

Embedded System (315338)

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

Q.1

a) Four applications of embedded systems.

2 M

1. **Consumer electronics:** washing machine, microwave oven, digital camera, smart TV.
2. **Automobile:** engine control unit, ABS, airbag controller, infotainment.
3. **Medical:** pacemaker, ECG monitor, infusion pump, glucose meter.
4. **Industrial / communication:** PLC, robot controller, router, mobile phone, ATM, smart energy meter.

(Any four, ½ M each)

b) Two protocols supported by AVR.

2 M

- **USART (UART):** asynchronous / synchronous serial communication with programmable baud rate; full duplex using TxD and RxD pins (used for Serial Monitor on Arduino).
- **SPI:** high-speed synchronous full-duplex serial interface (MOSI, MISO, SCK, $\overline{SS}$); also used for in-system programming of Flash.
- **TWI (I²C):** two-wire (SDA, SCL) addressed bus for RTC, EEPROM, sensors; up to 400 kHz.

(Any two with feature, 1 M each)

c) Need of watchdog timer.

2 M

- Embedded systems usually run unattended; a software bug, endless loop or electrical noise may hang the program and nobody is there to press reset.
- The watchdog timer must be refreshed regularly by the program; if the program hangs, the WDT times out and **automatically resets** the microcontroller, so the system recovers and reliability increases.

(2 M)

d) Four features of Arduino Mega.

2 M

1. Microcontroller ATmega2560 at 16 MHz; operating voltage 5 V; input 7–12 V.
2. 54 digital I/O pins (15 PWM) and 16 analog inputs.
3. 256 KB Flash (8 KB boot-loader), 8 KB SRAM, 4 KB EEPROM.
4. 4 hardware UARTs, USB-B connector, DC jack, ICSP header — suitable for large projects (3D printers, robots).

(Any four, ½ M each)

e) I²C vs SPI.

2 M

I ² C	SPI
2 lines: SDA, SCL.	4 lines: SCK, MOSI, MISO, $\overline{SS}$.

Half duplex; 100 / 400 kbps (up to 3.4 Mbps).	Full duplex; several MHz.
Slaves selected by 7/10-bit address; ACK after each byte.	Slave selected by separate $\overline{SS}$ line; no ACK.
Multi-master possible.	Single master.

(Any two, 1 M each)

f) `analogRead()` and `analogWrite()`.

2 M

`analogRead(pin)`: Reads the voltage on an analog input pin (A0–A5) using the 10-bit ADC and returns 0 to 1023 for 0 to 5 V. Example: `int v = analogRead(A0);`

`analogWrite(pin, value)`: Generates a PWM signal on a PWM pin (3, 5, 6, 9, 10, 11 on Uno) with duty cycle value/255 (0–255); used for LED brightness or motor speed. Example: `analogWrite(9, 128);` gives $\approx 50\%$ duty.

(Each 1 M)

g) Define deadlock; four conditions.

2 M

Deadlock: a situation in which two or more tasks wait indefinitely for resources held by each other, so none of them can proceed.

Conditions: (1) Mutual exclusion (2) Hold and wait (3) No pre-emption (4) Circular wait.

(Definition 1 M, conditions 1 M)

Q.2

a) Advantages of PIC over 8051.

4 M

- RISC architecture:** PIC (mid-range) has only about 35 simple instructions against 111 in 8051 (CISC) — easier to learn and faster to execute.
- Higher speed:** PIC needs 4 clock cycles per instruction cycle and most instructions take one instruction cycle; a standard 8051 needs 12 clock cycles per machine cycle.
- More on-chip peripherals:** PIC has built-in ADC, PWM (CCP), EEPROM, comparators, I²C/SPI (MSSP) and USART; the basic 8051 has none of ADC, PWM or EEPROM.
- Harvard architecture with wider instruction word:** separate 14-bit program bus and 8-bit data bus allow single-word instructions and pipelining (fetch + execute overlap).
- Other features:** in-circuit serial programming (ICSP), Flash program memory, watchdog timer, brown-out reset, low-power sleep mode, high sink/source current (25 mA) per pin.

(Any four, 1 M each)

b) GPOS vs RTOS.

4 M

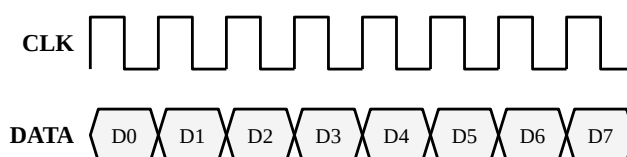
General purpose OS (GPOS)	Real time OS (RTOS)
Designed for throughput and fairness among many applications.	Designed to meet deadlines of time-critical tasks.
Scheduling is fairness / time-sharing based; high-priority task may be delayed.	Priority-based pre-emptive, deterministic scheduling.
Response time and interrupt latency are not predictable.	Response time and latency are bounded and predictable.

Large size; needs more memory, disk, virtual memory.	Small and scalable; fits in a few KB of ROM/RAM.
Used in PCs, servers, laptops.	Used in embedded / control systems.
Examples: Windows, Linux, macOS.	Examples: FreeRTOS, VxWorks, QNX, μ C/OS-II.

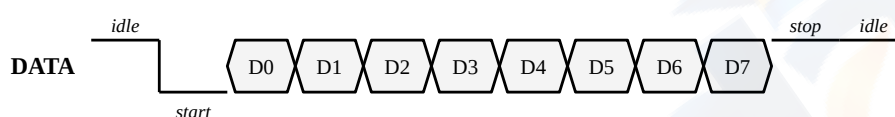
(Any four points, 1 M each)

c) Synchronous and asynchronous communication with waveforms.

4 M



a) Synchronous: common clock line, continuous block of data



b) Asynchronous: no clock, each character framed by start and stop bits

- **Synchronous:** transmitter and receiver are synchronised by a common clock sent on a separate line (e.g. SPI SCK, I²C SCL). Data bits are sampled on clock edges; a block of data is sent continuously without start/stop bits — high speed, low overhead. Examples: SPI, I²C.
- **Asynchronous:** no clock line; both sides agree on the baud rate. Each character is framed by a **start bit (0)**, 5–8 data bits (LSB first), optional parity and **stop bit(s) (1)**. The receiver re-synchronises on each start bit. Simpler but slower due to framing overhead. Example: UART / RS-232.

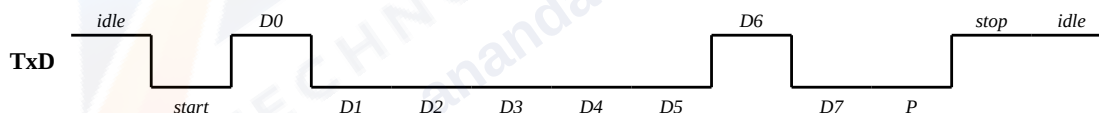


Fig.: Asynchronous (UART) frame for data 0x41 (LSB first, even parity = 0)

(Explanation 2 M, waveforms 2 M)

d) 16×2 LCD interfacing in 4-bit mode; control pins.

4 M

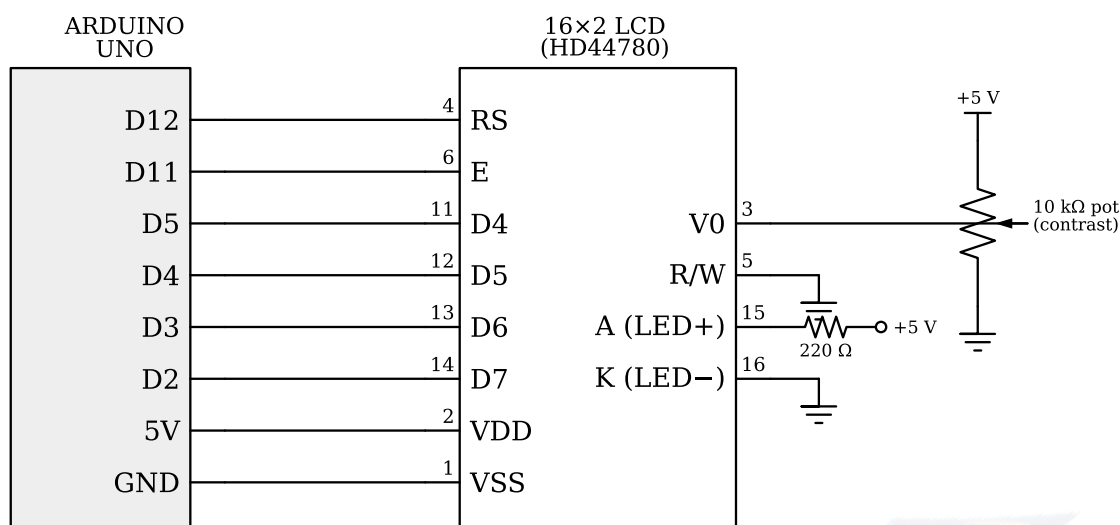


Fig.: 16×2 LCD (HD44780) interfacing with Arduino Uno in 4-bit mode

- Data lines D4–D7 of the LCD are connected to Arduino D5, D4, D3, D2 (4-bit mode — each byte is sent as two nibbles; D0–D3 unused). RS → D12, E → D11.
- **RS (pin 4):** 0 = command register, 1 = data register.
- **R/W (pin 5):** 0 = write to LCD, 1 = read; tied to GND because we only write.
- **E (pin 6):** enable — a high-to-low pulse latches the data/command.
- V0 (pin 3) through a 10 kΩ pot adjusts contrast; pins 15/16 power the backlight (with 220 Ω).

Program:

```
// 16x2 LCD in 4-bit mode with Arduino Uno: display a message and a counter
#include <LiquidCrystal.h>
//          RS  E  D4 D5 D6 D7
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int count = 0;

void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("MSBTE  K-SCHEME");
}

void loop() {
  lcd.setCursor(0, 1);
  lcd.print("Count: ");
  lcd.print(count);
  count++;
  delay(1000);
}
```

LCD contents after two passes of loop() (compiled with g++ using Arduino/LiquidCrystal stub headers — not from real hardware)

```
+-----+
|MSBTE  K-SCHEME |
|Count: 0         |
+-----+
```

```

+-----+
| MSBTE  K-SCHEME |
| Count: 1         |
+-----+

```

(Diagram 2 M, control pins 2 M)

Q.3

a) ATmega8 vs ATmega328.

4 M

Parameter	ATmega8	ATmega328 (P)
Flash program memory	8 KB	32 KB
SRAM	1 KB	2 KB
EEPROM	512 bytes	1 KB
Maximum speed	16 MHz (ATmega8L: 8 MHz)	20 MHz
Operating voltage	4.5–5.5 V (ATmega8L: 2.7–5.5 V)	1.8–5.5 V
PWM channels	3	6
Pin-change interrupts	Not available (only INT0, INT1)	Available on all I/O pins
Watchdog	1 MHz oscillator, reset only, up to ≈ 2.1 s	128 kHz oscillator, interrupt and/or reset, up to 8 s
Sleep modes	5: Idle, ADC noise reduction, Power-down, Power-save, Standby	6: the same + Extended Standby (picoPower in 328P)
Common features	23 I/O, 32 registers, 2×8 -bit + 1×16 -bit timers, USART, SPI, TWI, 10-bit ADC	Same peripheral set — pin compatible (28-pin PDIP)

(Memory 1 M, speed 1 M, peripherals 1 M, sleep modes 1 M)

b) Characteristics: CPU type, speed, processing power, memory, performance.

4 M

- 1. CPU type:** the processor used — 8-bit / 16-bit / 32-bit microcontroller, microprocessor, DSP or SoC (e.g. AVR, PIC, ARM Cortex-M). It decides the instruction set, peripherals, cost and power.
- 2. Maximum CPU speed:** the highest clock frequency at which the CPU can work (e.g. ATmega328P — 20 MHz, ARM Cortex-M4 — 168 MHz). Higher speed executes instructions faster but increases power consumption.
- 3. Processing power:** the amount of work the CPU can do per second, expressed in MIPS / DMIPS; depends on word length, clock speed, pipelining and special units (multiplier, FPU, DSP instructions).
- 4. Memory:** type and size of program memory (ROM/Flash) and data memory (RAM, EEPROM), on-chip or external. Embedded systems have limited memory, so code must be compact.

5. **Performance:** overall ability to meet the requirement — throughput, response time (latency), real-time deadlines, power efficiency and reliability.

(Any four explained, 1 M each)

c) Interrupt handling and resource allocation in RTOS.

4 M

Interrupt handling:

- When an interrupt occurs, the CPU saves the context and the RTOS calls the corresponding **ISR** (interrupt service routine) through the vector table.
- ISRs are kept very short; heavy processing is **deferred** to a task — the ISR only gives a semaphore or posts a message to a queue, which wakes a high-priority task.
- RTOS supports interrupt priorities and nesting and keeps **interrupt latency** small and bounded; after the ISR the scheduler may switch to a newly-ready higher-priority task.

Resource allocation:

- The RTOS allocates CPU time (scheduler), memory and shared devices (UART, LCD, ADC) to tasks.
- Memory is allocated from **fixed-size block pools** (deterministic, no fragmentation) instead of general malloc.
- Shared resources are protected by **mutexes / semaphores**; priority-inheritance or priority-ceiling protocols prevent priority inversion and deadlock.

(Interrupt handling 2 M, resource allocation 2 M)

d) HC-SR04 ultrasonic sensor interfacing.

4 M

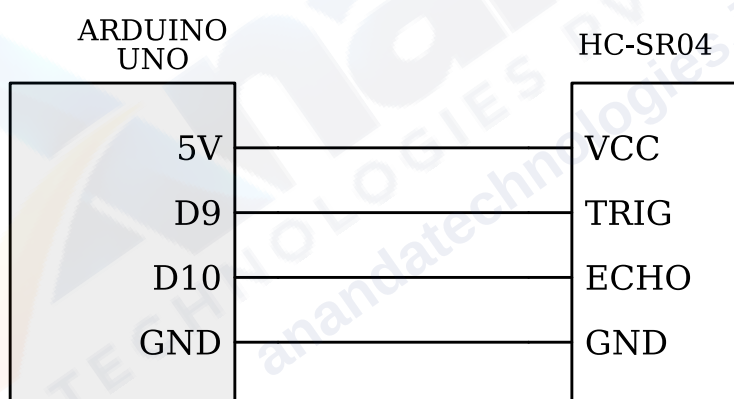


Fig.: HC-SR04 ultrasonic sensor interfacing with Arduino Uno

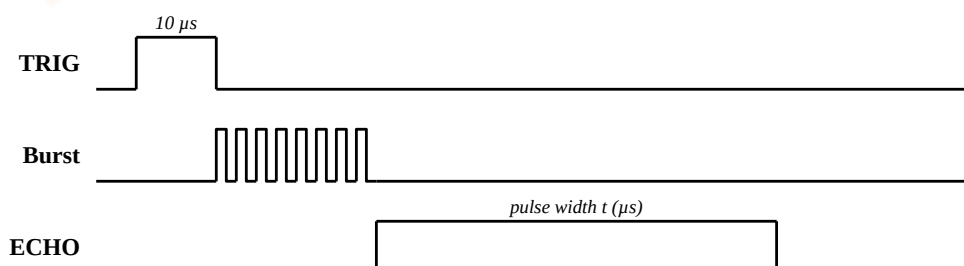


Fig.: HC-SR04 timing — 10 μ s trigger, 8-cycle 40 kHz burst, echo width $\propto$ distance

- Arduino sends a 10 μs HIGH pulse on **TRIG**. The module transmits a burst of 8 ultrasonic pulses at 40 kHz.
- The **ECHO** pin goes HIGH and stays HIGH until the reflected sound returns; its width t is the time of travel to the object and back. `pulseIn(echoPin, HIGH)` measures t in μs .
- **Distance** = (speed of sound $\times$ time) / 2 = $t (\mu\text{s}) \times 0.0343 \text{ cm}/\mu\text{s} \div 2$ (speed of sound $\approx 343 \text{ m/s}$ at 20 $^{\circ}\text{C}$). Range $\approx 2 \text{ cm}$ to 4 m.

Program:

```
// HC-SR04 ultrasonic sensor: TRIG = pin 9, ECHO = pin 10
const int trigPin = 9, echoPin = 10;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);          // 10 us trigger pulse
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH); // echo high time in us
  float distance = duration * 0.0343 / 2; // speed of sound 343 m/s
  Serial.print("Echo time = ");
  Serial.print(duration);
  Serial.print(" us   Distance = ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(500);
}
```

Output (compiled with g++ using Arduino stub headers; echo widths 583, 1166, 2915 μs simulated — not from real hardware)

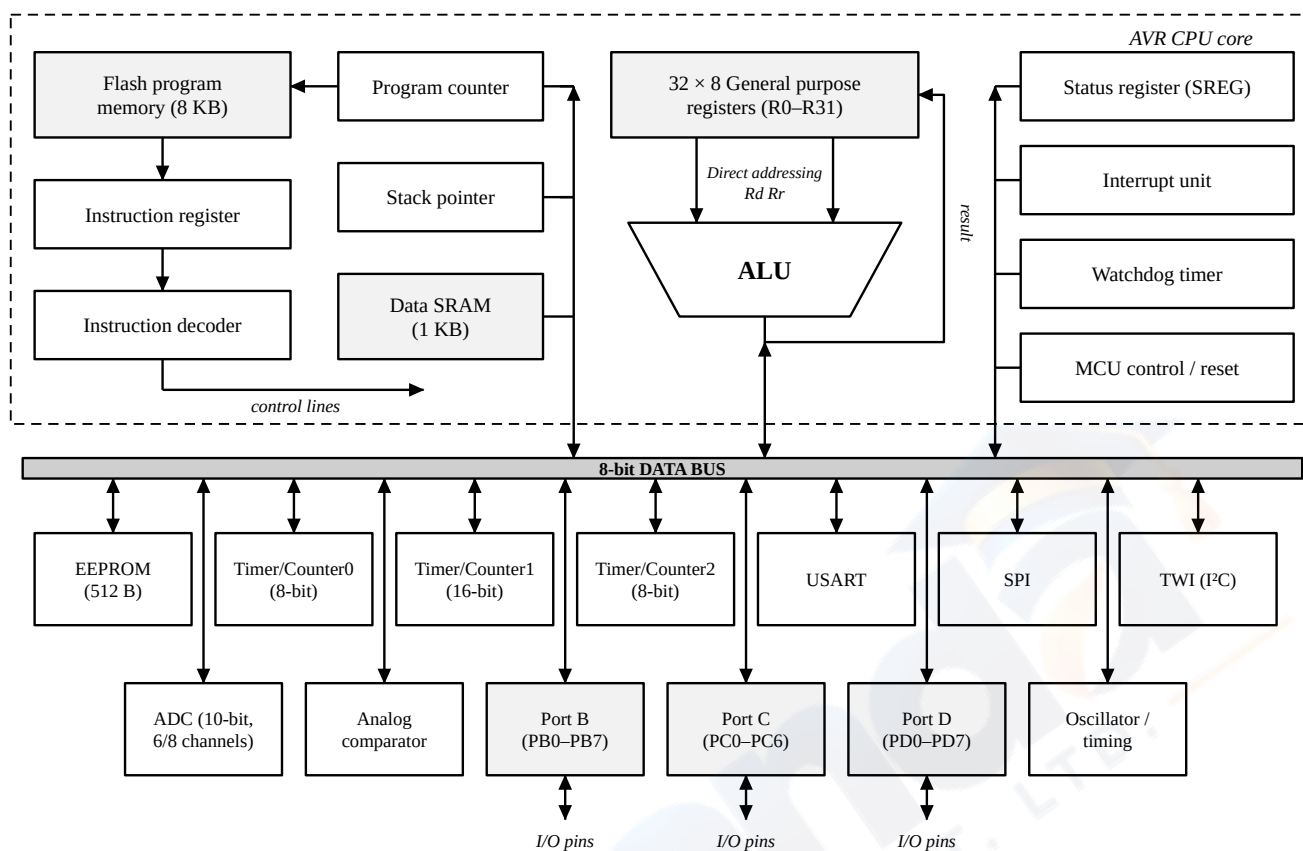
Echo time = 583 us	Distance = 10.00 cm
Echo time = 1166 us	Distance = 20.00 cm
Echo time = 2915 us	Distance = 49.99 cm

(Diagram 1 M, timing 1 M, explanation + formula 2 M)

Q.4

a) ATmega8 block diagram — memory, I/O port and timer blocks.

4 M



Program memory and data memory have separate buses (Harvard architecture)

Fig.: Internal architecture (block diagram) of ATmega8

- **Memory blocks:** 8 KB Flash for program (10,000 write/erase cycles, in-system programmable, boot section); 1 KB internal SRAM for variables and stack; 512 bytes EEPROM for data that must be kept after power off; 32 general purpose registers and 64 I/O registers mapped in data space.
- **I/O port blocks:** Port B (PB0–PB7), Port C (PC0–PC6), Port D (PD0–PD7). Each pin is bidirectional with DDRx (direction), PORTx (output / pull-up) and PINx (input) registers; pins also have alternate functions (ADC, USART, SPI, TWI, INT0/1, timer pins).
- **Timer blocks:** Timer/Counter0 — 8-bit, overflow interrupt, external count input T0; Timer/Counter1 — 16-bit, two output compare units, input capture, PWM; Timer/Counter2 — 8-bit with compare/PWM and asynchronous operation from a 32 kHz crystal (RTC). Watchdog timer has its own oscillator.

(Diagram 2 M, description 2 M)

b) USB — topology, features, transfer types.

4 M

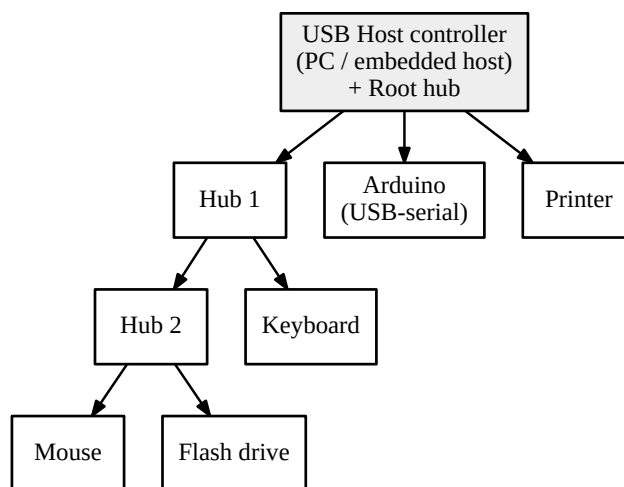


Fig.: USB tiered-star topology (one host, hubs, up to 127 devices)

- **Topology:** tiered star — one host controller with root hub; hubs add ports; up to 127 devices (7-bit address); host controls all transfers (polling).
- **Features:** 4-wire cable — VBUS (+5 V), D+, D-, GND; differential NRZI signalling; plug-and-play and hot-swapping with automatic enumeration; bus power to devices (500 mA in USB 2.0); speeds 1.5 Mbps (low), 12 Mbps (full), 480 Mbps (high), 5 Gbps+ (USB 3.x).
- **Packets:** token (SETUP, IN, OUT, SOF), data (DATA0/DATA1) and handshake (ACK, NAK, STALL).

Transfer type	Use
Control	Configuration / enumeration commands (all devices)
Bulk	Large data, no time guarantee, error-checked — printers, flash drives
Interrupt	Small data with guaranteed polling interval — keyboard, mouse
Isochronous	Real-time streaming at constant rate, no retry — audio, webcam

(Topology 1 M, features 1 M, transfer types 2 M)

c) Round Robin with priority scheduling.

4 M

Round Robin with priority: tasks are grouped by priority. The scheduler always selects the highest-priority ready group; **tasks of the same priority share the CPU in round-robin** order with a fixed time slice. A higher-priority task that becomes ready pre-empts the running task. It combines urgency (priority) with fairness (time slicing) among equal tasks.

Working: 0–2: T1 (slice), 2–4: T2 (slice). At 4 ms T3 (priority 1) arrives and runs 4–6 to completion. Then T1 and T2 (priority 2) continue in round robin: T1 6–8 (finishes), T2 8–9 (finishes).

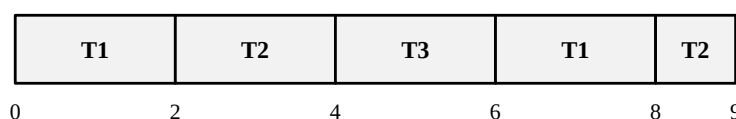


Fig.: Gantt chart (time in ms)

Task	Priority	Arrival	Burst	Completion	Turnaround	Waiting
T1	2	0	4	8	8	4
T2	2	0	3	9	9	6
T3	1	4	2	6	2	0

Average waiting time = $(4 + 6 + 0) / 3 = 3.33$ ms

(Explanation 2 M, Gantt chart 2 M)

d) Zigbee — device types and topologies.

4 M

Zigbee is a low-power, low-data-rate wireless mesh protocol built on IEEE 802.15.4 (2.4 GHz, 250 kbps, 10–100 m), maintained by the Zigbee Alliance (CSA). It supports up to 65,000 nodes with AES-128 security.

- **Coordinator (C):** one per network; starts the network, selects channel and PAN ID, stores security keys; always powered.
- **Router (R):** relays data from other nodes, extends range, allows devices to join; mains powered.
- **End device (E):** simple battery-powered sensor/actuator; talks only to its parent; sleeps most of the time to save power.

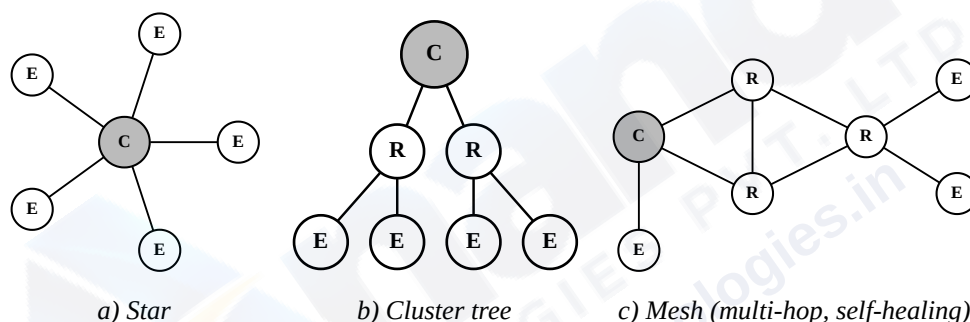


Fig.: Zigbee network topologies (C = coordinator, R = router, E = end device)

Star: all end devices talk to the coordinator — simple, but limited range. **Tree:** routers form branches. **Mesh:** routers have multiple paths — long range, self-healing. **Applications:** home automation, smart lighting, smart meters, industrial sensor networks.

(Description 1 M, device types 1.5 M, topologies 1.5 M)

e) Selection criteria: OS, unit cost, time to market, maintainability.

4 M

1. **Operating system:** decide whether the system needs no OS (super-loop), an RTOS (FreeRTOS, VxWorks) for multitasking and deadlines, or an embedded GPOS (embedded Linux) for networking/GUI. It affects memory needs, licence cost and development effort.
2. **Unit cost:** money spent to manufacture each copy of the system (components, PCB, assembly, testing), excluding NRE. For mass products a small saving per unit is very important.
3. **Time to market:** time from the start of design to the product being available for sale. A delay reduces revenue as competitors capture the market window.
4. **Maintainability:** ease of modifying and repairing the system after release — clear documentation, modular firmware, diagnostics and field/OTA firmware updates.

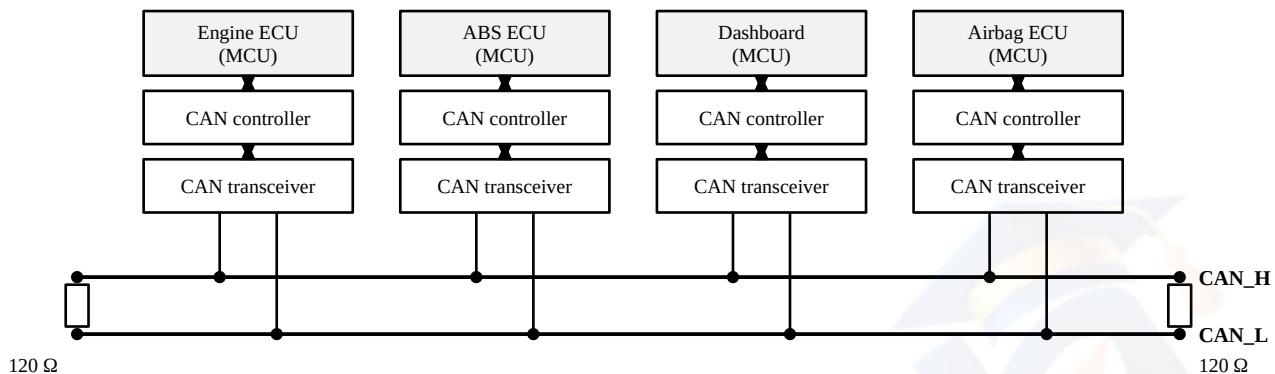
(Each 1 M)

Q.5

a) CAN — bus structure, arbitration, data frame.

6 M

CAN (Controller Area Network), developed by Robert Bosch (1986), is a serial multi-master broadcast bus widely used in vehicles and industry. Speed up to 1 Mbps (≈ 40 m); longer distances at lower speeds.

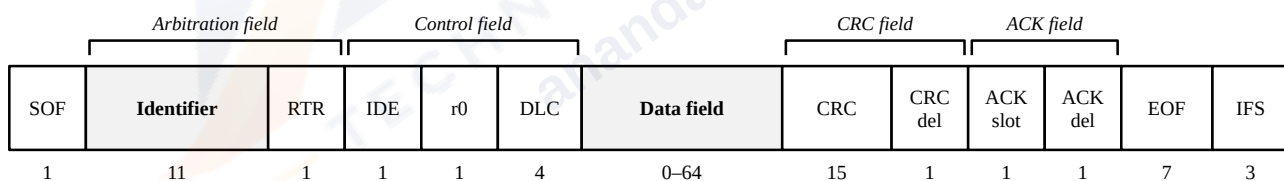


Differential signalling: dominant bit 0 (CAN_H ≈ 3.5 V, CAN_L ≈ 1.5 V), recessive bit 1 (both ≈ 2.5 V)

Fig.: CAN bus — nodes on a twisted pair with 120 Ω terminations

Bus structure: each node = microcontroller + CAN controller + CAN transceiver. All nodes connect to a two-wire twisted pair **CAN_H / CAN_L** (differential — good noise immunity) terminated by 120 Ω at both ends. Messages are broadcast; each node filters messages by identifier.

Bus arbitration (CSMA/CD + AMP): a node transmits only when the bus is idle. If several nodes start together, each sends its identifier bit by bit and monitors the bus. A **dominant 0** overwrites a recessive 1. A node that sends 1 but reads 0 loses and stops; the node with the **lowest identifier (highest priority)** wins without its message being destroyed (non-destructive bit-wise arbitration). Losers retry automatically.



SOF = start of frame, RTR = remote transmission request, IDE = identifier extension, DLC = data length code, EOF = end of frame, IFS = inter-frame space

Fig.: CAN 2.0A (standard) data frame format

- SOF (1 bit) — start of frame; Identifier (11 bits) + RTR — arbitration field; IDE, r0, DLC (4 bits) — control field; Data field 0-8 bytes; CRC (15 bits + delimiter); ACK slot + delimiter; EOF (7 recessive bits); IFS (3 bits).
- Error handling: CRC, bit monitoring, bit stuffing (after 5 identical bits), frame check, ACK check; error frames and automatic retransmission; faulty nodes are switched off (bus-off).

(Bus structure 2 M, arbitration 2 M, data frame 2 M)

b) 4×4 keypad interfacing with Arduino; program.

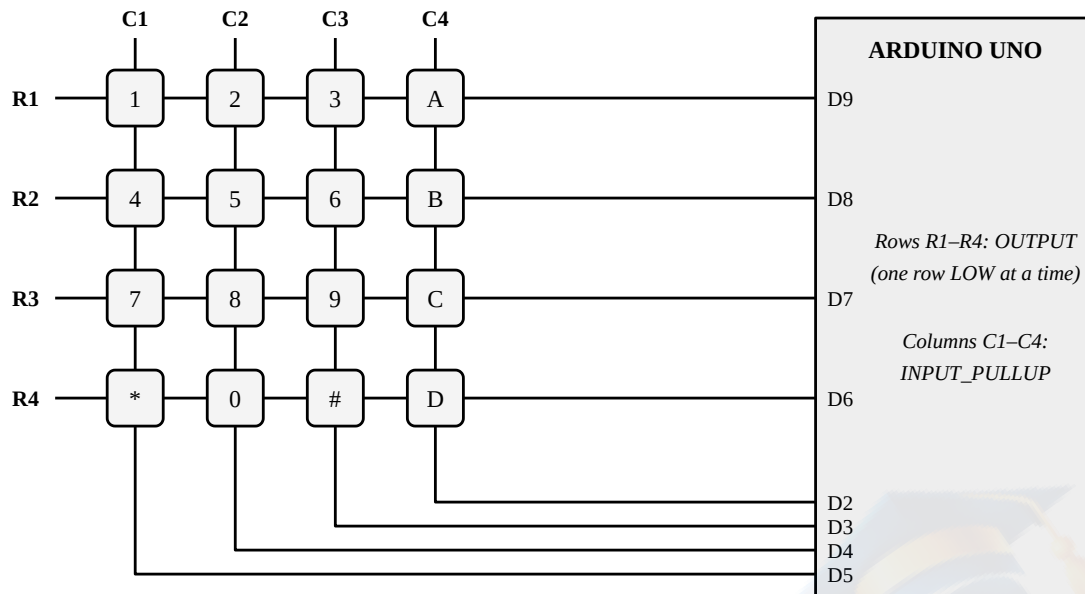


Fig.: 4×4 matrix keypad interfacing with Arduino

- The 16 keys are arranged in 4 rows (R1–R4) and 4 columns (C1–C4); a key connects its row line to its column line when pressed. Only 8 pins are needed.
- Rows (D9–D6) are outputs; columns (D5–D2) are inputs with internal pull-up, so they read HIGH when no key is pressed.
- **Scanning:** make one row LOW and others HIGH; read all columns. If a column reads LOW, the key at that row–column crossing is pressed. Repeat for each row. A 20 ms delay is used for debounce.

Program:

```
// 4x4 matrix keypad scanned without library
// rows R1..R4 = pins 9..6, columns C1..C4 = pins 5..2
const int rowPin[4] = {9, 8, 7, 6};
const int colPin[4] = {5, 4, 3, 2};
const char keyMap[4][4] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

char getKey() {
    for (int r = 0; r < 4; r++) {
        for (int i = 0; i < 4; i++) digitalWrite(rowPin[i], HIGH);
        digitalWrite(rowPin[r], LOW); // make one row LOW
        for (int c = 0; c < 4; c++) {
            if (digitalRead(colPin[c]) == LOW) { // pressed key pulls column LOW
                delay(20); // debounce
                return keyMap[r][c];
            }
        }
    }
    return 0; // no key pressed
}
```

```

}

void setup() {
  for (int i = 0; i < 4; i++) {
    pinMode(rowPin[i], OUTPUT);
    pinMode(colPin[i], INPUT_PULLUP);           // columns idle HIGH
  }
  Serial.begin(9600);
}

void loop() {
  char k = getKey();
  if (k) {
    Serial.print("Key pressed: ");
    Serial.println(k);
  }
  delay(100);
}

```

Output (compiled with g++ using Arduino stub headers; key presses '5', '#', 'A' simulated by connecting the matching row and column — not from real hardware)

```

Key pressed: 5
Key pressed: #
Key pressed: A

```

(Diagram 2 M, program 4 M)

c) IEC 61850 GOOSE and its significance.

6 M

IEC 61850 is the international standard for communication networks and systems in electrical substations. It defines data models of IEDs and three services: MMS (client–server reporting/control), **GOOSE** and Sampled Values.

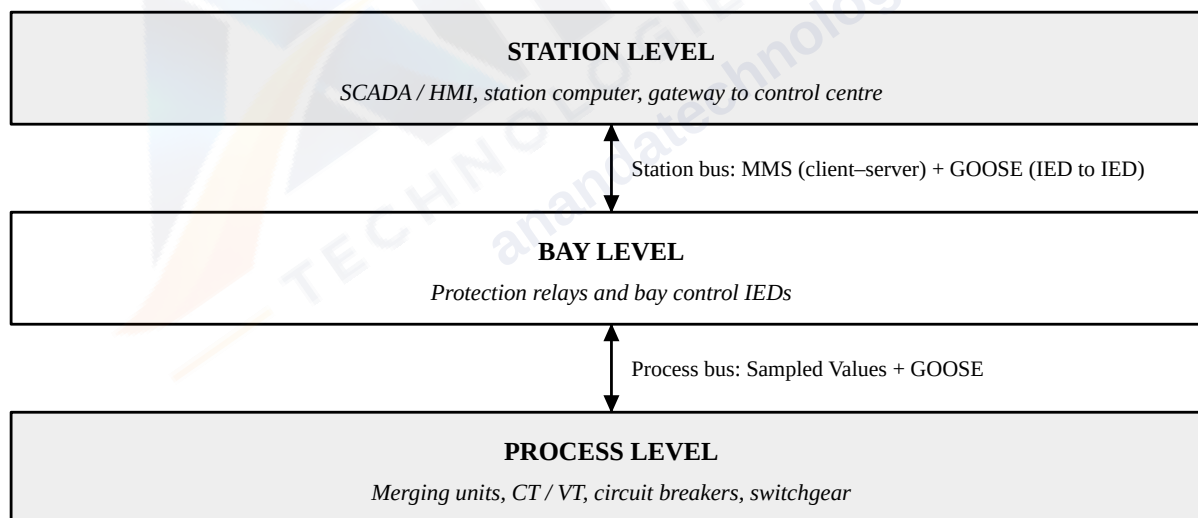
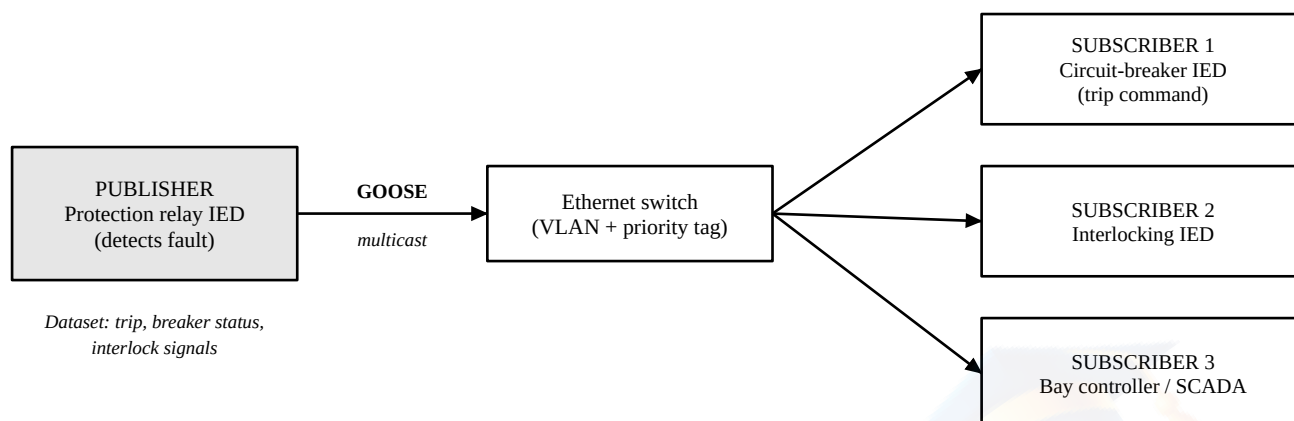


Fig.: IEC 61850 substation communication levels

GOOSE (Generic Object Oriented Substation Event):

- Event-driven, time-critical messages (trip, block, breaker status, interlock) exchanged directly between IEDs.

- Uses a **publisher–subscriber** model with Ethernet **multicast** at layer 2 (no TCP/IP), VLAN priority tagging — so it reaches all subscribers within a few milliseconds.
- No acknowledgement — reliability by **retransmission**: repeated quickly after a change and periodically as heartbeat; stNum/sqNum identify new states and repeats.



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

Significance in electric power systems:

1. Fast protection: relay-to-relay trip, breaker-failure and reverse-blocking schemes meet strict timing.
2. Replaces many hard-wired copper connections — lower cost, less wiring and easier commissioning.
3. Interoperability between IEDs of different manufacturers using a standard data model.
4. Self-supervision of communication (heartbeat) and easy modification by reconfiguration instead of rewiring.
5. Supports interlocking, load shedding and automation in digital substations and smart grids.

(Explanation 3 M, significance 3 M)

Q.6

a) Types of embedded systems; real-time (hard and soft) in detail.

6 M

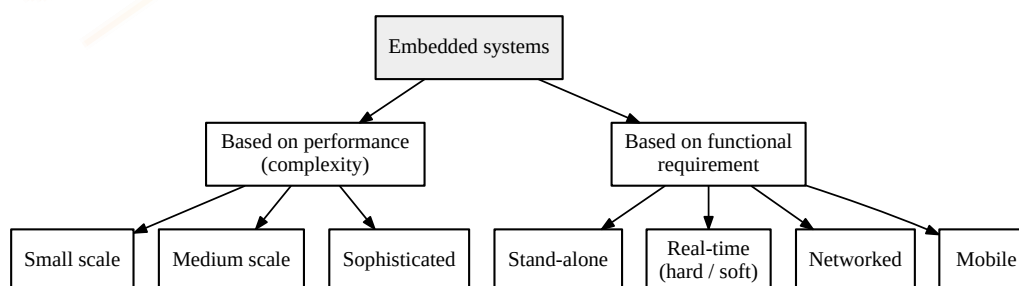


Fig.: Classification of embedded systems

Types: by performance — small scale, medium scale, sophisticated; by function — stand-alone, real-time, networked, mobile.

Real-time embedded systems must produce the correct output **within a specified time (deadline)**; a late result is a wrong result. They are event-driven, use priority-based pre-emptive scheduling and often run an RTOS.

Hard real-time system	Soft real-time system
Deadline must never be missed; a miss means system failure and may cause loss of life or property.	Deadline may be missed occasionally; only performance / quality degrades.
Response time is strict and deterministic (μs – ms).	Response time is less strict; average response is considered.
Validated for worst-case timing.	Validated statistically.
Examples: airbag deployment, ABS braking, cardiac pacemaker, flight control, missile guidance.	Examples: video/audio streaming, online gaming, ATM, washing machine display, mobile phone.

Example — airbag (hard): crash sensor signal must be processed and the airbag fired within about 30 ms; a later firing is useless. **Example — video streaming (soft):** if a frame is late it is skipped, the picture only glitches.

(Types 2 M, hard RT 2 M, soft RT 2 M)

b) Stepper motor — one revolution CW and CCW.

6 M

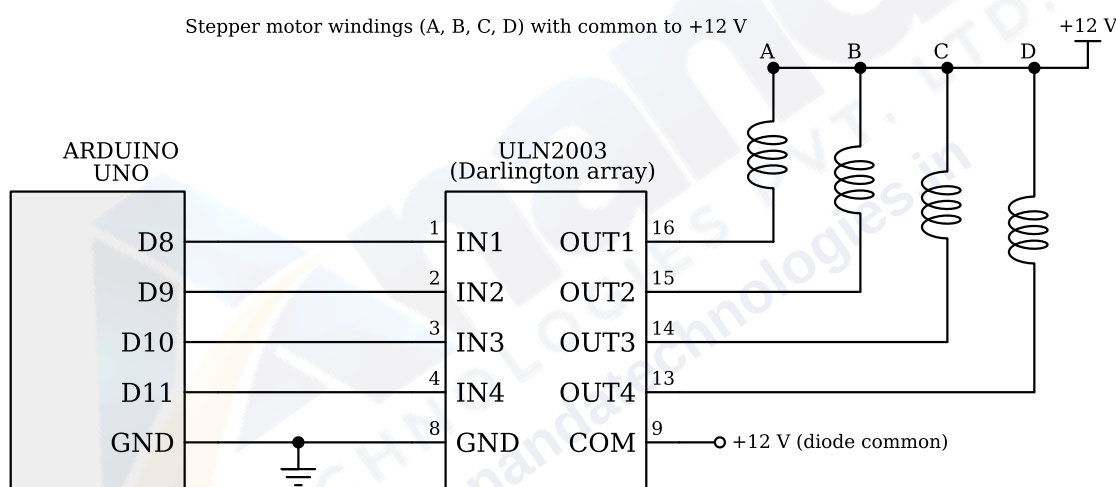


Fig.: Unipolar stepper motor interfacing with Arduino through ULN2003 driver

Motor: 1.8° per step $\rightarrow 360^\circ / 1.8^\circ = 200$ steps per revolution. Wave-drive sequence (one winding ON at a time): A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D for clockwise and D $\rightarrow$ C $\rightarrow$ B $\rightarrow$ A for anticlockwise. Step delay 5 ms $\rightarrow 200 \times 5 \text{ ms} = 1 \text{ s}$ per revolution.

Step	A (D8)	B (D9)	C (D10)	D (D11)
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Program:

```
// Stepper motor (1.8 deg/step, 200 steps/rev) via ULN2003 on pins 8..11
// One revolution clockwise, pause, one revolution anticlockwise
const int coil[4] = {8, 9, 10, 11};
// wave drive: one coil ON at a time
const byte seq[4] = {0b1000, 0b0100, 0b0010, 0b0001};
const int stepsPerRev = 200;
int idx = 0; long stepCount = 0;

void oneStep(int dir) {
    idx = (idx + dir + 4) % 4;
    for (int i = 0; i < 4; i++)
        digitalWrite(coil[i], (seq[idx] >> (3 - i)) & 1);
    stepCount++;
    delay(5); // 5 ms per step -> 1 s per revolution
}

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

void loop() {
    unsigned long t0 = millis();
    for (int s = 0; s < stepsPerRev; s++) oneStep(+1);
    Serial.print("Clockwise: "); Serial.print(stepsPerRev);
    Serial.print(" steps = 360 deg in ");
    Serial.print(millis() - t0); Serial.println(" ms");
    delay(1000);
    t0 = millis();
    for (int s = 0; s < stepsPerRev; s++) oneStep(-1);
    Serial.print("Anticlockwise: "); Serial.print(stepsPerRev);
    Serial.print(" steps = 360 deg in ");
    Serial.print(millis() - t0); Serial.println(" ms");
    delay(1000);
}
```

Output (compiled with g++ using Arduino stub headers; time from simulated millis() — not measured on a motor)

```
Clockwise: 200 steps = 360 deg in 1000 ms
Anticlockwise: 200 steps = 360 deg in 1000 ms
```

(Diagram 2 M, program 4 M)

c) ATmega328 register-level LM35 + fan control.

6 M

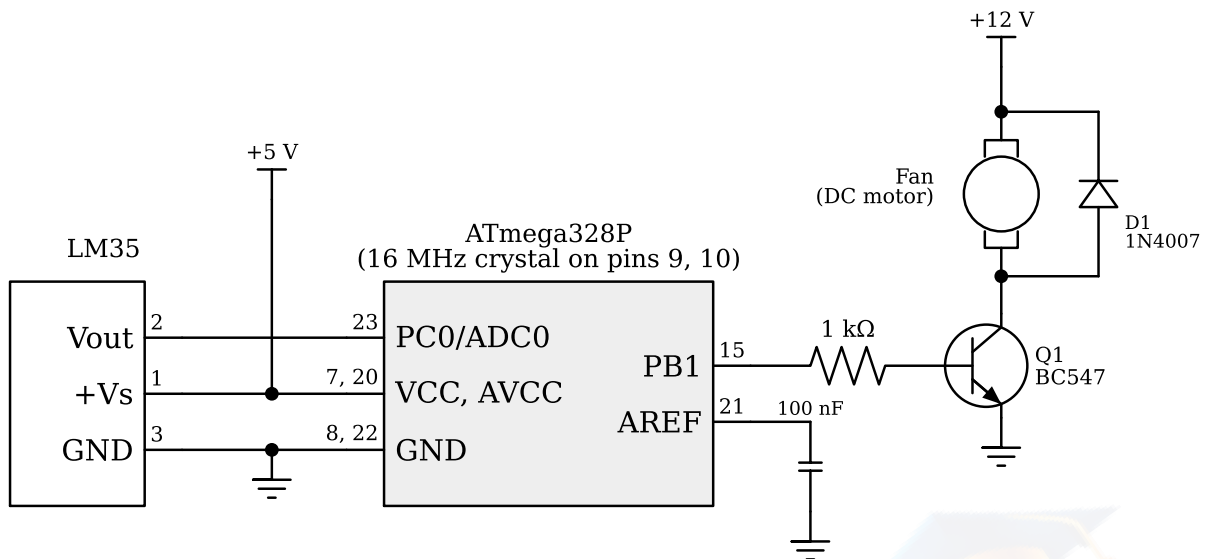


Fig.: ATmega328P — LM35 on ADC0 (PC0) and fan driver on PB1

ADC set-up: ADMUX: REFS0 = 1 → AVCC (5 V) reference, MUX3:0 = 0000 → channel ADC0 (PC0). ADCSRA: ADEN = 1 (enable), ADPS2:0 = 111 → prescaler 128 → 16 MHz / 128 = 125 kHz ADC clock (within 50–200 kHz). Conversion is started by setting ADSC and is over when ADSC returns to 0.

Formula: $V_{in} = \text{ADC} \times 5 \text{ V} / 1024$; LM35 = 10 mV/°C $\Rightarrow T = \text{ADC} \times 500 / 1024$ °C. Fan ON when $T > 35$ °C ($\text{ADC} \geq 72$).

Program (AVR-GCC):

```
/* ATmega328P @ 16 MHz (register level, AVR-GCC): read LM35 on ADC0 (PC0),
   switch ON fan (relay/transistor) on PB1 when temperature > 35 C */
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

void adc_init(void)
{
    ADMUX = (1 << REFS0); /* AVcc (5 V) reference, channel ADC0 */
    /* enable ADC, prescaler 128: 16 MHz / 128 = 125 kHz ADC clock */
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
}

unsigned int adc_read(unsigned char ch)
{
    ADMUX = (ADMUX & 0xF0) | (ch & 0x0F); /* select channel */
    ADCSRA |= (1 << ADSC); /* start conversion */
    while (ADCSRA & (1 << ADSC)) /* wait till complete */
        ;
    return ADC; /* 10-bit result */
}

int main(void)
{
    unsigned int raw;
    float temp;
    DDRB |= (1 << PB1); /* PB1 output -> fan driver */
}
```

```

adc_init();
while (1) {
    raw = adc_read(0);
    temp = raw * 500.0 / 1024.0; /* 5 V / 1024 steps, 10 mV per C */
    if (temp > 35.0) {
        PORTB |= (1 << PB1); /* fan ON */
    } else {
        PORTB &= ~(1 << PB1); /* fan OFF */
    }
    _delay_ms(500);
}
return 0;
}

```

ADC result	Vin (mV)	Temperature (°C)	Fan
61	297.9	29.8	OFF (PB1 = 0)
70	341.8	34.2	OFF (PB1 = 0)
75	366.2	36.6	ON (PB1 = 1)
80	390.6	39.1	ON (PB1 = 1)
69	336.9	33.7	OFF (PB1 = 0)

Simulated run — ADC results 61, 70, 75, 80, 69 fed to the program (compiled with g++ using an ATmega328P register stub; not measured on hardware)

```

t = 0.000 ms : ADC conversion, result = 61
t = 500.104 ms : ADC conversion, result = 70
t = 1000.208 ms : ADC conversion, result = 75
t = 1000.312 ms : PORTB = 0x02 (00000010)
t = 1500.312 ms : ADC conversion, result = 80
t = 2000.416 ms : ADC conversion, result = 69
t = 2000.520 ms : PORTB = 0x00 (00000000)
(simulation stopped at 2500.5 ms)

```

(Circuit 2 M, program 4 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