Lab 10: I2C (Inter integrated circuit)

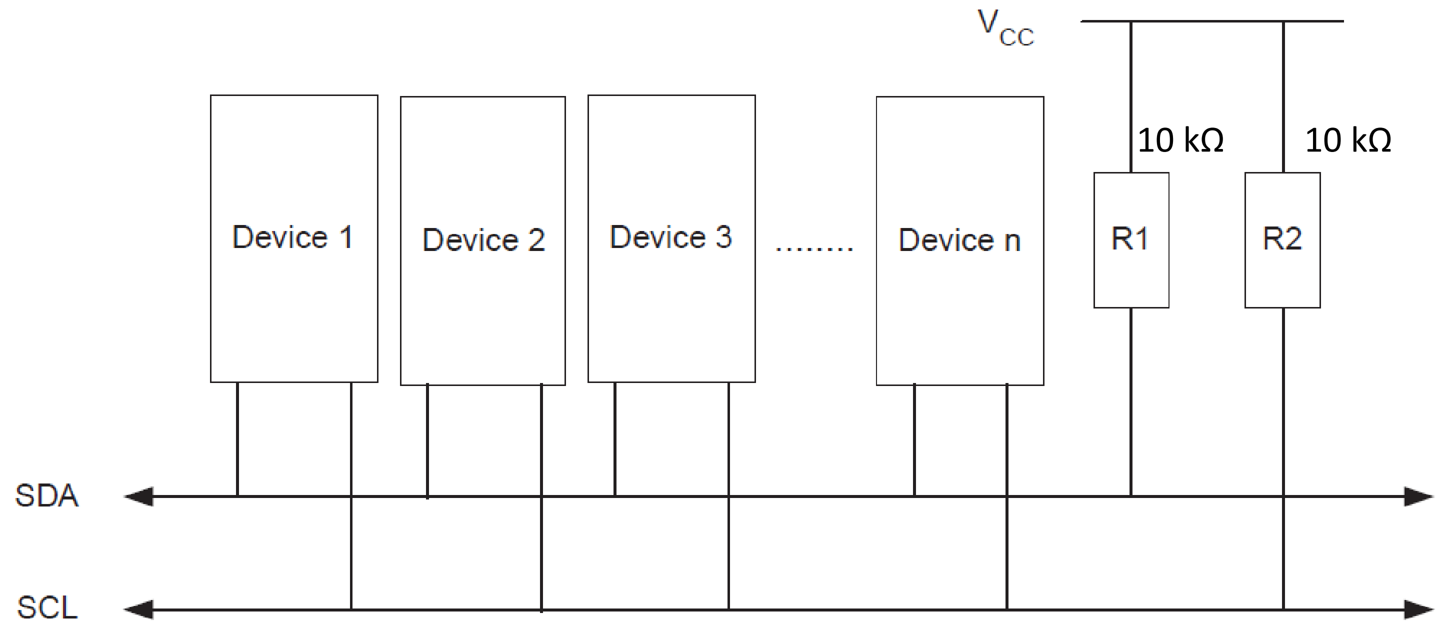
Instructor: Sung-Yeul Park
TA: S M Rakiul Islam

ECE 3411
Department of Electrical and Computer Engineering
University of Connecticut

February 21st, 2018

I^2C: Inter Integrated Circuit

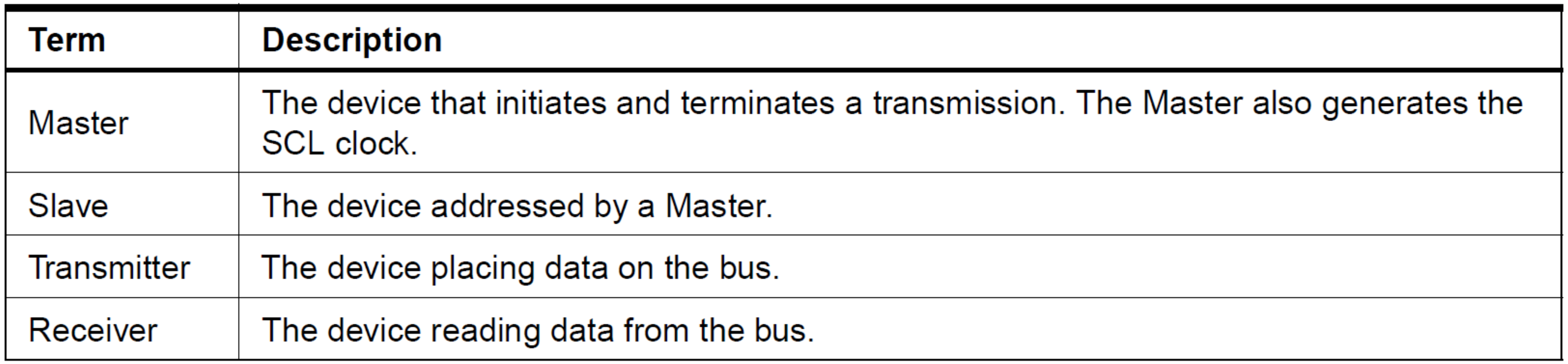
- Also known as Two Wire Interface (TWI)
- Allows up to 128 different devices to be connected using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA).
- A pull-up resistor (typically 10 kΩ) is needed for each of the TWI bus lines.
- All devices connected to the bus have individual addresses.



I^2C Terminologies

- I^2C (TWI) protocol allows several devices (up to 128) to be connected.
- Each device is identified by a configurable 7-bit address.
- Each device can communicate with any other device
  - The transmitter address the receiver by its 7-bit address.

Table 21-1. TWI Terminology

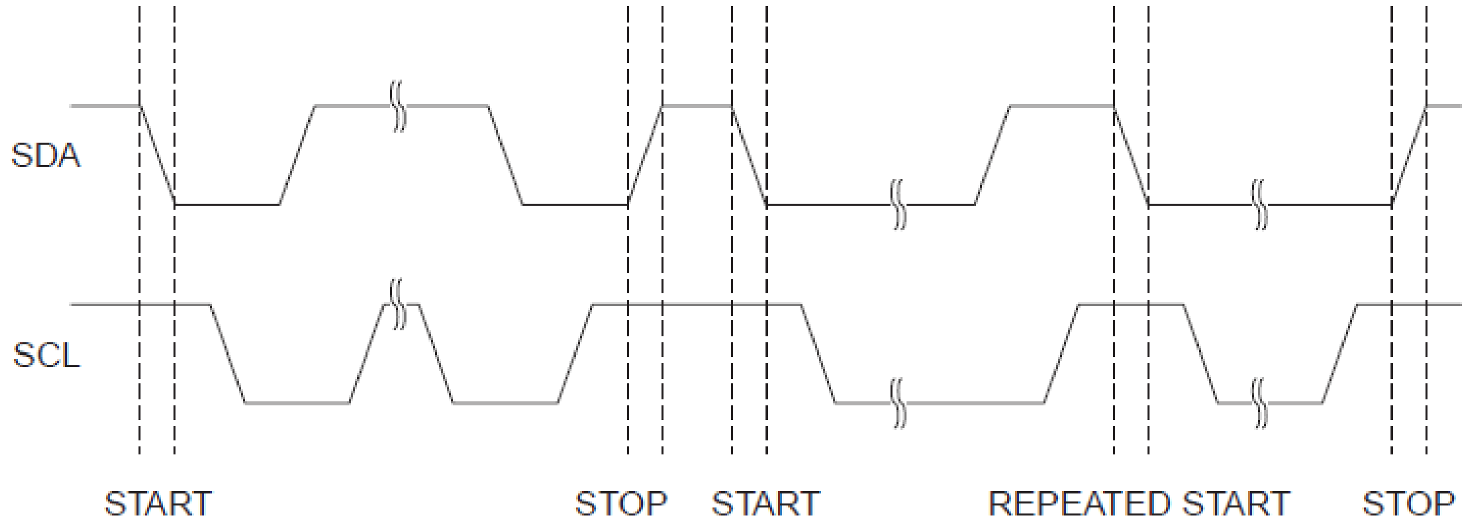


Term	Description
Master	The device that initiates and terminates a transmission. The Master also generates the SCL clock.
Slave	The device addressed by a Master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

I^{2}C START and STOP Conditions

- START and STOP conditions are signaled by changing the level of the SDA line when the SCL line is high.
- When a new START condition is issued between a START and STOP condition, this is referred to as a REPEATED START condition

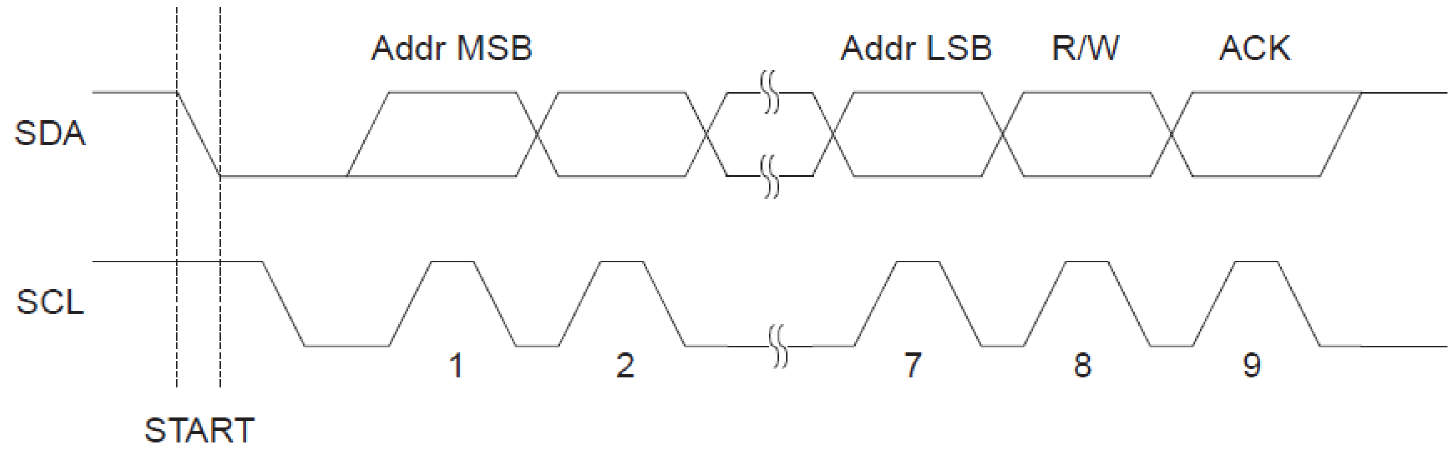
Figure 21-3. START, REPEATED START and STOP conditions



I^2C Address Packet Format

- All address packets transmitted on the TWI bus are 9 bits long:
  - 7 address bits, one READ/WRITE control bit and an acknowledge bit.
- When a Slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.
- The Master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission.

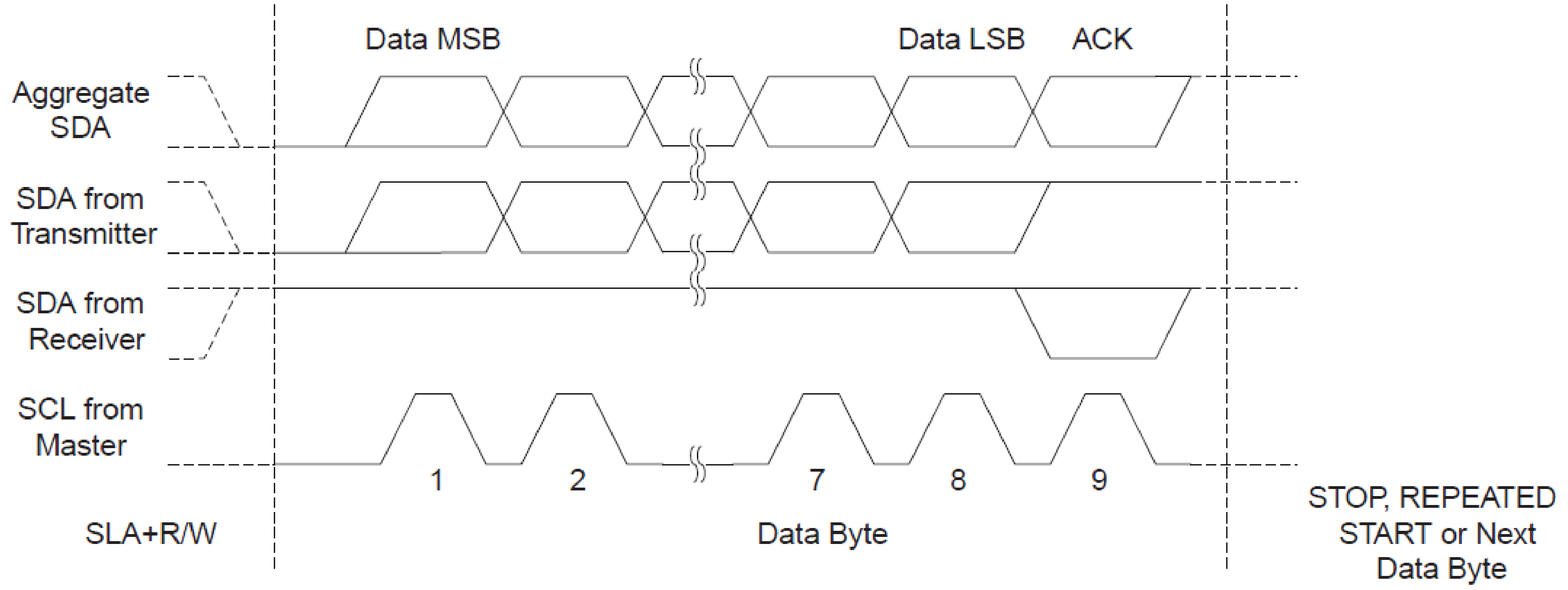
Figure 21-4. Address Packet Format



I^{2}C Data Packet Format

- All data packets transmitted on the TWI bus are 9 bits long:
  - One data byte and one acknowledge bit.
- An Acknowledge (ACK) is signaled by the Receiver pulling the SDA line low during the ninth SCL cycle. If the Receiver leaves the SDA line high, a NACK is signaled.

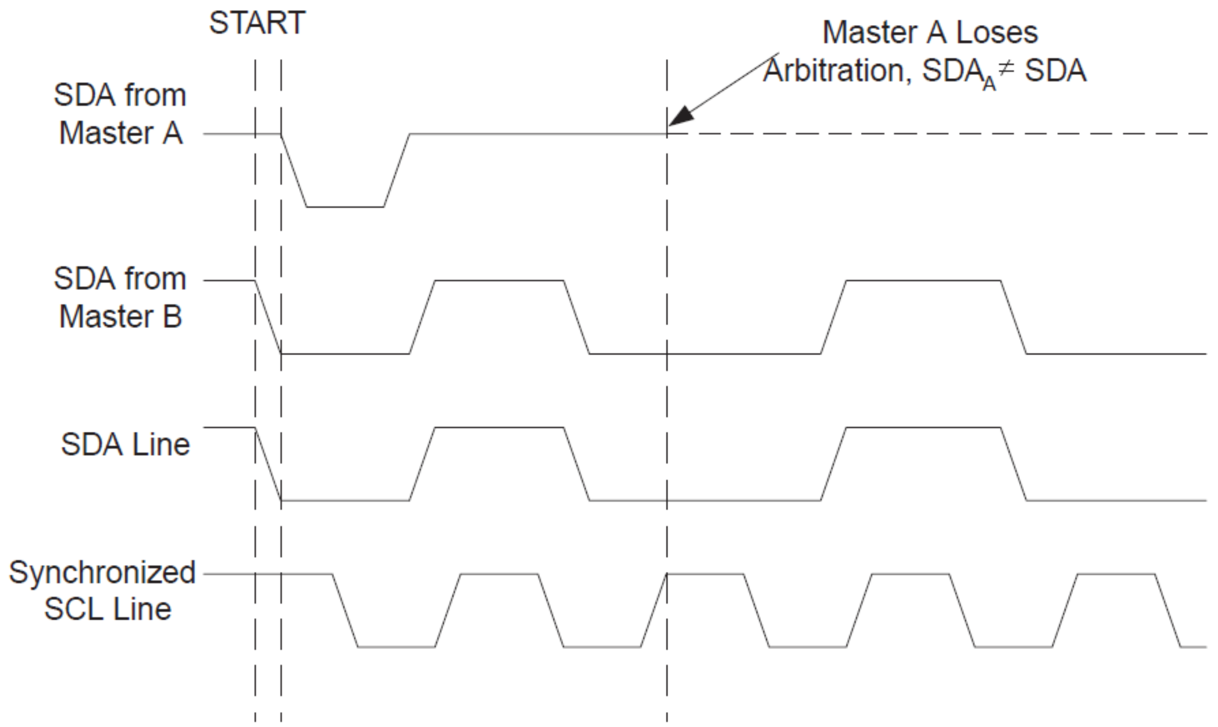
Figure 21-5. Data Packet Format



I^2C Bus Arbitration

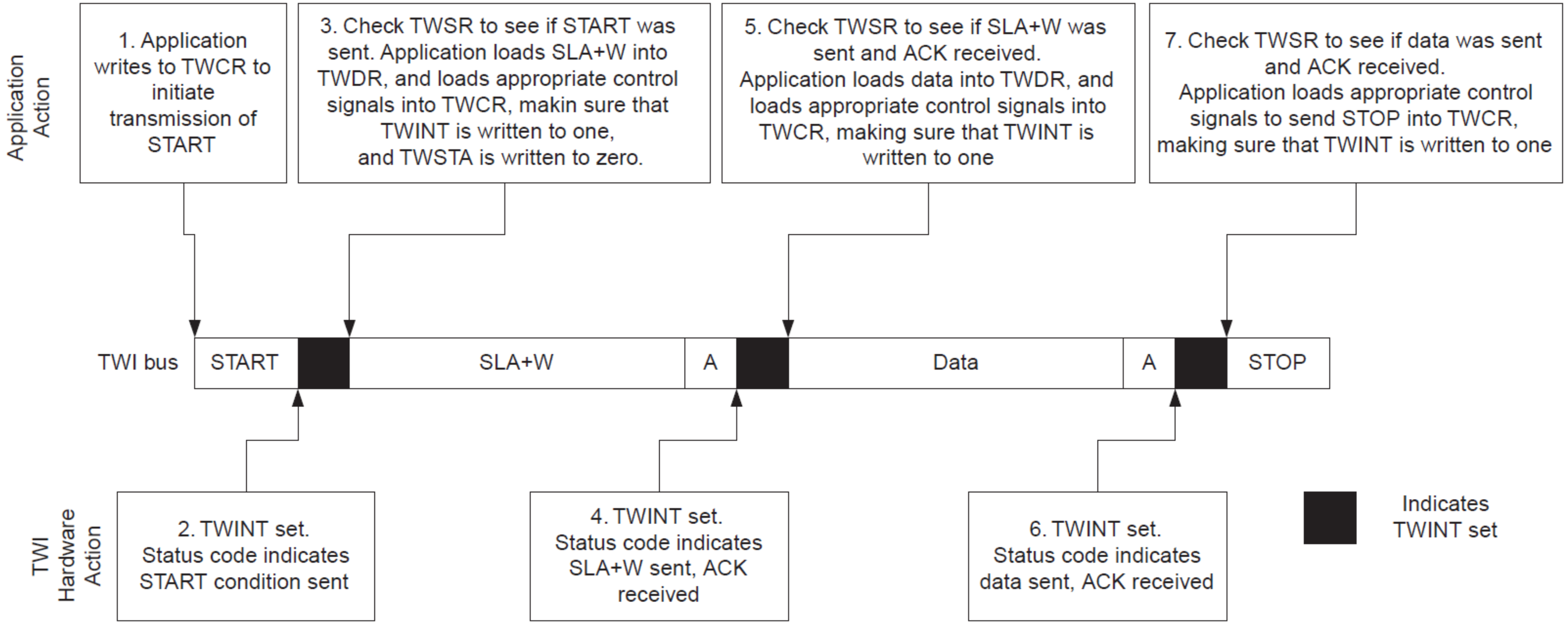
- Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data.
- If the value read from the SDA line does not match the value the Master had output, it has lost the arbitration.

Figure 21-8. Arbitration Between Two Masters



A typical I^2C Transmission

Figure 21-10. Interfacing the Application to the TWI in a Typical Transmission



Application Action
1. Application writes to TWCR to initiate transmission of START
3. Check TWSR to see if START was sent. Application loads SLA+W into TWDR, and loads appropriate control signals into TWCR, makin sure that TWINT is written to one, and TWSTA is written to zero.
5. Check TWSR to see if SLA+W was sent and ACK received. Application loads data into TWDR, and loads appropriate control signals into TWCR, making sure that TWINT is written to one
7. Check TWSR to see if data was sent and ACK received. Application loads appropriate control signals to send STOP into TWCR, making sure that TWINT is written to one
TWI bus
START
SLA+W
A
Data
A
STOP
TWI Hardware Action
2. TWINT set. Status code indicates START condition sent
4. TWINT set. Status code indicates SLA+W sent, ACK received
6. TWINT set. Status code indicates data sent, ACK received
Indicates TWINT set

A typical I^{2}C Transmission Summary

- When the TWI has finished an operation and expects application response, the TWINT Flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT Flag is set, the user must update all TWI Registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set.
- Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

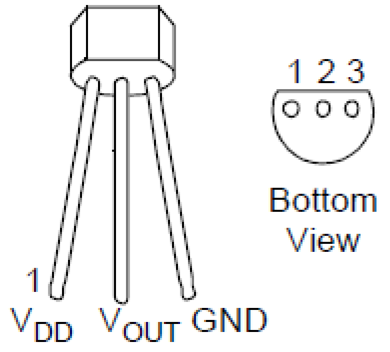
ADC based temperature sensor

$$V_{OUT} = T_C \times T_A + V_{0°C}$$

Where:

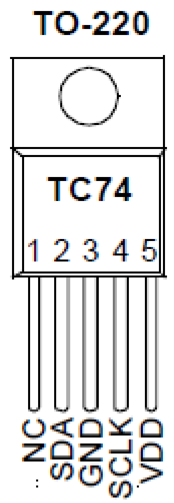
T_A	=	Ambient Temperature
V_{OUT}	=	Sensor Output Voltage
$V_{0°C}$	=	Sensor Output Voltage at 0°C (see **DC Electrical Characteristics** table)
T_C	=	Temperature Coefficient (see **DC Electrical Characteristics** table)

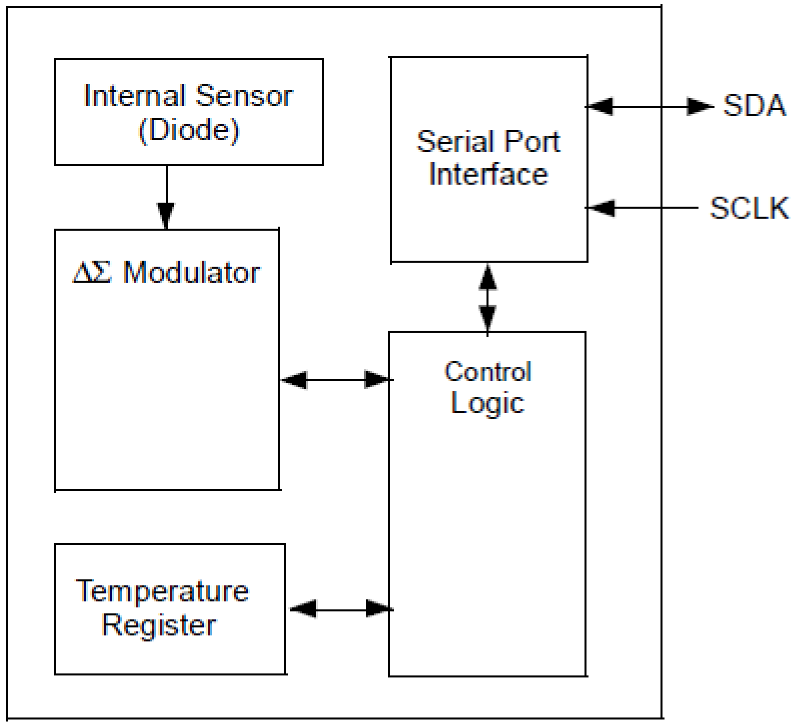
3-Pin TO-92
MCP9700/9700A
MCP9701/9701A

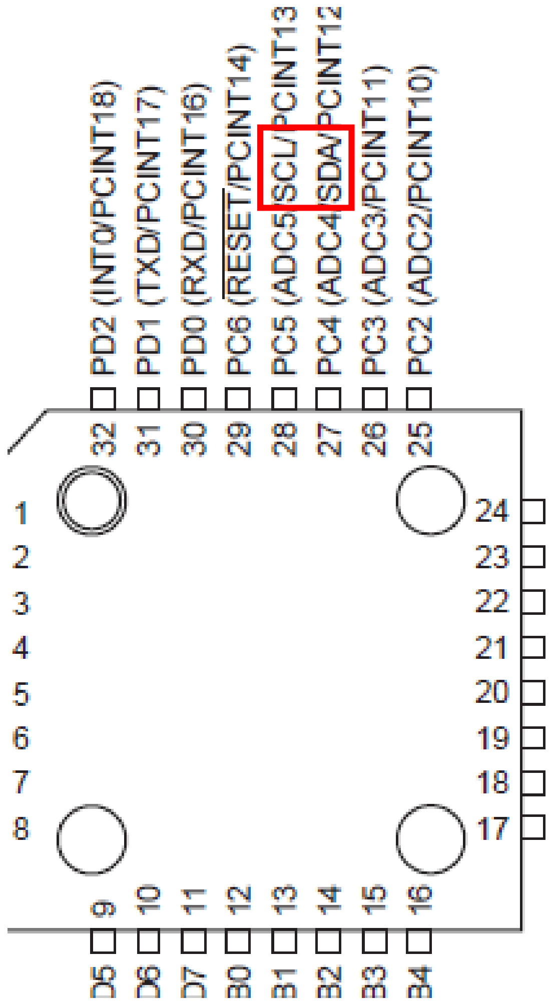


Sensor Output						
Output Voltage, T_A = 0°C	$V_{0°C}$	—	500	—	mV	**MCP9700/9700A**
Output Voltage, T_A = 0°C	$V_{0°C}$	—	400	—	mV	**MCP9701/9701A**
Temperature Coefficient	T_C	—	10.0	—	mV/°C	**MCP9700/9700A**
	T_C	—	19.5	—	mV/°C	**MCP9701/9701A**
Output Nonlinearity	V_{ONL}	—	±0.5	—	°C	T_A = 0°C to +70°C (**Note 3**)

I2C based temperature sensor







Connections

- Connect the board as in Fig.1.

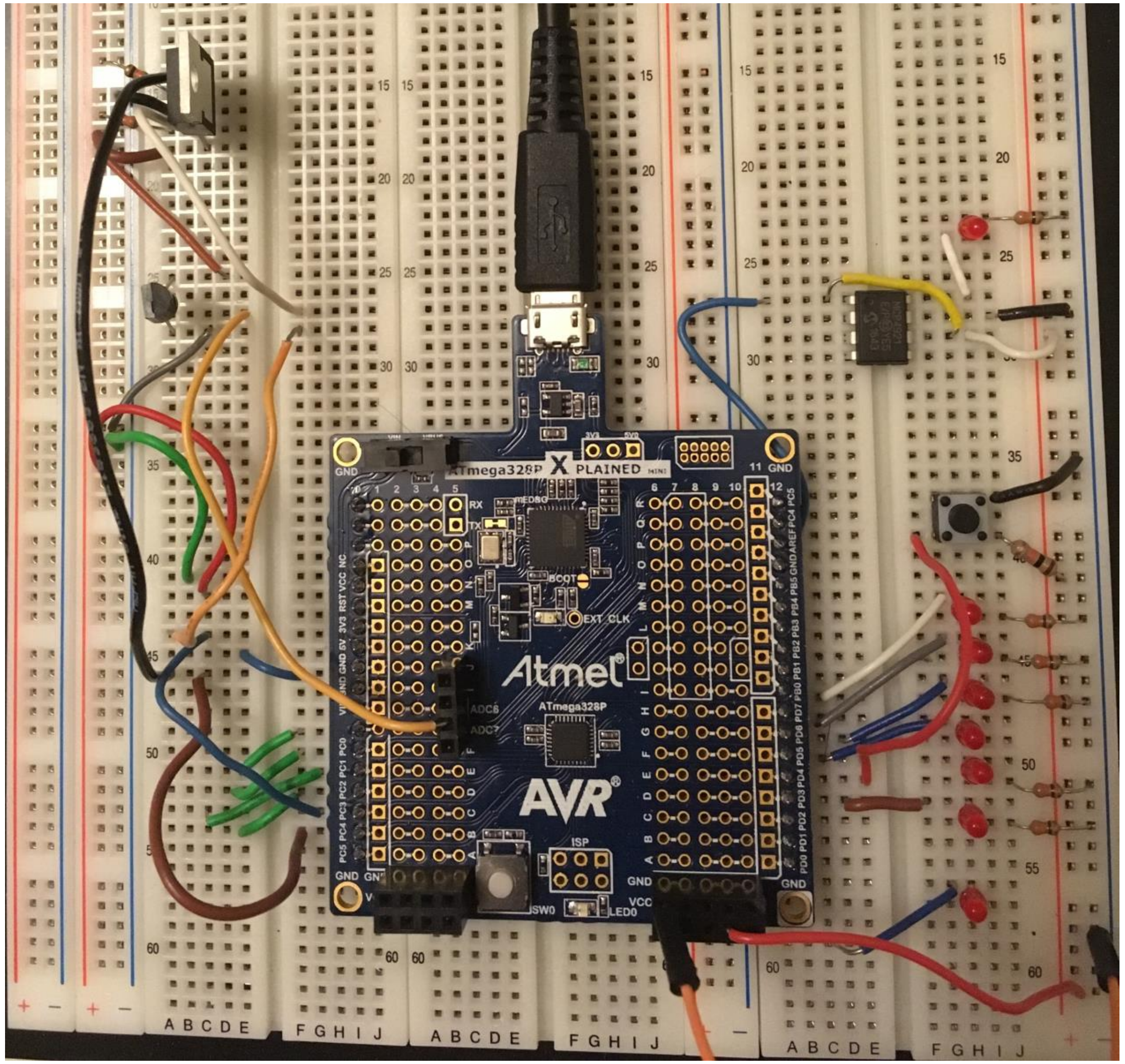


Fig1. Connections Temperature sensing using I2C and ADC

I^{2}C Transmission Example

```
uint8_t TWI_Master_Transmit(uint8_t Address, uint8_t Data)
{
            TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);    // Send START condition
            while (!(TWCR & (1<<TWINT)));                         // Wait for TWINT Flag set.
            if ((TWSR & 0xF8) != START)                           // Check value of TWI Status Register.
                        ERROR();
            TWDR = (Address << 1) | (WRITE);                      // Load SLA_W (Slave Address & Write) into TWDR Register.
            TWCR = (1<<TWINT) | (1<<TWEN);             // Clear TWINT bit in TWCR to start transmission of address.
            while (!(TWCR & (1<<TWINT)));                         // Wait for TWINT Flag set.
            if ((TWSR & 0xF8) != MT_SLA_ACK)                      // Check value of TWI Status Register.
                        ERROR();
            TWDR = Data;                                          // Load DATA into TWDR Register.
            TWCR = (1<<TWINT) | (1<<TWEN);             // Clear TWINT bit in TWCR to start transmission of data.
            while (!(TWCR & (1<<TWINT)));                         // Wait for TWINT Flag set.
            if ((TWSR & 0xF8) != MT_DATA_ACK)                     // Check value of TWI Status Register.
                        ERROR();
            TWCR = (1<<TWINT)|(1<<TWEN)| (1<<TWSTO);   // Transmit STOP condition.
}
```

Note: The code above assumes that several definitions have been made, for example by using include-files.

I^{2}C Reception Example

```
void TWI_Slave_Initialize(uint8_t Address)
{
		TWAR = (Address << 1)|(1);				// Load Slave Address into TWAR Register.
		TWCR = (1<<TWEA)|(1<<TWEN);				// Enable TWI & Acknowledgements.
}
```

```
uint8_t TWI_Slave_Receive(void)
{
		TWCR = (1<<TWEA)|(1<<TWEN);					// Enable TWI & Acknowledgements.
		while (!(TWCR & (1<<TWINT)));					// Wait for TWINT Flag set (once this slave is addressed)
		if ((TWSR & 0xF8) != 0x60)					// Check value of TWI Status Register.
			ERROR();
		TWCR = (1<<TWINT) | (1<<TWEN);				// Clear TWINT bit start reception of data.
		while (!(TWCR & (1<<TWINT)));					// Wait for TWINT Flag set.
		if ((TWSR & 0xF8) != 0x80)					// Check if Data has been received & ACK has been returned
			ERROR();
		TWCR = (1<<TWINT) | (1<<TWEN);				// Clear TWINT bit.
		return TWDR;							// Read TWDR Register.
}
```

Note: The code above assumes that several definitions have been made, for example by using include-files.