

How to create the SPI Flash loader

This is the 9th tutorial in the [W25Q Flash series](#), and today we will see how to create and use the External Loader for the W25Q flash memory with the SPI peripheral. In the [previous tutorial](#) we saw how to create an external loader using QSPI peripheral. Today's tutorial is going to be somewhat similar, but we will use the SPI peripheral instead of the QSPI.

Using the QSPI peripheral for external flash memories have major advantages like faster speed, memory mapped mode, etc. But we know that not all the STM32 MCUs supports the QSPI mode, specially the popular and cheaper dev boards like F4 discovery, bluepill, F4 nucleo etc. Using SPI flash loader, we can still use the W25Q flash memories to store the data to the external flash using the SPI peripheral

Th

X

]

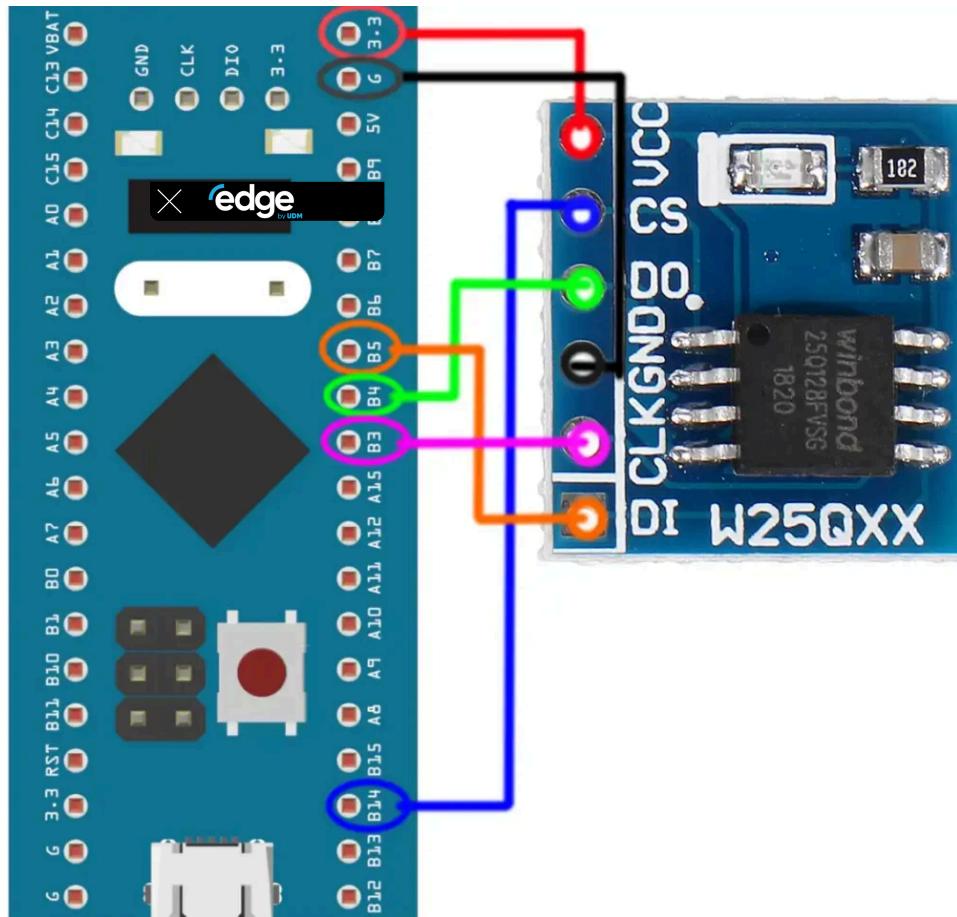


With SPI flash loader, we can use the cube programmer to flash the binary file to the external flash memory, which can contain data like images or fonts. The data can later be read in the IDE itself. Although **we can not use the memory mapped mode**, so the internal flash is not extended in a way.

The process of creating the external loader will be the same as we did in the [previous QSPI tutorial](#). Although I have made some changes in the files used and I will explain those changes here.

Connection

The connection diagram between the Flash module and the STM32 is shown below. By using this site, you agree with it. [Privacy Policy](#).



- 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.

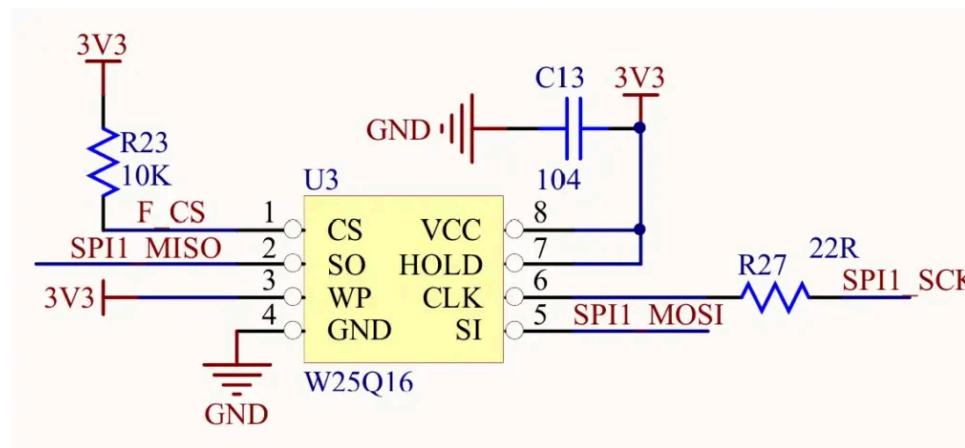
X

This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

- 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.



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

CubeMX Setup



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

SPI1 Mode and Configuration

Mode

Mode: Full-Duplex Master

Hardware NSS Signal: Disable

Configuration

Reset Configuration

NVIC Settings DMA Settings GPIO Settings

Parameter Settings User Constants

Configure the below parameters:

Basic Parameters

Frame Format	Motorola
Data Size	8 Bits
First Bit	MSB First

Clock Parameters

Prescaler (for Baud Rate)	32
Baud Rate	2.625 MBits/s
Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge

Advanced Parameters

CRC Calculation	Disabled
NSS Signal Type	Software

MODE 0

This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

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  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.

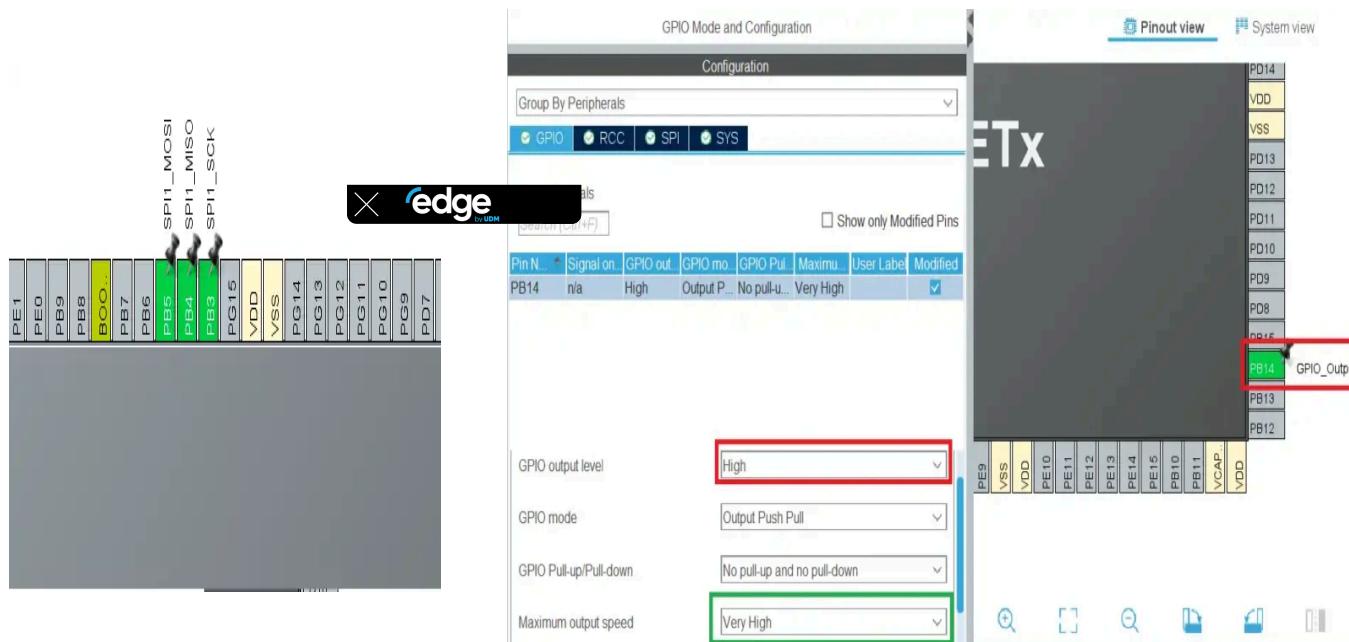
SPI bus operation Mode 0 (0,0) and 3 (1,1) are supported. The primary difference between Mode 0 and Mode 3 concerns the normal state of the CLK signal when the SPI bus master is in standby and data is not being transferred to the Serial Flash. For Mode 0, the CLK signal is normally low on the falling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



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).



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.



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#)



Create the Loader

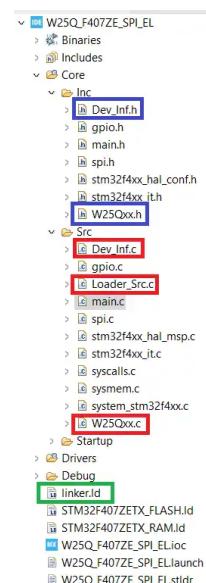
We need to first copy certain files into our project. You can get the files after downloading the project at the end of this post.



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

Name	Date modified	Type
W25Qxx	24-11-2023 21:43	H File
Dev_Inf	24-11-2023 17:02	H File
linker	24-11-2023 19:00	LD File
Dev_Inf	24-11-2023 17:24	C File
Loader_Src	03-12-2023 11:53	C File
W25Qxx	24-11-2023 21:43	C File

Now copy the *****.c** files in the **src** directory of the project, *****.h** file in the **inc** directory and the **linker** file in the main project folder. The final project structure is shown below.



You can set the Loader name in the **Dev_inf.c** file as shown below. Also make sure to cross check the **QSPI start Address** (0x90000000 in this case) with the reference manual of your

MCU. If the MCU does not have the QSPI address assigned, leave it to 0x90000000, but make sure there is no other peripheral assigned to this address.

```
#else
struct StorageInfo storageInfo = {
#endif
    "W25Q16_F407ZE_SPI_E1", // Device Name + version number
    NOR_FLASH, // Device Type
    0x90000000, // Device Start Address
    MEMORY_FLASH_SIZE, // Device Size in Bytes
    MEMORY_PAGE_SIZE, // Programming Page Size
    0xFF, // Initial Content of Erased Memory

    // Specify Size and Address of Sectors (view example below)
    {
        (MEMORY_FLASH_SIZE / MEMORY_SECTOR_SIZE), // Sector Numbers,
        (uint32_t) MEMORY_SECTOR_SIZE
    }, //Sector Size

    { 0x00000000, 0x00000000 }
};

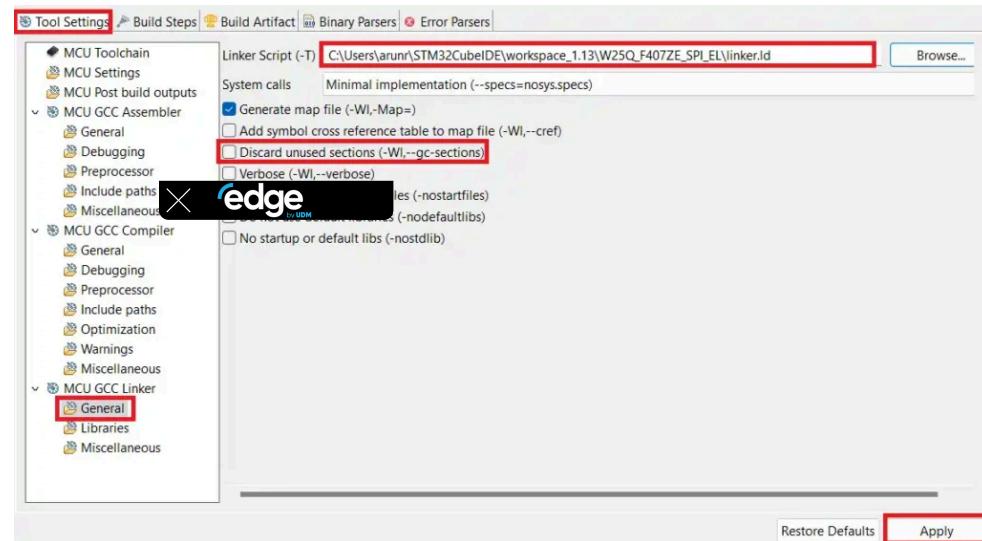
};
```

Other details like **MEMORY_FLASH_SIZE**, **PAGE_SIZE**, **SECTOR_SIZE** will be fetched from the **W25Qxx.h** file.

Now we need to set the linker file for the project.

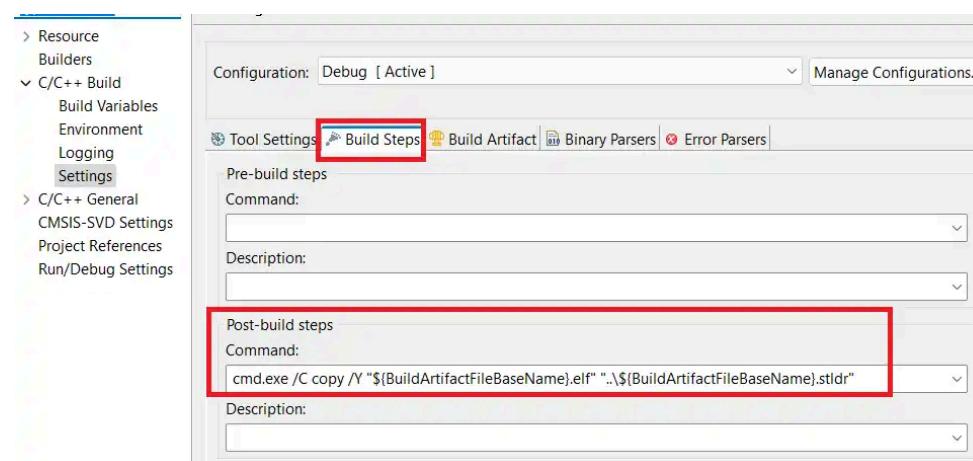
Go to **Project Properties -> C/C++ Build -> Settings -> MCU GCC Linker -> General**.

Here change the **Linker Script** to the Linker file we copied.



As shown above i have changed the **Linker Script** to the **linker.ld** file. Also make sure to uncheck the **Discard unused sections**. After making the changes, click on **Apply**.

We also need to copy the post build command to the Build Steps tab as shown below.



The command is **cmd.exe /C copy /Y "\${BuildArtifactFileBaseName}.elf"**

This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).
"..\\${BuildArtifactFileBaseName}.stldr". It will copy the stldr file (loader) to the main

project folder.

Now build the code, and you should see the Loader file (*****.stldr**) in the main project folder itself. This is shown below.

Name	Date modified	Type	Size
Debug	06-12-2023 18:40	File folder	
Drivers	03-12-2023 12:22	File folder	
.cproject	06-12-2023 18:40	CPROJECT File	27 KB
.mxproject	03-12-2023 12:23	MXPROJECT File	8 KB
.project	03-12-2023 12:22	PROJECT File	2 KB
linker	03-12-2023 12:24	LD File	4 KB
STM32F407ZETX_FLASH	03-12-2023 12:23	LD File	6 KB
STM32F407ZETX_RAM	03-12-2023 12:22	LD File	6 KB
W25Q_F407ZE_SPI_EL	03-12-2023 12:23	IOC File	6 KB
W25Q_F407ZE_SPI_EL.launch	03-12-2023 12:44	LAUNCH File	9 KB
W25Q_F407ZE_SPI_EL.stldr	06-12-2023 18:40	STLDR File	1,269 KB



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

The files explained



Changes in W25Qxx

There is no major changes in the W25Qxx library files. I used the same files which we created in the previous tutorials of this series and modified them a little.

I added the memory details in the **W25Qxx.h** file as shown below.

```
#define MEMORY_FLASH_SIZE          0x200000 /* 16Mbit =>2Mbyte */
#define MEMORY_BLOCK_SIZE          0x10000   /* blocks of 64KBytes */
#define MEMORY_SECTOR_SIZE          0x1000    /* 4kBytes */
#define MEMORY_PAGE_SIZE           0x100     /* 256 bytes */
```

Here you only need to change the `MEMORY_FLASH_SIZE`, if you are using any other chip from winbond. The blocks, sectors and page size remain the same.

i have also added few functions which will be used by the loader source file.

```
void flash_WriteMemory(uint8_t* buffer, uint32_t address, uint32_t buffer_size);
void flash_ReadMemory(uint32_t address, uint32_t buffer_size, uint8_t* buffer);
```

```
void flash_SectorErase(uint32_t EraseStartAddress, uint32_t EraseEndAddress);  
void flash_ChipErase (void);  
void flash_Reset (void);
```



Although we already have the functions to Read, Write, erase, etc, but the loader uses a different set of parameters for them. This is why I have defined new functions for the same. The functions are written in the **W25Qxx.c** file.

The **read** function is exactly how we used in the previous tutorials of this series. The loader uses the Address instead of page+offset, so we need to first extract the page and offset from the address and then call our read function.

```
void flash_ReadMemory (uint32_t Addr, uint32_t Size, uint8_t* buffer)  
{  
    uint32_t page = Addr/256; // 1 page occupies 256 bytes  
    uint16_t offset = Addr%256;  
  
    W25Q_FastRead(page, offset, Size, buffer);  
}
```

Similarly the **sector erase function** we used takes the sector number as the parameter. On the other hand, the loader uses the start sector and end sector address as the parameters.

So we extract the sector numbers from the parameter and then call our function.

This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

```
void flash_SectorErase(uint32_t EraseStartAddress, uint32_t EraseEndAddress)
{
    uint16_t startSector = EraseStartAddress/4096; // 1 sector occupies 4096 bytes
    uint16_t endSector = EraseEndAddress/4096;
    uint16_t numSectors = endSector-startSector+1;
    for (uint16_t i=0; i<numSectors; i++)
    {
        W25Q_Erase_Sector(startSector++);
    }
}
```

The **chip erase** and **reset** functions does not have any parameters, so they are called as it is.

```
void flash_ChipErase (void)
{
    W25Q_Chip_Erase();
}
```

```
void flash_Reset (void)
{
    W25Q_Reset();
}
```

The **write function** has some changes. The loader itself takes care of when to erase the sector and when to edit the sector, so we don't need to do it ourselves. We will remove the

sector erasing part from our write function and keep the rest of it same. This is shown below.

```
void flash_WriteMemory(uint8_t* buffer, uint32_t address, uint32_t buffer_size)
{
    uint32_t page = address/256;
    uint16_t offset = address%256;
    uint32_t size = buffer_size;
    uint8_t tData[266];
    uint32_t startPage = page;
    uint32_t endPage = startPage + ((size+offset-1)/256);
    uint32_t numPages = endPage-startPage+1;

    uint32_t dataPosition = 0;

    // write the data
    for (uint32_t i=0; i<numPages; i++)
    {
        uint32_t memAddr = (startPage*256)+offset;
        uint16_t bytesremaining = bytestowrite(size, offset);
        uint32_t indx = 0;

        write_enable();

        if (numBLOCK<512) // Chip Size<256Mb
        {
            tData[0] = W25Q_PAGE_PROGRAM; // page program
            tData[1] = (memAddr>>16)&0xFF; // MSB of the memory Address
```



Potężna broń na grzybicę. Już po kilku dniach paznokcie są lśniące i zdrowe.
bettermethod24.com

Ad



Changes in the loader_src file

The original loader files are taken from the [ST's github repo](#). These files are written for the QSPI peripheral, therefore we need to modify them to be used with the SPI peripheral.

I will compare both the modified code with the original code, so that you understand how it works.

At first we have the **loader initialization function**.

```
int Init(void) {  
    .....  
    __HAL_RCC_QSPI_FORCE_RESET(); //  
    __HAL_RCC_QSPI_RELEASE_RESET();  
    .....  
  
    __HAL_RCC_SPI1_FORCE_RESET(); //  
    __HAL_RCC_SPI1_RELEASE_RESET();  
    if (CSP_QUADSPI_Init() != HAL_OK)  
        __set_PRIMASK(1); //disable ir  
        return LOADER_FAIL;  
}
```

MX_SPI1_Init(~~This~~ website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

```
flash_Reset();  
    if (CSP_QSPI_EnableMemoryMappedMode()  
        __set_PRIMASK(1); //disable interrupt  
        return LOADER_FAIL;  
    }  
  
    __set_PRIMASK(1); //enable interrupt  
    return LOADER_OK;  
}
```

The default initialization function initializes the QSPI and enables the memory mapped mode. Since we can't use the memory mapped mode with the SPI peripheral, we will only initialize the SPI here.

Next is the **write** function.

```
int Write(uint32_t Address, uint32_t Value)  
{  
    __set_PRIMASK(0); //enable interrupt  
  
    if (HAL_QSPI_Abort(&hqspi) != HAL_OK)  
        __set_PRIMASK(1); //disable interrupt  
    return LOADER_FAIL;  
}  
  
flash_WriteMemory(buffer, Address,
```

This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

```
if (CSP_QSPI_WriteMemory((uint8_t*)buffer, Address, Value))  
    return LOADER_FAIL;
```

```

__set_PRIMASK(1); //disable interrupt
return LOADER_OK;
}

}

__set_PRIMASK(1); //disable interrupt
return LOADER_FAIL;
}

```



```

__set_PRIMASK(1); //disable interrupt
return LOADER_OK;
}

```

Here the original file calls for the `QSPI_WriteMemory` function. Instead we will call our `flash_WriteMemory` function, which uses the SPI peripheral to write the data.

Similar changes are made in the **Read**, **sector erase** and **chip erase** functions. You can download the files and check the code.

There are some major changes in the verify function.

```

uint32_t Verify(uint32_t MemoryAddr, uint32_t RAMBufferAddr, uint32_t Size, uint32_t misalignment) {
    __set_PRIMASK(0); //enable interrupts
    uint32_t VerifiedData = 0;
    uint32_t checksum;
    Size *= 4;
    while (Size > 0) {
        uint32_t product;
        uint32_t sum;
        checksum = Checksum((uint32_t)MemoryAddr + misalignment * 4, Size - (misalignment >> 2) * 4, 0);
        while (Size >= 4) {
            if (Flash_ReadMemory(RAMBufferAddr, 4, Buffer)) {
                sum += Buffer[0];
                if ((uint32_t)(MemoryAddr + misalignment * 4) == (uint32_t)RAMBufferAddr + VerifiedData) {
                    __set_PRIMASK(1); //enable interrupt
                    return (checksum << 4) + MemoryAddr + VerifiedData;
                }
                VerifiedData += sum;
                sum = 0;
            }
            __set_PRIMASK(1); //enable interrupt
        }
        __set_PRIMASK(0); //enable interrupts
        return (checksum << 4);
    }
}

```

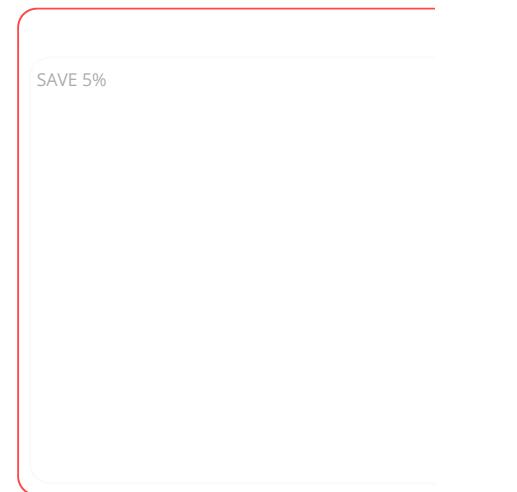
```

uint32_t Verify(uint32_t MemoryAddr, uint32_t RAMBufferAddr, uint32_t Size, uint32_t misalignment) {
    __set_PRIMASK(0); //enable interrupts
    uint32_t VerifiedData = 0;
    uint32_t checksum;
    Size *= 4;
    if (QSPI_SetFlashMode(QSPI_MODE_0) != QSPI_OK) {
        __set_PRIMASK(1); //enable interrupt
        return (checksum << 4);
    }
    checksum = Checksum((uint32_t)MemoryAddr + misalignment * 4, Size - (misalignment >> 2) * 4, 0);
    if (*uint32_t*)MemoryAddr == *(uint32_t*)RAMBufferAddr + VerifiedData) {
        __set_PRIMASK(1); //enable interrupt
        return (checksum << 4);
    }
    VerifiedData += checksum;
    __set_PRIMASK(1); //enable interrupt
    return (checksum << 4);
}

```

To verify the data, the original code first enables the memory mapped mode and then compares the data at the external memory location. Since we can't use the memory mapped mode, we will read the data from the external memory and then compare it with our data.

The rest of the operations are same to that of the original file.

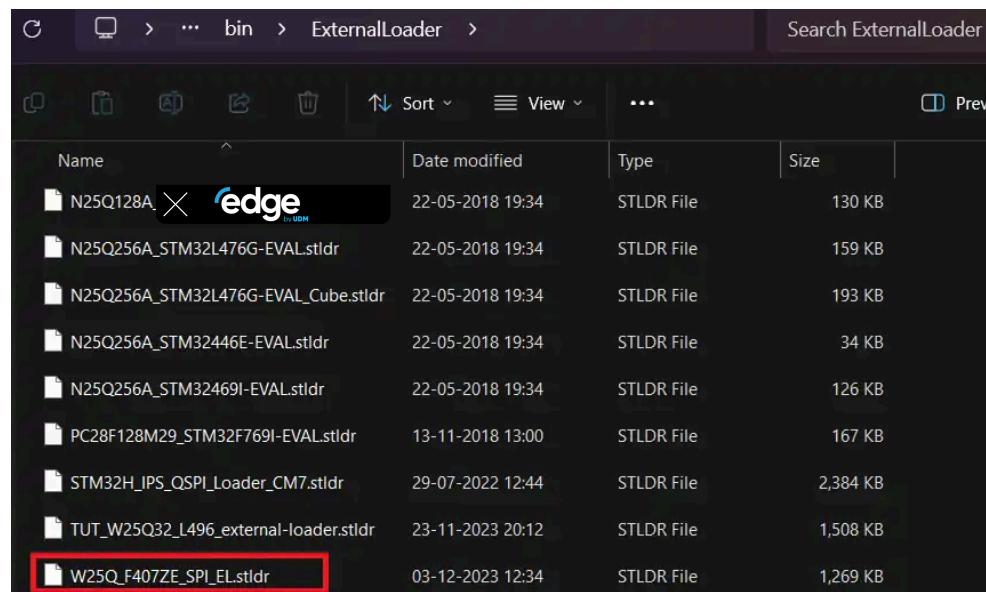


How to use the SPI Loader

As I mentioned earlier, we can use the cube programmer to flash the binary file to the external memory and the loader is needed for this operation.

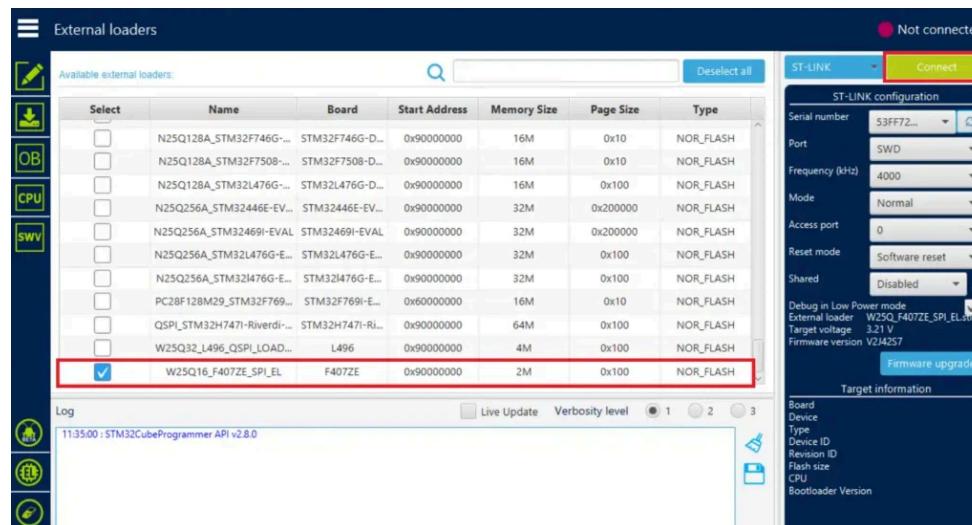
We need to first copy the Loader to the cubeprogrammer directory. Copy the *****.stldr** file to the **C:\Program**

[Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\ExternalLoader](https://www.st.com/en/development-tools/stm32cube-stm32cube-programmer.html#ExternalLoader), you agree with it. [Privacy Policy](#).



Name	Date modified	Type	Size
N25Q128A_Xedge.stldr	22-05-2018 19:34	STLDR File	130 KB
N25Q256A_STM32L476G-EVAL.stldr	22-05-2018 19:34	STLDR File	159 KB
N25Q256A_STM32L476G-EVAL_Cube.stldr	22-05-2018 19:34	STLDR File	193 KB
N25Q256A_STM32446E-EVAL.stldr	22-05-2018 19:34	STLDR File	34 KB
N25Q256A_STM32469I-EVAL.stldr	22-05-2018 19:34	STLDR File	126 KB
PC28F128M29_STM32F769I-EVAL.stldr	13-11-2018 13:00	STLDR File	167 KB
STM32H_IPS_QSPI_Loader_CM7.stldr	29-07-2022 12:44	STLDR File	2,384 KB
TUT_W25Q32_L496_external-loader.stldr	23-11-2023 20:12	STLDR File	1,508 KB
W25Q_F407ZE_SPI_El.stldr	03-12-2023 12:34	STLDR File	1,269 KB

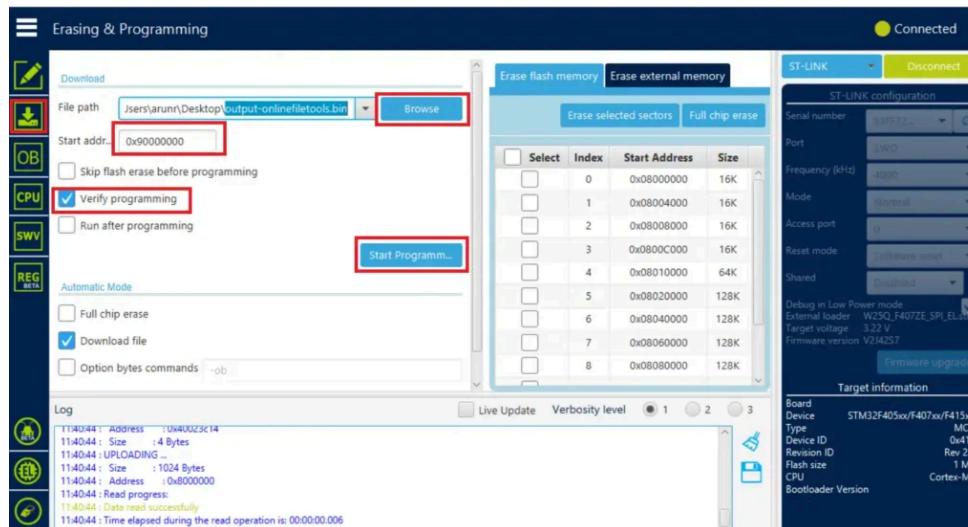
Now open the cube programmer. Go to the **EL** (External Loader) section and select the loader. Then **connect** the board to the programmer.



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

We will program the bin file directly to the flash memory. You can download a test binary file from <https://onlinefiletools.com/generate-random-binary-file>.

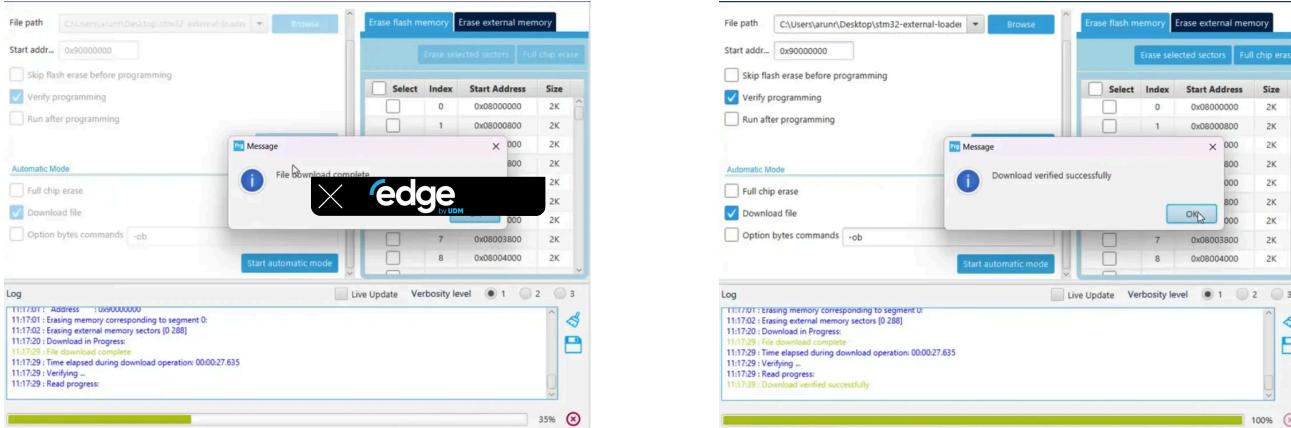
Go to the Download section, select the ***.bin, enter the QSPI Start Address, and start programming.



You should see 2 notifications, the first one will pop up once the file has been downloaded to the memory, and another when the downloaded file has been verified.



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#)



The file has been downloaded and verified means that the external loader is working fine.

You can see the content of the external flash in the **memory tab**.

Address	0	4	8	C	ASCII
0x90000000	36383135	37383938	30313535	39323231	5186898755101229
0x90000010	35353137	35363137	35343433	37373935	7155716534455977
0x90000020	39323234	35383035	37363334	35303835	4229508543675805
0x90000030	33393732	31323932	30363831	34393337	2793292118607394
0x90000040	37383037	33373933	31343534	32353737	7087397345417752
0x90000050	33353137	36393136	34333131	36393535	7153619611345596
0x90000060	34393133	36303039	32303633	38393831	3194900636021898
0x90000070	33313438	35343737	35303535	39323939	8413774555059929
0x90000080	31323633	33333332	35303332	32393334	3621233323054392
0x90000090	32353734	35353536	39393834	31323335	4752655548995321
0x900000A0	31373534	30373438	33363837	32353339	4571847078639352
0x900000B0	37353038	38353237	39303734	30393937	8057725847097990

We can use the cube programmer to flash a binary file to the SPI flash memory. We will use this idea to flash the images, fonts or videos needed for the display, to the external flash, and then read them later in the project.

This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

LVGL have the tools to generate the binary files for the images and fonts. On the other hand, the touchGFX can directly use the SPI flash using the external loader. I will make tutorials on both, the LVGL and the touchGFX, with the SPI flash.



Check out the Video Below



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

W25Q FLASH Memory || Part 9 || External Loader in SPI Mode



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).



4800 m² - Sob... 2174 m² - Sob...

456 000 zł

432 000 zł

1130 m² - Sob... 900 m² - Grab...

259 000 zł

215 000 zł

This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

1180 m² - Sob... 1199 m² - Sob...

295 000 zł

240 000 zł



902 m² - Biały... 1138 m² - Sob...

184 910 zł

250 000 zł



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#)

DOWNLOAD SECTION



SAVE 19%

SAVE 5%

SAVE 37%



Info

You can help with the development by DONATING

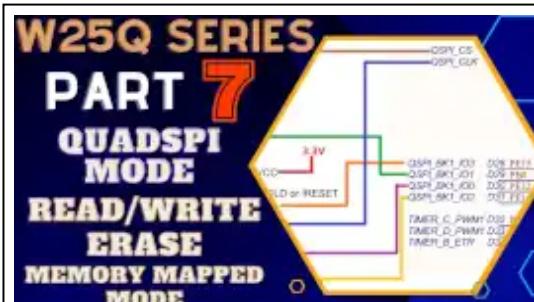
***To download the code, click DOWNLOAD button and view the Ad. The project
will download after the Ad is finished.***



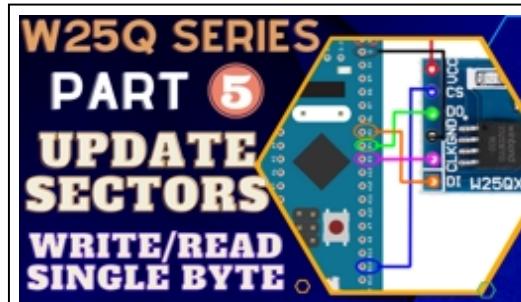
This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

[DOWNLOAD](#)[DONATE](#)

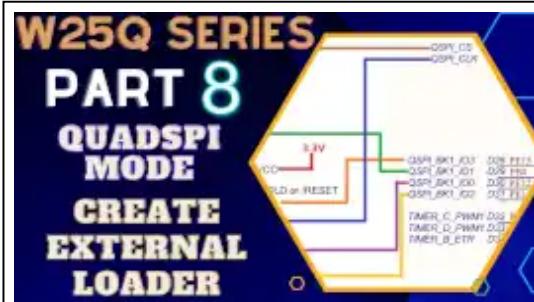
Related Posts:



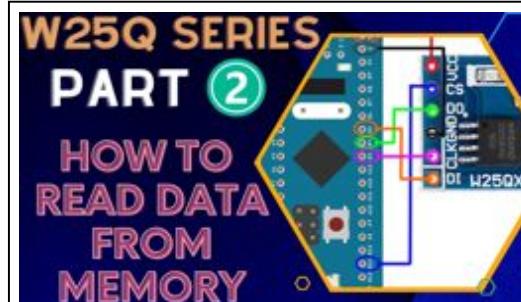
[W25Q Flash Series || Part 7 || QUADSPI Write, Read, Memory](#)



[W25Q Flash Series || Part 5 || how to update sectors](#)



[W25Q Flash Series || Part 8 || QUADSPI External Loader](#)



[W25Q Flash Series || Part 2 || Read Data from Device](#)



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).



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



W25Q Flash Series || Part 1 || Read ID



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



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

Leave a Reply

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

Comment *



This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).

Name * Email ***Post Comment**[Privacy Policy](#)

This website uses cookies to improve your experience. If you continue to use this site, you agree with it. [Privacy Policy](#).