

8.3 Memory and I/O

Name: _____ Class: _____ Date: _____ **Total: 21 marks**

KEY WORDS memory-mapped I/O I/O • GPIO • polling

Objective

Configure GPIO and memory-mapped peripherals, cost polling loops against the clock, and reason about DMA versus programmed I/O – every cost checked in cycles and microseconds.

1 Key knowledge and worked examples

■ The peripheral contract

A memory-mapped peripheral is a set of registers with a datasheet contract: configure, then read or write. GPIO needs direction before value; a status register reports completion; a data register moves the payload.

■ Worked example: costing the polling wait

Waiting for a DONE bit at 84 MHz, ~4 cycles per loop iteration (load-AND-branch-increment): 1 ms burns $84000000 \times 10^{-3} / 4 = 21000$ cycles – 100% of the CPU. The same 1 ms by interrupt costs one entry, ~20 cycles of service, one exit: the CPU is free for the other 99.98%. The comparison is the unit’s design argument, and it is pure arithmetic.

■ DMA: the zero-CPU transfer

DMA moves data without the CPU: a controller with source, destination and count registers does the transfer while the CPU runs other code. The cost model: programmed I/O costs one store per item (plus loop overhead); DMA costs one setup (~tens of cycles) regardless of length. The break-even is the transfer length where setup beats per-item cost – for a 10-cycle setup and 4-cycle stores, ~3 items; real systems use DMA for buffers of hundreds, where it wins by two orders of magnitude.

2 Practice

2.1 A polling loop at 48 MHz takes 5 cycles per iteration. How many iterations fit in 100 μ s? [2]

2.2 A status register’s bit 7 means busy. Write the C line that waits (assuming volatile REG). [2]

2.3 State one thing polling does better than interrupts (besides simplicity) and one thing it always does worse. [2]

3 Exam-style questions

3.1 A 64-byte SPI buffer must be sent. Option A: programmed I/O, one status-pollled store per byte (each byte: ~12 cycles of polling + 2 of store). Option B: DMA with a 40-cycle setup that then transfers one byte every 8 cycles without CPU help.

Compute each option’s cycle cost at 48 MHz and the CPU time freed by DMA (as a percentage of option A). Then: the application must ALSO run a 40 000-cycle control computation in the same period – compute whether each option meets the deadline (total available cycles in the period), and state which option scales when the buffer doubles to 128 bytes. [8]

3.2 A button connects to GPIO pin 6, active low (pressed = 0). The port: DIR at 0x2000_0100, IN at 0x2000_0108 (32-bit volatile).

Write C to configure the pin as input with its internal pull-up enabled by a PULL register at 0x2000_0110 (bit 6 = 1 enables pull-up), then a polling read that sets a variable **pressed** to 1 exactly when the pin reads 0. Compute the polling cost of checking 1000 times per second at 24 MHz with 3 cycles per check, as a percentage of the CPU. State why the pull-up matters for the electrical correctness of the read. [7]
