Embedded_session_9

Embedded System Session 9 - SPI & I2C Communication Protocols

---

Created: 2025-07-23
Author: Fares Hesham Mahmoud
Tags: AVR, Embedded system, Communication Protocol, SPI, I2C
Status: #Formatted

اللهمَّ علِّمنا ما ينفعنا، وانفعنا بما علمتنا، وزدنا علمًا. وافتح علينا فتحًا عظيمًا.

Multi-ECU Communication Systems

In complex boards with multiple Electronic Control Units (ECUs), communication protocols enable interaction between units following specific policies.

System Design Criteria

Communication protocols are ranked and rated according to:

1. Complexity - How difficult to implement and maintain
2. Reliability - Fault tolerance and error handling
3. Dependency - System resilience to component failures

> **Design Principles** >
>
> - Complex systems → Split across multiple nodes
> - ECU Power → Must match performance expectations
> - Avoid single points of failure → System shouldn't depend on one ECU

Automotive Industry Structure

Tier	Role	Responsibility
OEM	Original Equipment Manufacturer	Main vehicle manufacturer
Tier 1	Primary Suppliers	Major system integration
Tier 2	Secondary Suppliers	Specialized components (e.g., Vector)
Tier 3	Raw Material Suppliers	Basic materials and components

Protocol Classification

Protocol Types

Type	Examples	Characteristics
Standard Protocols	I2C, CAN	Globally standardized
Common Protocols	UART, SPI	Widely used, not standardized

Communication Medium Comparison

Aspect	Wired	Wireless
Speed	High (matched in modern systems)	High
Cost	Higher (installation, maintenance)	Lower
Mobility	Limited	High
Safety	Better for health	Potential health concerns
Security	More secure	Vulnerable to interference

Protocol Specifications Framework

1. Data Transfer Mode

Mode	Examples	Characteristics
Serial	UART, SPI, I2C, CAN, USB, HDMI	Single data line, slower but simpler
Parallel	Traditional memory buses	Multiple wires, faster but complex

Parallel Challenges:

- Interference: Magnetic fields affect multiple lines
- Data Skew: Timing delays between different wires
- Complexity: Many connections difficult to manage
- Noise: Cross-talk between adjacent wires

2. Network Topology

Topology	Authority Model	Examples
Peer-to-Peer	Equal authority	UART
Single Master, Single Slave	Master controls	Radio
Single Master, Multi Slave	Master manages multiple	SPI, LIN
Multi Master, Multi Slave	Multiple masters	I2C
Multi Master, No Slave	Distributed control	CAN

3. Synchronization

- Synchronous: Frame or clock-based synchronization
- Asynchronous: No shared timing reference
- Dynamic Data Rate: Rate can change during runtime (sync only)

4. Data Direction

- Simplex: One direction only
- Half Duplex: Bidirectional, not simultaneous
- Full Duplex: Simultaneous bidirectional

5. Throughput

Ratio of useful data bits to total frame bits:

$$\text{Trouput} = \frac{\text{Data its}}{\text{rame ize}}$$

Parallel to Serial Conversion >

Shift Register (74595): Converts parallel data to serial transmission
Useful when you have parallel data but need serial communication

SPI (Serial Peripheral Interface)

SPI Specifications

- Serial communication
- Wired medium
- Single Master, Multi Slave topology
- Synchronous (dedicated clock wire)
- Full Duplex communication
- High Throughput (no address overhead)

Hardware Connections

Method 1: Independent Slaves

4-Wire Connection per Slave:

Wire	Function	Description
SCLK	Serial Clock	Master-generated clock signal
MISO	Master In, Slave Out	Data from slave to master
MOSI	Master Out, Slave In	Data from master to slave
SS/CS	Slave Select/Chip Select	Individual slave selection

```
Master                      Slave 1
------                      -------
SCLK  ──────────────────── SCLK
MISO  ──────────────────── MISO
MOSI  ──────────────────── MOSI
SS1   ──────────────────── SS

Master                      Slave 2
------                      -------
SCLK  ──────────────────── SCLK
MISO  ──────────────────── MISO
MOSI  ──────────────────── MOSI
SS2   ──────────────────── SS
```

Characteristics:

- Independent Operation: Adding/removing slaves doesn't affect others
- Pin Limitation: Max slaves = Available GPIO pins on master
- Simultaneous Access: Only one slave active at a time

Method 2: Daisy Chain

3-Wire Shared + Chained Data:

```
Master → Slave1 → Slave2 → Slave3 → Master
       SCLK (shared to all)
       MOSI→MOSI  MISO→MOSI  MISO→MOSI  MISO
       SS (tied to active level)
```

Characteristics:

- **Transfer Delay:** Data takes time to propagate through chain
- **Single Point of Failure:** One failed slave breaks entire chain
- **Scalability:** Easy to add slaves without extra master pins

Slave Selection Methods

Method	Protocol	Description
Unique Address	I2C	Each slave has unique identifier
Message ID	CAN	Broadcast with specific message types
Hardware Selection	SPI	Physical chip select lines

SPI Frame Format & Clock Configuration

Clock Polarity & Phase

- **Clock Polarity (CPOL):** Idle state of clock (high/low)
- **Clock Phase (CPHA):** Data sampling edge (rising/falling)
- **Idle State:** State when no data transfer occurs

SPI Modes

Mode	CPOL	CPHA	Idle Clock	Sample Edge
0	0	0	Low	Rising
1	0	1	Low	Falling
2	1	0	High	Falling
3	1	1	High	Rising

Communication Sequence

1. **Master Write/Toggle/Send:** Initiates transfer and sets data
2. **Slave Read/Capture/Sample:** Reads data on appropriate clock edge

Configuration Types

Type	When Configured	Pros	Cons
Pre-build	Before compilation (#define, headers)	Fast execution, no runtime overhead	Inflexible, requires recompilation
Post-build	During runtime (pointers, variables)	Flexible, adaptive	Runtime overhead, memory usage
Linking	During linking phase	Moderate flexibility	Complex setup

Toolchain Overview

The software development process involves 4 main tools:

Tool	Input	Output	Function	Target Dependency
Preprocessor	file.c	file.i	Text replacement, directive processing	Target Independent
Compiler	file.i	file.asm	High-level to assembly conversion	Target Independent
Assembler	file.asm	file.obj	Assembly to binary conversion	Target Dependent
Linker	file.obj	file.exe	Object linking and address resolution	Target Dependent

Compiler Phases

Frontend (Target Independent)

- **Tokenizer:** Remove comments, create tokens
- **Syntax Analysis:** Check language syntax rules
- **Parsing:** Verify object definitions and usage

Backend (Target Dependent)

- **Code Generation:** Create target-specific assembly
- **Optimizer:** Improve code efficiency and size

Symbol Table Example

Object	Type	State	Logical Address
calc	void→int	needed	0x20005120
func	void→void	needed	0x94A6
temp	int	provided	-
x	char	needed	-

States:

- Provided: Object defined in current file
- Needed: Object referenced but defined elsewhere

Memory Sections

Section	Content	Location
.text	Program code	ROM/Flash
.data	Initialized global/static variables	ROM→RAM
.bss	Uninitialized global/static variables	RAM
.rodata	Read-only data	ROM/Flash

I2C (Inter-Integrated Circuit)

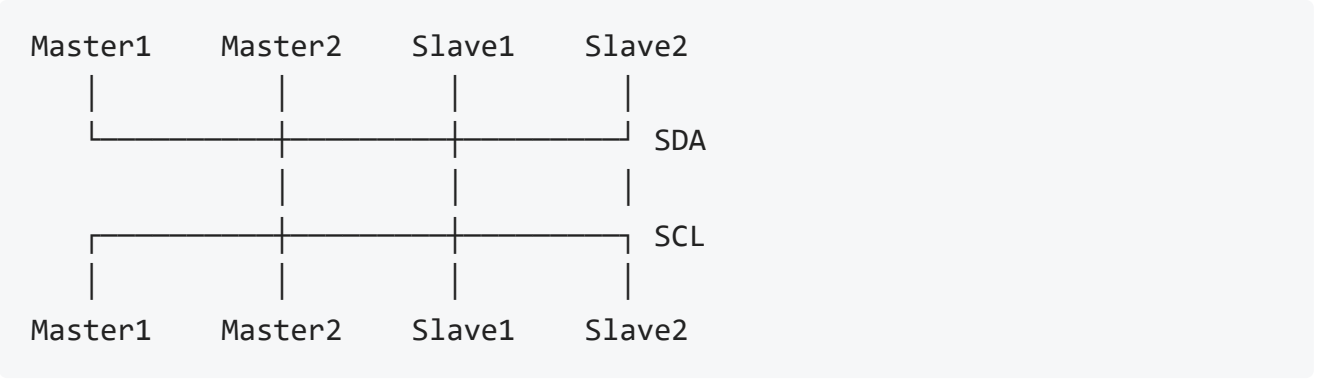
I2C Specifications

- Serial communication
- Multi Master, Multi Slave topology
- Synchronous (shared clock)
- Half Duplex communication
- 2-Wire Interface: SDA (data) + SCL (clock)

Hardware Connection

Two-Wire Bus:

- SDA: Serial Data Line
- SCL: Serial Clock Line



Line Types

Push-Pull Configuration

Node1	Node2	Line Result
0	0	GND
1	1	VCC
0	1	SHORT CIRCUIT
1	0	SHORT CIRCUIT

Open-Drain Configuration (I2C Standard)

Node1	Node2	Line Result
0	0	GND (dominant)
1	1	VCC (recessive)
0	1	GND (0 dominates)
1	0	GND (0 dominates)

I2C Line Characteristics

- Dominant Bit: 0 (appears if any node writes it)
- Recessive Bit: 1 (appears only if all nodes write it)
- Pull-up Resistors: Required for proper operation
- Open-Drain: Prevents short circuits in multi-master systems

Master vs Slave Roles

Master Responsibilities

- **Clock Control:** Generate SCL timing
- **Transfer Direction:** Determine read/write operations
- **Start/Stop Control:** Initiate and terminate transfers
- **Device Addressing:** Select target slaves

Slave Responsibilities

- **Address Recognition:** Respond when addressed
- **Clock Following:** Synchronize with master's timing
- **Data Response:** Provide requested data or acknowledge

I2C Data Transmission

Start and Stop Conditions

```
Start Condition: SDA goes LOW while SCL is HIGH
Stop Condition:  SDA goes HIGH while SCL is HIGH

Idle ──┐      ┌─── Data ──┐      ┌─── Idle
       │      │           │      │
SDA    └──────┘           └──────┘

          ┌──────┐           ┌──────┐
SCL    ───┘      └───────────┘      └───
     START              STOP
```

Address Frame (9 bits)

```
Bit:  7    6    5    4    3    2    1    0    ACK
     ┌────┬────┬────┬────┬────┬────┬────┬────┬────┐
     │         7-bit Address          │R/W │ACK │
     └────┴────┴────┴────┴────┴────┴────┴────┴────┘
```

Special Addresses:

- **0000000 (0x00):** General Call Address
- **0000001 (0x01):** NULL Address
- **1111111 (0x7F):** Extension Address

Data Transmission Patterns

Master Write to Slave

```
START → Address+W → ACK → Data → ACK → Data → ACK → STOP
  M        M        S      M      S      M      S     M
```

Master Read from Slave

```
START → Address+R → ACK → Data → ACK → Data → NACK → STOP
  M        M        S      S      M      S       M      M
```

Combined Format

```
START → Addr+W → ACK → Data → ACK → RESTART → Addr+R → ACK → Data → NACK
→ STOP
  M       M       S      M      S       M         M       S      S
M      M
```

Multi-Master Arbitration

When multiple masters attempt simultaneous transmission:

```
Master1: START → 0 → 1 → 0 → 1...
Master2: START → 0 → 0 → 1 → 0...
Bus:     START → 0 → 0 → ?
                     ↑
                Master1 loses arbitration
                (reads 0 when sending 1)
```

Arbitration Rules:

- Masters compare their transmitted data with bus state
- If master sends 1 but reads 0, it loses arbitration
- Losing master stops transmission and waits
- Process continues until one master wins

I2C Advantages & Disadvantages

Advantages ✅

- **Flexible Data Rate:** Can adjust speed during operation
- **Individual Addressing:** Each device independently addressable
- **Simple Master-Slave Relationship:** Clear communication hierarchy
- **Minimal Wiring:** Only two signals required

- **Multi-Master Capable:** Supports multiple controlling devices

Disadvantages ❌

- **Higher Power Consumption:** Pull-up resistors consume power
- **Lower Speed:** Slower than SPI due to protocol overhead
- **Pull-up Resistors Required:** ~10kΩ resistors needed on both lines
- **Distance Limitations:** Capacitance limits bus length
- **Complex Error Handling:** Arbitration and collision detection needed

I2C Implementation Example

```
// I2C Initialization
void I2C_Init(void) {
    // Set SCL frequency = CPU_freq / (16 + 2*TWBR*prescaler)
    TWBR = 72;  // For 100kHz at 16MHz CPU
    TWSR = 0x00; // Prescaler = 1
}

// I2C Start Condition
void I2C_Start(void) {
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
    while (!(TWCR & (1<<TWINT)));
}

// I2C Stop Condition
void I2C_Stop(void) {
    TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);
}

// I2C Write Data
void I2C_Write(uint8_t data) {
    TWDR = data;
    TWCR = (1<<TWINT) | (1<<TWEN);
    while (!(TWCR & (1<<TWINT)));
}

// I2C Read Data with ACK
uint8_t I2C_ReadAck(void) {
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    while (!(TWCR & (1<<TWINT)));
    return TWDR;
}
```

```
// I2C Read Data with NACK
uint8_t I2C_ReadNack(void) {
    TWCR = (1<<TWINT) | (1<<TWEN);
    while (!(TWCR & (1<<TWINT)));
    return TWDR;
}

// Complete I2C Write Transaction
void I2C_WriteToDevice(uint8_t device_addr, uint8_t data) {
    I2C_Start();
    I2C_Write(device_addr << 1);  // Address + Write bit
    I2C_Write(data);
    I2C_Stop();
}

// Complete I2C Read Transaction
uint8_t I2C_ReadFromDevice(uint8_t device_addr) {
    uint8_t data;
    I2C_Start();
    I2C_Write((device_addr << 1) | 1);  // Address + Read bit
    data = I2C_ReadNack();
    I2C_Stop();
    return data;
}
```

References

- I2C Bus Specification (NXP/Philips)
- SPI Protocol Documentation
- ATmega32 TWI (I2C) Documentation
- Multi-Master System Design Guidelines