

Embedded System (315338)

MSBTE Diploma · K-Scheme · Model Question Paper — Set B · Model Answers

Q.1

a) Four characteristics of an embedded system.

2 M

- Single-functioned / dedicated:** performs one specific task repeatedly (e.g. washing machine control).
- Tightly constrained:** low cost, small size, low power and limited memory.
- Reactive and real-time:** continuously reacts to inputs from sensors and must respond within a fixed time.
- Microcontroller based with firmware in ROM/Flash;** highly reliable and stable — works for long periods without user interaction.

(Any four, ½ M each)

b) Role of RTC.

2 M

- An RTC (Real Time Clock) is a clock chip / peripheral (e.g. DS1307, DS3231) that keeps track of the **current time and date** (seconds, minutes, hours, day, date, month, year) using a 32.768 kHz crystal.
- It runs from a **backup battery** when main power is off, so time is not lost.
- Used for time-stamping of data (data loggers), scheduling events and alarms, periodic wake-up from sleep, and time display (clocks, attendance systems).

(2 M)

c) Synchronous vs asynchronous communication.

2 M

Synchronous	Asynchronous
Transmitter and receiver use a common clock signal (sent on a separate line).	No common clock; both sides use the same agreed baud rate.
Data is sent as continuous blocks / frames.	Data is sent character by character, each framed with start and stop bits.
Faster, less overhead.	Slower due to extra start/stop bits.
Examples: SPI, I ² C, USRT.	Examples: UART (RS-232), USB serial adapters.

(Any two points, 1 M each)

d) Four features of microkernel.

2 M

- Kernel contains only minimum essential services — inter-process communication (message passing), basic scheduling and memory/address-space management.
- Other services (file system, device drivers, network stack) run as separate user-mode servers.
- High reliability and security — failure of a server does not crash the kernel.
- Small size, modular, easy to extend and port; examples: QNX, MINIX 3, L4.

(Any four, ½ M each)

e) LCD pins RS and E.

2 M

i) RS (Register Select, pin 4): RS = 0 selects the **command (instruction) register** — codes such as clear display, cursor position; RS = 1 selects the **data register** — the byte is an ASCII character to be displayed.

ii) E (Enable, pin 6): The LCD latches the data/command present on D0–D7 when a **high-to-low pulse** is applied on E. Without the E pulse the LCD ignores the data lines.

(Each pin 1 M)

f) Four applications of ARM microcontrollers.

2 M

1. Smartphones, tablets and smart watches (Cortex-A / Cortex-M).
2. Automotive ECUs — engine control, infotainment, ABS.
3. IoT devices and wearable / medical devices (fitness bands, glucose meters).
4. Industrial control, PLCs, routers, printers, single-board computers (Raspberry Pi).

(Any four, ½ M each)

g) Four features of Arduino Nano.

2 M

1. Microcontroller ATmega328P at 16 MHz; operating voltage 5 V; input 7–12 V on Vin pin.
2. 14 digital I/O pins (6 PWM) and 8 analog inputs A0–A7.
3. 32 KB Flash, 2 KB SRAM, 1 KB EEPROM.
4. Very small (about 18 × 45 mm), breadboard-friendly; mini-USB port for programming; no DC power jack.

(Any four, ½ M each)

Q.2

a) Selection criteria: flexibility, maintainability, reliability, time to prototype.

4 M

1. **Flexibility:** Ability to change the functionality of the system without a large NRE cost — e.g. re-programmable Flash, configurable firmware, spare I/O pins, modular hardware.
2. **Maintainability:** Ease of modifying, correcting and upgrading the system after release — good documentation, modular code, diagnostic ports, firmware / OTA update support.
3. **Reliability:** Ability of the system to work correctly for a long time without failure under given conditions (measured as MTBF). Watchdog timer, error checking and good-quality components improve reliability; critical in medical and automotive systems.
4. **Time to prototype:** Time needed to build a working version of the system to test the idea. Development boards (Arduino), simulators and libraries reduce it.

(Each criterion 1 M)

b) RISC vs CISC.

4 M

RISC (Reduced Instruction Set Computer)	CISC (Complex Instruction Set Computer)
Small number of simple instructions.	Large number of complex instructions.

Fixed-length instructions; few addressing modes.	Variable-length instructions; many addressing modes.
Most instructions execute in one clock cycle.	Instructions take several clock cycles.
Hard-wired control unit; pipelining is easy.	Micro-programmed control unit; pipelining is difficult.
Load-store architecture: only LOAD/STORE access memory; many general purpose registers.	Arithmetic instructions can operate directly on memory; fewer registers.
Larger program (code) size; simpler hardware, lower power.	Smaller code size; complex hardware.
Examples: AVR, ARM, PIC.	Examples: 8051, 8085, x86.

(Any four points, 1 M each)

c) RTOS characteristics.

4 M

- 1. Consistency:** The RTOS gives the same (deterministic) behaviour and timing every time the same task is executed — results and response times do not vary from run to run.
- 2. Reliability:** The RTOS must run for long periods (months/years) without failure or restart; it handles errors, provides watchdog support and protects critical resources.
- 3. Scalability:** The RTOS can be scaled up or down — only the required modules (scheduler, file system, networking) are included — so the same RTOS suits a tiny 8-bit controller or a large 32-bit system.
- 4. Predictability:** Worst-case time of every system call, interrupt latency and context switch is known and bounded, so it can be guaranteed that every task meets its deadline.

(Each 1 M)

d) Seven segment (common cathode) with Arduino; program 0–9.

4 M

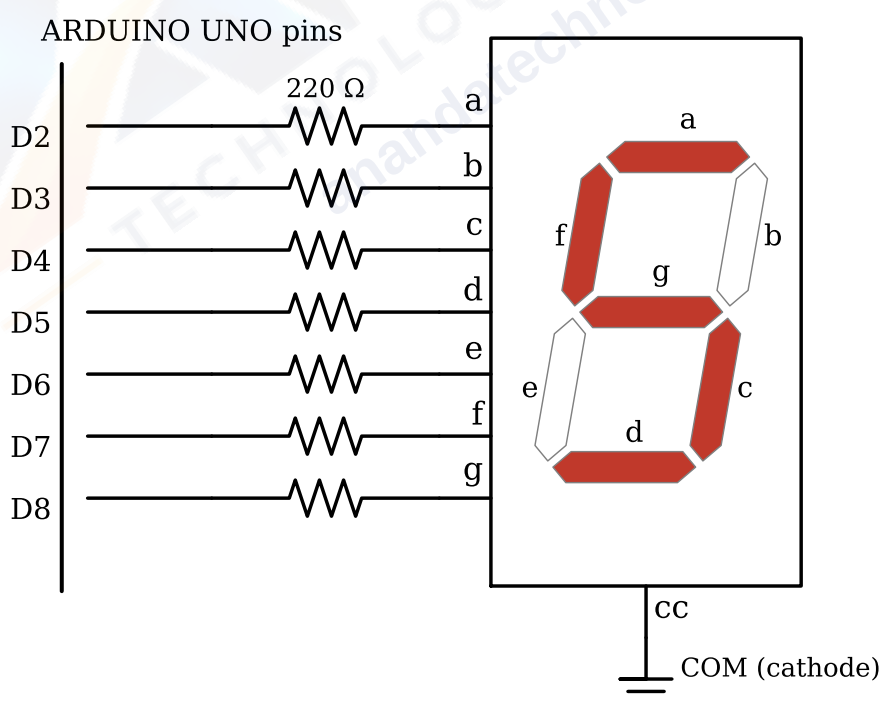


Fig.: Common-cathode seven segment display interfacing with Arduino

Segments a–g are connected to D2–D8 through 220 Ω current-limiting resistors; the common cathode goes to GND. A segment glows when its pin is HIGH.

```
// Common cathode 7-segment display: segments a..g on pins 2..8, count 0 to 9
const int segPin[7] = {2, 3, 4, 5, 6, 7, 8};          // a, b, c, d, e, f, g
// bit 0 = a ... bit 6 = g ; 1 = segment ON (common cathode)
const byte digitCode[10] = {
    0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F
};

void showDigit(int n) {
    for (int s = 0; s < 7; s++)
        digitalWrite(segPin[s], (digitCode[n] >> s) & 1);
}

void setup() {
    for (int s = 0; s < 7; s++) pinMode(segPin[s], OUTPUT);
    Serial.begin(9600);
}

void loop() {
    for (int n = 0; n <= 9; n++) {
        showDigit(n);
        Serial.print("Digit "); Serial.print(n);
        Serial.print("  code = 0x"); Serial.print(digitCode[n], HEX);
        Serial.print("  gfedcba = ");
        for (int s = 6; s >= 0; s--) Serial.print((digitCode[n] >> s) & 1);
        Serial.println();
        delay(1000);
    }
}
```

Output — segment codes printed by the program (compiled with g++ using Arduino stub headers; not from real hardware)

```
Digit 0  code = 0x3F  gfedcba = 0111111
Digit 1  code = 0x06  gfedcba = 0000110
Digit 2  code = 0x5B  gfedcba = 1011011
Digit 3  code = 0x4F  gfedcba = 1001111
Digit 4  code = 0x66  gfedcba = 1100110
Digit 5  code = 0x6D  gfedcba = 1101101
Digit 6  code = 0x7D  gfedcba = 1111101
Digit 7  code = 0x07  gfedcba = 0000111
Digit 8  code = 0x7F  gfedcba = 1111111
Digit 9  code = 0x6F  gfedcba = 1101111
```

(Diagram 2 M, program 2 M)

Q.3

a) Layered model of embedded system.

4 M

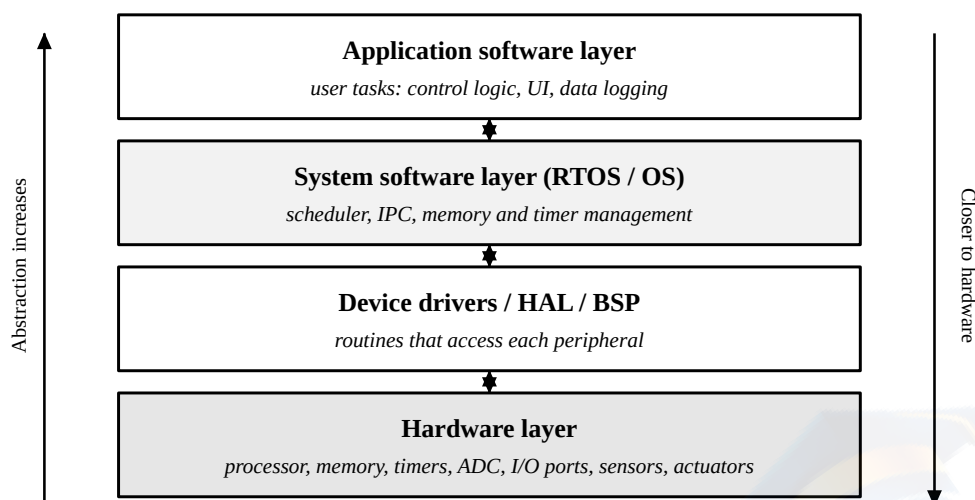


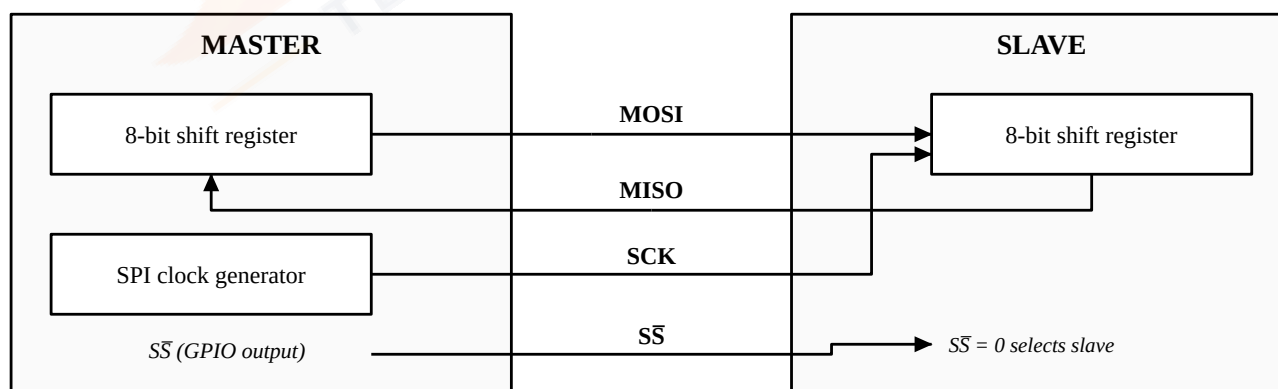
Fig.: Layered model of an embedded system

- **Hardware layer:** processor, memory, timers, ADC, I/O ports, sensors and actuators — the physical platform.
- **Device drivers / HAL / BSP layer:** low-level routines that initialise and access each peripheral (UART driver, ADC driver). It hides hardware details from upper layers and makes porting easy.
- **System software layer (RTOS / OS):** kernel services — task scheduling, inter-task communication, memory, timer and interrupt management. Very small systems may use a simple super-loop instead.
- **Application software layer:** user tasks that implement the actual function of the product — control algorithm, user interface, communication, data logging.

(Diagram 2 M, explanation 2 M)

b) SPI master–slave; SPI vs I²C.

4 M



In 8 clocks the master and slave exchange their bytes (full duplex)

Fig.: SPI master–slave configuration (shift registers form a ring)

- SPI (Serial Peripheral Interface, Motorola) is a synchronous, **full-duplex** serial bus with four lines: **SCK** (clock from master), **MOSI** (master out, slave in), **MISO** (master in, slave out) and **SS** (slave select, active LOW).
- The master selects a slave by making its $\overline{SS}$ LOW and generates clock pulses on SCK. On every clock one bit shifts out of the master shift register on MOSI and one bit shifts out of the slave on MISO — after 8 clocks the two bytes are exchanged.
- For more slaves, SCK/MOSI/MISO are common and each slave has a separate $\overline{SS}$ line. Clock polarity/phase (CPOL, CPHA) give 4 SPI modes.

SPI	I ² C
4 wires (SCK, MOSI, MISO, $\overline{SS}$) + one $\overline{SS}$ per slave.	2 wires (SDA, SCL) for all devices.
Full duplex; very fast (several MHz to tens of MHz).	Half duplex; 100 kbps / 400 kbps / 3.4 Mbps.
No device address, no acknowledgement.	7/10-bit addressing with ACK bit.
Single master.	Multi-master supported.

(Working with diagram 2 M, two differences 2 M)

c) Semaphore — binary and counting.

4 M

A **semaphore** is a kernel object (an integer variable with a waiting list) used for **synchronisation** between tasks / ISRs and for **mutual exclusion** on shared resources. Two atomic operations: **wait (P)** — if count > 0 decrement it, else block the task; **signal / give (V)** — increment count or wake one waiting task.

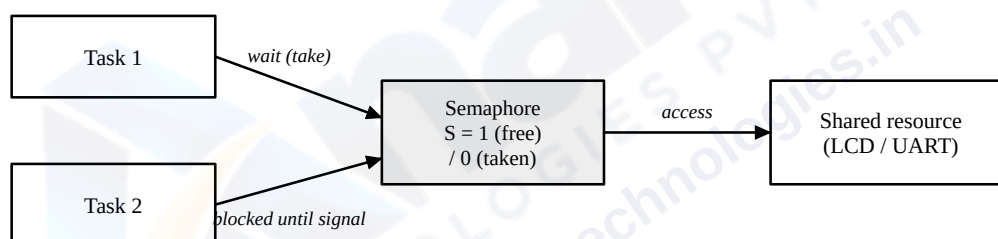


Fig.: Binary semaphore protecting a shared resource

- **Binary semaphore:** value is only 0 or 1. Used to protect a single resource or to signal an event. *Example:* only one task at a time may write to the LCD — a task takes the semaphore (1 → 0), writes, and gives it back (0 → 1); another task trying to take it is blocked. An ISR can give a binary semaphore to wake a task when ADC conversion completes.
- **Counting semaphore:** value can be 0 to N. Used when N identical resources exist or to count events. *Example:* 3 UART buffers — semaphore initialised to 3; each task takes one (3 → 2 → 1 → 0); the fourth task waits until one is given back.

(Definition 1 M, binary 1.5 M, counting 1.5 M)

d) Arduino Nano vs Arduino Mega.

4 M

Parameter	Arduino Nano	Arduino Mega 2560
Microcontroller	ATmega328P	ATmega2560
Digital I/O pins (PWM)	14 (6 PWM)	54 (15 PWM)

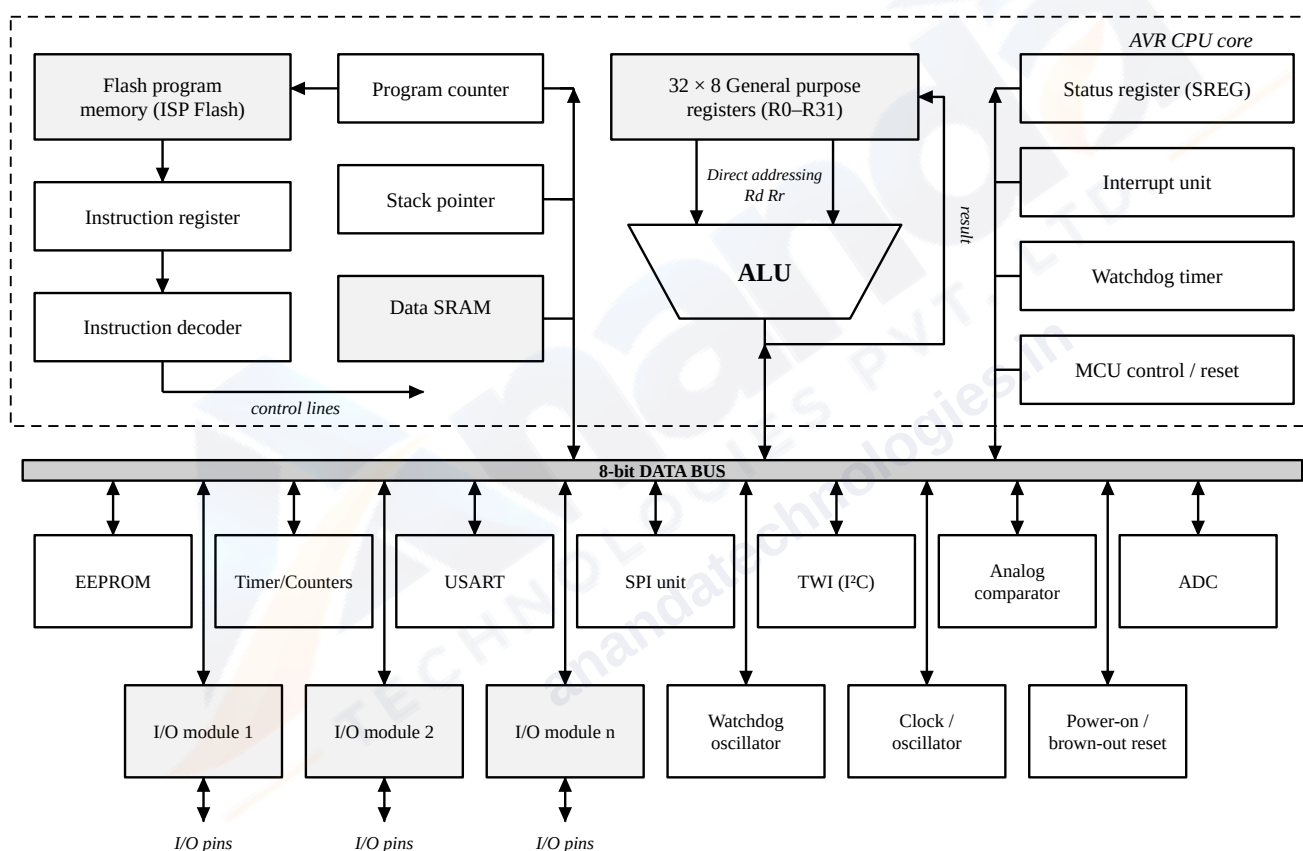
Analog inputs	8 (A0–A7)	16 (A0–A15)
Flash / SRAM / EEPROM	32 KB / 2 KB / 1 KB	256 KB / 8 KB / 4 KB
Hardware UARTs	1	4
Size	About 18 × 45 mm (breadboard friendly)	About 53 × 102 mm
USB / power jack	Mini-USB; no DC jack	USB-B; DC jack
Typical use	Small, compact projects, wearables	Large projects with many I/O: 3D printers, robots

(Any four points, 1 M each)

Q.4

a) AVR microcontroller architecture.

4 M



Harvard architecture: separate program and data buses; most instructions in one clock cycle

Fig.: Block diagram of AVR microcontroller architecture

- AVR is an 8-bit **RISC** microcontroller with **Harvard** architecture — separate memories and buses for program and data, so an instruction is fetched while the previous one executes (single-level pipeline); most instructions take one clock cycle.
- **Flash program memory** holds the program; the **program counter** addresses it; instructions go through the **instruction register** and **decoder** which generate control signals.

- **32 × 8 general purpose registers** are directly connected to the **ALU**; two operands are read and the result written back in one cycle. R26–R31 form the X, Y, Z 16-bit pointer registers. **SREG** holds flags (C, Z, N, V, S, H, T, I).
- **Data SRAM** and **EEPROM**, stack pointer, interrupt unit, watchdog timer and on-chip peripherals (timers, USART, SPI, TWI, ADC, analog comparator, I/O ports) are connected through the **8-bit data bus**.

(Diagram 2 M, description 2 M)

b) USB vs CAN.

4 M

Parameter	USB	CAN
Topology	Tiered star — host, hubs and devices	Linear bus (twisted pair) with 120 Ω termination
Bus control	Host-centric (single master); devices respond only to host	Multi-master; any node can transmit; priority by identifier
Speed	1.5 Mbps (LS), 12 Mbps (FS), 480 Mbps (USB 2.0), 5 Gbps+ (USB 3.x)	Up to 1 Mbps (CAN FD: data phase up to several Mbps)
Wires / signalling	4 wires: VBUS (5 V), D+, D–, GND; supplies power to devices	2 wires: CAN_H, CAN_L; differential, no power
Distance	About 5 m per cable	About 40 m at 1 Mbps; up to ~1 km at low speed
Number of devices	Up to 127 per host controller	Limited by transceiver load (typically up to ~110 nodes)
Applications	PC peripherals, programming / power of boards	Automobiles, industrial automation, medical equipment

(Any four points, 1 M each)

c) Priority-based pre-emptive scheduling with Gantt chart.

4 M

In **priority-based pre-emptive scheduling** every task has a priority. The scheduler always runs the **highest-priority ready task**. If a higher-priority task becomes ready, the running lower-priority task is pre-empted immediately (its context is saved) and resumes later. Used in most RTOS because urgent tasks get the CPU at once.

Working: t = 0: only T1 → runs. t = 1: T2 (priority 1) arrives → pre-empts T1. t = 2: T3 (priority 2) arrives but T2 is higher → T2 continues till 4. t = 4: T3 (2) is higher than T1 (3) → T3 runs 4–6. t = 6: T1 resumes and finishes at 9.

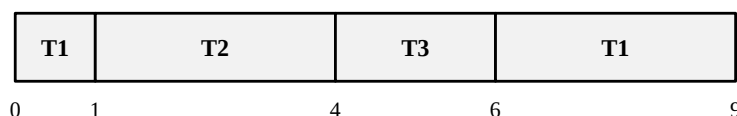


Fig.: Gantt chart (time in ms)

Task	Arrival	Burst	Priority	Completion	Turnaround	Waiting
------	---------	-------	----------	------------	------------	---------

T1	0	4	3	9	9	5
T2	1	3	1	4	3	0
T3	2	2	2	6	4	2

Average waiting time = $(5 + 0 + 2) / 3 = 2.33 \text{ ms}$

(Explanation 2 M, Gantt chart 2 M)

d) Stepper motor interfacing with Arduino using ULN2003.

4 M

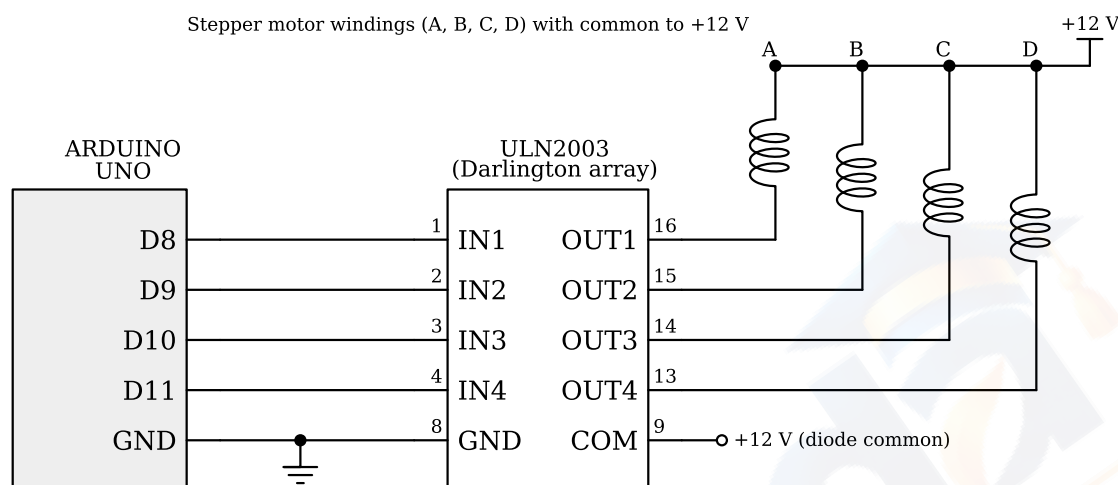


Fig.: Unipolar stepper motor interfacing with Arduino through ULN2003 driver

- A unipolar stepper motor (e.g. 1.8° per step $\rightarrow$ 200 steps/revolution) has four windings A, B, C, D with a common wire to the motor supply (+12 V).
- Arduino pins cannot give the coil current, so **ULN2003** (7 Darlington pairs, up to 500 mA each, built-in flyback diodes) is used. D8–D11 drive IN1–IN4; OUT1–OUT4 sink current from the windings; COM (pin 9) goes to +12 V and GND is common.
- Energising the windings in a fixed sequence rotates the rotor step by step; reversing the sequence reverses the direction; the delay between steps controls the speed.

Step	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Full-step (two-phase ON) sequence for clockwise rotation; read upwards for anticlockwise.

Program:

```
// Unipolar stepper motor through ULN2003: IN1..IN4 = pins 8, 9, 10, 11
// Full-step (two-phase ON) sequence; 1.8 deg motor -> 200 steps/rev
const int coil[4] = {8, 9, 10, 11};
const byte seq[4] = {0b1100, 0b0110, 0b0011, 0b1001}; // A B C D
int stepIndex = 0;

void stepMotor(int dir) { // dir = +1 clockwise, -1 anticlockwise
    stepIndex = (stepIndex + dir + 4) % 4;
```

```

for (int i = 0; i < 4; i++)
    digitalWrite(coil[i], (seq[stepIndex] >> (3 - i)) & 1);
}

void setup() {
    for (int i = 0; i < 4; i++) pinMode(coil[i], OUTPUT);
    Serial.begin(9600);
}

void loop() {
    for (int s = 0; s < 4; s++) {                // show first 4 steps
        stepMotor(+1);
        Serial.print("Step "); Serial.print(s + 1);
        Serial.print(" ABCD = ");
        for (int i = 3; i >= 0; i--) Serial.print((seq[stepIndex] >> i) & 1);
        Serial.println();
        delay(10);                                // step delay decides speed
    }
}

```

Output (compiled with g++ on a PC using Arduino/AVR stub headers and simulated inputs — not captured from real hardware)

```

Step 1 ABCD = 0110
Step 2 ABCD = 0011
Step 3 ABCD = 1001
Step 4 ABCD = 1100

```

(Diagram 2 M, explanation + sequence 2 M)

e) Zigbee and 6LoWPAN in low-power applications.

4 M

Both work on **IEEE 802.15.4** radios (2.4 GHz, 250 kbps), which are designed for low data rate and very low power battery-operated devices.

Application / ZDO	Application (CoAP / MQTT-SN)
APS (application support)	UDP
NWK — Zigbee network layer	IPv6 + 6LoWPAN adaptation
IEEE 802.15.4 MAC	IEEE 802.15.4 MAC
IEEE 802.15.4 PHY	IEEE 802.15.4 PHY

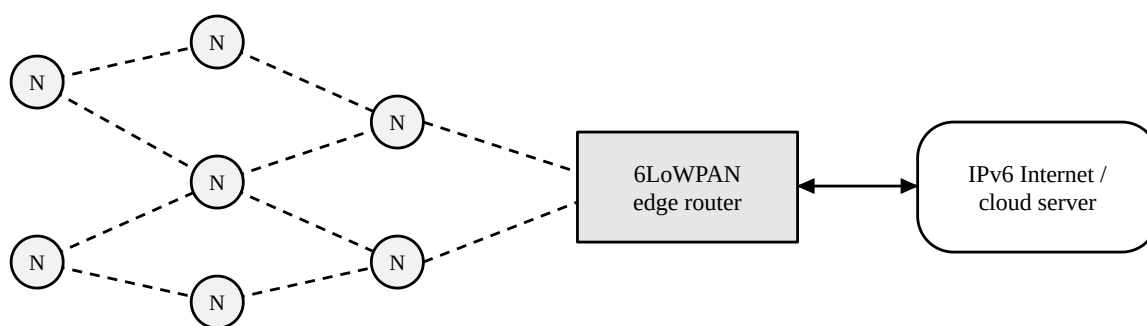
Zigbee stack

6LoWPAN stack

Why Zigbee: very low power (end devices sleep most of the time; battery life of years), mesh networking with routers gives long coverage and self-healing, up to 65,000 nodes, AES-128 security. *Examples:* smart lighting and home automation, smart energy meters, wireless sensor networks in industry.

Why 6LoWPAN: carries **IPv6** packets over 802.15.4 by header compression (40-byte IPv6 header reduced to a few bytes) and fragmentation (127-byte frames), so each tiny sensor gets its own IP address and can talk directly to Internet/cloud through an edge router, with low power. *Examples:*

smart street lighting, building automation (Thread-based devices), environmental monitoring sensors.



IEEE 802.15.4 low-power mesh (sensor nodes, battery operated)

Fig.: 6LoWPAN network — IPv6 sensor nodes reach the Internet through an edge router

(Zigbee 2 M, 6LoWPAN 2 M)

Q.5

a) Watchdog timer — purpose, block diagram, use in AVR.

6 M

Purpose: The watchdog timer (WDT) protects the system from software failure. If the program hangs (endless loop, corrupted program counter, noise), the WDT is not refreshed, it times out and **resets the microcontroller**, so the system restarts automatically.

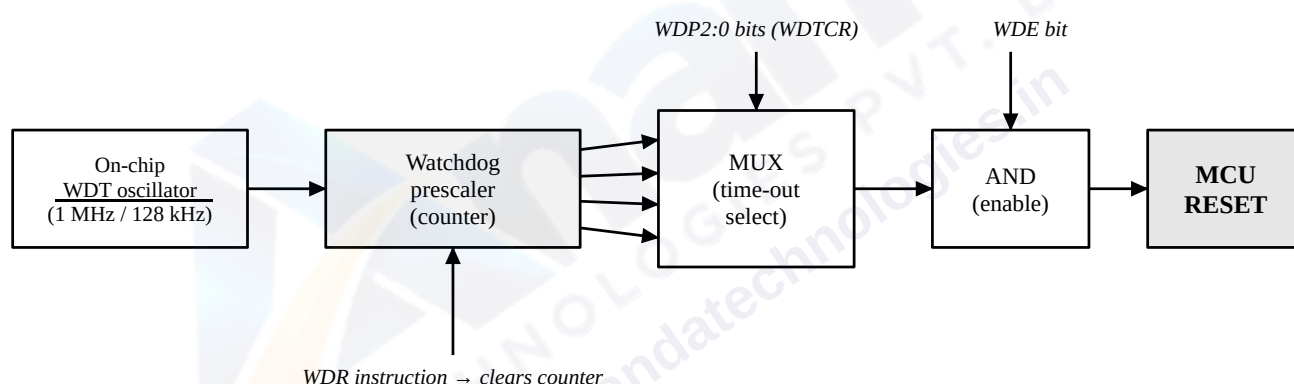


Fig.: Watchdog timer of AVR (ATmega8 / ATmega328)

Working in AVR (ATmega8):

- The WDT is clocked from a separate **on-chip oscillator** (≈ 1 MHz in ATmega8; 128 kHz in ATmega328P), so it works even if the main clock fails.
- The prescaler counts the oscillator cycles. Bits **WDP2:0** in register **WDTCR** select the time-out from 16K cycles (≈ 16 ms) to 2048K cycles (≈ 2.1 s at 5 V).
- Setting **WDE** = 1 enables the watchdog. The program must execute the **WDR** (watchdog reset) instruction before the time-out; this clears the counter.
- If WDR is not executed in time (program stuck), the WDT generates a system reset; the WDRF flag in MCUCSR shows that the reset came from the watchdog. To disable it, a timed sequence using WDCE and WDE is needed (protection against accidental disabling).

```
#include <avr/wdt.h>
wdt_enable(WDTO_2S);      /* enable watchdog, time-out about 2 s      */
while (1) {
    read_sensors();
    update_outputs();
    wdt_reset();           /* WDR instruction - 'kick the dog'      */
}
```

(Purpose 1 M, diagram 2 M, working 3 M)

b) Deadlock — RAG, conditions, prevention.

6 M

Deadlock: A set of tasks is in deadlock when every task in the set is waiting for a resource that is held by another task of the same set, so none can continue.

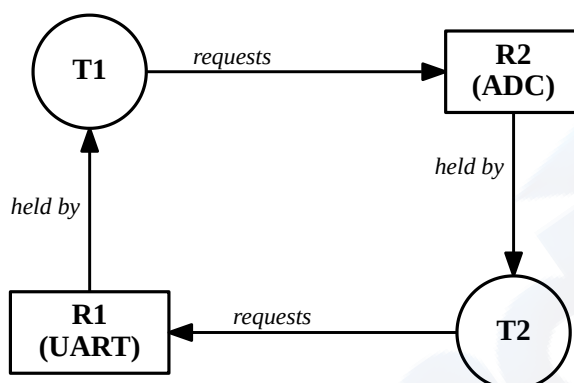


Fig.: Resource allocation graph showing deadlock (circular wait)

In the resource allocation graph, R1 (UART) is held by T1, T1 requests R2 (ADC), R2 is held by T2 and T2 requests R1 — a closed cycle, so T1 and T2 are deadlocked.

Necessary (Coffman) conditions:

1. **Mutual exclusion:** a resource can be used by only one task at a time.
2. **Hold and wait:** a task holds one resource and waits for another.
3. **No pre-emption:** a resource cannot be taken forcibly from a task; it is released only voluntarily.
4. **Circular wait:** a circular chain of tasks exists, each waiting for a resource held by the next.

Prevention strategies (break any one condition):

- **Avoid hold and wait:** a task must request all required resources at once before it starts, or release what it holds before asking for more.
- **Avoid circular wait — resource ordering:** number all resources and require every task to acquire them only in increasing order (e.g. always UART before ADC). Then a cycle can never form.
- **Allow pre-emption / time-outs:** if a task cannot get a resource within a time-out, it releases the resources it holds and retries later.
- **Priority ceiling / mutex protocols** in RTOS also avoid deadlock and priority inversion.

(Definition + RAG 2 M, conditions 2 M, two strategies 2 M)

c) LM35 temperature on 16×2 LCD — ATmega328 (Arduino Uno).

6 M

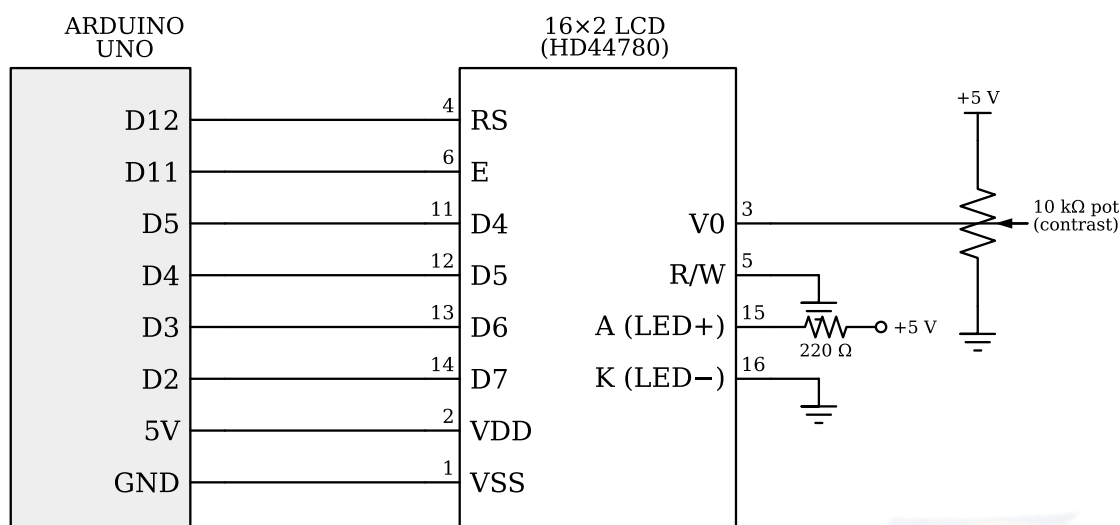


Fig.: 16×2 LCD (HD44780) interfacing with Arduino Uno in 4-bit mode

LM35: +Vs → 5 V, Vout → A0, GND → GND (as below).

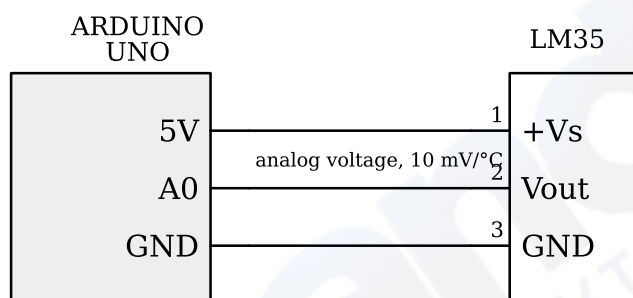


Fig.: Interfacing LM35 temperature sensor with Arduino Uno

Formula: $T (^{\circ}\text{C}) = \text{ADC} \times (5 \text{ V} / 1023) / 10 \text{ mV} = \text{ADC} \times 500 / 1023$.

Program:

```
// LM35 temperature displayed on 16x2 LCD (4-bit mode)
#include <LiquidCrystal.h>
// LCD pins:      RS  E  D4 D5 D6 D7
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
const int lm35 = A0;

void setup() {
    lcd.begin(16, 2);           // 16 columns, 2 rows
    lcd.print("Temp Monitor");
}

void loop() {
    int raw = analogRead(lm35);
    float tempC = raw * 500.0 / 1023.0;    // (raw x 5 V / 1023) / 10 mV
    lcd.setCursor(0, 1);                  // column 0, row 1
    lcd.print("T = ");
    lcd.print(tempC, 1);
    lcd.print((char)223);                  // degree symbol in HD44780 ROM
    lcd.print("C  ");
    delay(1000);
}
```

LCD contents after each reading (compiled with g++ using Arduino/LiquidCrystal stub headers; ADC values 65 and 78 simulated — not from real hardware)

```

+-----+
|Temp Monitor|
|T = 31.8°C  |
+-----+
+-----+
|Temp Monitor|
|T = 38.1°C  |
+-----+

```

(Diagram 2 M, program 4 M)

Q.6

a) Sleep modes of AVR (ATmega328).

6 M

Sleep modes reduce power by switching off unused clock domains. The mode is selected by bits **SM2:0** in **SMCR** (MCUCR in ATmega8), sleep is enabled by bit **SE** and entered by the **SLEEP** instruction. An enabled interrupt or reset wakes the MCU.

SM2:0	Sleep mode	What keeps running	Wake-up sources
000	Idle	All peripherals (timers, USART, SPI, ADC, TWI, WDT); only CPU and Flash clocks stopped	Any interrupt
001	ADC Noise Reduction	ADC, external interrupts, TWI address match, Timer2 (async), WDT	ADC complete, external / pin-change interrupts, TWI, Timer2, WDT
010	Power-down	Only external interrupts, TWI address match and WDT; main oscillator stopped	INT0/INT1 level, pin change, TWI address match, WDT, reset
011	Power-save	As power-down + Timer2 running asynchronously (32.768 kHz crystal)	As power-down + Timer2 interrupts
110	Standby	As power-down but main oscillator kept running	As power-down; wakes in 6 clock cycles
111	Extended Standby	As power-save but main oscillator kept running	As power-save; fast wake-up

- Idle mode:** CPU stops but all peripherals keep working; the MCU wakes on any interrupt (e.g. UART receive, timer overflow). Used when the CPU only waits for an interrupt — saves CPU power with very fast wake-up.
- ADC Noise Reduction mode:** CPU and I/O clocks are stopped, but the ADC keeps converting. Digital noise is reduced, so ADC readings are more accurate; the ADC-complete interrupt wakes the CPU.
- Power-down mode:** the external oscillator is stopped — lowest current (below 1 μ A with WDT off). Only asynchronous sources (external level / pin-change interrupt, TWI address match, watchdog) can wake it. Used in battery products that wake on a key press.

4. **Power-save mode:** like power-down, but Timer2 keeps running from a 32.768 kHz watch crystal, so the device can keep real time and wake up periodically.

(List of modes 2 M, any three explained 4 M)

b) IEC 61850 GOOSE — transmission, publisher–subscriber, significance.

6 M

IEC 61850 is the international standard for communication in electrical substations. **GOOSE** (Generic Object Oriented Substation Event) is its fast, event-driven message service used between IEDs for protection and control.

Message transmission:

- GOOSE messages are mapped **directly on Ethernet (layer 2)** with EtherType 0x88B8 — no TCP/IP — to avoid delay. IEEE 802.1Q VLAN tag with high priority is used so that switches forward them first.
- Each message carries a **dataset** (e.g. trip, breaker position, interlock status), a **state number (stNum)** that increases on every change and a **sequence number (sqNum)** for repeats, and a **time-allowed-to-live** value.
- When an event occurs the message is sent immediately and repeated at short, increasing intervals (fast retransmission), then repeated periodically as a **heartbeat (T0)**. Subscribers detect link failure if the heartbeat stops. Trip messages must reach in a few milliseconds.

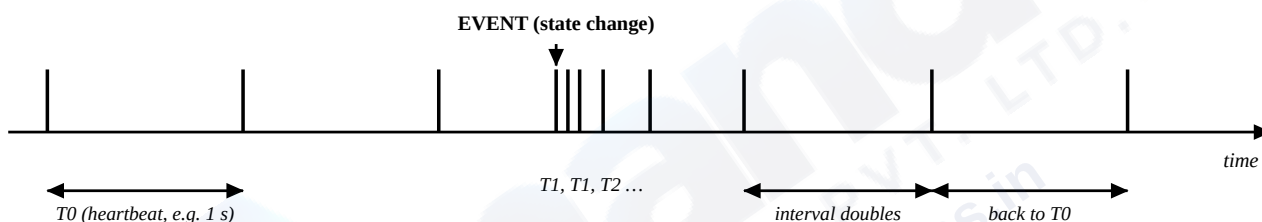
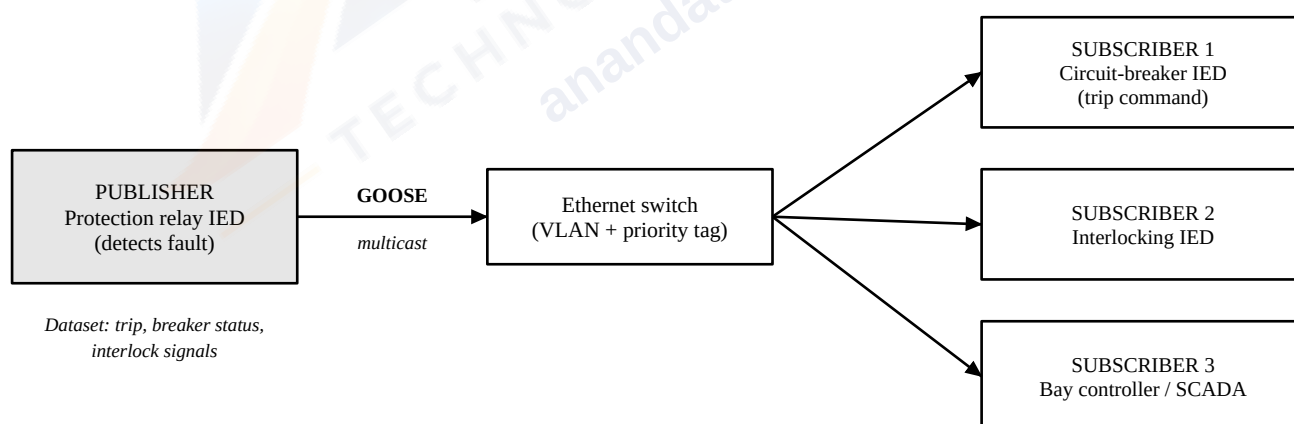


Fig.: GOOSE retransmission — fast repeats after an event, interval doubling back to heartbeat T0

Publisher–subscriber model:



One message from the publisher reaches every subscriber that has subscribed to that dataset; no handshake / ACK

Fig.: GOOSE publisher–subscriber model on the IEC 61850 station/process bus

- The **publisher** IED multicasts the message once; it does not know or wait for receivers.

- **Subscribers** are configured (in the SCL configuration file) to accept particular GOOSE control blocks and act on them, e.g. a breaker IED trips.

Significance in substation automation: fast protection (trip, breaker-failure, reverse blocking); replaces hard-wired copper cables between relays — less cost and wiring; interoperability of IEDs from different vendors; continuous self-supervision of the communication path; easy changes by reconfiguration instead of rewiring.

(Transmission 2 M, publisher-subscriber 2 M, significance 2 M)

c) Classification based on functional requirements.

6 M

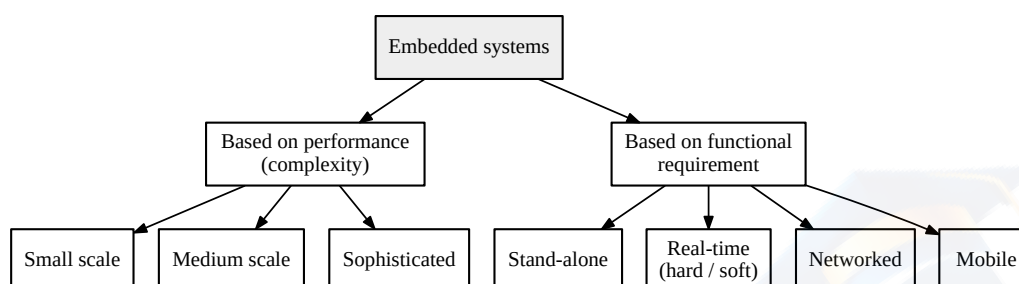


Fig.: Classification of embedded systems

1. **Stand-alone embedded systems:** Work on their own without a host computer. Take inputs, process them and give outputs/display. *Example:* digital camera, microwave oven, MP3 player, washing machine.
2. **Real-time embedded systems:** Must give the output within a specified time (deadline). **Hard real-time** — missing a deadline is a failure (*example:* airbag controller, pacemaker). **Soft real-time** — occasional miss only reduces quality (*example:* video streaming set-top box).
3. **Networked embedded systems:** Connected to a wired or wireless network (LAN, WAN, Internet) to share data or be controlled remotely. *Example:* IP camera, home automation hub, ATM, smart energy meter.
4. **Mobile embedded systems:** Portable, battery-powered, small size and memory, low power. *Example:* mobile phone, smart watch, fitness band, PDA.

(Diagram / classification 2 M, each type with example 1 M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in