I2C Bus in AVR

(Chapter 18 of the Mazidi’s book)
Contents

• Serial communication with I2C (inter-IC) bus
• I2C in AVR and programming in C
Serial ports in AVR

- **USART**: an old and widely used standard (pins 14 and 15 of ATmega32)
- **SPI**: very high-speed (~ two orders of magnitude faster than USART) synchronous serial port
  - For example for high-speed data transfer to EPROM
  - Pins 5 to 8 of Atmega32
- **I2C (Inter IC)**
  - Connection using a shared bus connecting up to 128 devices
  - Pins 22 and 23 of ATmega32
I2C

• IIC (Inter-Integrated Circuit)
• A bus interface connection incorporated into many devices such as sensors, RTC, and EEPROM
• Also known as I2C or \( I^2C \)
• Originally started by Philips
  – Widely used standard adapted by many semiconductor companies
I2C

• Ideal for attaching low-speed peripherals to a motherboard or embedded system
  – 100 to 400 Kb/s
  – At most 3 meter distance
  – Up to 120 devices
• Provides a connection-oriented communication with acknowledge
• Uses only 2 pins for data transfer instead of the 8 or more pins used in traditional parallel buses
  – Low pin: low packaging and wiring cost
I2C

• 2 pins for data transfer
  – SCL (Serial Clock): which synchronize the data transfer between two chips
  – SDA (Serial Data)
• In many application notes, including AVR datasheets, 12C is referred to as *Two-Wire Serial Interface (TWI)*
• Wired-and logic

![Diagram of I2C connections](image)
I2C

• In the AVR up to 120 different devices can share an I2C bus
  – Each of these devices is called a *node*
• Each node can operate as either master or slave
  – Master is a device that generates the clock for the system
  – Slave is the node that receives the clock and is addressed by the master
  – In I2C, both master and slave can receive or transmit
I2C timing

- Each data bit transferred on the SDA line is synchronized by clock on the SCL line
- The data line cannot change when the clock line is high
  - Can change only when the clock line is low
  - The STOP and START conditions are the only exceptions to this rule
I2C timing

• Start and stop condition
• I2C is a connection-oriented communication protocol
  – Each transmission is initiated by a START condition and is terminated by a STOP condition.
  – Generated by the master
• START: high-to-low change in the SDA line when SCL is high
• STOP: low-to-high change in the SDA line when SCL is high
I2C timing

- Use repeated start in case of the need for reading/writing multiple data without any intervening operations
Data format

• Each packet is 9 bits long
  – The first 8 bits are put on the SDA line by the transmitter
  – The 9th bit is an ACK (acknowledge) or NACK by the receiver

• Transmitter releases the SDA line during the ninth clock

• Receiver pulls the SDA line low for ACK or keep it high for NACK
Address format

- Each packet may contain either data or address
- An address packet consists of 9 bits
  - 7 address bits (the MSB is transmitted first)
  - 1 READ/WRITE control bit (1=read / 0=write)
  - 1 acknowledge bit
Address format

• Write to a slave with address 1001101
Addressing

• Addresses are 7 bit
  – Potentially 128 addressable devices, but only 119 addresses can be used
• 1111xxx is reserved
• 0000000 is used for broadcast
  – All slaves should ack
Data format

- 9 bits: The first 8 bits are a byte of data to be transmitted, and the 9th bit is ACK
- MSB is transmitted first
- If the receiver has received the last byte of data and there is no more data to be received, or the receiver cannot receive or process more data, it will signal a NACK
Data format

- A transmission is started by a START, followed by an address packet, one or more data packets, and finished by a STOP
Data write

• A master writes the value \(1111,0000\) to a slave with address \(1001,101\)
Multi-byte memory write

- Burst mode writing, as you already know, is an effective means of loading consecutive locations

1. Generate a START condition
2. Transmit the slave address followed by zero (for write)
3. Transmit the address of the first location
4. Transmit the data for the first location and from then on, simply provide consecutive bytes of data to be placed in consecutive memory locations
5. Generate a STOP condition
Multi-byte memory read

1. Generate a START
2. Transmit the slave address followed by zero (for address write)
3. Transmit the address of the first location
4. Generate a START (REPEATED START) condition
5. Transmit the slave address followed by one (for read)
6. Read the data from the first location and from then on, bring contents out from consecutive memory locations
7. Generate a STOP condition
Clock stretching

• kind of flow control
• If an addressed slave device is not ready to process more data it will stretch the clock
  – The master will not be able to raise the clock line
Arbitration

- Each transmitter has to check the level of the bus and compare it with the level it expects; if it doesn't match, that transmitter has lost the arbitration.
- Master A wants to put 0010000 on the bus.
- Master B wants to put 0001111 on the bus.
- Master A lost in the third cycle.

![Arbitration Diagram]

---

University of Tehran 21
I2C in PCs

• System Management Bus (SMBus) is a subset of the I2C bus/protocol and was developed by Intel
• Generally not user configurable or accessible
• Like I2C with some minor modifications
  – The main modification is adding another signal called SMBALERT
  – Used by slave to notify the master to start communication (e.g. when a sensor prepares a data)
    » Like an interrupt
I2C in PCs

• ICH-8 has two SMbus interfaces: one is always master and the other is always slave
  – The Slave Interface allows an external master to read from or write to the ICH8
  – The master interface allows ICH8 to read from or write to external devices

• Used for communication with low-bandwidth devices on a motherboard
  – Battery, temperature sensors, lid switches in laptops,…
I2C in ICH-8
I2C in ICH-8

- SMBus pins in ICH-8
- Two pins for I2C operation plus one pin for SMBAlert
I2C in industry

• DS1307 RTC (real-time clock)
• RTCs keep the time in a digital system
• are present in almost any electronic device that needs to keep accurate time
  – PCs, servers, mobile phones, …

• RTC needs a battery to work always, even when the computer is turned off
  • Some RTCs have internal battery
  • Some other (e.g. PCs) use an external lithium battery (called CMOS)

Picture source: pctechnotes.com
I2C example- DS1307

• DS1307 is a serial RTC with I2C bus interface

<table>
<thead>
<tr>
<th>PIN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1,X2</td>
<td>Connections to Quartz Crystal</td>
</tr>
<tr>
<td>Vcc</td>
<td>Primary Power Supply (when computer is on)</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>Vbat</td>
<td>Battery power supply (when computer is off)</td>
</tr>
<tr>
<td>SQW</td>
<td>Wave output</td>
</tr>
<tr>
<td>SDA and SCL</td>
<td>I2C connections</td>
</tr>
</tbody>
</table>
### DS1307 registers

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>Bit7</th>
<th>Bit6</th>
<th>Bit5</th>
<th>Bit4</th>
<th>Bit3</th>
<th>Bit2</th>
<th>Bit1</th>
<th>Bit0</th>
<th>FUNCTION</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>CH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Seconds</td>
<td>00–59</td>
</tr>
<tr>
<td>01H</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Minutes</td>
<td>00–59</td>
</tr>
<tr>
<td>02H</td>
<td>0</td>
<td>12</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hours</td>
<td>00–23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24</td>
<td>PM/AM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+AM/PM</td>
<td>00–23</td>
</tr>
<tr>
<td>03H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>Day</td>
<td>01–07</td>
</tr>
<tr>
<td>04H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>Date</td>
<td>01–31</td>
</tr>
<tr>
<td>05H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Month</td>
<td>01–12</td>
</tr>
<tr>
<td>06H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Year</td>
<td>00–99</td>
</tr>
<tr>
<td>07H</td>
<td>OUT</td>
<td>0</td>
<td>0</td>
<td>SQWE</td>
<td>0</td>
<td>0</td>
<td>RS1</td>
<td>RS0</td>
<td>Control</td>
<td>—</td>
</tr>
<tr>
<td>08H-3FH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RAM 56 x 8</td>
<td>00H–FFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 64 byte registers
- 8 bytes are used for time, 56 bytes (from 08-3F) as a general purpose RAM
DS1307 registers

• The first 6 registers keep time
• Register 7 (at 0x07) is control register:

<table>
<thead>
<tr>
<th>BIT 7</th>
<th>BIT 6</th>
<th>BIT 5</th>
<th>BIT 4</th>
<th>BIT 3</th>
<th>BIT 2</th>
<th>BIT 1</th>
<th>BIT 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>0</td>
<td>0</td>
<td>SQWE</td>
<td>0</td>
<td>0</td>
<td>RS1</td>
<td>RS0</td>
</tr>
</tbody>
</table>

• Bit 4 Square-Wave Enable (SQWE): when set to logic 1, enables the oscillator output
  – If some logic need a clock
• Bits 1, 0: Rate Select (RS1, RS0). These bits control the frequency of the square-wave output

<table>
<thead>
<tr>
<th>RS1</th>
<th>RS0</th>
<th>SQUARE-WAVE OUTPUT FREQUENCY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1Hz</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4.096kHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8.192kHz</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>32.768kHz</td>
</tr>
</tbody>
</table>
### DS1307 registers

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>Bit7</th>
<th>Bit6</th>
<th>Bit5</th>
<th>Bit4</th>
<th>Bit3</th>
<th>Bit2</th>
<th>Bit1</th>
<th>Bit0</th>
<th>FUNCTION</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>CH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Seconds</td>
<td>00–59</td>
</tr>
<tr>
<td>01H</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Minutes</td>
<td>00–59</td>
</tr>
<tr>
<td>02H</td>
<td>0</td>
<td></td>
<td>12</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10 Hour</td>
<td>1–12, +AM/PM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>24</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PM/AM</td>
<td>00–23</td>
</tr>
<tr>
<td>03H</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>DAY</td>
<td>01–07</td>
</tr>
<tr>
<td>04H</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>Date</td>
<td>01–31</td>
</tr>
<tr>
<td>05H</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>10</td>
<td>Month</td>
<td>01–12</td>
</tr>
<tr>
<td>06H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>Year</td>
<td>00–99</td>
</tr>
</tbody>
</table>

- Time and date are kept in BCD format
- Bit 6 of register 2 determines hour mode: 0=12-hour mode, 1=24-hour mode
- In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM
- In the 24-hour mode, bit 5 is the second 10-hour bit
DS1307 I2C

- DS1307 has a Pointer Register that points to the byte that will be accessed in the next read/write
  - Used by I2C to transfer data
  - Should be set first before data exchange
DS1307 I2C- writing data to RTC

To set the value of the register pointer and write one or more bytes of data to DS1307, you can use the following steps:

1. To access the DS1307 for a write operation, after sending a START condition, you should transmit the address of DS1307 (1001101) followed by 0 to indicate a write operation.

2. The first byte of data in the write operation will set the register pointer. For example, if you want to access the control register you should send 0x07.

3. If you want only to set the register pointer you should skip this step. If you want to write one or more bytes of data, you should transmit them one byte at a time. Remember that the register pointer is automatically incremented and you can simply transmit bytes of data to consecutive locations in a multibyte burst write.

4. Transmit a STOP bit condition.
Data write

Diagram showing the process of data write with labels and explanations:

- S — START
- A — ACKNOWLEDGE
- P — STOP
- R/W — READ/WRITE OR DIRECTION BIT ADDRESS = D0H

Diagram details:
- Slave Address
- Word Address (n)
- Data (n)
- Data (n+1)
- Data (n+X)
- Data Transferred (X+1 bytes + ACKNOWLEDGE)
1. To access the DS1307 for a read operation, after sending a START condition, you should transmit the address of DS1307 (1001101) followed by 1 to indicate a read operation.

2. Now you can read one or more bytes of data. Remember that the register pointer indicates which address will be read. Also notice that the register pointer is automatically incremented and you can simply receive consecutive bytes of data in a multibyte burst read.

3. Transmit a STOP bit condition.
Data read

S — START
A — ACKNOWLEDGE
P — STOP
Â — NOT ACKNOWLEDGE
R/W — READ/WRITE OR DIRECTION BIT ADDRESS = D1H

DATA TRANSFERRED
(X+1 BYTES + ACKNOWLEDGE); NOTE: LAST DATA BYTE IS FOLLOWED BY A NOT ACKNOWLEDGE (Â) SIGNAL
RTC in PCs

• Older PCs used to have an RTC chip and CMOS battery on motherboard
• Now, RTC is integrated into Southbridge chip
  – CMOS still on motherboard
RTC in ICH-8

- Input power pin from CMOS battery is not shown
I2C in AVR

- In AVR this module is called TWI
- The TWI module in the AVR is composed of four sub-modules
  - Bit rate generation unit
  - Bus interface unit
  - Address match unit
  - Control unit
I2C in AVR

• In the AVR microcontroller, five major registers are associated with the TWI
  – TWBR (TWI Bit rate Register)
  – TWCR (TWI Control Register)
  – TWSR (TWI Status Register)
  – TWAR (TWI Address Register)
  – TWDR (TWI Data Register)
TWI Bit Rate Register (TWBR)

\[
SCL \text{ frequency } = \frac{\text{CPU Clock frequency}}{16 + 2 \times (\text{TWBR}) \times 4^{\text{TWPS}}}
\]

Calculate the SCL frequency if the value of TWPS bits in TWSR is 01 (1 Dec) and the value of TWBR is 00100110 (38 Dec). Assume that CPU clock frequency is 8 MHz.

**Solution:**

The SCL frequency will be: 

\[
8 \text{ MHz} / ((16 + 2 \times (38)) \times 4) = 25 \text{ kHz}
\]
TWI Status Register (TWSR)

<table>
<thead>
<tr>
<th>TWS7</th>
<th>TWS6</th>
<th>TWS5</th>
<th>TWS4</th>
<th>TWS3</th>
<th>–</th>
<th>TWPS1</th>
<th>TWPS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>–</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TWPS1</th>
<th>TWPS0</th>
<th>Prescaler Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>64</td>
</tr>
</tbody>
</table>
**TWI Status Register (TWSR)**

<table>
<thead>
<tr>
<th>TWS7</th>
<th>TWS6</th>
<th>TWS5</th>
<th>TWS4</th>
<th>TWS3</th>
<th>TWPS1</th>
<th>TWPS0</th>
<th>TWSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>TWDR</th>
<th>TWCR</th>
<th>TWINT</th>
<th>TWEA</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x88</td>
<td>Slave Address Request (SLA+W) not satisfied, Address Not Acknowledged</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0x90</td>
<td>Slave Alternative Address Request (SLA+A) not satisfied, Address Not Acknowledged</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0x98</td>
<td>Master Stop or Repeated Start</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

*Note: These codes and actions are associated with specific states and operations in TWI communication.*
TWI Status Register (TWSR)

<table>
<thead>
<tr>
<th>TWS7</th>
<th>TWS6</th>
<th>TWS5</th>
<th>TWS4</th>
<th>TWS3</th>
<th>TWS2</th>
<th>TWS1</th>
<th>TWS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>بیشترین تعداد (TWSR) وضعیت و پورت</th>
<th>پاسخ نمایش کاربری</th>
<th>مقادیر ثابت TWSR</th>
<th>عملیات بعدی که به وسیله پورت سری TWI انجام می‌شود</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x60</td>
<td></td>
<td>STA</td>
<td>پایت داده دریافت خواهده شد ولی NOT ACK پاسخ داده خواهده شد.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TWD</td>
<td>پایت داده دریافت خواهده شد ولی ACK پاسخ داده خواهده شد.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TWDR</td>
<td>پایت داده دریافت خواهده شد ولی NOTACK پاسخ داده خواهده شد.</td>
</tr>
<tr>
<td>0x70</td>
<td></td>
<td></td>
<td>پایت داده دریافت خواهده شد ولی ACK پاسخ داده خواهده شد.</td>
</tr>
<tr>
<td>0x80</td>
<td></td>
<td></td>
<td>پایت داده دریافت خواهده شد ولی NOTACK پاسخ داده خواهده شد.</td>
</tr>
</tbody>
</table>

University of Tehran 43
TWI Control Register (TWCR)

<table>
<thead>
<tr>
<th>Bit 7 – TWINT: TWI Interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>This bit is set by hardware when the TWI module has finished its current job. If the TWI and general interrupt are enabled, changing TWINT to one will cause the MCU to jump to the TWI interrupt vector. Clearing this flag starts the operation of the TWI. TWINT must be cleared by software.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 6 – TWEA: TWI Enable Acknowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Making this bit HIGH will enable the generation of ACK when needed in slave or receiver mode.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 5 – TWSTA: TWI START condition Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Making this bit HIGH will generate a START condition if the bus is free; otherwise, the TWI module waits for the bus to become free and then generates aSTART condition</td>
</tr>
</tbody>
</table>
TWI Control Register (TWCR)

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWINT</td>
<td>TWEA</td>
<td>TWSTA</td>
<td>TWSTO</td>
<td>TWWC</td>
<td>TWEN</td>
<td>–</td>
<td>TWIE</td>
</tr>
<tr>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Bit 4 – TWSTO: TWI STOP condition bit**
In master mode, making this bit HIGH causes the TWI to generate a STOP condition. This bit is cleared by hardware when the STOP condition is transmitted.

**Bit 3 – TWWC: TWI Write Collision Flag**
This bit is set HIGH when we attempt to access the TWI Data Register when TWINT is low. This flag is cleared by writing to the TWDR register when TWINT is high.

**Bit 2 – TWEN: TWI Enable**
Making this bit HIGH enables the TWI module.

**Bit 0 – TWIE: TWI Interrupt Enable**
Making this bit HIGH enables the TWI interrupt if the general interrupt is enabled.
TWI Address Register (TWAR)

• Contains the 7-bit slave address
  – When working as slave

• The eighth bit (LSB) of TWAR is TWGCE (TWI General Call Recognition Enable)
  – If this bit is set to one, receiving of a general call address will cause an interrupt request
TWI Data Register (TWDR)

• In Receive mode, the last received byte will be in the TWDR
• In Transmit mode, you should write the next byte into TWDR to be transmitted
Programming of the AVR TWI in master operating mode

• We must be able to
  – Initialize the TWI
  – Transmit a START condition
  – Send or receive data
  – Transmit a STOP condition
Initialize the TWI in master operating mode

1. Set the TWI module clock frequency by setting the values of the TWBR register and the TWPS bits in the TWSR register
2. Enable the TWI module by setting the TWEN bit in the TWCR register to one
Transmit START condition

- Set the TWSTA bit to one: tells the TWI to initiate a START condition when the bus is free
- Set the TWINT bit to one: clears the interrupt flag to initiate operation of the TWI module to transmit the START condition
- Poll the TWINT flag in the TWCR register to see whether the START condition transmitted completely
Send data

• Copy the data byte to the TWDR
• Set the TWEN and TWINT bits of the TWCR register to one to start sending the byte
• Poll the TWINT flag in the TWCR register to see whether the byte transmitted completely
Receive data

• Set the TWEN and TWINT bits of the TWCR register to one to start receiving a byte
• If you want to return ACK after receiving data you should also set the TWEA bit of the TWCR register to one
• Poll the TWINT flag in the TWCR register to see whether a byte has been received completely
• Copy the received byte from the TWDR to another register to save it
Transmit STOP condition

- Set the TWEN, TWSTO, and TWINT bits of the TWCR register to one
I2C programming

• A program to write 1111,0000 to a slave with address 1101,000

```c
#include <avr/io.h>

void i2c_write(unsigned char data) {
    TWDR = data;
    TWCR = (1<<TWINT)|(1<<TWEN);
    while (((TWCR & (1<<TWINT)) == 0));
}

void i2c_start(void) {
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
    while (((TWCR & (1<<TWINT)) == 0));
}

```

//*********************************************************/
I2C programming

- A program to write 1111,0000 to a slave with address 1101,000

```c
void i2c_stop()
{
    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
}

/*-----------------------------*/

void i2c_init(void)
{
    TWSR=0x00; //set prescaler bits to zero
    TWBR=0x47; //SCL frequency is 50K for XTAL = 8M
    TWCR=0x04; //enable the TWI module
}

int main (void)
{
    i2c_init();
    i2c_start(); //transmit START condition
    i2c_write(0b11010000); //transmit SLA + W(0)
    i2c_write(0b11110000); //transmit data
    i2c_stop(); //transmit STOP condition
    while(1); //stay here forever
    return 0;
}
```
**DS1307 registers**

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>Bit7</th>
<th>Bit6</th>
<th>Bit5</th>
<th>Bit4</th>
<th>Bit3</th>
<th>Bit2</th>
<th>Bit1</th>
<th>Bit0</th>
<th>FUNCTION</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>CH</td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Seconds</td>
<td>00–5</td>
</tr>
<tr>
<td>01H</td>
<td>0</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Minutes</td>
<td>00–5</td>
</tr>
<tr>
<td>02H</td>
<td>0</td>
<td></td>
<td>12</td>
<td></td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td>Hours</td>
<td>1–12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hours</td>
<td>1–12</td>
</tr>
<tr>
<td>03H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>Day</td>
<td>01–0</td>
</tr>
<tr>
<td>04H</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Date</td>
<td>01–3</td>
</tr>
<tr>
<td>05H</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Month</td>
<td>01–1</td>
</tr>
<tr>
<td>06H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>Year</td>
<td>00–9</td>
</tr>
<tr>
<td>07H</td>
<td>OUT</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>Control</td>
<td>—</td>
</tr>
<tr>
<td>08H–3FH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RAM 56 x 8</td>
<td>00H–F</td>
</tr>
</tbody>
</table>

**AVR TWCR Register**

![TWCR Register Diagram](image)