

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:

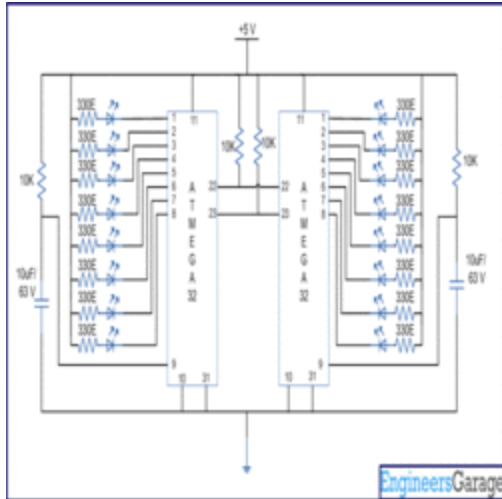
1. MASTER as a transmitter.
2. MASTER as a receiver.
3. SLAVE as a receiver.
4. SLAVE as a transmitter.

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



#### Circuit description:

Make the connections as shown in the circuit diagram.

#### Code explanation for MASTER Controller:

##### Step 1: Initialization of master.

Initialization of MASTER means to set the TWI clock frequency (SCL). It is done by setting bit rate in TWBR and pre scaler bits in TWSR.

$$\text{SCL freq} = \frac{\text{CPU clock freq.}}{16+2(\text{TWBR}).(4^{\text{TWPS}})}$$

```
void TWI_init_master(void) // Function to initialize master
{
    TWBR=0x01;    // Bit rate
    TWSR=(0<<TWPS1)|(0<<TWPS0);    // Setting prescaler 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
-----------------	---------------------	--------------------	-----------

These three processes are controlled by [AVR's TWI registers](#).

1. Put the seven bit slave address and the direction control bit in TWDR.
2. Clear TWINT flag.
3. Enable TWI by writing logic one to TWEN bit.
4. Monitor the status of TWINT flag, the TWINT flag will get cleared when the data in TWDR is been transmitted.
5. Check for the correct 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 acknowledgement
}
```

## 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
-----------------	---------------------	--------------------	-----------	-------------------------	--------------

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 acknowledgement
}
```

#### Step 5: Send the STOP condition

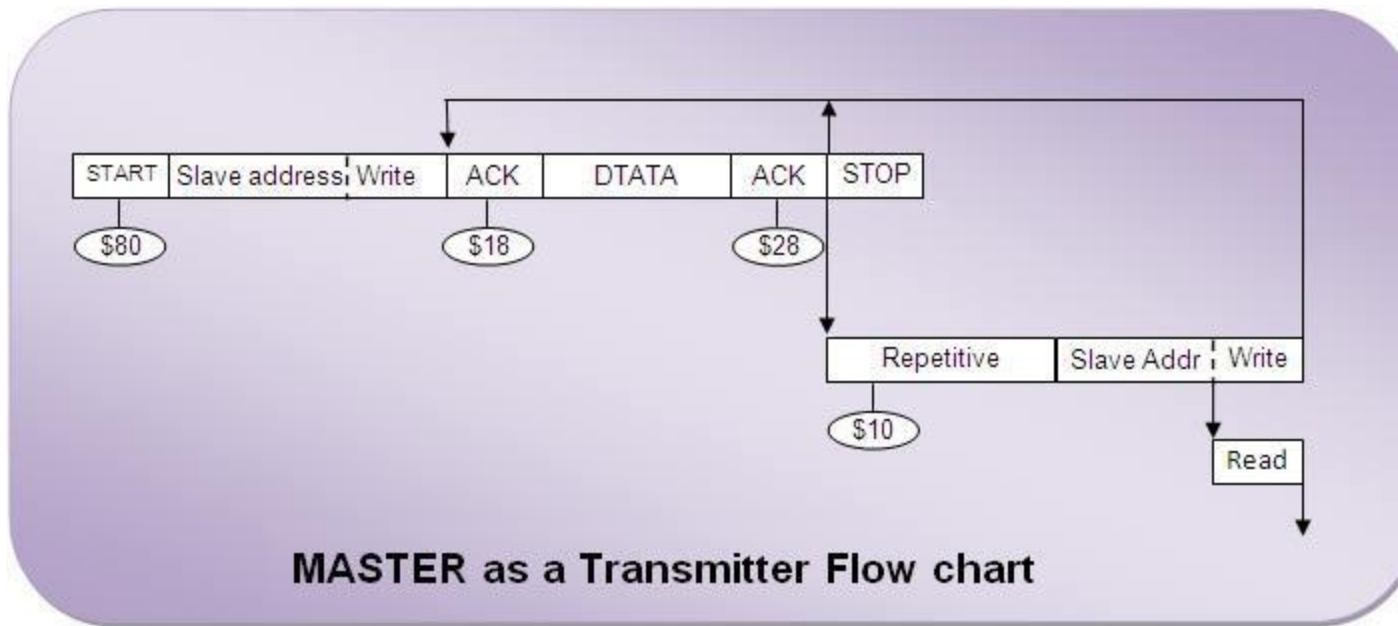
START condition	7 bit slave address	Data direction bit	Slave ACK	8 bit data transmission	Receiver ACK	STOP condition
-----------------	---------------------	--------------------	-----------	-------------------------	--------------	----------------

To send the stop condition use TWSTO

1. Clear TWINT flag.
2. Set TWEN bit
3. Write logic one to TWSTO bit so send STOP condition on SDA and SCL line.
4. Monitor the status of TWSTO bit, as TWSTO bit get cleared means the stop condition has been transmitted.

```
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
}
```

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.



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 start or stop condition in between. The second data can be transmitted just after receiving the acknowledgement of first data byte (as shown in the above flow chart).

#### Step 7: Send the slave address and data direction bit (read) and wait for the ACK signal

1. Put the 8 bit data in TWDR.
- 8 bits = 7 bit slave address + Data direction bit (read = 1).
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 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 acknowledgement
}

```

#### 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 TWINT 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 slave 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.

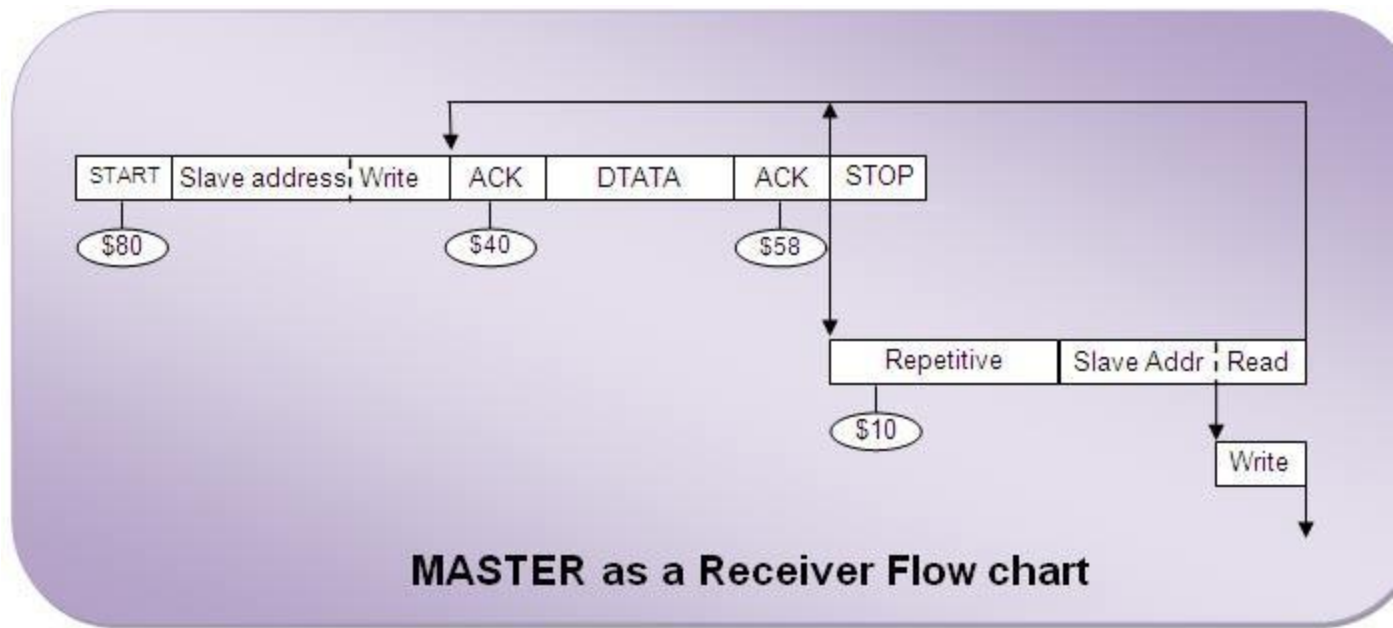
```

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 acknowledgement
    recv_data=TWDR;
    PORTB=recv_data;
}

```

#### Step 9: Send STOP condition

The stop condition is already explained.



### 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 used to enable the slave to acknowledge for the general call address (0x00).

```
void TWI_init_slave(void) // Function to initialize 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 be 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.
5. Match the status of TWSR. If the status is 0x60 read data or else jump to (1)

```
void TWI_match_read_slave(void) //Function to match the slave
address and slave direction bit(read)
{
```



```

    while((TWSR & 0xF8) != 0x60) // Loop till correct
    acknowledgement have been received
    {
        // Get acknowledgement, Enable TWI, Clear TWI interrupt
        flag
        TWCR=(1<<TWEA)|(1<<TWEN)|(1<<TWINT);
        while (!(TWCR & (1<<TWINT))); // Wait for TWINT flag
    }
}

```

### Step 3: Read data

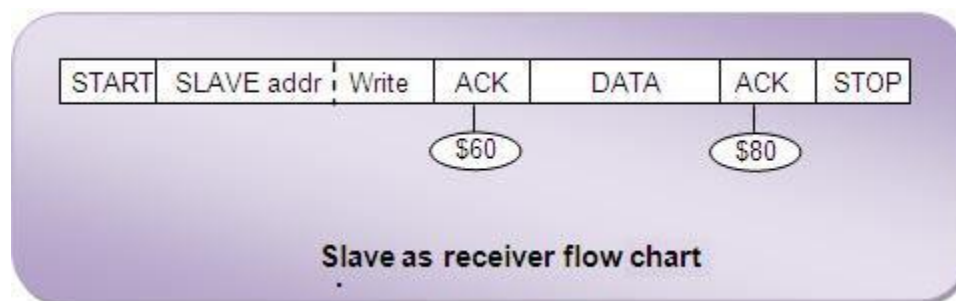
Read the data sent by the MASTER.

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 acknowledgement, 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
}

```



From here the slave becomes a transmitter on the request of master to send data. The AVR TWI works in mode 4.

### Step 4: Check the status of TWSR register



If the value of TWSR is 0xA8, it means the master wants to receive data from the particular slave and the slave sends back the acknowledgement to the master corresponding to the write operation.

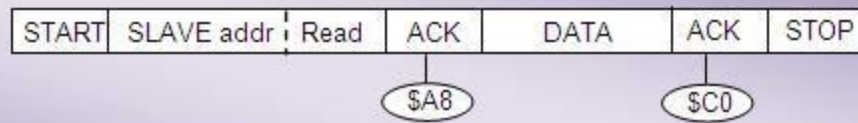
1. Clear TWINT flag.
2. Enable TWI.
3. Set TEWA to receive acknowledgement.
4. Monitor the status of TWINT flag.
5. Match the status of TWSR. If the status is 0xA8 send data or else jump to (1)

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

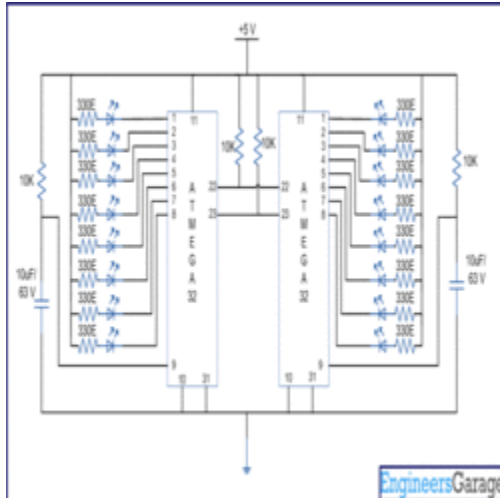
#### Step 5: Write the data on SDA bus

1. Put the data in TWDR.
2. Enable TWI.
3. Clear the TWINT flag.
4. Monitor the TWINT flag. As it gets cleared signifies the data has been sent.
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 sent and in turn it receives a NOT ACK and the status of TWSR becomes 0xC0. And if the TWEA bit was set during the data transmission it receives an ACK and the status of TWSR 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 with
    the data to be sent
    TWCR= (1<<TWEN)|(1<<TWINT); // Enable TWI, Clear TWI
    interrupt flag
    while((TWSR & 0xF8) != 0xC0); // Wait for the acknowledgement
}
```



Slave as transmitter flow chart



## MASTER

```
// 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=1;
unsigned char write_data=0x01, recv_data;

int main(void)
{
    _delay_ms(2000);
    DDRB=0xff;
    TWI_init_master();          // Function to initialize TWI
    while(1)
    {
        if(write_data==0x00)
            write_data=1;

        TWI_start(); // Function to send start condition
        TWI_write_address(address+write); // Function to write
address and data direction bit(write) on SDA
        TWI_write_data(write_data);          // Function to write
data in slave
        TWI_stop();          // Function to send stop condition

        _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 prescaler 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 acknowledgement
}

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
    while((TWSR & 0xF8)!= 0x18); // Check for the 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 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 acknowledgement
}

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 acknowledgement
        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
}

```

## SLAVE

```

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

void TWI_init_slave(void);
void TWI_match_read_slave(void);
void TWI_read_slave(void);
void TWI_match_write_slave(void);
void TWI_write_slave(void);

unsigned char write_data,recv_data;

int main(void)
{
    DDRB=0xff;
    TWI_init_slave(); // Function to initilaize slave
    while(1)
    {
        TWI_match_read_slave(); //Function to match the slave address
and slave direction 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 direction 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 with 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(write)
{
    while((TWSR & 0xF8)!= 0xA8)    // Loop till correct acknoledgement
have been received
    {
        // Get acknowledgement, 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 acknowledgement, 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
}

void TWI_match_read_slave(void) //Function to match the slave address and
slave dirction bit(read)
{
    while((TWSR & 0xF8)!= 0x60) // Loop till correct acknoledgement have
been received
    {
        // Get acknowledgement, Enable TWI, Clear TWI interrupt flag
        TWCR=(1<<TWEA)|(1<<TWEN)|(1<<TWINT);
        while (!(TWCR & (1<<TWINT))); // Wait for TWINT flag
    }
}

```

