Osek assignment report

STUDENT: Dalmasso Luca, s281316

PROPOSED ARCHITECTURE FOR OSEK EE_ARDUINO

For this project I decided to use the following configuration:

- STARTUP HOOK:

  Used only for serial debug purpose, it basically allows the board to print on serial a welcome message for user in order to understand when the Task starts the conversion.

- BASIC TASK:

  The application I developed doesn't need any Resources, Messages, Events..

  Just an Alarm is needed, so there's no need for extended tasks.

- HARDWARE COUNTER:
  I configured an hardware counter that's going to be used by the Alarm for timing purpose.
  The counter is associated to the 'TIMER1_COMPA' hardware timer of the Arduino Uno board, I configured it to 'tick' every 0.001 s.
- ALARM:

  Configured in such a way to start automatically, and to activate Task1 every 0.100 s.

I made some other important configuration to the Osek kernel (By configuring KERNEL_TYPE in .oil) in order to save as much memory as possible, important ones:

1. *Moving from ECC1 to BCC1*, a simple Basic task with only 1 pending activation is needed, so no sense to configure the kernel with more complex conformance classes.
2. *Using just a single Ready Queue* implemented as a simple Linked List, this is a very important change to do because the default DEMO where to start uses Multiple Queues option with 10 priorities!

Only those two changes allow you to save up to 1 Kb on executable!

NOTE: In order to make all the configuration, I used the OSEK_EE manual as reference.

PROPOSED APPLICATION ALGORITHM

I've chosen to implement a single function that translates the set of messages on the fly by consulting a table containing all morse symbols for given letter.

The function is executed by the Task every time he's woken up by the alarm, so every 100ms.

Basically the function is divided into 3 blocks:

#1: every time the task is woken up, it controls if the traduction of the current letter has been already computed, if not, than an 'on the fly' traduction is computed, else it skips.

#2:this block of code has the traduction saved into a char array, and every time the task is woken up, it reads the char value (can be '0' or '1') and decides to switch on/off the led.

So every 100ms the led is switched on/off by this block of code.

When the last char for the current traduction is read then some variables are resetted so that the block #1 can start translate the next letter.

#3:this block of code checks the time.

If 180 s are passed (180s = 1800 ms = 1800 times the task has been woken up) it put the task into an "idle"(the taks is not stopped nor suspended, it just doesn't do any useful operation for next 5 calls) state for 0.5 s, after that some variables are resetted and the translation process goes to the next message.

All this operations are done by checking the 'Timer' variable that is incremented every time the task is woken up by the alarm.

FURTHER CONSIDERATIONS:

1. Debug on serial: in order to see what happen I decided to use the Arduino's serial line function that basically can print what is the current translated bit (char) and letter that the program is computing.

   Due to the fact that the library used by Serial is very slow in terms of execution time and very big in terms of memory weight, all the debug

feature must be enabled at compile time with #DEBUG_ON_SERIAL define.

Just to be precise:

- I went from 6000 Bytes to 4800 Bytes without the debug feature enabled..
- Using Serial line and Digital at the same time leads into strange behaviours, timing is no more precise or the led doesn't work..

2. Due to the fact that String.h is not working on the standard EE VM, I decided to not use it.
3. I decided to statically allocate all arrays on the maximum possible value ex:

```
char messages[NUM_MESSAGES][MAX_MESSAGE_L+1]=
{
        "A FEATHER IN THE HAND IS BETTER THAN……."
        "….."
        …
};
```

   This leads into a waste of memory, but at the same time into a more simpler and readable algorithm.

4. Memory Occupation analysis:
   By using the Default .oil configuration of the DEMO application where I started, and with my application code, the initial size of the executable flashed was 6900 Byte.
   By changing .oil KERNEL_TYPE the size went down to 6000 Byte.
   By disabling debug feature I went from 6000 to 4800.
   The EE DEMO project that just switch on a led weights 4100 Byte.

5. Debug session read from serial line:

[1.57835603]HELLO FROM ARDUINO
[1.57852507]1
[1.67654896]0
[1.77881503]1
[1.87708497]1
[1.98054910]1
[181.50953889]IDLE
[181.61225510]IDLE
[181.70959592]IDLE
[181.80870700]IDLE
[181.91001296]IDLE
[182.00851202]1