Electronics Guide

Learn it Gain it

I2C Communication in AVR Microcontroller : PART 2

I2C Bus Protocol : The IIC (Inter-Integrated Circuit) is a bus interface connected which used in many devices such as sensors, RTC, ADC, DAC, Motion sensor and EEPROM. I2C communication is ideal for attaching low-speed peripherals to a motherboard or embedded system that require the reliable short distance communication. I2C devices use only 2 pins i.e. SDA which is used to transfer the data between two chips and SCL which is used to synchronize that data transfer.

I2C (TWI) in AVR : I2C is referred as Two-wire Serial Interface (TWI). In this section we will see how to program the AVR to address a slave device and send or receive data using TWI.

Programming of the AVR TWI in Master operating mode :
To work in master operating mode, we must be able to initialize the TWI, transmit a START condition , send or receive data , and transmit a STOP condition. Now lets see the essential steps in details and same are implemented in our code.

Initialization :
To initialize the TWI module to operate in master operating mode, we should do the following steps:
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 TWEN bit in TWCR register to one.

Transmit START condition :
To start data transfer in master mode, we must transmit the START condition. This is done by setting TWEN, TWSTA and TWINT bits of TWCR to one.
Setting TWEN bit to one enables the TWI module.
Setting TWASTA bit to one tells the TWI module to initiate the START condition when bus is free
Setting TWINT bit to one clears the interrupt flag to initiate the operation of TWI module to transmit START condition.
Then we should check the TWINT flag in TWCR register to see whether START condition transmitted completely.

Send Data :
To send a byte of data, after transmitting the START condition, following steps needs to be followed.
1. Copy the data byte to the TWDR
2. Set the TWEN and TWINT bits of TWCR register to one to start sending the byte data.
3. Observe the TWINT flag in the TWCR register to see whether the byte transmitted completely.

Note that right after the START condition, master should make sure that whether it wants to write or read from slave. For this purpose, i2c_write function is used. In this function master send
SLA + W(Slave Address + Write bit) or SLA + R(Slave Address + Read bit). Right after sending SLA+W we should write to the slave and after sending SLA+R we should read from slave.

Receive Data :
As stated earlier, after transmitting SLA+R, master is ready to receive a byte of data. Following steps should be done
1. Set the TWEN and TWINT bits of TWCR register to one to start receiving a byte.
2. Check the TWINT flag in the TWCR register to see whether a byte is received completely.
3. Copy the received byte from TWDR to another register in order to save it.

Transmit STOP condition :
To stop data transfer, STOP condition must be transmitted. This is done by setting TWEN, TWSTO and TWINT bits of TWCR register to one.

Note : Follow the below github link to see the detailed codes for operating arduino in master mode for write as well as read application.

Programming of the AVR TWI in Slave operating mode :
To work in slave operating mode, we must be able to initialize the TWI and able to send or receive data.
In slave mode, we cannot transmit START and STOP condition Now lets see the essential steps in details and same are implemented in our code.

Initialization:
1. Set the slave address by setting the value for TWAR register. The upper seven bits of TWAR register are address bits and eighth bit is TWGCE. If this bit is set to one , TWI module will respond to general call address otherwise it will ignore it.
2. Enable the TWI module by setting the TWEN bit in the TWCR register.
3. Set the TWEN,TWINT and TWEA bits of TWCR register to one to enable TWI and acknowledgement generation.

Listen To The Bus:
After initializing the TWI module, a slave device should listen to the bus to detect when it is addressed by a master device. When TWI module detects its own address on the bus, it returns ACK and sets the TWINT flag to one. One should check the TWINT flag to see when slave is being address by the master device.

Send Data :
After being addressed by the master, we should do following steps to send a byte of data.
1. Copy the data byte to the TWDR
2. Set the TWEN and TWINT bits of TWCR register to one to start sending the byte data.
3. Observe the TWINT flag in the TWCR register to see whether the byte transmitted completely.

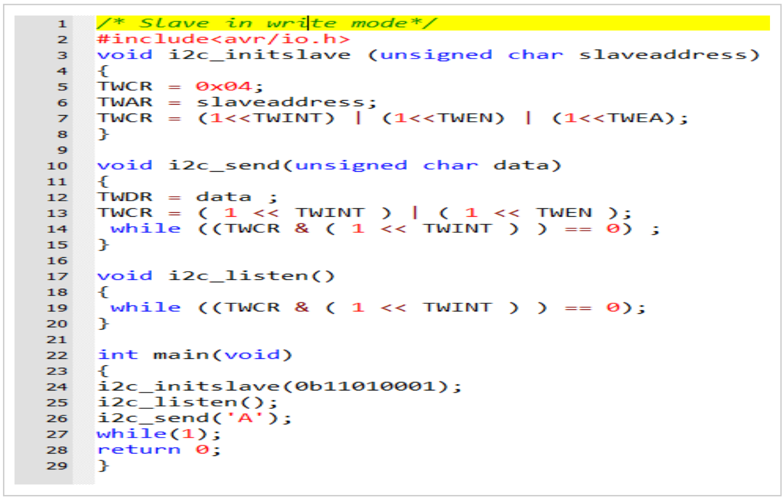
Receive Data :

After being addressed by the master, we should do following steps to receive a byte of data.

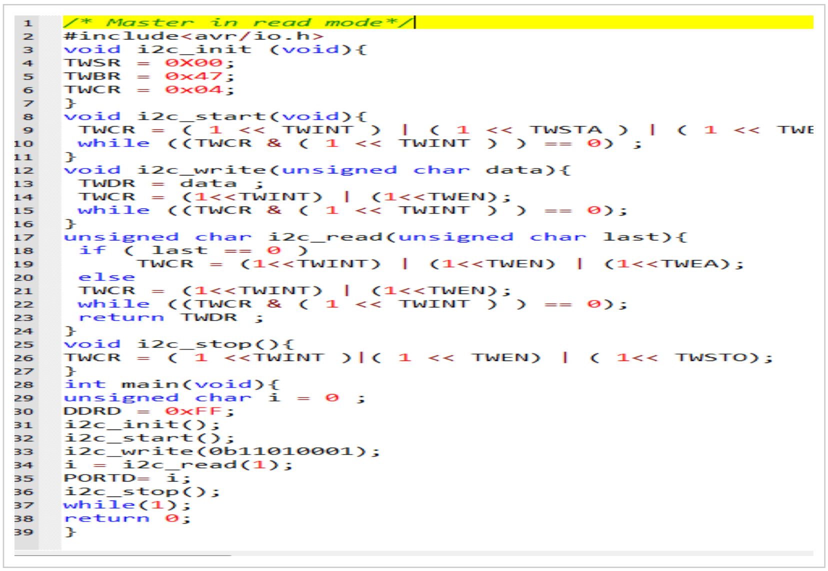
1. Set the TWEN and TWINT bits of TWCR register to one to start receiving a byte. If we want to return ACK after receiving the data , we should set the TWEA bit of the TWCR register to one.
2. Check the TWINT flag in the TWCR register to see whether a byte is received completely.
3. Copy the received byte from TWDR to another register in order to save it.

I2C communication between two Arduinos

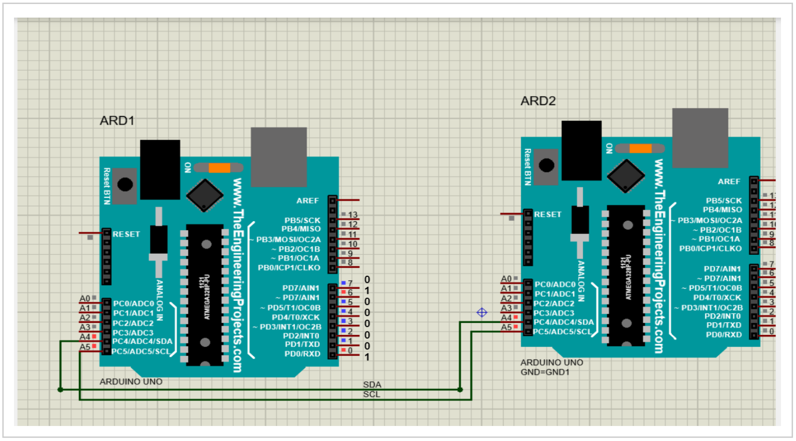
In this example we will see the simulation in which one arduino to work as slave in write mode which sends char 'A' to master and other arduino to work as master in read mode receives the input
char 'A' (01000001) and display its ASCII value on PORTD.



```c
/* Slave in write mode*/
#include<avr/io.h>
void i2c_initslave (unsigned char slaveaddress)
{
TWCR = 0x04;
TWAR = slaveaddress;
TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
}

void i2c_send(unsigned char data)
{
TWDR = data ;
TWCR = ( 1 << TWINT ) | ( 1 << TWEN );
 while ((TWCR & ( 1 << TWINT ) ) == 0) ;
}

void i2c_listen()
{
 while ((TWCR & ( 1 << TWINT ) ) == 0);
}

int main(void)
{
i2c_initslave(0b11010001);
i2c_listen();
i2c_send('A');
while(1);
return 0;
}
```



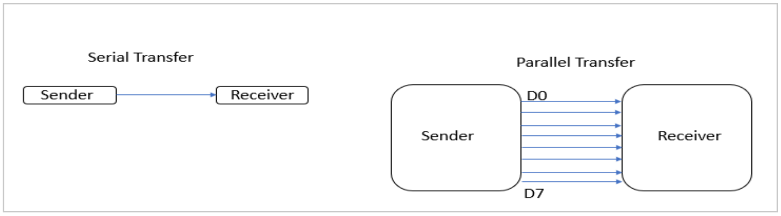
```c
/* Master in read mode*/
#include<avr/io.h>
void i2c_init (void){
TWSR = 0X00;
TWBR = 0x47;
TWCR = 0x04;
}
void i2c_start(void){
 TWCR = ( 1 << TWINT ) | ( 1 << TWSTA ) | ( 1 << TWE
 while ((TWCR & ( 1 << TWINT ) ) == 0) ;
}
void i2c_write(unsigned char data){
 TWDR = data ;
 TWCR = (1<<TWINT) | (1<<TWEN);
 while ((TWCR & ( 1 << TWINT ) ) == 0);
}
unsigned char i2c_read(unsigned char last){
 if ( last == 0 )
     TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
 else
 TWCR = (1<<TWINT) | (1<<TWEN);
 while ((TWCR & ( 1 << TWINT ) ) == 0);
 return TWDR ;
}
void i2c_stop(){
TWCR = ( 1 <<TWINT )|( 1 << TWEN) | ( 1<< TWSTO);
}
int main(void){
unsigned char i = 0 ;
DDRD = 0xFF;
i2c_init();
i2c_start();
i2c_write(0b11010001);
i = i2c_read(1);
PORTD= i;
i2c_stop();
while(1);
return 0;
}
```



In above fig ARD1 is Master in Read Mode and ARD2 is Slave in Write Mode.

I2C Communication : PART 1

Basics Of Serial Communication :
When microprocessor communicate with the outside world, it provides data in byte-sized chunk of data.
Computers transfer data in two ways:
1. Parallel
- Usually 8 or more lines are used to transfer data to a device that is only a few feet away
- Furthermore, 8 bit data path is expensive.
2. Serial
- To transfer to a device located many meters away, the serial method is used.
- The data is sent one bit at a time



In serial communication single data line is used to transfer data instead of 8 bit data line of parallel communication which makes serial transfer cheap and enable to communicate over **large distance**.
For serial data communication to work , the byte of data must be converted to serial bits which is done by using a parallel in serial out **shift register**, then it can be transmitted over a single data line. Similarly, at receiving end there must be a serial in parallel out shift register to receive serial data and pack it into byte.
Serial data communication uses two methods , **asynchronous and synchronous**. The synchronous method transfers a block of data at time while asynchronous method transfers a single byte at a time.
It is possible to write a code for these methods but code may get long and tedious. Pease check out this blog which explains the asynchronous transmission of data serially using single pin.
Due to this reason special IC chips are made by manufacturer for serial data communication i.e. **UART,USART,I2C,SPI** etc.

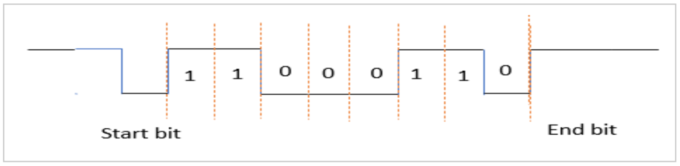
Now lets see one more terminology in communication i.e. **duplex communication** , if the data can be both can be both transmitted and received it is called as duplex communication. Duplex communication can be half or full duplex. if data is transmitted one way at a time is known as half duplex. If the data is transmitted and received simultaneously at a time, is known as full duplex. As suggested full duplex require two wire conductors. One for transmission and other for reception.

what is **protocol** in communication ? - The data coming at the receiving end of the data line in a serial data transfer is all 0's and 1's. It is difficult to make sense of the data unless sender and receiver agree on a set of rules i.e. the protocol, which gives more understanding on received data like how data is packed, how many bits constitute a character and when data start and start condition happen.

Asynchronous serial data communication is widely used for **character-oriented** transmissions. Each character is placed in between start and stop bits i.e. UART, this is called framing . **Block-oriented** data transfers use the synchronous method i.e. I2C.
The rate of data transfer in serial data communication is stated in bps (bits per second) which is also known as baud rate. In asynchronous serial data communication, the **baud rate** is limited to 100K bps.

Before understand I2C communication, lets have some look on how **UART** communication in general work which is asynchronous method of communication. UART uses Tx (transmitter) and Rx
(Receiver) pin along with common ground reference. Now, how receiver will know that the data is coming and how it is coming, so that's why transmitter and receiver should have common configuration such as 1. transmission speed in baud rate 2. data length in bits 3. start and stop bit.
So lets see what are these common configuration settings between transmitter and receiver of UART.

Suppose we want to send data of 198 number serially. It's digital pulse waveform representation looks like below if observe in oscilloscope.



198 = 11000110
Now this 1 byte of data can be set as data length. This sets data length configuration.
Usually Tx and Rx pins are always high so in order to send this data we need to initiate the communication and this done by pulling low this Tx pin. So this transition of HIGH to LOW pulse on Tx line will ensure the receiver that initiation has been started and this is known as start bit. Each time low pulse is detected means receiver will start to read the input data. This our start and stop bit configuration. Time period of these bits depends on baud rate which you will see next.

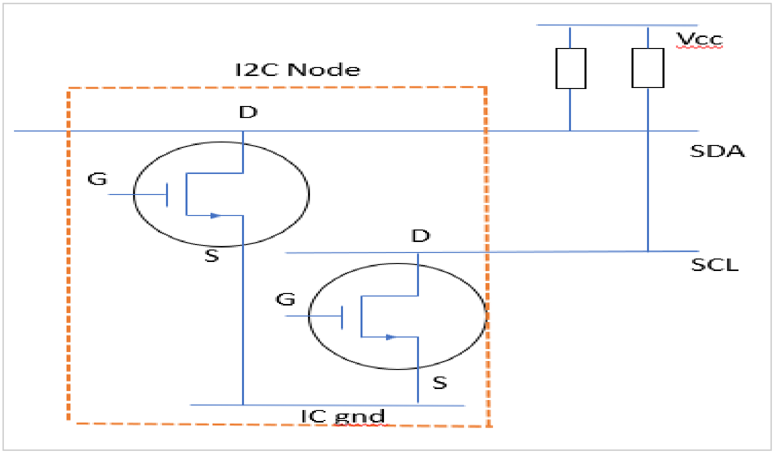
Now lets see the transmission speed configuration. Without knowing the speed, receiver will not be able to know when one bit start and ends. and they may lead to read a wrong value so that is why we need transmission speed configuration which is also known as baud rate setting.
Very common baud rate is 9600 bauds/second. This means that the length of bit will be 1/9600 = 104μs. After detecting the start bit i.e. LOW bit, the receiver has to count the time. We count for 104μs and we are at beginning of the 1st input bit i.e. start bit then again count for 104μs which will give bit 1 and again count for 104μs then we get 1 bit, again count for 104μs it will give 0 bit and this way we will end up to 8 bit data counting of 11000110 and after that stop bit which is HIGH bit is sent to know receiver that the communication has been ended. Then we can send the received data from the buffer to other system.
Data length and transmission speed can vary according to requirement but the start and stop bit configuration is almost same.
Application of UART communication in AVR - This protocol is used by AVR microcontroller to upload the code and also to send data to serial monitor.

I2C Bus Protocol :
So from above discussion what we observed that it was a receiver's double duty to read the bit period length while reading that bit. but in I2C communication we don't need to count the time as in UART communication. Lets see how - In I2C communication one wire i.e. SDA will send the data and other is clock i.e. SCL as it is a synchronous communication. In this case we at a same time when we sent the data, we also create a clock pulse as the same frequency as the bit of the data so receiver will exactly know when one bit is start and end and this makes it faster as compared to UART as counting the time task is eliminated.

I2C (Inter-integrated Circuit) is a **bus interface** which can be consists of many devices such as sensors, RTC and memory devices etc. **So why it is popular ?** - I2C is ideal for short distance communication. I2C uses only 2 pins. one SCL which synchronize the data transfer between two chips and other SDA through which data is transferred. This reduction in communication pins make it ideal to use where power and space constraints are important. It is often called as two wire interface.

I2C open drain pins :

I2C uses bidirectional open-drain (in case of MOSFET) / open-collector (in case of BJT) pins for communication. It has a open-drain/collector configuration at output. Also to implement I2C, generally 4.7kΩ pull-up register is used for each bus line which make a wire-AND interface. So what is its significance and how this arrangement works.



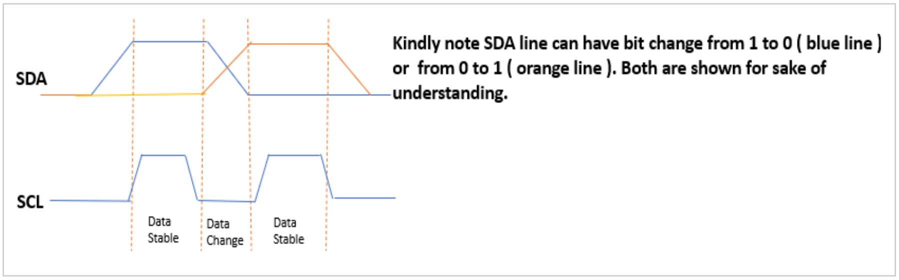
In open-drain arrangement, the control signal is applied to gate terminal of MOSFET whose drain is open and source is connected to ground. So when control signal is not applied and MOSFET is turned off, due to the external pullup resistor, the I2C bus remains HIGH. This means that the devices can pull the corresponding signal line to LOW but cannot pull it to HIGH. This eliminate the conflict that one device is trying to pull it LOW and another is trying to pull it HIGH which eventually solves the power damage problems. Pull up resistors help to restore signal to HIGH when no device is pulling it LOW.

I2C Nodes :

The number of nodes which can exist on a given I^2C bus is limited by the address space and also by the total bus capacitance. in AVR up to 120 different devices can share I2C bus. In I2C communication each node can acts as a slave or master. It is a multi master/multi slave communication. **So what differentiate between master and slave ?** - Master is a device which generates clock for the system and it is also responsible for initiating and terminating the transmission. Slave is a device which receives the clock from master and which can be addressed by master.

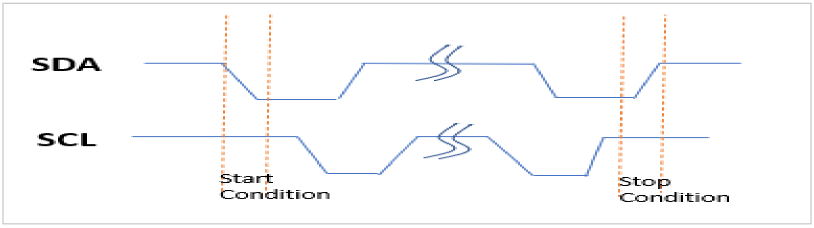
In I2C communication, both master and slave can transmit and receive data which leads to four modes of operation in I2C i.e.
1. master as transmitter 2. master as receiver 3. slave as transmitter 4. slave as receiver.

I2C Bit Format :
As I2C is synchronous communication what it indicates that the each bit on SDA is line is synchronized by HIGH to LOW pulse of clock line. So In I2C protocol the data line cannot change when clock is high; it can only change when clock is low. In below fig you can clearly see that data is stable when clock is high but at low clock line, data line changes its bit.


SDA
Kindly note SDA line can have bit change from 1 to 0 (blue line) or from 0 to 1 (orange line). Both are shown for sake of understanding.
SCL
Data Stable
Data Change
Data Stable

START and STOP bit format :

The above condition for data bits that we have seen above does not apply to the start and stop bit. It is exception to above rule. Let see how start and stop bit works. Each i2c transmission is initiated by putting START condition at SDA line and terminated by putting STOP condition. As seen earlier, both these conditions are generated by master only. In order to differentiate from data bits, START and STOP bit conditions generated differently. Let's see how ? START and STOP condition is generated by keeping the level of SCL line high and then changing the bit level of SDA line.
The START condition is generated by a HIGH to LOW transition of SDA line when SCL is high.
The STOP condition is generated by a LOW to HIGH transition of SDA line when SCL is low.
The bus is considered busy between each pair of START and STOP condition, no other master can access the bus when it is busy.



Packet format in I2C :

In i2c communication, if you want to send either address or data, it should be in a packet.
Each packet is 9 bits long out of which first 8 bits are put on SDA line by transmitter and 9th bit
(acknowledgement bit) is put by receiver which can be ACK (acknowledged) or NACK
(not - acknowledged). Now how receiver put this bit on SDA line ?
For this purpose, transmitter releases the SDA line during ninth clock and that's how receiver can have control to SDA line to pull it to LOW to denote ACK and if not pulled LOW, it is considered as NACK. Now in order to complete transfer data,
START condition + address packet + one or more data packet +STOP condition together form a complete data transfer. Now lets each packet format in details:

Address packet :

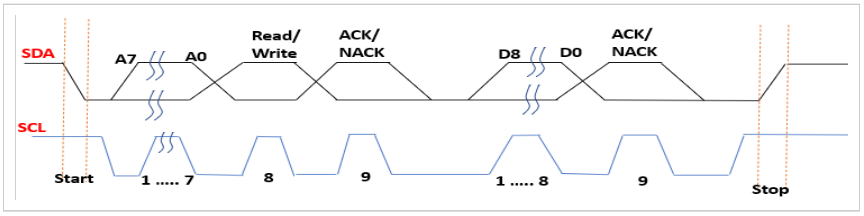
As stated earlier each packet is consists of 9 bits. In address packet first 7th bits form slave address.
The 8th bit is READ/WRITE control bit. If this bit is set, the master will read data from the slave,
otherwise the master will write data to the slave.
When slave detects that the address on SDA line, it knows that it is being addressed by master and if it is ready, it should acknowledge this address, slave must pull SDA line LOW at nineth clock.
If slave is not ready by any reason it should SDA line as it is i.e. HIGH in order to send NACK.
This will denote the master that it can either STOP the transmission or repeat a START condition in order to reinitiate the transmission.
We can say that address packet = Slave address + READ/WRITE
Special Note : The address 0000000 is reversed for general call. That means if transmitter wants to transmit the same data byte to all slaves in system, master can put this address on SDA line. But note that this is not valid to read data from all slaves as only one slave can write data on SDA line.

Data packet :

This packet is also consists of 9 bits long. The first 8 bits consists of byte of data to be transmitted and 9th bit is ACK bit. If receiver has received the last byte of data and there is no more data to be received, or receiver cannot receive more data, it will send NACK signal on SDA line by leaving SDA line HIGH as it is.

Combine address and data packet :

In I2C, generally a transmission is started by START condition, followed by an address packet
(slave address + READ/WRITE), data packets and finished by a STOP condition. Refer below fig.



Here in I2C data frame, at beginning START condition is there after that, A7 .. A0 (MSB first) denotes slave address for clock cycle from 1 to 7 . Then READ/WRITE bit at clock 8 and ACK/NACK bit at clock 9. D8 to D0 denotes data byte from clock cycle 1 to 8 then at 9th cycle there is ACK/NACK bit followed by STOP condition.

Home Older Posts

Subscribe to: Posts (Atom)

Powered by Blogger.