

8.4 Interrupts and timers

Name: _____ Class: _____ Date: _____

Total: 22 marks

KEY WORDS interrupt • interrupt service routine • timer
• prescaler

Objective

Configure interrupt vectoring, priorities and flags; design timer periods from prescaler $\times$ reload; and verify latency budgets – every period checked against register widths and the clock.

1 Key knowledge and worked examples

■ The interrupt contract

1. Hardware raises the request; 2. the CPU finishes the current instruction, saves state, vectors to the ISR; 3. the ISR does its (short) work and **clears the flag**; 4. return-from-interrupt restores state. The **NVIC** assigns priorities: a higher-priority ISR can preempt a lower one (**nesting**).

■ Worked example: the 1 Hz tick, designed and checked

84 MHz clock, 1 Hz interrupt: prescaler 8400 $\rightarrow$ 10 kHz; reload 10 000 $\rightarrow$ 1 s. Checks: $8400 \leq 65\,535$ (16-bit prescaler) ✓; $10\,000 \leq 65\,535$ (16-bit reload) ✓; the product $8400 \times 10\,000 = 84\,000\,000 =$ the clock ✓. A first draft (“prescaler 84 000”) fails the width check – the check IS the design step.

■ ISR discipline

Short, no busy-waiting, no printf; clear the flag before returning (or the ISR re-enters immediately); communicate via volatile flags/buffers with the main loop. **Latency** = longest disabled-interrupt time + vectoring – the budget to check against every deadline.

2 Practice

2.1 A 16 MHz clock needs a 10 ms timer tick. Choose prescaler and reload (both 16-bit) and verify both widths and the product. [4]

2.2 An ISR forgets to clear its flag. What happens? [2]

2.3 Two interrupts arrive together: a 1 kHz tick and a UART byte. Which should have higher priority, and why? [2]

3 Exam-style questions

3.1 A system runs a 2 kHz control loop from a timer interrupt and a 10 kHz encoder counter interrupt, both at 84 MHz.

Design the 2 kHz tick: choose prescaler and reload (16-bit each), verify the widths and the product. The encoder ISR takes 2 μs and the control ISR 8 μs ; compute the total interrupt load as a percentage of the CPU ($10\text{ kHz} \times 2\text{ }\mu\text{s} + 2\text{ kHz} \times 8\text{ }\mu\text{s}$), and state whether the 2 kHz control computation (which takes a further 100 μs inside its ISR) still meets its 500 μs period budget. [7]

3.2 A debounce design: a 1 kHz timer interrupt samples a button pin (active low); a state machine requires 5 consecutive equal samples to declare a change.

Write the ISR logic (in C or pseudocode): the sample, the counter/state update, the declared `pressed` variable. Trace the sequence for the bouncing input 1, 0, 0, 0, 0, 0, 1, 1, 1, 1 (10 samples, was released) and give the sample index at which each edge is declared. Compute the input's total response time to a genuine press. State why the ISR must be short. [7]
