

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]

Solutions

2.1

- Prescaler 160, reload 1000: $16\text{ MHz}/160 = 100\text{ kHz}$, $1000\text{ ticks} = 10\text{ ms}$.
- Widths: $160 \leq 65\,535$ ✓, $1000 \leq 65\,535$ ✓; product $160 \times 1000 = 160\,000 = 16\text{ MHz} \times 10\text{ ms}$ ✓.

2.2

- The flag stays set, so the instant the ISR returns, the interrupt re-triggers – the ISR runs back-to-back forever and the main loop starves.

2.3

- The encoder: at 10 kHz its deadline ($100\text{ }\mu\text{s}$) is far tighter than the tick's $500\text{ }\mu\text{s}$ – a missed encoder sample is lost position data (unrecoverable), while the tick can tolerate a few μs of delay.

3.1

- 2 kHz from 84 MHz : prescaler 840 ($\rightarrow 100\text{ kHz}$), reload $50 \rightarrow 2\text{ kHz}$. Checks: $840 \leq 65\,535$ ✓, $50 \leq 65\,535$ ✓, $840 \times 50 = 42\,000 = 84 \times 10^6/2000$ ✓.
- Encoder load: $10\,000/\text{s} \times 2\text{ }\mu\text{s} = 20\text{ ms/s} = 2.0\%$ of the CPU.
- Control load: $2000/\text{s} \times 8\text{ }\mu\text{s} = 16\text{ ms/s} = 1.6\%$.
- Control ISR total $8 + 100 = 108\text{ }\mu\text{s}$ of work per $500\text{ }\mu\text{s}$ period, plus interference: worst case one encoder ISR ($2\text{ }\mu\text{s}$) inside it $\rightarrow 110\text{ }\mu\text{s} < 500\text{ }\mu\text{s}$ ✓ meets the budget with $390\text{ }\mu\text{s}$ to spare.

3.2

```
volatile uint32_t pressed = 0;           // declared state
static uint32_t stable = 1;             // last declared level (released = 1)
static uint32_t count = 0;

void TIM2_IRQHandler(void) {
    clear_timer_flag();
    uint32_t sample = (*IN >> 6) & 1u;
    if (sample == stable) { count = 0; return; } // agrees with declared
↪ state
    count++;
    if (count >= 5) { stable = sample; pressed = (sample == 0); count = 0;
↪ }
}
```

- Trace (declared state 1, count 0): sample 1 (agrees, count 0); sample 0 (differs, count 1); 0 (2); 0 (3); 0 (4); 0 ($5 \geq 5 \rightarrow$ declare pressed at index 5); sample 1 (differs from new state 0, count 1); 1 (2); 1 (3); 1 (4) – the release is not yet declared within these 10 samples (it would declare at the 5th consecutive 1, index 10).
- Response time: $5\text{ samples} \times 1\text{ ms} = 5\text{ ms}$ (4 ms of latency beyond the first change – the price of noise immunity). The ISR must be short: at 1 kHz , every wasted microsecond is spent 1000 times a second.