1 Expansion of ROSA

1.1 More about context switching

There are two methods for context switching in the ROSA kernel.

1. Voluntary context switch from a task (cooperative mode).
2. Timer interrupts (preemptive mode).

With some tweaks cooperative and preemptive modes can be used simultaneously in ROSA.

1.2 Context switch

A cooperative, voluntary context switch is performed by the *ROSA_yield()* function call. By this call the task will give up its remaining CPU time to let other tasks run. Later, when the ROSA scheduler put the yielding task into RUN mode again, execution continue at the instruction following *ROSA_yield()*.

Example:

```
void task(void)
{
     ...
     while(1) {
          ...
          ROSA_yield();    //Voluntary context switch
          ...              //Execution continue here
                           //next time this task run.
     }
}
```

1.3 Timer interrupts

Preemptive kernel mode is activated by initiating and starting the timer interrupts once, in which case the task switches will be automatic until the timer interrupts are stopped.

ROSA utilizes timer 0 of the AVR32 to generate periodic interrupts. The periodicity is set during the initialization by the *timerInit()* function. Context switching is performed in the interrupt service routine, *timerISR(),* using the *contextSaveFromISR()* function.

The timer is started after initialization by the *timerStart()* function.

Example:

```
__attribute__((__interrupt__)) void timerISR(void)
{
    int sr;
    volatile avr32_tc_t * tc = &AVR32_TC;
    //Read the timer status register to determine if this is a valid
        interrupt
    sr = tc->channel[0].sr;
    if(sr & AVR32_TC_CPCS_MASK)
    {
        ROSA_yieldFromISR();
    }
}
```

Imagine we had a way to make a delay in a task. The code could look like this:

```
void task(void)
{
    ...
    while(1) {
        ...
        ROSA_sysTickWait(10);     //Delay 10 ticks
        ...
    }
}
```

1.4 Tick?

What is a tick? A tick is closely related to timer interrupts, which are generated by a timer circuit inside the MCU. When an interrupt happens an interrupt service routine (ISR) is run. The ISR increase the system time by one "tick". Thereafter it checks if there are any tasks which should be activated, i.e. the tasks which have previously been waiting but want to run now. For a real time system, 1 ms usually is a good period between ticks. This is often called *system tick*, or *sysTick* for short. In ROSA it would be appropriate to call the function *ROSA_sysTickWait()*.

1.5 ROSA_sysTickWait()

The delay function work according to the following:

- Link the executing task into a DELAY queue.
- Link out a task from the READY queue.

- Call the scheduler to see which task should begin execute.

As we can see this require a number of queues to handle the DELAY. At least one queue for all waiting tasks and one queue for the tasks which are ready.

None of this is yet implemented in ROSA.