IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2015

MSc and EEE PART IV and EIE PART III: MEng and ACGI

Corrected Copy

EMBEDDED SYSTEMS

Friday, 15 May 10:00 am

Time allowed: 1:30 hours

There are TWO questions on this paper.

Answer BOTH questions.

The questions carry equal marks.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible	First Marker(s) :	T.J.W. Clarke
	Second Marker(s) :	J.V. Pitt

The Questions

1. An ARM Cortex processor uses a periodic timer interrupt of group priority 2 with ISR *pollcount* to poll a single digital input that is driven from an external pulse source. The pulse high and low times are guaranteed to be no less than $1/(2f_p)$, where f_p is the maximum allowed pulse frequency. You may assume that the non-interrupt code running on the processor never disables interrupts and never changes the CPU PRIOMASK register. You may also assume that the timer period T_2 is such as to make CPU utilisation from *pollcount* much smaller than 100%.

   a) Explain the function of group-priority and sub-priority bit fields, and threaded and handler modes, when nested interrupts occur in the Cortex architecture.

   [2]

   b) Ignoring interrupt response, state, giving reasons, the relationship between the period of the timer T_2 and f_p that ensures all pulses are counted by software using this method.

   [2]

   c) Explain in what manner *late arrival* and *tail chaining* optimisations in the Cortex NVIC can alter the interrupt response of a given interrupt of group priority 2 in the presence of one or more other interrupts.

   [4]

   d) Taking interrupt response into account, calculate when and how these optimisations would change interrupt jitter and hence the maximum allowed value of f_p for fixed T_2.

   [4]

   e) Now consider the *pollcount* interrupt running under FreeRTOS with possible changes to `PRIOMASK` in the RTOS. Describe the function of:

      configKERNEL_INTERRUPT_PRIORITY
      configMAX_SYSCALL_INTERRUPT_PRIORITY

      How do these two numbers affect *pollcount* jitter and correct operation of the system in the case that the *pollcount* ISR does not contain any RTOS API calls. How would your answer change if the ISR did contain API calls?

   [4]

   f) Discuss whether selection of the FreeRTOS tick period and T_2 could be used to reduce *pollcount* jitter, and if possible what would be the likely benefit.

   [4]

2. a) Figure 2.1 shows a subroutine `swap` which implements an atomic swap operation using the new `ldrex` and `strex` instructions.

i) Explain how this code works, giving an execution trace of its simultaneous overlapping execution on two different CPUs.

[2]

ii) In the case of overlapping calls to `swap` from different CPUs can the system stay in livelock? Justify your answer.

[2]

iii) Give two advantages of `ldrex/strex` over an atomic swap machine instruction such as `swp` when implementing synchronisation in an RTOS running on multiple CPUs.

[2]

b) Figure 2.2 shows the CPU length and period of a set of 101 tasks implementing jobs and running on a single CPU core in an RTOS application. Tasks have priority equal to their number, where higher priorities are larger numbers, and have pre-emptive priority scheduling. There is no blocking other than that caused by the use of a semaphore. The critical section length is shown where the semaphore is used. Each task using the semaphore has one critical section of the specified length.

i) Which tasks suffer blocking due to the semaphore?

[2]

ii) Determine the maximum blocking for each task. Explain this result.

[4]

iii) State what extended rate monotonic analysis says about the schedulability of this system both with and without the use of Priority Inheritance Protocol in the semaphore.

[2]

iv) Determine the schedulability of this system using the maximum completion time method.

[3]

v) Discuss the merits of Priority Threshold Scheduling and Priority Inheritance Protocol in this system.

[3]

```
; atomic swap of r0 with mem[r1]
swap  ldrex   r2,[r1]
      strex   r3, r0, [r1]
      cmp     r3, #0
      bne     swap              ; branch if r3 != 0
      mov     r0, r2            ; r0 := r2
      bx      lr                ; return
```

Figure 2.1: Implementation of atomic swap using `ldrex/strex`.

Task	CPU length	Period	Semaphore critical section length
101	1	5	n/a
n where $1 \leq n \leq 100$	12	$2100 - n$	4

Figure 2.2: 101 Tasks. Note that CPU length includes critical section.