THE ANSWERS

A: Analysis, B: Bookwork, D: Design

1. An ARM Cortex processor clocked at 50MHz interfaces with a custom real time clock (RTC) module. Time is provided as the values of 4 32 bit registers memory mapped at offsets (0,4,8,12) from address EXT_RTC_BASE. Each register holds 16 bits of the double-word time value in its least significant half-word and zero in its most significant half-word. Registers with lower offsets are less significant. The clock module hardware increments its internal registers synchronously every ARM clock cycle, or 20ns.

   a) Write a C function GetRTC32() without critical sections that in absence of interrupts is guaranteed to return the correct least significant 32 bits of the time value. Explain what is the data consistency problem in this operation, and how your code overcomes it.

D.
The problem is that the LS two registers must be read as consistent values but can be changed at any time by hardware. Hence there is an unavoidable possibility that between reading the first and the second the hardware change will happen and therefore the values will be inconsistent. The code overcomes this through detecting the possible inconsistency by reading tmp0 twice, and correcting.

```
int GetRTC32()
{
   unsigned int tmp0, tmp1, tmp0_new;
   tmp0 = *(EXT_RTC_BASE);
   tmp1 = *(EXT_RTC_BASE+4);
   tmp0_new = *(EXT_RTC_BASE);
   if (tmp0_new < tmp0) { //overflow
      tmp1 = *(EXT_RTC_BASE+4);
      }
   return (tmp1 << 16) + tmp0;
}
```

[5]

   b) The error in the function GetRTC32() is defined to be the difference between the returned value of time, and the time at the end of the return from the function. If there is an interrupt of period 1ms with ISR of length 100 clock cycles including context switch overhead, estimate, with reasons, the largest and smallest value of this error.

A.
If no interrupt the error is the time between reading the LS register from the clock and finishing GetRTC32(). This is approximately 10 cycles. An interrupt can occur any time and therefore can increase this to 110 cycles.

[4]

   c) Now suppose there are two similar asynchronous interrupts both of period approximately 10μs and length, not counting context switch overhead, of 80 cycles each. What is the maximum error as defined in the last part, assuming that FPU state is not saved on exceptions, and ignoring any effects from interrupt late arrival and tail chaining optimisations in the NVIC?

A.
Worst case the two interrupts both happen one after another adding 160 cycles + context switch of 16 cycles/interrupt. Total 192 cycles + approx 10 cycles = 202 cycles.

[3]

d) Detail, giving reasons for your answers, for the case described in part c, how are the maximum and the average error affected by:

i) Late arrival optimisation in the NVIC.

B/A.
No change, this affects which interrupt is executed but not overall time

[2]

ii) Tail chaining optimisation in the NVIC.

B/A.
This will have no affect on maximum times because worst case interrupts happen one after another. It will slightly reduce average time, because when interrupts overlap the total delay will be less by 12 cycles.

[2]

e) In multi-processor system a task on one CPU reads the 64 bit RTC module time as in part a and writes this value at intervals into a double-word `time[0]`, `time[1]` in shared memory. This value is read by tasks running on other CPUs. Both reading and writing is least significant word first. Provide one or more execution traces to illustrate all the distinct ways in which this memory read operation could result in inconsistent data read, and state what is the possible error and the conditions under which such a trace might happen.

D. There are two distinct traces that cause problems:
Write 0
Read 0
Read 1
Write 1

This could happen if a read task from another CPU happened in the middle of the write task executing. It could lead to time read too small by 2^{32}. $2^{32} - 1$ would be equally acceptable.
Read 0
Write 0
Write 1
Read 1

This could happen if the write task happened in the middle of a read operation on another CPU. It could lead to time read too large by $2^{32} - 1$. 2^{32} would be equally acceptable.

[4]

Question 1. has a total of 20 marks.

2. Figure 2.1 details the CPU time and period of a set of 4 tasks in a real-time system. The tasks are specified using a job model in which new work of the specified CPU time is available periodically with the given fixed period. Each task has the hard deadline that it must complete its work before the end of its period. CPU work is specified in terms of CPU clock cycles, and all context switch and semaphore API overheads should be ignored. The minimum CPU clock frequency at which the system is guaranteed to be schedulable, f_0, is calculated below under various different conditions.

   a) Suppose that there is no blocking. Using Rate Monotonic Analysis (RMA) determine f_0. Specify the relative priority of tasks required for this guarantee.

   Calculated example.
   RMA limit is $4 * (2^{1/4} - 1) = 0.757$
   Priority $D > B > A > C$. $1 + 0.4 + 0.05 + 0.5 = 1.5$ clocks/us utilisation. So $1.5 < f * 0.757 \Rightarrow f > 1.98$ Mhz
   [5]

   b) Suppose all tasks share a resource used once per task period. Task priorities are as determined by RMA above. The resource access is protected by a binary semaphore that introduces blocking. The CPU time using the resource is 10 cycles, and included in the time shown in Figure 2.1. Calculate the maximum extra time (in clock cycles) for each task due to semaphore blocking assuming that each task blocks any other task at most once. State which tasks suffer priority inversion.

   A.
   $D > B > A > C$
   C no extra, no priority inv (all tasks are higher priority) (0.05)
   A 10 cycles/1000 extra, no priority inv (only C can block, any other task is higher than A) (1) + 10 cycles = 0.01 cycles/us blocking
   B: A and C can block. C blocking can be delayed by A so B suffers priority inversion wrt A. Total blocking 10 cycles + 1000 cycles (A executes) = (0.4) + 1010 cycles = 2.02 cycles/us blocking
   D: B,A,C can block. prio inv wrt B and A. Total time: 10 + 1000 + 200=1210 cycles block in 100us $\Rightarrow$ (0.5)+12.1 blocking.
   [4]

   c) Repeat the analysis in part b assuming that a mutex implementing Priority Inheritance Protocol protects the resource.

   A.
   Now blocking can be at most 10 cycles for every lower priority task:
   C (none)
   A (10 cycles) $\Rightarrow$ 0.01 cycles/ us
   B (20 cycles) $\Rightarrow$ 0.04 cycles/us
   D (30 cycles) $\Rightarrow$ 0.3 cycles/us
   [4]

   d) Determine f_0 in this system in each of the binary semaphore and mutex cases.

   Calculated example.
   PIP: $1.5 \Rightarrow 1.5 + 0.3 + 0.04 + 0.01 = 1.85$ cycles/us$\Rightarrow 1.85/0.757 = 2.44$MHz
   normal: $1.5 \Rightarrow 1.5 + 14.13 \Rightarrow 15.63/0.757 = 20.65$MHz
   [2]

   e) State the potential advantages and disadvantages of using Priority Threshold Scheduling (PTS) in the system of part a. Suppose all tasks are implemented with no pre-emption using co-routines and priorities as

determined by RMA. Determine the maximum delay in start time of each task due to lack of preemption. Therefore calculate f_0 in this case. How can this no pre-emption system be modelled using PTS?

Calculated example.
C: no change
A: 100 cycles (from C) $\Rightarrow +0.1U$
B: 1000 cycles (from A) $\Rightarrow +2U$
D: 1000 cycles (from A) $\Rightarrow +10U$

1.5 $\Rightarrow$ 13.6 cycles/us$\Rightarrow$ max frequency is $13.6/0.757 = 17.97$MHz
PTS can reduce task switching. useful if context switch overhead is significant. It can reduce stack use since tasks can be grouped and handled as sets of co-routines. It reduces schedulability as here where the system is equivalent to using PTS with all thresholds at maximum priority.

[5]

Question 2. has a total of 20 marks.

Task	CPU time clocks	Period μs
A	1000	1000
B	200	500
C	100	2000
D	50	100

Figure 2.1: Task Specification.