I2C

I2C → Inter-Integrate circuit

→ Widly used in these protocals in the companies

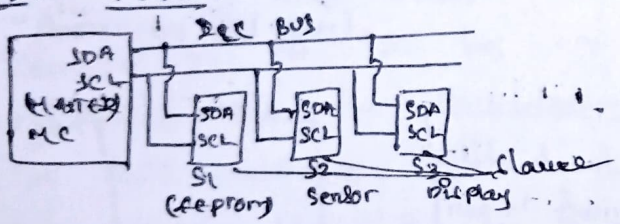
- UART, I2C, SPI → Embedded
- CAN, CAN FD, LIN, UDS → Automotive industry

Protocal:- It is not a hardware & software
* It is a set of Hd rules.

→ I2C protocal:- Means the set of Hd rules to the circuit writing the programming.

→ I2C devices: sensors, LCD-display

* Multi slave communication is not possible in the UART protocal.
* To over-come that I2C is designed
* It is a two-wire communication (SDA, SCL).
* We can communicate multiple device through the multiple slave in the I2C protocol



→ Master can communicate with slaves not at a same time, it communicate with one by one

→ Advantage of I2C :-

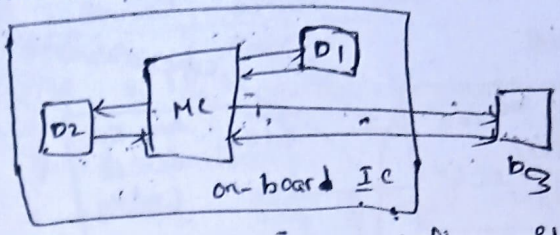
→ By these two wires we can communicate many devices one by one.

→ Power consumption is more while connecting partical (more devices)

→ We need pull-up register in I2C protocal to match input v/g of the devices

Introduction of I2C

→ It was designed by Philips (NXP level for on-board IC communication (or) short distance communication.



→ while MC is communicating with D1 & D2 it is called on-board IC communication

→ in case MC is communicating with D3 it is called off-board I2C

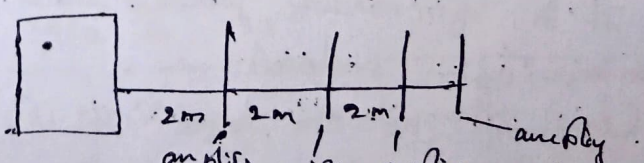
→ off-board I2C communication is also possible using I2C protocal with a small distance (less 2meters)

→ I2C RS232 cable is used for longer distance

*** How to send data for a longer distance using I2C protocal?

⇒ I2C protocal is designed only for shorter communication, by using repeaters we can communication for longer distance

Repeater is a amplifier



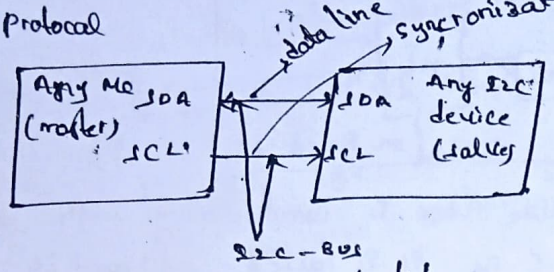
(amplify = Repeaters)

* for longer distance communication we can use UART, CAN, CAN FD

→ Features of I2C protocol
→ Hardware setup → I2C-fields
→ Data frames of I2C

FEATURES

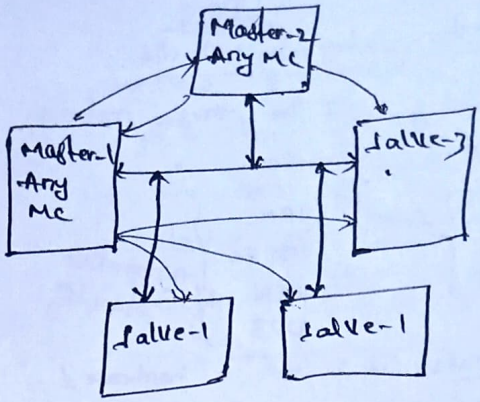
2. It is two wired Commo protocol



→ Master is modern like,
→ It will control the device
* only master can generate the clk plus pulses
* SDA is also called as data line we can read & write the data from the slave.
* SCL line is used for syncronas tion, when we send the data when we get the data it will control.
→ clk pulse speed set by the master
→ syncronization is controlled by the clk pulse.

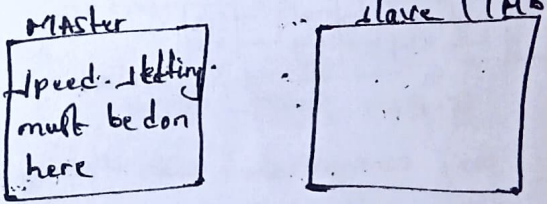
3. I2C is Half-duplex
It is syncronous serial communication protocol (common clk line used in master & slaves)

4. I2C is multi-slave & master Communication protocol:



→ One Master can communication with all slaves & one master can communicate with another master also it is called Multi-slave & multi master Communication
Note:- Not communicate at a time; one by one communicate.

5. Speed of I2C is upto 5Mbps (Mega bits per second)



→ we can set communication speed <= 1Mbps

upto 100Kbps:- standard speed mode
upto 400Kbps:- fast speed mode (FM)
upto 1Mbps:- FM+
upto 3.4Mbps:- High speed mode (Hs)
upto 5Mbps:- Ultra, High speed mode (UHS)

* Note only Master can do these settings

~~b. Every~~

6. Every slave of I2C protocal carries with predefined address which is given by the designer
* size of address is 7 bits (or) 10 bits (2^7 & 2^{10})
* Every I2C devices comes with its own address (0x53)

→ IPV6 - Versions

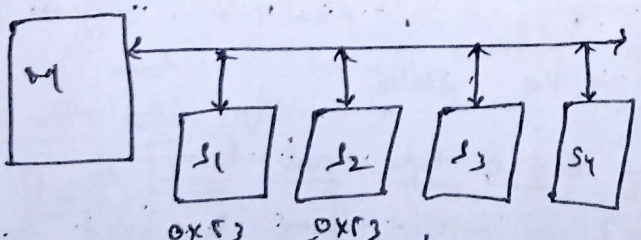
⇒ We can give every elements on earth including human, they gives IP addresses

⇒ IOT → Internet of things

⇒ Now Companies are using IOT (Internet of Everything)

7. The Max no. of slaves that are connected on the I2C bus are restricted by the slave address (or device address)

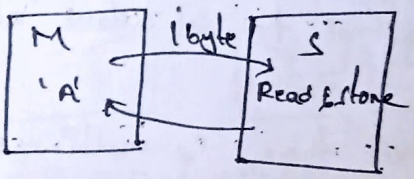
Note: Slave address is also called device address



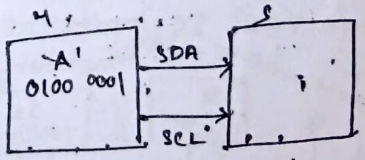
* We can connect multiple slaves but restricted.
* We can't connect same type of slave to slave bus, it will be corrupted the data

→ How to connect two slaves with the same address?
* We can connect two same slaves with the same address through the I2C Multiplexer.

8. It is an acknowledgement based protocol
* In the I2C protocal, master will get acknowledgement after every 8 bits written into slave.
* Acknowledgement represents the data has been received successfully (replay)
* After writing 1 byte of data slave will give acknowledgement to master.



9. Data transfer direction is from MSB to LSB.



→ Byte by Byte transmitted from MSB to LSB

10. It supports multi-master arbitration mechanism.

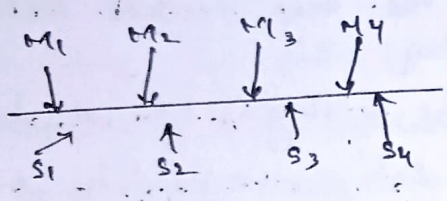
→ What is arbitration?
* When 2 or more than two masters are sending a data at a same time on the bus.

10. It supports multi-master arbitration mechanism.

What is arbitration?

When two or more than two masters are sending a data at the same time on the bus, it will clash the data & data gets corrupted.

To avoid this data Corruption the mechanism used in the multi-master communication protocol is called as, arbitration.



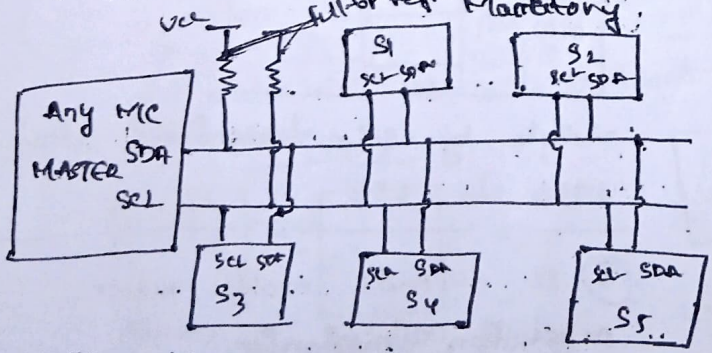
→ winner will decides by the parity of the slave address.

Note:- Where there is multi-master comms there is arbitration

---

I2C Hardware Setup

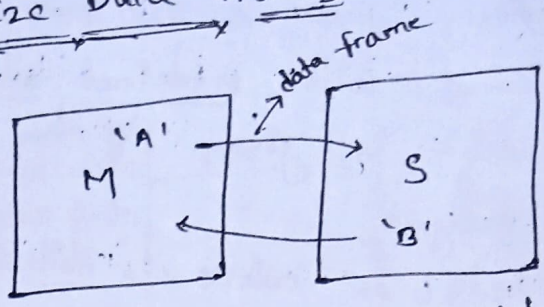
* ONE MASTER IS MANDATORY.



→ pull-up (1k-10k) registers are used to adjust i/p utg's of the slaves.

→ Master utg's is different & slave utg also diff.

I2C Data frames



→ Master wants to write data into slave, in I2C-protocal master can not write data into the frame

→ for writing a data, master creates the write data frame, if we read the data, master creates the read data frame.

→ stop & start conditions are available not stop & start pins.

→ I2C-protocal data frame is bigger as compare to UART

---

they are two types of frames

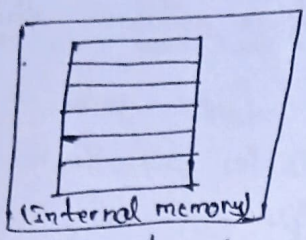
1. I2C byte write frame:-

this frame is used by the master to write 1byte of data into a particular address of the slave.

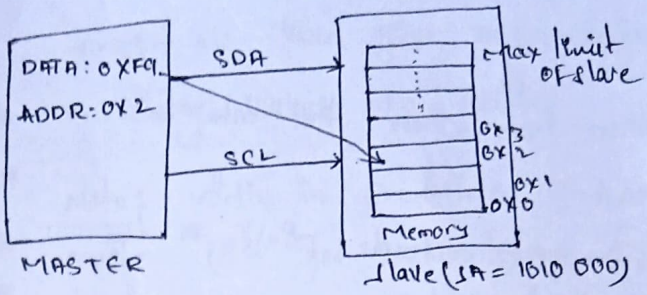
2. I2C byte read frame:-

this frame is used by the master to read 1byte of data from the particular address of the slave.

understanding slave address & Memory address of the slave I2C device



Any I2C device (I2C slave)

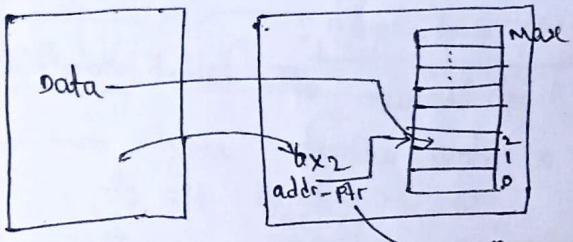


start	0XA0	ACK	0X2	ACK	0XF9	ACK	stop

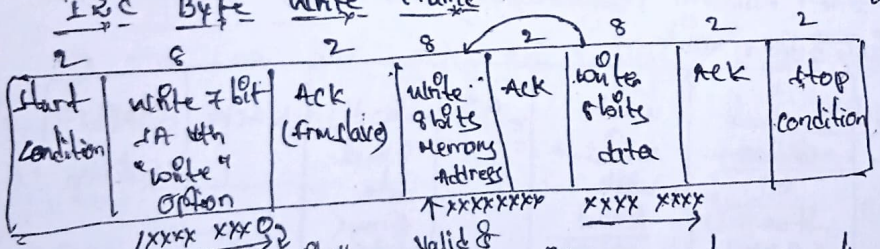
* Every I2C device has its unique device address also called as slave address → slave address is the name of the slave
→ Memory Address is the where the data to be store at a particular location

* Every I2C device comes with internal memory (volatile / non-volatile)
volatile means changes.

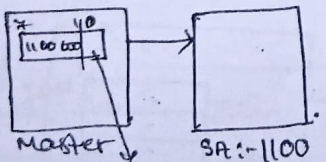
* Every I2C has
* size of the internal memory changes from devices to devices



I2C Byte Write Frame



* SA with write option:-



0th bit means R/W → for write '0'
→ Read '1'.

→ size of the frame is 34

Note:- If the master is generating start Condition: master should slave address with read/write option

* If master is given write operation then after Ack, Mr_address
* If master is read operation then after Ack, they can write mr_address

Start → 1010 0001 → A → read_data
↑
Read

where from where the data to read ?

ans:- where addr_ptr pointing of the location. now.

we want read_data frm the particular location. How?

ans:- Address to Addr_Ptr

Start → 1010 0000 → A → MA → A → Start → 1010 0001
↑ W 0X0 ↑ R
slave address

Stop ← NA ← read_data ← A

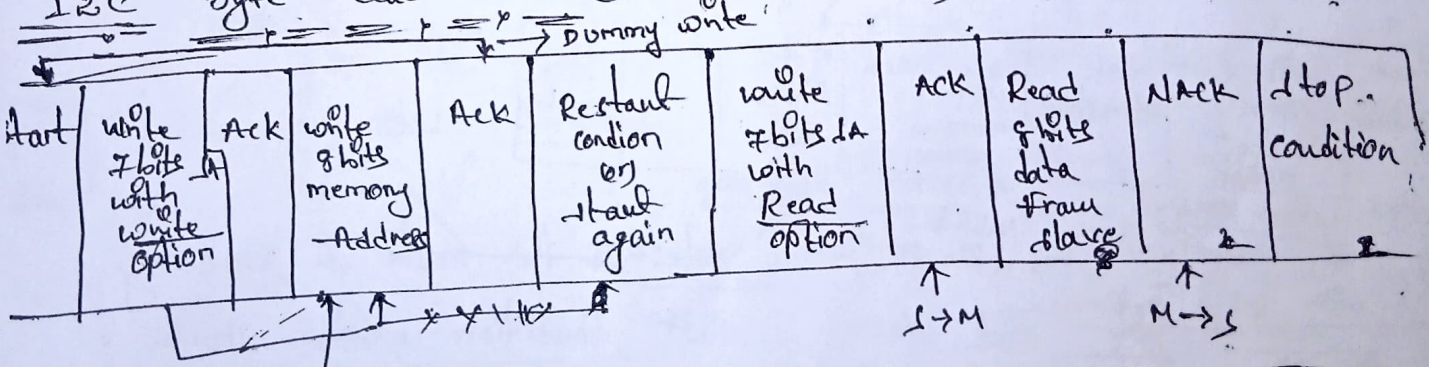
→ why start again ?

⇒ Because after start there will a slave address.

I2C Fields in the I2C Data Frames

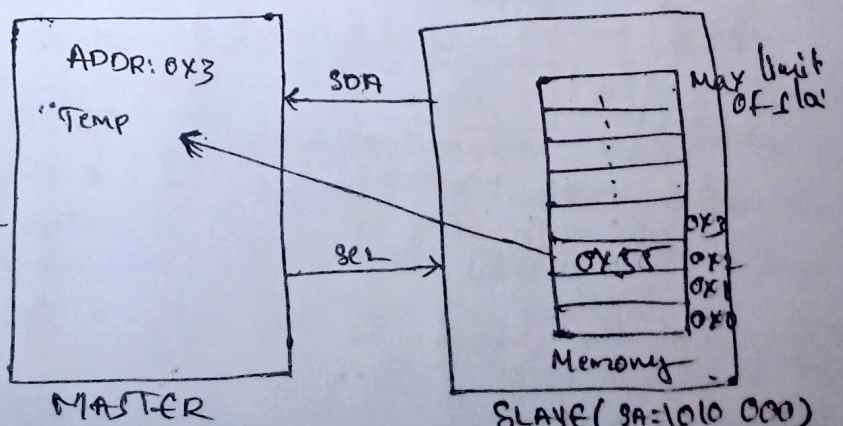
* I2C start
* I2C stop
* I2C write
* I2C read
* I2C Ack.
* I2C Nack.

I2C Byte Read Frame (Random read frame)



location from which master want to read data

Start → 0XA0 → Ack → 0X3 → Ack → Restart
W Read data.
NA ← 0X55 ← Ack ← 0XA1
R
Stop



Site :- 46

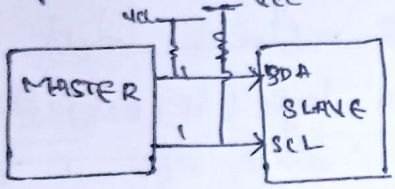
* understanding I2C Fields
* Implementation of I2C protocal using Bit Banging Method

* Idle Condition of the I2C bus

* when Master & slave not communicating wt should be the level of wires SDA & SCL?

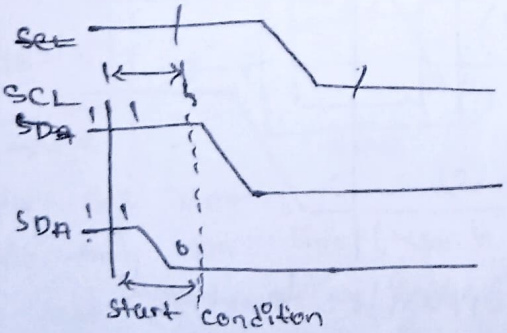
→ Idle Condition of the I2C bus representing with '1' (high volt)

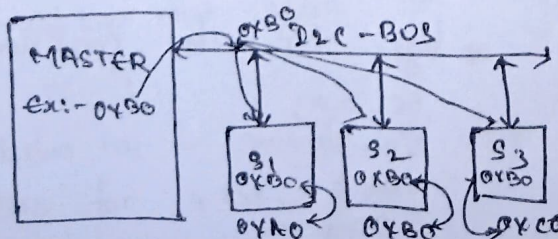
→ A bus considered idle if both SDA & SCL lines are high after a STOP condition.



I2C start Condition

A high to low transition (change) on the SDA line while SCL is high, defines a start condition.





Q:- Suppose if slave is not present wt would will happen?

Ans:- No - slave give acknowledgement remaining slaves are waiting

Note:- with in predefined time master will get acknowledgement from slave (3.4ms) 100kbps

for 3.4ms w.r.t to master understand No such a slave is not present.

* when stop condition occurs the became to the Idle-condition (I2C-bus).
* It can't possible to send two to a slave address at a time

Protocol implementation methods

→ two types

(i) Bit Banging Method - Software method of protocal
(ii) Peripheral method

(i) Bit Banging Method

Software method of protocal implementation. Here master has to generate clock pulses using software & it has to send and receive each bit using software only.

(ii) Peripheral Method

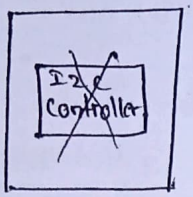
Hardware method of implementing protocal. In the master if I2C controller is available then the peripheral method can be used to implement protocal.

I2C - controller :- dedicated hardware used for I2C communication

→ This I2C Controller will take care of generating clock pluses & sending & receiving of data bits serially.

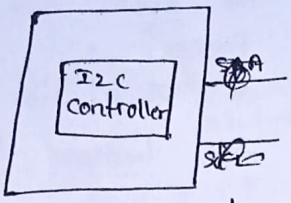
→ This i2c Controller reduces programming burden & increases efficiency.

eg:- any version of 8051



Bit-Banging method

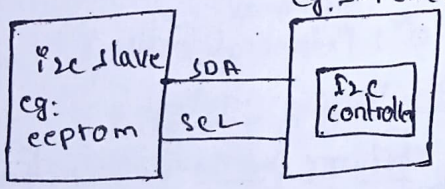
eg: LPC 2148/LPC2129



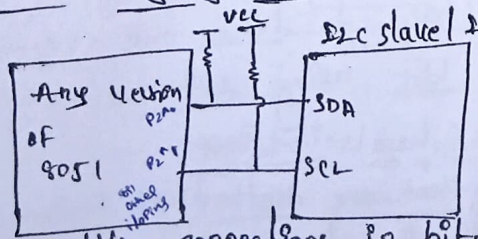
Peripheral method/ Bit-banging method

Note:- Suppose in peripheral method in-built I2C controller they fixed pin numbers available for SDA & SCL.

eg:- LPC2148/LPC2129



Bit Banging Method



H/W connections in bit-banging Method

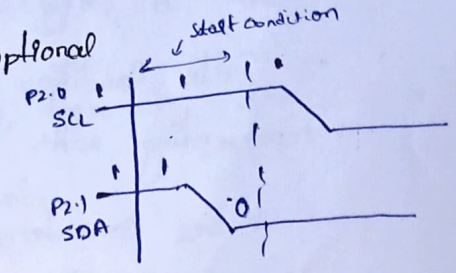
Note:- In the bit-banging method to interface i2c device we can use any two IO pins

Program for start condition

```
/* i2c_fields.c */
#include<reg51.h>
#include<header.h>
sbit SDA = P2^0;
sbit SCL = P2^1;
```

} up-on reset port pins are high

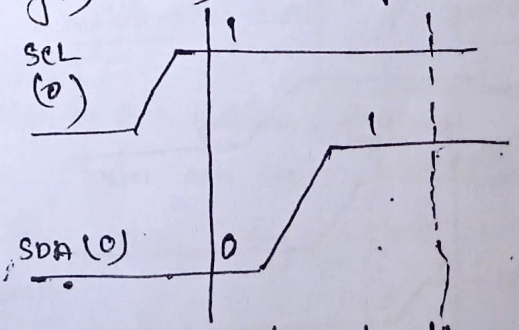
```
void i2c_start(void)
{
SCL=1; SDA=1;   } optional
SCL=1;
SDA=1;
SDA=0;
}
```



Note:- In bit-banging method for multi-master we must written a logic (if (SCL==1 && SDA==1))

I2C_Stop Condition (change)

* A low to high transition on the SDA line while SCL is high, defines a stop condition.

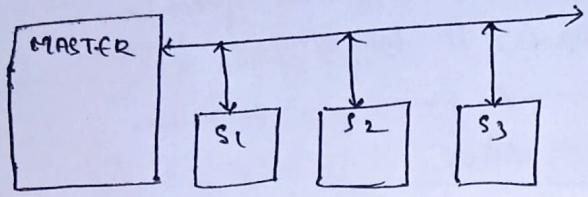


(opposite to start)

* logic for stop condition

```
SCL=0;
SDA=0;
SCL=1;
SDA=1;
```

```
void i2c_stop(void)
{
  SCL = 0;
  SDA = 0;
  SCL = 1;
  SDA = 1;
}
```



S | SA+W | A | Mr-add | A | data | A | P → stop condition

↓ start condition

→ BUS become free while bus becomes Idel condition i.e., stop condition

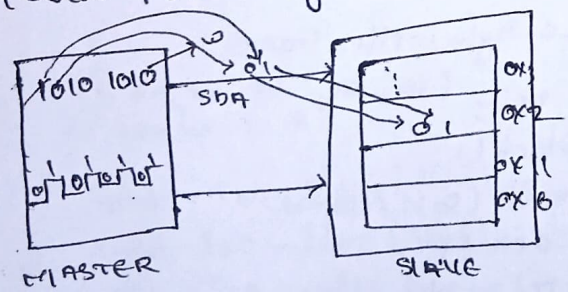
---

I2C_write

* How to write 8-bits into slave ?

it may be SA address, memory address 8 bits data.

Note:- Master can write 1 bit using 1 clock pulse only.



→ when SCL becomes '1' then read 1 bit from SDA line (MSB to LSB) for next bit read we must clear SCL '0' and read again like through the SCL '1'.

→ Master has to generate '8' clock pluses for 8 bit data.

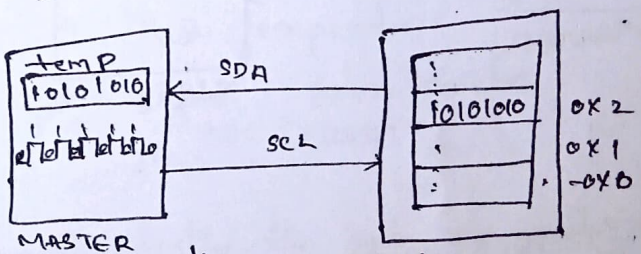
```
void i2c_write(u8 d)
{
  s8, i;   → signed variable
  for (i=7; i>=0; i--)
  {
    SCL = 0;
    SDA = ((d>>i)&1); // result '0' or '1'
    SCL = 1;
  }
}
```

(Bit by Bit writing in slave)

I2C_Read

(Reading 8 bits from the slave)

```
u8 i2c_read(void)
{
  u8, temp = 0x0;
  s8, i;
  for(i=7; i>=0; i--)
  {
    SCL = 1;
    if(SDA==1)
      temp |= (1<<i);
    SCL = 0;
  }
  return temp;
}
```



(7 bit to 8th position in temp)

Note:- Master can read 1 data bit from slave using 1 clock pulse only

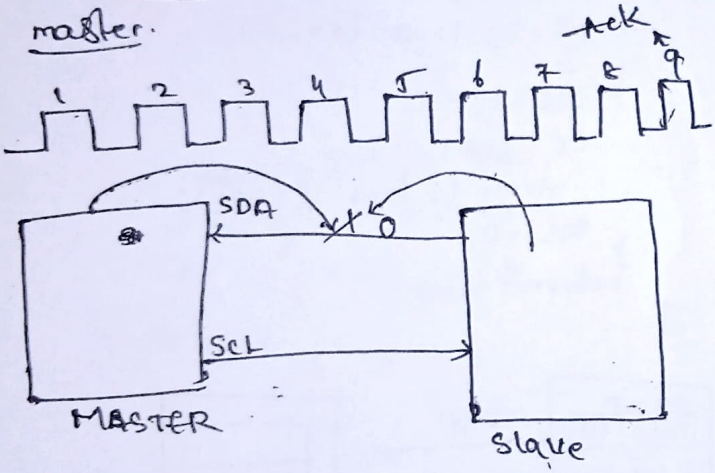
* for setting 7th bit at the msb location we want to check SDA line status (SDA==1)

Note:- Pheripheral method is mostly preffered, because of less burden to CPU also

I2C_Ack

→ Suppose slave have not given Ack, then automatically master generates stop Condition. (Display the Error Message)

* In the I2C protocol, master gets acknowledgment after every 8 bits written into the slave.
* Every 9th clock pulse is reserved for acknowledgement.
* By writing 0 (zero) on the SDA line slave will acknowledge to master.




```
bit i2c_ack(void)
{
  SCL=0;
  SDA=1; //master write 1 to SDA line
  SCL=1; //9th clock pulse
  SCL=0; // 3.4ms delay Function
  if(SDA==0) //master get
  { SCL=0;
    return 0; // got the Ack
  }
  else { SCL=0;
    return 1; // Error to stop condition
  }
}
```

#include <intrins.h>

→ Predefined fun's _nop_();

Macro

It gives 1ms delay.

```
_nop_(); // 1mc time
_nop_(); // 1mc time
_nop_(); // 1mc time
_nop_(); // 1mc time
```

} 3.4ms delay approx 4mc

I2C_NACK

→ NACK gives master to slave.
→ It is only read mode.

```
void i2c_nack(void)
{
  SCL=0;
  SDA=1;
  SCL=1;
}
```

---

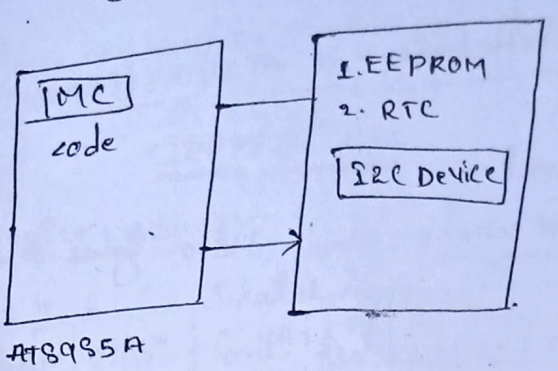
i2c_frame.c

```
/* i2c_frame.c */
#include "header.h"
void i2c_byte_write_frame (u8 sa, u8 mr, u8 d)
{ bit ret;
  i2c_start();
  i2c_write(sa); //SA+w
ret=i2c_ack();
  if(ret==1)
    goto out;
  i2c_write(mr);
  ret=i2c_ack();
  if(ret==1)
    goto out;
  i2c_write(d);
  ret=i2c_ack();
  if(ret==1)
    goto out;
out: i2c_stop();
}
```

```
u8 i2c_byte_read_frame(u8 sa, u8 mr)
{  u8 temp; bit ret;
   i2c_start();
   i2c_write(sa); //SA+w
   ret = i2c_ack();
   if(ret==1)
      goto out;
   i2c_write(mr)
   ret = i2c_ack();
   if(ret==1)
      goto out;
   i2c_start(); //restart condition
   i2c_write(sa|1); //SA+R
   ret = i2c_ack();
   if(ret==1)
      goto out;
   temp = i2c_read();
   i2c_nack(); //
out: i2c_stop();
   return temp;
}
```

for Aek checking

```
/* header.h */
void i2c_start(void);
void i2c_stop(void);
void i2c_write(u8 d);
u8 i2c_read(void);
bit i2c_ack(void);
void i2c_nack(void);
void i2c_byte_write_frame(u8 sa,
                          u8 mr, u8 d);
u8 i2c_byte_read_frame(u8 sa, u8 mr);
```



Interfacing of EEPROM with AT89S52 / W78E052D (Nuvoton 8051)

List of I2C Devices

* EEPROM (AT24C08 / AT24C04)
* RTC (DS1307 / DS3231)
* Digital thermometer (DS1621)
* OLED display
* Accelerometer
* Pressure sensor
* Temperature Sensor
* Port expander
* Digital potentiometer
* ADC (ADS1115)
* etc.

→ For testing the code we must to connect to the I2c device

Interfacing of AT24C08 (EEPROM) with AT89S52

Atmel

AT24C08 : Atmel 1Kilo Bytes of EEPROM
→ eeprom series

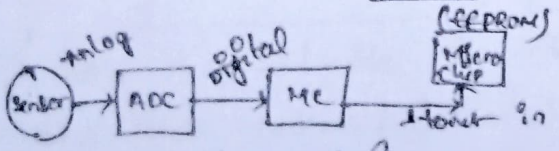
AT24C08 :- last two digits represents the size of the EEPROM
→ 8 represents 8kbits (8192 bits) 8x1024
→ 8t means that 8192/8 (1Kilo Bytes)
→ 8kB → kilo bytes
8kb → kilo bits

AT24C04 :- 4 kilo bits (4096 bits)
4096/8 → 512 Bytes

→ wt is EEPROM? → non-volatile memory
→ Ans :- EEprom is Permanent R/w data memory.
(Electrically erasable programmable ROM)

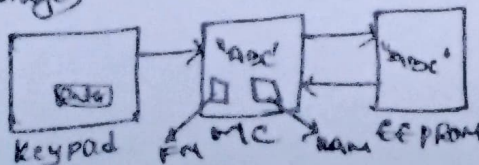
Ex:
Volatile → RAM (temporary memory)
Non-volatile → Hard Disk (Permeat Memory)

→ why EEPROM? ⇒ used in the
1. * Data logger system (DLS)
ex: Black box system in planes
→ It will stores all the informations of reading of sensors that is called logger



Data-logger system

2. * Security based system (Password storage)



* MC read from the eeprom & Compare with scaner if scaned password is match password which is stored in the eeprom.

3. * Electric Meters also to store little bit of inform

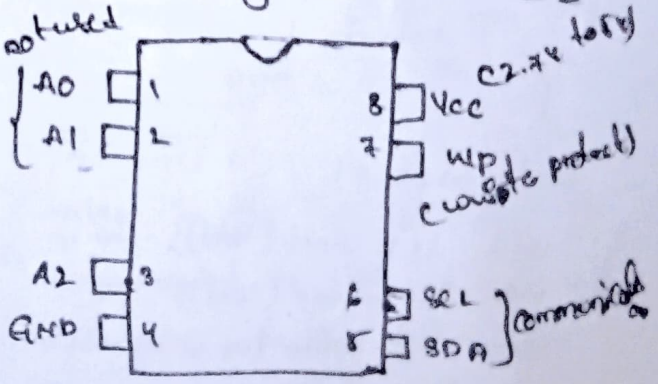
Introduction to AT24C08

supply → 5.0 (Vcc = 4.5 to 5.5V)
→ 2.7 (Vcc = 2.7 to 5.5V)

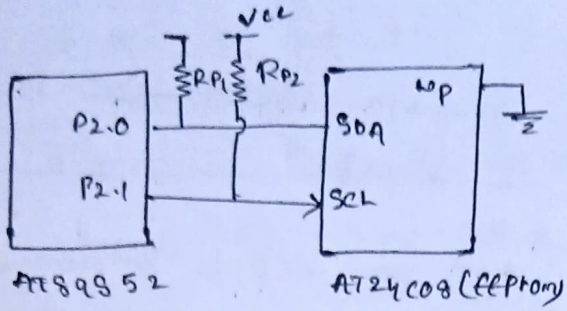
→ two-wire serial interface
* Schmitt trigger, filtered inputs for noise suppression.
* Bidirectional data transfer protocol
* 100kHz (2.7V and) 400kHz (5V) compatibility
* High-reliability
Endurance : 1M write cycles
Data Retention : 100 years

---

pin-diagram of EEprom

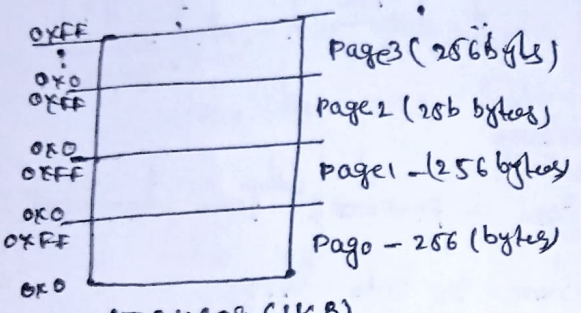


Interfacing of EEPROM to MC



* Pull-up registers are required

Memory Mapping of EEPROM



* There are four pages in the EEPROM
* Every page starts with 0X0 & ends with 0XFF
* Page selection can be done using program.

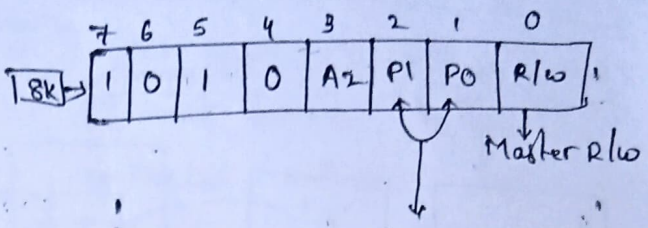
(S → SA+W → A → mr → 8 bits = 1 byte

How to read / write from EEPROM?

Slave Address / Device address of AT24C08 must required.

Slave Address / Device Address of AT24C08

* Device address:



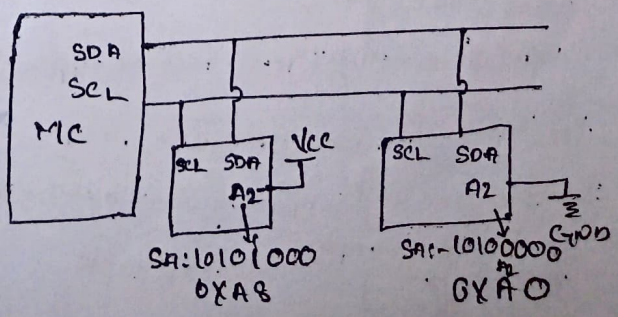
Eg:- * 0 to W (only)

	7	6	5	4	3	2	1	0
Page0 (SA) -	1	0	1	0	0	0	0	0
		A				0		⇒ 0XA0
Page1 (SA)	1	0	1	0	0	0	1	0
		A				1		⇒ 0XA2
Page2 [0XA4]	1	0	1	0	0	1	0	0
Page3 [0XA6]	1	0	1	0	0	1	1	0

* A2 pin:-

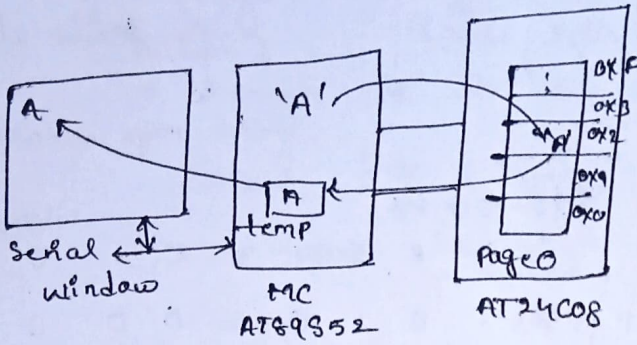
This pin is used to change device address of EEPROM (AT24C08), so that we can connect two EEPROM's on the same I2C bus.

Note:- In the I2C protocol, it is recommended to not connect same addressable device on the same bus.



* Through the AT24C04 we can use 4 EEproms, there is A1 & A2 pins available.

AT24C08 EEPROM Programming



* Taking 1 byte of data writing into eeprom (one of the address location)
* And read back data in a temp and display on serial window

* loopback test

↑ It is used for checking the device status i.e EEPROM if it is working properly (or) not.

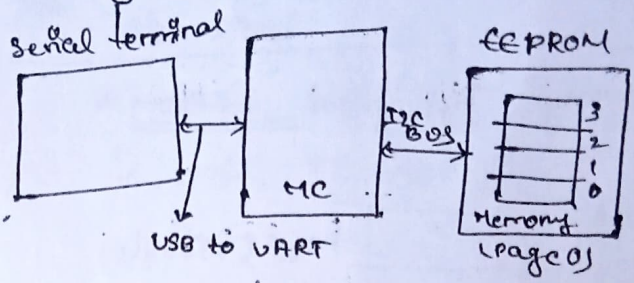
M24C08

```
/* main_eeprom.c */
#include "header.h"
main() {
  uf temp;
  uart_Init(9600);
  uart_tx_string("\r\n testing EEPROM");
i2c_byte_write_frame(0xA0,0x3,'A');
temp= i2c_byte_read_frame (0xA0,0x3);
  uart_tx (temp);   ↓ lookback.
  while(1); }
```

PC → MC → EEPROM → MC → PC

---

* Write an ECP, to implement password authentication program using UART & EEPROM.



Procedure

1. Scan a password from terminal & store in into array.
2. Read password from eeprom & Compare it with scanned password.
3. Display result on Serial terminal & repeat.

I2C Multiplexer

- Allow you to Connect same type of i2C devices (same addressble i2C devices) on the same bus.

---

extension & program for read data in runtime

```
while(1)
{ uart_tx_string("\r\n Character to write in
                  EEPROM: ");
  ch=uart_rx();
  uart_tx(ch); //loop back.
```

RTC – Real-time-Clock

* RTC will not sleep (or) not shutdown
* RTC will continuously running through the CMOS Battery (coin battery)

→ Why RTC?

* through the RTC we get current time & current date.
* It takes less v/g & current.
* Note:- RTC will not take the CMOS battery v/g all the time, only when the system to shutdown the PC.

→ Power-failure detect & switch circuitary is available in the RTC

eg:- Wheater monitering system

Introduction to DS1307 of RTC

↓
Dalla Semiconductor

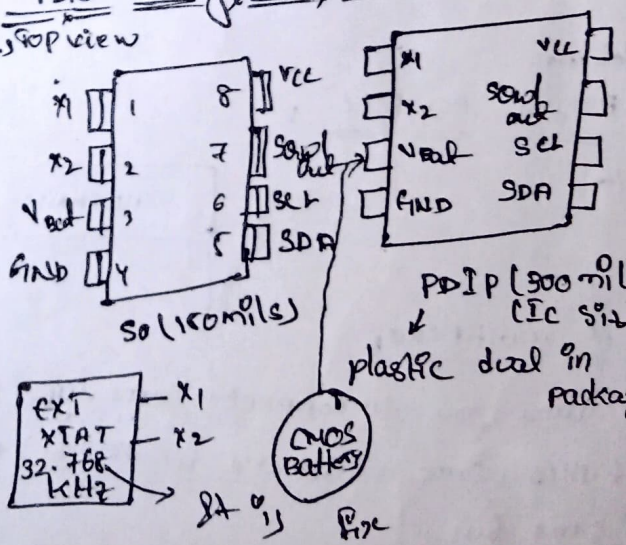
→ It is migrated maxim integrated
→ It is dedicated hardware.
→ 64 Bytes memory in the RTC.

Benefits & Features

* Completely managed All timekeeping functions (Real-time clock counts, secs, mins, hrs, days, mths, years)
  (i) 56-Byte, Battery-Backed, general purpose ram with unlimited writes
  (ii) Programmable square-wave o/p signal
* Simple Serial port interface to most Microcontrollers
  (i) I²C Serial interface
* Low power Operation Extends Battery Backup Run time.
  (i) less than 500nA (consumes) in backup mode
  (ii) Automatic power-fail detect and switch

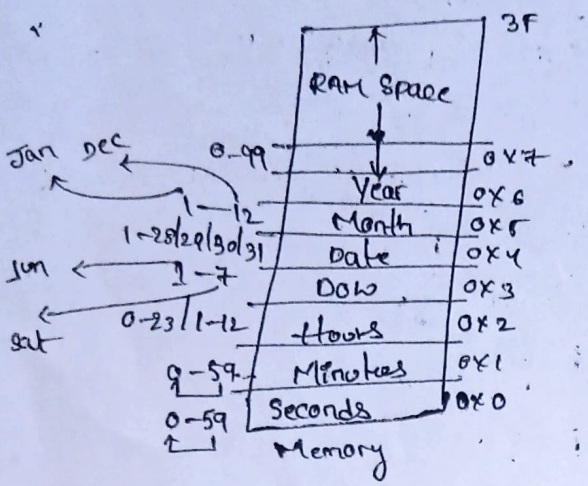
PIN Configurations

Top view



* memory size inside the RTC is 64 Bytes (0-63) memory locations (all address) are available.

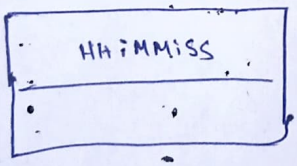
Memory Mapping of DS1307



* Data is safe in the RAM space
* Will when Memeury is dead the data will lose.
* only once to set the setting.

Address of DS1307 11010000
D
write
Device address is 0XD0

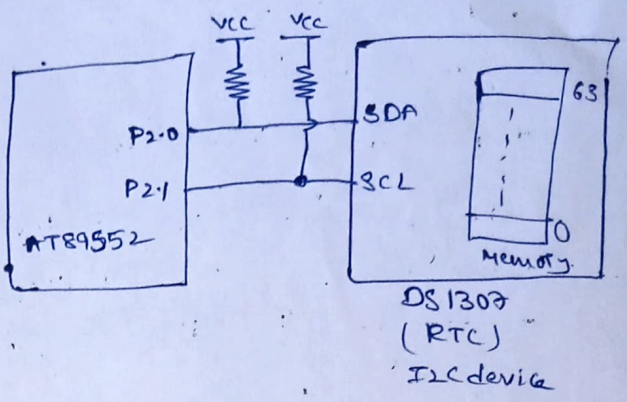
Program on RTC



I2C multiplexer:

=> allow you to connect same type of i2c devices (same addressable i2c devices) on the same bus.

Interfacing of RTC with AT89S52/ W78E052D




```
/* main - rtc . c */
#include "header.h"
main()
{ u8 h,m,s;
  lcd_init();
=> /* setting RTC time to 11:59:0PM */
  i2c_byte_write_frame(0xD0,0x2,0x23);
                          //setting hour
  i2c_byte_write_frame(0xD0, 0x1, 0x59);
                          // setting mins
  i2c_byte_write_frame(0xD0, 0x0, 0x50);
  lcd_cmd(0xc);//disable cusor  // setting second
=> /* Read RTC time & Print it on a lcd */
  while(1)
  { h=i2c_byte_read_frame(0xD0, 0x2); //read
    m=i2c_byte_read_frame(0xD0, 0x1); //read m
    s=i2c_byte_read_frame(0xD0, 0x0); //read s
    lcd_cmd(0x80);
    lcd_data(h/16+48);
    lcd_data(h%16+48);
    lcd_data(':');

    lcd_data(m/16+48);
    lcd_data(m%16+48);
    lcd_data(':');

    lcd_data(s/16+48);
    lcd_data(s%16+48);
  }
}
```

3