ReadMe file for KEA Scheduler for Atmel ATmega32

Using this project:
The project is designed to be used on an Atmel ATmega32 microprocessor running at 16 MHz. It is compiled with speed optimization and takes up 740 bytes of code and 12 bytes of data. Without optimization the code requires 1404 bytes.

If you change the processor or the speed, you must modify the Timer and Ports modules. The timer module requires a 16 bit counter and it uses Timer 1.

The port module is used to control individual port bits. These must be properly renamed in PORTS.H.

The example shows three tasks. These can be activated synchronously (running at regular intervals in steps of 10 ms) or asynchronously (by calls from other tasks or an ISR). The synchronous tasks can be enabled and disabled (default). The intervals can be changed on the fly in order to create delays in steps of 10 ms.

Working with a task:
Create a module for each task. The supplied example has three empty tasks – start with these.

You can run a task as a synchronous task (activated by the timer) or an asynchronous task (activated e.g. by interrupts).

Synchronous mode: Set the timer interval by calling `SetTaskInterval (taskA, interval)`. The unit is steps of 10 ms. Enable the task by calling `EnableSyncTask(taskC, true);` The second parameter turns the task on / off. The flags are defined in OS.H. The interval is controlled by a countX variable, which is reset to a predefined value every time it is decremented to zero.

Asynchronous mode: Call `SetAsyncFlag(taskA);` in order to run the task. The task will be activated once.

Don't use calls to `delay()` after entering the `while(true)` loop in `main()`. A delay will block the execution of all other tasks. You can make a delay by changing the countX variable for the task, that is controlling when the task should be called next time.

Priority:
The first task in the `while(true)` loop in `main()` has the highest priority and the last task has the lowest. All tasks will run to the end when started (non-preemptive mode) and no shared memory resources are used for the tasks.

Great care should be taken when designing the tasks. You should rather make many small and efficient tasks than a few big and complex tasks. No task should be allowed to block the system for any extended period.

ELR, October 6th, 2014