2020/2021

Operating systems for embedded systems

Course Professor: Massimo Violante

Osek project implemented with arduino

Introduction

The solution that I have done is written in Osek and is implemented so that can be executed on Arduino uno. The goal of the project is to send five sentences in morse code reducing as much used memory.
To do that I have decided to translate one character at time in morse code and to translate it immediately in switch on/off of the led. So every time I run into a "." or a "–" a set of control instructions decide how to manage them and therefore what action to do with the led.
My C code is composed of one basic task that is controlled by two alarms in its execution.
These alarms communicate with the task thanks to the use of two events: "ev_termina" and "ev_rallenta". The first event is sent repeatedly to the task each 180s from the start of the task, in this way it can communicate to the task that it has to change the sentences to send. Moreover, I use this event also to implement the pause of 0.5s between two different sentences. The second event is used to control switching on and off of the led, it is set in a way to send a signal after every 0.1s; without this it would be impossible to understand the morse code from the led, because it would blinking too rapidly.

The .Oil File

In the oil file there is the definition of the two alarms, the two types of events, the counter and the task.
I give to the counter the name of Systemcounter, this counter is the clock for the two alarms; I set the counter to increment one value each 0.1s.
Then we have two alarms: al_rallenta, al_termina; both of them are relative to the task1 and use the same counter (Systemcounter). Al_rallenta generates each 0.1s an instance of the event "ev_rallenta", to do that I set the cycle time of the alarm on 10 and the starting time on 0s.
On the other hand al_termina is set in a way to generate an instance of the event "ev_termina". In contrast to al_rallenta, al_termina is not initialized in the oil file, but in the C code, in this way the alarm star at the beginning of the sending of the sentences and in this way it is able to send the event of stop (ev_termina) after exactly 180s. Moreover, this alarm helps me to implement the pause between two different sentences, to do that I change many times the setting of alarm, in the C code.First I set it to 0.5s to implement the pause and then to 180s to implement the sending of the sentences; at the end of each sending I have to repeat all this steps.
At the end of the oil file I declared the only task of my program and I called it Task1. It is set in a way to autostart and I give it a stack for receiving the two events that it can receive during its execution (ev_termina, ev_rallenta).

The .C file

At the beginning of the C code I define two arrays of pointers: stringhe, morse.
In the first one there are the sentences that we have to send during the execution of the program and in the second there is the morse code of each letter of the alphabet. I could create a single array with the sentences translated in morse code or better in binary, but the memory allocated for this implementation would be very big.

I write below a little scheme of the memory occupation for each case:

Solution	Calculation	Tot (bits)
First Solution (morse code+sentencese)	(8*4*27)+(66*5*8)	3.504
Second Solution (Sentences translated in morse code)	66*5*4*8	10.560
Third Solution (Sentences translated in morse code)	66*5*4*13*8	137.280

number	meaning
8	numer of bits for each element of array
4	number of "-"/"." for each letter
27	letter of alphabet including space for morse code
66	letter in each sentences)
5	number of sentences
13	number of 1/0 for each element of morse code

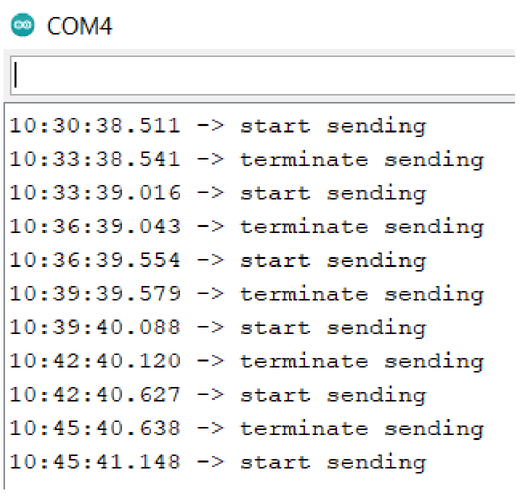
After that there are several char variables that I use in the Task1 as iterators. The task1 is the only task of this program, at the beginning of that there is a label where the execution goes at the end of each sending. then I set the alarm al_termina to implement the pause of 0.5s. The pause is implemented here because it can be executed after each sending. After the pause I set again the al_termina to send an event after 180s. Now there is the beginning of the sending phase, here there are several cycles where I take each character of the sentences and I translate it into morse code, then I switch on/off the led to send the morse code. Before each blinking I put a check to test if the event "ev_termina" is come, in this way the code is able to terminate the sending of the sentences and to come back to the beginning label to start a new sending.

Final Results and comments

Here I write a short comment about some choice that I make in the code. In the c file there are many while cycles, they unfortunately increase the memory occupation and the time of running of the program, they can be removed with loop unrolling but the code would be very difficult to understand, for this reason I have decided to introduce them and lose some optimization.
The occupation of my code is equal to 5882bytes, probably this is not the best solution in terms of memory occupation, but it is well structured and it follows very well the timing constraints.
Below I report some details about file.elf:

```
LD   arduino.elf
  text     data     bss     dec     hex   filename
  5106     608      168     5882    16fa  arduino.elf
```

Moreover, I had to make important choices to respect the timing constraints, in fact in the code every time I have to send new sentences I reset the al_termina, in this way I am able to avoid the accumulation of delay. If we have a look to the picture below, we can see that the delay is very small.



```
10:30:38.511 -> start sending
10:33:38.541 -> terminate sending
10:33:39.016 -> start sending
10:36:39.043 -> terminate sending
10:36:39.554 -> start sending
10:39:39.579 -> terminate sending
10:39:40.088 -> start sending
10:42:40.120 -> terminate sending
10:42:40.627 -> start sending
10:45:40.638 -> terminate sending
10:45:41.148 -> start sending
```

	Max Delay		Average Delay	
	Max value	Percentage	Max value	Percentage
Sentences	0,032s	0,017%	0,025s	0,014%
Pause	0,025s	5%	0,012s	2,4%