1. Introduction

This application note provides assistance and guidance on how to use GIANTEC I^2C serial EEPROM products. The following topics are discussed one by one:

- Power supply & power on reset
- Power saving
- IO Configuration
- Check completion of Write Cycle
- Write-protect application
- Data throughput
- Schematic of typical application
- Recommendation of PCB Layout
- Reference design of software

2. Power supply & power on reset

GIANTEC 24 series EE products work well under stable voltage within operating range specified in datasheet respectively. For a robust and reliable system design, please pay more attention to the following items:

2.1 Ensure VCC stable

In order to filter out small ripples on VCC, connect a decoupling capacitor (typically 0.1μf) between VCC and GND is recommended (Shown in figure 1). In addition, it is recommended to tie the pull-up resistor to the same VCC power source as EEPROM, if MCU is powered by a different VCC power source.

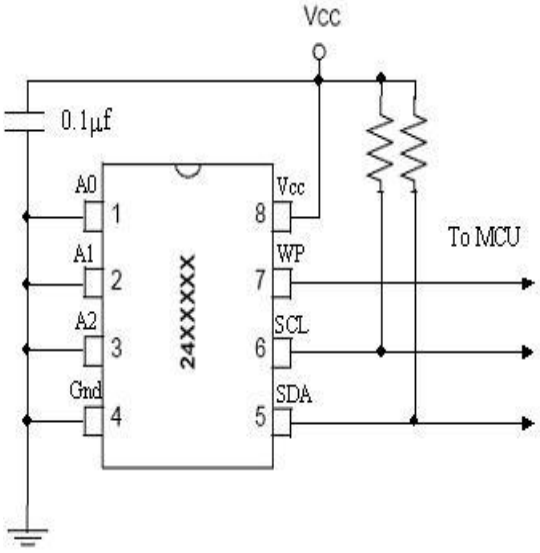


Figure 1: GIANTEC 24 series EEPROM recommended connections

2.2 Power on reset

During power ramp up, once VCC level reaches the power on reset threshold, the EEPROM internal logic is reset to a known state. While VCC reaches the stable level above the minimum operation voltage, the EEPROM can be operated properly. Therefore, in a good power on reset, VCC should always begin at 0V and rise straight to its normal operating level, instead of being at an uncertain level. Shown in figure 2. Only after a good power on reset, can EEPROM work normally. The operating range of VCC can be found in the datasheet. It is recommended to do software reset by MCU immediately after POR to further ensure the proper initialization of the device.

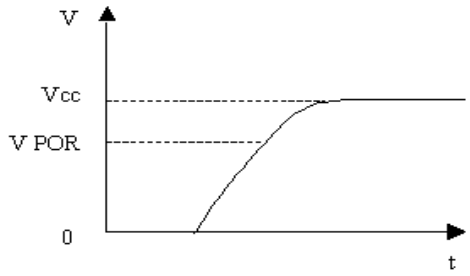


Figure 2: Power on reset

2.3 Power down

During power down, the minimum voltage level that VCC must drop to prior resume back to the normal operating level is 0.2V to ensure the proper POR process, Shown in figure 3

.

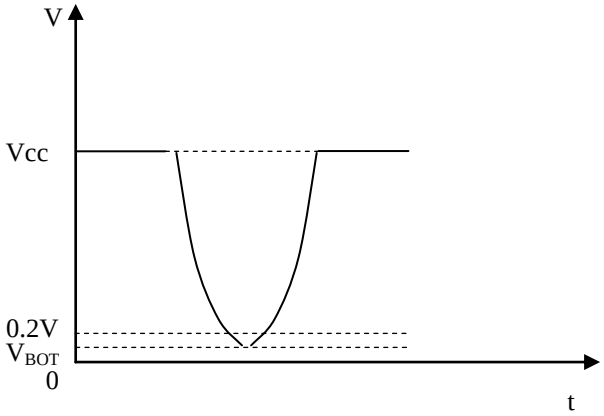


Figure 3: Power Down

2.4 Software reset

In case of no way to know the current state of EEPROM or want to cancel the current operation of EEPROM, usually software reset is used to make EEPROM enter standby mode, but software reset cannot reset the internal address counter of EEPROM device. Software reset is caused by a START condition that is sent by master device. During the process of EEPROM read/write operation, whenever recognizes the START condition, current operation will be stopped. However, the following cases need to be considered:

1) Master device sends a START condition while the EEPROM is executing a read instruction and sending bit "0" to master device, because SDA is being driven low by EEPROM, EEPROM cannot recognize this START condition and thus cannot make software reset happen. In order to solve this kind of issue, master device need to send nine sequential bits "1" after START condition, thus EEPROM will not be able to get the response from master device, therefore force an internal reset to be generated, Shown in figure 4.

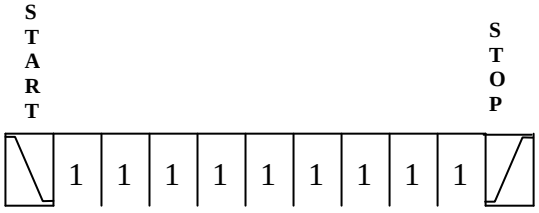


Figure 4: Software reset

2) Master device sends a START condition while the EEPROM is acknowledging a WRITE instruction and is driving SDA low, even though the master device sends nine sequential bits "1", the EEPROM cannot be reset. In this case, master device need to send a START condition after nine sequential bits "1" to make EEPROM reset, Shown in figure 5.

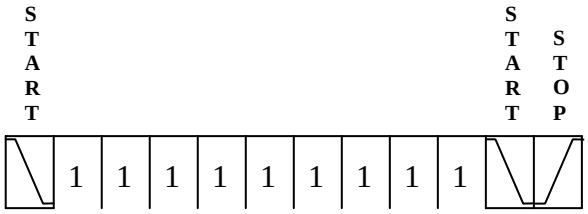


Figure5: Software reset

After device internal reset, master device may send a STOP condition to bring EEPROM into standby mode. Therefore, a full reliable software reset process will comprise a START condition, 9 sequential bits "1", a START condition and a STOP condition. The reference code can be found in chapter 10.

3. Power saving

To reduce the power consumption, the following cases need to be considered:

3.1 Whenever no need to operate EEPROM, bring it into standby mode. In this mode, the power consumption is minimum correspondingly. Following cases can bring EEPROM into standby mode:
   1) After power-up, and remain this state until SCL or SDA toggles;
   2) Recognize a STOP signal after a non-read operation is initiated;
   3) Completion of internal write operation.

3.2 Usually, pull-up or pull-down resistor contributes to power consumption too. Under the same conditions, big resistor consumes less power, and small resistor consumes more power;

3.3 The power consumption is maximum correspondingly during its write cycle. If a large amount of data needs to be written into the EEPROM, definitely the page write mode consumes less time and power than byte write mode. Therefore, if there are a lot of data to be written, the page write mode should be used instead of byte write mode. If there are a lot of data to be read, the sequential read mode will be recommended. The sequential read mode can improve the read efficiency and reduce the power consumption as well.

4. IO Configuration

In order to reduce the possibility of wrong operation inadvertently due to noise, it is recommended that SDA pin and SCL pin should be tied to a proper pull-up resistor to improve the anti-jamming capability of EEPROM. If the pull-up resistors are not used in a ready-made application, it is recommended to set SCL and SDA high after POR. It will bring EEPROM into a known high-level state after POR. In general applications, the pull-up resistor needs to be considered at the beginning of design according to following suggestion:

1) Since pull-up resistor affects the coupling capacitor on SDA bus as well as the rise time of SDA, thus it will impact the bus transmission efficiency. For example, if read operation is executed quickly enough, SDA will be sampled already by MCU before it rises to stable high level, then MCU may get the wrong data. To solve these kinds of issues, the frequency of SCL need to be reduced. Usually reducing the MCU frequency or adding a fixed delay during the read operation will help a lot, however, meantime the transmission efficiency between MCU and EEPROM will be reduced too. Therefore, another way may be used to speed up bus rising time, technical speaking, small pull-up resistor will make SDA bus rise more quickly than big pull-up resistor, thus data transmission efficiency will be higher as well. In

general application, it is recommended to select pull-up resistor from 1.5k to 3.5k for fast mode and 3.5K to 12K for standard mode.

2) Technical speaking, only if the MCU IO pin which is tied to SCL is in open drain mode, a pull-up resistor is need by SCL, but in order to reduce the noise on SCL after POR, always it is recommended to tie a proper pull-up resistor to SCL. This will be helpful to make EEPROM device enter a known stable high-level state after POR. The pull-up resistor's value is recommended to be same with the pull-up resistor value on SDA.

5. Check completion of Write Cycle

Definitely, effective checking of the completion of write cycle will improve the efficiency of the write operation. Once recognize a STOP condition, EEPROM will start its internal write cycle, and then the checking for write cycle can be started. The steps for checking completion of write cycle are listed as below:

1) Send a dummy write operation to EEPROM, the dummy write operation includes a START condition and a slave address, shown in figure 6.

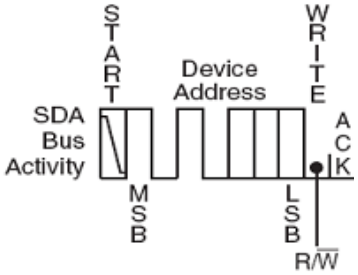


Figure 6: Dummy write operation

2) Poll ACK returned from EEPROM. If the write cycle is over, the ACK will be returned, if the write cycle is underway, the ACK will not be returned and step 1 and step 2 need to be repeated until the write cycle is over.

The referenced code can be found in chapter 10.

6. Write-protect application

GIANTEC I^2C serial EEPROM provides hardware write protection function. This function can be enabled or disabled depends WP pin, if WP is high, the hardware write protection function is enabled and the write operation on EEPROM will be ignored, if WP is low or floating, the hardware protection function is disabled and the write operation on EEPROM is valid. In general applications, WP can be tied to VCC, GND or IO pin of MCU. If the write protection function is controlled by software, it is recommended to drive WP high or low and keep WP stable before the START condition of write operation. This will help EEPROM to check whether the write protection function is enabled or not in time.

7. Data throughput

To improve the data throughput, the following solutions are recommended:

1) In order to improve the data throughput, hardware-wise, the operation frequency between MCU and EEPROM may be improved, for example, a faster MCU or a higher frequency oscillator may be chosen, software-wise, the delay between SCK transitions need be reduced, those instructions which need less machine cycles will be preferred, for example, SETB can save a machine cycle time comparing with MOV. The pull-up resistor value on SDA need also be chosen as smaller as possible to match the MCU operation speed, but this way will increase the power consumption.
2) The page write mode is recommended to write a large amount of data instead of byte write mode. The page write mode consumes less time than byte write mode, so it can improve the transmission efficiency;

3) The sequential read is recommended to read serial data instead of byte read. The sequential read consumes less time than byte read, so it can improve the transmission efficiency;
4) While an internal write cycle is underway, please consider the solution recommended in chapter 5 to check if write cycle is over. The traditional fixed delay solution always consumes more time and thus reduces the transmission efficiency.

8. Schematic of typical application

1) If there is only one EEPROM on I^2C bus, the recommended connection is shown in figure 1. If WP needs not to be controlled by MCU, the WP pin can be tied to GND (hardware protection disabled) or VCC (hardware protection enabled).
2) If there are several EEPROM devices on I^2C bus, the connection is similar to previous case. The difference is that each EEPROM need to be set an address, A0, A1 and A2 need to be configured. Shown in figure 7, the address is from 0 to 7 respectively as the order from left to right. If master device need access one device of them, A0, A1 and A2 bit in slave address need to be configured properly, then the corresponding EEPROM device can be accessed. The format of slave address is shown in figure 8. For example, if A0, A1 and A2 of some EEPROM are set as *high, high and low* respectively, the slave address should be binary format "10100110".

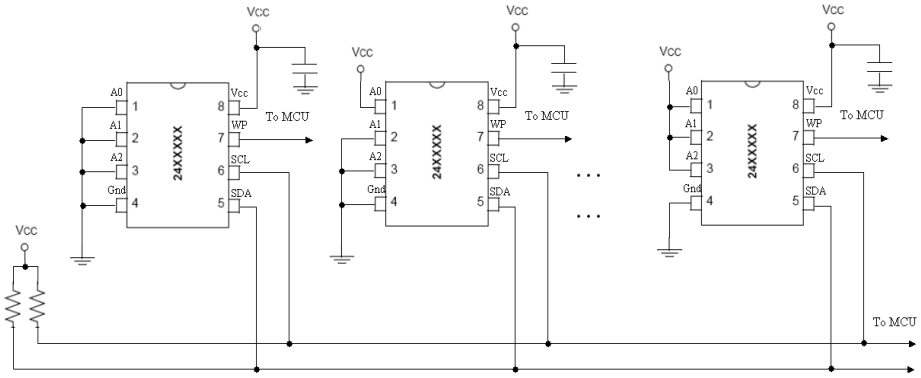


Figure 7: Several EEPROM on a bus

BIT	7	6	5	4	3	2	1	0
	1	0	1	0	A2	A1	A0	R/$\overline{W}$

Figure 8: Slave address

9. Recommendation of PCB Layout

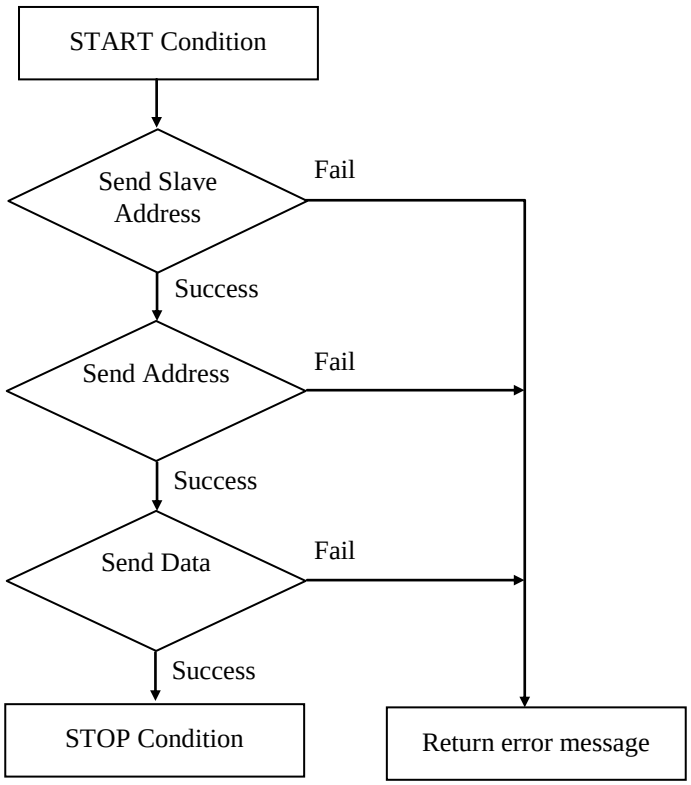
In order to reduce the crosstalk interference on I^2C bus, it is recommended to lay SDA line and SCL line in pairs. The wire length of SDA and SCL is recommended to lay as shorter as possible. The longer wire and crossed wire should be avoided. If PCB size is large enough, the GND line should be lay in the middle of these bus lines. If the length of the bus lines exceeds 10 cm, the recommended wiring pattern is listed as below:

1) Bus with VCC and GND together:

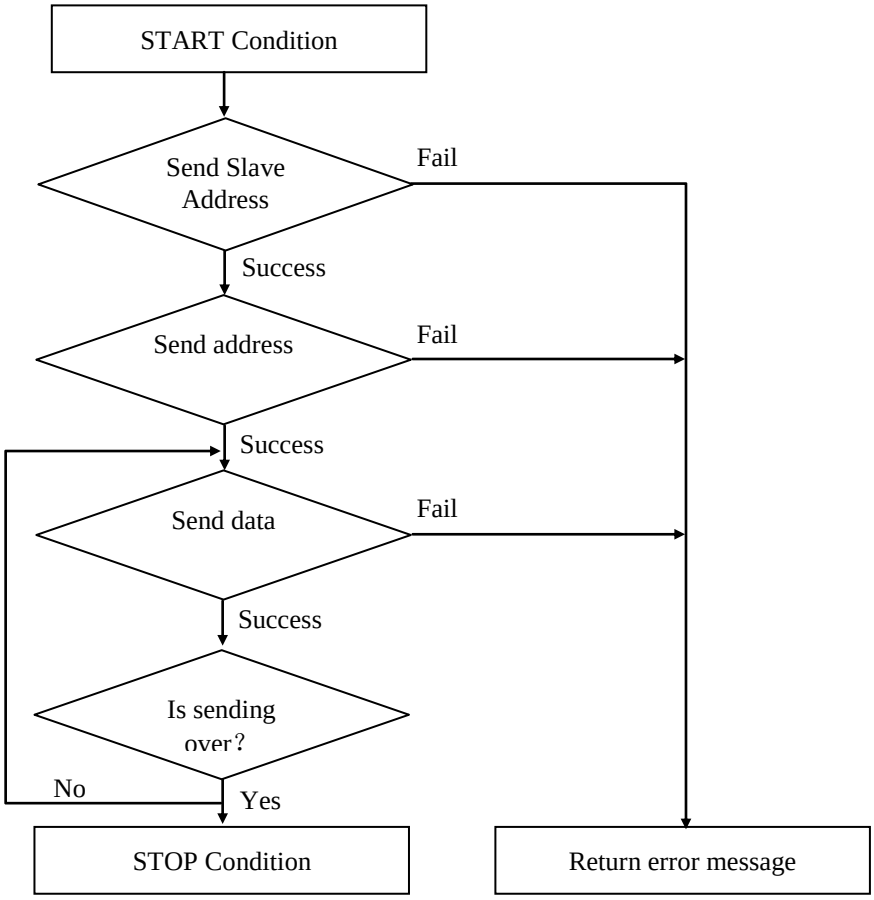
2) Bus with only the GND together:

10. Reference design of software

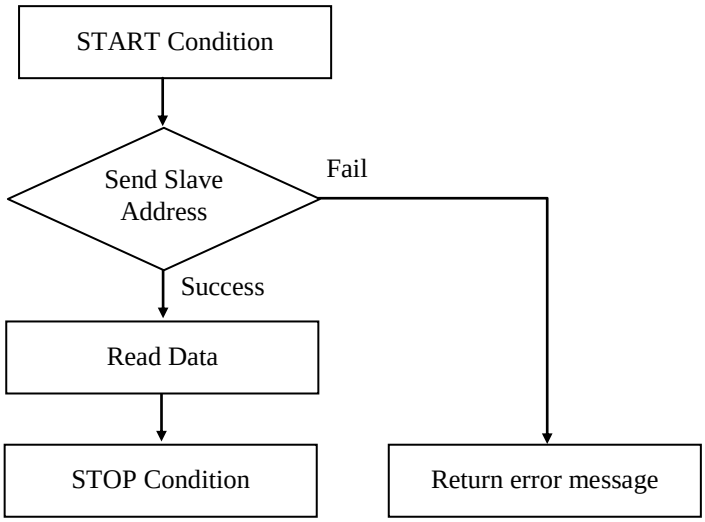
10.1 Byte write flow chart



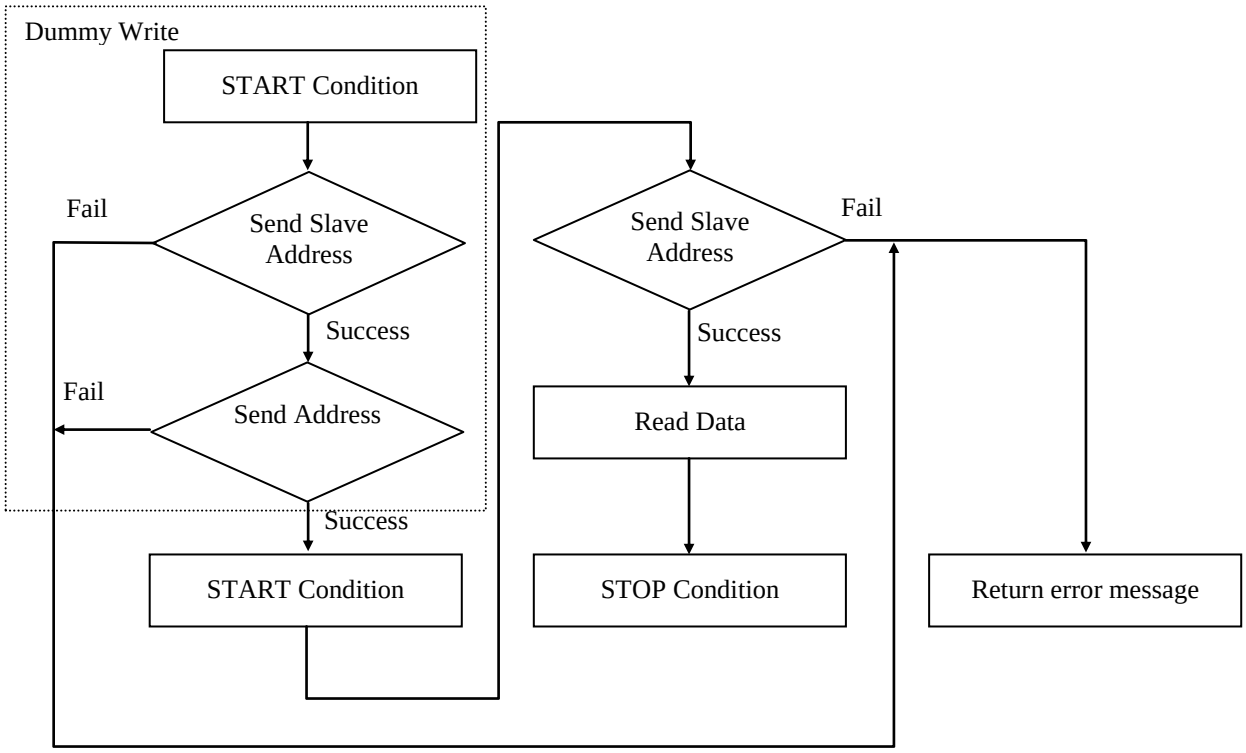
10.2 Page write flow chart



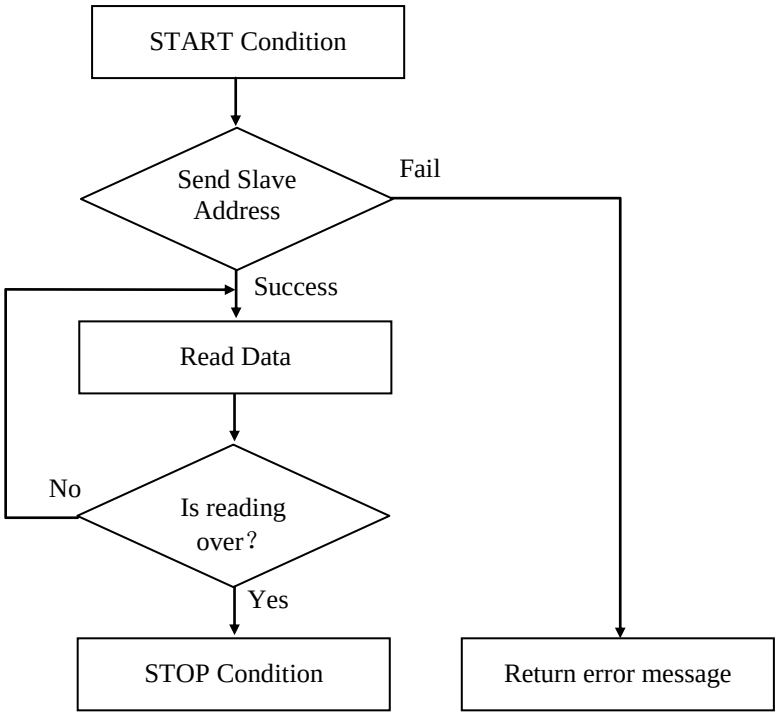
10.3 Current address read flow chart



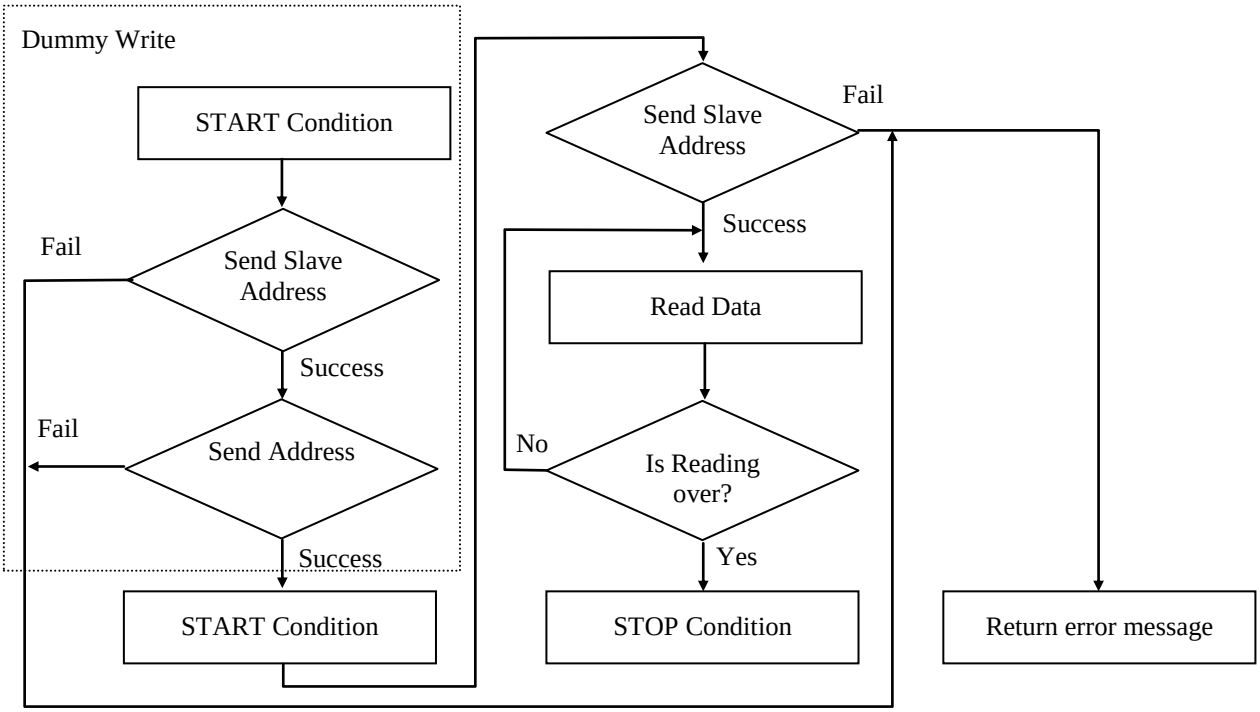
10.4 Random address read flow chart



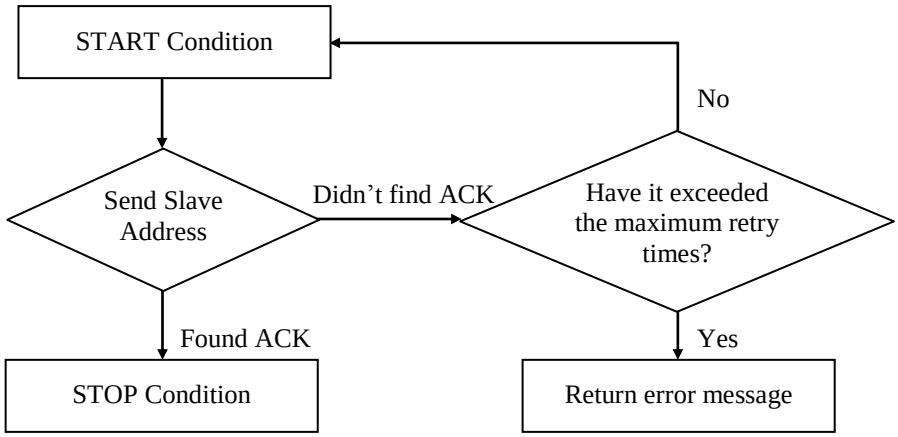
10.5 Current address sequential read flow chart



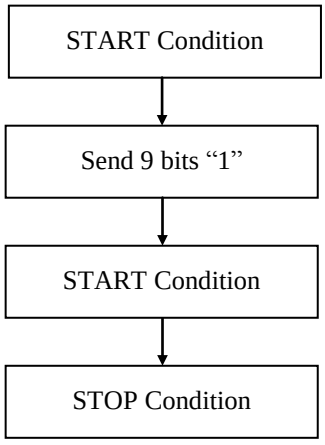
10.6 Random address sequential read flow chart



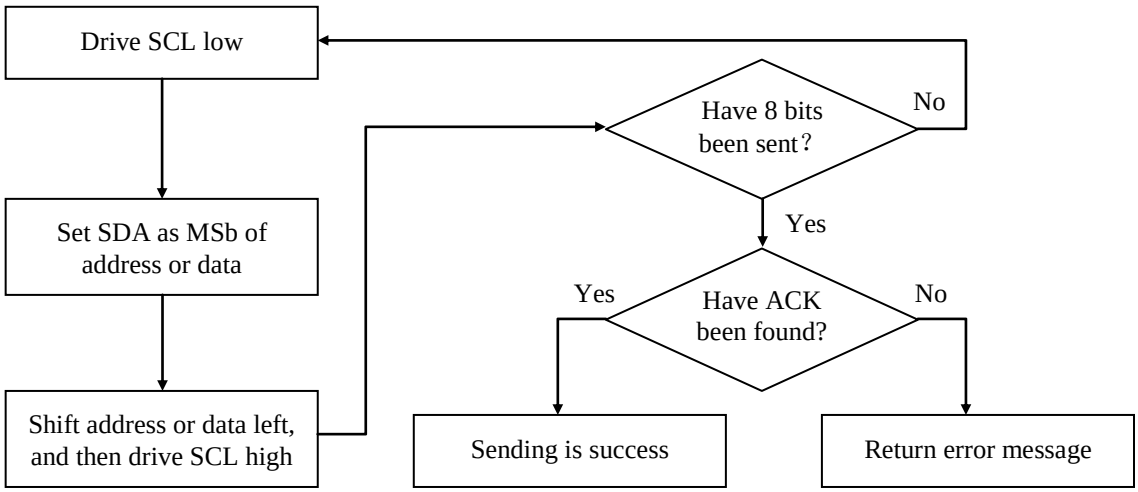
10.7 Write Cycle polling flow chart



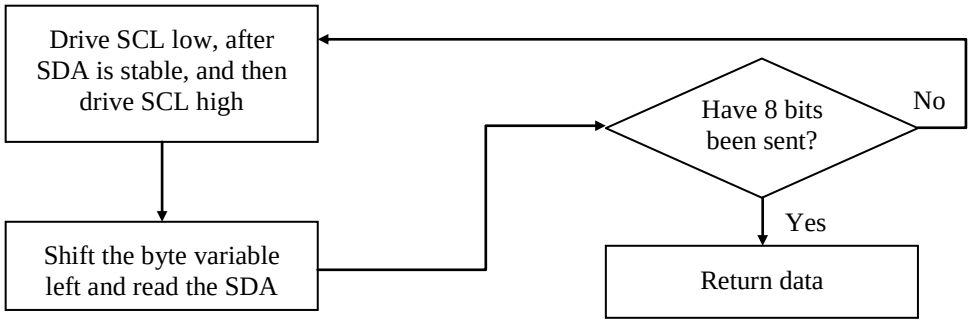
10.8 Software reset flow chart



10.9 Shift bit out flow chart



10.10 Shift bit in flow chart



10.11 Reference code

The schematic referenced by this program is shown in figure 1:

```
/**************************************************************************
24cxx.c
Description:
1.This program is based on GIANTEC I2C EEPROM 24C01 and Keil C51 7.50.
2.The highest oscillator frequency with a traditional standard 8051 MCU supported by this program is 48Mhz.
3.Main functions are listed as followed:
  1) Set status register and addresses from 0x0600-0x07FF as read-only;
  2) Write 16 bytes from buf1 into address from 0x00-0x0F by page write,
     then read these data from these address and put them into buf2.
**************************************************************************/

#include "reg51.h"
#include "intrins.h"

sbit SCL=P1^1;
sbit SDA=P1^0;
sbit WP=P1^2;

#define StartCondition SDA=1;SCL=1;SDA=0
#define StopCondition SDA=0;SCL=1;SDA=1
#define SendAck   SDA=0;SCL=1;SCL=0;SDA=1
```

```
#define PAGESIZE 8

/*****************************************************************************
Pattern description：
1.DeviceNO      Device number on I2C bus
1.Address       EEPROM internal storage address, from 0x00-0x7F in 24C01B;
2.Data          Data written into EEPROM or read from EEPROM;
3.Length        Byte number written into EEPROM or read from EEPROM;
4.Pdata         a pointer to data storage buffer;
*****************************************************************************/

//Byte Write, if success return 1, if fail return 0.
bit WriteByte(unsigned char DeviceNO,unsigned char Address,unsigned char Data);

//Random read, if success return 1, if fail return 0.
bit ReadByte_RND(unsigned char DeviceNO,unsigned char Address,unsigned char *Pdata);

//Write data bit by bit, if success return 1, if fail return 0.
bit WriteEE(unsigned char Data);

//Read data bit by bit, return byte data.
unsigned char ReadEE();

//Polling write cycle, if success return 1, if timeout return 0.
bit PollingACK(unsigned char DeviceNO);

//Page write, if success return 1, if fail return 0.
bit WritePage(unsigned char DeviceNO,unsigned char Address,unsigned char *Pdata);

//Current address read, if success return 1, if fail return 0.
bit ReadByte(unsigned char DeviceNO,unsigned char *Pdata);

//Random sequential read, if success return 1, if fail return 0.
bit Read_SEQU_RND(unsigned char DeviceNO,unsigned char Address,unsigned char *Pdata,unsigned char Length);

//Current address sequential read, if success return 1, if fail return 0.
bit Read_SEQU(unsigned char DeviceNO,unsigned char *Pdata,unsigned char Length);

//software reset
void ResetEE();

main()
{
 unsigned char a;
 unsigned char buf1[]={7,6,5,4,3,2,1,0};
 unsigned char buf2[]={0,0,0,0,0,0,0,0};

 SCL=1;
 SDA=1;
 WP=0;                                  //Disable EEPROM write protection

 if(WritePage(0,0,buf1))                //Transfer data in buf1 to page 0 in device 0
  if(PollingACK(0))
   if(WriteByte(0,8,0x55))              //Write 0x55 into address 8 in device 0
    if(PollingACK(0))
     if(ReadByte_RND(0,0,&a))           //Read data from address 0 in device 0 and put it to a
      if(ReadByte(0,&a))                //Read data from current address in device 0 and put it to a
       if(Read_SEQU(0,buf2,3))          //Transfer 3 sequential bytes from current address in device 0 to buf2
        if(Read_SEQU_RND(0,0,buf2,8))   //Transfer 8 sequential bytes from address 0 in device 0 to buf2
          if(ReadByte(0,&a));           //Transfer data in current address in device 0 to a

bit WriteByte(unsigned char DeviceNO,unsigned char Address,unsigned char Data)
```

```
{
 unsigned char SAddr_W=0xa0;
 SAddr_W|=DeviceNO<<1;                  //Translate device NO to slave address
 StartCondition;
 if(WriteEE(SAddr_W))                   //Send Slave Address
   if(WriteEE(Address))                 //Send address
     if(WriteEE(Data))                  //Send data
         {
                 StopCondition;
                 return 1;
                 }
 return 0;
 }

bit ReadByte_RND(unsigned char DeviceNO,unsigned char Address,unsigned char *Pdata)
{
 unsigned char SAddr_W=0xa0;
 unsigned char SAddr_R=0xa1;
 SAddr_W|=DeviceNO<<1;                  // Translate device NO to slave address
 SAddr_R|=DeviceNO<<1;                  // Translate device NO to slave address
 StartCondition;
 if(!WriteEE(SAddr_W))                  //Send slave address
    return 0;
 if(!WriteEE(Address))                  //Send data
    return 0;
 StartCondition;
 if(!WriteEE(SAddr_R))                  //Send slave address
    return 0;
 *Pdata=ReadEE();                       //read one byte
 StopCondition;
 return 1;
 }

bit WriteEE(unsigned char Data)
{
 unsigned char i;
 for(i=0;i<8;i++)
 {
    SCL=0;
    SDA=(bit)(Data&0x80);               //Shift data to SDA from MSb to LSb
    Data<<=1;
    SCL=1;
 }
 //Release SDA
 SCL=0;
 SDA=1;
 SCL=1;
 //Check ACK
 if(SDA)
    return 0;
 SCL=0;
 return 1;
}

unsigned char ReadEE()
{
 unsigned char i,j,rdata;
 SCL=0;
 rdata=0;
 for(i=0;i<8;i++)
 {
    SCL=1;
```

```
    //Read SDA
    if(SDA==1)
            j=1;
    else
            j=0;
    rdata=(rdata<<1)|j;
    SCL=0;
  }
   return(rdata);
 }

bit PollingACK(unsigned char DeviceNO)
{
 unsigned char SAddr_W=0xa0;
 unsigned char Polling_Num=100;
 SAddr_W|=DeviceNO<<1;                  //Translate device NO to Slave Address
 while(Polling_Num--)                   //Check the maximum retry times
 {
    StartCondition;
    if(WriteEE(SAddr_W))                //Send slave address
    {
            StopCondition;
             return 1;
    }
  }
 return 0;
 }

bit WritePage(unsigned char DeviceNO,unsigned char Address,unsigned char *Pdata)
{
  unsigned char SAddr_W=0xa0;
  unsigned char i;
  SAddr_W|=DeviceNO<<1;                 //Translate device NO to Slave Address
  StartCondition;
  if(WriteEE(SAddr_W))                  //Send Slave Address
    if(WriteEE(Address))                //Send address
            for(i=0;i<PAGESIZE;i++)     //Check page size
            {
             if(!WriteEE(*(Pdata+i)))
                    return 0;
             }
    else
            return 0;
 else
    return 0;
  StopCondition;
  return 1;
 }

bit Read_SEQU_RND(unsigned char DeviceNO,unsigned char Address,unsigned char *Pdata,unsigned char Length)
{
 unsigned char SAddr_W=0xa0;
 unsigned char SAddr_R=0xa1;
 unsigned char i;
 SAddr_W|=DeviceNO<<1;                  //Translate device NO to Slave Address
 SAddr_R|=DeviceNO<<1;                  //Translate device NO to Slave Address
 StartCondition;
 if(!WriteEE(SAddr_W))                  //Send Slave Address
    return 0;
 if(!WriteEE(Address))                  //Send address
    return 0;
 StartCondition;
```

```
 if(!WriteEE(SAddr_R))                    //Send Slave Address
    return 0;
 for(i=1;i<Length;i++)
 {
    *Pdata=ReadEE();
    Pdata++;
    SendAck;
 }
 *Pdata=ReadEE();
 StopCondition;
 return 1;
 }

bit Read_SEQU(unsigned char DeviceNO,unsigned char *Pdata,unsigned char Length)
{
 unsigned char SAddr_R=0xa1;
 unsigned char i;
 SAddr_R|=DeviceNO<<1;                    //Translate device NO to Slave Address
 StartCondition;
 if(!WriteEE(SAddr_R))                    //Send Slave Address
    return 0;
 for(i=1;i<Length;i++)
 {
    *Pdata=ReadEE();
    Pdata++;
    SendAck;
 }
 *Pdata=ReadEE();
 StopCondition;
 return 1;
 }

bit ReadByte(unsigned char DeviceNO,unsigned char *Pdata)
{
 unsigned char SAddr_R=0xa1;
 SAddr_R|=DeviceNO<<1;                    //Translate device NO to Slave Address
 StartCondition;
 WriteEE(SAddr_R);                        //Send Slave Address
 *Pdata=ReadEE();
 StopCondition;
 return 1;
 }

//software reset
void ResetEE()
{
 unsigned char i;
 StartCondition;
 //Send nine sequential bits '1'
 for(i=0;i<9;i++)
 {
    SCL=0;
    SDA=1;
    SCL=1;
  }
 StartCondition;
 StopCondition;
 }
```