By Ashutosh Bhatt

This article explores the TWI interfacing between two **ATmega32** controllers. Readers are advised to go through **TWI Communication** and **TWI registers of ATmega32** before going further.

TWI works in four modes:

1. MASTER as a transmitter.
2. MASTER as a receiver.
3. SLAVE as a receiver.
4. SLAVE as a transmitter.

Generally modes 1 & 3 and modes 2 & 4 are used together. This article explains the use of these four modes by an experiment.

Objective: To establish the communication between two ATmega32 using TWI interface. First the Master starts by sending data then the slave transmits complement of the received data to the master. When the Master receives the complemented data, it shifts the original data to left. This process of transmitting and receiving continues. As the data value reaches 0x80 the whole process is repeated. At the starting, value of the original data is 0x01. The received value is displayed on PORTB at both the ends.

Circuit description:

Make the connections as shown in the circuit diagram.

Code explanation for MASTER Controller:

Step 1: Initialization of master.

Fig. 2: Equation Of TWI Clock Frequency to initialize the Master in AVR

```
void TWI_init_master(void) // Function to initialize master
{
    TWBR=0x01;    // Bit rate
    TWSR=(0<<TWPS1)|(0<<TWPS0);    // Setting prescalar bits
    // SCL freq= F_CPU/(16+2(TWBR).4^TWPS)
}
```

Step 2: Send start condition

The start condition in TWI is explained before. The **AVR microcontroller** has in built registers which makes this job much easier.

1. Clear TWINT flag by writing a logic one to it.
2. Set TWSTA bit to send start condition.
3. Set TWEN bit to initialize the TWI.
4. Monitor the status of TWINT flag.
5. Check the ACK byte (using while condition since the SCL frequency is very small as compared to micro controller clock frequency). The ACK byte can be compared by monitoring the status of TWSR.

```
void TWI_start(void)
{
    // Clear TWI interrupt flag, Put start condition on SDA, Enable TWI
    TWCR= (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
    while(!(TWCR & (1<<TWINT))); // Wait till start condition is transmitted
    while((TWSR & 0xF8)!= 0x08); // Check for the acknowledgement
}
```

Step 3: Send the slave address, data direction bit (write) and wait for the ACK signal

START condition	7 bit slave address	Data direction bit	Slave ACK

```
void TWI_read_address(unsigned char data)
{
    TWDR=data;    // Address and read instruction
    TWCR=(1<<TWINT)|(1<<TWEN);    // Clear TWI interrupt flag,Enable TWI
    while (!(TWCR & (1<<TWINT))); // Wait till complete TWDR byte received
    while((TWSR & 0xF8)!= 0x40);  // Check for the acknoledgement
}
```

Step 4: Send the 8-bit data and wait for the ACK

START condition	7 bit slave address	Data direction bit	Slave ACK	8 bit data transmission	Receiver ACK

Fig. 4: Data transfer in TWI of AVR

1. Put the 8 bit data in TWDR.

8 bits = 7 bit slave address + Data direction bit (write = 0).

2. Clear TWINT flag.
3. Set TWEN bit to enable TWI.
4. Monitor the status of TWINT flag to get data transmission completed.
5. Check for the acknowledgement.

```
void TWI_write_data(unsigned char data)
{
    TWDR=data;    // put data in TWDR
    TWCR=(1<<TWINT)|(1<<TWEN);    // Clear TWI interrupt flag,Enable TWI
    while (!(TWCR & (1<<TWINT))); // Wait till complete TWDR byte transmitted
    while((TWSR & 0xF8) != 0x28); // Check for the acknoledgement
}
```

Step 5: Send the STOP condition

START condition	7 bit slave address	Data direction bit	Slave ACK	8 bit data transmission	Receiver ACK	STOP condition

Fig.5: TWI STOP Condition Bit

To send the stop condition use TWSTO

```
    while(!(TWCR & (1<<TWSTO)));  // Wait till stop condition is transmitted
}
```

Up till here the data transmission from slave side is complete, the MASTER is working in mode one. As per the objective the data received by MASTER is displayed on PORTB. The flow chart for MASTER as a Transmitter (mode one) is given below.

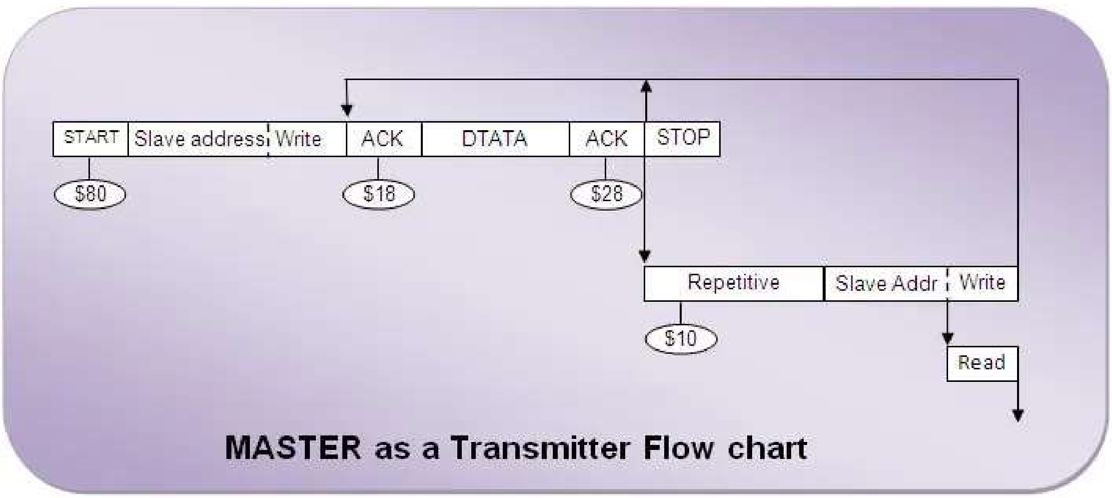


Fig. 6: Flow Chart of MASTER as Transmitter in TWI Interfacing using AVR

From here the MASTER would be working in mode two i.e., MASTER becomes a receiver. The AVR TWI works in mode 2.

Step 6: Send the START condition on bus lines

This step is as similar to the previous one.

Note: In the Step 6 the START condition is sent after the STOP condition. If one more start condition is sent before the STOP condition in between then it is called as repetitive start condition. The repetitive start condition is same as the START condition but the only difference is between the acknowledgements. For more details about repetitive start refer to the data sheet. If the data is sent continuously in same direction then there would be no need of start condition, repetitive

3. Set TWEN bit to enable TWI.
4. Monitor the status of TWINT flag to get data transmission completed.
5. Check acknowledgement.

```
void TWI_read_address(unsigned char data)
{
    TWDR=data;    // Address and read instruction
    TWCR=(1<<TWINT)|(1<<TWEN);    // Clear TWI interrupt flag,Enable TWI
    while (!(TWCR & (1<<TWINT))); // Wait till complete TWDR byte received
    while((TWSR & 0xF8)!= 0x40);  // Check for the acknoledgement
}
```

Step 8: Read the data from SDA bus

1. Clear TWINT flag
2. Set TWEN bit, enable TWI
3. Monitor the status of TWINT flag, as the TIWNT flag get set indicates the value in TWDR has been received.
4. Check for the acknowledgement. If the master wants to receive the last byte from slave, the status of TWSR register will be 0x58. After receiving the last byte either a repetitive start condition is issued by the master to continue the communication or a stop condition must be given by the master to stop the communication process. Else if the master wants to keep on receiving more byte from slave the status of TWSR register will be 0x50.

To acknowledge salve about last byte TWEA bit is used while transmitting the data. If TWEA bit is set, reception continuous from the MASTER side. And if TWEA bit is low, MASTER orders slave to send the last byte.

5. Get the received data. And send it on PORTB.

```
void TWI_read_data(void)
{
    TWCR=(1<<TWINT)|(1<<TWEN);    // Clear TWI interrupt flag,Enable TWI
    while (!(TWCR & (1<<TWINT))); // Wait till complete TWDR byte transmitted
    while((TWSR & 0xF8) != 0x58); // Check for the acknoledgement
    recv_data=TWDR;
    PORTB=recv_data;
}
```

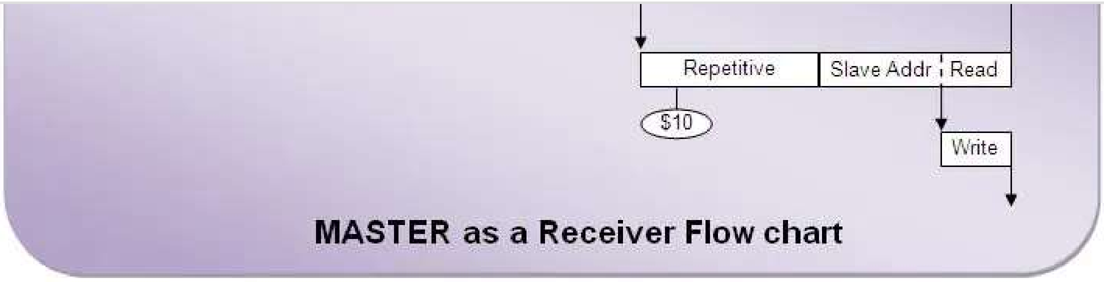


Fig. 7: Flow Chart of MASTER as Receiver in TWI Interfacing using AVR

Code explanation for SLAVE controller:

Step 1: Initialization of the Slave controller

Initialization of the slave controller is done by assigning address to the slave. The seven bit slave address is filled in TWI slave Address Register (TWAR). The LSB of TWAR i.e., TWGCE bit is use to enable the slave to acknowledge for the general call address (0x00).

```
void TWI_init_slave(void) // Function to initilaize slave
{
    TWAR=0x20;    // Fill slave address to TWAR
}
```

Step 2: Check the status of TWSR register

If the value of TWSR is 0x60, it means the data sent by the master in the next step is meant to read by this particular slave only and the slave sends back the acknowledgement to the master corresponding to the read operation. If the TWSR status is 0x70 the SLAVE is requested to read the data at the general call (0x00). At this stage the SLAVE acts as receiver. The AVR TWI is working in mode 3.

1. Clear TWIN flag.
2. Enable TWI.
3. Set TEWA to receive acknowledgement.
4. Monitor the status of TWINT flag.

```
        while (!(TWCR & (1<<TWINT)));  // Wait for TWINT flag
    }
}
```

Step 3: Read data

Read the data sent by the MASTER.

1. Clear TWINT flag.
2. Enable TWI.
3. Set TWEA for receiving ACK.
4. Get the data form TWDR, display it on PORTB.

```
void TWI_read_slave(void)
{
  // Clear TWI interrupt flag,Get acknowlegement, Enable TWI
  TWCR= (1<<TWINT)|(1<<TWEA)|(1<<TWEN);
  while (!(TWCR & (1<<TWINT)));   // Wait for TWINT flag
  while((TWSR & 0xF8)!=0x80);     // Wait for acknowledgement
  recv_data=TWDR;                 // Get value from TWDR
  PORTB=recv_data;                // send the receive value on PORTB
}
```

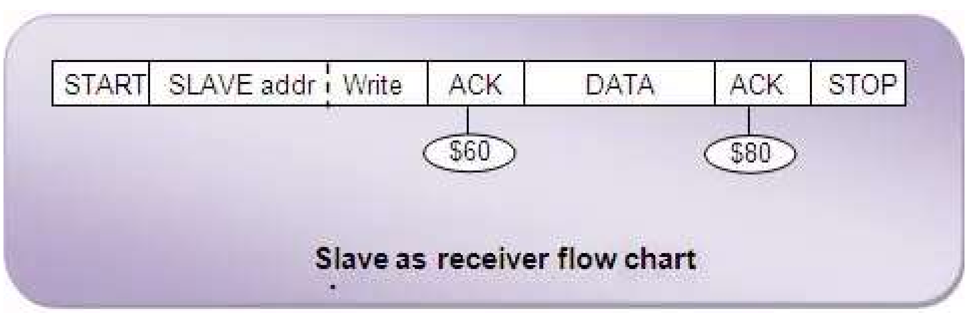


Fig. 8: Slave as Receiver in TWI Interfacing using AVR

3. Set TEWA to receive acknowledgement.
4. Monitor the status of TWINT flag.
5. Match the status of TWSR. If the status is 0xA8 send data or else jump to (1)

```
void TWI_match_write_slave(void) //Function to match the slave address and slave dirction bit
(write)
{
    while((TWSR & 0xF8)!= 0xA8)    // Loop till correct acknoledgement have been received
    {
        // Get acknowlegement, Enable TWI, Clear TWI interrupt flag
        TWCR=(1<<TWEA)|(1<<TWEN)|(1<<TWINT);
        while (!(TWCR & (1<<TWINT)));  // Wait for TWINT flag
    }
}
```

Step 5: Write the data on SDA bus

1. Put the data in TWDR.
2. Enable TWI.
3. Cleat the TWINT flag.
4. Monitor the TWINT flag. As it get cleared signifies the data has been send.
5. Check for the ACK. Since the TWEA bit was not set during writing data on SDA bus, it signifies master that this is the last data to be send and in turn it receives a NOT ACK and the status of TWSR is becomes 0xC0. And if the TWEA bit was set during the data transmission it receives an ACK and the status of TWSR is becomes 0xB8. For more details refer to the data sheet.

```
void TWI_write_slave(void) // Function to write data
{
   TWDR= write_data;            // Fill TWDR register whith the data to be sent
   TWCR= (1<<TWEN)|(1<<TWINT);   // Enable TWI, Clear TWI interrupt flag
```

Fig. 9: Slave as Transmitter in TWI Interfacing using AVR

Project Source Code

```
###




// Program for Master Mode
// Check Code2 for Slave Mode Program
#include<avr/io.h>
#include<util/delay.h>
#include<inttypes.h>

void TWI_start(void);
void TWI_repeated_start(void);
void TWI_init_master(void);
void TWI_write_address(unsigned char);
void TWI_read_address(unsigned char);
void TWI_write_data(unsigned char);
void TWI_read_data(void);
void TWI_stop(void);

unsigned char address=0x20, read=1, write=0;
unsigned char write_data=0x01, recv_data;

int main(void)
{
_delay_ms(2000);
DDRB=0xff;
TWI_init_master();  // Function to initialize TWI
while(1)
```

```
_delay_ms(10); // Delay of 10 mili second

TWI_start();
TWI_read_address(address+read); // Function to write address and data direction bit(read) on SDA
TWI_read_data(); // Function to read data from slave
TWI_stop();

_delay_ms(1000);

write_data = write_data * 2;
}


}

void TWI_init_master(void) // Function to initialize master
{
TWBR=0x01; // Bit rate
TWSR=(0<<TWPS1)|(0<<TWPS0); // Setting prescalar bits
// SCL freq= F_CPU/(16+2(TWBR).4^TWPS)
}

void TWI_start(void)
{
// Clear TWI interrupt flag, Put start condition on SDA, Enable TWI
TWCR= (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
while(!(TWCR & (1<<TWINT))); // Wait till start condition is transmitted
while((TWSR & 0xF8)!= 0x08); // Check for the acknowledgement
}

void TWI_repeated_start(void)
{
// Clear TWI interrupt flag, Put start condition on SDA, Enable TWI
TWCR= (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
while(!(TWCR & (1<<TWINT))); // wait till restart condition is transmitted
while((TWSR & 0xF8)!= 0x10); // Check for the acknoledgement
}

void TWI_write_address(unsigned char data)
{
TWDR=data; // Address and write instruction
TWCR=(1<<TWINT)|(1<<TWEN);    // Clear TWI interrupt flag,Enable TWI
while (!(TWCR & (1<<TWINT))); // Wait till complete TWDR byte transmitted
```

```
void TWI_write_data(unsigned char data)
{
TWDR=data; // put data in TWDR
TWCR=(1<<TWINT)|(1<<TWEN);    // Clear TWI interrupt flag,Enable TWI
while (!(TWCR & (1<<TWINT))); // Wait till complete TWDR byte transmitted
while((TWSR & 0xF8) != 0x28); // Check for the acknoledgement
}

void TWI_read_data(void)
{
TWCR=(1<<TWINT)|(1<<TWEN);    // Clear TWI interrupt flag,Enable TWI
while (!(TWCR & (1<<TWINT))); // Wait till complete TWDR byte transmitted
while((TWSR & 0xF8) != 0x58); // Check for the acknoledgement
recv_data=TWDR;
PORTB=recv_data;
}

void TWI_stop(void)
{
// Clear TWI interrupt flag, Put stop condition on SDA, Enable TWI
TWCR= (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
while(!(TWCR & (1<<TWSTO)));  // Wait till stop condition is transmitted
}
###
```

Project Source Code

```
###
```

```
// Program for Slave mode
#include<avr/io.h>
#include<util/delay.h>
```

```
DDRB=0xff;
TWI_init_slave(); // Function to initilaize slave
while(1)
{
TWI_match_read_slave(); //Function to match the slave address and slave dirction bit(read)
TWI_read_slave(); // Function to read data

write_data=~recv_data; // Togglem the receive data

TWI_match_write_slave(); //Function to match the slave address and slave dirction bit(write)
TWI_write_slave();       // Function to write data
}
}

void TWI_init_slave(void) // Function to initilaize slave
{
TWAR=0x20; // Fill slave address to TWAR
}

void TWI_write_slave(void) // Function to write data
{
TWDR= write_data;           // Fill TWDR register whith the data to be sent
TWCR= (1<<TWEN)|(1<<TWINT);   // Enable TWI, Clear TWI interrupt flag
while((TWSR & 0xF8) != 0xC0); // Wait for the acknowledgement
}

void TWI_match_write_slave(void) //Function to match the slave address and slave dirction bit(wr
ite)
{
while((TWSR & 0xF8)!= 0xA8) // Loop till correct acknoledgement have been received
{
// Get acknowlegement, Enable TWI, Clear TWI interrupt flag
TWCR=(1<<TWEA)|(1<<TWEN)|(1<<TWINT);
while (!(TWCR & (1<<TWINT)));  // Wait for TWINT flag
}
}

void TWI_read_slave(void)
{
// Clear TWI interrupt flag,Get acknowlegement, Enable TWI
TWCR= (1<<TWINT)|(1<<TWEA)|(1<<TWEN);
while (!(TWCR & (1<<TWINT))); // Wait for TWINT flag
while((TWSR & 0xF8)!=0x80); // Wait for acknowledgement
```

```
while (!(TWCR & (1<<TWINT)));  // Wait for TWINT flag
}
}

###
```

Circuit Diagrams

Circuit-Diagram-of-How-to-use-I2C-TWI-Two-Wire-Interface-in-AVR-ATmega32

Project Components

- **ATmega32**
- **LED**

Project Video

AVR ATmega16 TWI (Two Wire Interface) Tutorial

Filed Under: AVR Microcontroller
Tagged With: atmega32**,** avr**,** i2c**,** microcontroller**,** twi

Questions related to this article?
👉**Ask and discuss on EDAboard.com and Electro-Tech-Online.com forums.**

Tell Us What You Think!!

You must be logged in to post a comment.

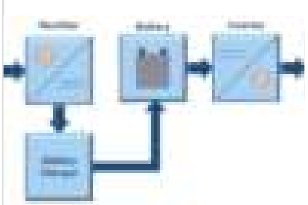
Search this website GO

engineering forums EDABoard.com and Electro-Tech-Online.com where you can get those questions asked and answered by your peers!

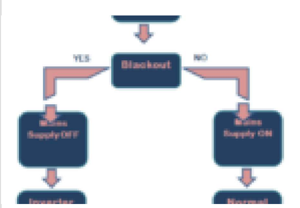
EDA BOARD

ELECTRO-TECH-ONLINE

FEATURED TUTORIALS

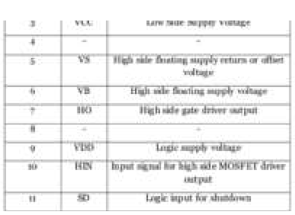


Designing an Online UPS – (Part 11/17)



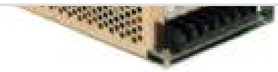
Designing an Offline UPS – Part (12 /17)

High and Low Side Switching of MOSFET – (Part 13/17)



3	VCC	Low side supply voltage
4	-	-
5	VS	High side floating supply return or offset voltage
6	VB	High side floating supply voltage
7	HO	High side gate driver output
8	-	-
9	VDD	Logic supply voltage
10	HIN	Input signal for high side MOSFET driver output
11	SD	Logic input for shutdown

Testing IR2110 Gate Driver IC- (Part 14/17)



STAY UP TO DATE

Sign up and receive our weekly newsletter for latest Tech articles, Electronics Projects, Tutorial series and other insightful tech content.

EE TRAINING CENTER

EE CLASSROOMS

EE DESIGN GUIDES

RECENT ARTICLES

- How to get input from USB mouse on Arduino

EDABOARD.COM DISCUSSIONS

- Cadence layout 'ghost' cells
- question about switch capacitor
- STM32 timer PWM
- Give me some tips for Highspeed interface IP beginner Layout eng'r
- vibration degrades phase noise of PLL

ELECTRO-TECH-ONLINE.COM DISCUSSIONS

- A question about static electricity
- 1W RGB LED drive from Wifi controller
- wich oscilloscope is better.
- High frequency 25-45kHz sound generator circuit, where to buy ready made?
- Funny Images Thread!

SUBMIT A GUEST POST