update sectors & write a single byte

T in the W25Q Flash series, and today we will see how to update data in s **us tutorial** we saw how to program pages, but the pages or sectors w writing data, and hence any previous data in the page was lost.

I will update the data in sectors rather than completely cleaning them. A e sector will remain intact, unless it is modified by the user.

W n 4 steps:

rom the sector and store it in the RAM.

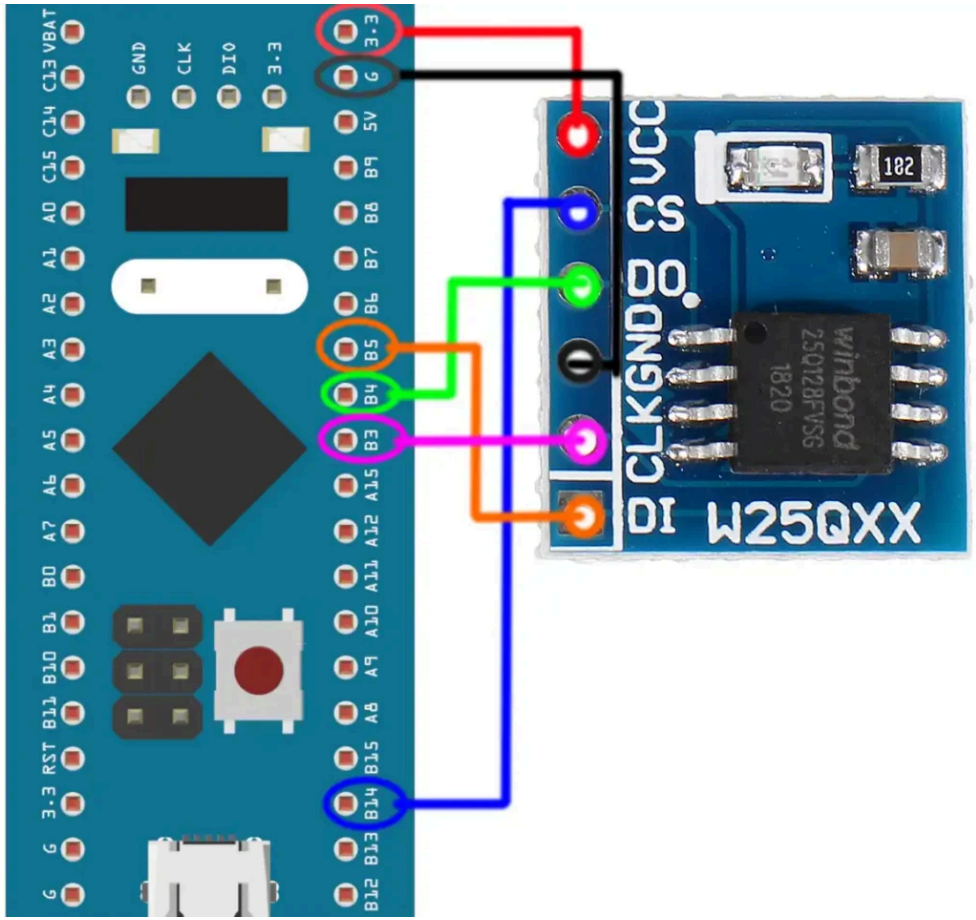
a

- Erase the sector.
- Write the modified sector data back to the sector.

We are performing these operations to the entire sector because a sector is the smallest memory segment we can erase. And hence we must deal with the entire sector even when updating a small amount of data into it.

The W25Q NOR Flash memories can use the SPI in Single/Dual/Quad mode. For the simplicity of the operation, we will stick to the Single SPI mode, where we would need 4 pins to connect the Flash with the controller.

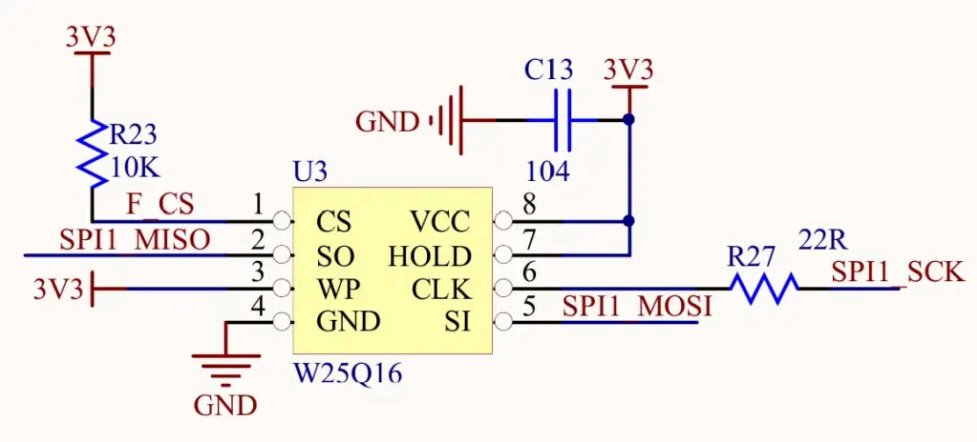
The connection diagram between the Flash module and the STM32 is shown below.



- The **CS** (Chip Select) Pin is connected to the pin PB14. It will be used to select or unselect the slave Device.

- The **DO** (Data Out) pin is connected to the pin PB4 (**MISO**). The device outputs the data on this pin
- The **CLK** (Clock) pin is connected to the pin PB3 (**CLK**). The clock is used to synchronise the master and the slave device.
- The **DI** (Data In) pin is connected to the pin PB5 (**MOSI**). The master send the data to the device on this pin.
- The Device is powered with 3.3v from the MCU itself.

The Module provides 6 pins (including the Vcc and Gnd). But the chip has 8 pins in total. If you have the chip rather than the module, you can connect it as shown below.

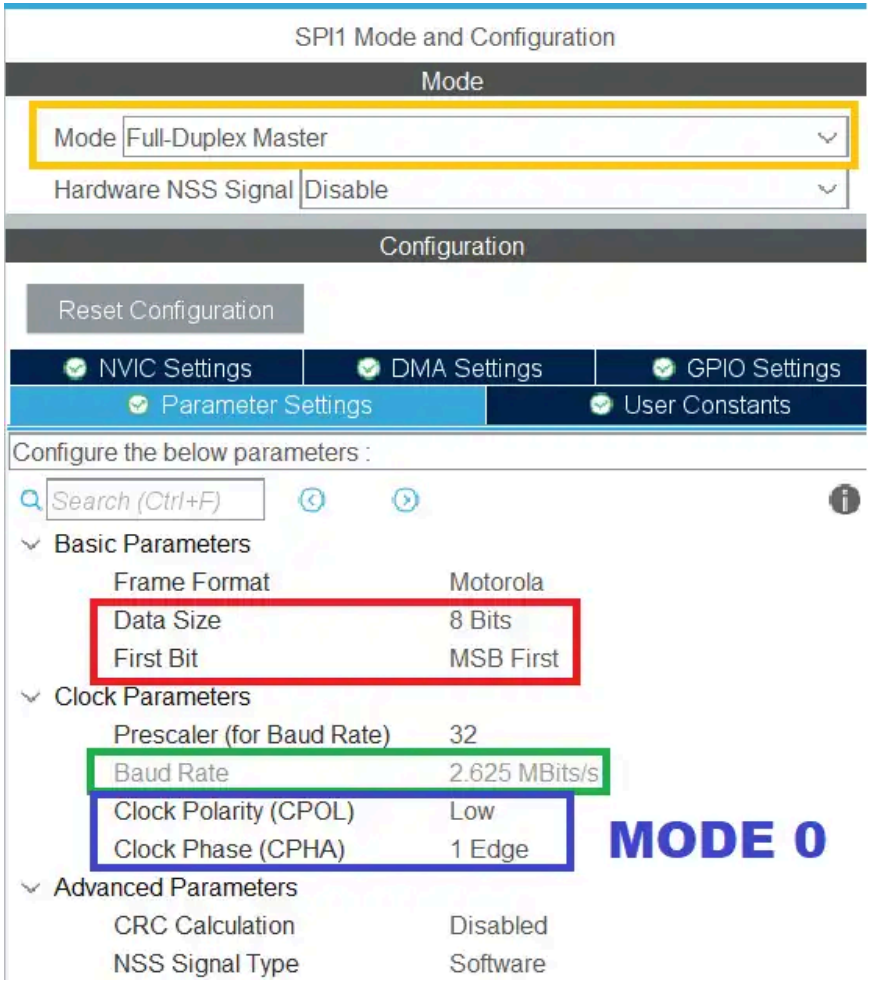


Note that the **WP** (Write Protect) pin is active Low pin, so it must be pulled HIGH when you want to modify the flash, and pulled LOW when you want to disable the modification.

The connections shown above are for the Single SPI mode, not for Dual or Quad modes.

CubeMX Setup

We will enable the SPI in **Full Duplex master mode**. The configuration is shown below.



The Data width is set to **8 bits** and the data should be transferred as the **MSB first**. The prescaler is set such that the Baud Rate is around **2.5 Mbits/sec**.

According to the datasheet of the W25Q Flash, the SPI Mode 0 and Mode 3 are supported. Below is the image from the datasheet.

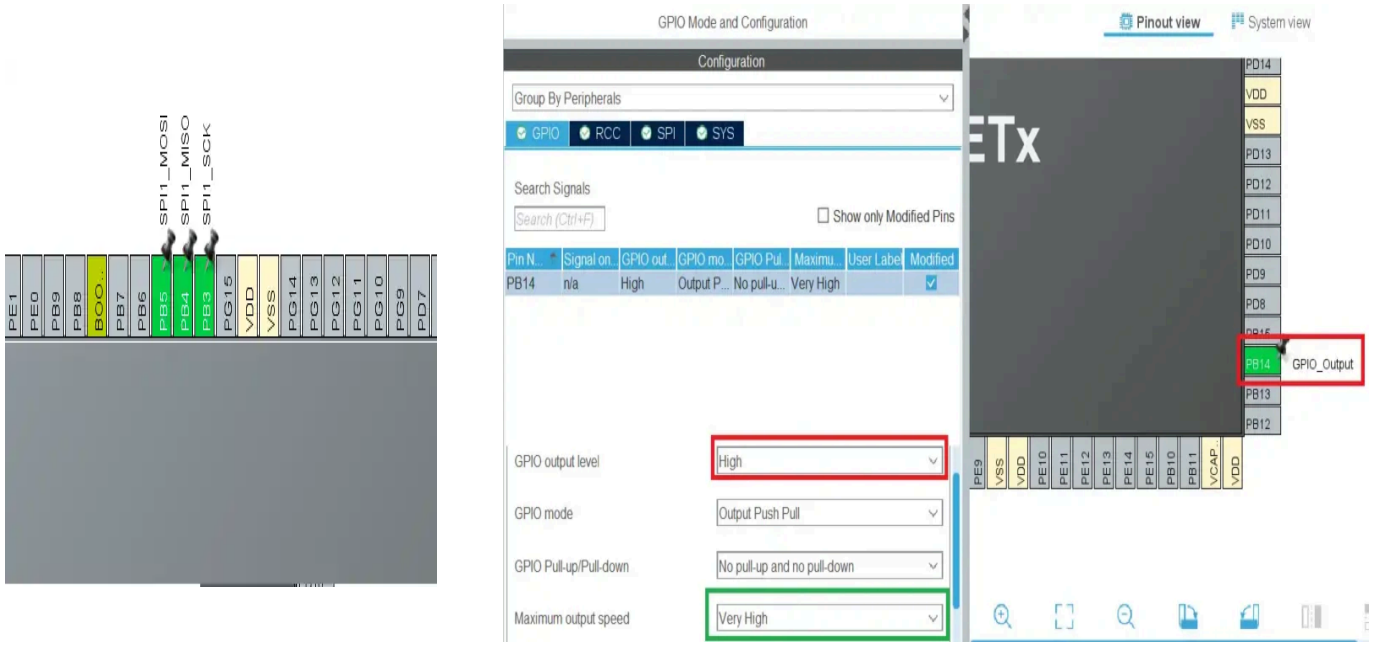
6.1 Standard SPI Instructions

The W25Q16JV is accessed through an SPI compatible bus consisting of four signals: Serial Clock (CLK), Chip Select (/CS), Serial Data Input (DI) and Serial Data Output (DO). Standard SPI instructions use the DI input pin to serially write instructions, addresses or data to the device on the rising edge of CLK. The DO output pin is used to read data or status from the device on the falling edge of CLK.

SPI bus operation Mode 0 (0,0) and 3 (1,1) are supported. The primary difference between Mode 0 and Mode 3 concerns the normal state of the CLK signal when the SPI bus master is in standby and data is not being transferred to the Serial Flash. For Mode 0, the CLK signal is normally low on the falling and rising edges of /CS. For Mode 3, the CLK signal is normally high on the falling and rising edges of /CS.

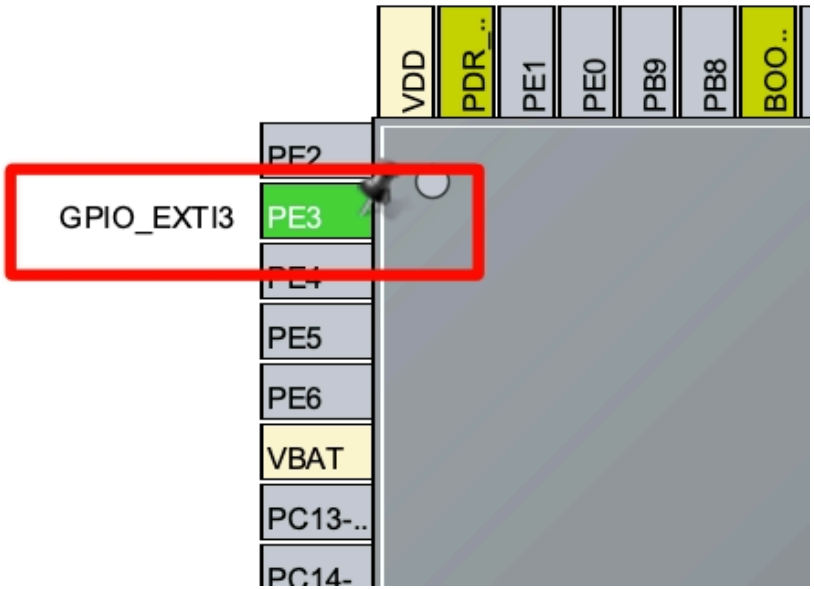
In the SPI configuration, we keep the Clock Polarity **(CPOL) low** and the Clock Phase **(CPHA) to 1 edge**. Here 1 edge means that the data will be sampled on the **first edge of the clock**. And when the CPOL is Low, the first edge is the **rising edge**. Basically we are using the **SPI Mode 0**.

In Full duplex Mode, SPI uses 3 pins, **MOSI**, **MISO** and **CLK**. We need to set one more pin as output so to be used as the Chip Select (**CS**) Pin.



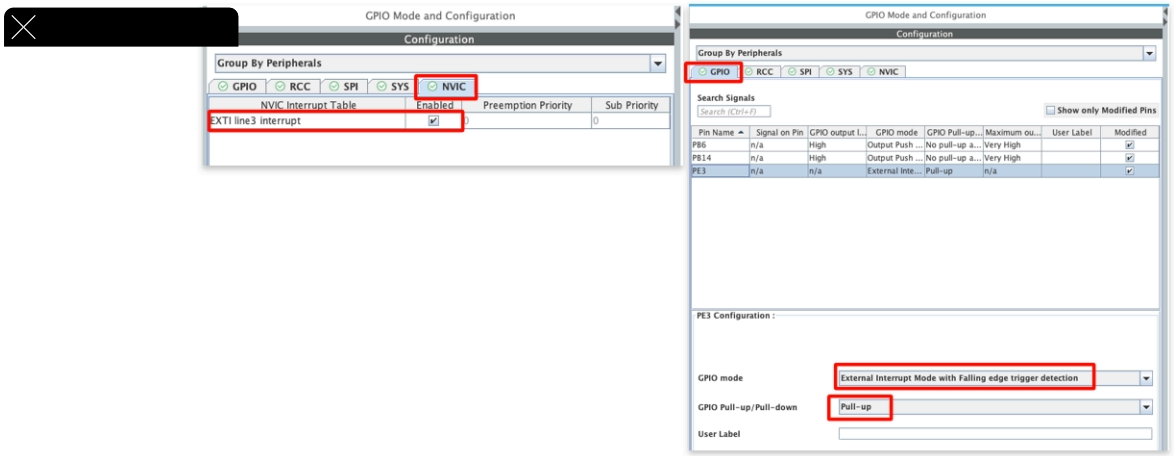
The Pin **PB14** is set as the CS pin. The initial output level is set to **HIGH**. This is because the pin needs to be pulled low in order to select the slave device, so we keep it high initially to make sure the slave device isn't selected. The Output speed is set to **Very High** because we might need to select and unselect the slave at higher rate.

We will also add a button to the project. Whenever the button is pressed, the data will be stored in the flash memory. The board I am using already has a user button, Its connection is shown below.



The pin PE3 is set as the EXTI (External Interrupt) pin.

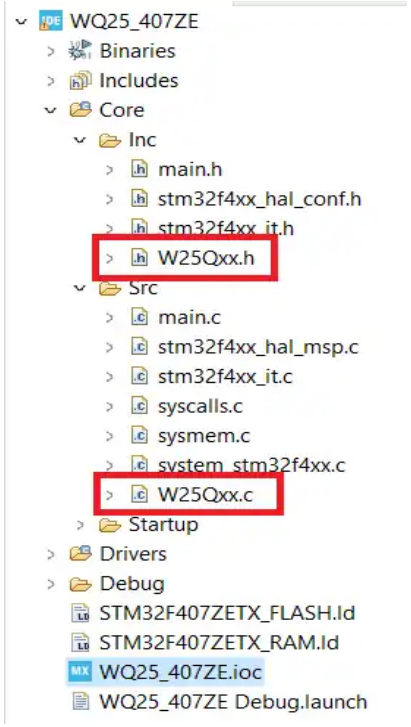
In the GPIO configuration, I have enabled the Interrupt for the EXTI line 3.



The GPIO mode is set to falling edge detection. Basically when the the button is pressed, the pin goes LOW. On the detection of the falling edge, the interrupt will be triggered.

We created separate library files for the device. Today we will simply update the code in these files itself.

The files are: **W25Qxx.c** file in the **src** directory and **W25Qxx.h** file in the **Inc** directory.


WQ25_407ZE
Binaries
Includes
Core
Inc
main.h
stm32f4xx_hal_conf.h
stm32f4xx_it.h
W25Qxx.h
Src
main.c
stm32f4xx_hal_msp.c
stm32f4xx_it.c
syscalls.c
sysmem.c
system_stm32f4xx.c
W25Qxx.c
Startup
Drivers
Debug
STM32F407ZETX_FLASH.ld
STM32F407ZETX_RAM.ld
WQ25_407ZE.ioc
WQ25_407ZE Debug.launch

Let’s Write the Code

We will first start with how to update the sectors with new data. Later we will see how to write a single byte data into the given memory address.

Updating the sector

```
void W25Q_Write (uint32_t page, uint16_t offset, uint32_t size, uint8_t *data)
{
    uint16_t startSector  = page/16;
    uint16_t endSector  = (page + ((size+offset-1)/256))/16;
    uint16_t numSectors = endSector-startSector+1;
```

The W25Q_Write function will be used to update the data. It takes the following parameters

- **@page** is the start page, where the write will start from.
- **@offset** is the offset on the first page. This can vary between 0 to 255.
- **@size** is the size of data we want to write.
- **@data** is the pointer to the data that we want to write.

First we will start with calculating the start sector, where the writing starts from. Then we will calculate the end sector (this calculations is explained in the previous tutorial) and the number of sectors we are updating.

```
uint8_t previousData[4096];
uint32_t sectorOffset = ((page%16)*256)+offset;
uint32_t dataindx = 0;
```

We will define some variables to be used later in the code.

- The array, **previousData**, will be used to store the current data in the sector.
- **sectorOffset** is the offset with respect to the start of the sector. This variable represents the position in the sector, where the modification will start.
- **dataindx** will be used to keep track of how many bytes have been used from the data pointer.

```
for (uint16_t i=0; i<numSectors; i++)
{
    uint32_t startPage = startSector*16;
    W25Q_FastRead(startPage, 0, 4096, previousData);
```

Now we will call the for loop as many times as the number of sectors we have to update.

- Here we will first calculate the **startPage**. This value depends on the startSector and is a multiple of 16. So this can be either 0, 16, 32, 48, etc.
- Then we will read the entire sector to fetch the data stored in it. We will save this data in the array we created earlier.

```
uint16_t bytesRemaining = bytestomodify(size, sectorOffset);
for (uint16_t i=0; i<bytesRemaining; i++)
{
    previousData[i+sectorOffset] = data[i+dataindx];
}
```

- Next we will calculate the bytesRemaining to be modified in the current sector. This value depends on the sector offset.
- Then modify the previousData array with the new data values. The modification will start from the sectorOffset and we will modify the number of bytes as the value of bytesRemianing variable.

```
W25Q_Write_Clean(startPage, 0, 4096, previousData);
```

After modifying the array, we will write the entire array (4096 bytes) to the sector, starting from the start of the sector. This will result in the update of the previous data with newly

modified data.

```
            startSector++;
            sectorOffset = 0;
            dataindx = dataindx+bytesRemaining;
            size = size-bytesRemaining;
        }
    }
```

After writing the sector we will update the variables for the next transfer.

- We will **increment** the startSector and set the sectorOffset to **0**, so that the new data can start from the beginning of the new sector.
- We have already modified the bytesRemaining number of bytes, so the dataindx will increment by this amount, so that we have the offset in the data pointer for the next transfer.
- Also the size of the total data will be reduced by the number of bytesRemaining bytes.

Writing single byte

Now we will see another function to write a single byte of data into the memory. This function will write data without erasing the entire sector, but the condition is that the particular memory location should have been already erased (has 0xFF in it).

```
void W25Q_Write_Byte (uint32_t Addr, uint8_t data)
{
    uint8_t tData[6];
    uint8_t indx;

    if (numBLOCK<512)   // Chip Size<256Mb
    {
        tData[0] = 0x02;  // page program
        tData[1] = (Addr>>16)&0xFF;  // MSB of the memory Address
        tData[2] = (Addr>>8)&0xFF;
        tData[3] = (Addr)&0xFF; // LSB of the memory Address
        tData[4] = data;
        indx = 5;
    }
    else  // we use 32bit memory address for chips >= 256Mb
    {
        tData[0] = 0x12;  // Write Data with 4-Byte Address
        tData[1] = (Addr>>24)&0xFF;  // MSB of the memory Address
        tData[2] = (Addr>>16)&0xFF;
        tData[3] = (Addr>>8)&0xFF;
        tData[4] = (Addr)&0xFF; // LSB of the memory Address
```

```
        tData[5] = data;
        indx = 6;
    }
```

This function is very similar to writing page. It takes the parameters **@Addr** the address of the memory to be written and **@data** the data byte to be written.

Here we first define an array and store the instruction along with the data to be sent.

Now before writing the data, we will first check if the memory location is empty (erased). If it is, then write a single byte data into the memory.

We must enable the write before writing the data into the memory and wait for the write cycle to finish at the end of the fucntion.

Result

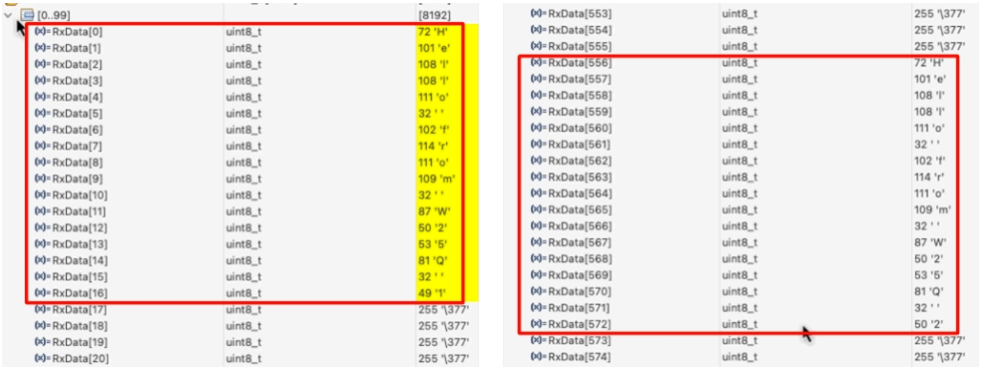
Updating the data

In the main function we will store the data at different locations in in the sector. We should see the data stored in both the locations.

```
    while (1)
    {
        if (isPressed == 1)
        {
            indx++;
            if (indx == 1)
            {
                sprintf ((char *)TxData, "Hello from W25Q %d", indx);
                W25Q_Write(0, 0, strlen ((char *)TxData), TxData);
            }

            else if (indx == 2)
            {
                sprintf ((char *)TxData, "Hello from W25Q %d", indx);
                W25Q_Write(2, 44, strlen ((char *)TxData), TxData);
            }

            isPressed = 0;
        }

        W25Q_FastRead(0, 0, 4096, RxData);

        HAL_Delay(500);
    }
```

Whenever the button is pressed, the indx value gets incremented. For different value of the indx variable, we store the data at different locations in the same sector. Since the function updates the sector, we should see both the data in our final output.

Below is the image showing the data at the location **0**, and at the location **556 (2,44)**. Both of these locations lies in sector 0 and hence the updating sector works fine.



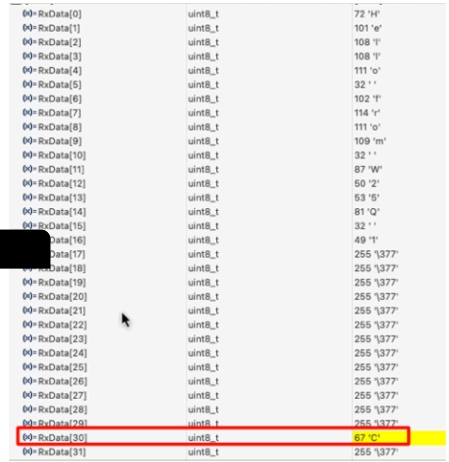
Writing single byte

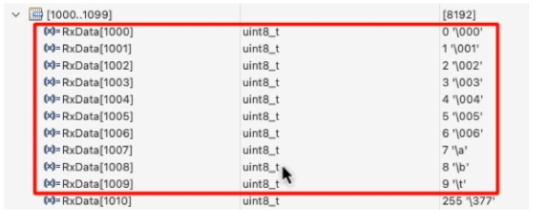
In the main function we will write a single byte to the different memory locations.

```
    while (1)
    {
        if (isPressed == 1)
        {
            W25Q_Write_Byte(30, 'C');

            for (int i=0; i<10; i++)
            {
                W25Q_Write_Byte(i+1000, i);
            }

            isPressed = 0;
        }

        W25Q_FastRead(0, 0, 8192, RxData);

        HAL_Delay(500);
    }
```

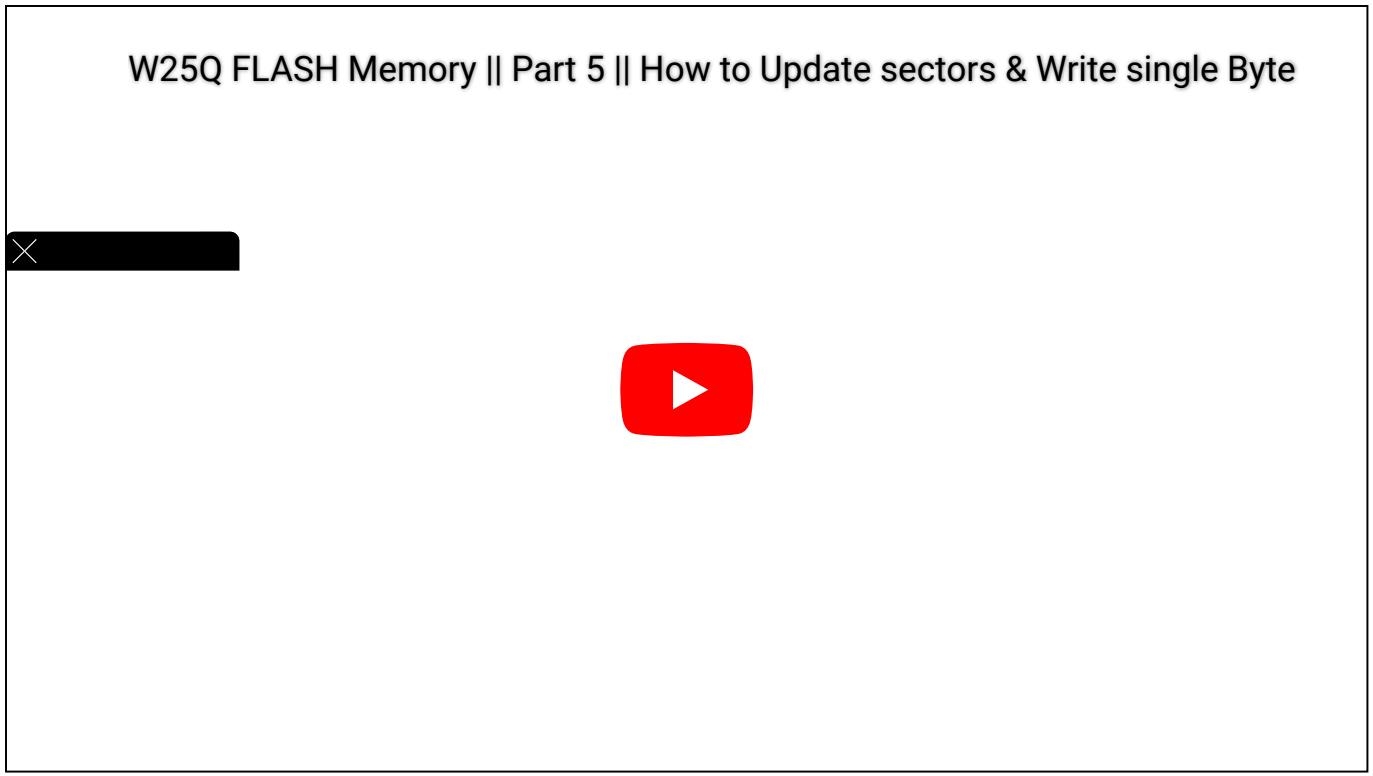
Here I am writing the data (‘C’) to the memory location 30. And also 10 bytes of data starting from the location 1000.

Below is the image showing the data at the respective locations.





Check out the Video Below


W25Q FLASH Memory || Part 5 || How to Update sectors & Write single Byte

PART 4 **PART 6**

DOWNLOAD SECTION

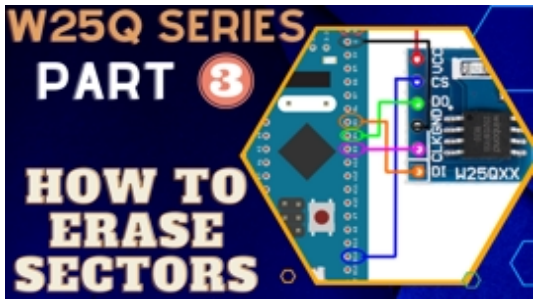
Info

You can help with the development by DONATING

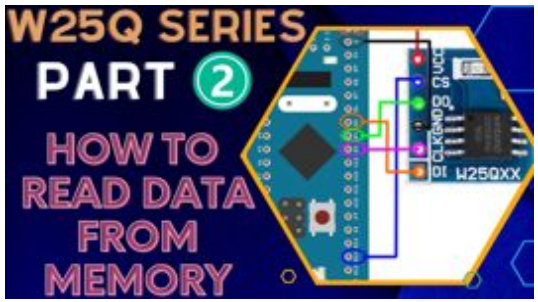
To download the code, click DOWNLOAD button and view the Ad. The project will download after the Ad is finished.

DOWNLOAD DONATE

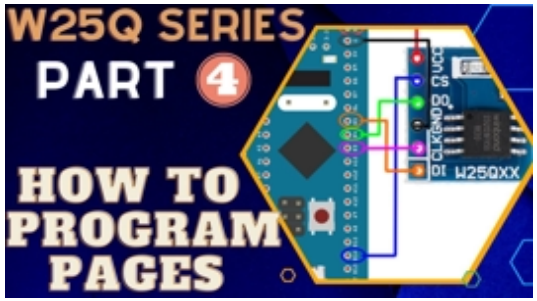
Related Posts:



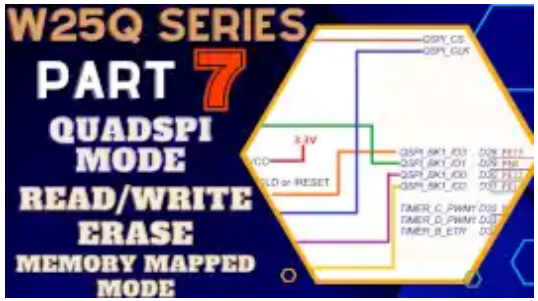
W25Q Flash Series || Part 3 || How to Erase Sectors



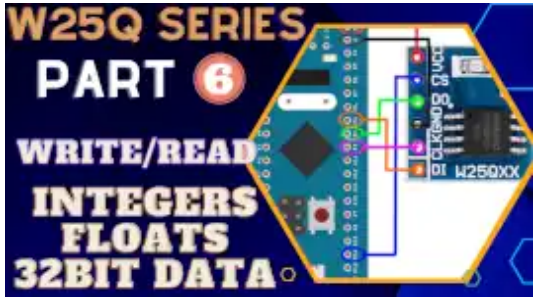
W25Q Flash Series || Part 2 || Read Data from Device



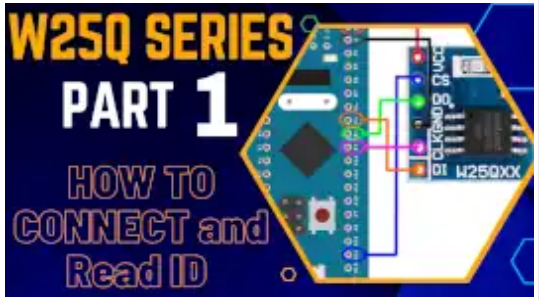
W25Q Flash Series || Part 4 || How to Program Pages



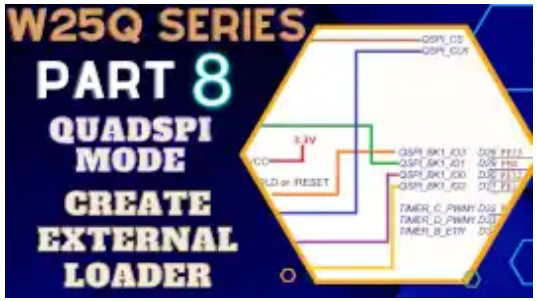
W25Q Flash Series || Part 7 || QUADSPI Write, Read, Memory



W25Q Flash Series || Part 6 || Integers floats and 32bit Data



W25Q Flash Series || Part 1 || Read ID



W25Q Flash Series || Part 8 || QUADSPI External Loader

STM32 as I2C SLAVE || PART 5

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name *

Email *

Post Comment

Privacy Policy