A BRIEF OVERVIEW OF FLASH MEMORY: -



* Flash memory is a non – volatile type semiconductor memory i.e., the data stored in it will be preserved even after the power supply is cut – off.
* Flash memory is the improved version of an EEPROM because of the way the data is handled in them.
* In EEPROM, only 1 byte of data can be stored, retrieved and erased at a time but in case of flash memory, an entire block of data can be stored, retrieved and erased at a time and due to this reason, the operations of a flash memory are much faster than that of an EEPROM.



* Flash memory finds its application in electronic devices like mobile phones, computers, etc. and microcontrollers also have flash memory in them where the application program is stored.



* Flash memory is also used as an alternative to EEPROM in many embedded system applications for storing important parameters like reference values, calibration data, back – up data, etc.

FLASH MEMORY

|  |  |
| --- | --- |
| NOR FLASH MEMORY | NAND FLASH MEMORY |
| 1. First developed by Intel Corporation in 1988. | 1. First developed by Toshiba Corporation in 1989. |
| 2. Memory block size: 64 kB to 128 kB. | 2. Memory block size: 8 kB to 32 kB. |
| 3. Slow data operations due to larger block size. | 3. Fast data operations due to smaller block size. |
| 4. Parallel connection of memory cells, thus, allowing individual access to each memory byte. | 4. Series connection of memory cells, thus, allowing sequential access to the memory cells and they must be accessed serially. |
|  |  |

\*\*In most embedded system applications, NOR flash memory is preferred to NAND flash memory because the MCU's and other peripherals mainly require byte level access to the memory which is provided by the NOR flash.

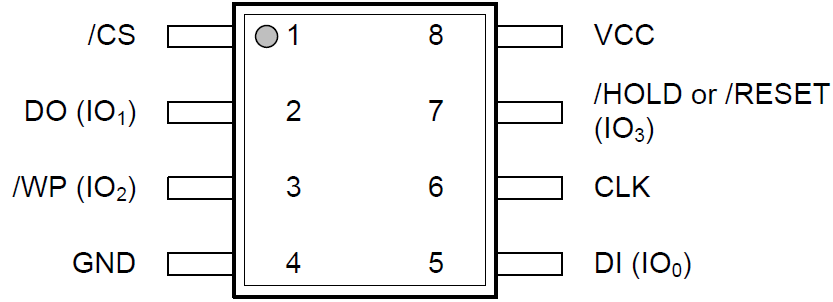
INTERFACING NOR FLASH MEMORY WITH ATMEGA32 MCU: -

Specifications: - \*(Taken from Page No. 5 of its datasheet)

|  |  |
| --- | --- |
| Chip Number | W25Q64FWSIG (Win-Bond) |
| Type of Flash Memory | NOR Flash |
| Number of Pins | 8 Pins |
| Chip Package | SOIC / VSOP, 208 - mils |
| Interfaces Supported | Standard SPI, Dual SPI, Quad SPI and QPI |
| Operating Voltage | 2.7 V – 3.6 V |
| SPI Clock Frequency | 104 MHz (for Std., Dual & Quad SPI), 208 / 416 MHZ (for Dual & Quad SPI) |
| Data Transfer Rate | ~50 MB per second |
| No. of Program / Erase Cycles | ≥ 100000 cycles |
| Data Retention Period | ≥ 20 years |
| Total Memory Capacity | 64 Mb / 8 MB |
| Block Memory Capacity | 128 blocks; 64 kB each |
| Sector Memory Capacity | 16 sectors per block; 4 kB each |
| Current Consumption | ~4 mA (active) and ≤ 1 µA (power – down) |
| Operating Temperature | - 40°C to + 85°C |

Pin Diagram: -

This is the 8 – pin SOIC / VSOC chip package of the NOR flash memory that will be interfaced with the ATMEGA32 MCU. (Taken from Page No. 6 of its datasheet)



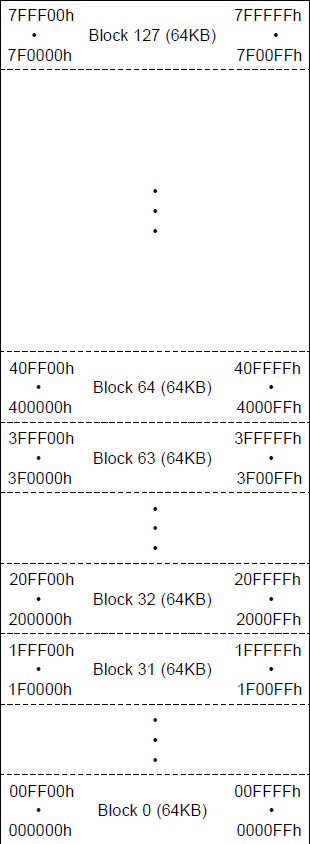
The pin descriptions are as follows: -

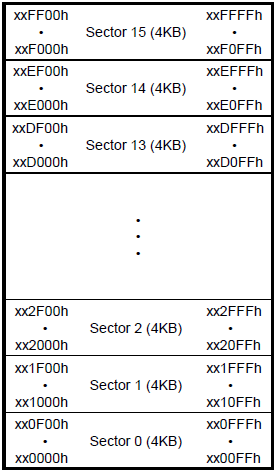
|  |  |  |
| --- | --- | --- |
| Pin Name | Pin Type (Input / Output) | Pin Description |
| VCC | ------ | Power Supply Pin (2.7 V – 3.6 V) |
| GND | ------ | Ground Pin |
| CLK | Input | SPI Serial Clock Pin |
| /CS | Input | Chip Select Pin |
| DI (IO0) | Input / Output | Data Input Pin (Data Input Output Pin 0) |
| DO (IO1) | Input / Output | Data Output Pin (Data Input Output Pin 1) |
| /WP (IO2) | Input / Output | Write Protect Pin (Data Input Output Pin 2) |
| /HOLD or /RESET (IO3) | Input / Output | Data Hold / Reset Pin (Data Input Output Pin 3) |

\*\* Pins for Standard SPI Interface are CLK, /CS, DI, DO, /WP, /HOLD or /RESET.

Pins for Dual SPI Interface are CLK, /CS, IO0, IO1, /WP, /HOLD or /RESET.

Pins for Quad SPI or QPI Interface are CLK, /CS, IO0, IO1, IO2, IO3.

The memory organization of W25Q64FWSIG NOR flash is shown in the following diagram: -



\*Diagrams have been taken from Page No. 9 of its datasheet.

* The 64 Mb memory is divided into 128 memory blocks i.e., block 0 to block 127 and each block has a memory size of 64 kB as seen in the left block diagram.
* The address range for each block goes from "xx 0000 h" to "xx FFFF h" where "xx" is the block number in hexadecimal format.
* Each memory block is further divided into 16 memory sectors i.e., sector 0 to sector 15 and each sector has a memory size of 4 kB as seen in the upper block diagram.
* The address range for each sector goes from "xx y 000 h" to "xx y FFF h" where "xx" is the block number in hexadecimal where the sector is present and "y" is the sector number in hexadecimal format.

Circuit Connections: -

W25Q64FWSIG NOR FLASH MEMORY MODULE

VIRTUAL

TERMINAL

USB

IN

USB

OUT

PD0 (RXD)

PD1 (TXD)

RXD

TXD

CLK

DO

DI

/CS

GND

VCC

VOUT-

VOUT+

GND

VIN+

VCC (+5V)

PB4 (SS)

PB5 (MOSI)

PB6 (MISO)

PB7 (SCK)

PC / COMPUTER / LAPTOP

5V to 3.3V BUCK CONVERTER

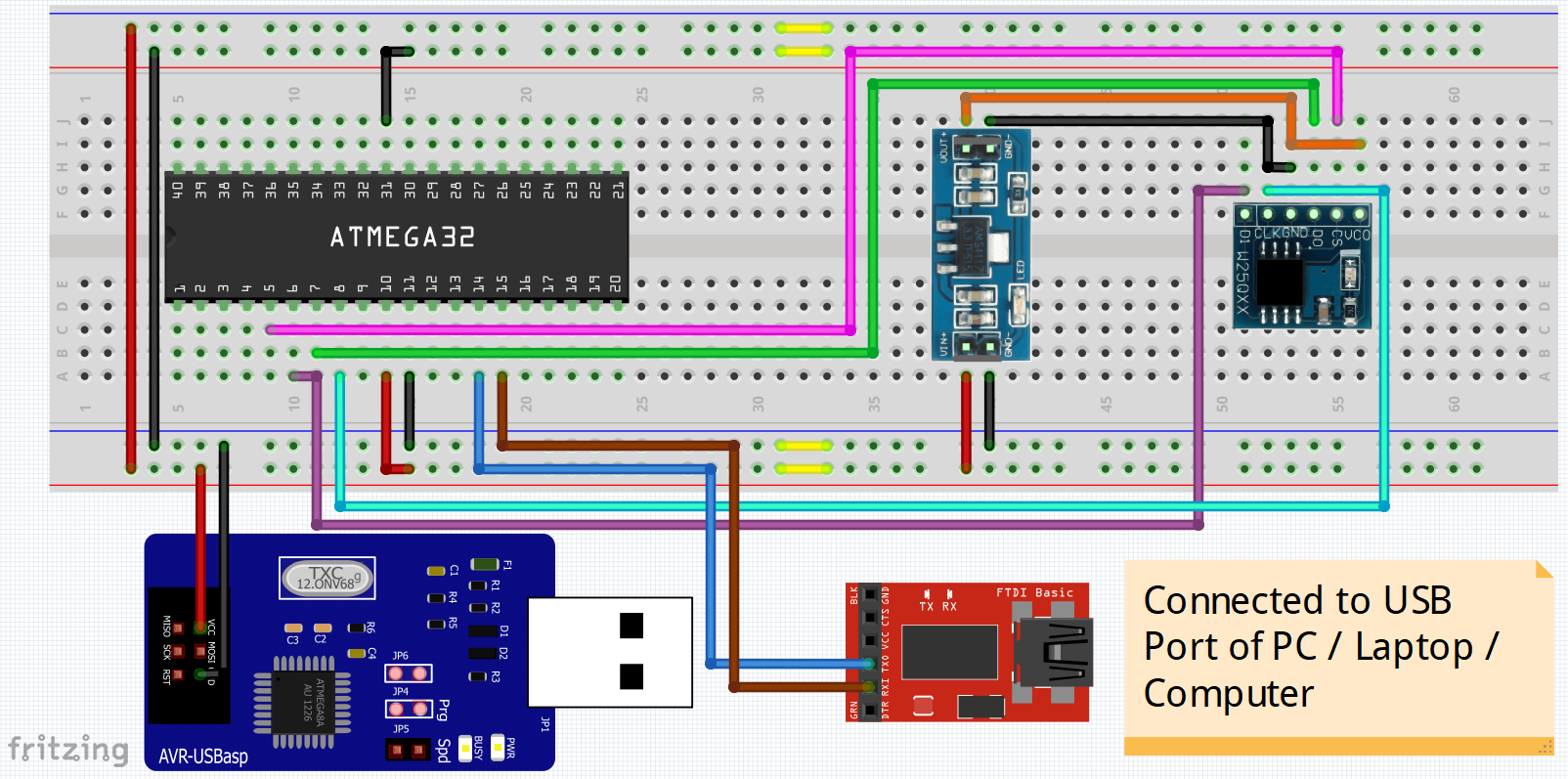
CP210X UART TO USB CONVERTER

ATMEGA32 MCU

VIN-

The block diagram for interfacing the W25Q64FWSIG Winbond NOR Flash with ATMEGA32 MCU is shown above. Since this module operates within 2.7 to 3.6 volts, a 5V to 3.3V buck converter is needed to provide power to the power pins of the module. Also, a UART to USB converter is required as all the data being extracted from the flash memory module will be displayed on the virtual terminal of the PC / computer / laptop to which it is connected.

The breadboard implementation of this circuit is shown in the following diagram: -



The pin connections are as follows: -

|  |  |  |
| --- | --- | --- |
| CONNECTIONS BETWEEN ATMEGA32 MCU & W25QXX NOR FLASH MEMORY | | |
| ATMEGA32 PIN | W25QXX PIN | WIRE COLOUR IN DIAGRAM |
| PB4 (SS) | /CS | Pink |
| PB5 (MOSI) | DI | Purple / Violet |
| PB6 (MISO) | DO | Green |
| PB7 (SCK) | CLK | Cyan / Light Blue |

|  |  |  |
| --- | --- | --- |
| CONNECTIONS BETWEEN ATMEGA32 MCU & UART TO USB FTDI CONVERTER | | |
| ATMEGA32 PIN | FTDI CONVERTER PIN | WIRE COLOUR IN DIAGRAM |
| PD0 (RXD) | TXD | Deep Blue |
| PD1 (TXD) | RXD | Brown |

* If the circuit is assembled on a breadboard, then the +5V power supply is obtained from the AVR – USB – asp programmer which is connected to the PC / Computer's USB port; but if an ATMEGA32 development board is used, then the +5V power supply is obtained by directly plugging the board with the USB port of PC / Computer.
* In the above diagram, all +5V power supply wires are shown in "red" colour and all the ground wires have been shown in "black" colour.
* The +3.3V power supply coming from the VOUT+ pin of buck converter to the VCC pin of W25QXX NOR flash memory has been shown in "orange" colour.
* The 5V to 3.3V buck converter is based on the chip AMS1117 which is a low drop-out voltage regulator IC.

Application Code and its Explanation: -

The application code that is written for interfacing W25Q64FWSIG NOR Flash Memory with ATMEGA32 MCU has the following objectives: -

* Obtain Device ID, Manufacturer ID and Unique ID of the W25Q64FWSIG NOR Flash Memory.
* Erasing information from the entire memory and understanding how to erase specific blocks and sectors of the memory.
* Writing data to and reading data from a particular flash memory address.

Since the entire source code is really lengthy, it will become very congested and tough to debug if it is all written in just one file. Thus, the entire application code is fragmented and written in 3 different files included in the same Atmel Studio Project folder. The code in these 3 files will be explained in 3 major parts as follows: -

PART – 1: Create a custom header file "w25qxxflash.h", where all the function prototypes, basic macro definitions, basic header files and the required standard SPI protocol commands to be given to the flash memory in order to achieve these objectives.

Following is the source code for "w25qxxflash.h" header file: -

Section 1

#ifndef \_\_W25QXXFLASH\_H\_\_

#define \_\_W25QXXFLASH\_H\_\_

Section 2

#define F\_CPU 8000000UL

Section 3

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#include <stdint.h>

Section 4

#define CS\_PIN (1 << 4)

#define CS\_ON PORTB &= ~CS\_PIN

#define CS\_OFF PORTB |= CS\_PIN

#define WR\_ENA\_CMD 0x06

Section 5

#define WR\_DIS\_CMD 0x04

#define RD\_SR1\_CMD 0x05

#define WR\_SR1\_CMD 0x01

#define RD\_SR2\_CMD 0x35

#define WR\_SR2\_CMD 0x31

#define RD\_SR3\_CMD 0x15

#define WR\_SR3\_CMD 0x11

#define CHIP\_ERASE 0xC7

#define JEDEC\_ID 0x9F

#define UNIQUE\_ID 0x4B

#define PAGE\_PROG 0x02

#define READ\_DATA 0x03

Section 6

void uart\_init();

void transmit(char a);

void uart\_send\_str(char \*str);

Section 7

void SPI\_Master\_Init();

void SPI\_Master\_Tx(char a);

char SPI\_Master\_Rx(char a);

Section 8

void get\_device\_specs();

void get\_status\_regs();

void write\_enable();

void write\_disable();

void write\_status\_regs(char sr1\_val, char sr2\_val, char sr3\_val);

void perf\_busy\_chk();

void erase\_flash();

void write\_flash(char addr1, char addr2, char addr3, char data);

void read\_flash(char addr1, char addr2, char addr3);

Section 9

#endif

The source code is explained in a section-wise manner as follows: -

|  |  |
| --- | --- |
| Section 1 | Here, the header file "w25qxxflash.h" definition is created. These 2 statements are always present at the beginning of any custom – made header file. |
| Section 2 | Here, the ATMEGA32 MCU's operating frequency is defined i.e., 8 MHz. |
| Section 3 | Here, the basic header files that are needed for the application are included. |
| Section 4 | Here, some macros are defined for activating and de-activating the chip select pin.   * CS\_PIN macro is used to represent the chip select pin on the MCU i.e., PB4 pin. * CS\_ON macro is used to activate the chip select pin by making the PORTB4 bit in the PORTB register as logic 0. * CS\_OFF macro is used to de-activate the chip select pin by making the PORTB4 bit in PORTB register as logic 1. |
| Section 5 | Here, macros are defined for the various standard SPI protocol commands needed to perform basic operations in the NOR flash memory like: -   * Extraction of device ID, manufacturer ID and unique ID. * Flash erase operation. * Performing data read and data write operations. * Enabling and disabling standard write operations. * Enabling and performing write operation on status registers 1, 2 and 3. * Reading data from status registers 1, 2 and 3.   All of these commands have been taken from the NOR flash memory's datasheet (pages 21 and 22). |
| Section 6 | Here, function prototypes have been defined for initializing UART communication in ATMEGA32 MCU at 9600 baud rate and for transmitting a single character and an entire string at a time to the serial terminal. |
| Section 7 | Here, function prototypes have been defined for initializing the ATMEGA32 MCU as the master device and to transmit and receive data from the NOR flash memory using standard SPI protocol. |
| Section 8 | Here, the function prototypes have been defined for performing the basic operations of NOR flash memory as have been outlined in the explanation of "Section 5". |
| Section 9 | Here, the header file source code is ended and this statement is always the last statement in a header file. |

PART – 2: Create a new C – program file by the name "w25qxxflash.c" where the entire function definition will be written for all the function prototypes present in "w25qxxflash.h" header file. The steps for implementing each function in this file will be explained below in relation with the source code.

The first step is to include the "w25qxxflash.h" header file and then the functions are defined.

A). UART RELATED FUNCTIONS: - The functions related to UART are explained as follows: -

uart\_init() – This function is used to initialize the UART communication of ATMEGA32 MCU at 9600 baud rate for 8 MHz operating frequency. The steps to implement this function are: -

1. Initialize UCSRA register to 0x00 in order to set all writable flags in this register to logic 0 (default value).
2. Feed UCSRB register with 0x18 in order to enable UART transmission and reception by setting the RXEN and TXEN bits at logic 1.
3. Feed UCSRC register with 0x06 in order to enable access to UBRRH register by making URSEL bit as logic 0 and selecting asynchronous mode of communication (UMSEL bit is made logic 0), without any parity check (UPM1 and UPM0 bits are made logic 0), having only 1 stop bit (USBS bit is made logic 0) and selecting 8 – bit word length (UCSZ2, UCSZ1 and UCSZ0 bits are given logic 0, logic 1 and logic 1 respectively).
4. Since for 8 MHz operating frequency of ATMEGA32 MCU and 9600 baud rate of communication, the baud rate register must take the value 51 i.e., UBRRH is given as 0x00 and UBRRL is given as 0x33.

The source code for this function as per the above steps is given as follows: -

void uart\_init()

{

UCSRA = 0x00;

UCSRB = 0x18;

UCSRC = 0x06;

UBRRH = 0x00;

UBRRL = 0x33;

}

transmit() – This function will take a single character as an input parameter and display that character on the serial terminal. The steps to implement this function are: -

* 1. Feed the character to be transmitted onto the UDR register.
  2. Monitor the TXC bit in UCSRA register and wait for it to become logic 1 which indicates that data transmission is successful.
  3. Give a time delay of 50 – 60 ms before transmission of next character in order to avoid data overrun.

The source code for this function as per the above steps is given as follows: -

void transmit(char a)

{

UDR = a;

while((UCSRA & 0x40) == 0);

\_delay\_ms(60);

}

uart\_send\_str() – This function takes character array in the form of a pointer as its input parameter and it displays that string on the serial terminal. The steps to implement this function are: -

1. Declare a while loop and set its execution condition to check if each character in that string is not equal to NULL character.
2. If the execution condition is true, then transmit that character to the serial terminal and check the next character by incrementing the pointer address by 1.
3. The loop keeps on executing until a NULL character is encountered in the character array and in this way, the entire string is displayed on the serial terminal in a character by character manner.

The source code for this function as per the above steps is given as follows: -

void uart\_send\_str(char \*str)

{

while((\*str) != '\0')

{

transmit(\*str);

str++;

}

}

B). SPI PROTOCOL RELATED FUNCTIONS: - The functions related to SPI protocol execution in ATMEGA32 MCU are explained as follows: -

SPI\_Master\_Init() – This function is used to setup the SPI protocol parameters in the ATMEGA32 MCU. The steps to implement this function are: -

1. De-activate the chip select line by using the "CS\_OFF" macro which will set the chip select pin at logic 1.
2. Feed 0x50 to the SPCR register which means: -

* SPI interrupt is disabled as SPIE bit is at logic 0.
* SPI communication based processes are enabled as SPE bit is at logic 1.
* The MSB of data will be shifted out of the SPI shift registers first i.e., MSB 1st – bit shifting sequence as DORD bit is at logic 0.
* ATMEGA32 MCU is set as the master device by making MSTR bit as logic 1.
* Idle state of SPI clock is logic 0 as CPOL bit is at logic 0.
* Data will be sampled on the rising edge and shifted out at the falling edge of each SPI clock pulse as CPHA bit is at logic 0.
* Since, SPR1 and SPR0 bits along with SPI2X bit in SPSR register are at logic 0 which means pre – scaler value for SPI clock frequency is 4 and hence, SPI clock frequency generated by the ATMEGA32 MCU is equal to 8 MHz / 4 i.e., 2 MHz.

The source code for this function as per the above steps is given as follows: -

void SPI\_Master\_Init()

{

CS\_OFF;

SPCR = 0x50;

}

SPI\_Master\_Tx() – This function takes a data / command as an input parameter and transmits it from the master device (ATMEGA32 MCU) to the slave device (NOR Flash Memory) using SPI protocol. The steps to implement this function are: -

1. Declare a character type variable "x" and initialize it to NULL value.
2. Feed the data / command to the SPDR register to send it from master to slave device.
3. Monitor SPIF bit in SPSR register and wait for it to become logic 1 which indicates that SPI data transmission has been completed successfully.
4. Flush the SPDR register by storing it in the variable 'x' that was declared in step 1.

The source code for this function as per the above steps is given as follows: -

void SPI\_Master\_Tx(char a)

{

char x = '\0';

SPDR = a;

while((SPSR & 0x80) == 0);

x = SPDR;

}

SPI\_Master\_Rx() – This function takes a dummy byte as an input parameter and sends it as a request by the master device (ATMEGA32 MCU) to receive data from the slave device (NOR Flash Memory) using SPI protocol and the received data from the slave is returned by this function. The steps to implement this function are: -

1. Feed dummy byte (usually 0x00) to the SPDR register in order to send it to the slave device for requesting data.
2. Monitor SPIF bit in SPSR register and wait for it to become logic 1 which indicates that SPI data reception has been completed successfully.
3. Return the data received from slave device which is present in the SPDR register.

The source code for this function as per the above steps is given as follows: -

char SPI\_Master\_Rx(char a)

{

SPDR = a;

while((SPSR & 0x80) == 0);

return SPDR;

}

C). NOR FLASH MEMORY RELATED FUNCTIONS: - The functions related to performing various operations and processes in NOR Flash Memory are explained as follows: -

get\_device\_specs() – This function is used for extracting the Manufacturer's ID, Device ID and Unique ID of the NOR Flash Memory. The steps to implement this function are: -

1. Declare a looping variable "i" and two character arrays i.e., "buf" and "tx\_buf" to store the data received from the slave and to store the string to be displayed on the serial terminal respectively.
2. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
3. Use SPI\_Master\_Tx() to send the "JEDEC\_ID" command i.e., 0x9F to the flash memory.
4. Receive 3 bytes of data from the flash memory using SPI\_Master\_Rx() and sending the dummy byte 0x00 and store them in the 1st 3 positions of "buf" array.
5. Deactivate chip select line using the "CS\_OFF" macro.
6. Display the Manufacturer and Device IDs on the serial terminal using uart\_send\_str() after storing it in the "tx\_buf" array using the "sprintf" function.
7. Empty both "buf" and "tx\_buf" arrays by assigning NULL character to their cells using "for" loop.
8. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
9. Send the "UNIQUE\_ID" command i.e., 0x4B to the flash memory using SPI\_Master\_Tx().
10. Use SPI\_Master\_Tx() to send 4 dummy bytes (0x00) to the flash memory using "for" loop.
11. Receive 8 bytes of data from the flash memory consecutively in a "for" loop, using SPI\_Master\_Rx() and dummy byte (0x00) and store it in the "buf" array.
12. Deactivate chip select line using the "CS\_OFF" macro.
13. Display the Unique ID on the serial terminal using uart\_send\_str() after storing it in the "tx\_buf" array using the "sprintf" function.
14. Empty both "buf" and "tx\_buf" arrays by assigning NULL character to their cells using "for" loop.

The source code for this function as per the above steps is given as follows: -

void get\_device\_specs()

{

int i;

char buf[8] = {'\0'}, tx\_buf[60] = {'\0'};

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(JEDEC\_ID);

buf[0] = SPI\_Master\_Rx(0x00);

buf[1] = SPI\_Master\_Rx(0x00);

buf[2] = SPI\_Master\_Rx(0x00);

CS\_OFF;

sprintf(tx\_buf, "Manufacturer ID: 0x%02X\n\rDevice ID: 0x%02X%02X\n\r", buf[0], buf[1], buf[2]);

uart\_send\_str(tx\_buf);

for(i = 0; i < 60; i++)

{

tx\_buf[i] = '\0';

}

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(UNIQUE\_ID);

for(i = 0; i < 4; i++)

{

SPI\_Master\_Tx(0x00);

}

for(i = 0; i < 8; i++)

{

buf[i] = SPI\_Master\_Rx(0x00);

}

CS\_OFF;

sprintf(tx\_buf, "Unique ID: 0x%02X%02X%02X%02X%02X%02X%02X%02X\n\r\n\r",

buf[0], buf[1], buf[2], buf[3], buf[4], buf[5], buf[6], buf[7]);

uart\_send\_str(tx\_buf);

for(i = 0; i < 60; i++)

{

tx\_buf[i] = '\0';

}

}

get\_status\_regs() – This function will get the values present in all the 3 status registers of the flash memory. These status registers will show the status of various operations that are happening in the flash memory along with the condition of all memory blocks and sectors present in it. The steps to implement this function are: -

1. Declare two character arrays i.e., "buf" and "tx\_buf" to store the data received from the slave and to store the string to be displayed on the serial terminal respectively.
2. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
3. Use SPI\_Master\_Tx() to send the "RD\_SR1\_CMD" command i.e., 0x05 to the flash memory.
4. Receive 1 byte of data from the flash memory using SPI\_Master\_Rx() and sending the dummy byte (0x00) and store it in the 1st position of "buf" array.
5. Deactivate chip select line using the "CS\_OFF" macro and then, provide a 300 ms time delay.
6. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
7. Use SPI\_Master\_Tx() to send the "RD\_SR2\_CMD" command i.e., 0x35 to the flash memory.
8. Receive 1 byte of data from the flash memory using SPI\_Master\_Rx() and sending the dummy byte (0x00) and store it in the 2nd position of "buf" array.
9. Deactivate chip select line using the "CS\_OFF" macro and then, provide a 300 ms time delay.
10. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
11. Use SPI\_Master\_Tx() to send the "RD\_SR3\_CMD" command i.e., 0x15 to the flash memory.
12. Receive 1 byte of data from the flash memory using SPI\_Master\_Rx() and sending the dummy byte (0x00) and store it in the 3rd position of "buf" array.
13. Deactivate chip select line using the "CS\_OFF" macro and then, provide a 300 ms time delay.
14. Display the 3 status register values on the serial terminal using uart\_send\_str() after storing it in the "tx\_buf" array using the "sprintf" function.
15. Empty both "buf" and "tx\_buf" arrays by assigning NULL character to their cells using "for" loop.

The source code for this function as per the above steps is given as follows: -

void get\_status\_regs()

{

char buf[3] = {'\0'}, tx\_buf[60] = {'\0'};

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(RD\_SR1\_CMD);

buf[0] = SPI\_Master\_Rx(0x00);

CS\_OFF;

\_delay\_ms(300);

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(RD\_SR2\_CMD);

buf[1] = SPI\_Master\_Rx(0x00);

CS\_OFF;

\_delay\_ms(300);

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(RD\_SR3\_CMD);

buf[2] = SPI\_Master\_Rx(0x00);

CS\_OFF;

\_delay\_ms(300);

sprintf(tx\_buf, "SR1: 0x%02X SR2: 0x%02X SR3: 0x%02X\n\r\n\r", buf[0], buf[1], buf[2]);

uart\_send\_str(tx\_buf);

for(int i = 0; i < 60; i++)

{

tx\_buf[i] = '\0';

}

}

write\_enable() – This function enables the flash memory to allow the MCU to write / modify / erase data from its memory locations. The steps to implement this function are: -

1. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
2. Use SPI\_Master\_Tx() to send the "WR\_ENA\_CMD" command i.e., 0x06 to the flash memory.
3. Deactivate chip select line using the "CS\_OFF" macro.

The source code for this function as per the above steps is given as follows: -

void write\_enable()

{

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(WR\_ENA\_CMD);

CS\_OFF;

}

write\_disable() – This function causes the flash memory to prevent the MCU to write / modify / erase data from its memory locations. The steps to implement this function are: -

1. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
2. Use SPI\_Master\_Tx() to send the "WR\_DIS\_CMD" command i.e., 0x04 to the flash memory.
3. Deactivate chip select line using the "CS\_OFF" macro.

The source code for this function as per the above steps is given as follows: -

void write\_disable()

{

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(WR\_DIS\_CMD);

CS\_OFF;

}

write\_status\_regs() – This function will allow the MCU to write / modify the values of all 3 status registers of the flash memory. The 3 values to be written to these registers will be passed as input parameters to this function. The steps to implement this function are: -

1. Enable write operations in flash memory using "write\_enable()" and provide a time delay of 300 ms.
2. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
3. Use SPI\_Master\_Tx() to send the "WR\_SR1\_CMD" command i.e., 0x01 to the flash memory.
4. Use SPI\_Master\_Tx() to send the value to be written to status register 1 to the flash memory.
5. Deactivate chip select line using the "CS\_OFF" macro and then, provide a 300 ms time delay.
6. Disable write operations in flash memory using "write\_disable()" and provide a time delay of 300 ms.
7. Enable write operations in flash memory using "write\_enable()" and provide a time delay of 300 ms.
8. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
9. Use SPI\_Master\_Tx() to send the "WR\_SR2\_CMD" command i.e., 0x31 to the flash memory.
10. Use SPI\_Master\_Tx() to send the value to be written to status register 2 to the flash memory.
11. Deactivate chip select line using the "CS\_OFF" macro and then, provide a 300 ms time delay.
12. Disable write operations in flash memory using "write\_disable()" and provide a time delay of 300 ms.
13. Enable write operations in flash memory using "write\_enable()" and provide a time delay of 300 ms.
14. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
15. Use SPI\_Master\_Tx() to send the "WR\_SR3\_CMD" command i.e., 0x11 to the flash memory.
16. Use SPI\_Master\_Tx() to send the value to be written to status register 3 to the flash memory.
17. Deactivate chip select line using the "CS\_OFF" macro and then, provide a 300 ms time delay.
18. Disable write operations in flash memory using "write\_disable()" and provide a time delay of 300 ms.

The source code for this function as per the above steps is given as follows: -

void write\_status\_regs(char sr1\_val, char sr2\_val, char sr3\_val)

{

write\_enable();

\_delay\_ms(300);

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(WR\_SR1\_CMD);

SPI\_Master\_Tx(sr1\_val);

CS\_OFF;

\_delay\_ms(300);

write\_disable();

\_delay\_ms(300);

write\_enable();

\_delay\_ms(300);

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(WR\_SR2\_CMD);

SPI\_Master\_Tx(sr2\_val);

CS\_OFF;

\_delay\_ms(300);

write\_disable();

\_delay\_ms(300);

write\_enable();

\_delay\_ms(300);

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(WR\_SR3\_CMD);

SPI\_Master\_Tx(sr3\_val);

CS\_OFF;

\_delay\_ms(300);

write\_disable();

\_delay\_ms(300);

}

perf\_busy\_chk() – This function is used to check whether the flash memory is busy with some ongoing operation or not. This busy check must be done in order to know if the flash memory is ready to accept and execute any new instructions coming from the MCU. The steps to implement this function are: -

1. Declare a character variable "buf" which will hold the status register 1 value and a character array "tx\_buf" which will hold the string to be displayed on the serial terminal and also declare a looping variable "i".
2. Begin the "do – while" loop to perform busy check operation.
3. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
4. Use SPI\_Master\_Tx() to send the "RD\_SR1\_CMD" command i.e., 0x05 to the flash memory.
5. Receive 1 byte of data from the flash memory using SPI\_Master\_Rx() and sending the dummy byte (0x00) and store it in the "buf" variable.
6. Deactivate chip select line using the "CS\_OFF" macro.
7. Display the status register 1 value on the serial terminal using uart\_send\_str() after storing it in the "tx\_buf" array using the "sprintf" function.
8. Empty "tx\_buf" array by assigning NULL character to their cells using "for" loop.
9. Provide a time delay of 1000 ms or 1 second.
10. Terminate "do – while" loop with the condition to check if the last bit of the value present in "buf" variable is equal to 1 or not.
11. Send newline and carriage return characters to the serial terminal.

The source code for this function as per the above steps is given as follows: -

void perf\_busy\_chk()

{

char buf = '\0', tx\_buf[20] = {'\0'};

do

{

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(RD\_SR1\_CMD);

buf = SPI\_Master\_Rx(0x00);

CS\_OFF;

sprintf(tx\_buf, "SR1: 0x%02X\n\r", buf);

uart\_send\_str(tx\_buf);

for(int i = 0; i < 20; i++)

{

tx\_buf[i] = '\0';

}

\_delay\_ms(1000);

} while((buf & 0x01) == 0x01);

uart\_send\_str("\n\r");

}

erase\_flash() – This function is used to erase data from the entire flash memory. The steps to implement this function are: -

1. Enable write operations in flash memory using "write\_enable()" and provide a time delay of 300 ms.
2. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
3. Use SPI\_Master\_Tx() to send the "CHIP\_ERASE" command i.e., 0xC7 to the flash memory.
4. Deactivate chip select line using the "CS\_OFF" macro.
5. Disable write operations in flash memory using "write\_disable()" and provide a time delay of 300 ms.
6. Check whether the flash memory is busy erasing data or not using "perf\_busy\_chk()".
7. Display message on serial terminal to indicate that flash memory has been fully erased.

The source code for this function as per the above steps is given as follows: -

void erase\_flash()

{

uart\_send\_str("Erasing flash......\n\r\n\r");

write\_enable();

\_delay\_ms(300);

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(CHIP\_ERASE);

CS\_OFF;

write\_disable();

\_delay\_ms(300);

uart\_send\_str("Performing busy check......\n\r\n\r");

\_delay\_ms(300);

perf\_busy\_chk();

uart\_send\_str("Flash Erased\n\r\n\r");

}

read\_flash() – This function is used to read data from a particular address in the flash memory and display it on the serial terminal. The memory address from which the data has to be read is sent as input parameters to this function in 3 parts i.e., the higher 8 bits, the middle 8 bits and the lower 8 bits. The steps to implement this function are: -

1. Declare an integer type looping variable "i" and a character type variable "data" to store the data present in the flash memory address and a character array "tx\_buf" to store the string to be displayed on the serial terminal.
2. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
3. Use SPI\_Master\_Tx() to send the "READ\_DATA" command i.e., 0x03 to the flash memory.
4. Use SPI\_Master\_Tx() to send the higher 8 bits of address to the flash memory.
5. Use SPI\_Master\_Tx() to send the middle 8 bits of address to the flash memory.
6. Use SPI\_Master\_Tx() to send the lower 8 bits of address to the flash memory.
7. Receive 1 byte of data from the flash memory using SPI\_Master\_Rx() and sending the dummy byte (0x00) and store it in the "data" variable.
8. Deactivate chip select line using the "CS\_OFF" macro.
9. Display the data read from the flash memory address on the serial terminal using uart\_send\_str() after storing it in the "tx\_buf" array using the "sprintf" function.
10. Empty "tx\_buf" array by assigning NULL character to their cells using "for" loop.

The source code for this function as per the above steps is given as follows: -

void read\_flash(char addr1, char addr2, char addr3)

{

int i;

char tx\_buf[60], data;

uart\_send\_str("Reading data from flash.....\n\r\n\r");

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(READ\_DATA);

SPI\_Master\_Tx(addr1);

SPI\_Master\_Tx(addr2);

SPI\_Master\_Tx(addr3);

data = SPI\_Master\_Rx(0x00);

CS\_OFF;

sprintf(tx\_buf, "Data read at 0x%02X%02X%02X address: 0x%02X\n\r\n\r",

addr1, addr2, addr3, data);

uart\_send\_str(tx\_buf);

for(i = 0; i < 60; i++)

{

tx\_buf[i] = '\0';

}

}

write\_flash() – This function is used to write or modify data at a particular address of the flash memory. The memory address along with the data to be written are sent as input parameters to this function in 4 parts i.e., the higher 8 bits, the middle 8 bits and the lower 8 bits and data to be written. The steps to implement this function are: -

1. Declare a character array "tx\_buf" to store the string to be displayed on the serial terminal.
2. Enable write operations in flash memory using "write\_enable()" and provide a time delay of 300 ms.
3. Activate chip select pin using the "CS\_ON" macro and then, provide a 100 ms time delay.
4. Use SPI\_Master\_Tx() to send the "PAGE\_PROG" command i.e., 0x02 to the flash memory.
5. Use SPI\_Master\_Tx() to send the higher 8 bits of address to the flash memory.
6. Use SPI\_Master\_Tx() to send the middle 8 bits of address to the flash memory.
7. Use SPI\_Master\_Tx() to send the lower 8 bits of address to the flash memory.
8. Use SPI\_Master\_Tx() to send the data to be written to the flash memory.
9. Deactivate chip select line using the "CS\_OFF" macro.
10. Disable write operations in flash memory using "write\_disable()" and provide a time delay of 300 ms.
11. Display a message on the serial terminal to indicate the completion of data write process, using uart\_send\_str() after storing it in the "tx\_buf" array using the "sprintf" function.
12. Empty "tx\_buf" array by assigning NULL character to their cells using "for" loop.

The source code for this function as per the above steps is given as follows: -

void write\_flash(char addr1, char addr2, char addr3, char data)

{

char tx\_buf[40] = {'\0'};

uart\_send\_str("Writing data to flash.....");

write\_enable();

\_delay\_ms(300);

CS\_ON;

\_delay\_ms(100);

SPI\_Master\_Tx(PAGE\_PROG);

SPI\_Master\_Tx(addr1);

SPI\_Master\_Tx(addr2);

SPI\_Master\_Tx(addr3);

SPI\_Master\_Tx(data);

CS\_OFF;

write\_disable();

\_delay\_ms(300);

sprintf(tx\_buf, "Write Successful at 0x%02X%02X%02X\n\r\n\r", addr1, addr2, addr3);

uart\_send\_str(tx\_buf);

for(int i = 0; i < 40; i++)

{

tx\_buf[i] = '\0';

}

}

PART – 3: Finally, after defining the "w25qxxflash.h" and "w25qxxflash.c" files, it is time to write the main program in the "main.c" file. The steps to write the source code in this file are as follows: -

1. Include "w25qxxflash.h" header file.
2. Start "main()" with integer return type.
3. Set pin directions by configuring the DDRB and DDRD registers. Here, the MCU pins PB4, PB5, PB7 and PD1 pins are configured as output pins whereas MCU pins PB6 and PD0 are configured as input pins.
4. Initialize PORTB and PORTD registers to 0x00 in order to turn all pins off.
5. Setup UART communication by calling "uart\_init()".
6. Setup SPI communication by calling "SPI\_Master\_Init()".
7. Display flash memory specifications by calling "get\_device\_specs()".
8. Display the values of the 3 status registers by calling "get\_status\_regs()".
9. Reset all 3 status registers to their default values i.e., 0x00, 0x00 and 0x60 respectively using the "write\_status\_regs()".
10. Display the values of the 3 status registers (again after resetting them) by calling "get\_status\_regs()".
11. Erase all previous content in the flash memory by calling the "erase\_flash()".
12. Write a data to a specific address in the flash memory by calling the "write\_flash()" and passing the necessary parameters to it (as mentioned previously).
13. After a time delay of 3000 ms i.e., 3 seconds, read the data from that flash memory address and display it on the serial terminal by calling the "read\_flash()" and passing the necessary parameters as input to it.
14. Terminate "main()" with self – centralized, infinite "while" loop.

The source code for this function as per the above steps is given as follows: -

#include "w25qxxflash.h"

int main()

{

DDRB = 0xB0; DDRD = 0xFE;

PORTB &= 0x00; PORTD &= 0x00;

uart\_init();

SPI\_Master\_Init();

uart\_send\_str("W25QXX NOR FLASH Testing....\n\r\n\r");

get\_device\_specs();

get\_status\_regs();

uart\_send\_str("Resetting SR1, SR2, SR3.....");

write\_status\_regs(0x00, 0x00, 0x60);

uart\_send\_str("SR1, SR2, SR3 reset\n\r\n\r");

get\_status\_regs();

erase\_flash();

write\_flash(0x12, 0x11, 0x17, 0xBA);

\_delay\_ms(3000);

read\_flash(0x12, 0x11, 0x17);

while(1);

}

NOTES: Some important points that must be understood while programming the flash memory are mentioned as follows: -

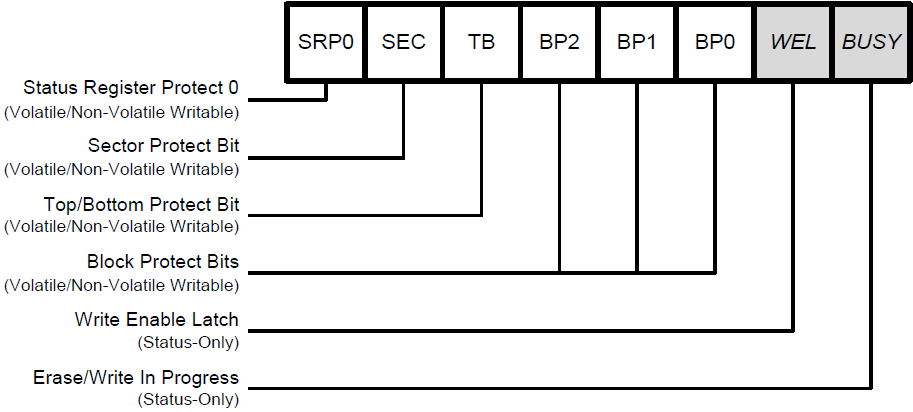
* While downloading / burning the program into ATMEGA32 MCU, it must be made sure that the flash memory is disconnected from the MCU pins because the program is fed to the ATMEGA32 MCU via the SPI interface.
* Failure to comply with the above precaution might cause unwanted changes in the configuration bits of the 3 status registers which could cause the memory blocks to get protected thereby, preventing any kind of write / modify / erase data operations in the flash memory.
* It is due to the above reason that before executing the "erase\_flash()", it is important to read and reset the values of the 3 status registers to their default values in order to remove any kind of memory protection and clear unwanted status conditions.
* Erasing data from the entire flash memory usually takes 15 – 20 seconds.
* After data has been erased from the flash memory, the default value that will be present in all memory locations is 0xFF; so this can be used to check whether the flash memory data has been properly erased or not.
* After the "erase\_flash()" is executed, if any value other than 0xFF is seen in any flash memory location, then the erase operation was not successful and it needs to be done again.
* The entire program's output is shown in the screenshot below: -



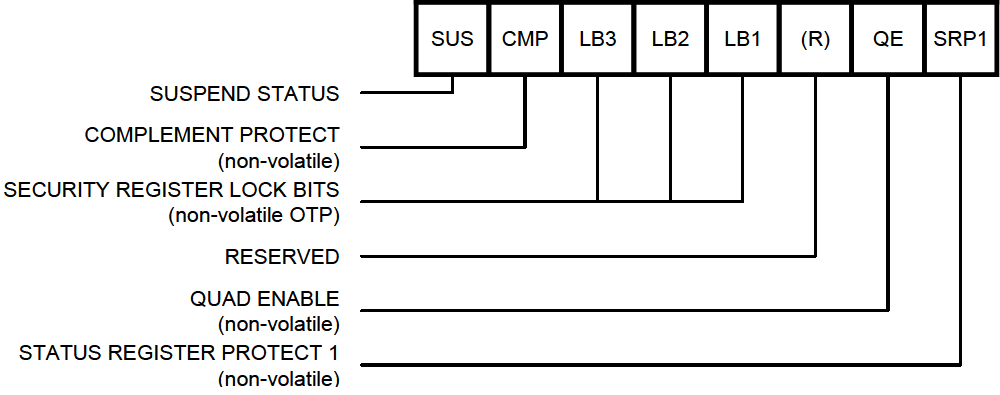
* If any of the above values are not coming properly, then check the hardware connections and make sure that they are all tight and secure because a slight loose connection can provide incorrect data on the serial terminal. The code is fully functional and tested but can be improved further.
* The detailed working of all standard SPI protocol commands are provided in the W25Q64FWSIG NOR Flash Memory's datasheet (pages 22 and 23).

STATUS REGISTERS IN W25Q64FWSIG NOR FLASH MEMORY: -

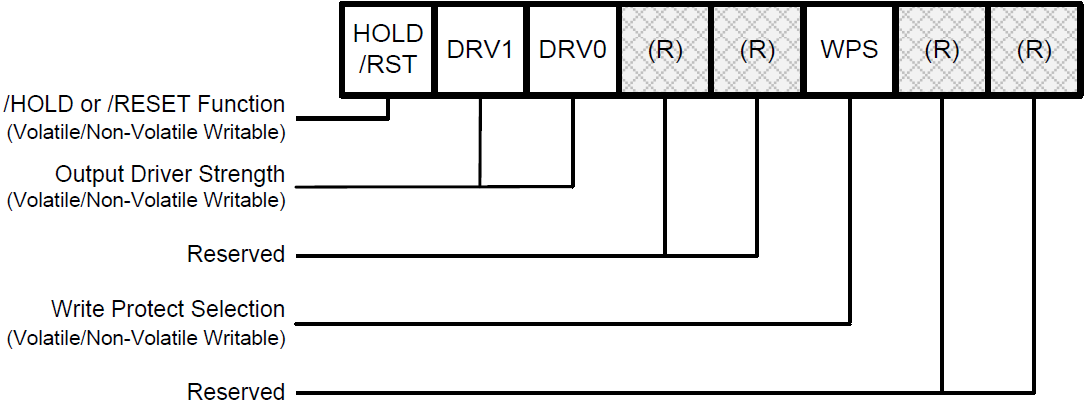
* The W25Q64FWSIG NOR Flash Memory contains 3 status registers which contain various configuration bits for memory protection and also some status bits to show the state of certain operations happening in it.
* The 3 status registers are shown below: -



STATUS REGISTER 1



STATUS REGISTER 2



STATUS REGISTER 3

* Some bits in these status registers are volatile, some other bits are non – volatile and certain other bits are both volatile and non – volatile in nature based on some conditions.
* Volatile bits will lose their value after power – on reset but non – volatile bits will retain their value after a power – on reset and some bits are status bits which are read – only bits.
* The bits of these registers and their descriptions are provided below: -

A). Erase/Write In Progress (BUSY) – Status Only

* BUSY is a read only bit in the status register (S0) that is set to a 1 state when the device is executing a Page Program, Quad Page Program, Sector Erase, Block Erase, Chip Erase, Write Status Register or Erase/Program Security Register instruction.
* During this time the device will ignore further instructions except for the Read Status Register and Erase/Program Suspend instruction.
* When the program, erase or write status/security register instruction has completed, the BUSY bit will be cleared to a 0 state indicating the device is ready for further instructions.

B). Write Enable Latch (WEL) – Status Only

* Write Enable Latch (WEL) is a read only bit in the status register (S1) that is set to 1 after executing a Write Enable Instruction.
* The WEL status bit is cleared to 0 when the device is write disabled.
* A write disable state occurs upon power-up or after any of the following instructions: Write Disable, Page Program, Quad Page Program, Sector Erase, Block Erase, Chip Erase, Write Status Register, Erase Security Register and Program Security Register.

C). Block Protect Bits (BP2, BP1, BP0) – Volatile/Non-Volatile Writable

* The Block Protect Bits (BP2, BP1, BP0) are non-volatile read/write bits in the status register (S4, S3, and S2) that provide Write Protection control and status.
* Block Protect bits can be set using the Write Status Register Instruction.
* All, none or a portion of the memory array can be protected from Program and Erase instructions.
* The factory default setting for the Block Protection Bits is 0, none of the array protected.

D). Top/Bottom Block Protect (TB) – Volatile/Non-Volatile Writable

* The non-volatile Top/Bottom bit (TB) controls if the Block Protect Bits (BP2, BP1, BP0) protect from the Top (TB = 0) or the Bottom (TB = 1) of the array as shown in the Status Register Memory Protection table.
* The factory default setting is TB = 0. The TB bit can be set with the Write Status Register Instruction depending on the state of the SRP0, SRP1 and WEL bits.

E). Sector/Block Protect Bit (SEC) – Volatile/Non-Volatile Writable

* The non-volatile Sector/Block Protect bit (SEC) controls if the Block Protect Bits (BP2, BP1, BP0) protect either 4KB Sectors (SEC = 1) or 64KB Blocks (SEC = 0) in the Top (TB = 0) or the Bottom (TB = 1) of the array.
* The default setting is SEC = 0.

F). Complement Protect (CMP) – Volatile/Non-Volatile Writable

* The Complement Protect bit (CMP) is a non-volatile read/write bit in the status register (S14).
* It is used in conjunction with SEC, TB, BP2, BP1 and BP0 bits to provide more flexibility for the array protection.
* Once CMP is set to 1, previous array protection set by SEC, TB, BP2, BP1 and BP0 will be reversed.
* For example, when CMP = 0, a top 64KB block can be protected while the rest of the array is not; when CMP = 1, the top 64KB block will become unprotected while the rest of the array become read-only.
* The default setting is CMP = 0.

G). Status Register Protect (SRP1, SRP0) – Volatile/Non-Volatile Writable

* The Status Register Protect bits (SRP1 and SRP0) are non-volatile read/write bits in the status register (S8 and S7).
* The SRP bits control the method of write protection: software protection, hardware protection, power supply lock-down or one time programmable (OTP) protection.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SRP1 | SRP0 | /WP | Status Register | Description |
| 0 | 0 | X | Software Protection | /WP pin has no control. The Status register can be written to after a Write Enable instruction, WEL = 1. [Factory Default] |
| 0 | 1 | 0 | Hardware Protected | When /WP pin is low the Status Register locked and cannot be written to. |
| 0 | 1 | 1 | Hardware Unprotected | When /WP pin is high the Status register is unlocked and can be written to after a Write Enable instruction, WEL = 1. |
| 1 | 0 | X | Power Supply Lock-Down | Status Register is protected and cannot be written to again until the next power-down, power-up cycle. |
| 1 | 1 | X | One Time Program | Status Register is permanently protected and cannot be written to. |

H). Erase/Program Suspend Status (SUS) – Status Only

* The Suspend Status bit is a read only bit in the status register (S15) that is set to 1 after executing an Erase/Program Suspend (75h) instruction.
* The SUS status bit is cleared to 0 by Erase/Program Resume (7Ah) instruction as well as a power-down, power-up cycle.

I). Security Register Lock Bits (LB3, LB2, LB1) – Volatile/Non-Volatile OTP Writable

* The Security Register Lock Bits (LB3, LB2, LB1) are non-volatile One Time Program (OTP) bits in Status Register (S13, S12, S11) that provide the write protect control and status to the Security Registers.
* The default state of LB3-1 is 0, Security Registers are unlocked. LB3-1 can be set to 1 individually using the Write Status Register instruction.
* LB3-1 are One Time Programmable (OTP), once it’s set to 1, the corresponding 256-Byte Security Register will become read-only permanently.

J). Quad Enable (QE) – Volatile/Non-Volatile Writable

* The Quad Enable (QE) bit is a non-volatile read/write bit in the status register (S9) that allows Quad SPI and QPI operation.
* When the QE bit is set to a 0 state (factory default for part number with ordering options “IG” and “IF”), the /WP pin and /HOLD are enabled.
* When the QE bit is set to a 1(factory default for Quad Enabled part numbers with ordering option “IQ”), the Quad IO2 and IO3 pins are enabled, and /WP and /HOLD functions are disabled.
* QE bit is required to be set to a 1 before issuing an “Enter QPI (38h)” to switch the device from Standard/Dual/Quad SPI to QPI, otherwise the command will be ignored.
* When the device is in QPI mode, QE bit will remain to be 1. A “Write Status Register” command in QPI mode cannot change QE bit from a “1” to a “0”.
* If the /WP or /HOLD pins are tied directly to the power supply or ground during standard SPI or Dual SPI operation, the QE bit should never be set to a 1.

K). Write Protect Selection (WPS) – Volatile/Non-Volatile Writable

* The WPS bit is used to select which Write Protect scheme should be used.
* When WPS = 0, the device will use the combination of CMP, SEC, TB, BP[2:0] bits to protect a specific area of the memory array.
* When WPS = 1, the device will utilize the Individual Block Locks to protect any individual sector or blocks.
* The default value for all Individual Block Lock bits is 1 upon device power on or after reset.

L). Output Driver Strength (DRV1, DRV0) – Volatile/Non-Volatile Writable

* The DRV1 & DRV0 bits are used to determine the output driver strength for the Read operations.

|  |  |
| --- | --- |
| DRV1, DRV0 | Driver Strength |
| 0, 0 | 100% |
| 0, 1 | 75% |
| 1, 0 | 50% |
| 1, 1 | 25% (default) |

M). /HOLD or /RESET Pin Function (HOLD/RST) – Volatile/Non-Volatile Writable

* The HOLD/RST bit is used to determine whether /HOLD or /RESET function should be implemented on the hardware pin for 8-pin packages.
* When HOLD/RST = 0 (factory default), the pin acts as /HOLD; when HOLD/RST = 1, the pin acts as /RESET.
* However, /HOLD or /RESET functions are only available when QE = 0. If QE is set to 1, the /HOLD and /RESET functions are disabled, the pin acts as a dedicated data I/O pin.

N). Reserved Bits – Non Functional

* There are a few reserved Status Register bits that may be read out as a “0” or “1”. It is recommended to ignore the values of those bits.
* During a “Write Status Register” instruction, the Reserved Bits can be written as “0”, but there will not be any effects.
* So, from the above discussion, it is seen that status registers 1 and 2 are mainly associated with memory protection and only 1 bit of status register 3 is involved in this as well.
* The memory protection bits in status register 1 are SEC, TB, BP2, BP1 and BP0 whereas the memory protection bits in status registers 2 and 3 are CMP and WPS respectively.
* Thus, various memory protection schemes can be created for the flash memory by using different combinations of these 7 bits and these memory protection schemes are shown in the following tables: -

Table A: W25Q64FW Status Register Memory Protection (WPS = 0, CMP = 0): -

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| STATUS REGISTER | | | | | W25Q64FW (64M-BIT) MEMORY PROTECTION | | | | |
| SEC | TB | BP2 | BP1 | BP0 | PROTECTED BLOCK(S) | PROTECTED ADDRESSES | PROTECTED DENSITY | PROTECTED PORTION |
| X | X | 0 | 0 | 0 | NONE | NONE | NONE | NONE |
| 0 | 0 | 0 | 0 | 1 | 126 and 127 | 7E0000h – 7FFFFFh | 128KB | Upper 1/64 |
| 0 | 0 | 0 | 1 | 0 | 124 thru 127 | 7C0000h – 7FFFFFh | 256KB | Upper 1/32 |
| 0 | 0 | 0 | 1 | 1 | 120 thru 127 | 780000h – 7FFFFFh | 512KB | Upper 1/16 |
| 0 | 0 | 1 | 0 | 0 | 112 thru 127 | 700000h – 7FFFFFh | 1MB | Upper 1/8 |
| 0 | 0 | 1 | 0 | 1 | 96 thru 127 | 600000h – 7FFFFFh | 2MB | Upper 1/4 |
| 0 | 0 | 1 | 1 | 0 | 64 thru 127 | 400000h – 7FFFFFh | 4MB | Upper 1/2 |
| 0 | 1 | 0 | 0 | 1 | 0 and 1 | 000000h – 01FFFFh | 128KB | Lower 1/64 |
| 0 | 1 | 0 | 1 | 0 | 0 thru 3 | 000000h – 03FFFFh | 256KB | Lower 1/32 |
| 0 | 1 | 0 | 1 | 1 | 0 thru 7 | 000000h – 07FFFFh | 512KB | Lower 1/16 |
| 0 | 1 | 1 | 0 | 0 | 0 thru 15 | 000000h – 0FFFFFh | 1MB | Lower 1/8 |
| 0 | 1 | 1 | 0 | 1 | 0 thru 31 | 000000h – 1FFFFFh | 2MB | Lower 1/4 |
| 0 | 1 | 1 | 1 | 0 | 0 thru 63 | 000000h – 3FFFFFh | 4MB | Lower 1/2 |
| X | X | 1 | 1 | 1 | 0 thru 127 | 000000h – 7FFFFFh | 8MB | ALL |
| 1 | 0 | 0 | 0 | 1 | 127 | 7FF000h – 7FFFFFh | 4KB | U – 1/2048 |
| 1 | 0 | 0 | 1 | 0 | 127 | 7FE000h – 7FFFFFh | 8KB | U – 1/1024 |
| 1 | 0 | 0 | 1 | 1 | 127 | 7FC000h – 7FFFFFh | 16KB | U – 1/512 |
| 1 | 0 | 1 | 0 | X | 127 | 7F8000h – 7FFFFFh | 32KB | U – 1/256 |
| 1 | 1 | 0 | 0 | 1 | 0 | 000000h – 000FFFh | 4KB | L – 1/2048 |
| 1 | 1 | 0 | 1 | 0 | 0 | 000000h – 001FFFh | 8KB | L – 1/1024 |
| 1 | 1 | 0 | 1 | 1 | 0 | 000000h – 003FFFh | 16KB | L – 1/512 |
| 1 | 1 | 1 | 0 | X | 0 | 000000h – 007FFFh | 32KB | L – 1/256 |

Table B: W25Q64FW Status Register Memory Protection (WPS = 0, CMP = 1): -

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| STATUS REGISTER | | | | | W25Q64FW (64M-BIT) MEMORY PROTECTION | | | | |
| SEC | TB | BP2 | BP1 | BP0 | PROTECTED BLOCK(S) | PROTECTED ADDRESSES | PROTECTED DENSITY | PROTECTED PORTION |
| X | X | 0 | 0 | 0 | 0 thru 127 | 000000h – 7FFFFFh | 8MB | ALL |
| 0 | 0 | 0 | 0 | 1 | 0 thru 125 | 000000h – 7DFFFFh | 8,064KB | Lower 63/64 |
| 0 | 0 | 0 | 1 | 0 | 0 thru 123 | 000000h – 7BFFFFh | 7,936KB | Lower 31/32 |
| 0 | 0 | 0 | 1 | 1 | 0 thru 119 | 000000h – 77FFFFh | 7,680KB | Lower 15/16 |
| 0 | 0 | 1 | 0 | 0 | 0 thru 111 | 000000h – 6FFFFFh | 7MB | Lower 7/8 |
| 0 | 0 | 1 | 0 | 1 | 0 thru 95 | 000000h – 5FFFFFh | 5MB | Lower 3/4 |
| 0 | 0 | 1 | 1 | 0 | 0 thru 63 | 000000h – 3FFFFFh | 4MB | Lower 1/2 |
| 0 | 1 | 0 | 0 | 1 | 2 thru 127 | 020000h – 7FFFFFh | 8,064KB | Upper 63/64 |
| 0 | 1 | 0 | 1 | 0 | 4 thru 127 | 040000h – 7FFFFFh | 7,936KB | Upper 31/32 |
| 0 | 1 | 0 | 1 | 1 | 8 thru 127 | 080000h – 7FFFFFh | 7,680KB | Upper 15/16 |
| 0 | 1 | 1 | 0 | 0 | 16 thru 127 | 100000h – 7FFFFFh | 7MB | Upper 7/8 |
| 0 | 1 | 1 | 0 | 1 | 32 thru 127 | 200000h – 7FFFFFh | 5MB | Upper 3/4 |
| 0 | 1 | 1 | 1 | 0 | 64 thru 127 | 400000h – 7FFFFFh | 4MB | Upper 1/2 |
| X | X | 1 | 1 | 1 | NONE | NONE | NONE | NONE |
| 1 | 0 | 0 | 0 | 1 | 0 thru 127 | 000000h – 7FEFFFh | 8,188KB | L – 2047/2048 |
| 1 | 0 | 0 | 1 | 0 | 0 thru 127 | 000000h – 7FDFFFh | 8,184KB | L – 1023/1024 |
| 1 | 0 | 0 | 1 | 1 | 0 thru 127 | 000000h – 7FBFFFh | 8,176KB | L – 511/512 |
| 1 | 0 | 1 | 0 | X | 0 thru 127 | 000000h – 7F7FFFh | 8,160KB | L – 255/256 |
| 1 | 1 | 0 | 0 | 1 | 0 thru 127 | 001000h – 7FFFFFh | 8,188KB | L – 2047/2048 |
| 1 | 1 | 0 | 1 | 0 | 0 thru 127 | 002000h – 7FFFFFh | 8,184KB | L – 1023/1024 |
| 1 | 1 | 0 | 1 | 1 | 0 thru 127 | 004000h – 7FFFFFh | 8,176KB | L – 511/512 |
| 1 | 1 | 1 | 0 | X | 0 thru 127 | 008000h – 7FFFFFh | 8,160KB | L – 255/256 |

W25Q64FW Individual Block Memory Protection (WPS = 1): -

