

Embedded System (315338)

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

Q.1

a) Define embedded system. Give two examples.

2 M

Embedded system: An embedded system is a combination of computer hardware (microcontroller / microprocessor) and software (firmware), designed to perform one specific dedicated function, usually as a part of a larger system, often with real-time constraints.

Examples: Washing machine controller, microwave oven, digital camera, ATM, car airbag / ABS controller, smart watch.

(Definition 1 M, any two examples 1 M)

b) Four features of ATmega328 (Arduino-specific AVR).

2 M

1. 8-bit AVR RISC CPU, 131 instructions, most executed in one clock cycle; 32×8 general purpose registers; up to 20 MHz (16 MHz on Arduino Uno).
2. Memory: 32 KB Flash (0.5 KB used by Arduino boot-loader), 2 KB SRAM, 1 KB EEPROM.
3. 23 programmable I/O lines; 6-channel 10-bit ADC; 6 PWM channels.
4. Two 8-bit and one 16-bit timer/counter; USART, SPI and TWI (I²C) serial interfaces; watchdog timer; six sleep modes; 1.8–5.5 V operation.

(Any four features, ½ M each)

c) Serial vs parallel communication.

2 M

Serial communication	Parallel communication
Data is sent one bit at a time over a single line.	Several bits (e.g. 8) are sent at the same time over separate lines.
Needs fewer wires — low cost, suitable for long distance.	Needs many wires — costly, suitable only for short distance.
Slower for the same clock rate.	Faster for the same clock rate.
Examples: UART, I ² C, SPI, USB, CAN.	Examples: printer port, data bus of a processor, LCD 8-bit interface.

(Any two points, 1 M each)

d) Define RTOS; name two.

2 M

RTOS: A Real Time Operating System is an operating system that manages tasks and resources of an embedded system so that each task gets its response within a fixed, predictable time (deadline). Correctness depends on the result and on the time at which the result is produced.

Examples: FreeRTOS, VxWorks, QNX, µC/OS-II, RTLinux, Zephyr.

(Definition 1 M, two names 1 M)

e) Purpose of watchdog timer.

2 M

- A watchdog timer (WDT) is a hardware timer that **resets the microcontroller** if the program hangs or gets stuck in an endless loop because of software bugs or electrical noise.
- The program must clear (refresh) the watchdog periodically (WDR instruction in AVR). If it is not refreshed before the time-out, the WDT generates a reset, so the system recovers automatically without human help.

(2 M)

f) pinMode() and digitalWrite().

2 M

pinMode(pin, mode): Configures a digital pin as INPUT, OUTPUT or INPUT_PULLUP. It is normally written in `setup()`. Example: `pinMode(7, INPUT);`

digitalRead(pin): Reads the logic level of a digital input pin and returns HIGH (1) or LOW (0). Example: `int s = digitalRead(7);` reads a push-button connected to pin 7.

(Each function 1 M)

g) Four features of Arduino Uno.

2 M

1. Microcontroller ATmega328P at 16 MHz; operating voltage 5 V; input voltage 7–12 V (recommended) through DC jack or 5 V through USB.
2. 14 digital I/O pins (6 of them give PWM: 3, 5, 6, 9, 10, 11) and 6 analog inputs A0–A5 (10-bit ADC).
3. 32 KB Flash (0.5 KB boot-loader), 2 KB SRAM, 1 KB EEPROM.
4. USB-B connector with ATmega16U2 USB-to-serial chip, reset button, ICSP header, on-board LED on pin 13; programmed with Arduino IDE.

(Any four, ½ M each)

Q.2

a) PIC vs AVR vs ARM.

4 M

Parameter	PIC	AVR	ARM
Manufacturer	Microchip	Atmel (now Microchip)	ARM Ltd. designs the core; licensed to NXP, ST, TI etc.
Bus width	8 / 16 / 32-bit	8-bit (also 32-bit AVR32)	32-bit (also 64-bit)
Architecture	Harvard, RISC	Modified Harvard, RISC	Modified Harvard / Von Neumann, RISC (load-store)
Instruction set	About 35 instructions (mid-range)	About 130 instructions	ARM + Thumb / Thumb-2 instruction sets
Speed	4 clocks per instruction cycle (8-bit)	1 clock per instruction (most)	1 instruction per clock with pipeline; up to hundreds of MHz
Memory	Flash, SRAM, EEPROM	Flash, SRAM, EEPROM	Flash, SRAM (large); EEPROM usually emulated

Typical use	Small industrial and consumer control	Arduino boards, hobby and small products	Smartphones, automotive, IoT, high-end embedded
-------------	---------------------------------------	--	---

(Any four points, 1 M each)

b) Selection criteria: NRE cost, time to market, size, correctness and safety.

4 M

- 1. NRE cost (Non-Recurring Engineering cost):** One-time cost of designing the system — research, design, software development, prototype and testing. Once the design is complete it is not paid again for each extra unit. A low NRE cost is preferred when the number of units is small.
- 2. Time to market:** Time taken from starting the design to selling the product in the market. If the product is late, competitors capture the market and revenue is lost. Ready-made boards (Arduino, modules) and RTOS reduce time to market.
- 3. Size:** Physical space (and weight) and memory size of the system. Portable products such as mobile phones and wearables need small size; it decides the choice of package, PCB and processor.
- 4. Correctness and safety:** The system must give correct outputs for all inputs and must not harm people or property even when a fault occurs (e.g. pacemaker, airbag, ABS). Requires thorough testing and fail-safe design.

(Each criterion 1 M)

c) Hard real-time vs soft real-time.

4 M

Hard real-time system	Soft real-time system
Missing a deadline is a total system failure and may be catastrophic.	Occasional missing of deadline is tolerable; only quality degrades.
Response time is strict and deterministic (micro- to milliseconds).	Response time is less strict; average response time matters.
Must be validated for worst-case conditions (formal verification / exhaustive testing).	Validated statistically / by testing typical conditions.
Performance is measured by meeting every deadline (predictability).	Performance is measured by average throughput.
Small size; usually little or no file system and small data files.	Larger size; may use large files and databases.
Examples: airbag controller, ABS, pacemaker, missile guidance.	Examples: video streaming, online gaming, ATM, mobile phone.

(Any four points, 1 M each)

d) DC motor interfacing with Arduino Uno using L293D.

4 M

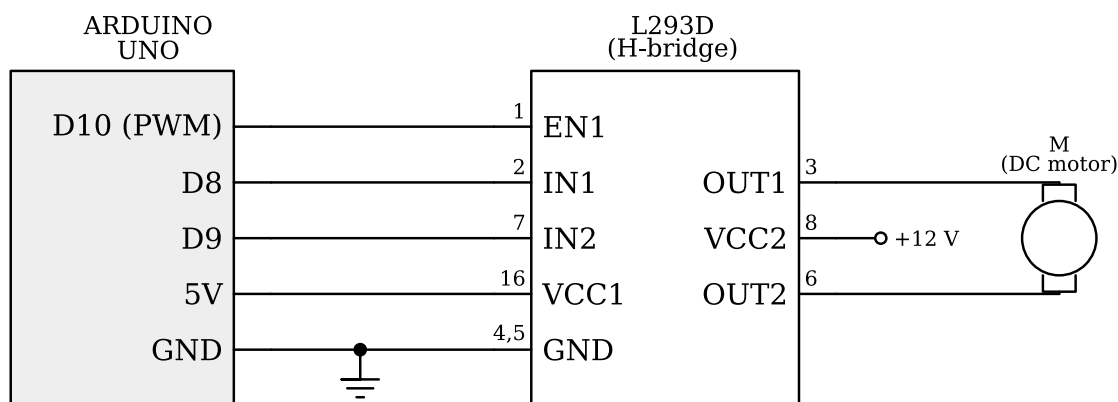


Fig.: DC motor interfacing with Arduino Uno using L293D motor driver

- Arduino pins cannot supply the current of a motor (max. about 20–40 mA per pin), so an **L293D H-bridge driver** is used between the Arduino and the motor.
- **IN1 (pin 2)** and **IN2 (pin 7)** of L293D are driven by Arduino D8 and D9; they decide the direction. **EN1 (pin 1)** is driven by PWM pin D10 to control speed by `analogWrite()`.
- The motor is connected between **OUT1 (pin 3)** and **OUT2 (pin 6)**. VCC1 (pin 16) = 5 V logic supply, VCC2 (pin 8) = motor supply (e.g. 12 V); grounds are common. L293D has internal clamp diodes for the inductive load.

IN1	IN2	Motor
1	0	Clockwise
0	1	Anticlockwise
0	0	Stop
1	1	Brake (stop)

Program (optional, for understanding):

```
// DC motor control with L293D: IN1 = pin 8, IN2 = pin 9, EN1 (PWM) = pin 10
const int in1 = 8, in2 = 9, en1 = 10;

void setup() {
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(en1, OUTPUT);
}

void loop() {
  analogWrite(en1, 200); // speed (0 - 255)
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW); // clockwise
  delay(3000);
  digitalWrite(in1, LOW); digitalWrite(in2, LOW); // stop
  delay(1000);
  digitalWrite(in1, LOW); digitalWrite(in2, HIGH); // anticlockwise
  delay(3000);
  digitalWrite(in1, LOW); digitalWrite(in2, LOW); // stop
  delay(1000);
}
```

(Diagram 2 M, explanation 2 M)

Q.3

a) Block diagram of a basic embedded system.

4 M

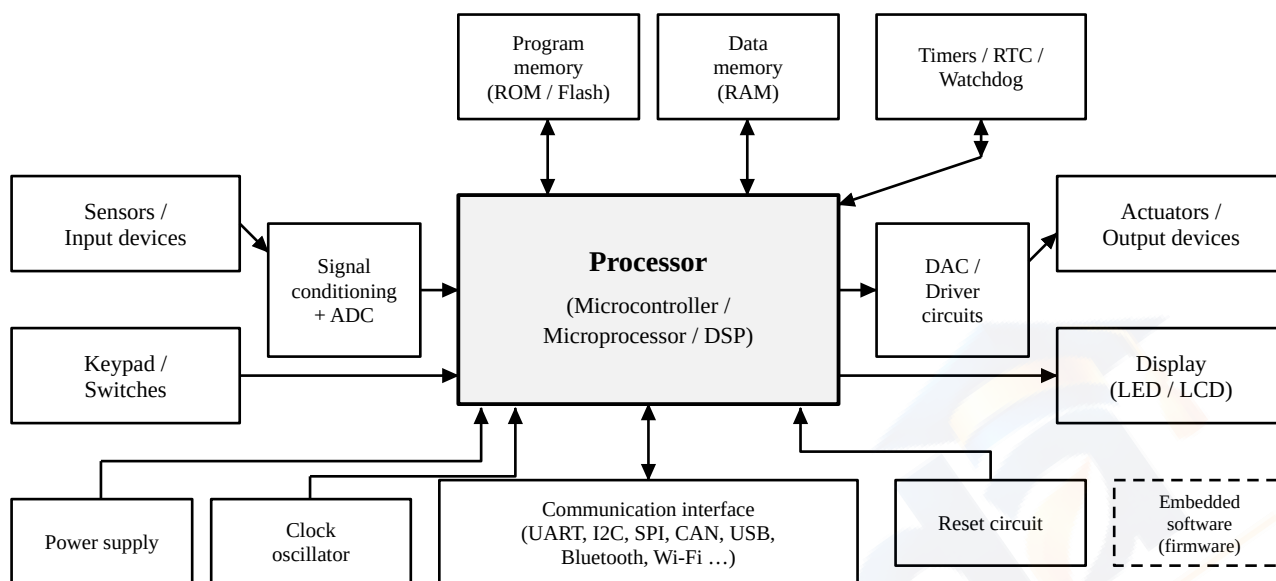


Fig.: Block diagram of a basic embedded system

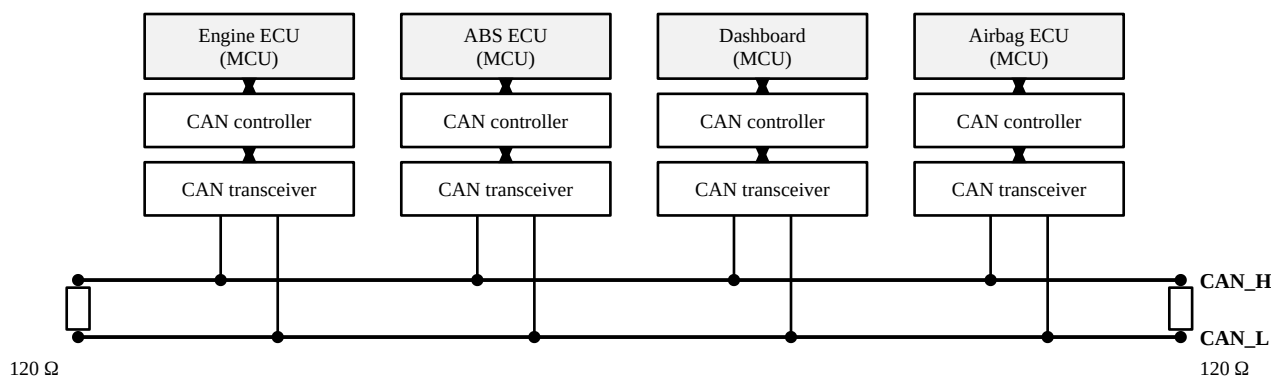
- **Processor:** Microcontroller / microprocessor / DSP — the brain that executes the program and controls all blocks.
- **Memory:** ROM / Flash stores the program (firmware); RAM stores temporary data and stack.
- **Input devices and sensors:** Keypad, switches, temperature or light sensors give data. Analog signals are converted to digital by **signal conditioning + ADC**.
- **Output devices / actuators:** LED, LCD, relay, motor, buzzer; driven through **DAC / driver circuits**.
- **Timers / RTC / watchdog:** Generate delays, keep real time and reset a hung system.
- **Communication interface:** UART, I²C, SPI, CAN, USB, wireless — to talk with other devices.
- **Power supply, clock oscillator and reset circuit** are needed for working of the processor.

(Diagram 2 M, explanation 2 M)

b) CAN protocol with data frame format.

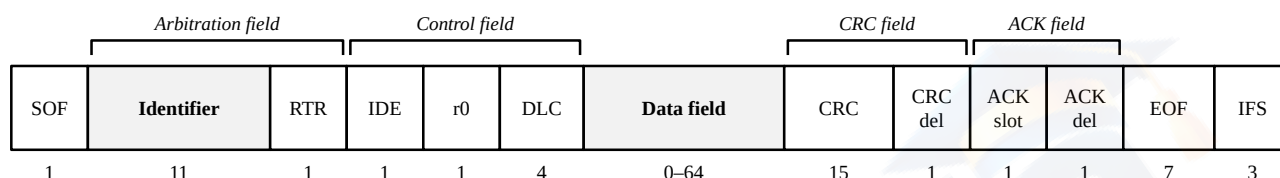
4 M

CAN (Controller Area Network) is a serial, multi-master, message-based bus developed by Bosch for vehicles. All nodes are connected to a two-wire twisted pair (CAN_H, CAN_L) terminated by 120 Ω resistors. Speed is up to 1 Mbps (up to about 40 m).



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

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



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 dominant bit) marks the start of the frame and synchronises all nodes.
- **Arbitration field**: 11-bit identifier (29-bit in CAN 2.0B) + RTR bit. The identifier gives message priority — lower value = higher priority; arbitration is bit-wise and non-destructive.
- **Control field**: IDE, reserved bit r0 and 4-bit DLC (number of data bytes, 0-8).
- **Data field**: 0 to 8 bytes. **CRC field**: 15-bit CRC + delimiter for error detection.
- **ACK field**: receivers write a dominant bit in the ACK slot if the frame was received correctly.
- **EOF**: 7 recessive bits; **IFS**: 3 bits gap before next frame.

(Explanation 2 M, frame format 2 M)

c) Deadlock in RTOS; strategies.

4 M

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

Example: Task T1 holds the UART (R1) and requests the ADC (R2). At the same time task T2 holds the ADC (R2) and requests the UART (R1). Both wait for ever — circular wait.

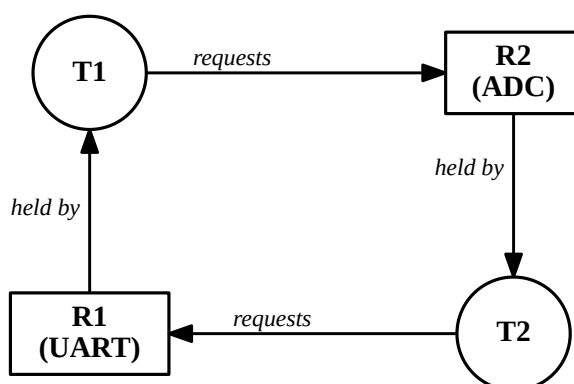


Fig.: Resource allocation graph showing deadlock (circular wait)

Necessary conditions (all four must hold): mutual exclusion, hold and wait, no pre-emption, circular wait.

Strategies to handle deadlock:

1. **Prevention:** Break one of the four conditions — e.g. request all resources at once (no hold and wait), or take resources always in a fixed order (no circular wait).
2. **Avoidance:** Grant a resource only if the system stays in a safe state (Banker's algorithm).
3. **Detection and recovery:** Detect the cycle and recover by aborting a task or pre-empting its resource; use time-outs on semaphore waits.
4. **Ignore (ostrich method):** used where deadlock is very rare.

(Explanation with example 2 M, any two strategies 2 M)

d) Four Arduino math functions.

4 M

Function	Syntax	Use	Example → result
abs()	abs(x)	Absolute (positive) value	abs(-25) → 25
min() / max()	min(x, y), max(x, y)	Smaller / larger of two numbers	min(-25, 40) → -25
constrain()	constrain(x, a, b)	Limits x to the range a to b	constrain(300, 0, 255) → 255
map()	map(v, fromLow, fromHigh, toLow, toHigh)	Re-scales a value from one range to another (integer)	map(512, 0, 1023, 0, 255) → 127
pow()	pow(base, exponent)	Power (returns double)	pow(2, 5) → 32.00
sqrt() / sq()	sqrt(x), sq(x)	Square root / square	sqrt(144) → 12.00, sq(7) → 49

Demonstration program:

```
// Demonstration of Arduino math functions
void setup() {
  Serial.begin(9600);
  int a = -25, b = 40;
  Serial.print("abs(-25)          = "); Serial.println(abs(a));
  Serial.print("min(-25, 40)       = "); Serial.println(min(a, b));
  Serial.print("max(-25, 40)          = "); Serial.println(max(a, b));
  Serial.print("constrain(300,0,255) = ");
  Serial.println(constrain(300, 0, 255));
  Serial.print("map(512,0,1023,0,255) = ");
  Serial.println(map(512, 0, 1023, 0, 255));
  Serial.print("pow(2, 5)                  = "); Serial.println(pow(2, 5));
  Serial.print("sqrt(144)                = "); Serial.println(sqrt(144));
  Serial.print("sq(7)                   = "); Serial.println(sq(7));
}

void loop() {
}
```

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

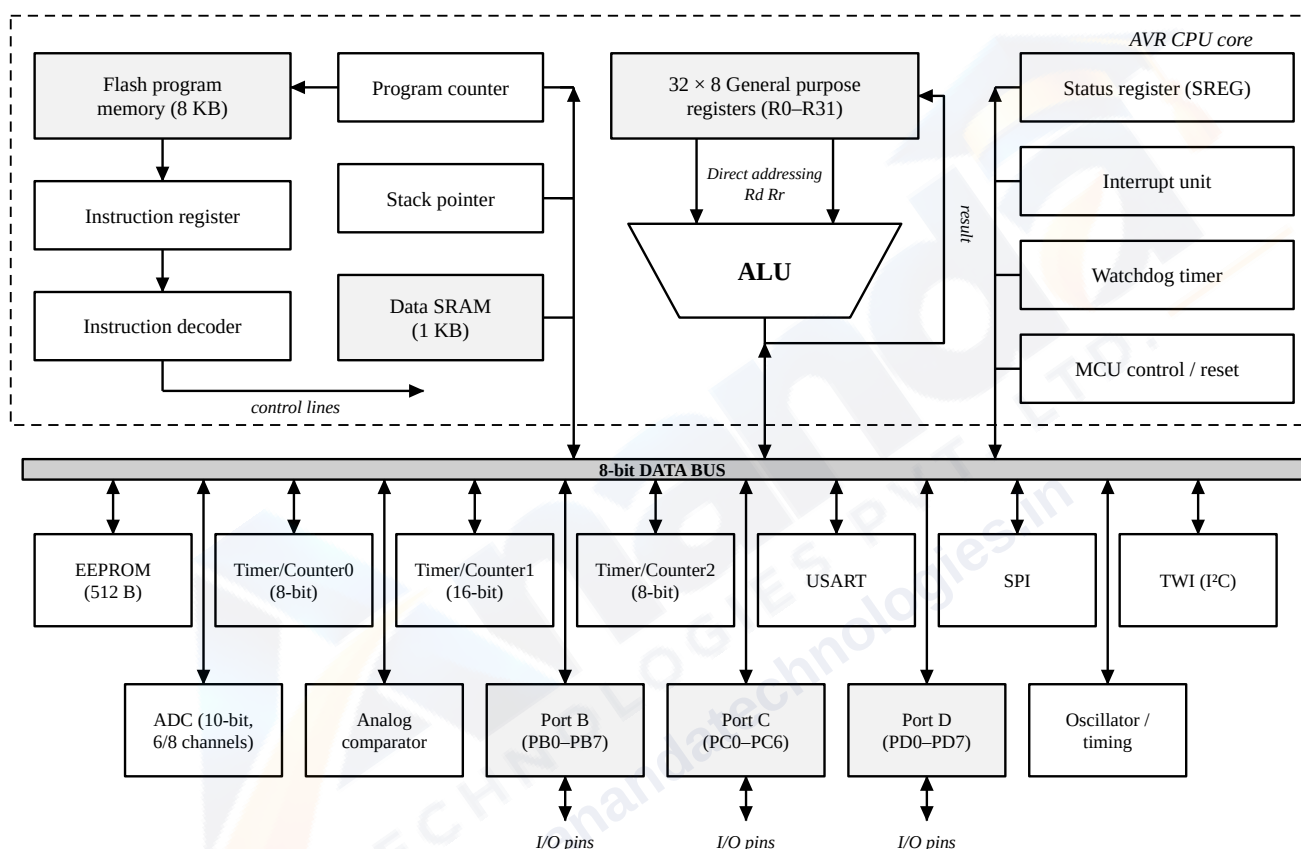
```
abs(-25)          = 25
min(-25, 40)      = -25
max(-25, 40)      = 40
constrain(300,0,255) = 255
map(512,0,1023,0,255) = 127
pow(2, 5)         = 32.00
sqrt(144)         = 12.00
sq(7)             = 49
```

(Any four functions, 1 M each)

Q.4

a) Internal architecture of ATmega8.

4 M



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

Fig.: Internal architecture (block diagram) of ATmega8

- **AVR CPU:** 8-bit RISC core with 32×8 general purpose registers connected directly to the ALU, so most instructions execute in one clock cycle (up to 16 MIPS at 16 MHz).
- **Memory:** 8 KB in-system programmable Flash for program, 1 KB SRAM for data and stack, 512 bytes EEPROM for non-volatile data. Separate program and data buses (Harvard).
- **I/O ports:** Port B (8 bits), Port C (7 bits), Port D (8 bits) — 23 I/O lines, each with DDRx, PORTx and PINx registers.
- **Timers:** Timer/Counter0 (8-bit), Timer/Counter1 (16-bit with compare, capture and PWM), Timer/Counter2 (8-bit, PWM, asynchronous).

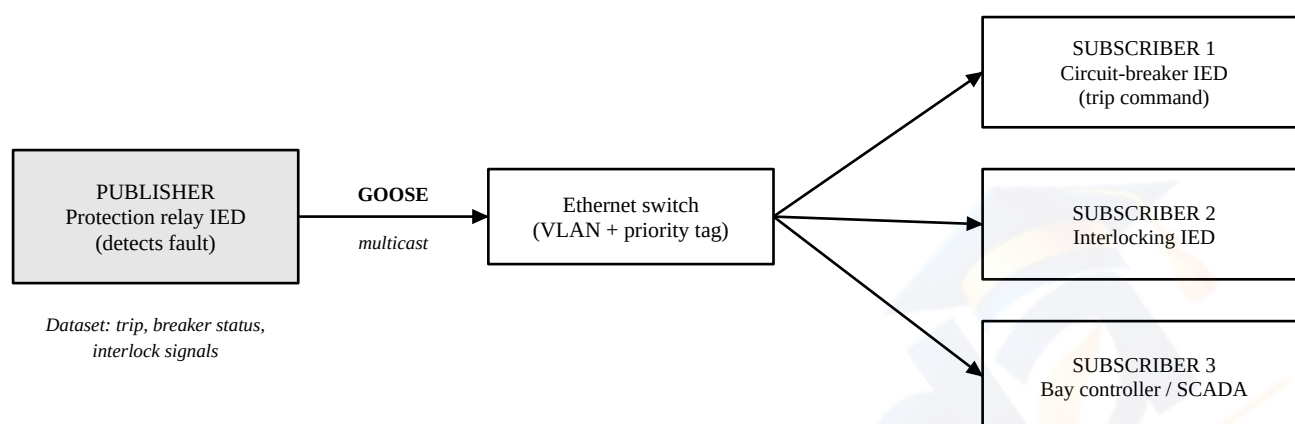
- **Serial interfaces:** USART, SPI and TWI (I²C). **ADC:** 6 channels in PDIP (8 in TQFP), 10-bit. Analog comparator, interrupt unit, watchdog timer with own oscillator.

(Diagram 2 M, any four blocks 2 M)

b) GOOSE publisher–subscriber model and significance.

4 M

GOOSE (Generic Object Oriented Substation Event) is a fast messaging service of IEC 61850 used to send events (trip, interlock, breaker status) between IEDs (Intelligent Electronic Devices) in a substation over Ethernet.



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

- **Publisher:** An IED (e.g. protection relay) publishes a GOOSE message containing a dataset whenever its status changes. The message is sent as an Ethernet **multicast** directly at layer 2 (no TCP/IP), with VLAN priority tag.
- **Subscribers:** Every IED that needs the information (breaker IED, interlocking IED, bay controller) is configured to subscribe to that multicast address and accepts the message; others ignore it.
- There is no acknowledgement; reliability is achieved by **repeated transmission** — fast repeats after an event and then a periodic heartbeat.

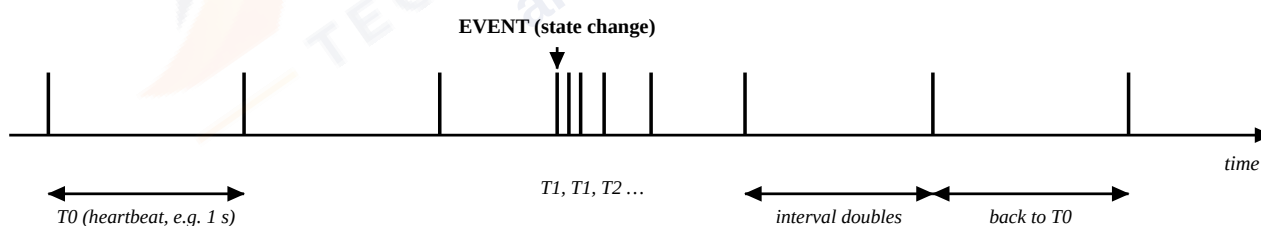


Fig.: GOOSE retransmission — fast repeats after an event, interval doubling back to heartbeat T_0

Significance: very fast transfer (a few ms) for protection trip; replaces many hard-wired copper connections between relays; supports interlocking, breaker-failure protection and load shedding; interoperable devices of different makes; continuous supervision of the link through heartbeat messages.

(Model with diagram 2 M, significance 2 M)

c) Zigbee vs Bluetooth.

4 M

Parameter	Zigbee	Bluetooth (Classic)
Standard	IEEE 802.15.4 (PHY/MAC) + Zigbee Alliance stack	IEEE 802.15.1 (Bluetooth SIG)
Frequency	2.4 GHz (also 868 / 915 MHz)	2.4 GHz, frequency hopping (79 channels, 1600 hops/s)
Data rate	20–250 kbps	1–3 Mbps
Range	10–100 m (more with mesh)	About 10 m (class 2), up to 100 m (class 1)
Power	Very low — battery life of months to years	Higher — battery recharged frequently
Network size / topology	Up to 65,000 nodes; star, tree, mesh	Piconet: 1 master + 7 active slaves; scatternet
Applications	Home / industrial automation, smart meters, sensor networks	Headsets, file transfer, mobile accessories

(Any four points, 1 M each)

d) Round Robin scheduling with Gantt chart.

4 M

Round Robin (time slicing) scheduling: All ready tasks are kept in a circular (FIFO) queue. Each task gets the CPU for a fixed time slice (quantum). If it does not finish in its slice, it is pre-empted and put at the end of the ready queue, and the next task runs. It is fair and no task starves.

Given: T1 = 5 ms, T2 = 3 ms, T3 = 4 ms, all ready at 0, time slice = 2 ms.

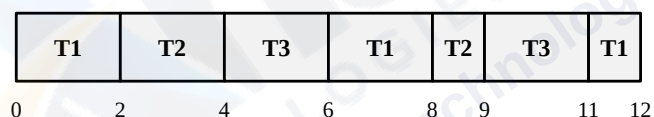


Fig.: Gantt chart (time in ms)

Task	Execution time	Completion time	Turnaround time	Waiting time
T1	5	12	12	7
T2	3	9	9	6
T3	4	11	11	7

Average waiting time = $(7 + 6 + 7) / 3 = 6.67$ ms

(Explanation 2 M, Gantt chart 2 M)

e) Arduino program — LM35 temperature on serial monitor.

4 M

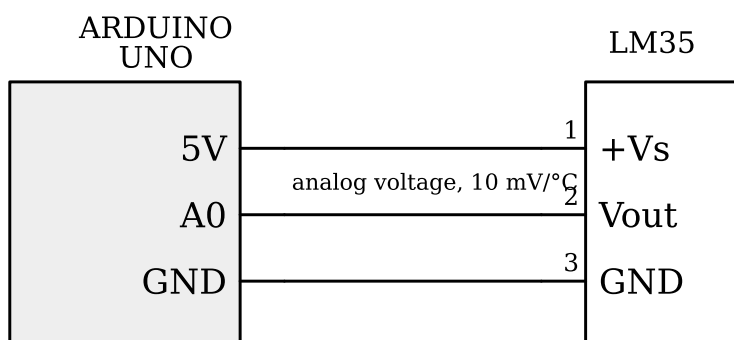


Fig.: Interfacing LM35 temperature sensor with Arduino Uno

Calculation: LM35 gives 10 mV/°C. With 5 V reference and 10-bit ADC, voltage = $\text{ADC} \times 5 / 1023$ V, temperature = voltage $\times 100$ °C.

```
// Read temperature from LM35 (Vout -> A0), show it on Serial Monitor
const int sensorPin = A0; // LM35 output connected to analog pin A0

void setup() {
  Serial.begin(9600); // start serial communication at 9600 baud
}

void loop() {
  int adcValue = analogRead(sensorPin); // 0 to 1023 for 0 to 5 V
  float voltage = adcValue * (5.0 / 1023.0); // convert ADC count to volts
  float tempC = voltage * 100.0; // LM35: 10 mV per degree C
  Serial.print("ADC = ");
  Serial.print(adcValue);
  Serial.print("    Temperature = ");
  Serial.print(tempC);
  Serial.println(" C");
  delay(1000); // read once every second
}
```

Output (compiled with g++ using Arduino stub headers; ADC readings 62, 64, 72 were simulated — not from real hardware)

```
ADC = 62    Temperature = 30.30 C
ADC = 64    Temperature = 31.28 C
ADC = 72    Temperature = 35.19 C
```

(Program 3 M, connection / formula 1 M)

Q.5

a) I²C protocol — frame format, timing, advantages, disadvantages.

6 M

I²C (Inter-Integrated Circuit), developed by Philips (NXP), is a synchronous, half-duplex, multi-master serial bus that uses only two open-drain lines — **SDA** (serial data) and **SCL** (serial clock) — with pull-up resistors. Each slave has a 7-bit (or 10-bit) address. Speeds: 100 kbps (standard), 400 kbps (fast), 3.4 Mbps (high speed).

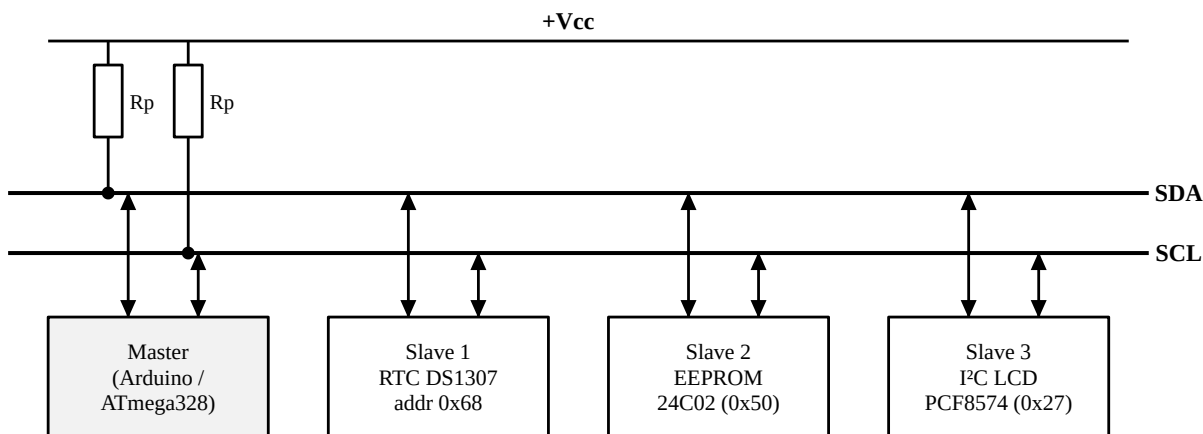


Fig.: I²C bus — one master, many slaves on two open-drain lines

← repeated data byte + ACK for multi-byte transfer →

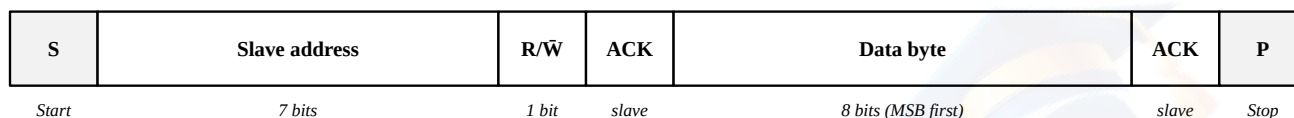


Fig.: I²C frame format (master writing one data byte)

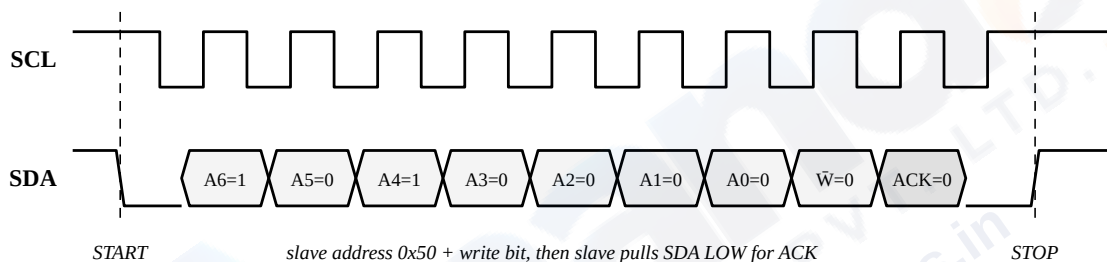


Fig.: I²C timing — START, data bits valid while SCL is HIGH, ACK, STOP

- **START (S):** SDA goes HIGH → LOW while SCL is HIGH.
- **Address + R/ $\bar{W}$:** 7-bit slave address, MSB first, followed by R/ $\bar{W}$ bit (0 = write, 1 = read).
- **ACK:** In the 9th clock the receiver pulls SDA LOW (acknowledge). NACK = SDA left HIGH.
- **Data byte(s):** 8 bits each, every byte followed by ACK. Data on SDA changes only while SCL is LOW.
- **STOP (P):** SDA goes LOW → HIGH while SCL is HIGH.

Advantages: (1) Only two wires for many devices — saves pins and PCB area. (2) Addressing and ACK built in; supports multiple masters with arbitration.

Disadvantages: (1) Slower than SPI; half duplex. (2) Short distance only (bus capacitance limit $\approx$ 400 pF); address conflicts possible between devices with the same fixed address.

(Explanation 2 M, frame + timing 2 M, advantages/disadvantages 2 M)

b) Classification — small, medium, sophisticated embedded systems.

6 M

Embedded systems are classified (a) **by performance / complexity:** small scale, medium scale and sophisticated; and (b) **by functional requirement:** stand-alone, real-time (hard / soft), networked and mobile.

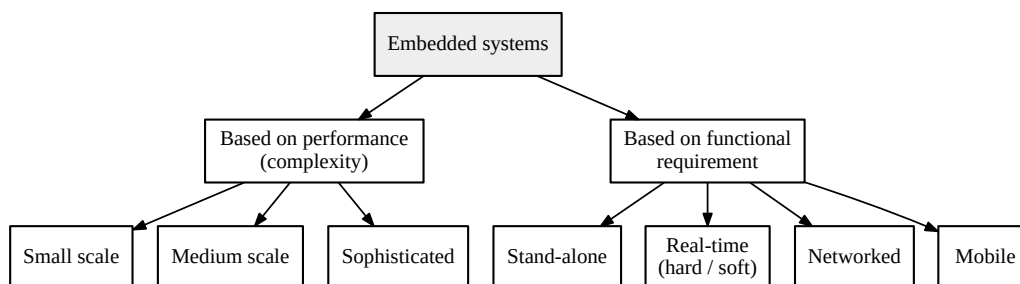


Fig.: Classification of embedded systems

1. **Small scale embedded systems:** Designed with a single 8-bit or 16-bit microcontroller; little hardware and software complexity; may run without an OS; programmed in assembly or C; often battery operated. *Example:* electronic toy, TV remote, digital thermometer, washing-machine controller.
2. **Medium scale embedded systems:** Use a 16-bit or 32-bit microcontroller, DSP or more than one controller; hardware and software are more complex; use RTOS, C/C++, IDE, debuggers and simulators. *Example:* ATM, network router, vending machine, digital storage oscilloscope.
3. **Sophisticated embedded systems:** Very high hardware and software complexity; 32/64-bit multi-core processors, configurable processors, FPGA/ASIC; need hardware–software co-design; cutting-edge applications with strict performance and safety needs. *Example:* smartphone, aircraft flight control, satellite / missile guidance, medical imaging systems.

(Classification 1 M, each type with example 1 M + diagram / points 2 M)

c) Relay interfacing with Arduino — AC lamp control.

6 M

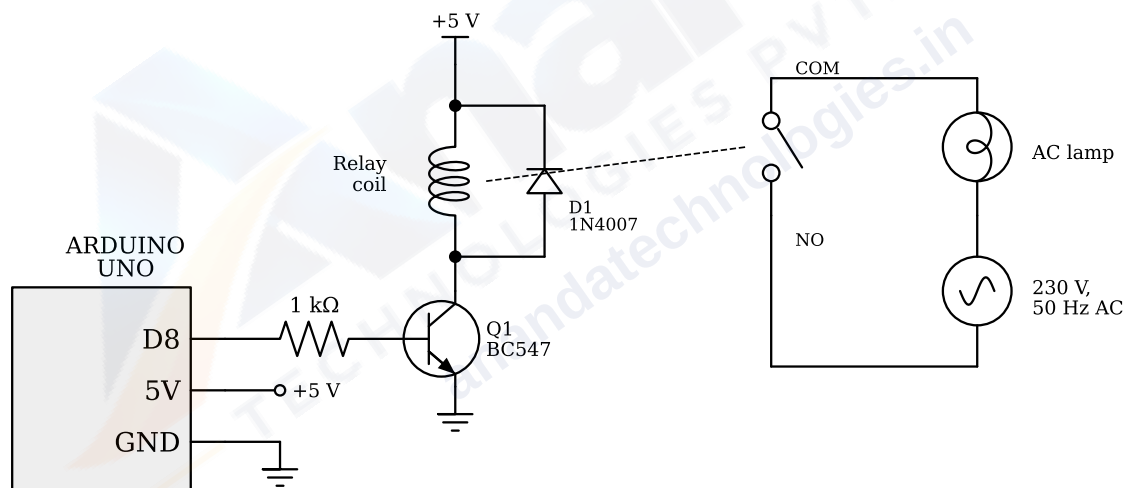


Fig.: Relay interfacing with Arduino to switch a 230 V AC lamp

Hardware connections:

- Arduino digital pin **D8** drives the base of NPN transistor **BC547** through a 1 kΩ resistor (a ready relay module has this driver on board; its IN pin goes to D8, VCC to 5 V, GND to GND).
- The relay coil is connected between +5 V and the collector. When D8 = HIGH the transistor saturates, current flows through the coil and the relay energises.
- Diode **1N4007** across the coil (flyback / free-wheeling diode) absorbs the reverse voltage spike when the coil is switched OFF and protects the transistor.

- The **AC lamp** and 230 V AC mains are connected in series through the relay contacts **COM** and **NO** (normally open). Contacts close when the coil is energised → lamp ON. The relay gives electrical isolation between the 5 V circuit and AC mains.

Program:

```
// Control an AC lamp through a relay module connected to digital pin 8
// Serial command: '1' = lamp ON, '0' = lamp OFF
const int relayPin = 8;           // IN pin of relay module (active HIGH)

void setup() {
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, LOW);    // lamp OFF at start
  Serial.begin(9600);
  Serial.println("Send 1 = Lamp ON, 0 = Lamp OFF");
}

void loop() {
  if (Serial.available()) {
    char ch = Serial.read();
    if (ch == '1') {
      digitalWrite(relayPin, HIGH); // coil energised, NO closes
      Serial.println("Lamp ON");
    } else if (ch == '0') {
      digitalWrite(relayPin, LOW);  // relay released, lamp circuit open
      Serial.println("Lamp OFF");
    }
  }
}
```

Output (compiled with g++ using Arduino stub headers; serial input '1' then '0' simulated; pin changes traced — not from real hardware)

```
[t=0 ms] pin 8 -> LOW
Send 1 = Lamp ON, 0 = Lamp OFF
[t=0 ms] pin 8 -> HIGH
Lamp ON
[t=0 ms] pin 8 -> LOW
Lamp OFF
```

Caution: 230 V AC is dangerous — use an insulated relay module and do not touch the AC side while powered.

(Diagram 2 M, connections 1 M, program 3 M)

Q.6

a) ATmega8 I/O port and Timer0 configuration; LED toggle program.

6 M

I/O ports: ATmega8 has Port B (PB0–PB7), Port C (PC0–PC6) and Port D (PD0–PD7). Each port is controlled by three registers:

Register	Function
DDRx (Data Direction Register)	Bit = 1 → pin is output, bit = 0 → pin is input.

PORTx (Data Register)	For output pin: value sent to the pin. For input pin: 1 enables the internal pull-up resistor.
PINx (Input Pins Address)	Read-only; gives the actual logic level present on the pins.

Timer/Counter0 (8-bit): In ATmega8, Timer0 works in normal mode — TCNT0 counts up from the loaded value to 255 and on the next clock overflows to 0, setting flag **TOV0** in TIFR (and interrupt if TOIE0 in TIMSK = 1). It can also work as a **counter** of external pulses on pin T0 (PD4).

CS02	CS01	CS00	Clock source (TCCR0)
0	0	0	Timer stopped
0	0	1	clk (no prescale)
0	1	0	clk/8
0	1	1	clk/64
1	0	0	clk/256
1	0	1	clk/1024
1	1	0	External pin T0, falling edge (counter)
1	1	1	External pin T0, rising edge (counter)

Delay calculation: $F_{CPU} = 1 \text{ MHz}$ (internal RC), prescaler 1024 $\rightarrow$ one count = 1024 μs . Loading TCNT0 = 12 gives $(256 - 12) = 244$ counts $\rightarrow 244 \times 1.024 \text{ ms} = 249.856 \text{ ms}$. Two such delays $\approx 0.5 \text{ s}$.

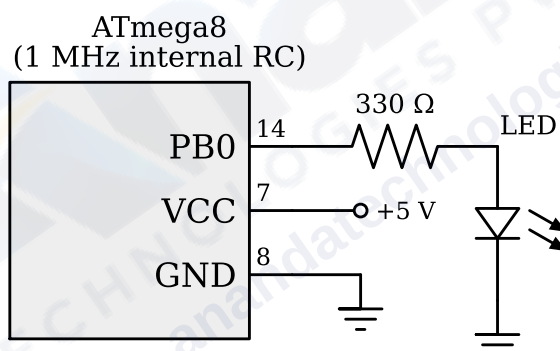


Fig.: LED connected to PB0 of ATmega8 (active HIGH)

Program (AVR-GCC):

```

/* ATmega8 @ 1 MHz (internal RC oscillator)
   Toggle LED on PB0 about every 0.5 s using Timer0 delay */
#define F_CPU 1000000UL
#include <avr/io.h>

/* Timer0, prescaler 1024, preload 12:
   (256 - 12) x 1024 / 1 MHz = 249.856 ms */
void delay_250ms(void)
{
    TCNT0 = 12; /* initial count */
    TCCR0 = (1 << CS02) | (1 << CS00); /* start timer: clk/1024 */
    while (!(TIFR & (1 << TOV0))) /* wait for overflow flag */

```

```

;
TCCR0 = 0;          /* stop timer */
TIFR = (1 << TOV0); /* clear TOV0 by writing logic 1 */
}

int main(void)
{
    DDRB |= (1 << PB0); /* PB0 as output (LED) */
    while (1) {
        PORTB ^= (1 << PB0); /* toggle LED */
        delay_250ms();
        delay_250ms();        /* total ~ 0.5 s */
    }
    return 0;
}

```

Simulated PORTB changes (program compiled with g++ using an ATmega8 register stub that models Timer0 overflow at 1 MHz — not measured on hardware)

```

t = 0.000 ms : PORTB = 0x01 (00000001)
t = 499.712 ms : PORTB = 0x00 (00000000)
t = 999.424 ms : PORTB = 0x01 (00000001)
t = 1499.136 ms : PORTB = 0x00 (00000000)
(simulation stopped at 1749.0 ms)

```

(Ports 1 M, timer configuration 2 M, program 3 M)

b) LoRa — working, advantages, IoT uses.

6 M

LoRa (Long Range) is a low-power, long-range wireless technology developed by Semtech. It works in licence-free sub-GHz ISM bands (865–867 MHz in India, 868 MHz Europe, 915 MHz USA). **LoRaWAN** is the network (MAC) protocol defined by the LoRa Alliance.

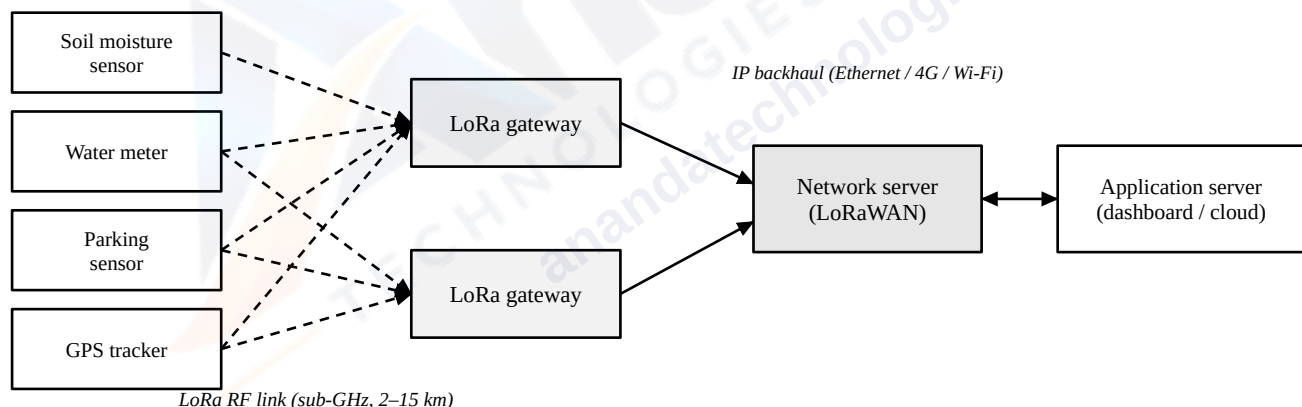


Fig.: LoRaWAN architecture (star-of-stars)

i) Working:

- LoRa uses **Chirp Spread Spectrum (CSS)** modulation — each symbol is a frequency sweep (chirp). A higher spreading factor (SF7–SF12) gives longer range and better noise immunity but lower data rate.
- Battery-powered **end nodes** (sensors) send small packets by radio. Every **gateway** in range receives them and forwards them over IP (Ethernet / 4G / Wi-Fi) to the **network server** — star-of-stars topology.

- The network server removes duplicates, checks security (AES-128) and sends data to the **application server** (dashboard / cloud). Class A, B and C devices decide when downlink is possible.

ii) Advantages:

- Long range: about 2–5 km in cities and up to 15 km in open areas.
- Very low power — battery life of several years.
- Licence-free band, low cost modules and gateways; one gateway serves thousands of nodes.
- Good penetration through buildings; secure (AES-128 end-to-end encryption).

iii) IoT uses: smart agriculture (soil moisture, weather stations), smart metering (water, gas, electricity), smart parking and street lighting, asset and livestock tracking, environment and pollution monitoring, industrial monitoring.

Limitation: low data rate (about 0.3–50 kbps) — not suitable for audio/video.

(Working with diagram 3 M, advantages 1.5 M, IoT uses 1.5 M)

c) Task management in RTOS; inter-task communication.

6 M

Task: An independent piece of program (thread) with its own stack, priority and Task Control Block (TCB). **Task management** is the part of the RTOS kernel that creates, deletes, schedules and switches tasks (multitasking) so that every task meets its deadline.

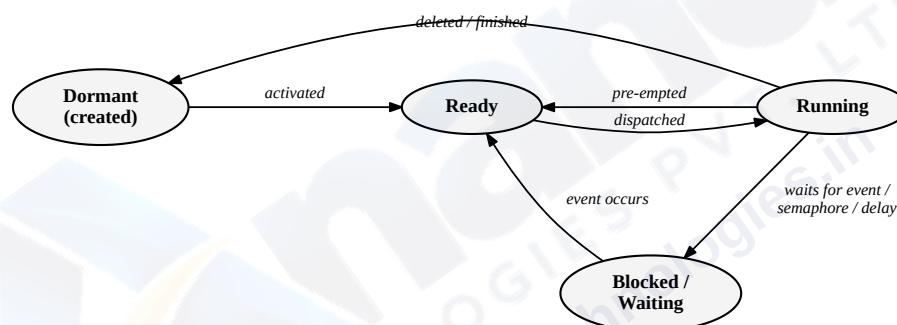


Fig.: Task state diagram in an RTOS

- **Dormant:** task is created but not yet active.
- **Ready:** task is ready to run and waits for the CPU.
- **Running:** task is executing on the CPU (only one at a time on a single-core CPU).
- **Blocked / Waiting:** task waits for an event, delay, semaphore or message; it returns to Ready when the event occurs.
- The **scheduler** selects the highest-priority ready task; on a task switch the kernel saves the context (registers, PC, SP) of the old task in its TCB and restores that of the new task (**context switching**).

Inter-task communication (ITC): Tasks share data and synchronise through kernel objects:

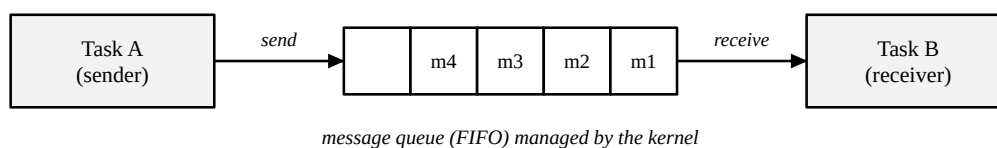


Fig.: Inter-task communication through a message queue

- **Message queue:** FIFO buffer of messages; a sender task posts, a receiver task waits (blocks) until a message arrives.
- **Mailbox:** holds one message (pointer) at a time between tasks.
- **Pipe:** byte-stream channel from one task to another.
- **Semaphore / mutex:** used for synchronisation and for mutual exclusion on shared resources.
- **Event flags / signals:** a task waits for one or more event bits set by other tasks or ISRs. **Shared memory** (global data) protected by a mutex.

(Task management + state diagram 3 M, inter-task communication 3 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