1. Introduction

This application note provides assistance and guidance on how to use GIANTEC Micro wire 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
- Recommended of PCB Layout
- Reference design of software

2. Power supply & power on reset

GIANTEC 93 series EEPROM 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, if ERAL or WRALL instruction is used to operate EEPROM, VCC is required to be above 4.5V, otherwise, these two instructions may not function properly.

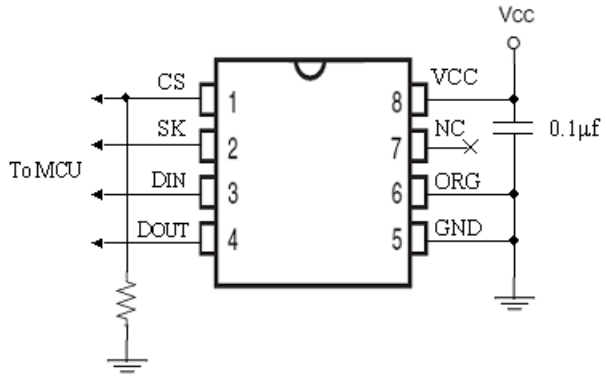

Figure 1: GIANTEC 93 series EEPROM recommended connections

2.2 Power on reset

During power ramp up, once VCC level reaches to 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-up 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.

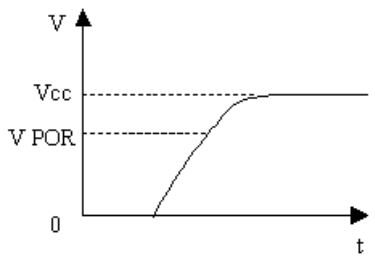

Figure 2: Power on reset

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

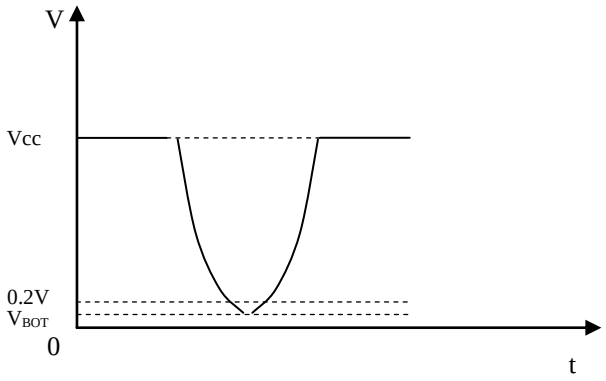

Figure 3: Power down

3. Power saving

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

3.1 Whenever no need to operate EEPROM, CS pin should be driven low. Once CS is low, the device enters into standby mode for power-saving purpose unless an internal write operation is underway. In this mode, the power consumption is minimum correspondingly;

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 16-bit memory organization can consume less time and power than 8-bit memory organization. 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

The IO configuration of GIANTEC 93 series EEPROM products needs to be considered accordingly as below:

4.1 In order to reduce the possibility of wrong operation inadvertently due to noise, it is recommended that CS pin should be tied to a proper pull-down resistor to improve the anti-jamming capability of EEPROM. The pull-down resistor should ensure MCU could drive EEPROM normally, namely, while the working voltage of EEPROM is within its operating range, MCU can provide enough current (more than 2.5μA) for each input pin of EEPROM. For example, when the P1 port of P89C52 of Philips is used to operate EEPROM, the pull-down resistor value is usually from 100K to 470K. General speaking, a larger pull-down resistor is recommended to reduce the power consumption.

4.2 GT93C46, GT93C56, GT93C66, GT93C76 and GT93C86 not only support 16-bit memory organization, but also support 8-bit memory organization. The ORG pin is used to switch the memory organization, if ORG is low, 8-bit memory organization is enabled, if ORG is high, 16-bit memory organization is enabled. Connect the ORG pin straight to Gnd or Vcc is recommended. If 8-bit memory organization is enabled, each address will store an 8 bit data. If 16-bit memory organization is enabled, each address will store a 16-bit data. For example, figure 4 shows the instruction set of GT93C46, figure 5 shows the instruction set of GT93C66 and figure 6 shows the instruction set of GT93C86.

INSTRUCTION SET

Instruction	Start Bit	OP Code	8-bit Organization (ORG = GND) Address (1)	8-bit Organization (ORG = GND) Input Data	16-bit Organization (ORG = Vcc) Address (1)	16-bit Organization (ORG = Vcc) Input Data
READ	1	10	(A_6-A_0)	—	(A_5-A_0)	—
WEN (Write Enable)	1	00	11xxxxx	—	11xxxx	—
WRITE	1	01	(A_6-A_0)	(D_7-D_0) (3)	(A_5-A_0)	(D_{15}-D_0) (2)
WRALL (Write All Registers)	1	00	01xxxxx	(D_7-D_0) (3)	01xxxx	(D_{15}-D_0) (2)
WDS (Write Disable)	1	00	00xxxxx	—	00xxxx	—
ERASE	1	11	(A_6-A_0)	—	(A_5-A_0)	—
ERAL (Erase All Registers)	1	00	10xxxxx	—	10xxxx	—

Notes:
1. x = Don't care bit.
2. If input data is not 16 bits exactly, the last 16 bits will be taken as input data.
3. If input data is not 8 bits exactly, the last 8 bits will be taken as input data.

Figure 4: GT93C46 instruction set

INSTRUCTION SET

Instruction(2)	Start Bit	OP Code	8-bit Organization (ORG = GND) Address (1)	8-bit Organization (ORG = GND) Input Data	16-bit Organization (ORG = Vcc) Address (1)	16-bit Organization (ORG = Vcc) Input Data
READ	1	10	(A_8-A_0)	—	(A_7-A_0)	—
WEN (Write Enable)	1	00	11xxxxxxx	—	11xxxxxx	—
WRITE	1	01	(A_8-A_0)	(D_7-D_0)	(A_7-A_0)	(D_{15}-D_0)
WRALL (Write All Registers)	1	00	01xxxxxxx	(D_7-D_0)	01xxxxxx	(D_{15}-D_0)
WDS (Write Disable)	1	00	00xxxxxxx	—	00xxxxxx	—
ERASE	1	11	(A_8-A_0)	—	(A_7-A_0)	—
ERAL (Erase All Registers)	1	00	10xxxxxxx	—	10xxxxxx	—

Notes:
1. x = Don't care bit.
2. If the number of bits clocked-in does not match the number corresponding to a selected command, all extra trailing bits are ignored, and WRITE, WRALL, ERASE, and ERAL are also ignored, but READ, WEN, WDS are accepted.

Figure 5: GT93C66 instruction set

INSTRUCTION SET

Instruction(2)	Start Bit	OP Code	8-bit Organization (ORG = GND) Address (1)	8-bit Organization (ORG = GND) Input Data	16-bit Organization (ORG = Vcc) Address (1)	16-bit Organization (ORG = Vcc) Input Data
READ	1	10	(A_{10}-A_0)	—	(A_9-A_0)	—
WEN (Write Enable)	1	00	11x xxxx xxxx	—	11 xxxx xxxx	—
WRITE	1	01	(A_{10}-A_0)	(D_7-D_0)	(A_9-A_0)	(D_{15}-D_0)
WRALL (Write All Registers)	1	00	01x xxxx xxxx	(D_7-D_0)	01 xxxx xxxx	(D_{15}-D_0)
WDS (Write Disable)	1	00	00x xxxx xxxx	—	00 xxxx xxxx	—
ERASE	1	11	(A_{10}-A_0)	—	(A_9-A_0)	—
ERAL (Erase All Registers)	1	00	10x xxxx xxxx	—	10 xxxx xxxx	—

Notes:
1. x = Don't care bit.
2. If the number of bits clocked-in does not match the number corresponding to a selected command, all extra trailing bits are ignored, and WRITE, WRALL, ERASE, ERAL, WEN, and WDS instructions are rejected, but READ is accepted.

Figure 6: GT93C86 instruction set

5. Check completion of Write Cycle

Once EEPROM receive a WRITE, WRALL, ERASE or ERAL instruction and recognize a transition from high to low on CS, it will enter its internal write cycle. During this cycle, EEPROM writes data to specified address. Because the write cycle time (normally less than 5ms) cannot be foreseen accurately, and the EEPROM cannot respond to any instructions during this internal cycle, it is necessary to wait till the write cycle completion to execute other instructions. Usually a fixed delay process is executed immediately after those WRITE types of instruction, this maybe a simple and convenient solution, but not the best one obviously. Because most probably the write cycle will take less time than delay process offered, that is to say most likely fixed delay process will reduce the efficiency of data transmission. Therefore, it is recommended to use polling the DOUT pin solution. In this way, the wait time will be reduced to minimum level, and the data transmission efficiency will be improved as well. The software flowchart of this solution is shown in figure 7, and the referenced code can be found in chapter 10.

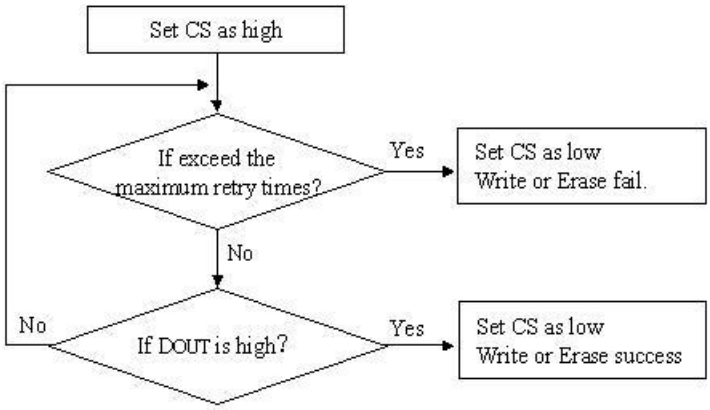

Figure 7: Checking for Write Cycle flowchart

6. Write-protect application

GIANTEC micro wire serial EEPROM provides hardware write protection function. The data in EEPROM under hardware write protection cannot be written or erased. In applications, please pay attention to the following cases:

6.1 After POR, EEPROM is under hardware write protection. So WEN instruction must be executed before send any WRITE type of instructions to device. Once receive WEN instruction, device can be written or erased.

6.2 After a WRITE or ERASE instruction is executed successfully, it is a good practice to issue WDS instruction to bring device into hardware protection mode, which will reduce the possibility of wrong operation inadvertently.

7. Data throughput

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

7.1 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 SK transitions need to be reduced, those instructions which need less machine cycles will be preferred, for example, SETB can save a machine cycle time comparing with MOV. A traditional 8051 MCU with different frequency oscillator, the maximum SK frequency that can be realized theoretically is shown in figure 8. Please be noted that the actual SK frequency should not exceed the maximum frequency supported by EEPROM. (Shown in figure 9)

Description	Traditional 8051MCU oscillator (VCC=5V)				
	1MHz	6MHz	12MHz	24MHz	48MHz
Theoretic maximum SK frequency	41.7KHz	250KHz	500KHz	1MHz	2MHz

Figure 8: The theoretically maximum SK frequency

Description					
	93C46	93C56	93C66	93C76	93C86
Maximum supported frequency	3 MHz	3 MHz	3 MHz	3 MHz	3 MHz

Figure 9: The maximum supported frequency

7.2 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. The figure 10 shows the comparison of time consumed between sequential read and byte read in 8-bit memory organization. These data is from the test with a traditional standard 8051 MCU and the program in chapter 10.

Description	Bytes Number	Tranditional standard 8051MCU oscillator (VCC=5V)		
		6MHz	12MHz	24MHz
Sequential Read	16	8.242ms	4.121ms	2.061ms
Byte Read	16	14.692ms	7.346ms	3.673ms

Figure 10: Comparison between sequential read and byte read

7.3 The 16-bit memory organization is recommended to write a large amount of data instead of 8-bit memory organization. The 16-bit memory organization consumes less time than 8-bit memory organization, thus improve the transmission efficiency as well. The figure 11 shows the comparison of time consumed between 16-bit memory organization and 8-bit memory organization. These data is collected from the test with a traditional standard 8051 MCU and the program in chapter 10.

Description	Bytes Number	Tranditional standard 8051MCU oscillator (VCC=5V)		
		6MHz	12MHz	24MHz
8-bit organization	16	38.436ms	31.202ms	27.625ms
16-bit organization	16	22.884ms	17.434ms	14.717ms

Figure 11: Comparison between 16-bit organization and 8-bit organization

7.4 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

The recommended connections are shown in figure 1.

9. Recommendation of PCB Layout

In order to reduce the crosstalk interference on Micro Wire bus, the wire length of DIN, DOUT and SK are 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.

10. Reference design of software

The schematic referenced by this program is shown in figure 12:

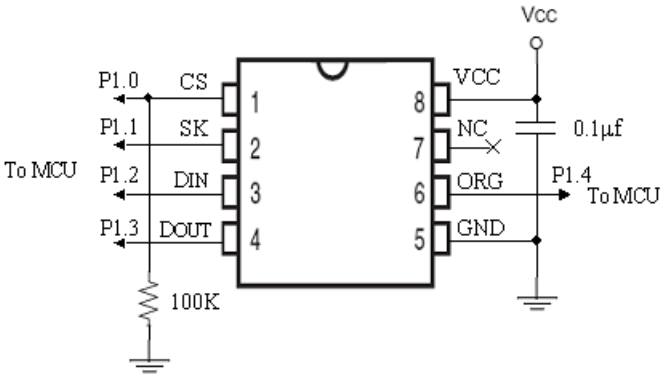

Figure 12: Reference connections

The reference code are shown as followed:

```
/*****************************************************************************
93xx.c
Description:
1.This program is based on GIANTEC Micro Wire EEPROM 93C46 and Keil C51 7.50.
        2.The highest oscillator frequency with a traditional standard 8051 MCU supported by this program is 48Mhz.
        3.This program demonstrates the different operation in 8-bit and 16-bit memory organization.
*****************************************************************************/
#include "reg51.h"

#define BYTE unsigned char
#define WORD unsigned int
#define BOOL bit
#define TRUE 1
#define FALSE 0

sbit IO_CS = P1^0;
sbit IO_SK = P1^1;
sbit IO_DIN = P1^2;
sbit IO_DOUT = P1^3;
sbit IO_ORG = P1^4;
//Define polling Write Cycle maximum retry times
#define POLLING_NUM 5000

#define ShiftToEE(Bit) IO_SK=0;IO_DIN=Bit;IO_SK=1
#define ShiftFromEE(Bit) IO_SK=0;IO_SK=1;Bit=IO_DOUT
#define DATA_BIT_LEN (IO_ORG ? 16 : 8)
#define ADDR_BIT_LEN (IO_ORG ? 6 : 7)
/*****************************************************************
Pattern description：
1.Address           EEPROM internal storage address
2.Data              Data written into EEPROM
3.Length            Byte number read from EEPROM
4.Pdata             a pointer to data storage buffer
*****************************************************************/
// Polling write cycle, if success return 1, if timeout return 0
```

```
BOOL PollingEE();

//Write operation
void WRITE(BYTE Address,WORD Data);

//Read operation
void READ(BYTE Address,BYTE *Pdata,BYTE Length);

//Write enable
void WEN();

//Write all operation
void WRALL(WORD Data);

//Write disable
void WDS();

//Erase operation
void ERASE(BYTE Address);

//Erase all operation
void ERAL();

void main()
{
    BYTE buf_byte[16];
    WORD buf_word[8];

    IO_CS=0;
    IO_SK=1;
    IO_DIN=1;
    IO_DOUT=1;

    //8-bit memory organization demo
    IO_ORG=0;
    WEN();                          //Disable EEPROM write protection
    WRALL(0x55);                    //Write 0x55 into all address of EEPROM
    PollingEE();
    WRITE(0x00,0xaa);               //Write 0xaa into address 0x00 of EEPROM
    PollingEE();
    WDS();                          //Enable EEPROM write protection
    WRITE(0x01,0x01);               //Try to write 0x01 into address 0x01
    PollingEE();
    READ(0x00,buf_byte,16);         //Read 16 sequential bytes from address 0x00
    WEN();
    ERASE(0x02);                    //Erase the data in address 0x02
    PollingEE();
    READ(0x00,buf_byte,16);
    ERAL();                         //Erase all data in EEPROM
    PollingEE();
    READ(0x00,buf_byte,16);

    //16-bit memory organization demo
    IO_ORG=1;
    WEN();                          // Disable EEPROM write protection
    WRALL(0x55AA);                  // Write 0x55AA into all address of EEPROM
    PollingEE();
    WRITE(0x00,0x3366);             //Write 0x3366 into address 0x00 of EEPROM
    PollingEE();
    WDS();                          // Enable EEPROM write protection
    WRITE(0x01,0x4477);             // Try to write 0x4477 into address 0x01
    PollingEE();
    READ(0x00,(BYTE *)buf_word,8);  // Read 8 sequential words from address 0x00
```

```
    WEN();
    ERASE(0x02);                        // Erase the data in address 0x02
    PollingEE();
    READ(0x00,(BYTE *)buf_word,8);
    ERAL();                             // Erase all data in EEPROM
    PollingEE();
    READ(0x00,(BYTE *)buf_word,8);

    while(1);
}

//Write operation
void WRITE(BYTE Address,WORD Data)
{
    BYTE i;
  bit temp_bit;
  WORD temp_word;

    IO_CS=1;
    //Send Start bit
    ShiftToEE(1);
    //Send WRITE Opcode
    ShiftToEE(0);
    ShiftToEE(1);

    Address<<=8-ADDR_BIT_LEN;

    for(i=0;i<ADDR_BIT_LEN;i++)
    {
            temp_bit=Address&0x80;
            ShiftToEE(temp_bit);
            Address<<=1;
    }

    temp_word=Data;
    if(!IO_ORG)
            temp_word<<=8;
    for(i=0;i<DATA_BIT_LEN;i++)
    {
            temp_bit=temp_word&0x8000;
            ShiftToEE(temp_bit);
            temp_word<<=1;
    }

    IO_CS=0;
}

//Read operation
void READ(BYTE Address,BYTE *Pdata,BYTE Length)
{
    BYTE i;
  bit temp_bit;
  WORD temp_word;

    IO_CS=1;
    //Send Start bit
    ShiftToEE(1);
    //Send READ Opcode
    ShiftToEE(1);
    ShiftToEE(0);

    Address<<=8-ADDR_BIT_LEN;
```

```
    for(i=0;i<ADDR_BIT_LEN;i++)
    {
            temp_bit=Address&0x80;
            ShiftToEE(temp_bit);
            Address<<=1;
    }
    //Receive data
    while(Length)
    {
            temp_word=0;

            for(i=0;i<DATA_BIT_LEN;i++)
            {
                    temp_word<<=1;
                    ShiftFromEE(temp_bit);
            temp_word|=temp_bit;
            }
            if(IO_ORG)
            {
            *(WORD *)Pdata=temp_word;
            Pdata+=2;
            }
            else
            {
                    *Pdata=temp_word;
                    Pdata++;
            }
            Length--;
    }

    IO_CS=0;
}

//Write enable
void WEN()
{
    BYTE temp_byte;
    BYTE i;
    bit temp_bit;

    IO_CS=1;
    //Send Start bit
    ShiftToEE(1);
    //Send Opcode
    ShiftToEE(0);
    ShiftToEE(0);

    temp_byte=0xc0;
    for(i=0;i<ADDR_BIT_LEN;i++)
    {
            temp_bit=temp_byte&0x80;
            ShiftToEE(temp_bit);
            temp_byte<<=1;
    }

    IO_CS=0;
}


// Polling write cycle, if success return 1, if timeout return 0
BOOL PollingEE()
{
    WORD i;
```

```
    IO_CS=1;
    for(i=0;i<POLLING_NUM;i++)
    {
        if(IO_DOUT)
        {
            IO_CS=0;
            return TRUE;
        }
    }
    IO_CS=0;
    return FALSE;
}

//Write all
void WRALL(WORD Data)
{
    BYTE i;
   bit temp_bit;
   WORD temp_word;

    IO_CS=1;
    //Send Start bit
    ShiftToEE(1);
    //Send Opcode
    ShiftToEE(0);
    ShiftToEE(0);

    temp_word=0x4000;

    for(i=0;i<ADDR_BIT_LEN;i++)
    {
        temp_bit=temp_word&0x8000;
        ShiftToEE(temp_bit);
        temp_word<<=1;
    }
    //Send Data
    temp_word=Data;
    if(!IO_ORG)
        temp_word<<=8;

    for(i=0;i<DATA_BIT_LEN;i++)
    {
        temp_bit=temp_word&0x8000;
        ShiftToEE(temp_bit);
        temp_word<<=1;
    }

    IO_CS=0;
}

//Write disable
void WDS()
{
    BYTE temp_byte;
    BYTE i;
    bit temp_bit;

    IO_CS=1;
    //Send Start bit
    ShiftToEE(1);
    //Send Opcode
    ShiftToEE(0);
    ShiftToEE(0);
```

```
    temp_byte=0x00;
    for(i=0;i<ADDR_BIT_LEN;i++)
    {
            temp_bit=temp_byte&0x80;
            ShiftToEE(temp_bit);
            temp_byte<<=1;
    }

    IO_CS=0;
}

//Erase
void ERASE(BYTE Address)
{
    BYTE i;
   bit temp_bit;

    IO_CS=1;
    //Send Start bit
    ShiftToEE(1);
    //Send Opcode
    ShiftToEE(1);
    ShiftToEE(1);

    Address<<=8-ADDR_BIT_LEN;

    for(i=0;i<ADDR_BIT_LEN;i++)
    {
            temp_bit=Address&0x80;
            ShiftToEE(temp_bit);
            Address<<=1;
    }

    IO_CS=0;
}

//Erase all
void ERAL()
{
    BYTE temp_byte;
    BYTE i;
    bit temp_bit;

    IO_CS=1;
    //Send Start bit
    ShiftToEE(1);
    //Send Opcode
    ShiftToEE(0);
    ShiftToEE(0);

    temp_byte=0x80;
    for(i=0;i<ADDR_BIT_LEN;i++)
    {
            temp_bit=temp_byte&0x80;
            ShiftToEE(temp_bit);
            temp_byte<<=1;
    }

    IO_CS=0;
}
```