Your first RTX Application

To make it easy understandable we use a simple example, which could of course be implemented without RTX Kernel as well.
Our application has to execute two tasks, we name them "do-this" and "do-that". These activities have to be repeated after a certain pause, for example 50 ms. The pause starts when both activities have completed. Between the two activities is an additional, small pause of 20 ms ("do-that" has to follow "do-this" after this pause).

Now we will build the application step-by-step:

- We place the code for the two activities into two tasks (task1 and task2). Tasks can be declared in RVCT as a special type of functions by using a C language extension key word **__task**.

```
void task1 (void) __task {
  .... code of task 1 placed here ....
}

void task2 (void) __task {
  .... code of task 2 placed here ....
}
```

- When the system starts up the Real-Time Executive has to be started before any tasks may run. This is done by calling the function **os_sys_init()** In the C main function. As a parameter a task identification may be used. Instead of continuing program execution in the main function this task will be executed next.

  In our example we start the task1 first. At the beginning of task1 a call of **os_tsk_create()** will be used to start task2:

```
void task1 (void) __task {
  os_tsk_create (task2, 0);
  .... code of task 1 placed here ....
}

void task2 (void) __task {
  .... code of task 2 placed here ....
}

void main (void) {
  os_sys_init (task1);
}
```

- Now we have to implement the requested execution structure. As both activities are repeated indefinitely we place them into endless loops inside of the two tasks.
  The system function **os_dly_wait()** pauses a task for a number of system intervals. The RTX Kernel starts a system timer by programming one of the on-chip hardware timers of the ARM processors. By default the system interval is 10 ms and timer 0 is used (configurable).
  The functions **os_evt_wait_or()** and **os_evt_set()** are used to wait for event flags and to set them. In our example we use the event flag (bit position) number 3. The listing shown below contains all statements required to run the example:

```
/* Include type and function declarations for RTX */
#include "RTL.h"

/* id1, id2 will contain task identifications at run-time */
OS_TID id1, id2;
```

```
/* Forward reference. */
void task1 (void) __task;
void task2 (void) __task;

void task1 (void) __task {
  /* Obtain own system task identification number */
  id1 = os_tsk_self ();
  /* Assign system identification number of task2 to id2 */
  id2 = os_tsk_create (task2, 0);
  for (;;) {    /* do-this */
    /* Indicate to task2 completion of do-this */
    os_evt_set (0x0004, id2);
    /* Wait for completion of do-that (0xffff means no time-out)*/
    os_evt_wait_or (0x0004, 0xffff);
    /* Wait now for 50 ms */
    os_dly_wait (5);
  }
}

void task2 (void) __task {
  for (;;) {
    /* Wait for completion of do-this (0xffff means no time-out) */
    os_evt_wait_or (0x0004, 0xffff); /* do-that */
    /* Pause for 20 ms until signaling event to task1 */
    os_dly_wait (2);
    /* Indicate to task1 completion of do-that */
    os_evt_set (0x0004, id1);
  }
}

void main (void) {
  os_sys_init (task1);
}
```

- Now we have to compile our sample program and to link it with the RTX function library. Select the RTX Operating System for the project under: **Options for Target - Target - Operating System - RTX Kernel**. The final product (absolute file) may be run on your target or under uVision2 Simulator like a program without RTX Kernel.
- You can find this example in **\Keil\ARM\RV30\RTL\Kernel\Examples\RTX_ex1** folder.