EMBEDDED SYSTEMS

Mastering AVR I2C (TWI) Communication: A Deep Dive into Assembly and C Implementation for Embedded Systems

Published: February 18, 2025 | Index ID: #422

In the expansive landscape of embedded systems, the **Inter-Integrated Circuit (I2C)** protocol—often referred to as the **Two-Wire Interface (TWI)** in the context of AVR microcontrollers—stands as a cornerstone of serial communication. Developed originally by Philips Semiconductors (now NXP), I2C was designed to minimize the physical connections between a central processor and various peripheral chips. For engineers working with the Atmel/Microchip AVR architecture, such as the ubiquitous **ATmega328P**, **ATmega16**, or **ATmega32**, mastering the TWI peripheral is essential for interfacing with sensors, EEPROMs, LCDs, and real-time clocks.

The Theoretical Framework of I2C/TWI

The I2C protocol is a synchronous, multi-master, multi-slave, packet-switched, single-ended, serial communication bus. Its primary advantage lies in its simplicity: it requires only two wires—**SDA (Serial Data)** and **SCL (Serial Clock)**—to facilitate communication between dozens of devices. Unlike SPI (Serial Peripheral Interface), which requires a dedicated 'Chip Select' line for every slave device, I2C uses unique 7-bit or 10-bit addresses to identify targets on the bus.

Physical Layer and Electrical Characteristics

I2C operates on an **open-drain** (or open-collector) logic. This means that devices can only pull the lines low; they cannot drive them high. External **pull-up resistors**are required to bring the lines to a logic high state when the bus is idle. This wired-AND configuration is what allows multiple masters to exist on the same bus without causing short circuits during contention.

The value of these pull-up resistors is critical. If the resistance is too high, the rise time of the signals will be too slow, leading to data errors at high frequencies. If too low, the power consumption increases and the devices may struggle to pull the line low enough to register a logic zero. Typically, values between **2.2k:' æB** ³@are used, depending on the bus capacitance and the desired clock speed (Standard mode at 100 kHz or Fast mode at 400 kHz).

AVR TWI Hardware Architecture

Inside an AVR microcontroller, the TWI unit consists of several key functional blocks: the Bit Rate Generator, the Bus Interface Unit, and the Address Match Unit. To program these effectively, a developer must manipulate specific registers.

Key TWI Registers

- TWBR (TWI Bit Rate Register): This register controls the SCL frequency. In Master mode, the cycle frequency is determined by the value in TWBR and the prescaler bits in TWSR.
- TWCR (TWI Control Register): The heart of the TWI peripheral. It contains bits like TWINT (Interrupt Flag), TWEA (Enable Acknowledge), TWSTA (Start Condition), TWSTO (Stop Condition), and TWEN (Enable).
- TWSR (TWI Status Register): Provides the current status of the bus. The upper 5 bits reflect the internal state machine's status code (e.g., Start condition transmitted, Address acknowledged, etc.).
- TWDR (TWI Data Register): The buffer for data to be transmitted or data just received.

- TWAR (TWI Address Register): In Slave mode, this register holds the device's 7-bit address.

The Mathematical Model for Bit Rate

The SCL frequency (F_{SCL}) is calculated using the following formula provided in the AVR datasheets:

$F_{SCL} = \frac{CPU\ Clock}{16 + 2(TWBR) \cdot 4^{TWPS}}$

Where **TWPS**represents the prescaler value (0, 1, 2, or 3) bits in the TWSR. For a standard 16 MHz clock to achieve a 100 kHz I2C clock, the calculation would be:

$100,000 = \frac{16,000,000}{16 + 2(TWBR) \cdot 1} \Rightarrow TWBR = 72$

Technical Analysis: The TWI State Machine

Programming I2C on an AVR is essentially a process of managing a state machine. Every action (sending a start bit, sending an address) triggers an update in the **TWSR**status register. A robust driver must check these status codes to ensure the bus is behaving as expected.

Table 1: Common TWI Status Codes (Master Mode)

Status Code (Hex)	Meaning	Required Action
0x08	START condition transmitted	Load SLA+W/R into TWDR
0x10	Repeated START transmitted	Load SLA+W/R into TWDR
0x18	SLA+W transmitted; ACK received	Load Data into TWDR
0x20	SLA+W transmitted; NACK received	Send STOP or Repeated START
0x28	Data byte transmitted; ACK received	Send next Data or STOP
0x40	SLA+R transmitted; ACK received	Wait for Data reception

Implementation: Assembly vs. C Programming

When developing for AVR, engineers often choose between the efficiency of **Assembly (ASM)** and the readability of **Embedded C**. Assembly allows for cycle-accurate timing and the smallest possible binary footprint, which is vital for chips with limited flash like the ATmega8. C, using libraries like avr-libc, offers faster development cycles and easier maintenance.

I2C Master Initialization in C

```
void TWI_Init(void) {
  TWSR = 0x00; // Prescaler = 1
  TWBR = 0x48; // 72 for 100kHz at 16MHz clock
  TWCR = (1 &lt;&lt; TWEN); // Enable TWI
}I2C Master Start Sequence in Assembly
```

```
TWI_Start:
  ldi r16, (1&lt;&lt;TWINT)|(1&lt;&lt;TWSTA)|(1&lt;&lt;TWEN)
  out TWCR, r16
Wait_Start:
  in r16, TWCR
  sbrs r16, TWINT
  rjmp Wait_Start
  retPractical Case Study: Interfacing the 24C16 EEPROM
```

The 24C16 is a 16K-bit I2C serial EEPROM. It is organized as 128 pages of 16 bytes each. To write a byte to this device using an ATmega328P, the following workflow is executed:

1. Send START: Initiate the bus sequence.
2. Send Device Address: The 24C16 address usually starts with 1010. The last three bits of the control byte often select the memory block. For writing, the LSB is 0.
3. Send Word Address: Specify the internal memory location.
4. Send Data: Transfer the byte to be stored.
5. Send STOP: Release the bus and trigger the EEPROM's internal write cycle (which takes about 5-10ms).

Interfacing an I2C LCD (PCF8574)

Standard 16x2 character LCDs usually require 4-bit or 8-bit parallel data. An I2C backpack using the **PCF8574**I/O expander allows the AVR to control the LCD using only two wires. The complexity moves from hardware wiring to software logic, where the developer must toggle the LCD's 'Enable' bit by sending specific bitmasks over I2C to the PCF8574.

Troubleshooting and Performance Optimization

Even seasoned developers encounter issues with I2C. Below are common failure modes and their technical solutions.

1. The "Bus Hang" Scenario

If a Master is reset during a Read operation while the Slave is driving the SDA line low, the Master will see the bus as busy indefinitely. **Solution:**Implement a "Bus Clear" routine at startup. Toggle the SCL line manually 9 times to force the slave to release the SDA line, followed by a START and STOP condition.

2. Addressing Errors

A common mistake is confusing the 7-bit address with the 8-bit transmission byte. If a sensor's datasheet says the address is 0x68, you must shift it left by one bit (0xD0) before adding the Read/Write bit.

3. Noise and Signal Integrity

In industrial environments, electromagnetic interference (EMI) can cause glitches. **Solution:**Use shielded cables for long runs, keep SDA and SCL lines close to each other, and implement software-level checksums or CRCs for critical data transfers.

Table 2: Comparison of Communication Protocols in AVR

Feature	I2C / TWI	SPI	UART
Wires Required	2 (SDA, SCL)	4 (MISO, MOSI, SCK, SS)	2 (TX, RX)
Max Speed	Typically 400kHz	Up to CPU Clock/2	Typically 115.2kbps
Addressing	7-bit / 10-bit Software	Hardware (Slave Select)	Point-to-Point (usually)
Complexity	High (State Machine)	Low (Shift Register)	Medium (Baud Rate)

Advanced Topics: Multi-Master Arbitration

One of the most powerful features of I2C is multi-master support. If two AVRs attempt to start a transmission at the exact same time, the TWI hardware performs **arbitration**. It monitors the SDA line; if a master detects that the line is low when it intended to send a high bit, it has lost arbitration to another master. The losing master immediately stops transmitting and switches back to slave mode to see if it is being addressed. This process is seamless and handled entirely by the AVR hardware, provided the **TWEN** bit is set.

Synthesis and Broader Implications

The ability to implement I2C/TWI communication on AVR microcontrollers is a foundational skill for modern electronics engineering. Whether writing tight, optimized Assembly code for an ATtiny or high-level C for an ATmega, understanding the underlying state machine is the difference between a flaky prototype and a robust industrial product. As the Internet of Things (IoT) continues to grow, the demand for efficient, low-power sensor integration—where I2C excels—remains at an all-time high.

By mastering the registers (TWCR, TWSR, TWBR), understanding the electrical constraints of pull-up resistors, and being able to debug bus conditions using status codes, developers can unlock the full potential of the AVR ecosystem. The transition from parallel to serial communication represents a shift towards more compact, efficient, and scalable designs, solidifying I2C's place as a vital protocol for the foreseeable future.

Baca Juga (Artikel Terkait)

- **Technical Deep Dive into 1.8" TFT Display Breakouts and Shields: A Comprehensive Guide to ST7735 Integration**
  URL: http://alaskawriters.com/comprehensive-guide-1-8-tft-display-st7735-integration.pdf
- **Comprehensive Guide to Electronic Dice Design: From PICAXE Microcontrollers to Digital Randomization Logic**
  URL: http://alaskawriters.com/comprehensive-guide-electronic-dice-design-picaxe-logic.pdf
- **Mastering 1-Wire Protocol: Technical Deep Dive into sockit_owm and FPGA Implementation**
  URL: http://alaskawriters.com/mastering-1-wire-protocol-fpga-implementation-sockit-owm.pdf
- **Comprehensive Guide to Arduino TFT Displays: Engineering, Interfacing, and Real-Time Data Visualization**
  URL: http://alaskawriters.com/arduino-tft-display-comprehensive-guide.pdf

Tags: #AVR #I2C #TWI #Atmega328p #Assembly #Embedded C