Embedded Artificial Intelligence Complete Study Guide

Exam Preparation Notes

November 7, 2025

Contents

1 Introduction to Embedded AI

Embedded AI involves integrating artificial intelligence capabilities directly into embedded systems (microcontrollers, edge devices) rather than relying on cloud computing. This enables real-time processing, reduced latency, privacy preservation, and operation without continuous internet connectivity.

2 Training Word Recognition Using Edge Impulse

2.1 Overview

Edge Impulse is a platform for developing machine learning models optimized for edge devices like Arduino BLE Sense Nano. It enables on-device audio classification and word recognition.

2.2 Using Mobile for Data Collection

Steps:

1. Create an Edge Impulse account and project
2. Use Edge Impulse mobile app or web browser
3. Record audio samples of target words (minimum 10 samples per word)
4. Label each recording with appropriate class name
5. Ensure varied conditions (different speakers, distances, noise levels)

2.3 Using Arduino BLE Sense Nano

Hardware Features:

- Built-in MP34DT05 digital microphone
- nRF52840 processor with BLE connectivity
- Suitable for low-power AI applications

Steps:

1. Install Edge Impulse CLI tools
2. Flash Edge Impulse firmware to Arduino
3. Connect Arduino via USB
4. Run `edge-impulse-daemon` to link device
5. Collect training data directly from microphone
6. Upload to Edge Impulse platform
7. Design impulse (preprocessing + learning blocks)
8. Train model and deploy back to Arduino

3 Data Augmentation

3.1 What is Data Augmentation?

Data augmentation is the process of artificially expanding training datasets by applying transformations to existing samples without collecting new data.

3.2 Addition of Noise

Why Add Noise?

- **Robustness:** Models must work in real-world noisy environments
- **Generalization:** Prevents overfitting to clean training data
- **Data diversity:** Simulates various acoustic conditions
- **Limited data:** Increases effective dataset size

Types of Noise:

- White noise (random across frequencies)
- Background noise (environmental sounds)
- Pink noise (1/f noise, more natural)
- Real-world recordings (traffic, crowds)

3.3 Other Augmentation Techniques

- Time stretching (speed variation)
- Pitch shifting
- Time shifting (temporal offset)
- Volume adjustment

4 Sample/Data Matching

4.1 Concept

Sample matching ensures that training and testing data distributions are similar. For embedded AI:

- Match sampling rates between collection and deployment
- Ensure consistent audio format (bit depth, channels)
- Balance classes (equal samples per word)
- Split data properly (80% train, 20% test)

4.2 Importance

- Prevents domain mismatch between training and inference
- Ensures model performance metrics are realistic
- Critical for successful deployment on resource-constrained devices

5 Fourier Transform

5.1 Basic Concept

The Fourier Transform decomposes a time-domain signal into its constituent frequencies. It reveals which frequencies are present in a signal and their amplitudes.

5.2 Mathematical Definition

For a continuous signal $x(t)$:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

Where:

- $X(f)$ is frequency domain representation
- f is frequency
- j is imaginary unit

5.3 Why Use in Audio Processing?

- Audio signals are naturally analyzed in frequency domain
- Human hearing perceives frequencies (pitch)
- Speech contains distinct frequency patterns for different sounds
- Enables feature extraction for ML models

6 Fast Fourier Transform (FFT)

6.1 Overview

FFT is an efficient algorithm to compute the Discrete Fourier Transform (DFT). It reduces computational complexity from $O(N^2)$ to $O(N \log N)$.

6.2 Why FFT for Embedded Systems?

- **Speed:** Much faster than direct DFT computation
- **Real-time:** Enables live audio processing on microcontrollers
- **Memory efficient:** In-place algorithms available
- **Power efficient:** Less computation = less energy

6.3 FFT Implementation

Most common: Cooley-Tukey algorithm (radix-2)

- Works best with sample sizes of 2^n (256, 512, 1024)
- Recursively divides signal into smaller segments
- Combines results using butterfly operations

7 Best Sampling Rate

7.1 Nyquist Theorem

Sampling rate must be at least twice the highest frequency in the signal:

$$f_s \geq 2 \times f_{max}$$

7.2 Common Sampling Rates for Speech

- **8 kHz:** Telephone quality, captures up to 4 kHz
- **16 kHz:** Wideband speech, good for recognition (commonly used)
- **44.1 kHz:** CD quality, overkill for speech
- **48 kHz:** Professional audio

7.3 Choosing Sampling Rate

For speech recognition: 16 kHz is optimal because:

- Human speech frequencies: 80 Hz - 8 kHz
- Captures all phonetic information
- Lower than 16 kHz loses intelligibility
- Higher than 16 kHz wastes memory/computation
- Trade-off between quality and resource constraints

8 MFCC (Mel-Frequency Cepstral Coefficients)

8.1 What is MFCC?

MFCC is a feature extraction technique that represents audio signals in a way that mimics human auditory perception.

8.2 Why MFCC?

- **Perceptual scale:** Mel scale matches human hearing (logarithmic)
- **Dimensionality reduction:** Compresses spectral information
- **Robust:** Less sensitive to noise and speaker variation
- **Industry standard:** Proven for speech recognition
- **Compact:** Typically 13-40 coefficients represent entire spectrum

8.3 MFCC Computation Steps

1. **Pre-emphasis:** Amplify high frequencies
2. **Framing:** Divide signal into 20-40ms windows
3. **Windowing:** Apply Hamming window to each frame
4. **FFT:** Convert to frequency domain
5. **Mel Filter Bank:** Apply triangular filters on Mel scale
6. **Logarithm:** Take log of filter bank energies
7. **DCT:** Discrete Cosine Transform to get MFCCs

8.4 Mel Scale Formula

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

Where f is frequency in Hz and m is Mel frequency.

9 MFCC vs MFE (Mel-Filter Bank Energies)

9.1 Comparison Table

Aspect	**MFCC**	**MFE**
Final step	DCT applied	No DCT
Decorrelation	Coefficients are decorrelated	Filter banks are correlated
Dimensionality	Lower (13-40 coefficients)	Higher (20-40 filter banks)
Computation	More complex (includes DCT)	Simpler, faster
Best for	Speech recognition, speaker ID	Deep learning (CNNs learn correlations)
Memory usage	Less	More
Edge devices	Preferred for classical ML	Preferred for neural networks

9.2 When to Use Each

- **MFCC:** Classical ML algorithms (SVM, Random Forest), very limited memory
- **MFE:** Neural networks (they can learn feature relationships), slightly more resources available

10 ATMega328 and ATMega328P

10.1 Overview

ATMega328/328P are 8-bit AVR microcontrollers used in Arduino Uno. The 'P' variant has lower power consumption.

10.2 Key Specifications

- **Architecture:** 8-bit RISC
- **Flash memory:** 32 KB (0.5 KB bootloader)
- **SRAM:** 2 KB
- **EEPROM:** 1 KB
- **Clock speed:** Up to 20 MHz
- **I/O pins:** 23 programmable
- **ADC:** 10-bit, 6 channels
- **Timers:** Two 8-bit, one 16-bit

10.3 Pinout Description

Power Pins:

- **VCC (Pin 7):** Supply voltage (+5V)
- **GND (Pins 8, 22):** Ground
- **AVCC (Pin 20):** Analog supply voltage
- **AREF (Pin 21):** Analog reference for ADC

Port B (Pins 14-19, 9-10):

- PB0-PB5: Digital I/O
- PB6, PB7 (Pins 9-10): Crystal oscillator connections (XTAL1, XTAL2)

Port C (Pins 23-28):

- PC0-PC5: Digital I/O or Analog inputs (ADC0-ADC5)
- PC6 (Pin 1): Reset pin (active low)

Port D (Pins 2-6, 11-13):

- PD0-PD7: Digital I/O
- PD0, PD1: UART (RX, TX)
- PD2, PD3: External interrupts
- PD3, PD5, PD6: PWM outputs

10.4 Use Cases of Each Pin Type

- **Digital I/O:** LEDs, buttons, sensors with digital output
- **Analog input:** Temperature sensors, potentiometers, light sensors
- **PWM:** Motor speed control, LED dimming, servo control
- **UART:** Serial communication with PC or other devices
- **SPI/I2C:** Communication with sensors, displays, memory
- **Interrupts:** Respond to urgent events without polling

11 Serial vs Parallel Communication

11.1 Serial Communication

Characteristics:

- Data sent one bit at a time over single wire
- Examples: UART, SPI, I2C, USB
- Slower data transfer per wire

Advantages:

- **Cost:** Fewer wires, cheaper cables
- **Long distance:** Can transmit over longer distances
- **Pin count:** Saves microcontroller pins
- **Simplicity:** Easier routing on PCB

11.2 Parallel Communication

Characteristics:

- Multiple bits sent simultaneously over multiple wires
- Example: 8-bit parallel bus sends 8 bits at once

Advantages:

- **Speed:** Much faster data transfer
- **Simple timing:** All bits arrive together

Disadvantages:

- **Cost:** Expensive (many wires, connectors)
- **Space:** Bulky cables and connectors
- **Pin usage:** Uses many microcontroller pins
- **Signal integrity:** Crosstalk and skew over distance

11.3 Cost-wise Comparison

Factor	Serial	Parallel
Cable cost	Low	High
Connector cost	Low	High
PCB routing	Simple	Complex
Pin count	Low (saves cost)	High (expensive)

Conclusion: Serial is preferred in modern embedded systems for cost-effectiveness.

12 Digital and Analog I/O

12.1 Digital Input

- Reads HIGH (5V/3.3V) or LOW (0V)
- Used for: Buttons, switches, digital sensors
- Configuration: Set DDR register bit to 0 (input)
- Pull-up resistors prevent floating inputs

12.2 Digital Output

- Outputs HIGH or LOW
- Used for: LEDs, relays, digital control signals
- Configuration: Set DDR register bit to 1 (output)
- Write to PORT register to set output value

12.3 Analog Input

- Reads continuous voltage (0-5V on ATMega328)
- Uses ADC (Analog-to-Digital Converter)
- 10-bit resolution: 1024 levels (0-1023)
- Used for: Potentiometers, temperature sensors, light sensors
- Resolution: $\frac{5V}{1024} \approx 4.88$ mV per step

12.4 Analog Output

- True analog output requires DAC (not on ATMega328)
- Simulated using PWM (Pulse Width Modulation)
- PWM varies duty cycle to create average voltage
- Used for: LED brightness, motor speed control
- Frequency typically 490 Hz or 980 Hz on Arduino

13 Voltage Usage and Precautions

13.1 Critical Voltage Considerations

- **Operating voltage:** ATMega328 operates at 5V (or 3.3V variants)
- **Maximum voltage:** Never exceed VCC + 0.5V on any pin
- **Current limits:** Max 40 mA per I/O pin, 200 mA total

13.2 Safety Precautions

- Use current-limiting resistors with LEDs (typically 220-1k)
- Level shifters for 3.3V 5V interfacing
- Protection diodes for inductive loads (motors, relays)
- Decoupling capacitors (100nF) near VCC pins
- Never connect outputs together (risk of short circuit)
- Always check polarity of polarized components

13.3 LED Current Calculation

For an LED with forward voltage $V_f = 2V$:

$$R = \frac{V_{CC} - V_f}{I_{LED}} = \frac{5V - 2V}{20mA} = 150\Omega$$

Use standard 220 resistor for safety margin.

14 Microchip Studio

14.1 Overview

Microchip Studio (formerly Atmel Studio) is the official IDE for AVR microcontrollers. It provides tools for writing, compiling, and debugging assembly and C code.

14.2 Common Microcontroller Schematics

Typical microcontroller circuit includes:

- **Power supply:** VCC, GND with decoupling capacitors
- **Reset circuit:** Pull-up resistor on RESET pin, optional button
- **Clock:** External crystal (16 MHz) with 22pF capacitors, or internal RC oscillator
- **Programming header:** ISP connector (MISO, MOSI, SCK, RESET, VCC, GND)
- **I/O connections:** Peripheral devices on port pins

14.3 Three Types of Memory

14.3.1 1. Flash Memory (Program Memory)

- **Size:** 32 KB in ATMega328
- **Purpose:** Stores program code (.hex file)
- **Type:** Non-volatile (retains data without power)
- **Endurance:** 10,000 write cycles
- **Why needed:** Persistent storage of firmware

14.3.2 2. SRAM (Static RAM)

- **Size:** 2 KB in ATMega328
- **Purpose:** Stores variables, stack, runtime data
- **Type:** Volatile (lost when power off)
- **Speed:** Very fast access
- **Why needed:** Fast temporary storage during program execution

14.3.3 3. EEPROM (Electrically Erasable Programmable ROM)

- **Size:** 1 KB in ATMega328
- **Purpose:** Stores configuration, calibration data, user settings
- **Type:** Non-volatile
- **Endurance:** 100,000 write cycles
- **Why needed:** Persistent storage of data that changes occasionally

14.4 Why Three Types?

- **Different access patterns:** Code (read-only), variables (fast read/write), settings (occasional write)
- **Cost optimization:** Each memory type has different cost/performance
- **Endurance:** EEPROM handles more writes than Flash
- **Volatility needs:** Flash/EEPROM persist, SRAM doesn't need to
- **Speed vs persistence:** SRAM fastest but volatile; Flash/EEPROM slower but persistent

15 Discrete vs Digital Signals

15.1 Discrete Signals

- Values at specific time instances (sampled)
- Can have continuous amplitude values
- Example: Sampled audio signal before quantization

15.2 Digital Signals

- Both time and amplitude are discrete (quantized)
- Example: Digital audio (16-bit samples at 16 kHz)
- What microcontrollers actually process
- Represented by finite binary numbers

15.3 Inputs to Microcontrollers

- Analog sensors produce continuous signals
- ADC converts to digital (sampling + quantization)
- Digital sensors output binary values directly
- Microcontroller processes only digital signals internally

16 Proteus Simulation

16.1 Overview

Proteus is a simulation tool for electronic circuits and microcontrollers. It allows testing designs before physical implementation.

16.2 Common Components and Use Cases

Passive Components:

- **Resistors:** Current limiting, voltage division, pull-up/down
- **Capacitors:** Filtering, decoupling, timing circuits
- **Inductors:** Filtering, energy storage
- **Crystals:** Clock generation (no polarity - can connect either way)

Active Components:

- **LEDs:** Visual indicators (note polarity!)
- **Transistors:** Switching, amplification
- **7-segment displays:** Numeric output
- **LCDs:** Text/graphics display
- **Motors:** Actuators

Input Devices:

- **Buttons/Switches:** User input
- **Potentiometers:** Variable analog input
- **Sensors:** Temperature, light, etc.

16.3 Crystal Oscillator

Does crystal have polarity?

- **NO**, crystals are non-polarized
- Can be connected in either orientation
- Both pins are electrically equivalent
- Always use two 22pF capacitors to ground

17 ATMega328P I/O Simulation Example

17.1 Simple LED Blinking with Assembly

Assembly Code:

```
.include "m328pdef.inc"  ; Include ATMega328P definitions

.org 0x0000              ; Reset vector
    rjmp MAIN            ; Jump to main program

MAIN:
    ; Configure PB5 (Arduino pin 13) as output
    ldi r16, 0b00100000  ; Load immediate value
    out DDRB, r16        ; Set DDRB register (PB5 as output)

LOOP:
    ; Turn LED ON
    sbi PORTB, 5         ; Set bit 5 in PORTB (turn on)
    rcall DELAY          ; Call delay subroutine

    ; Turn LED OFF
    cbi PORTB, 5         ; Clear bit 5 in PORTB (turn off)
    rcall DELAY          ; Call delay subroutine

    rjmp LOOP            ; Repeat forever

DELAY:
    ; Delay subroutine
    ldi r17, 41          ; Outer loop counter
OUTER:
    ldi r18, 200         ; Middle loop counter
MIDDLE:
    ldi r19, 250         ; Inner loop counter
INNER:
    dec r19              ; Decrement inner counter
    brne INNER           ; Branch if not zero
    dec r18              ; Decrement middle counter
    brne MIDDLE          ; Branch if not zero
    dec r17              ; Decrement outer counter
```

```
    brne OUTER              ; Branch if not zero
    ret                     ; Return from subroutine
```

17.2 Why Delay is Needed?

- **Visible effect:** Without delay, LED toggles too fast to see (microseconds)
- **Human perception:** Need 100ms minimum for visible flashing
- **CPU speed:** ATMega328 at 16 MHz executes millions of instructions per second
- **Control timing:** Delays create specific time intervals for various applications

17.3 Delay and Hz Relationship

- **Frequency (Hz):** Number of complete cycles per second
- **Period (T):** Time for one complete cycle
- Relationship: $f = \frac{1}{T}$ or $T = \frac{1}{f}$

Example:

- If delay is 500 ms (0.5 s) for ON and 500 ms for OFF
- Total period: T = 1 second
- Frequency: $f = \frac{1}{1s} = 1$ Hz (1 blink per second)

Calculation: For a 16 MHz clock, each instruction takes $\frac{1}{16\times10^6} = 62.5$ ns
To create a 1-second delay, need approximately:

$$\frac{1 \text{ second}}{62.5 \text{ ns}} = 16,000,000 \text{ instruction cycles}$$

Nested loops multiply iteration counts to achieve long delays.

18 General Purpose Registers vs Special Function Registers

18.1 General Purpose Registers (GPR)

- **Location:** R0-R31 (32 registers in AVR)
- **Purpose:** Temporary data storage, arithmetic operations
- **Flexibility:** Can store any 8-bit value
- **Speed:** Fastest access (1 clock cycle)
- **Examples:** r16, r17, r18 in above assembly code

18.2 Special Function Registers (SFR)

- **Location:** Memory-mapped I/O space
- **Purpose:** Control and status of peripherals
- **Examples:** DDRB, PORTB, ADCSRA, TCCR1A
- **Hardware control:** Each bit controls specific hardware feature

18.3 Why GPR Cannot Work as SF Registers?

1. **Hardware connection:** SFRs are directly connected to peripheral hardware (ADC, timers, ports). GPRs have no hardware connections.
2. **Memory mapping:** SFRs occupy specific addresses in I/O space that hardware monitors. GPRs are in register file, separate from I/O.
3. **Side effects:** Writing to SFR triggers hardware actions (e.g., starting ADC conversion). GPRs have no side effects.
4. **Read behavior:** Reading SFR often returns hardware status (e.g., ADC result). GPRs only return stored values.
5. **Bit significance:** Each SFR bit has specific hardware meaning. GPR bits are just data.

Example:

- Writing to PORTB (SFR) changes output pin voltages immediately
- Writing to r16 (GPR) only stores value in CPU register, no external effect

19 Xeltek SuperPro 610P Universal Programmer

19.1 Overview

The Xeltek SuperPro 610P is a universal device programmer used to burn (program) firmware into microcontrollers, memory chips, and other programmable devices.

19.2 What is Burning/Programming?

- Transferring compiled code (.hex file) into Flash memory
- Makes code permanent on the chip
- Required before microcontroller can execute program
- Called "burning" historically from EPROM days

19.3 Hardware Setup Steps

1. Connect SuperPro 610P to PC via USB
2. Install drivers and programming software
3. Insert ATMega328P into appropriate socket/adapter
4. Ensure correct orientation (Pin 1 indicator)
5. Close the ZIF (Zero Insertion Force) socket lever

19.4 Programming Steps

1. **Launch software:** Open Xeltek programming software
2. **Select device:**
   - Search for "ATMega328P"
   - Select exact variant (DIP-28 package)
   - Verify pin count and package type
3. **Load hex file:**
   - Click "Load" or "Open File"
   - Navigate to .hex file from Microchip Studio
   - Software displays code/data to be programmed
4. **Configure fuses (optional):**
   - Set clock source (internal/external)
   - Configure brown-out detection
   - Set startup time
5. **Verify chip:**
   - Click "Check ID" or "Detect"
   - Confirms chip is properly inserted
   - Verifies communication
6. **Erase (if needed):**
   - Click "Erase" to clear existing code
   - Necessary for reprogramming
7. **Program:**
   - Click "Program" or "Write"
   - Software burns hex file to Flash memory
   - Progress bar shows completion

8. **Verify:**
    - Automatically or manually verify
    - Reads back Flash and compares with hex file
    - Ensures successful programming
    - Display shows "Verify OK" or similar
9. **Remove chip:**
    - Open ZIF socket
    - Carefully remove ATMega328P
    - Insert into target circuit

19.5 File Formats

- **.hex (Intel HEX):** Most common, contains address and data
- **.bin (Binary):** Raw binary data, also works
- Both contain same machine code in different formats
- Hex format includes checksums and addressing

19.6 Common Issues

- **Device not detected:** Check socket insertion, try different socket
- **Verification failed:** Bad chip, incorrect device selection, or hardware issue
- **Programming error:** Check power supply, clean socket contacts

20 Assembly Code Development Workflow

20.1 Complete Steps in Microchip Studio

20.1.1 Step 1: Create New Project

1. Launch Microchip Studio
2. File → New → Project
3. Select "Assembler" → "AVR Assembler Project"
4. Name project and choose location
5. Click OK

20.1.2 Step 2: Select Device

1. Device selection dialog appears
2. Search for "ATMega328P"
3. Select it and click OK
4. Project is created with template

20.1.3 Step 3: Write Assembly Code

1. Main .asm file opens automatically
2. Write or paste your assembly code
3. Include device definition: `.include "m328pdef.inc"`
4. Save file (Ctrl+S)

20.1.4 Step 4: Build Project

1. Click Build → Build Solution (F7)
2. Assembler compiles code
3. Check "Output" window for errors
4. If errors: fix code and rebuild
5. If successful: "Build succeeded" message appears

20.1.5 Step 5: Locate .hex File

1. Navigate to project folder
2. Open Debug or Release subfolder
3. Find `ProjectName.hex` file
4. This file is ready for programming
5. Copy to accessible location

20.2 Understanding Build Output

- **.hex:** Intel HEX format for programming
- **.eep:** EEPROM data (if any)
- **.lst:** Listing file (code with addresses)
- **.map:** Memory map (symbol locations)
- **.elf:** Debug information (for simulators)

20.3 Can .bin Files Work?

Yes! Binary files can also be programmed:

- Contains raw machine code
- No address information (assumes start at 0x0000)
- Smaller file size than .hex
- Some programmers prefer .hex for its error checking
- Xeltek 610P supports both formats

21 Practical Interface Examples

21.1 LED Interface

Circuit:

- ATMega328P pin (e.g., PB5) → 220 resistor → LED anode → LED cathode → GND
- Resistor limits current to safe value (20mA)

Code concept:

- Set DDR bit to 1 (output mode)
- Set PORT bit to 1 (LED ON)
- Clear PORT bit to 0 (LED OFF)

21.2 Button Interface

Circuit:

- Button between pin and GND
- Internal or external pull-up resistor to VCC
- Pin reads HIGH when button not pressed
- Pin reads LOW when button pressed

Code concept:

- Set DDR bit to 0 (input mode)
- Enable internal pull-up (set PORT bit to 1)
- Read PIN register to check button state

21.3 Seven-Segment Display Interface

Circuit:

- 7 segments + 1 decimal point = 8 outputs
- Each segment through current-limiting resistor
- Common cathode: segments connect to pins, common to GND
- Common anode: segments connect to pins through resistors, common to VCC

Code concept:

- Configure entire port as output (DDRB = 0xFF)
- Use lookup table for digit patterns
- Write pattern to PORT register to display digit

Example patterns (common cathode):

```
; Digit patterns (segments: gfedcba)
DIGITS:
    .db 0b00111111  ; 0
    .db 0b00000110  ; 1
    .db 0b01011011  ; 2
    .db 0b01001111  ; 3
    .db 0b01100110  ; 4
    .db 0b01101101  ; 5
    .db 0b01111101  ; 6
    .db 0b00000111  ; 7
    .db 0b01111111  ; 8
    .db 0b01101111  ; 9
```

21.4 Motor Control with Transistor

Circuit:

- Pin → Base resistor (1k) → NPN transistor base
- Transistor collector → Motor → VCC
- Transistor emitter → GND
- Flyback diode across motor (cathode to VCC)

Why flyback diode?

- Motors are inductive loads
- Generate voltage spike when turned off
- Diode protects microcontroller from reverse voltage

22 General Understanding Scenarios

22.1 Scenario 1: Choosing Sampling Rate

Question: You need to capture human voice. What sampling rate would you choose and why?

Answer:

- Choose 16 kHz
- Human speech fundamental: 80-250 Hz
- Speech harmonics extend to 8 kHz
- By Nyquist theorem: need at least 16 kHz
- Higher rates (44.1 kHz) waste resources
- Lower rates (8 kHz) lose intelligibility
- 16 kHz is standard for speech recognition

22.2 Scenario 2: Memory Type Selection

Question: Where would you store: (a) program code, (b) temporary sensor reading, (c) calibration constants?

Answer:

- **(a) Flash:** Program code - non-volatile, read frequently
- **(b) SRAM:** Sensor reading - volatile, changes constantly, fast access needed
- **(c) EEPROM:** Calibration - non-volatile, written rarely, read at startup

22.3 Scenario 3: Communication Protocol Selection

Question: You need to connect 5 sensors to a microcontroller with limited pins. Serial or parallel?

Answer:

- Choose Serial (I2C or SPI)
- Saves pins: I2C uses only 2 pins (SDA, SCL) for all sensors
- Parallel would need 40+ pins (8 data + control per sensor)
- More cost-effective cabling
- Easier PCB routing
- Better for expandability

22.4 Scenario 4: Delay Calculation

Question: You want LED to blink at 2 Hz with 16 MHz clock. What is the period and how to achieve delay?

Answer:

- Frequency = 2 Hz means 2 complete cycles per second
- Period $T = \frac{1}{2} = 0.5$ seconds per cycle
- Each cycle: 0.25s ON + 0.25s OFF
- Need 250 ms delays
- At 16 MHz: $16 \times 10^6 \times 0.25 = 4,000,000$ cycles
- Use nested loops or timer interrupts

22.5 Scenario 5: Why Data Augmentation?

Question: You have 50 clean voice samples. Why add noise during training?

Answer:

- Real-world deployment has noise (background, interference)
- Model trained on clean data fails in noisy environment
- Adding noise during training improves robustness
- Prevents overfitting to clean samples
- Better generalization to unseen conditions
- More cost-effective than collecting thousands of real samples

22.6 Scenario 6: MFCC vs MFE Selection

Question: You're deploying speech recognition on Arduino with limited memory. Classical ML model. MFCC or MFE?

Answer:

- Choose MFCC
- Lower dimensionality (13 coefficients vs 40 filter banks)
- Decorrelated features better for classical ML
- Uses less memory (critical on Arduino)
- Industry standard for speech with traditional algorithms
- MFE better for neural networks (can learn correlations)

22.7 Scenario 7: Voltage Protection

Question: You connect a 12V motor directly to ATMega328P output pin. What happens?

Answer:

- **DAMAGE!** Microcontroller likely destroyed
- ATMega328P maximum voltage: 5V + 0.5V
- 12V exceeds absolute maximum rating
- Current also exceeds 40 mA limit
- **Solution:** Use transistor/MOSFET driver
- Pin controls transistor, transistor switches motor

22.8 Scenario 8: Crystal Selection

Question: Internal oscillator is 8 MHz. Why use external 16 MHz crystal?

Answer:

- **Accuracy:** Crystal much more accurate (±50 ppm vs ±10%)
- **Stability:** Less affected by temperature
- **Speed:** 16 MHz faster than 8 MHz internal
- **Critical for:** UART communication (baud rate accuracy)
- **Critical for:** Precise timing applications
- Internal OK for: Non-critical timing, cost-sensitive designs

23 Tips for Exam Success

23.1 Key Concepts to Remember

1. **Interfacing:** Know how to connect LEDs, buttons, displays - include resistors!
2. **Schematics:** Understand ATMega328P pinout, power connections, crystal circuit
3. **Delays:** Relationship between frequency, period, and delay implementation
4. **Memory types:** When to use Flash, SRAM, EEPROM
5. **Assembly basics:** DDR for direction, PORT for output, PIN for input
6. **GPR vs SFR:** Understand the fundamental difference
7. **Practical workflow:** Microchip Studio → Build → .hex file → Xeltek programmer
8. **Signal processing:** FFT, MFCC, sampling rate selection
9. **Data augmentation:** Why noise is added to training data
10. **Safety:** Voltage limits, current limits, protection components

23.2 Common Exam Question Types

23.2.1 Type 1: Circuit Design

"Draw circuit to connect 8 LEDs to Port B with proper current limiting."

- Show all 8 pins (PB0-PB7)
- 220 resistor for each LED
- LED orientation (anode to resistor, cathode to GND)
- VCC and GND connections

23.2.2 Type 2: Register Configuration

"Write assembly to configure PC0-PC3 as outputs and PC4-PC7 as inputs."

```
ldi r16, 0b00001111  ; Lower nibble output, upper input
out DDRC, r16        ; Configure direction
ldi r16, 0b11110000  ; Enable pull-ups on inputs
out PORTC, r16       ; Set pull-up resistors
```

23.2.3 Type 3: Timing Calculations

"Calculate delay needed for 5 Hz blinking with 16 MHz clock."

- Period = $1/5 = 0.2$ seconds
- Half period (ON time) = 0.1 seconds = 100 ms
- Clock cycles needed = $16 \times 10^6 \times 0.1 = 1,600,000$
- Design nested loops to achieve this count

23.2.4 Type 4: Conceptual Understanding

"Why is 16 kHz chosen for speech recognition?"

- Speech frequencies up to 8 kHz
- Nyquist: need $2\times$ = 16 kHz
- Balance between quality and resources
- Standard in industry

23.2.5 Type 5: Troubleshooting

"LED doesn't light up. What could be wrong?"

- DDR not configured (pin still in input mode)
- PORT bit not set (output is LOW)
- LED reversed (polarity)
- No resistor or wrong resistor value
- Faulty LED or loose connection
- Pin damaged from overvoltage

23.3 Quick Reference Formulas

Frequency-Period:

$$f = \frac{1}{T} \quad T = \frac{1}{f}$$

LED Current:

$$R = \frac{V_{supply} - V_{LED}}{I_{desired}}$$

Nyquist Sampling:

$$f_{sample} \geq 2 \times f_{max_signal}$$

Mel Scale:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

ADC Resolution:

$$V_{step} = \frac{V_{ref}}{2^{bits}}$$

23.4 Last-Minute Checklist

Can explain ATMega328P pinout

Know DDR, PORT, PIN register functions

Understand delay-frequency relationship

Can draw basic LED/button circuits

Know three memory types and their purposes

Understand GPR vs SFR difference

Remember why MFCC is used

Know sampling rate selection criteria

Understand data augmentation importance

Remember voltage/current safety limits

Know Xeltek programming workflow

Can explain serial vs parallel tradeoffs

Understand crystal usage (no polarity)

Remember .hex file build process

24 Conclusion

This guide covers the fundamental concepts of Embedded Artificial Intelligence with emphasis on:

- Signal processing for audio (FFT, MFCC, sampling)
- Microcontroller architecture (ATMega328P)
- Practical interfacing and circuit design
- Assembly programming workflow
- Device programming with Xeltek
- ML model deployment on edge devices

Study Strategy:

1. Focus on interfacing and schematics (teacher's emphasis)
2. Practice delay calculations
3. Understand general scenarios thoroughly
4. Review ATMega328P pinout and register usage
5. Remember safety considerations (voltage, current)

Good luck with your exam!