Embedded Systems

Short notes for B.Tech ECE · EEE · CSE

Uncodemy Handwritten Notes · free for students

I. Introduction

- **Embedded system** = a computer system built inside a larger device to do one dedicated job, usually with real-time limits.
- Examples: washing machine controller, car ABS, pacemaker, router, smart watch, printer.
- Parts: processor, memory, input (sensors), output (actuators, display), communication, software (firmware), power supply.

➤ Characteristics

1. Single function (dedicated).
2. Tight limits on cost, size, power and memory.
3. Reactive and real-time (must respond within a deadline).
4. High reliability; often runs for years without a reset.
5. Hardware and software designed together.

➤ Classification

Basis	Types
Performance	Small scale (8/16-bit MCU), medium scale (16/32-bit), sophisticated (32/64-bit, complex)
Function	Stand-alone, real-time, networked, mobile
Timing	Hard real-time (missing a deadline = failure, e.g. airbag), soft real-time (late = poorer quality, e.g. video)

➤ General purpose computer vs embedded system

Point	General purpose computer	Embedded system
Task	Many tasks, user installs programs	One fixed task
OS	Desktop OS	None, small RTOS or embedded Linux
Response	Not time-critical	Often time-critical
Resources	Large	Limited

2. Processors for Embedded Systems

Type	Note	Example
Microcontroller	CPU + memory + peripherals on one chip	8051, AVR, PIC, ARM Cortex-M
Microprocessor	Needs external memory and peripherals	ARM Cortex-A, x86
DSP	Fast multiply-accumulate (MAC), Harvard bus	Audio, image, modem processing
FPGA / ASIC	Custom hardware, parallel	High-speed or high-volume designs
SoC	Whole system (CPU, GPU, radio, memory) on one chip	Phone processors

➤ RISC vs CISC

Point	RISC	CISC
Instructions	Few, simple, fixed length	Many, complex, variable length
Cycles per instruction	About 1 (pipelined)	Many
Memory access	Only load/store instructions	Many instructions access memory
Registers	Many	Fewer
Example	ARM, RISC-V, AVR	x86, 8051

- **Harvard architecture**: separate program and data buses (fetch and data access together). **Von Neumann**: one shared bus.
- **ARM Cortex families**: **A** (applications, runs OS), **R** (real-time, e.g. disk controllers), **M** (microcontroller, low power).

3. Memory

Memory	Volatile?	Note
SRAM	Yes	Fast, used for on-chip RAM and cache
DRAM	Yes	Dense, needs refresh
Mask ROM	No	Programmed at the factory
EPROM	No	Erased by UV light
EEPROM	No	Electrically erasable, byte-wise; stores settings
Flash	No	Block erase, stores program code (firmware)

- **Memory map**: tells which address range belongs to flash, RAM and peripheral registers.
- Most microcontrollers use **memory-mapped I/O** for peripherals.

4. Peripherals and Interfacing

➤ GPIO

- Each pin set as input or output with a direction register. Inputs often need pull-up / pull-down resistors.
- **Switch debouncing**: mechanical contacts bounce for a few ms; fix with an RC filter or a software delay.

➤ Timers, counters, watchdog

- **Timer**: counts clock pulses to make delays or measure time. **Counter**: counts external events.
- **Watchdog timer (WDT)**: resets the system if the software hangs and stops "kicking" (refreshing) it.

➤ PWM (pulse width modulation)

FORMULA Duty cycle D = Ton / T x 100%. Average output = D x Vmax.

EXAMPLE PWM at 1 kHz (T = 1 ms) with Ton = 250 µs: D = 25%. Average of a 5 V signal = 0.25 x 5 = 1.25 V. Used for motor speed and LED brightness.

ADC and DAC

FORMULA ADC resolution (step) = Vref / 2^n. Digital output = Vin / step (rounded down).

Solved example (ADC)

- 10-bit ADC, Vref = 5 V, input 2.5 V.
  - Step = 5 / 2^10 = 5 / 1024 = 4.88 mV
  - Output = 2.5 / 0.004883 = 512 (= 200H)
  - Maximum quantisation error = ± 1/2 LSB = ± 2.44 mV
- ADC types: **flash** (fastest, 2^n − 1 comparators), **successive approximation** (n clock cycles, common in MCUs), **dual slope** (slow, very accurate, noise rejection), **sigma-delta** (high resolution, audio).
- DAC types: **binary weighted resistor**, **R-2R ladder** (only two resistor values, preferred).

Interrupts vs polling

Point	Polling	Interrupt
How	CPU keeps checking the device	Device signals the CPU
CPU time	Wasted while waiting	Free for other work
Response	Depends on loop time	Fast, predictable
Use	Simple, slow devices	Time-critical events

- **ISR** (interrupt service routine): keep it short, save/restore context, clear the interrupt flag.
- **Interrupt latency** = time from interrupt request to the first ISR instruction.
- **DMA**: moves data between memory and peripherals without the CPU.

5. Communication Protocols

Protocol	Wires	Type	Speed (typical)	Note
UART	2 (TX, RX)	Asynchronous, full duplex	Up to about 1 Mbps (9600, 115200 baud common)	Start bit, data, optional parity, stop bit
SPI	4 (MOSI, MISO, SCLK, SS)	Synchronous, full duplex	Tens of Mbps	One master, slave select per device
I2C	2 (SDA, SCL)	Synchronous, half duplex	100 kHz, 400 kHz, faster modes	Addressing (7-bit), open drain with pull-ups, ACK bit
CAN	2 (CANH, CANL)	Differential, multi-master	Up to 1 Mbps (classic)	Cars; priority by message ID (arbitration)
USB	4 (VBUS, D+, D−, GND)	Host-controlled, differential	12 Mbps (full speed) and higher	Plug and play

EXAMPLE UART at 9600 baud, 8N1 frame (1 start + 8 data + 1 stop = 10 bits): 9600 / 10 = 960 bytes per second.

REMEMBER I2C = 2 wires + addresses (many devices, slower). SPI = more wires + chip select (fast). CAN = robust differential bus for vehicles.

6. Embedded C Programming

- `volatile`: tells the compiler the value can change outside the program flow (hardware register, ISR variable). Do not optimise it away.
- `const`: put fixed tables in flash. Use fixed-width types: `uint8_t`, `uint16_t`, `uint32_t`.
- Bit operations on registers:

```
PORTB |=  (1 << 3);    // set bit 3
PORTB &= ~(1 << 3);    // clear bit 3
PORTB ^=  (1 << 3);    // toggle bit 3
if (PINB & (1 << 2))   // test bit 2
```

- ***Super loop*** structure (no OS):

```
int main(void) {
    init_hardware();
    while (1) {          /* runs forever */
        read_sensors();
        process();
        update_outputs();
    }
}
```

- Tools: **cross compiler** (builds on PC for the target), linker, **debugger** (JTAG / SWD), **simulator**, **emulator (ICE)**, logic analyser, oscilloscope.

7. Real-Time Operating Systems (RTOS)

- **RTOS**: OS that gives predictable (deterministic) timing. Small kernel, fast context switch, priority-based preemptive scheduling.
- **Task states**: ready, running, blocked (waiting), suspended.
- Kernel services: task management, scheduling, timers, inter-task communication (queues, mailboxes, event flags), synchronisation (semaphores, mutexes).

➤ Scheduling algorithms

Algorithm	Rule	Note
Round robin	Equal time slices	No priorities
Priority preemptive	Highest priority ready task runs	Most RTOS default
Rate monotonic (RMS)	Shorter period → higher priority (fixed)	Optimal fixed-priority for periodic tasks
Earliest deadline first (EDF)	Nearest deadline runs first (dynamic)	Schedulable if U <= 1

FORMULA CPU utilisation U = sum of Ci / Ti. RMS is surely schedulable if U <= n (2^(1/n) − 1). For large n, the bound → ln 2 ≈ 0.693.

➤ Solved example (rate monotonic test)

- Three tasks (C, T in ms): T1 = (1, 4), T2 = (2, 6), T3 = (1, 10).
  - U = 1/4 + 2/6 + 1/10 = 0.25 + 0.333 + 0.10 = 0.683
  - RMS bound for n = 3: 3 (2^(1/3) − 1) = 3 (1.2599 − 1) = 0.780
  - 0.683 <= 0.780, so the task set is schedulable under RMS (and also under EDF since U <= 1).

- Priorities: T1 (period 4) highest, then T2, then T3.

➤ Synchronisation problems

- **Semaphore**: binary (signalling) or counting (resource pools). **Mutex**: lock with ownership; only the owner can release it.
- **Priority inversion**: a high-priority task waits for a resource held by a low-priority task, while a medium-priority task keeps running.
- Fix: **priority inheritance** (low task temporarily gets the high priority) or **priority ceiling** protocol.
- **Deadlock**: tasks wait for each other forever; avoid with a fixed lock order.

8. Embedded System Design

- **Design steps**: requirements → specification → architecture → hardware and software design → integration → testing.
- **Hardware–software co-design**: decide which function goes in hardware (fast) and which in software (flexible).
- **Design metrics**: unit cost, NRE (one-time engineering) cost, size, power, performance, time to market, maintainability.
- **Low power**: sleep / idle modes, clock gating, lower clock frequency and voltage, turn off unused peripherals.
- **IoT node**: sensor + MCU + radio (Wi-Fi, BLE, Zigbee, LoRa) + cloud link; protocols like MQTT.

9. Most Asked Questions (practice)

1. Define an embedded system and list its characteristics with examples.
2. Compare microprocessor and microcontroller; RISC and CISC; Harvard and von Neumann.
3. Explain the ARM Cortex-A, R and M families.
4. Explain the working of a watchdog timer and PWM.
5. Find ADC step size and digital output for a given input.
6. Compare UART, SPI, I2C and CAN.
7. Compare polling and interrupts. What should an ISR do?
8. What is the use of the volatile keyword? Write bit-manipulation code.

9. Explain RTOS task states and rate monotonic scheduling with an example.

10. What is priority inversion? How does priority inheritance solve it?

> **TIP** Embedded Systems is mostly a university paper (GATE EC touches only microcontroller basics). Learn protocol comparison tables, RTOS scheduling numericals and ADC/PWM calculations well.