXIT; s21-EXIT; s2-EXIT; s-INIT; s1-ENTRY; s11-ENTRY;
>H: s11-H; s11-EXIT; s1-EXIT; s-INIT; s1-ENTRY; s11-ENTRY;
>C: s1-C; s11-EXIT; s1-EXIT; s2-ENTRY; s2-INIT; s21-ENTRY; s211-ENTRY;
>G: s21-G; s211-EXIT; s21-EXIT; s2-EXIT; s1-ENTRY; s1-INIT; s11-ENTRY;
>C: s1-C; s11-EXIT; s1-EXIT; s2-ENTRY; s2-INIT; s21-ENTRY; s211-ENTRY;
>C: s2-C; s211-EXIT; s21-EXIT; s2-EXIT; s1-ENTRY; s1-INIT; s11-ENTRY;
>
  
```

### 2.2.5 Running the PELICAN Crossing Examples

The PELICAN Crossing examples are found in <qp>\examples\psoc\iccm8c\pelican\_psoceval1\ and <qp>\examples\psoc\iccm8c\pelican\_qk\_psoceval1\ for cooperative and preemptive QP-nano configurations, respectively. Both examples behave identically. The PEDS\_WAITING event is generated by pressing the SW button on the PSoCEVAL1 board. The LED1 corresponds to the signal for pedestrians (LED1=on corresponds to the “DON’T WALK” signal) and LED2 to light signals for cars (LED2=on when cars get the green light). LED3 visualizes the system clock tick activity (LED3 is inverted on every clock tick). Finally, LED4 visualizes the idle loop activity (the intensity of the LED4 corresponds to higher idle loop rate).

Every once in a while the PELICAN crossing is put into the “offline” state, in which pedestrians get the flashing “DON’T WALK” signal (LED1 blinks on and off) and the cars get flashing green signal (LED2 blinks on and off).

### 3 Non-Preemptive Configuration of QP-nano

The example of using QP-nano with the cooperative “Vanilla” kernel is located in the directory: <qpnan>\-examples\psoc\iccm8c\pelican\_psocevall\. This section describes the generic QP-nano configuration, which consist of the `qpnan_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 4.

#### 3.1 The `qpnan_port.h` Header File)

You configure and customize QP-nano through the header file `qpnan_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 `qpnan_port.h` is generic and should not need to change (except for the `QF_MAX_ACTIVE` definition) for other PSoC applications.

---

#### Listing 2 `qpnan_port.h` header file for the non-preemptive QP-nano configuration and ICCM8C compiler

```

#ifndef qpnan_port_h
#define qpnan_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 task level */
(5) #define QF_INT_LOCK()        M8C_DisableGInt
(6) #define QF_INT_UNLOCK()     M8C_EnableGInt

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

    /* exact-width integers (ImageCraft C compiler does NOT provide <stdint.h>) */
(8) typedef unsigned char  uint8_t;
    typedef signed   char  int8_t;
    typedef unsigned int   uint16_t;
    typedef signed   int   int16_t;
    typedef unsigned long  uint32_t;
    typedef signed   long  int32_t;

(9) #include <m8c.h>                /* M8C-specific constants and macros */

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

#endif                            /* qpnan_port_h */

```

(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]).
- (7) This QP-nano port to M8C does **not** allow nesting of interrupts (the macro `QF_ISR_NEST` is **not** defined).

The M8C CPU does not provide any support for prioritizing interrupts in hardware. Therefore, unlocking interrupts inside ISRs (the M8C 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 C99-standard exact-width integer types are defined explicitly, because the ICCM8C compiler does not provide the standard `<stdint.h>` header file.
- (9) This header file defines M8C-specific constants and macros, such as `M8C_DisableGInt/`  
`M8C_EnableGInt`.
- (10) The `qpn_port.h` must include the QEP-nano event processor interface `qepn.h`.
- (11) 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 ICCM8C compiler supports writing interrupts in C. In the “vanilla” port, the ISRs are identical as in the simplest of all “superloop” (main+ISRs), and there is nothing QP-specific in the structure of the ISRs. The only QP-specific requirement is that you provide a periodic time-tick ISR and you invoke `QF_tick()` in it.

**Listing 3 Time tick interrupt calling `QF_tick()` function to manage armed time events.**

```
(1) #pragma interrupt_handler BSP_tick
(2) void BSP_tick(void) {
    static uint8_t sw_debounced;
    static uint8_t debounce_state;
    uint8_t sw;

    LED_3_Invert(); /* toggle the LED */

(3)    QF_tick(); /* process armed timers */

(4)    sw = (PRT1DR & SW_1); /* read the switch */
    switch (debounce_state) { /* switch debounce state machine */
```

```

case 0:
    if (sw != sw_debounced) {
        debounce_state = 1;          /* transition to the next state */
    }
    break;
case 1:
    if (sw != sw_debounced) {
        debounce_state = 2;          /* transition to the next state */
    }
    else {
        debounce_state = 0;          /* transition back to state 0 */
    }
    break;
case 2:
    if (sw != sw_debounced) {
        sw_debounced = sw;          /* save the debounced switch value */

        if (sw == SW_1) {            /* is the button depressed? */
            QActive_postISR((QActive *)&AO_Pelican,
                PEDS_WAITING_SIG, 0);
        }
    }
    debounce_state = 0;              /* transition back to state 0 */
    break;
}
}

```

(5)

- (1) The `#pragma interrupt` designates a given function as an ISR
- (2) The ISR must be a `void (void)` function
- (3) The time-tick ISR must invoke `QF_tick()`, and can also perform other actions, 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 here interrupts are locked throughout the interrupt processing.
- (4) The clock tick ISR performs the debouncing of the SW1 user switch.
- (5) The ISR posts event to the Pelican active object by means of the `QActive_postISR()` function, which like `QF_tick()` is specifically designed to be called from the ISR context.

---

**NOTE:** QP-nano uses different interrupt locking policy for the task level and for the ISR level. Therefore, calling the task-level QP-nano services in the ISR context is not allowed. The only two QP-nano functions allowed in the ISR context are `QF_tick()` and `QActive_postISR()`.

---

### 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.

The M8C CPU supports several power-saving levels (consult the M8C data sheet for details). The following piece of code shows the `QF_onIdle()` callback that puts M8C into the idle power-saving mode.

Please note that M8C architecture allows for very **atomic** setting the low-power mode and enabling interrupts at the same time.

#### Listing 4 QF\_onIdle() for the non-preemptive (“vanilla”) QP-nano port to M8C

```
(1) void QF_onIdle(void) {           /* entered with interrupts LOCKED, see NOTE02 */
(2)     LED_4_On();                 /* blink LED_4, see NOTE03 */
    LED_4_Off();

    #ifdef NDEBUG
(3)     M8C_Sleep;                 /* tread-safe transition to the sleep mode, see NOTE04 */
    #endif

(4)     QF_INT_UNLOCK();
    }
```

- (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 LED\_4 is turned on and off to visualize the idle loop activity. Note that the LED is toggled with interrupts locked, to the period of the LED turned on is constant and does not include any time spent in the ISRs.
- (3) The sleep mode is activated with the intrinsic function, which emits the `SLEEP` instruction.

---

**NOTE:** The M8C CPU allows for atomic transition to sleep mode with interrupts still locked, as it should be done to avoid non-deterministic sleep.

---

- (4) Only after the CPU wakes up, interrupts are unlocked to service the interrupt. (In the debug mode the machine is not put to sleep, because sleep mode interferes with debugging.)

---

**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 `<qpn>\examples\psoc\iccm8c\pelican_qk_psocevall\`.

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 (see [Listing 2](#)).

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

```

#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 task level */
#define QF_INT_LOCK()          M8C_DisableGInt
#define QF_INT_UNLOCK()        M8C_EnableGInt

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

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

/* exact-width integers (ImageCraft C compiler does NOT provide <stdint.h>) */
typedef unsigned char  uint8_t;
typedef signed   char  int8_t;
typedef unsigned int   uint16_t;
typedef signed   int   int16_t;
typedef unsigned long  uint32_t;
typedef signed   long  int32_t;

#include <m8c.h>          /* M8C-specific constants and macros */

#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 */

#endif                  /* qpn_port_h */

```

- (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 (see also Chapter 12 in [PSiCC2]).
- (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 interrupts 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.

## 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](#).

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

```
#pragma interrupt_handler BSP_tick
void BSP_tick(void) {
    static uint8_t sw_debounced;
    static uint8_t debounce_state;
    uint8_t sw;

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

    LED_3_Invert(); /* toggle the LED */

    QF_tick(); /* process armed timers */

    sw = (PRT1DR & SW_1); /* read the switch */
    switch (debounce_state) { /* switch debounce state machine */
        case 0:
            if (sw != sw_debounced) {
                debounce_state = 1; /* transition to the next state */
            }
            break;
        case 1:
            if (sw != sw_debounced) {
                debounce_state = 2; /* transition to the next state */
            }
            else {
                debounce_state = 0; /* transition back to state 0 */
            }
            break;
        case 2:
            if (sw != sw_debounced) {
                sw_debounced = sw; /* save the debounced switch value */
                if (sw == SW_1) { /* is the button depressed? */
                    QActive_postISR((QActive *)&AO_Pelican,
```

```
                                PEDS_WAITING_SIG, 0);
                                }
                                }
                                debounce_state = 0;           /* transition back to state 0 */
                                break;
                                }
                                QK_ISR_EXIT();                 /* inform QK-nano about ISR exit */
                                }
```

---

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

---

## 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.

---

**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 M8C CPU.

### Listing 7 QK\_onIdle() callback for M8C.

```
void QK_onIdle(void) {
    QF_INT_LOCK();           /* blink LED_4, see NOTE02 */
    LED_4_On();
    LED_4_Off();
    QF_INT_UNLOCK();

    #ifdef NDEBUG
        M8C_Sleep;           /* tread-safe transition to the sleep mode */
    #endif
}
```

## 5 BSP for PSoCEVAL1 Board

The Board Support Package (BSP) for the PSoCEVAL1 board with non-hierarchical state machine and non-preemptive scheduler configuration is located in the directory: <qp>\examples\psoc\iccm8c\pelican\_psoceval1\ 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.
3. `qpn_port.h` contains the platform-specific customization of QP-nano (explained already)

### 5.1 Compiler Options Used

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

1. IMAGECRAFT compiler
2. Paging enabled
3. Stack page: Page 7

---

**NOTE:** You can access the compiler options by selecting the Project->Settings menu from the PSoC Designer IDE.

---

### 5.2 The BSP header file `bsp.h`

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

```
#ifndef bsp_h
#define bsp_h

(1) #include <m8c.h> /* part specific constants and macros */
(2) #include "PSoCAPI.h" /* PSoC API definitions for all User Modules */

/* Sys timer tick per seconds */
(3) #define BSP_TICKS_PER_SEC 64

/* street signals .....*/
enum BSP_CarsSignal {
    CARS_RED, CARS_YELLOW, CARS_GREEN, CARS_OFF
};
enum BSP_PedsSignal {
    PEDS_DONT_WALK, PEDS_BLANK, PEDS_WALK
};

(4) 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 Q_ROM *state);

#endif /* bsp_h */
```

- (1) The header file “m8c.h” provides the definitions of the M8C special function registers and macros.
- (2) The header file “PSoC\_API.h” defines the interface to the user module used in the design.
- (3) The BSP defines the desired ticking rate. This constant is useful for defining timeouts, which are always specified in units of clock ticks.
- (4) The BSP declares the board initialization interface

### 5.3 BSP initialization

The following `BSP_init()` function from the PELICAN application for the PSoCEVAL1 board configures the User LEDs and sets the desired Sleep Timer ticking rate as follows:

```
void BSP_init(void) {
    LED_1_Start();
    LED_2_Start();
    LED_3_Start();
    LED_4_Start();

    PRT1DR &= ~SW_1;          /* set the output direction for the SW pin */

    UART_1_Start(UART_PARITY_NONE);          /* enable UART */
    UART_1_PutCRLF();
    UART_1_CPutString("PELICAN example with QK-nano, QP-nano ");
    UART_1_CPutString(QP_getVersion());      /* QP-nano version */
    UART_1_PutCRLF();

    SleepTimer_1_Start();
    SleepTimer_1_SetInterval(SleepTimer_1_64_HZ); /* set interrupt rate */
}
```

### 5.4 Starting Interrupts in `QF_onStart()`

QP-nano invokes the `QF_onStart()` callback just before starting the background loop. The `QF_onStart()` function must configure and start interrupts. At the minimum, `QF_onStart()` must start the system clock tick interrupt. The following `QF_onStart()` function enables the Sleep Timer interrupt and unlocks the global interrupts at the M8C CPU level.

```
void QF_onStart(void) {
    SleepTimer_1_EnableInt();
    QF_INT_UNLOCK();          /* unlock interrupts */
}
```

### 5.5 Placing the ISRs in the Vector Table (Boot.tpl Template)

The PSoC Designer automatically generates the startup code `boot.asm` based on the `boot.tpl` template file. Typically, you don't need the Sleep Timer functionality provided by the PSoC Designer, because this API is not event-driven. In order to override the PSoC Designer code generation, you need to modify the `boot.tpl` template to hook the `BSP_tick()` ISR instead of the PSoC Designer's code for the Sleep Timer interrupt, as shown in the following code snippet:

```
org    64h                ;Sleep Timer Interrupt Vector
```

```
    ;`@INTERRUPT_25`                ; do NOT use the PSoC Designer's code  
    ljmp _BSP_tick                  ; go straight to BSP_tick  
    reti
```

## 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 M8C. The function locks all interrupts and enters a for-ever loop in which is slowly blinks LED4.

---

**NOTE:** This policy is only adequate for testing, but 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();                  /* make sure that all interrupts are disabled */  
    for (;;) {                       /* NOTE: replace the loop with reset for final version */  
        LED_4_On();  
        delay(10000);  
        LED_4_Off();  
        delay(10000);  
    }  
}
```

## 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 <a href="http://amazon.com">amazon.com</a> . See also: <a href="http://www.state-machine.com/psicc2.htm">http://www.state-machine.com/psicc2.htm</a>
[QP-nano 08] "QP-nano Reference Manual", Quantum Leaps, LLC, 2008	<a href="http://www.state-machine.com/doxygen/qpn/">http://www.state-machine.com/doxygen/qpn/</a>
[QL AN-Directory 07] "Application Note: QP Directory Structure", Quantum Leaps, LLC, 2007	<a href="http://www.state-machine.com/doc/AN_QP_Directory_Structure.pdf">http://www.state-machine.com/doc/AN_QP_Directory_Structure.pdf</a>
[QL AN-PELICAN 08] "Application Note: PELICAN Crossing Application", Quantum Leaps, LLC, 2008	<a href="http://www.state-machine.com/doc/AN_PELICAN.pdf">http://www.state-machine.com/doc/AN_PELICAN.pdf</a>
[Cypress 06] "PSoC Technical Reference Manual version 2.20.", Cypress, 2006	Cypress document PSoC TRM 2.20, available online at <a href="http://www.cypress.com">www.cypress.com</a>
[Cypress 05] "C Language Compiler User Guide", Cypress, 2005.	Cypress document # 38-12001 Rev. *E, available online at <a href="http://www.cypress.com">www.cypress.com</a>
"PSoCEVAL1 User Guide", Cypress 2004	Document included in the PSoCEVAL1 kit.
[Samek 07a] "Using Low-Power Modes in Foreground/Background Systems", Miro Samek, to be published in Embedded System Design magazine in 2007	<a href="http://www.embedded.com/mag.htm">www.embedded.com/mag.htm</a>

## 7 Contact Information

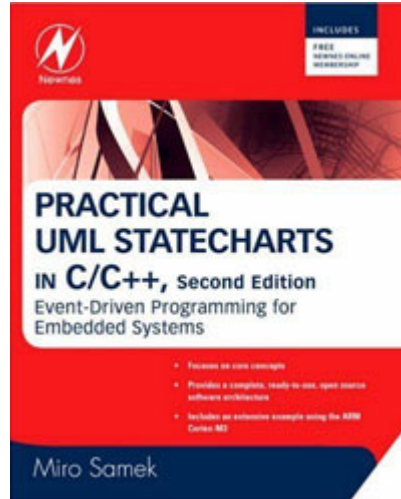
**Quantum Leaps, LLC**  
103 Cobble Ridge Drive  
Chapel Hill, NC 27516  
USA

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

e-mail: [info@quantum-leaps.com](mailto:info@quantum-leaps.com)  
WEB : <http://www.quantum-leaps.com>  
<http://www.state-machine.com>

**Cypress Microsystems, Inc.**  
2700 162nd Street SW, Building D  
Lynnwood, WA 98037

Phone (USA): 800.669.0557  
Web: <http://www.cypress.com>



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

