Embedded Software — ATmega328P

Master II Robotics Engineering — Exam Cheat Sheet

1 Embedded Systems — Fundamentals

Definition:
An **embedded system** is a *dedicated computing system* designed to perform a **specific function**, integrated into a larger mechanical or electrical system.

Main components:

- Hardware (processor, memory, sensors, actuators)
- Embedded software (firmware, drivers, control logic)
- Interaction with the physical world

Key characteristics:

- Dedicated task (not general purpose)
- Deterministic / real-time behavior
- Limited CPU, memory, power
- High reliability
- Often no OS or RTOS

> **Exam tip:** Embedded systems prioritize **predictable timing**, not raw performance.

2 Applications of Embedded Systems

- Industrial automation
- Automotive systems
- Medical devices
- Consumer electronics
- IoT
- Robotics (motor control, sensors, communication)

Robotics insight:
A robot is a **network of embedded systems** working together.

3 Processor — Heart of Embedded Systems

Role of processor:

- Acquire sensor data
- Execute control algorithms
- Drive actuators
- Manage communication

Moore's Law:

- Transistor count increases exponentially
- Performance and energy efficiency improve

> Embedded processors value **determinism** over speed.

4 Targets of Embedded Systems

4.1 ASIC and FPGA

- ASIC: fixed hardware, high performance, no flexibility
- FPGA: reconfigurable, parallel execution

Used for:

- High-speed processing
- Deterministic timing

4.2 Microprocessors (µP)

- CPU only
- External RAM, ROM, peripherals
- High performance
- High power consumption

Examples:

- Intel 8086
- Pentium
- Core i7

Not ideal for real-time robotics control.

4.3 Microcontrollers (MCU)

Definition:
A microcontroller integrates on a single chip:

- CPU
- RAM
- Flash
- EEPROM
- I/O ports
- Timers
- Communication interfaces

> MCUs are the **core controllers** in embedded robotics systems.

4.4 DSP Microcontrollers

Optimized for:

- Digital signal processing
- Filtering
- Audio/video
- Telecommunications

Key features:

- Fast multiply–accumulate
- Stream data processing

4.5 System on Chip (SoC)

- CPU + GPU + peripherals on one chip
- Example: Raspberry Pi Broadcom 2711
- High performance
- Runs Linux

Drawbacks:

- High power consumption
- Less deterministic timing

5 Processor Architectures

5.1 Von Neumann Architecture

- Single memory for data and program
- Single bus
- Memory bottleneck

Used in general-purpose computers.

5.2 Harvard Architecture

- Separate program and data memories
- Separate buses
- Parallel access

> Used in **microcontrollers** (e.g., ATmega328).

5.3 Harvard in Microcontrollers

Three buses:

- Address bus
- Data bus
- Control bus

Benefits:

- Deterministic execution
- Faster instruction execution

6 ATmega328 Microcontroller

Manufacturer: Atmel (Microchip)

Core features:

- 8-bit AVR RISC
- Harvard architecture
- 131 instructions
- CMOS (low power)
- Supply: 2.7V – 5.5V
- Clock: up to 16 MHz

6.1 Memory Organization

- Flash: program storage
- SRAM: runtime variables
- EEPROM: non-volatile data

> EEPROM retains data after power-off, SRAM does not.

6.2 Peripherals

- 23 digital I/O pins
- 6 analog inputs (10-bit ADC)
- Timers / counters
- UART
- SPI
- I^2C (TWI)

7 Arduino Uno Board

7.1 Power Supply

- USB 5V (500 mA limit)
- External 7–12V
- On-board 5V and 3.3V regulators

7.2 I/O Mapping

- Port D → Digital 0–7
- Port B → Digital 8–13
- Port C → Analog A0–A5

LED on pin 13 (PB5).

8 Programming the ATmega328

Programming steps:

1. Algorithm design
2. Code writing
3. Compilation
4. Upload via bootloader
5. Execution

8.1 High-Level vs Low-Level

High-level (Arduino):

```
digitalWrite(pin, HIGH);
```

Low-level (Registers):

```
PORTB |= 0b00100000;
```

Timing comparison:

- digitalWrite(): 50 µs
- Register access: 2 µs

> Low-level programming is **faster and deterministic**.

9 Programming Best Practices

- Choose correct data types
- Avoid unnecessary `int`
- Understand global vs local variables
- Minimize memory footprint

10 Key Exam Takeaways

- Embedded system definition
- MCU vs µP vs SoC
- Harvard vs Von Neumann
- ATmega328 architecture
- Arduino pin mapping
- Deterministic execution

11 Arduino Development Board — ATmega328P

11.1 Arduino Family Overview

The Arduino family consists of low-cost development boards based on Atmel AVR microcontrollers. The board used in this course is the **Arduino Duemilanove**, based on the **ATmega328P** with a USB interface provided by an **FT232 (FTDI) bridge**.

> Aside from power regulation and a bootloader, an Arduino board is essentially an **ATmega328 microcontroller**.

11.2 Electrical Characteristics

Microcontroller:

- ATmega328P — 8-bit AVR
- Clock frequency: 16 MHz

Operating voltage:

- 5 V logic

Power supply options:

- USB power (5 V regulated)
- External supply: 7–12 V recommended (limits 6–20 V)
- Vin pin for external battery (e.g., 9 V)

Power output pins:

- 5V: regulated 5 V (40 mA max)
- 3V3: regulated 3.3 V (50 mA max via FTDI)
- GND

11.3 Memory Organization

- Flash memory: 32 KB (including 2 KB bootloader)
- SRAM: 2 KB (runtime variables)
- EEPROM: 1 KB (non-volatile storage)

> Flash stores the program, SRAM stores variables, EEPROM preserves data after power-off.

12 Arduino Pin-Outs and I/O

12.1 General I/O Capabilities

- Total I/O pins: 20
- 14 Digital I/O
- 6 Analog inputs
- Logic level: 5 V
- Max current per pin: 40 mA
- Internal pull-up resistors: 20–50 kΩ (disabled by default)

12.2 Digital and Analog Pins

Digital I/O:

- 14 digital pins
- 6 PWM outputs (8-bit resolution)

Analog I/O:

- 6 analog inputs
- 10-bit ADC resolution
- Input range: 0–5 V

12.3 Specialized Pins

- Serial (UART): pins 0 (RX), 1 (TX)
- External interrupts: pins 2 and 3
- PWM: pins 3, 5, 6, 9, 10, 11
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)
- I^2C (TWI): 4 (SDA), 5 (SCL)
- AREF: analog reference voltage
- RESET: active low
- Built-in LED: pin 13

> Pins 0 and 1 are shared with USB serial communication — avoid using them during debugging.

13 Arduino Programming Model

13.1 Programming Environment

Arduino can be programmed using:

- Arduino IDE (C/C++ subset)
- AVR Studio
- avr-gcc + avrdude (Linux)
- MATLAB support package
- Simulink (automatic code generation)

13.2 Sketch Structure

Arduino programs are called **sketches** and must contain:

```
void setup() {
}
void loop() {
}
```

- `setup()`: executed once at startup
- `loop()`: executed repeatedly

13.3 Core Arduino Functions

Digital I/O:

- `pinMode(pin, INPUT/OUTPUT)`
- `digitalWrite(pin, HIGH/LOW)`
- `digitalRead(pin)`

Analog I/O:

- `analogRead(pin)` (0–1023)
- `analogWrite(pin, value)` (PWM: 0–255)

Timing:

- `delay(ms)`
- `millis()`

Serial communication:

- `Serial.begin(baud)`
- `Serial.print()`
- `Serial.println()`
- `Serial.read()`

14 Example: LED Blink Program

```
void setup() {
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

> Pin 13 is connected to the on-board LED on most Arduino boards.

15 Arduino with MATLAB and Simulink

15.1 MATLAB Integration

Arduino can be controlled directly from MATLAB using:

- Arduino support package
- USB communication with server firmware

Allows:

- Interactive control
- Data acquisition
- Rapid prototyping

15.2 Simulink Code Generation

- Graphical model-based design
- Automatic C/C++ code generation
- Direct upload to Arduino

Two approaches:

1. Simulink Support Package (basic)
2. Simulink Coder + Embedded Coder (advanced)

> Simulink enables rapid control prototyping for robotics and embedded systems.

15.3 Arduino Target Blocks

Available blocks include:

- Analog Input / PWM Output
- Digital Input / Output
- Serial Receive / Transmit
- Servo Read / Write

Example application:

- Multi-mode LED blinking
- Servo motor control
- Real-time I/O control

16 Digital Filters and FIR Theory

16.1 Numerical (Digital) Filters

A **digital filter** is a discrete-time system that modifies a signal by **selectively amplifying or attenuating frequency components**.

Digital filters are preferred over analog filters because:

- High design flexibility
- Stable and repeatable behavior
- Easy reconfiguration
- Precise control of frequency response

Typical filter types:

- Low-pass
- High-pass
- Band-pass
- Band-stop

17 Finite Impulse Response (FIR) Filters

17.1 Definition

A **Finite Impulse Response (FIR) filter** is a digital filter whose **impulse response has a finite duration**.

Key properties:

- Output depends only on present and past inputs
- No feedback (non-recursive)
- Always stable
- Can achieve linear phase

> FIR filters are **non-recursive** and **unconditionally stable**.

17.2 Mathematical Model of an FIR Filter

The output signal is defined by a convolution sum:

$$y(n) = \sum_{k=0}^{M} b_k \, x(n-k)$$

Where:

- $x(n)$: input signal
- $y(n)$: output signal
- b_k : FIR filter coefficients
- M : filter order

Transfer function:

$$H(z) = \sum_{k=0}^{M} b_k z^{-k}$$

> FIR filters have **only zeros**, no poles.

18 Frequency Analysis and FFT

18.1 Discrete-Time Signals and Sampling

A continuous signal becomes discrete by sampling:

$$T_e = \frac{1}{f_s}$$

Where:

- T_e : sampling period
- f_s : sampling frequency

The maximum observable frequency is the **Nyquist frequency**:

$$f_{Nyquist} = \frac{f_s}{2}$$

18.2 Fast Fourier Transform (FFT)

The FFT computes the **frequency spectrum** of a discrete signal.

Purpose of FFT:

- Identify frequency components
- Analyze spectral content
- Design and validate filters

Why only half of the spectrum is plotted:

- Real signals have symmetric spectra
- Negative frequencies mirror positive ones

Only the frequency range $[0, f_s/2]$ contains unique information.

19 Low-Pass FIR Filter Design

19.1 Objective of a Low-Pass Filter

A low-pass filter:

- Passes low frequencies
- Attenuates high frequencies

Key frequency parameters:

- f_c : cutoff frequency
- f_{pass} : passband edge
- f_{stop} : stopband edge

19.2 Amplitude Specifications

- A_{pass} : maximum ripple in passband (dB)
- A_{stop} : minimum attenuation in stopband (dB)

Lower ripple and higher attenuation require a **higher filter order**.

20 FIR Filter Design Principles

20.1 Filter Order

The **order** M determines:

- Transition sharpness
- Computational cost
- Memory usage

Higher order:

- Better frequency selectivity
- Higher delay
- More computations

20.2 Equiripple FIR Filters

Equiripple FIR filters are designed so that:

- Ripple is evenly distributed
- Maximum error is minimized

Properties:

- Optimal approximation (minimax)
- Linear phase
- Efficient for sharp transitions

Equiripple FIR filters minimize the **maximum deviation** in frequency response.

21 Impulse and Step Responses

21.1 Impulse Response

The impulse response $h(n)$:

- Fully characterizes the FIR filter
- Equals the filter coefficients b_k

21.2 Step Response

The step response shows:

- Transient behavior
- Filter delay
- Overshoot or ringing

A linear-phase FIR filter produces a constant group delay.

22 Implementation Insight (Embedded Context)

22.1 FIR Filters in Embedded Systems

In embedded systems (e.g., Arduino):

- FIR filters are implemented using convolution
- Each output sample requires $M + 1$ multiplications

Trade-offs:

- FIR: stable, predictable, higher cost
- IIR: efficient, but potentially unstable

FIR filters are preferred in real-time embedded systems when **stability and phase linearity** are critical.