1. Introduction

This application note provides assistance and guidance on how to use GIANTEC SPI 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 25 series EE product works 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.

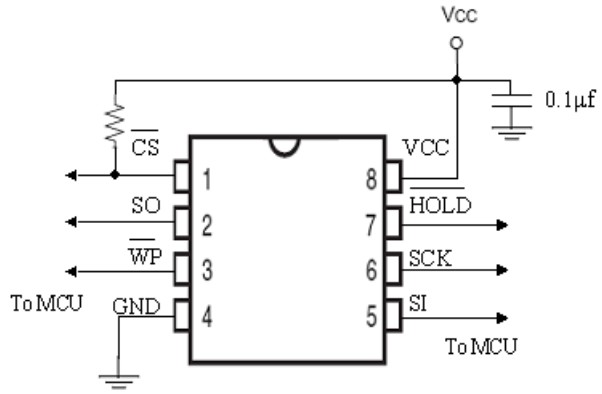

Figure 1: GIANTEC 25 series EEPROM recommended connections

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.

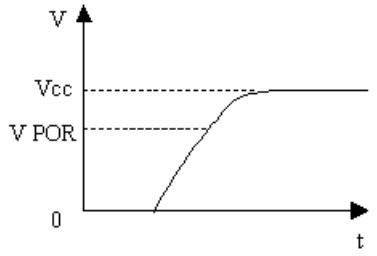

Figure 2: Power on reset

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

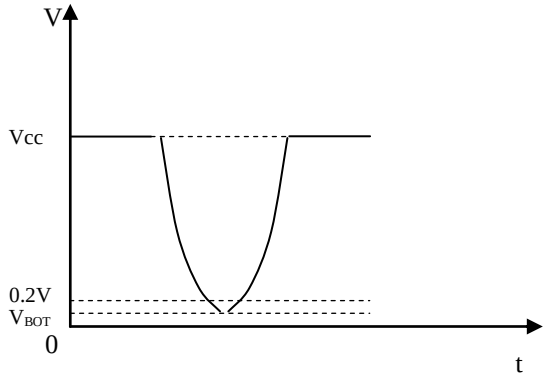

Figure 3: Power Down

3. Power saving

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

3.1 Whenever no need to operate EEPROM, the CS pin should be driven high. Once CS is high, the device will enter 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 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 all input and output pins should be tied to a proper pull-up resistor to improve the anti-jamming capability of EEPROM. General speaking, a larger pull-up resistor is recommended to reduce the power consumption. In general applications, the pull-up resistor value is usually from 4.7K to 10K.

5. Check completion of Write Cycle

Once EEPROM receives a WRITE instruction or WDSR instruction and recognize a transition from low to high on CS, it will enter its internal write cycle. During this cycle, EEPROM write data to specified address. Because the write cycle time (normally less than 5ms) cannot be foreseen accurately, and EEPROM only respond to RDSR instruction during this cycle, so it is necessary to wait till the write cycle completion to execute other instructions. Usually a fixed delay process is executed immediately after the WRITE instruction, maybe this is 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 the solution of polling the RDY bit in status register. Since EEPROM only respond to RDSR instruction while write cycle is in progress, and the RDY bit in status register indicates if EEPROM is in write cycle (if RDY is “1”, EEPROM is busy, if RDY is “0”, EEPROM is idle) or not, thus the status register should be read continuously for polling the RDY bit till the write cycle completion, after that other instructions can be sent. In this way, the wait time for write operation 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 4, and the referenced code can be found in chapter 10.

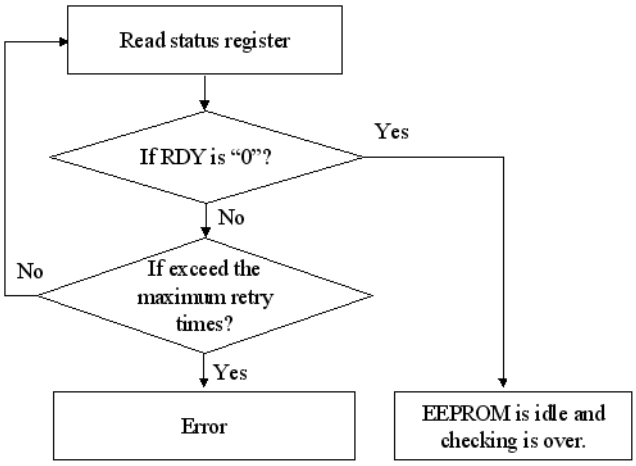

Figure 4: Checking for Write Cycle flowchart

6. Write-protect application

GIANTEC SPI serial EEPROM provides hardware write protection function. However, there are two different systems of write protection to be used in different products. The followed portion will describe the difference of these two systems.

6.1 GT25C01, GT25C02 and GT25C04

In these products' applications, write protection effect depends on the configuration of WP pin, WEN, BP0 and BP1 bit in status register. (Shown in figure 5)

WP	Hardware Write Protection	WEN	Inside Block	Outside Block	Status Register
0	Enabled	X	Read-only	Read-only	Read-only
1	Not Enabled	0	Read-only	Read-only	Read-only
1	Not Enabled	1	Read-only	Unprotected	Unprotected

Note: X = Don't care bit.

Figure 5: Write protection

Following items intends to help to better understand the write protection in real application:

6.1.1 Setting WP pin to low can protect all data storage area and status register. If WP is low, no matter what state the WEN, BP0 and BP1 are, all data storage area and status register will be read-only. In general applications, based on different requirements, WP can be tied to VCC (disable write protection), GND (set EEPROM as read-only) or I/O port of MCU (MCU can configure EEPROM on demand).

6.1.2 Setting WEN bit in status register can protect all data storage area and status register. If WEN is "0", no matter what state WP, BP0 and BP1 are, all data storage area and status register will be set as read-only. In addition, RDY bit and WEN bit cannot be set or cleared by writing status register. While write cycle is in progress, RDY bit will be set automatically. Once write cycle is over, RDY bit will be cleared automatically. The only way to set WEN is to send WREN instruction to EEPROM, but the ways of clearing WEN are listed as below:

1) After POR(Power on reset), WEN is reset as "0";
2) After a WRDI instruction, WEN is cleared as "0";
3) After a WRSR instruction, WEN is cleared as "0";
4) After a Write instruction, WEN is cleared as "0".

6.1.3 Setting BP0 or BP1 bit in status register can protect the specified data storage area (Shown in figure 6). The area protected by BP0 and BP1 is always read-only despite the WP and WEN. BP0 and BP1 can be modified by WRSR instruction. If a WRSR instruction is sent to modify

BP0 and BP1, except BP0, BP1 and WPEN, the values of other data bits incorporated into WRSR can be "0" or "1", and are not stored in the Status Register.

Level	Status Register Bits		Array Addressed Protected		
	BP1	BP0	GT25C01	GT25C02	GT25C04
0	0	0	None	None	None
1(1/4)	0	1	60h-7Fh	C0h-FFh	180h-1FFh
2(1/2)	1	0	40h-7Fh	80h-FFh	100h-1FFh
3(All)	1	1	00h-7Fh	00h-FFh	000h-1FFh

Figure 6: Write protection for specified area

6.1.4 Three methods mentioned above have an effect like logic "or". No matter what mode is effective, the protection range of whichever mode is not affected by other modes. In general applications, adopting these solutions can improve the security level of data. The recommended steps for setting write protection are listed as below:

1) If WP is low and configurable, set WP as high;
2) Send WREN instruction to EEPROM, then status register can be written;
3) Send WRSR instruction to EEPROM as figure 5 and write proper value into BP0 and BP1 in status register;
4) Check RDY bit with the flowchart in chapter 5 till EEPROM is in idle state;
5) WEN has been cleared after foregoing steps. If WP is configurable, it is recommended to set WP as low, then write protection configuration is completed.

The steps of writing data into specified protected area are listed as below:

1) If WP is low and configurable, set WP as high;
2) Send WREN instruction to EEPROM to set WEN as "1";
3) Check RDY bit with the flowchart in chapter 5 till EEPROM is in idle state;
4) Send WRSR instruction to EEPROM as figure 5 and write proper value into BP0 and BP1 in status register;
5) Check RDY bit with the flowchart in chapter 5 till EEPROM is in idle state;
6) Send WREN instruction to EEPROM to set WEN as "1";
7) Check RDY bit with the flowchart in chapter 5 till EEPROM is in idle state;
8) Send WRITE instruction to EEPROM and write data into specified area;
9) Check RDY bit with the flowchart in chapter 5 till EEPROM is in idle state;
10) Repeat steps 6 to steps 9 till all data have been written into EEPROM;
11) If WP is configurable, set WP as low.

6.2 GT25C08, GT25C16, GT25C32, GT25C64, GT25C128 and GT25C256

In these products' applications, write protection effect depends on the configuration of WP pin, WPEN bit, WEN bit, BP0 and BP1 bit in status register. (Shown in figure 7)

WPEN	$\overline{WP}$	Hardware Write Protection	WEN	Inside Block	Outside Block	Status Register (WPEN, BP1, BP0)
0	X	Not Enabled	0	Read-only	Read-only	Read-only
0	X	Not Enabled	1	Read-only	Unprotected	Unprotected
1	0	Enabled	0	Read-only	Read-only	Read-only
1	0	Enabled	1	Read-only	Unprotected	Read-only
X	1	Not Enabled	0	Read-only	Read-only	Read-only
X	1	Not Enabled	1	Read-only	Unprotected	Unprotected

Note: X = Don't care bit.

Figure 7: Write protection

Following items intends to help to better understand the write protection in real application:

6.2.1 Write protection for status register

The write protection for status register and hardware write protection is related to WEN bit. If hardware write protection is enabled, status register will be read-only. Hardware write protection is enabled or disabled by WPEN and WP. If WPEN is "1" and WP is low, hardware write protection is enabled, otherwise hardware protection is disabled. If hardware write protection is disabled, the write protection for status register is related to WEN bit. If WEN is "0", status register is read-only. If WEN is "1", status register can be written. WPEN and WP are only related to status register and are not related to data storage area. If status register is under protection, WPEN, BP0 and BP1 will be read-only. RDY and WEN is not affected by the write protection for status register. While write cycle is in progress, RDY will be set as "1" automatically. Once write cycle is over, RDY will be clear automatically. The way to set WEN as "1" is only to send WREN instruction to EEPROM, but there are four ways to clear WEN as "0":

1) After POR, WEN is reset as "0";
2) After WRDI instruction, WEN is cleared as "0";
3) After WRSR instruction, WEN is cleared as "0";
4) After WRITE instruction, WEN is cleared as "0".

The recommended flows for status register write protection are listed as below:

1) Send RDSR instruction to read status register. If WPEN is 1, go to step 6, otherwise go to next step.
2) Send WREN instruction to set WEN as "1";
3) Polling RDY bit in status register till write cycle is over.
4) Send WRSR instruction to set WPEN as "1";
5) Polling RDY bit till write cycle is over;
6) If WP is not low, set WP as low.

The recommended flows to write data into status register under write protection are listed as below:

1) Set WP as high;
2) Send WREN instruction to set WEN as "1";
3) Polling RDY bit till write cycle is over;
4) Send WRSR instruction to write data into status register;
5) Polling RDY bit till write cycle is over;
6) Set WP as low;

6.2.2 Write protection for data storage area

Write protection for data storage area is related to WEN, BP0 and BP1. If WEN is "0", all data storage area is read-only. If WEN is "1", BP0 and BP1 go into effect. The addresses inside write protection area brought by BP0 and BP1 are read-only. The addresses outside write protection area are written or read freely. Shown in figure 8.

Level	Status Register Bits		Array Addressed Protected					
	BP1	BP0	GT25C08	GT25C16	GT25C32	GT25C64	GT25C128	GT25C256
0	0	0	None	None	None	None	None	None
1(1/4)	0	1	0300h-03FFh	0600h-07FFh	0C00h-0FFFh	1800h-1FFFh	3000h-3FFFh	6000h-7FFFh
2(1/2)	1	0	0200h-03FFh	0400h-07FFh	0800h-0FFFh	1000h-1FFFh	2000h-3FFFh	4000h-7FFFh
3(All)	1	1	0000h-03FFh	0000h-07FFh	0000h-0FFFh	0000h-1FFFh	0000h-3FFFh	0000h-7FFFh

Figure 8: Write protection for specified area

The recommended flows to enable write protection for specified area are listed as below:

1) Set WP as high to disable hardware write protection;
2) Send WREN instruction to set WEN as "1";
3) Polling RDY bit till write cycle is over;
4) Send WRSR instruction to write proper data into BP0 and BP1 based on figure 7;
5) Polling RDY bit till write cycle is over;
6) Set WP as low.

The recommended flows to write data into write protection area are listed as below:

1) Set WP as high to disable hardware write protection;
2) Send WREN instruction to set WEN as "1";
3) Polling RDY bit till write cycle is over;
4) Send WRSR instruction to write proper value into BP0 and BP1. This value is based on figure 7, which will disable the write protection for specified area;
5) Polling RDY bit till write cycle is over;
6) Send WREN instruction to set WEN as "1";
7) Polling RDY bit till write cycle is over;
8) Send WRITE instruction to write data into EEPROM;
9) Polling RDY bit till write cycle is over;
10) Repeat steps 6 to steps 9 till all data have been written into EEPROM;
11) Set WP as low.

The referenced code can be found in chapter 10.

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 SCK 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 SCK frequency that can be realized theoretically is shown in figure 9. Please be noted that the actual SCK frequency should not exceed the maximum frequency supported by EEPROM. (Shown in figure 10)

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

Figure 9: The theoretically maximum SCK frequency

Description	GIANTEC 25series EEPROM								
	25C01	25C02	25C04	25C08	25C16	25C32	25C64	25C128	25C256
Maximum supported frequency	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz	10 MHz	2.1 MHz	2.1 MHz

Figure 10: The maximum supported frequency

7.2. 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. The figure 11 shows the comparison of time consumed between page write mode and byte write mode. 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)				
		1MHz	6MHz	12MHz	24MHz	48MHz
Page Write Mode	16	40.152ms	9.122ms	6.022ms	4.589ms	3.814ms
Byte Write Mode	16	234.492ms	77.962ms	62.343ms	56.430ms	52.824ms

Figure 11: Comparison between page write mode and byte write mode

7.3. 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. The figure 12 shows the comparison of time consumed between sequential read and byte read. 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)				
		1MHz	6MHz	12MHz	24MHz	48MHz
Sequential Read	16	29.844ms	4.974ms	2.487ms	1.244ms	0.622ms
Byte Read	16	72.996ms	12.166ms	6.083ms	3.042ms	1.521ms

Figure 12: Comparison between sequential read and byte read

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 of GIANTEC 25 series EEPROM are shown in figure 1. If WP need not to be controlled by MCU, the WP pin can be tied to GND (hardware protection enabled) or to VCC (hardware protection disabled). If HOLD need not be controlled by MCU, the HOLD pin can be tied to VCC, then the HOLD function will be disabled.

9. Recommendation of PCB Layout

In order to reduce the crosstalk interference on SPI bus, the wire length of SO, SI and SCK are recommended to lay as short 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 are shown in figure 1:

```
#include "reg51.h"

/*****************************************************************************
25C16.c
Description:
1.This program is based on GIANTEC SPI EEPROM 25C16 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.
```

```
*****************************************************************************/

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

sbit IO_CS=P1^0;
sbit IO_SCK=P1^1;
sbit IO_SI=P1^2;
sbit IO_SO=P1^3;
sbit IO_HOLD=P1^4;
sbit IO_WP=P1^5;

//Define op-code
#define CMD_WREN 0x06
#define CMD_WRDI 0x04
#define CMD_RDSR 0x05
#define CMD_WRSR 0x01
#define CMD_READ 0x03
#define CMD_WRITE 0x02

//Define maximum Write Cycle polling times
#define POLLING_NUM 100

/*********************************************************************
Pattern description：
1.Address    EEPROM internal storage address, from 0x0000-0x07FF in 25C16;
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;
*********************************************************************/

//Write data bit by bit
void WriteEE(BYTE Data);

//Read data bit by bit
BYTE ReadEE();

//Polling write cycle, if success, return 1, if time out, return 0
BOOL PollingEE();

//Write operation
void Write(WORD Address,BYTE *Pdata,BYTE Length);

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

//Set WEN as 1
void SetWEN();

//Set WEN as 0
void ClrWEN();

//read status register
BYTE ReadSR();

//Write status register
void WriteSR(BYTE Data);

BYTE buf1[]={0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F};
BYTE buf2[16];
```

```
void main()
{
    // Initialize IO
    IO_CS=1;
    IO_SI=1;
    IO_SO=1;
    IO_SCK=1;
    IO_WP=1;
    IO_HOLD=1;

    //Set status register and address from 0x0600 to 0x07ff as read-only
    SetWEN();                                   //set WEN as 1
    PollingEE();                                //Polling  write cycle
    WriteSR(0x84);                              //Set WPEN as 1, set BP1 as 0, set BP0 as 1
    PollingEE();
    IO_WP=0;

    //Transfer data between MCU and EEPROM by page write
    SetWEN();
    PollingEE();
    Write(0,buf1,sizeof(buf1));                 //Transfer data from buf1 to EEPROM
    PollingEE();
    Read(0,buf2,sizeof(buf2));                  //Transfer data from EEPROM to buf2

    while(1);
}

// Write data bit by bit
void WriteEE(BYTE Data)
{
    BYTE i;

    //Shift bit of Data from MSb to LSb
    for(i=0;i<8;i++)
    {
            IO_SCK=0;
            IO_SI=(bit)(Data&0x80);
            Data<<=1;
            IO_SCK=1;
    }

    //Release SI
    IO_SI=1;
}

//Read data bit by bit
BYTE ReadEE()
{
    BYTE i;
    BYTE SO_Temp;
    BYTE Data_In=0;

    IO_SCK=0;

  //Receive data bit by bit
    for(i=0;i<8;i++)
    {
            IO_SCK=1;

            if(IO_SO==1)

                    SO_Temp=1;
```

```
            else
                SO_Temp=0;

            Data_In=(Data_In<<1)|SO_Temp;
            IO_SCK=0;
    }

    return Data_In;
}

// Polling write cycle, if success, return 1, if time out, return 0
BOOL PollingEE()
{
    BYTE i=POLLING_NUM;
    BYTE Status_Reg;

    while(i--)                          // Check the maximum Write Cycle polling times
    {
            Status_Reg=ReadSR();        //Read EEPROM status register
            Status_Reg&=0x01;

            if(!Status_Reg)             //Polling RDY bit in status register

                return TRUE;
    }

    return FALSE;
}

//Set WEN as 1
void SetWEN()
{
    IO_CS=0;
    WriteEE(CMD_WREN);
    IO_CS=1;
}

//Set WEN as 0
void ClrWEN()
{
    IO_CS=0;
    WriteEE(CMD_WRDI);
    IO_CS=1;
}

//Read status register
BYTE ReadSR()
{
    BYTE Status_Reg;
    IO_CS=0;
    WriteEE(CMD_RDSR);
    Status_Reg=ReadEE();
    IO_CS=1;
    return Status_Reg;
}

//Write status register
void WriteSR(BYTE Data)
{
    IO_CS=0;
    WriteEE(CMD_WRSR);
    WriteEE(Data);
    IO_CS=1;
```

```
}

//Write operation
void Write(WORD Address,BYTE *Pdata,BYTE Length)
{
    IO_CS=0;
    WriteEE(CMD_WRITE);                    //Send write instruction
    WriteEE(*(BYTE *)&Address);            //Send MSB of address
    WriteEE((BYTE)Address);                //Send LSB of address
    while(Length--)
    {
        WriteEE(*Pdata);                   //Send data
        Pdata++;
    }

    IO_CS=1;
}

//Read operation
void Read(WORD Address,BYTE *Pdata,BYTE Length)
{
    IO_CS=0;
    WriteEE(CMD_READ);                     //Send read instruction
    WriteEE(*(BYTE *)&Address);            //Send MSB of address
    WriteEE((BYTE)Address);                //Send LSB of address
    while(Length--)
    {
        *Pdata=ReadEE();                   //Read data
        Pdata++;
    }

    IO_CS=1;
}
```