AVR107: Interfacing AVR serial memories

Features

- **Devices: AT25128A/256A, AT25F1024/2048/4096**
- **Full Serial Memory Functions Support**
- **Memory Array Burst Read**
- **Page Burst Write**
- **Write Protection Detection**
- **On Going Access Detection**
- **Non-blocking Write Access**
- **Access Status Information**

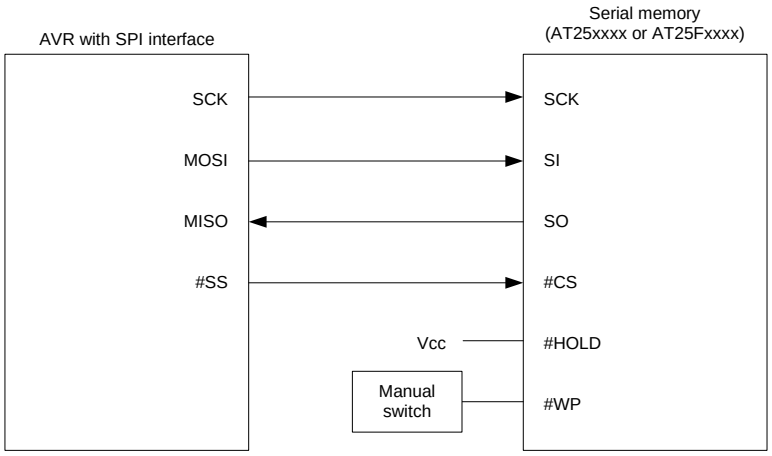
1 Introduction

Serial-interface memories are used in a broad spectrum of consumer, automotive, telecommunication, medical, industrial and PC related markets. Primarily used to store personal preference data and configuration/setup data, Serial memories are the most flexible type of non-volatile memory utilized today. Compared to other NVM solutions, serial memories devices offer a lower pin count, smaller packages, lower voltages, as well as lower power consumption.

Most of the AVR provide a SPI interface which enables a connection with a serial memory like the AT25128A/256A and AT25F1024/2048/4096.

To ease and accelerate the integration of SPI serial memories in customer's AVR systems, basic drivers were developed to efficiently access them. This application note describes the functionality and the architecture of these drivers as well as the motivation of the selected solution. The exhaustive list and description of the provided functions are part of the Doxygen Automated Document.

Figure 1-1. Hardware connections



8-bit AVR® Microcontrollers

Application Note

Rev. 2595A-AVR-03/05

2 Serial memories

The following sections describe connecting the memory to an AVR processor, memory access types, block write protection bits and busy detection. For more information refer to the data sheets.

2.1 Serial Memory to AVR Hardware Connection

The SPI interface is configured in the master mode. The SCK and MOSI ports are outputs and the MISO port is an input. Refer to Figure 1-1.

Note that in this application note we use the slave select signal #SS to control the chip select pin #CS of the serial memory. This requires taking some precautions during the initialization of the SPI interface (see Initialization of the SPI Interface). Another port such as PB2 could be used to control the chip select line.

The #HOLD signal can be held high in its inactive state because the SPI interface always halts the clock if the data not valid.

The user can choose the configuration of the write protect signal #WP.

Note: EEPROMs operating voltage range from 1,8V to 5,5V whereas Flashes operates from 2,7V to 3,6V.

2.2 SPI Serial Memory Access Types

In order to understand the structure of the proposed drivers, it is worth to review the different accesses to the serial memories.

All accesses follow this sequence:

1. The chip select line is driven low.
2. A number of serialized bytes are sent or received synchronously to the SCK clock on the data lines.
3. The chip select line is driven high.

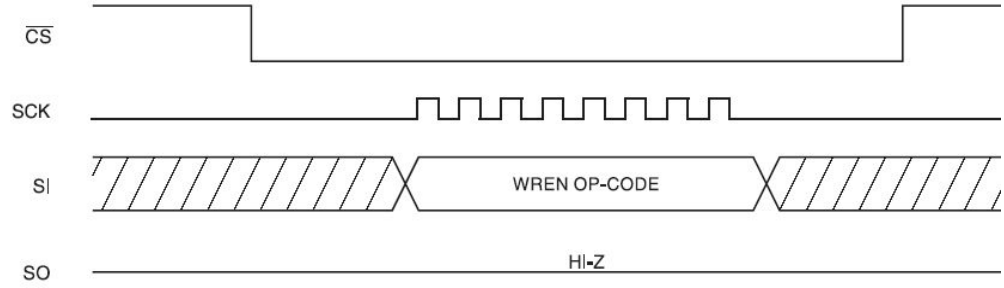
The number and value of the serialized byte depends on the access type.

The SPI memory devices of this application note both support the SPI mode 0(0,0) and mode 3 (1,1). In the proposed driver the mode 0 is selected (See serial memory datasheets for details).

2.2.1 Single Byte Write Commands: WREN, WRDI, CHIP ERASE

These accesses are one byte long. Only the instruction byte (later on called “op_code”) instruction is sent on the MOSI data line.

Figure 2-1. Example WREN Timing



2.2.2 Read/ Write Status Register: RDSR/WRSR

These accesses access are 2-byte long: The op_code byte is followed by the byte to be written (WRSR) or to be read (RDSR).

Note: The Write Status Register operation must be preceded by WREN command.

Figure 2-2. WRSR Timing

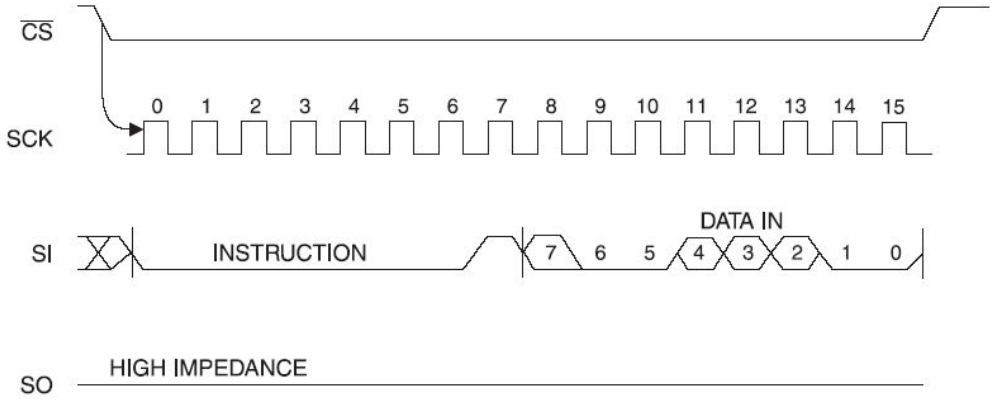
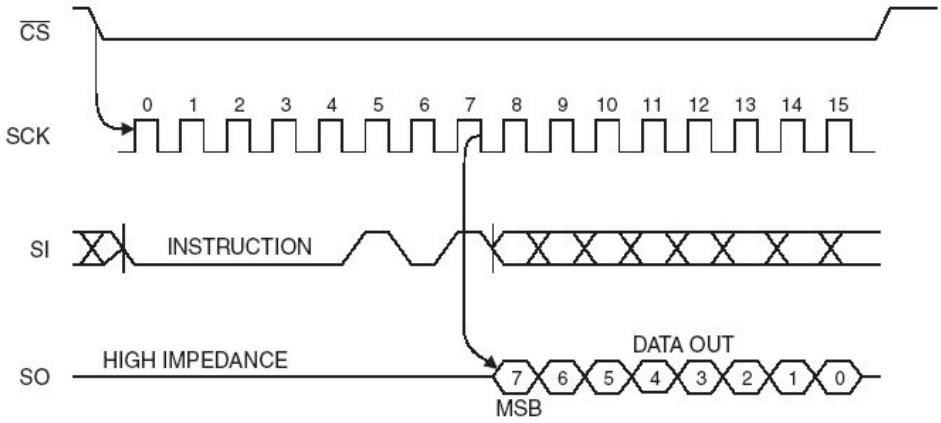


Figure 2-3. RDSR Timing

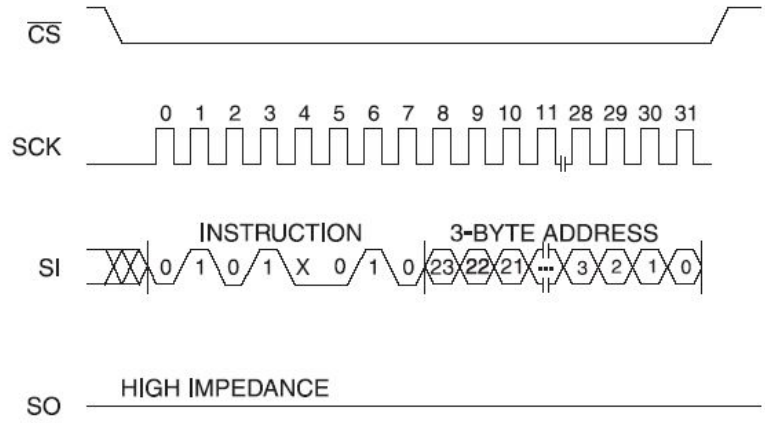


2.2.3 Sector Erase Command (only for AT25Fxxx Devices)

This access consists of a one-byte op_code and 3 address bytes.

Note: The Sector Erase operation must be preceded by WREN command.

Figure 2-4. Sector Erase Timing



X = Don't Care bit

2.2.4 Data Write and Data Read Accesses

These accesses consist of a one-byte op_code followed by an address phase (2-byte long for the AT25xxx devices, 3-byte long for the AT25Fxxx devices) and a data phase. The data phase length depends on the number of bytes to be written or read.

In case of a full memory array read, the number of bytes is the capacity in bytes of the memory.

Note: The WRITE and PROGRAM operations must be preceded by WREN command.

Figure 2-5. WRITE command Timing (AT25xxxx)

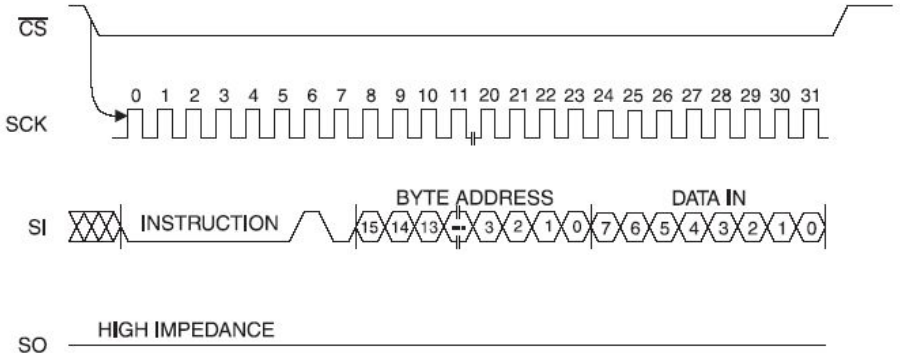
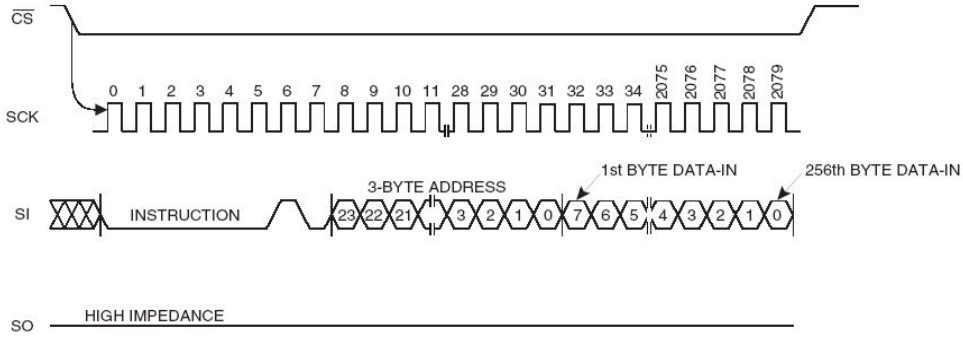


Figure 2-6. PROGRAM command Timing (AT25Fxxxx).



2.2.5 Access Type Summary

The following table summarises the access types and their related lengths:

Table 2-1. Access Type Summary Table

Command	Op_code	Address Phase Length (bytes)		Data Phase Length (bytes)	
		AT25xxxx	AT25Fxxxx	MOSI line	MISO line
WREN	0000 X110	0	0	0	0
WRDI	0000 X100	0	0	0	0
RDSR	0000 X101	0	0	0	1
WRSR	0000 X001	0	0	1	0
READ	0000 X011	2	3	0	1 to array size
WRITE	0000 X010	2	N/A(1)	1 to page size	0
PROGRAM	0000 X010	N/A(1)	3	1 to page size	0
SECTOR ERASE	0101 X110	N/A(1)	3	0	0
CHIP ERASE	0110 X110	N/A(1)	0	0	0
RDID	0001 X101	N/A(1)	0	0	2

Notes: 1. Command not applicable for this device.

2.3 SPI Peripheral Handling

To perform the accesses to serial memory we use the SPI interface which enables, synchronously to the SCK, to shift out on the MOSI line the byte that has been written in the SPDR register and to shift in the byte on the MISO line into the SPDR.

The SPI operates byte-wise. After a byte was sent or received, the flag SPIF is set in the SPSR register.

We can envision two ways of operation to handle the SPI peripheral:

1. Polling the SPSR register to detect if the SPIF was set, and then perform the next SPI transfer.
2. Enable the SPI interrupt and manage the next SPI transfer in the SPI interrupt handler.

2.3.1 Polling Operation Mode

The polling operation has the advantage to respond quickly to the end of a transfer and also to lead to a compact code size.

The crucial drawback is that a polling operation is a blocking mechanism, since the code must wait for the end of the polling loop.

We thus select the polling operation mode for short accesses:

- WREN
- WRDI
- WRSR
- RDSR
- SECTOR ERASE
- CHIP ERASE
- RDID

Since we assumed that the main routine waits for the read data to be received before going on with its code execution, we have also selected a polling operation for:

- READ

2.3.2 Interrupt Operation Mode

For the data burst write accesses, a polling operation mode is not well suited. Indeed the main routine does generally not need to wait for the end of the serial memory write cycle before going forward with its execution code.

We thus select the interrupt mode of operation to handle the accesses:

- WRITE
- PROGRAM

In this mode the SPI interrupt handler manages the next SPI transfer. This enables the main routine to execute its code between to SPI interrupts.

To afford the interrupt routines to handle the next SPI transfer we must save the access context into global variables (see 3.3.2 The Global Variables Description). This consumes a few byte of the memory space.

2.3.3 Interaction between both Operation Modes

To avoid any conflict on the SPI interface between functions which uses polling operation mode and those which uses the interrupt mode, the following precaution must be taken:

1. Before any polling operation, it is checked (trough the global variable state) that the SPI peripheral is ready to send. Then the SPI interrupt is disabled.
2. At the end of an access using the polling operation mode, the SPI interrupt is enabled.

2.4 Write Protected Block Detection

The serial memory devices provide a write protection function that also had to be considered when writing the code.

The protected sectors depend on the setting of the BPx bits of the serial memory status register. Please refer to the serial memory datasheet for more details.

Table 2-2. AT25F4096 memory protection options

	Status Register Bits			AT25F4096	
Level	BP2	BP1	BP0	Array Addresses Locked Out	Locked-out Sector(s)
0(none)	0	0	0	None	None
1(1/8)	0	0	1	070000 - 07FFFF	Sector 8
2(1/4)	0	1	0	060000 - 07FFFF	Sector 7, 8
3(1/2)	0	1	1	040000 - 07FFFF	Sector 5, 6, 7, 8
4(all)	1	x	x	000000 - 07FFFF	All sectors (1 - 8)

Note: 1. x = don't care

Table 2-3. AT25128A/ AT25256A memory protection options

	Status Register Bits		Array Addresses Protected	
Level	BP1	BP0	AT25128A	AT25256A
0	0	0	None	None
1(1/4)	0	1	3000 – 3FFF	6000 – 7FFF
2(1/2)	1	0	2000 – 3FFF	4000 – 7FFF
3(All)	1	1	0000 – 3FFF	0000 – 7FFF

The proposed driver includes the function SetWriteProtectedArea that enables the user to define and active a write protected area.
Note the function SetWriteProtectedArea will detect if the serial memory is hardware write protected (#WP line is low) by writing and reading back the status register. In case of a hardware write protection, nor the set up either the activation of a write protection configuration can be performed.

By a data write access (WRITE/PROGAM command) the status register of the serial memory is read. According to the value of the BPx bits and the address location of

the last byte to be written, the write function detects if the destination address is located in a write-protected area.

If yes, the returned access status code will be set to the DATA_WR_PROTECTED value and the access will be aborted.

For AT25F devices, the write protection detection is active when performing a SECTOR ERASE or a CHIP ERASE operation.

2.5 Busy Detection

The serial memory indicates its current state with the bit #RDY in its status register.

If the bit is set to ‘1’ then all operations except the RDSR access are forbidden.

Therefore each function will test if the #RDY bit is set. If yes the returned access status code will be set to the BUSY value and the access will be aborted.

Note: since during a memory write cycle operation all bits of the status register are set to ‘1’, the memory must be ready prior to read and evaluate the status registers information bits.

3 Code implementation

The following sections describe initialization of the SPI Interface, The Polling Mode Functions and the Interrupt Mode Functions.

3.1 Initialization of the SPI Interface

1. The SPI interface must be configured as a master. To ensure that the SPI does not switch to the slave mode through the slave select line, the SS port must be configured as an output with a high level.
    a. The SPI interface is first disabled to configure the PB2 port
    b. The PB2 port is configured as an output
    c. A high value is set on the port PB2
    d. The MRST bit is set
    e. The SPI is enabled
2. The SPI mode 0 must be selected to ensure among others that the clock signal is low on a chip select rising edge. The most significant bit must be the first transmitted bit. The choice of the data rate is user-defined.
3. The other port directions must be configured as following:
    a. Outputs: PB5/SCK, PB3/MOSI
    b. Inputs: PB4/MISO.
4. The SPIF flag must be cleared by reading the SPSR and SPDR register.
5. The SPI interrupt is enabled

3.2 Polling Operation Mode Functions

WREN, WRDI, WRSR, RDSR, SECTOR ERASE, CHIP ERASE, RDID and READ all call the elementary function spi_transfer.

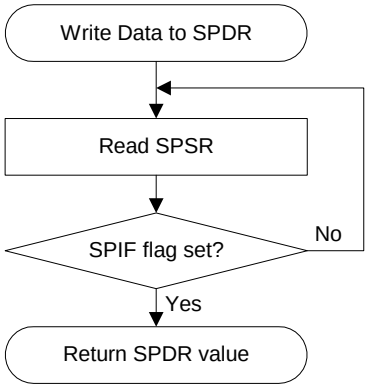
3.2.1 The SPI Transfer Elementary Function

The function spi_transfer is the elementary function that handles the SPI interface in polling mode. It sends a byte and stores the received one. Note that this function does not control the level of the chip select line.

Prototype:

```
char spi_transfer(char data);
```

Figure 3-1. spi_transfer flowchart



3.2.2 The Get Status Register Function

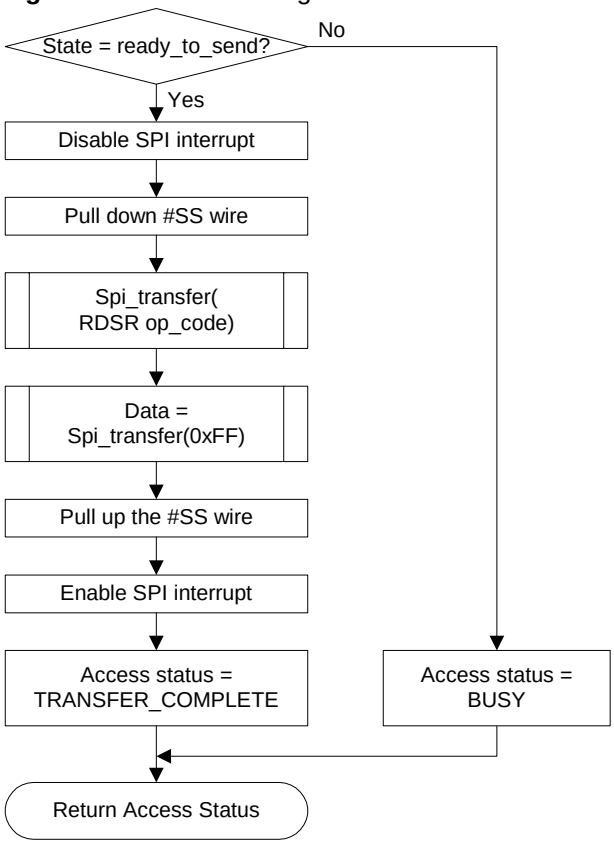
This function performs a RDSR access. It returns the access status code. The value of the status register is place into the output parameter.

It first checks that the SPI interface is not locked by a write process. (See the Data Write Accesses section)

Prototype:

```
char GetStatusRegister(char *data)
```

Figure 3-2. GetStatusRegister flowchart



3.2.3 The Single Byte Write Command

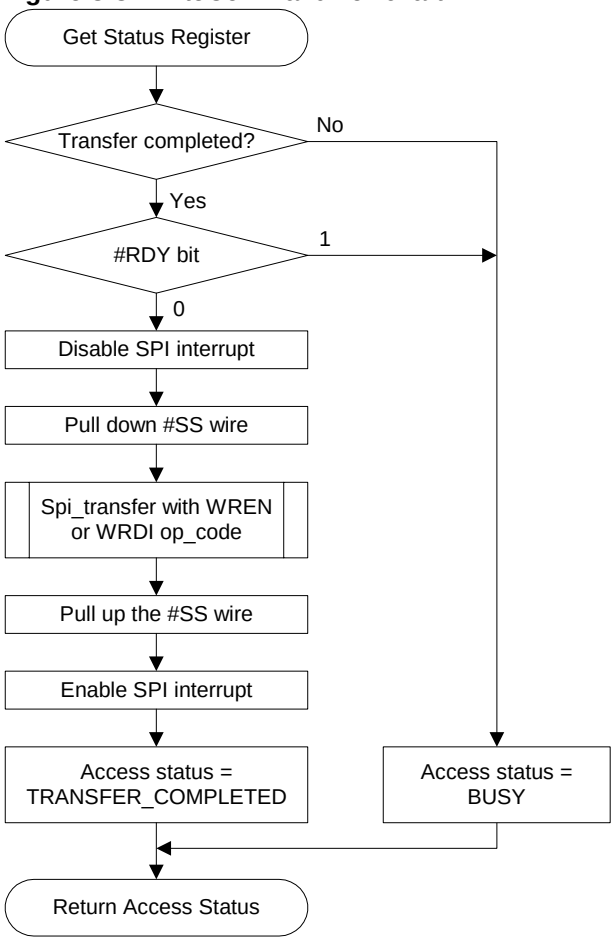
The WREN and WRDI accesses are on byte long and can be performed by the function WriteCommand. The serial memory must be ready before performing the access.

It returns the access status code.

Prototype :

```
char WriteCommand(char op_code)
```

Figure 3-3. WriteCommand flowchart



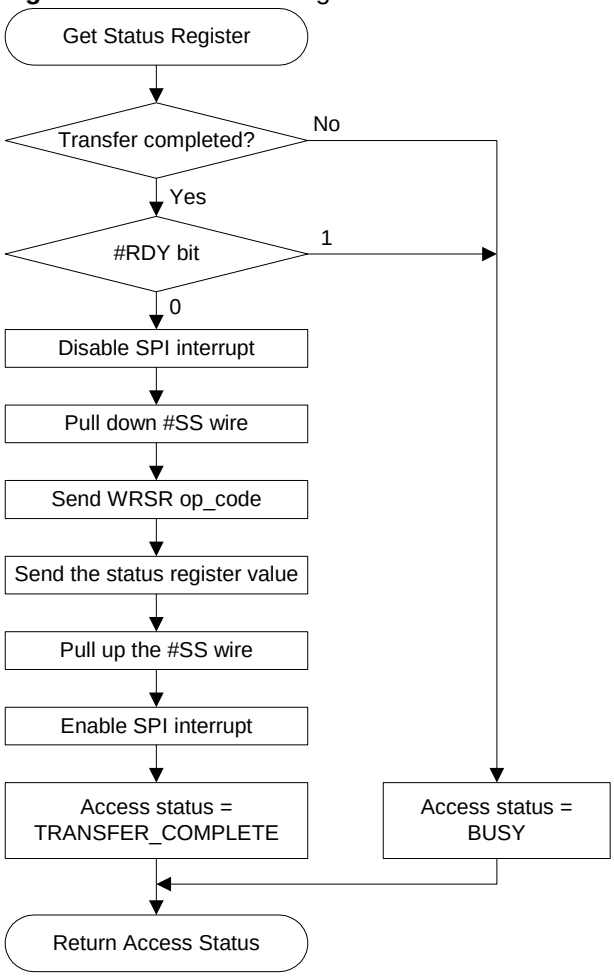
3.2.4 The Write Status Register Function

This function performs the RDSR access. It checks that the serial memory is ready to be written. It returns the same access status code as the WriteCommand function.

Prototype :

```
char WriteStatusReg(char sr)
```

Figure 3-4. WriteStatusReg flowchart



3.2.5 The Get Character Array Function

This function reads one or multiple bytes from the serial memory

If the serial memory or the SPI interface is busy, the function returns immediately with the according access status code.

Prototypes:

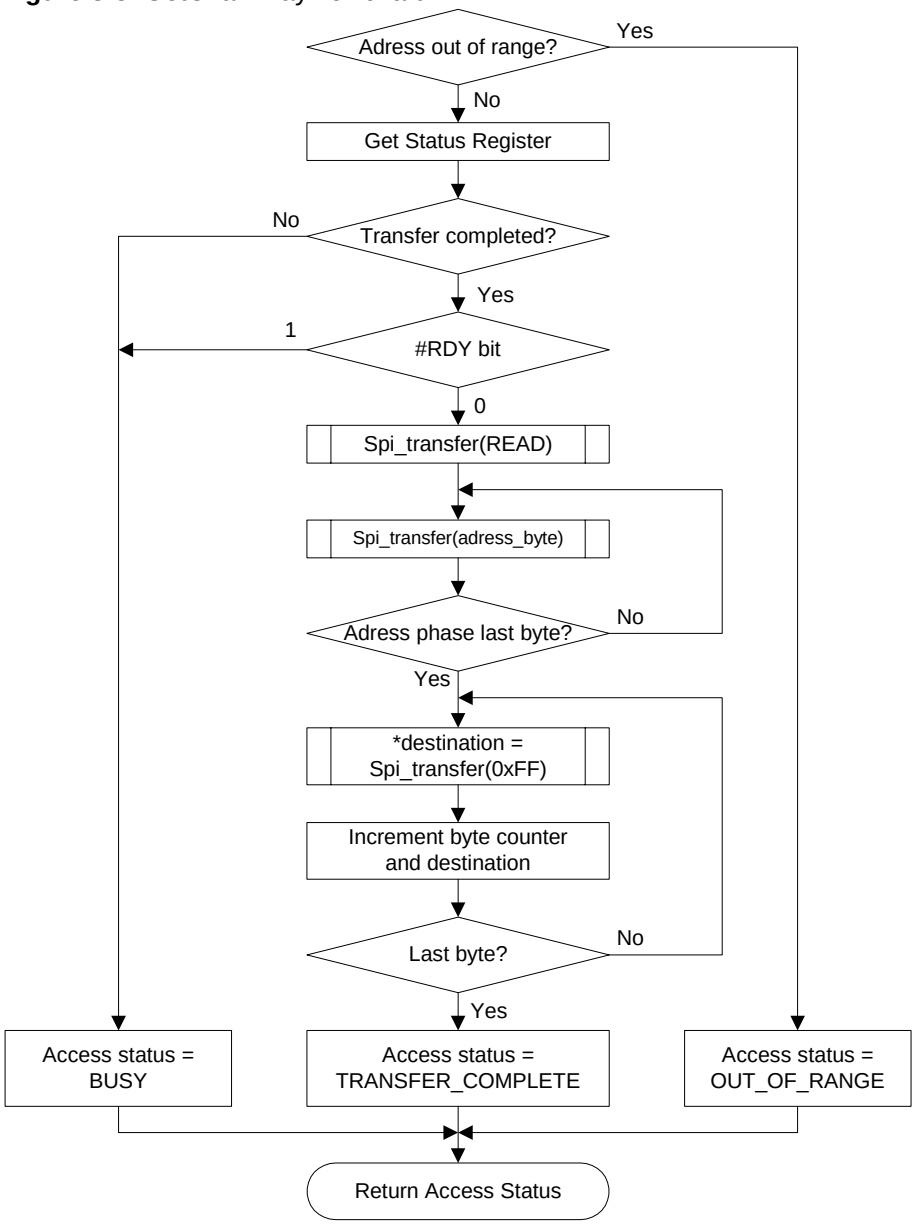
For AT25 devices:

```
char GetCharArray(int address, int nb_of_byte, char* destination);
```

For AT25F devices :

```
char GetCharArray(unsigned long address, int nb_of_byte, char*
destination);
```

Figure 3-5. GetCharArray flowchart



3.2.6 The Get Character Function

To minimise the code size, the GetChar function calls the GetCharArray function with only one byte as parameter.

Prototypes:

For AT25 devices:

```
char GetChar(int address, char *data);
```

For AT25F devices:

```
char GetChar(unsigned long address, char *data);
```

3.3 Interrupt Mode Functions

3.3.1 Principle

Like a read access, a write access to the serial MEMORY is composed of a SPI transfer sequence. But unlike to the read access, the end of a SPIF transfer will be determined by the SPI interrupt. In order to let the write access be fully performed in background vs. the main routine execution, the SPI interrupt handler must be able to prepare the next SPI transfer as well as to control the chip select wire.

To enable such a mechanism, the interrupt handler must know the current context. This context is saved onto global variables.

A state global variable contains the information about the current SPI transfer type (INSTRUCTION, ADDRESS, DATA).

When a SPI interrupt is raised, meaning the end of a SPI transfer, the interrupt handler will determine which type of SPI transfer, if any, must be started. The interrupt handler acts as a finite state machine reacting on the SPI interrupt event.

3.3.2 The Global Variables Description

The state global variable describes the phase of the write access (See the finite state machine diagram for more details):

INSTRUCTION: The op_code is being transferred

ADDRESS: The current address byte is being transferred

DATA: The current data byte is being transferred

The *nb_byte* global variable contains the total number of data byte to be written minus one.

The *byte_cnt* global variable counts the number of address or data byte already written.

The *address global* variable contains the address of the serial memory where the first byte must be written.

The *data_ptr global* variable contains the address of the SRAM where the first byte is located.

3.3.3 The Put Character Array Function

This function writes an array of bytes into the serial memory. The size of the array can be up to the page size of the serial memory. The parameter source is the pointer to this array.

The function checks if the address is out of range or matches a write protected area. Additionally the function returns an access status providing a detection of a busy state of the serial.

Note: If the read sequence crosses a page boundary, a page rollover will occur.

Please refer to the Doxygen documentation for further details.

Prototype:

For AT25 devices:

```
char PutCharArray(int address, char nb_of_byte, char* source);
```

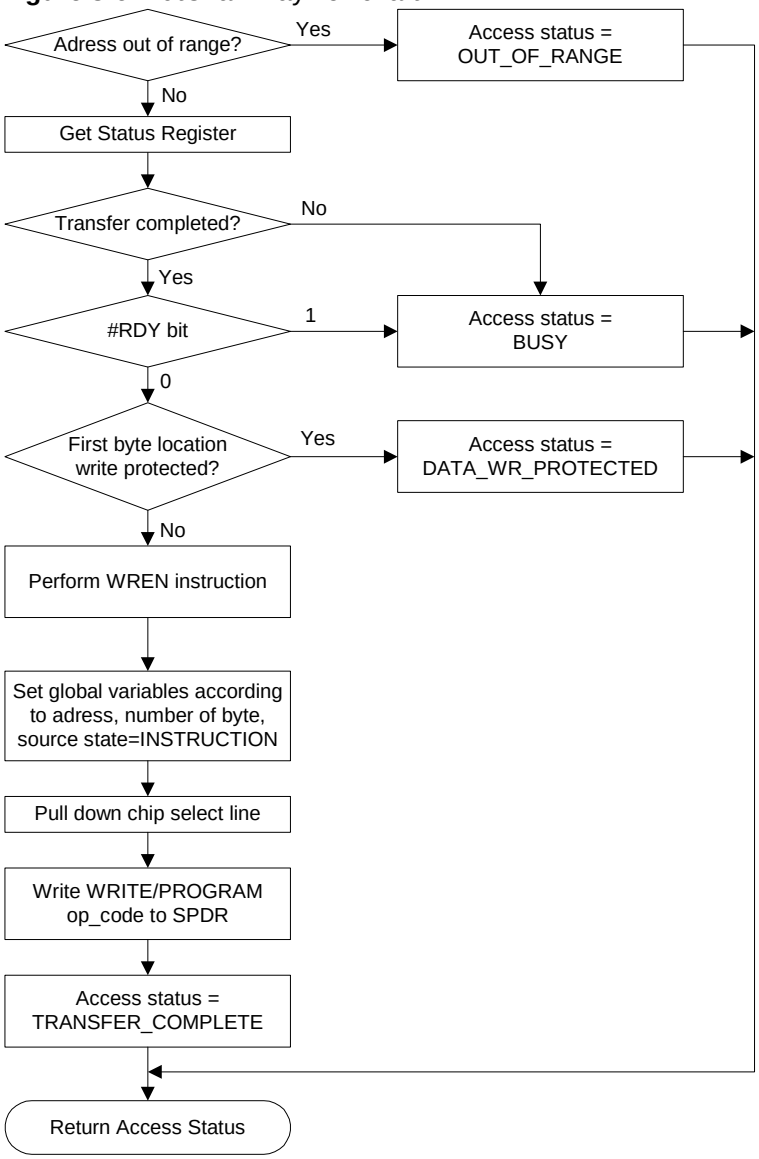
For AT25F devices:

```
char PutCharArray(unsigned long address, char nb_of_byte, char*
source);
```

Note:

- The parameter nb_of_byte is the number of byte decrement by one so that even if the page is 256-byte long, the global variable byte_cnt occupies only one byte.
- For the AT25F devices, the concerned sector must have been erased prior to a PutCharArray operation.
- The Chip and Sector Erase Function (only for AT25Fxxxx devices) have the same structure and access status code as the GetCharArray, except that the data phase does not exist. The function controls that the location to be erased is not write protected.

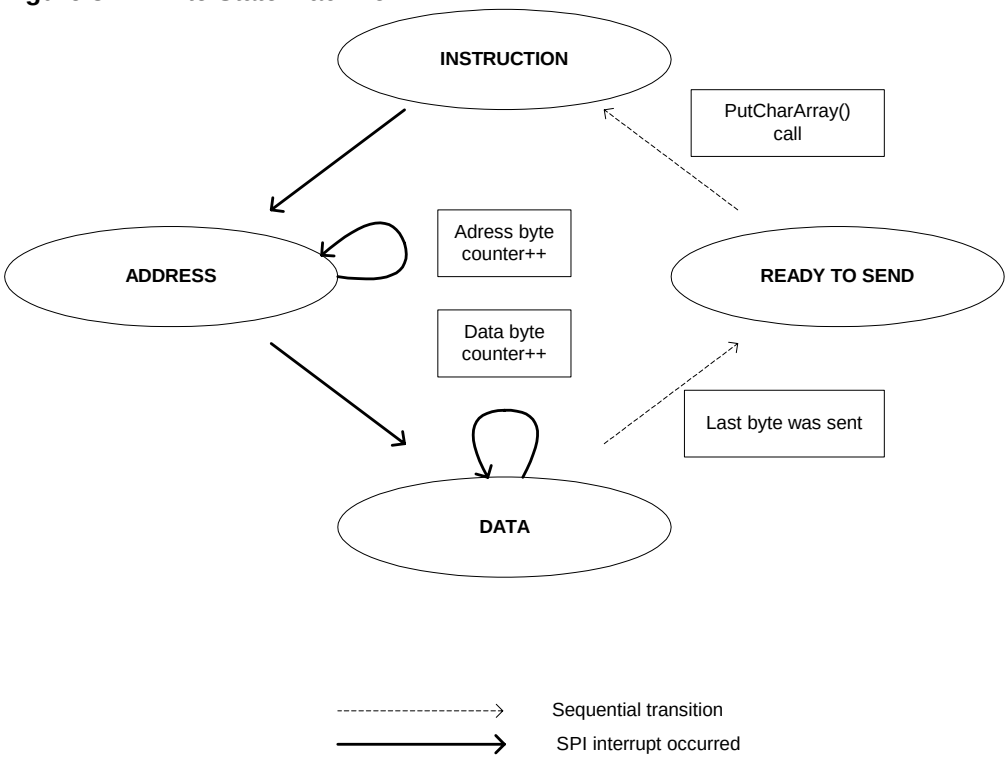
Figure 3-6. PutCharArray flowchart



3.4 The Finite State Machine

The following diagram describes the state global variable transitions during a write cycle. Each time the SPI interrupt handler is entered the state is evaluated and the next SPI transfer is performed. If the last bye was transferred, the state goes into the idle state called READY_TO_SEND:

Figure 3-7. Finite State Machine



3.4.1 The Instruction State

Entering the interrupt handler with the state variable set to INSTRUCTION means that the instruction code WRITE (or PROGRAM for AT25Fxxx Flashes) has been sent over the SPI interface.

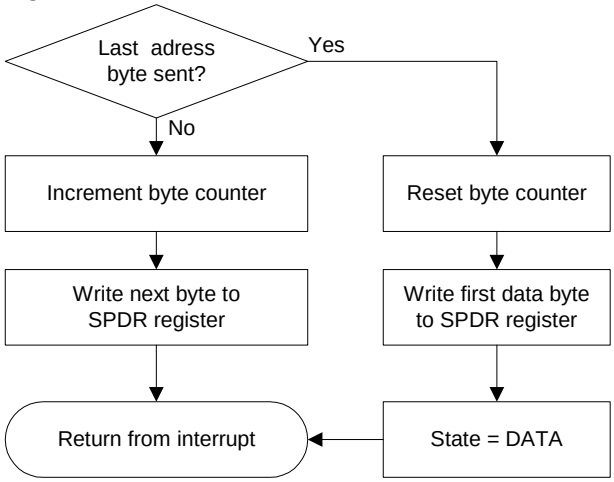
The next state becomes ADDRESS and the byte counter is initialized to 1.

The first byte of the address is written to the SPDR register.

3.4.2 The Address State

Entering the interrupt handler with the state variable set to ADDRESS means that one byte of the address has been sent over the SPI interface.

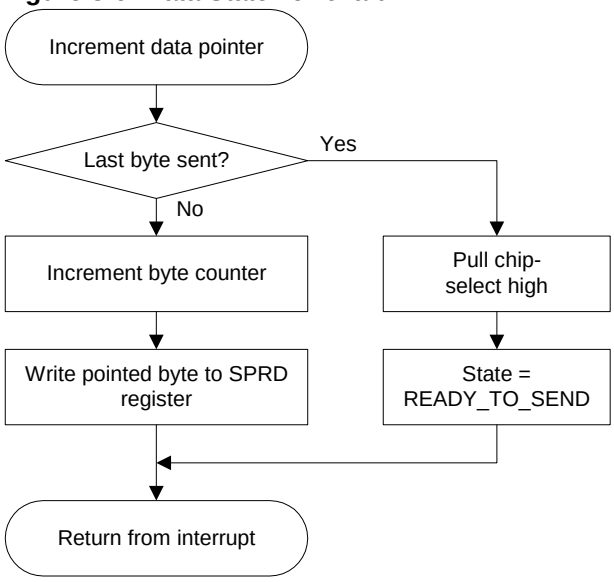
Figure 3-8. Address State flowchart



3.4.3 The Data State

Entering the interrupt handler with the state variable set to DATA means that a data byte has been sent over the SPI interface.

Figure 3-9. Data State flowchart



3.4.4 The Ready-to-Send State

This state cannot be active when entering the SPI interrupt handler. It signifies that there is no current data write access using the SPI interface.

Therefore the state global variable can be tested in order to determine the end of a SPI write transfer.

Note: The state value can be READY_TO_SEND while the serial memory is still in a busy state.

3.5 The PutChar Function (Only for the AT25128A/256A Devices)

The serial EEPROM devices are not organized with sectors and do not require to contain a 0xFF pattern prior a write access. Therefore there is no need to perform a read-modify-write operation, and the PutChar function can be build upon a call to the PutCharArray function passing one byte as parameter.

Note: For AT25F serial flashes a read-modify-write operation over a whole sector is needed to ensure the usual functionality of the PutChar function. Since such a operation is highly time and memory space consuming, we will not provide this function for the serial flashes.

4 AT25128A/256A Devices: Example of Driver Use

This example shows to benefit from the non-blocking PutCharArray function:

```
  main()
  {
    int start_address =  0x100;
    char* src = malloc(PAGE_SIZE* sizeof(char));
    char* dest = malloc(PAGE_SIZE * sizeof(char));
    char AccessStatus = BUSY;

  ////////////////////////////////////////////////////////////////////////////////
    //Init SPI peripheral
    ////////////////////////////////////////////////////////////////////////////////
    init_spi_master();


    //Disable Software write protection
    do {AccessStatus = SetWriteProtectedArea(NONE);}
    while (AccessStatus == BUSY);
    if (AccessStatus == HW_PROTECTED)
    {
      // Handle here a hardware protection detection
    }

    // Write a page
    do {AccessStatus = PutCharArray(start_address, (PAGE_SIZE-1), src);}
    while (AccessStatus == BUSY);
    // OPTIONAL : handle errors
    if (AccessStatus == OUT_OF_RANGE)
    {}
    else if (AccessStatus == DATA_WR_PROTECTED)
    {}

    ////////////////////////////////////////////////////////////////////
  // HERE USER CODE (Write Access is being performed in the SPI Interrupt Handler)
    ////////////////////////////////////////////////////////////////////

    // OPTIONAL : Disable Write on the SPI memory (wait for the memory to be ready)
    while (WriteCommand(WRDI) != TRANSFER_COMPLETED){}

    ////////////////////////////////////////////////////////////////
    // HERE ADDITIONAL USER CODE
    ////////////////////////////////////////////////////////////////

    //Read a page
    while (GetCharArray(start_address, (PAGE_SIZE-1), dest) != TRANSFER_COMPLETED) {}
    //the page is available in the SRAM buffer pointed by dest
}
```

5 AT25F1024/2048/4096 Devices: Example of Driver Use

This example shows to benefit from the non-blocking PutCharArray function:

```
  main()
  {
    int start_address =  0x100;
    char* src = malloc(PAGE_SIZE* sizeof(char));
    char* dest = malloc(PAGE_SIZE * sizeof(char));
    char AccessStatus = BUSY;

    ////////////////////////////////////////////////////////////////////////////////
    //Init SPI peripheral
    ////////////////////////////////////////////////////////////////////////////////
    init_spi_master();

    //Disable Software write protection
    do {AccessStatus = SetWriteProtectedArea(NONE);}
    while (AccessStatus == BUSY);
    if (AccessStatus == HW_PROTECTED)
    {
      // Handle here a hardware protection detection
    }

    //Erase concerned sector
    do {AccessStatus = EraseSector(start_address);}
    while (AccessStatus == BUSY);
    // OPTIONAL : handle errors
    if (AccessStatus == OUT_OF_RANGE)
    {}
    else if (AccessStatus == DATA_WR_PROTECTED)
    {}
    else
    {
      // Write a page
     while (PutCharArray(start_address, (PAGE_SIZE-1), src)== BUSY){}
    }
    ////////////////////////////////////////////////////////////////////
    // HERE USER CODE (Write Access is being performed in the SPI Interrupt Handler)
    ////////////////////////////////////////////////////////////////////

    // OPTIONAL : Disable Write on the SPI memory (blocking operation)
    while (WriteCommand(WRDI) != TRANSFER_COMPLETED){}

    ////////////////////////////////////////////////////////////
    // HERE ADDITIONAL USER CODE
    ////////////////////////////////////////////////////////////

    //Read a page
    while (GetCharArray(start_address, (PAGE_SIZE-1), dest) != TRANSFER_COMPLETED) {}
    //the page is available in the SRAM buffer pointed by dest
}
```

6 Development Environment

6.1 Hardware

- STK500 board with a 8-lead soldered PDIP socket on ports B
- AVR Atmega168
- AT25256A in a PDIP package
- AT25F4096 in a PDIP package
- JTAGICE mkII - In-Circuit Emulator with debugWIRE

6.2 Software

- IAR Embedded Workbench - EWAVR 3.20C
- AVR Studio Version 4.10

7 Code Size

The following figures are computation based on the map listing file. The code optimization option was set to high.

The main routine code size are not included.

Table 7-1. Code size.

Target device	Code size [bytes]
AT25256A	752 (to be updated)
AT25F4096	1086 (to be updated)

8 References

Datasheets:

- SPI Serial EEPROMS AT25128A and AT25256A
- SPI Serial Memory AT25F4096
- AVR ATmega168V

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

2595A-AVR-03/05