



Assignment OSEK

Trampoline & Arduino

Giovanni Pollo

290136

1 Structure & Algorithm

The structure chosen is based on a single extended task. The main reason is simplicity. Thanks to a single task, the solution is as straightforward as possible, and the memory occupation is also very low.

To guarantee the system's timing, I opted for an event triggered every $100ms$.

The conversion is online, thanks to the global variable *LED*. The external loop reads all sentences, while the internal one analyzes every single letter. The program compares every character against 'A', and it computes the value of *pos*.

Thanks to the variable *pos*, we can get the morse code of the analyzed letter, convert it into a sequence of 0 and 1, and then save the output inside the variable *LED* thanks to the **populateLED** function.

To implement the $180s$ pause, I opted for a counter (variable *cnt*) which maximum value is 1800. In fact:

$$max_cnt_value = \frac{pause_time}{event_time} = \frac{180\ s}{0.1\ s} = 1800 \quad (1)$$

Similarly, the $0.5s$ pause uses a counter in which the maximum value is 5.

2 Timing & Errors

2.1 Timing

As explained in the first paragraph, the code has a periodic alarm (every $100ms$) that activates an event. The main problem is that the Trampoline *System-Counter* is the same as the *Systick* used in Arduino, which counts a tick every $1024\mu s$. To obtain $100ms$ period, the value assigned to **CYCLETIME** must be:

$$\begin{aligned} CYCLETIME &= \frac{event_time}{tick_time} = \frac{100ms}{1024\mu s} = \frac{100 \cdot 10^{-3}s}{1024 \cdot 10^{-6}s} \\ &= 97.65625 \approx 98 \end{aligned}$$

The choice for **CYCLETIME** is, therefore, 98. We can now compute what the real value for $100ms$ is:

$$real_100ms = 98 \cdot 1024\mu s = 100.352ms$$

From this, it is easy to evaluate the default error:

$$Default_Error = \frac{real_100ms - 100ms}{100ms} \quad (2)$$

$$Default_Error = \frac{100.352ms - 100ms}{100ms} = 0.352\% \quad (3)$$

2.2 Errors

To analyze the program's timing errors, I used the Arduino function `micros()`. We can identify three errors:

1. *100ms*: I obtained 0.352% error

$$Error = \frac{value_with_micros - ideal_value}{ideal_value}$$

$$Error = \frac{100352\mu s - 100000\mu s}{100000\mu s} = 0.352\%$$

2. *500ms*: I obtained 0.352% error

$$Error = \frac{value_with_micros - ideal_value}{ideal_value}$$

$$Error = \frac{501760\mu s - 500000\mu s}{500000\mu s} = 0.352\%$$

3. *180s*: I obtained 0.352% error

$$Error = \frac{value_with_micros - ideal_value}{ideal_value}$$

$$Error = \frac{180633600\mu s - 180000000\mu s}{180000000\mu s} = 0.352\%$$

It's easy to notice that all the errors are the same, and they are all coherent with the *Default_Error* evaluated in section 2.1, with the equations 2 and 3

3 Memory Occupation

To analyze memory occupation, I compared my solution with a blank code (an empty `PeriodicTask` triggered every *100ms*).

Text	Data	Bss	Dec
5730 Bytes	278 Bytes	382 Bytes	6390 Bytes

Table 1: Blank code memory occupation

Text	Data	Bss	Dec
7034 Bytes	678 Bytes	260 Bytes	7972 Bytes

Table 2: My solution memory occupation

By comparing Table 1 and table Table 2, it's easy to see the benefit of the online translation. Because the program translates letter by letter, the data occupation doesn't increase too much. As a downside, the text size grows slightly, but because Arduino RAM is limited to *2kB*, I decided to optimize the program to use the least amount of RAM.