Electrical, Electronic & Computer Engineering
School of Engineering & Physical Sciences

B31DG: Embedded Software 2024
Assignment 2, Released on Week 7, Demo due on Week 12, Report Week 13

1. Problem

You need to a machine monitor system using FreeRTOS and your ESP32 kit.
The system must execute the following 8 tasks, with hard real-time deadlines at the end of each period.

1. Output a digital signal. This should be HIGH for 180μs, then LOW for 40μs, then HIGH again for 530μs, then LOW for 3.25ms and repeat the same pattern once every 4ms [Period = 4ms / Rate = 250Hz]
2. Measure the frequency of a 3.3v square wave signal, once every 20ms. The frequency will be in the range 333Hz to 1000Hz and the signal will be a standard square wave (50% duty cycle). Accuracy to 2.5% is acceptable. [Period = 20ms / Rate = 50Hz]
3. Measure the frequency of a second 3.3v square wave signal, once every 8ms. The frequency will be in the range 500Hz to 1000Hz and the signal will be a standard square wave (50% duty cycle). Accuracy to 2.5% is acceptable. [Period = 8ms / Rate = 125Hz]
4. Sample one analogue input and update a running average of the last 10 readings. [Period = 20ms / Rate = 50Hz]
   The analogue input must be connected to a maximum of 3.3Volts, using a potentiometer. The task should also visualise an error (using a LED) whenever (average_analogue_in > half of maximum range).
5. Log the following information once every 200ms [Period = 200ms / Rate = 5Hz] in comma delimited format, i.e. "%d,%d") to the serial port at a baud rate of 9600 bits per seconds.
   a) Frequency value measured by Task 2 (Hz, as integer)
   b) Frequency value measured by Task 3 (Hz, as integer)
   Important: frequencies should be scaled and bounded between 0 to 99. For example, the output "0,0" will mean Task 2 has measured a frequency of 333Hz, or less, and Task 3 has measured frequency of 500Hz or less. The output "99,99" will mean that both tasks have measured frequencies equal or above 1000Hz.
7. Control a LED. The state of the LED should be toggled by a pushbutton. The pushbutton should be debounced.
8. Write a function called "*CPU_work(int time)*" which causes the ESP32 to use about *time* ms. It should do this with a "for" loop. Call CPU_work(2) from a periodic task [Period = 20ms]

Your program needs to use FreeRTOS (so you can use any number of FreeRTOS tasks, timers, queues, semaphores, etc.). It is your choice whether you implement the required tasks using actual FreeRTOS Tasks or callback functions called by FreeRTOS software timers, or a combination of the two approaches.

In particular, your program needs to:

- Monitor the digital input and control the LED from two independent tasks.
- Use an event queue to enable the communication between those two tasks, i.e. then the pushbutton is pressed the monitor digital input task will queue an event, which needs to be received and acted upon by the control LED task.

- Use a global structure (struct), to store the frequencies measured by Task #2 and Task #3 and printed to the serial port by Task #5. Access to this structure must be adequately protected, using FreeRTOS'semaphore(s).
- Use any other FreeRTOS constructs you consider useful for your program.

3. Demo (Week 12)

Before the demo lab session (during Week 12) you will be given a library and a tutorial showing how you should use it instrument your program to show that it meets all its real time requirements, i.e. the rates are respected, and deadlines are not missed.

At the demo lab session (during Week 12), you will be asked to:

- Run your code and show it meets all its specification. You will demonstrate all the features and show the signal generated (requirement 1) and the execution time of task 2 using the oscilloscope.
- Modify the periods of any of the tasks you have implemented, and to run again your system.
- Answer questions on your system, e.g. design, testing, performance.

5. Submission (Week 13)

Submit the following paperwork:

- Link to your code repository, with fully documented source code
- Short (maximum 5 pages) report describing the design of your program (using one or more diagrams, e.g. data-flow, UML, as you consider necessary to document your design), and summarising and analysing results (tasks successfully implemented, performances ...).

  Your report should include answers to the following points:
  - How did you decide to set the priorities of your FreeRTOS tasks? Why?
  - How did you decide how to size the stacks of your tasks?
  - How did you protect the access to the global structure? Why?
  - What tests did you perform to check whether all the RT requirements are respected? Make sure to describe these tests in detail, providing evidence of the results of any tests you have performed, if any.
  - What is the worst-case delay (response time) between the time the push button is pressed and the time the LED is toggled? Justify your answer.
  - How would your FreeRTOS implementation compare with a custom cyclic executive implementation?
- Signed authorship statement.

6. Marking criteria

- Work as an individual. Students **MUST** work on this project on their own. Please do not use code from another student or offer code to another student. Code may be run through a similarity checker. You will be asked questions about your code during the demo session. The assignment will not be marked if plagiarism is suspected.
- The submission deadline is at the end of Week 13.
- The assignment contributes 30% towards the 50% continuous assessment mark.
  - Quality of the design and implementation is worth 72% of the mark for this assignment, with full marks **if Q&A is satisfactory** and for:
    - Functional correct tasks, i.e. all 8 tasks work correctly (adhering to functional and RT requirements). 9% points max for each.
    - Efficient, modular, easy to maintain and re-usable code; up to 5% points will be removed if the code shows issues for each of these aspects.

    In particular, your program should:
    - Share resources and decouple/ synchronise tasks, by keeping the use of global variables as limited as possible, and with functions/FreeRTOS tasks as independent as possible from each other.
    - Avoid wasting CPU and also memory (including by properly sizing stack sizes).
- Good programming style, including appropriate names for variables, good layout /indentation, and commenting style; up to 5% points will be removed if the code shows issues for each of these aspects.

- Documentation is worth 28%, with full marks for:
    - Clear and complete answers to all the questions highlighted in point #5. Max 20% points.
    - Well written, complete and well organised report, effective use of language and diagrams. Max 8% points.