The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN205378

FR, MB91460, SPI Communication to/from Serial EEPROM (For NM93CS46)

This application note describes how to communicate via SPI using the MB91467-USART with a serial EEPROM. In this note a NM93CS46 EEPROM from National Semiconductor is used. The described source codes were written for understanding not for code size or speed. Neither interrupts nor timers were used. Time critical coding is always done by simple flag polling or wait loop.

Contents

1 Introduction....1
2 NM93CS46....1
2.1 EEPROM....1
2.2 Connection to MB91460....2
2.3 Communication Timing....3
2.4 Used EEPROM Commands....4
2.5 Example Code....5
3 Related Documents....15
4 Additional Information....15
Document History....16

1 Introduction

This application note describes how to communicate via SPI using the MB91467-USART with a serial EEPROM. In this note a NM93CS46 EEPROM from National Semiconductor is used.

Please note that this document only gives a rough overview about the communication. The described source codes were written for understanding not for code size or speed. Neither interrupts nor timers were used. Time critical coding is always done by simple flag polling or wait loop.

2 NM93CS46

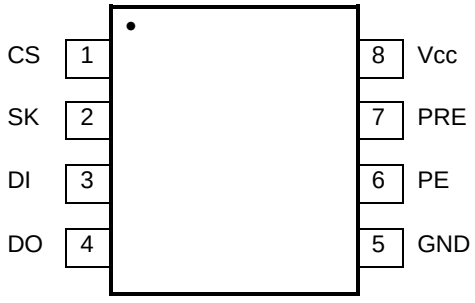
This Chapter Describes How to Communicate with The NM93CS46 EEPROM

2.1 EEPROM

The NM93CS46 serial EEPROM from National Semiconductor has 1024-Bit memory size, organized as 64 16-Bit-Words.

The NM93CS46 has the following pin-out:

Figure 1. NM93CS46 Pinout



Pin names:

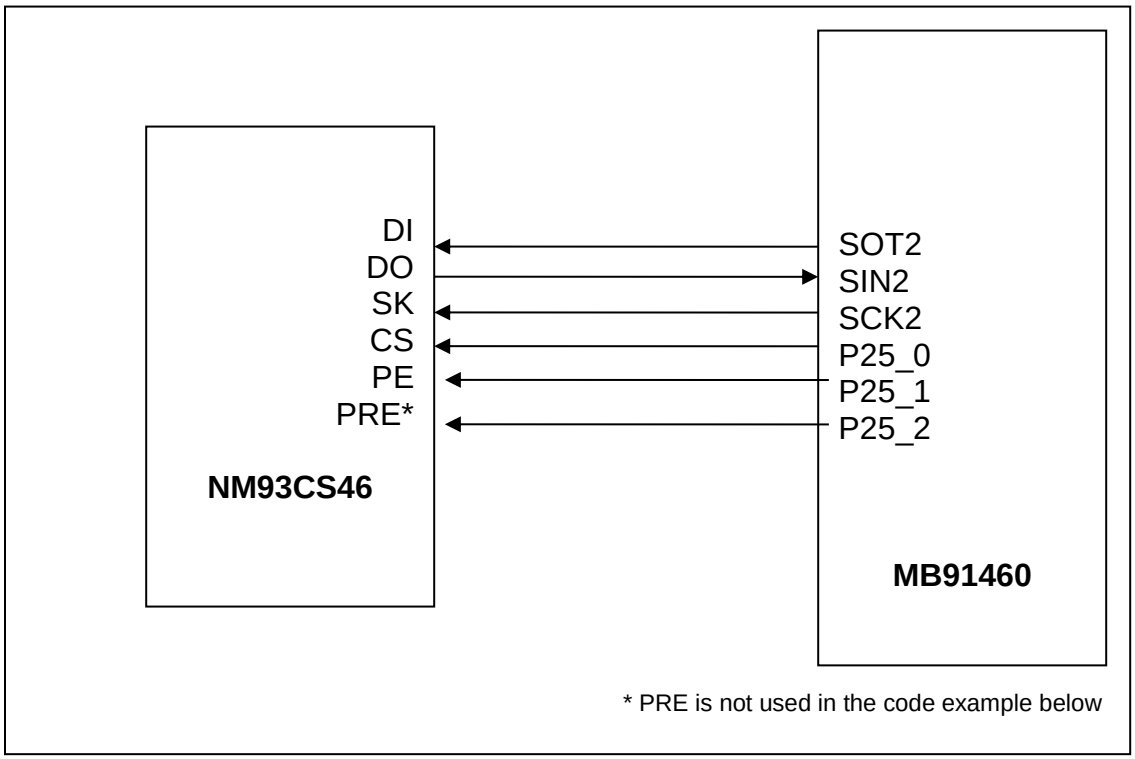
Table 1. NM93CS46 Pin Description

CS	Chip Select
SK	Serial Data Clock
DI	Serial Data Input
DO	Serial Data Output
GND	Ground
PE	Program Enable
PRE*	Protect Register Enable
Vcc	Power Supply (+ 5 volts)

2.2 Connection to MB91460

The EEPROM can be connected as shown in the following schematic. Please note that no power supply pins and other MCU-Pins are drawn than those for the connection to the EEPROM.

Figure 2. Connections between NM93CS46 and MB91460



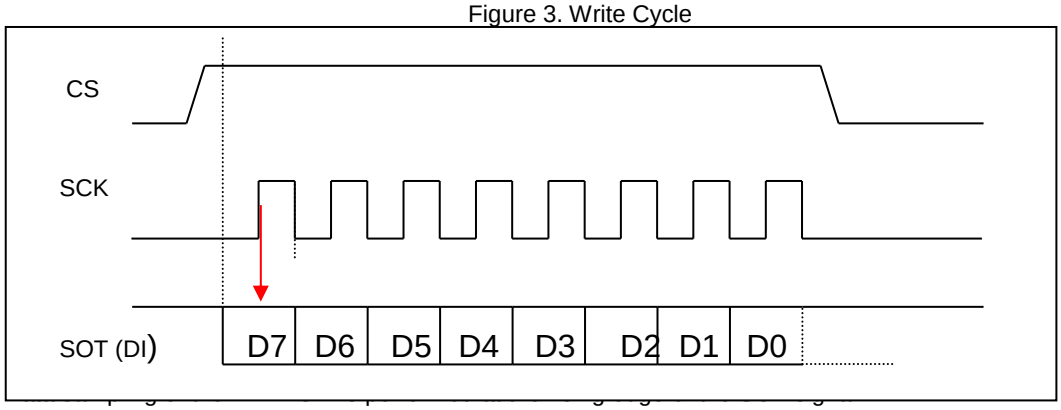
2.3 Communication Timing

Detailed description of the timing and timing parameters can be found in the corresponding datasheet of the NM93CS46 EEPROM.

2.3.1 Write Cycle Timing of NM93CS46

A write cycle (write to EEPROM) has the following bit timing (ESCR:SCES = 1, ECCR:SCDE = 1

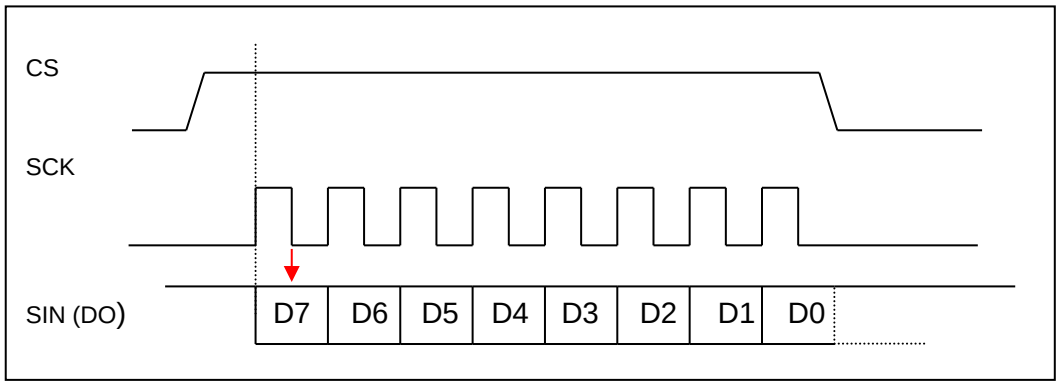
Figure 3. Write Cycle



2.3.2 Read Cycle Timing of NM93CS46

Unfortunately, the read cycle has a different timing. The NM93CS46-EEPROM does not assert the first data bit after CS goes “1”, but on the rising edge of the first serial clock:

Figure 4. Read Cycle



The red arrow denotes the sampling time of the USART, if the clock delay by a half cycle is disabled (ECCR:SCDE = 0).

The workaround, which is presented here, is to disable the clock delay (ECCR:SCDE = 0) just before reading the EEPROM out. This can be done, because writing to and reading from the EEPROM does not overlap. After reading the bits, the communication is set back to enable the clock delay (ECCR:SCDE = 1).

2.3.3 EEPROM Busy Timing

After writing data to the EEPROM it signals a busy state by setting a “0” to the serial output (DO/SIN). After the busy state the pin goes to “1”.

In the code example below, this busy wait is simply performed by polling the serial input of the USART (ESCR:SIOP) in two steps. First step is waiting for “0”, the second is waiting for “1”.

Because the busy state can take up to 10 ms, time critical software should use a timer interrupt for this, to save CPU performance.

2.3.4 Leading Zeros

The commands to the EEPROM are 9-Bit or 25-Bit wide. This does not fit into the 8-Bit pattern of the USART synchronous mode.

In the code example below a workaround for this is used. Because the EEPROM ignores leading Zero-Bits, the first byte is used to adjust the bit stream, using “leading Zeros”.

Example: A WEN (Write enable) command is sent to the EEPROM, which is composed of the following bit stream: 1 0011 XXXX. These are 9 Bits. The USART sends the following bit stream instead: 0000 0001 0011 XXXX. The blue “0s” are the “leading Zeros”, which are ignored by the EEPROM. This bit stream is represented by the bytes 0x01 and 0x3X (MSB first).

2.4 Used EEPROM Commands

The following EEPROM commands are used in the code example below:

Figure 5. EEPROM Commands

Instr.	Op Code	Address	Data	PE Pin	Comments
READ	110	A5-A0		X	Read data stored in memory
WEN	100	11XXXX		1	Enable programming (write enable)
WRITE	101	A5-A0	D15-D0	1	Write data to address
WDS	100	00XXXX	D15-D0	1	Disable programming (write disable)

Commands which need the PRE-Pin are not used in the code example below.

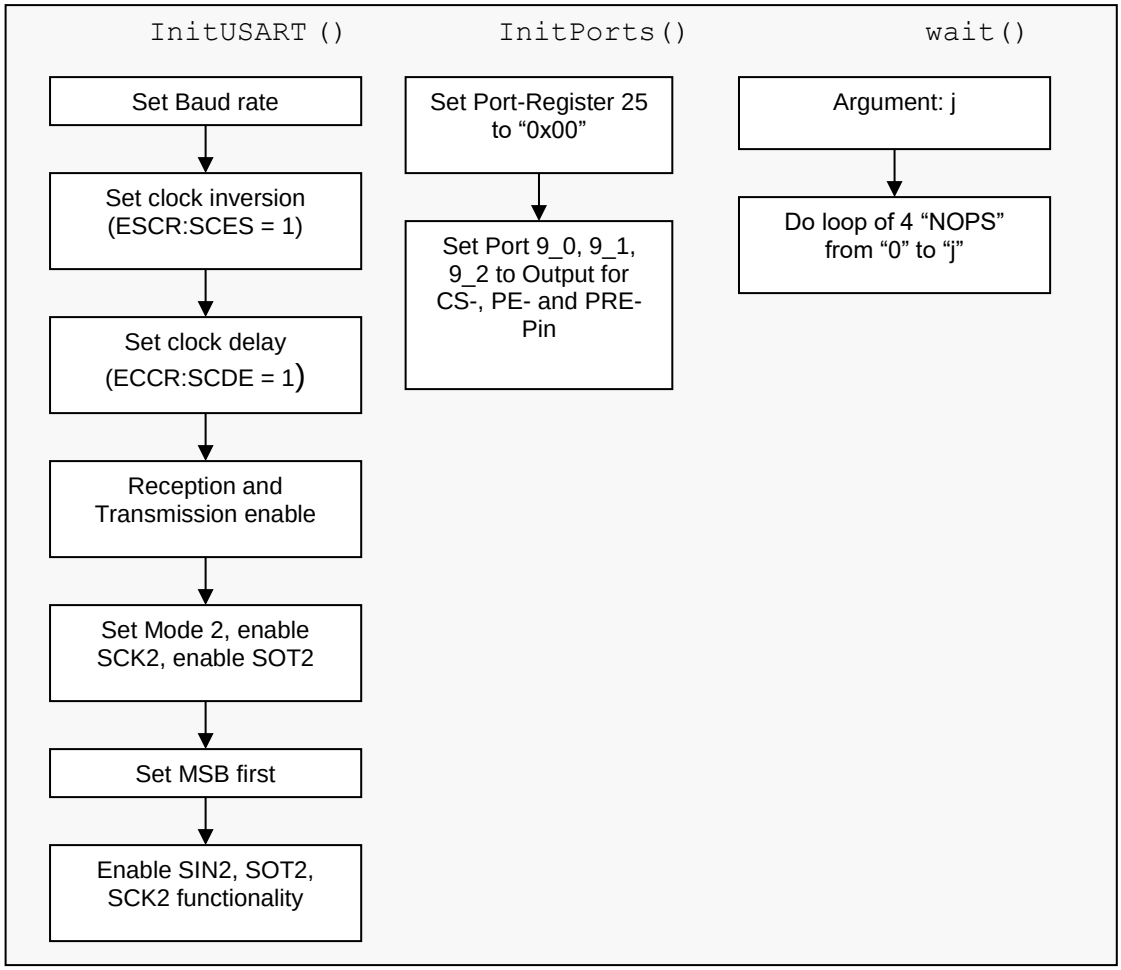
2.5 Example Code

The following code shows how to establish a communication to and from the EEPROM.

2.5.1 Initial Functions and Declarations

2.5.1.1 Flowchart

Figure 6. : Flowchart: Initialization Functions



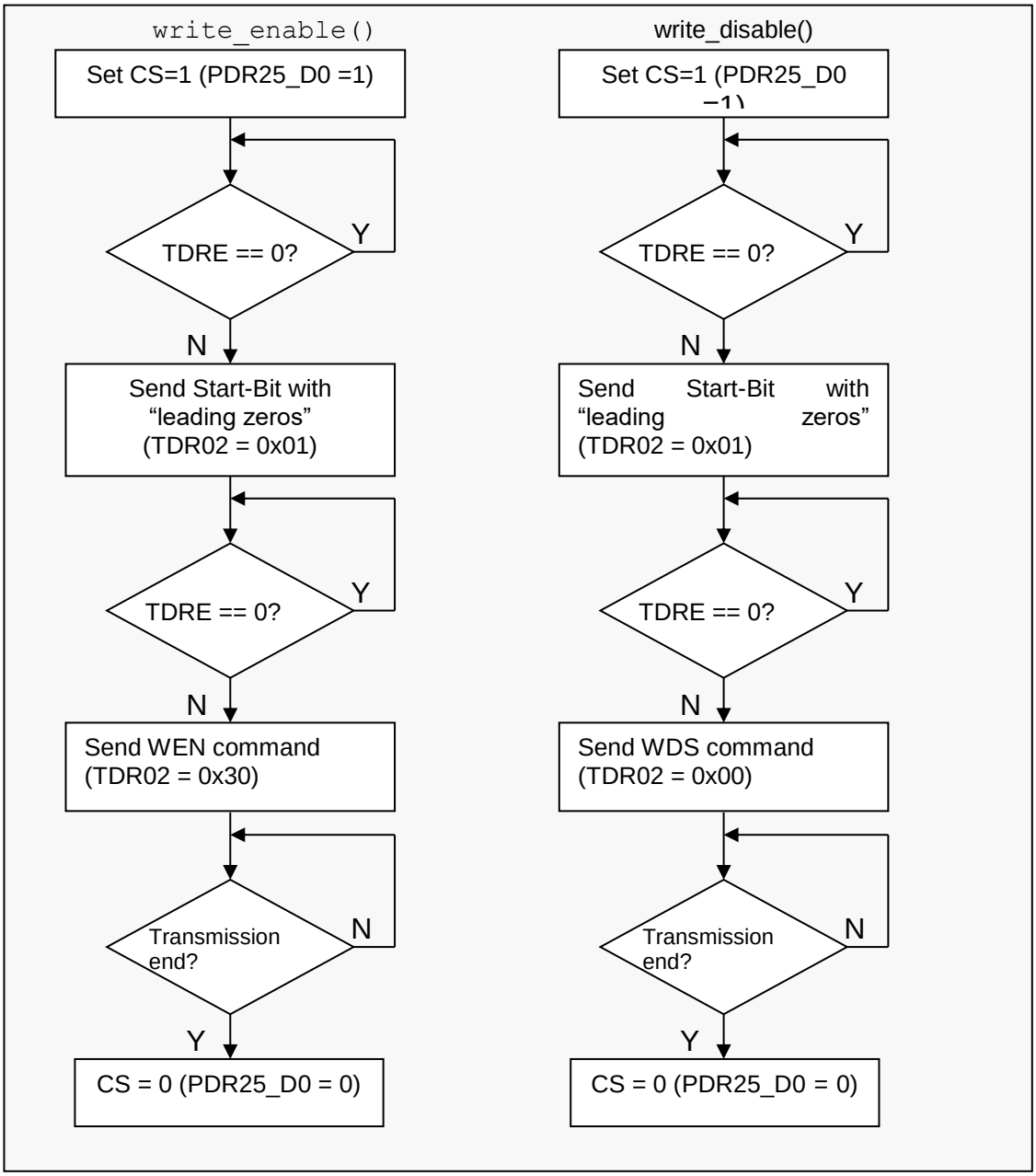
2.5.1.2 C Code

```
#define DATASIZE 64        // EEPROM memory size in words (16 Bit)

unsigned short data[DATASIZE];          // data to be sent to EEPROM
unsigned short readbuffer[DATASIZE];    // data received from EEPROM

void InitUSART(void)
{
    BGR02 = 15;             // 1M Bit/s @ 16 MHz
    ESCR02 = 0x01;          // SCES = 1
    ECCR02 = 0x10;          // SCDE = 1
    SCR02 = 0x03;    // reception and transmission enable
    SMR02 = 0x83;    // Mode 2, SCLK enable, SOT enable
    SSR02 = 0x04;    // MSB first, no interrupts

        PFR20_D0 = 1;              // enable SIN2 input
        PFR20_D1 = 1;              // enable SOT2 output
        EPFR20_D1 = 0;
        PFR20_D2 = 1;              // enable SCK2 output
        EPFR20_D2 = 0;
}

void InitPorts(void)
{
    // Bit#2: CS, Bit#1: PE, Bit#0: PRE
    PDR25 &= ~0x07;         // P25_0 to P25_2 low
    DDR25 |= 0x07;          // CS, PE, PRE to output
    PDR16 = 0x00;    // LED-Port
    DDR16 = 0xFF;
}

void wait(unsigned int j)
{
    volatile unsigned short i;

    for (i = 0; i < j; i++)
    {
        #pragma asm
        NOP
        NOP
        NOP
        NOP
        #pragma endasm
    }
}
```

2.5.2 Write Enable and Write Disable

2.5.1.3 Flowchart

Figure 7. Flowchart: Write Enable/Disable



2.5.1.4 C Code

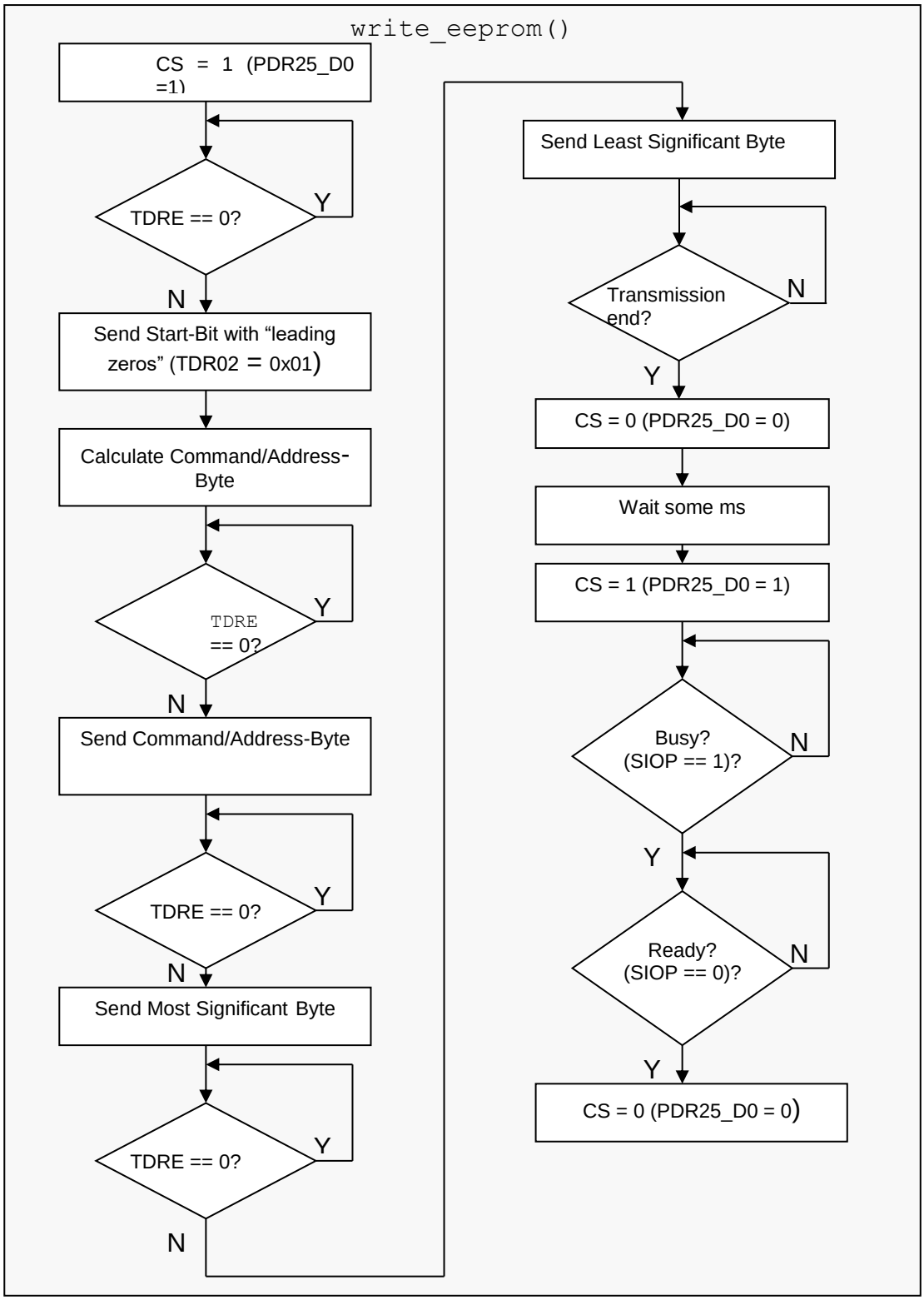
```
void write_enable(void)
{
    PDR25_D0 = 1;           // CS = 1

    while (SSR02_TDRE == 0);
    TDR02 = 0x01;           // Start-Bit (with "leading zeros")

    while (SSR02_TDRE == 0);
    TDR02 = 0x30;           // WEN command

    while (ECCR02 & 0x01);        // wait for start of transmission
       //  (or ongoing)
    while (!(ECCR02 & 0x01));     // wait for transmission finished

    PDR25_D0 = 0;           // CS = 0
}

void write_disable(void)
{
    PDR25_D0 = 1;           // CS = 1

    while (SSR02_TDRE == 0);
    TDR02 = 0x01;           // Start-Bit (with "leading zeros")

    while (SSR02_TDRE == 0);
    TDR02 = 0x00;           // WDS command

    while (ECCR02 & 0x01);        // wait for start of transmission
       //  (or ongoing)
    while (!(ECCR02 & 0x01));     // wait for transmission finished

    PDR25_D0 = 0;           // CS = 0
}
```

Note that the expression (ECCR02 & 0x01) masks the Transmission Bus Idle Bit (TBI).

2.5.2 Write to EEPROM

2.5.2.1 Flowchart

Figure 8. Flowchart: Write EEPROM



2.5.2.2 C Code

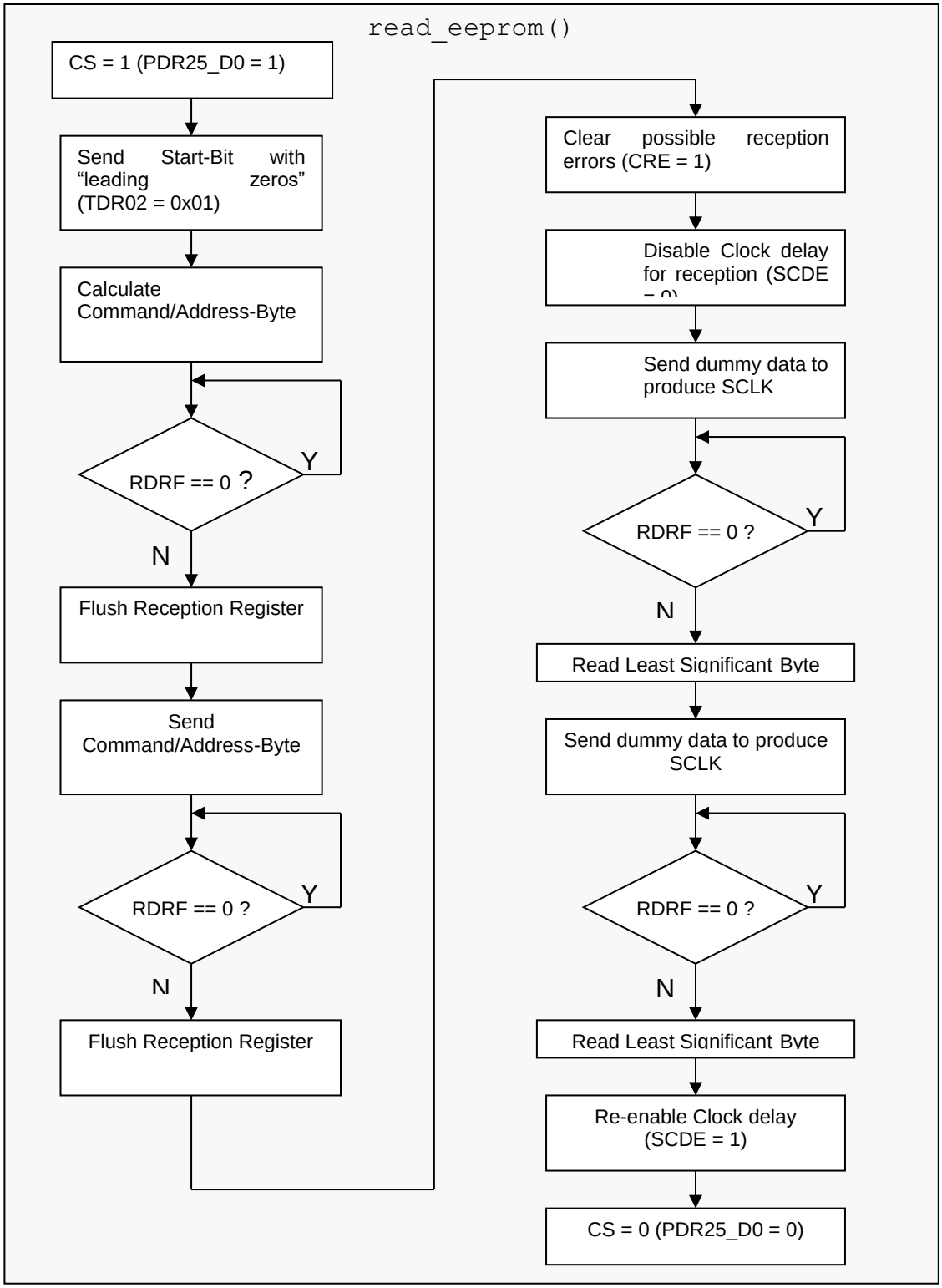
```
void write_eeprom(unsigned char adr)
{
    unsigned char dout, command;

    PDR25_D0 = 1;           // CS = 1

    while (SSR02_TDRE == 0);
    TDR02 = 0x01;           // Start-Bit (with "leading zeros")

    command = (adr & 0x3F) | 0x40; // Address and Write-Instruction
    dout = command;
    while (SSR02_TDRE == 0);
    TDR02 = dout;

    dout = (data[adr] >> 8) & 0xFF; // MSB
    while (SSR02_TDRE == 0);
    TDR02 = dout;

    dout = data[adr] & 0xFF;      // LSB
    while (SSR02_TDRE == 0);
    TDR02 = dout;

    while (ECCR02 & 0x01);        // wait for start of transmission
       //   (or ongoing)
    while (!(ECCR02 & 0x01));     // wait for transmission finished

    PDR25_D0 = 0;           // CS = 0

    wait(1);

    //   Next function (waiting for busy release) is made by
    //   polling. Please note, that for the NM93CS46 EEPROM the
    //   wait time can take till 10 ms! I. e. the CPU is then
    //   also busy. For fast application a timer should be used,
    //   which generates an interrupt after 10 ms from here,
    //   so that the CPU can perform other jobs in this time.
    PDR25_D0 = 1;           // CS = 1
    while(ESCR02_SIOP == 1);      // wait for eeprom busy
    while(ESCR02_SIOP == 0);      // wait for eeprom busy release
    PDR25_D0 = 0;           // CS = 0
}
```

Note, that the expression `(ECCR02 & 0x01)` masks the Transmission Bus Idle Bit (`TBI`).

2.5.3 Read from EEPROM

2.5.3.1 Flowchart

Figure 9. Flowchart: Read EEPROM



2.5.3.2 C Code

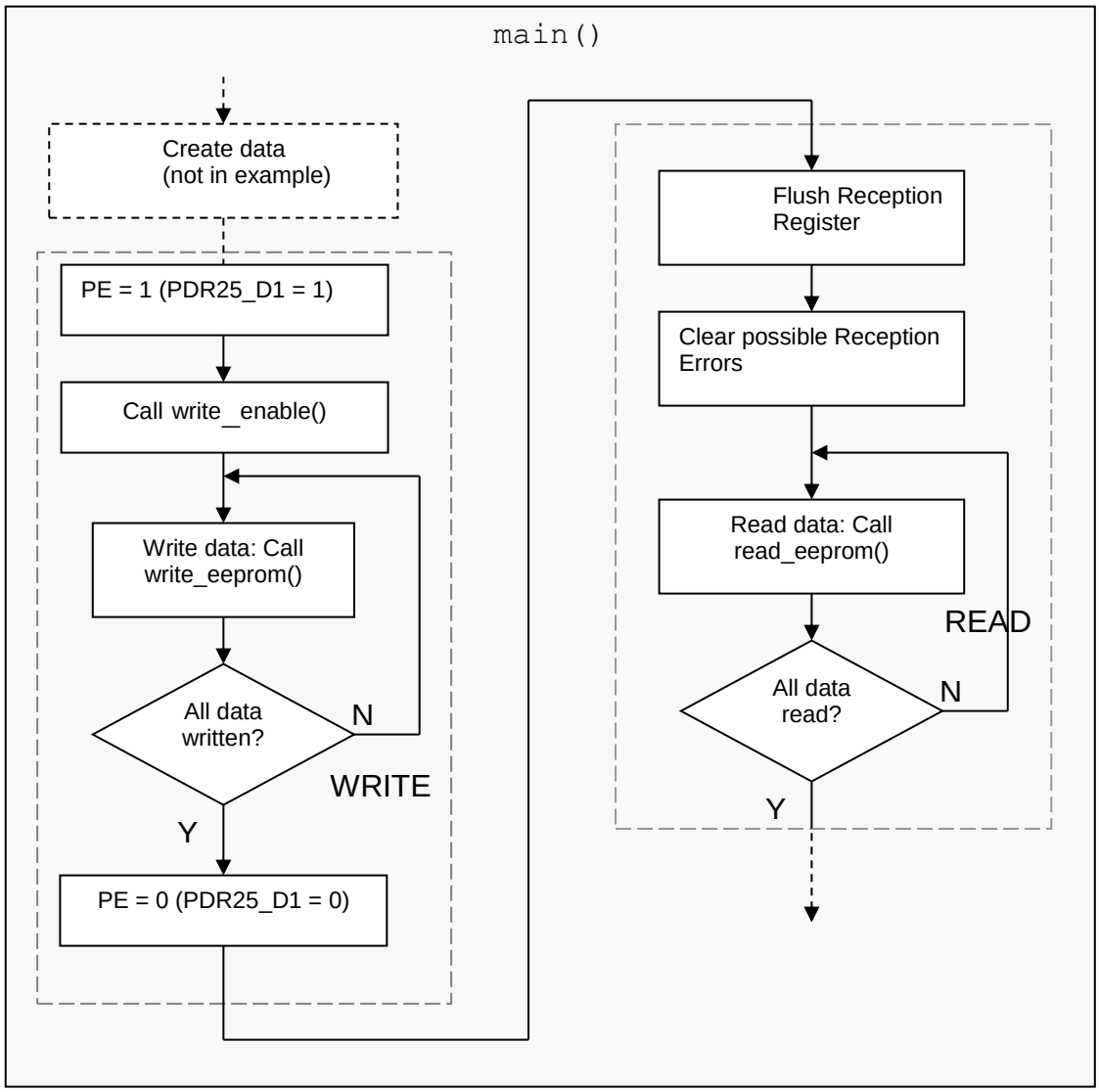
```
void read_eeprom(unsigned char adr)
{
    unsigned char din, command, dout;

        PDR25_D0 = 1;               // CS = 1

        while (SSR02_TDRE == 0);
        TDR02 = 0x01;         // Start-Bit (with "leading zeros")

    command = (adr & 0x3F) | 0x80; // Address and Write-Instruction
    dout = command;
    while (SSR02_RDRF == 0);      // transmission finished (via
        //   reception)?
    din = RDR02;            // flush reception register
    TDR02 = dout;

    while (SSR0_RDRF == 0);       // transmission finished (via
        //   reception)?
    din = RDR02;            // flush reception register
    SCR02_CRE = 1;                // Clear possible errors, reset
        //   reception state machine

    // NOTE: Make sure that SCK is "0" while setting SCDE to "0"
        //   (ECCR0 = 0x00;)
    //   In this case (1M bps) no check is needed. Be careful with
        //   slower baud rates!
    ECCR02 = 0x00;                // SCDE = 0 Needed
        //   for special read timing of
                    //   used EEPROM (may not be
        //   necessary for other EEPROMs)
    TDR02 = 0x00;           // set dummy byte to produce SCLK

    while (SSR02_RDRF == 0);      // transmission finished (via
        //   reception)?
    din = RDR02;            // MSB
    readbuffer[adr] = (din << 8);

    while (SSR02_TDRE == 0);
    TDR02 = 0x00;           // set dummy byte to produce SCLK

    while (SSR02_RDRF == 0);
    din = RDR02;            // LSB
    readbuffer[adr] = (readbuffer[adr] | din);

    ECCR02 = 0x10;                // SCDE = 1: Set back
        //   for write timing

        PDR25_D0 = 0;               // CS = 0
}
```

Note: Because reception is enabled and the USART is synchronous master, the USART always receives (dummy) data from SIN. Therefore the Reception Data Register Full Flag (`RDRF`) can be used to determine transmission end, because USART generates the serial clock during (dummy) transmission. This is an alternative to using the Transmission Bus Idle Flag (`TBI`).

2.5.4 Example of usage of EEPROM functions in Main Function

2.5.4.1 Flowchart

Figure 10. Flowchart: Main Function



2.5.4.2 C Code

The code above writes to/reads from all 64 16-Bit-Words of the EEPROM. The functions void write_eeprom(unsigned char address) and void read_eeprom(unsigned char address) can also be used separately for single address access.

```
void main(void)
{
    unsigned char i, error, dummy;

        InitIrqLevels();
        __set_il(31);           // allow all levels
        __EI();                 // globally enable interrupts
        PORTEN = 0x3;              // Enable I/O Ports

        InitPorts(); // Initialize PRE, PE, and CS
    InitUSART(); // Init USART for SPI communication

    // Put code or function call here to produce data for EEPROM

        // write data
    PDR25_D1 = 1; // PE = 1 must be set here
    write_enable();
    for (i = 0; i < DATASIZE; i++)
    {
        write_eeprom(i);
    }
    write_disable();
    PDR25_D1 = 0; // PE = 0 must be set here

    // Do something else ...

        // read data
    dummy = RDR02;   // flush read buffer
    SCR02_CRE = 1;   // clear possible overruns
    for (i = 0; i < DATASIZE; i++)
    {
        read_eeprom(i);
    }

    // Add further code here ...
}
```

It should be noted that the hardware watchdog clearing is not done in the code explained above and the real application should take care.

3 Related Documents

- DS93CS46 National Semiconductor data sheet of NM93CS46
- AN205101 - F(2)MC-8L/8FX/16LX16FX , MB89XXX / MB95XXX / MB90XXX / MB96XXX, PERFORMING SPI

4 Additional Information

Information about CYPRESS Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software example related to this application note is:

91460_spi_eeprom

It can be found on the following Internet page:

http://www.cypress.com/cypress-mcu-product-softwareexamples

Document History

Document Title: AN205378 - FR, MB91460, SPI Communication to/from Serial EEPROM (for NM93CS46)

Document Number: 002-05378

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	05/08/2008	V1.0; MPi initial version
*A	5133826	NOFL	02/11/2016	Converted Spansion Application Note “MCU-AN-300090-E-V10” to Cypress format
*B	5885034	AESATMP8	09/15/2017	Updated logo and Copyright.
*C	6070279	NOFL	02/13/2018	Sunset Review Migrated to new template Updated links

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709