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

B31DG: Embedded Software 2023
Assignment 2, Released on Week 5, Due on Week 9

1. Problem

A "Cyclic Executive" is an important way to sequence tasks in real-time systems, without relying on a real-time-operating systems (such as RTOS).

Your are being asked to design, build and test a simple cyclic executive to schedule the execution of N periodic tasks $T_1,\ldots,T_N$.

Each T_i is specified by (P_i, C_i) , where

- P_i is the period, and
- C_i is the worst-case execution/computation time (the longest possible time it takes for T_i to execute).

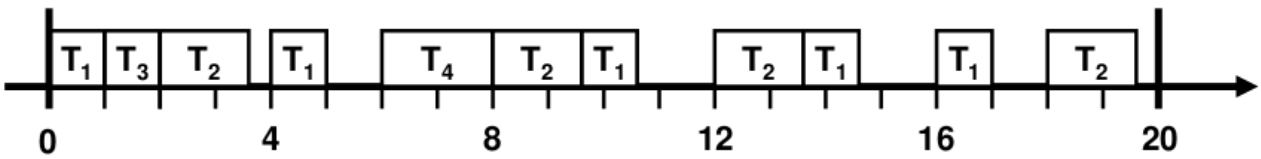
All tasks should be treated as hard real-time tasks: Task T_i is "released" at the start of its period, executes (at most) C_i time units, and must finish before P_i units have passed (its hard deadline), i.e. each task T_i must be done once at least every P_i time units.

Example

Consider a system with four tasks

- $T_1 = (4,1)$ // this task takes 1 time unit. It must repeat once every 4 time units.
- $T_2 = (5, 1.8)$ // this task takes 1.8 time unit. It must repeat once every 5 time units.
- $T_3 = (20, 1)$ // this task takes 1 time unit. It must repeat once every 20 time units.
- $T_4 = (20, 2)$ // this task takes 2 time units. It must repeat once every 20 time units.

The following timeline shows one possible schedule that meets all the deadlines.



Timeline showing when each tasks is active at which time. X axis is time.

Note that in its third instance, T1 could not start at t=8, as T2 would miss its next deadline (at t=10). It could also not start at time t=7 (e.g. if T4 was shifted one time unit earlier), as it has already occurred at time t=4 and its next release is t=8. Similar considerations apply to the 4th instance of T1.

1. Specification

Your cyclic executive will implement a machine monitor system.
The system must execute the following 5 periodic tasks, with specified timings.

1. Output a digital signal. This should be HIGH for 200µs, then LOW for 50µs, then HIGH again for 30µs, 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. Read one analogue input, and compute a filtered analogue value, by averaging the last 4 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 100ms [Period = 100ms / Rate = 10Hz] 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 1000H.

3. Task 1

Write a C/C++ program to implement the above problem for your (ESP32/ESP32C3)
The code should be clean, modular, with good layout and well commented (see marking criteria).

4. Task 2

At the demo lab session (during Week 9), you will be asked to

- Run your code and show it meets the specification. The lecturer / TA will ask you feed different inputs into your controller, e.g. by changing the frequency of the signals measured by tasks 2 and 3, and turning the potentiometer monitored by Task 4.
- Show the commit/revision history of your code repository
- Answer questions on your system, e.g. design, testing, performance.

5. Task 3

Submit the following paperwork

- Fully documented source code
- Short (about two/three pages) report describing the design of your cyclic executive and the implemented schedule (using appropriate diagrams), summarising and analysing results (tasks successfully implemented, performances ...).
- Screenshot from your code repository
- Your signed statement of authorship (you can doiwnload the template from canvas)

6. Notes and Suggestions

- Your code MUST be in the form of a cyclic executive (see material discussed in classes and also relevant publications, such as [1] [2] [3]). Other formats will not be accepted. Hints: Start by

---

1Locke, C. Douglass. "Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives." *Real-time systems* 4 (1992): 37-53.

2 Baker, Theodore P., and Alan Shaw. "The cyclic executive model and Ada." *Real-Time Systems* 1.1 (1989): 7-25.

3Burns, Alan, Neil Hayes, and M. F. Richardson. "Generating feasible cyclic schedules." *Control Engineering Practice* 3.2 (1995): 151-162.

(i) implementing single tasks and measuring their worst case execution times, e.g, using the function *micro()), (ii)* finding out the length of the major cycle for your cyclic executive, i.e. the hyperperiod - the least common denominator, (LCD) - of the periods of all the tasks, and (iii) selecting a suitable duration for the slot/frame time and designing the schedule for your tasks, i.e. deciding which tasks to execute in each slot/frame, without overrunning frames and without missing tasks' deadlines (note that many slots/frames may "do nothing" - these have to be accounted for if the cyclic executive is to work correctly).

- Your code may only use one Ticker object (**https://techtutorialsx.com/2021/08/07/esp32-ticker-library/),** no other similar or RTOS libraries. Solutions without Ticker will be perfectly acceptable.
- Interrupts cannot be used to implement the frequency measuring tasks, i.e. Tasks 2 and Tasks 3.
- Your program must use only a single core. You cannot distribute tasks across multiple CPUs in the case you have a dual core ESP32.
- Your micro-controller must run at the default clock speed (160 MHz).
- All the communicationt through the serial port should be at 9600 baud rate.
- For tasks 2 and 3, use one of the waveform generators in the GRID lab. Students unable to use the GRID could test their software by wiring the digital inputs to a PWM outputs, generated from the same micro-controller with the nominal frequencies and duty cycles.
- Use any of the usable I/O pins of your microcontroller. Study carefully the pinout and specifications concerning the specific board included in your kit, to see if any of them is reserved / should not be used.
- Documented source code and short report MUST be submitted in Week 9 on Canvas
- Demonstration will be during lab session on the same Week 9,
- Use a source code control system (e.g., Github), to be shown on test day.

6. Marking criteria

1. 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.
2. The submission deadline is on Week 9.
3. Assignment contributes 15% towards the 50% continuous assessment mark.
4. Quality of the design and implementation is worth 10%, with full marks for:
   - All 5 tasks implemented and working correctly, as specified. 2% max for each task, i.e. 1% if the task performs as specified (e.g. generating desired digital signal. measuring signals' frequencies, sampling and filtering analog input, printing summary to serial port in the specified output…) and 1% if the task meet ALL its deadlines.
   - Good programming style, including appropriate names for variables, good layout /indentation and commenting style; 1% point will be removed if the code shows issues for each of these aspects.
   - Efficient, modular, easy to maintain and re-usable code; 1% point will be removed if the code shows obvious issues for each of these aspects.
5. Documentation is worth 5%, with full marks for:
   - Well written, complete and well organised report, effective use of language, no typos.
   - Correct use of diagram style (among those studied in the first part of the course).