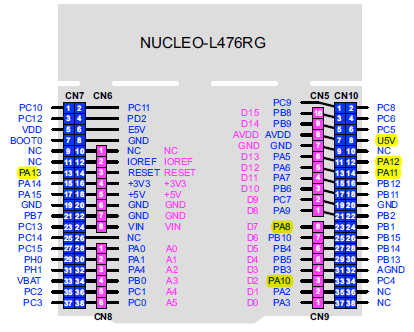
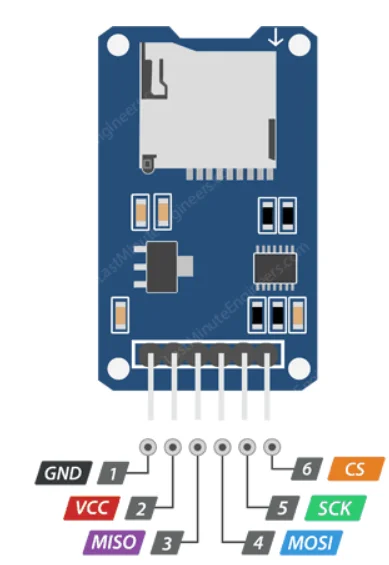
**STM32 SD interfacing**

**STM32 Pinouts**



**MicroSD**



**Connection Diagram**

The SD Card is connected to the STM32 through SPI2.

|  |  |
| --- | --- |
| STM32-NUCLEO-L476RG | SD CARD READER |
| 5V | VCC |
| GND | GND |
| GPIOP1 (OUTPUT) | CS |
| MISO - GPIOP2 | MISO |
| MOSI – GPIOP3 | MOSI |
| SCK - GPIOB10 | SCK |

**Project connection**

* Create the new project in STM32CubeIDE.
* In the .ioc file pinout and configuration, click Connectivity → SPI 2.
* Select the Mode as Full-Duplex Master.
* Then click GPIOC1 in the Pinout View and set that as a GPIO\_Output.
* This GPIO will be used for Chip Select (CS).
* Now we will enable the FATFS.
* Click **Middleware**→ **FATFS.**
* Then tick **User-defined**. In the FATFS **Configuration**, set **USE\_LFN**as **Enabled with static working buffer on the BSS** to use the long file name.
* Then set the **MAX\_SS**as **4096**.

- That’s all. Now save and generate the code.

- Create a new header file named **fatfs\_sd\_card.h (under the Core/Inc.).**

- Then copy and paste the below code to that file.

================================================================================================================================================

#ifndef \_\_FATFS\_SD\_H

#define \_\_FATFS\_SD\_H

/\* Definitions for MMC/SDC command \*/

#define CMD0 (0x40+0) /\* GO\_IDLE\_STATE \*/

#define CMD1 (0x40+1) /\* SEND\_OP\_COND \*/

#define CMD8 (0x40+8) /\* SEND\_IF\_COND \*/

#define CMD9 (0x40+9) /\* SEND\_CSD \*/

#define CMD10 (0x40+10) /\* SEND\_CID \*/

#define CMD12 (0x40+12) /\* STOP\_TRANSMISSION \*/

#define CMD16 (0x40+16) /\* SET\_BLOCKLEN \*/

#define CMD17 (0x40+17) /\* READ\_SINGLE\_BLOCK \*/

#define CMD18 (0x40+18) /\* READ\_MULTIPLE\_BLOCK \*/

#define CMD23 (0x40+23) /\* SET\_BLOCK\_COUNT \*/

#define CMD24 (0x40+24) /\* WRITE\_BLOCK \*/

#define CMD25 (0x40+25) /\* WRITE\_MULTIPLE\_BLOCK \*/

#define CMD41 (0x40+41) /\* SEND\_OP\_COND (ACMD) \*/

#define CMD55 (0x40+55) /\* APP\_CMD \*/

#define CMD58 (0x40+58) /\* READ\_OCR \*/

/\* MMC card type flags (MMC\_GET\_TYPE) \*/

#define CT\_MMC 0x01 /\* MMC ver 3 \*/

#define CT\_SD1 0x02 /\* SD ver 1 \*/

#define CT\_SD2 0x04 /\* SD ver 2 \*/

#define CT\_SDC 0x06 /\* SD \*/

#define CT\_BLOCK 0x08 /\* Block addressing \*/

/\* Functions \*/

DSTATUS SD\_disk\_initialize (BYTE pdrv);

DSTATUS SD\_disk\_status (BYTE pdrv);

DRESULT SD\_disk\_read (BYTE pdrv, BYTE\* buff, DWORD sector, UINT count);

DRESULT SD\_disk\_write (BYTE pdrv, const BYTE\* buff, DWORD sector, UINT count);

DRESULT SD\_disk\_ioctl (BYTE pdrv, BYTE cmd, void\* buff);

#define SPI\_TIMEOUT 100

extern SPI\_HandleTypeDef hspi2;

#define HSPI\_SDCARD &hspi2

#define SD\_CS\_PORT GPIOC

#define SD\_CS\_PIN GPIO\_PIN\_1 //Select your slave pin

#endif

================================================================================================================================================

* Create a new header file named **fatfs\_sd\_card.c** under the **Core/Src.**
* Then copy and paste the below code to that file.

================================================================================================================================================

#define TRUE 1

#define FALSE 0

#define bool BYTE

#include "stm32f1xx\_hal.h"

#include "diskio.h"

#include <fatfs\_sd\_card.h>

uint16\_t Timer1, Timer2; /\* 1ms Timer Counter \*/

static volatile DSTATUS Stat = STA\_NOINIT; /\* Disk Status \*/

static uint8\_t CardType; /\* Type 0:MMC, 1:SDC, 2:Block addressing \*/

static uint8\_t PowerFlag = 0; /\* Power flag \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SPI functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* slave select \*/

static void SELECT(void)

{

HAL\_GPIO\_WritePin(SD\_CS\_PORT, SD\_CS\_PIN, GPIO\_PIN\_RESET);

HAL\_Delay(1);

}

/\* slave deselect \*/

static void DESELECT(void)

{

HAL\_GPIO\_WritePin(SD\_CS\_PORT, SD\_CS\_PIN, GPIO\_PIN\_SET);

HAL\_Delay(1);

}

/\* SPI transmit a byte \*/

static void SPI\_TxByte(uint8\_t data)

{

while(!\_\_HAL\_SPI\_GET\_FLAG(HSPI\_SDCARD, SPI\_FLAG\_TXE));

HAL\_SPI\_Transmit(HSPI\_SDCARD, &data, 1, SPI\_TIMEOUT);

}

/\* SPI transmit buffer \*/

static void SPI\_TxBuffer(uint8\_t \*buffer, uint16\_t len)

{

while(!\_\_HAL\_SPI\_GET\_FLAG(HSPI\_SDCARD, SPI\_FLAG\_TXE));

HAL\_SPI\_Transmit(HSPI\_SDCARD, buffer, len, SPI\_TIMEOUT);

}

/\* SPI receive a byte \*/

static uint8\_t SPI\_RxByte(void)

{

uint8\_t dummy, data;

dummy = 0xFF;

while(!\_\_HAL\_SPI\_GET\_FLAG(HSPI\_SDCARD, SPI\_FLAG\_TXE));

HAL\_SPI\_TransmitReceive(HSPI\_SDCARD, &dummy, &data, 1, SPI\_TIMEOUT);

return data;

}

/\* SPI receive a byte via pointer \*/

static void SPI\_RxBytePtr(uint8\_t \*buff)

{

\*buff = SPI\_RxByte();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SD functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* wait SD ready \*/

static uint8\_t SD\_ReadyWait(void)

{

uint8\_t res;

/\* timeout 500ms \*/

Timer2 = 500;

/\* if SD goes ready, receives 0xFF \*/

do {

res = SPI\_RxByte();

} while ((res != 0xFF) && Timer2);

return res;

}

/\* power on \*/

static void SD\_PowerOn(void)

{

uint8\_t args[6];

uint32\_t cnt = 0x1FFF;

/\* transmit bytes to wake up \*/

DESELECT();

for(int i = 0; i < 10; i++)

{

SPI\_TxByte(0xFF);

}

/\* slave select \*/

SELECT();

/\* make idle state \*/

args[0] = CMD0; /\* CMD0:GO\_IDLE\_STATE \*/

args[1] = 0;

args[2] = 0;

args[3] = 0;

args[4] = 0;

args[5] = 0x95; /\* CRC \*/

SPI\_TxBuffer(args, sizeof(args));

/\* wait response \*/

while ((SPI\_RxByte() != 0x01) && cnt)

{

cnt--;

}

DESELECT();

SPI\_TxByte(0XFF);

PowerFlag = 1;

}

/\* power off \*/

static void SD\_PowerOff(void)

{

PowerFlag = 0;

}

/\* check power flag \*/

static uint8\_t SD\_CheckPower(void)

{

return PowerFlag;

}

/\* receive data block \*/

static bool SD\_RxDataBlock(BYTE \*buff, UINT len)

{

uint8\_t token;

/\* timeout 200ms \*/

Timer1 = 200;

/\* loop until receive a response or timeout \*/

do {

token = SPI\_RxByte();

} while((token == 0xFF) && Timer1);

/\* invalid response \*/

if(token != 0xFE) return FALSE;

/\* receive data \*/

do {

SPI\_RxBytePtr(buff++);

} while(len--);

/\* discard CRC \*/

SPI\_RxByte();

SPI\_RxByte();

return TRUE;

}

/\* transmit data block \*/

#if \_USE\_WRITE == 1

static bool SD\_TxDataBlock(const uint8\_t \*buff, BYTE token)

{

uint8\_t resp;

uint8\_t i = 0;

/\* wait SD ready \*/

if (SD\_ReadyWait() != 0xFF) return FALSE;

/\* transmit token \*/

SPI\_TxByte(token);

/\* if it's not STOP token, transmit data \*/

if (token != 0xFD)

{

SPI\_TxBuffer((uint8\_t\*)buff, 512);

/\* discard CRC \*/

SPI\_RxByte();

SPI\_RxByte();

/\* receive response \*/

while (i <= 64)

{

resp = SPI\_RxByte();

/\* transmit 0x05 accepted \*/

if ((resp & 0x1F) == 0x05) break;

i++;

}

/\* recv buffer clear \*/

while (SPI\_RxByte() == 0);

}

/\* transmit 0x05 accepted \*/

if ((resp & 0x1F) == 0x05) return TRUE;

return FALSE;

}

#endif /\* \_USE\_WRITE \*/

/\* transmit command \*/

static BYTE SD\_SendCmd(BYTE cmd, uint32\_t arg)

{

uint8\_t crc, res;

/\* wait SD ready \*/

if (SD\_ReadyWait() != 0xFF) return 0xFF;

/\* transmit command \*/

SPI\_TxByte(cmd); /\* Command \*/

SPI\_TxByte((uint8\_t)(arg >> 24)); /\* Argument[31..24] \*/

SPI\_TxByte((uint8\_t)(arg >> 16)); /\* Argument[23..16] \*/

SPI\_TxByte((uint8\_t)(arg >> 8)); /\* Argument[15..8] \*/

SPI\_TxByte((uint8\_t)arg); /\* Argument[7..0] \*/

/\* prepare CRC \*/

if(cmd == CMD0) crc = 0x95; /\* CRC for CMD0(0) \*/

else if(cmd == CMD8) crc = 0x87; /\* CRC for CMD8(0x1AA) \*/

else crc = 1;

/\* transmit CRC \*/

SPI\_TxByte(crc);

/\* Skip a stuff byte when STOP\_TRANSMISSION \*/

if (cmd == CMD12) SPI\_RxByte();

/\* receive response \*/

uint8\_t n = 10;

do {

res = SPI\_RxByte();

} while ((res & 0x80) && --n);

return res;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* user\_diskio.c functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* initialize SD \*/

DSTATUS SD\_disk\_initialize(BYTE drv)

{

uint8\_t n, type, ocr[4];

/\* single drive, drv should be 0 \*/

if(drv) return STA\_NOINIT;

/\* no disk \*/

if(Stat & STA\_NODISK) return Stat;

/\* power on \*/

SD\_PowerOn();

/\* slave select \*/

SELECT();

/\* check disk type \*/

type = 0;

/\* send GO\_IDLE\_STATE command \*/

if (SD\_SendCmd(CMD0, 0) == 1)

{

/\* timeout 1 sec \*/

Timer1 = 1000;

/\* SDC V2+ accept CMD8 command, http://elm-chan.org/docs/mmc/mmc\_e.html \*/

if (SD\_SendCmd(CMD8, 0x1AA) == 1)

{

/\* operation condition register \*/

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

{

ocr[n] = SPI\_RxByte();

}

/\* voltage range 2.7-3.6V \*/

if (ocr[2] == 0x01 && ocr[3] == 0xAA)

{

/\* ACMD41 with HCS bit \*/

do {

if (SD\_SendCmd(CMD55, 0) <= 1 && SD\_SendCmd(CMD41, 1UL << 30) == 0) break;

} while (Timer1);

/\* READ\_OCR \*/

if (Timer1 && SD\_SendCmd(CMD58, 0) == 0)

{

/\* Check CCS bit \*/

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

{

ocr[n] = SPI\_RxByte();

}

/\* SDv2 (HC or SC) \*/

type = (ocr[0] & 0x40) ? CT\_SD2 | CT\_BLOCK : CT\_SD2;

}

}

}

else

{

/\* SDC V1 or MMC \*/

type = (SD\_SendCmd(CMD55, 0) <= 1 && SD\_SendCmd(CMD41, 0) <= 1) ? CT\_SD1 : CT\_MMC;

do

{

if (type == CT\_SD1)

{

if (SD\_SendCmd(CMD55, 0) <= 1 && SD\_SendCmd(CMD41, 0) == 0) break; /\* ACMD41 \*/

}

else

{

if (SD\_SendCmd(CMD1, 0) == 0) break; /\* CMD1 \*/

}

} while (Timer1);

/\* SET\_BLOCKLEN \*/

if (!Timer1 || SD\_SendCmd(CMD16, 512) != 0) type = 0;

}

}

CardType = type;

/\* Idle \*/

DESELECT();

SPI\_RxByte();

/\* Clear STA\_NOINIT \*/

if (type)

{

Stat &= ~STA\_NOINIT;

}

else

{

/\* Initialization failed \*/

SD\_PowerOff();

}

return Stat;

}

/\* return disk status \*/

DSTATUS SD\_disk\_status(BYTE drv)

{

if (drv) return STA\_NOINIT;

return Stat;

}

/\* read sector \*/

DRESULT SD\_disk\_read(BYTE pdrv, BYTE\* buff, DWORD sector, UINT count)

{

/\* pdrv should be 0 \*/

if (pdrv || !count) return RES\_PARERR;

/\* no disk \*/

if (Stat & STA\_NOINIT) return RES\_NOTRDY;

/\* convert to byte address \*/

if (!(CardType & CT\_SD2)) sector \*= 512;

SELECT();

if (count == 1)

{

/\* READ\_SINGLE\_BLOCK \*/

if ((SD\_SendCmd(CMD17, sector) == 0) && SD\_RxDataBlock(buff, 512)) count = 0;

}

else

{

/\* READ\_MULTIPLE\_BLOCK \*/

if (SD\_SendCmd(CMD18, sector) == 0)

{

do {

if (!SD\_RxDataBlock(buff, 512)) break;

buff += 512;

} while (--count);

/\* STOP\_TRANSMISSION \*/

SD\_SendCmd(CMD12, 0);

}

}

/\* Idle \*/

DESELECT();

SPI\_RxByte();

return count ? RES\_ERROR : RES\_OK;

}

/\* write sector \*/

#if \_USE\_WRITE == 1

DRESULT SD\_disk\_write(BYTE pdrv, const BYTE\* buff, DWORD sector, UINT count)

{

/\* pdrv should be 0 \*/

if (pdrv || !count) return RES\_PARERR;

/\* no disk \*/

if (Stat & STA\_NOINIT) return RES\_NOTRDY;

/\* write protection \*/

if (Stat & STA\_PROTECT) return RES\_WRPRT;

/\* convert to byte address \*/

if (!(CardType & CT\_SD2)) sector \*= 512;

SELECT();

if (count == 1)

{

/\* WRITE\_BLOCK \*/

if ((SD\_SendCmd(CMD24, sector) == 0) && SD\_TxDataBlock(buff, 0xFE))

count = 0;

}

else

{

/\* WRITE\_MULTIPLE\_BLOCK \*/

if (CardType & CT\_SD1)

{

SD\_SendCmd(CMD55, 0);

SD\_SendCmd(CMD23, count); /\* ACMD23 \*/

}

if (SD\_SendCmd(CMD25, sector) == 0)

{

do {

if(!SD\_TxDataBlock(buff, 0xFC)) break;

buff += 512;

} while (--count);

/\* STOP\_TRAN token \*/

if(!SD\_TxDataBlock(0, 0xFD))

{

count = 1;

}

}

}

/\* Idle \*/

DESELECT();

SPI\_RxByte();

return count ? RES\_ERROR : RES\_OK;

}

#endif /\* \_USE\_WRITE \*/

/\* ioctl \*/

DRESULT SD\_disk\_ioctl(BYTE drv, BYTE ctrl, void \*buff)

{

DRESULT res;

uint8\_t n, csd[16], \*ptr = buff;

WORD csize;

/\* pdrv should be 0 \*/

if (drv) return RES\_PARERR;

res = RES\_ERROR;

if (ctrl == CTRL\_POWER)

{

switch (\*ptr)

{

case 0:

SD\_PowerOff(); /\* Power Off \*/

res = RES\_OK;

break;

case 1:

SD\_PowerOn(); /\* Power On \*/

res = RES\_OK;

break;

case 2:

\*(ptr + 1) = SD\_CheckPower();

res = RES\_OK; /\* Power Check \*/

break;

default:

res = RES\_PARERR;

}

}

else

{

/\* no disk \*/

if (Stat & STA\_NOINIT) return RES\_NOTRDY;

SELECT();

switch (ctrl)

{

case GET\_SECTOR\_COUNT:

/\* SEND\_CSD \*/

if ((SD\_SendCmd(CMD9, 0) == 0) && SD\_RxDataBlock(csd, 16))

{

if ((csd[0] >> 6) == 1)

{

/\* SDC V2 \*/

csize = csd[9] + ((WORD) csd[8] << 8) + 1;

\*(DWORD\*) buff = (DWORD) csize << 10;

}

else

{

/\* MMC or SDC V1 \*/

n = (csd[5] & 15) + ((csd[10] & 128) >> 7) + ((csd[9] & 3) << 1) + 2;

csize = (csd[8] >> 6) + ((WORD) csd[7] << 2) + ((WORD) (csd[6] & 3) << 10) + 1;

\*(DWORD\*) buff = (DWORD) csize << (n - 9);

}

res = RES\_OK;

}

break;

case GET\_SECTOR\_SIZE:

\*(WORD\*) buff = 512;

res = RES\_OK;

break;

case CTRL\_SYNC:

if (SD\_ReadyWait() == 0xFF) res = RES\_OK;

break;

case MMC\_GET\_CSD:

/\* SEND\_CSD \*/

if (SD\_SendCmd(CMD9, 0) == 0 && SD\_RxDataBlock(ptr, 16)) res = RES\_OK;

break;

case MMC\_GET\_CID:

/\* SEND\_CID \*/

if (SD\_SendCmd(CMD10, 0) == 0 && SD\_RxDataBlock(ptr, 16)) res = RES\_OK;

break;

case MMC\_GET\_OCR:

/\* READ\_OCR \*/

if (SD\_SendCmd(CMD58, 0) == 0)

{

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

{

\*ptr++ = SPI\_RxByte();

}

res = RES\_OK;

}

default:

res = RES\_PARERR;

}

DESELECT();

SPI\_RxByte();

}

return res;

}

================================================================================================================================================

* Now we will modify the existing code.
* Open the **Core/Src/stm32f1xx\_it.c** file.
* Add the below line between “**USER CODE BEGIN PV**” and “**USER CODE END PV**“.

======================================================================== extern uint16\_t Timer1, Timer2;

========================================================================

* in the **SysTick\_Handler**function, add the below code between the **USER CODE BEGIN SysTick\_IRQn 0** and **USER CODE END SysTick\_IRQn 0**.

========================================================================

if(Timer1 > 0)

Timer1--;

if(Timer2 > 0)

Timer2--;

* Now, open the **FATFS/Target/user\_diskio.c**.
* Include the **fatfs\_sd\_card.h** (**#include <fatfs\_sd\_card.h>**)
* In that, add the below line to the function **USER\_initialize**.

========================================================================

return SD\_disk\_initialize(pdrv);

========================================================================

* Then add the below line to the function **USER\_status**.

========================================================================

return SD\_disk\_status(pdrv);

========================================================================

* Then add the below line to the function **USER\_read**.

========================================================================

return SD\_disk\_read(pdrv, buff, sector, count);

========================================================================

* Then add the below line to the function **USER\_write**.

========================================================================

return SD\_disk\_write(pdrv, buff, sector, count);

========================================================================

* Finally, add the below line to the function **USER\_ioctl**.

========================================================================

return SD\_disk\_ioctl(pdrv, cmd, buff);

================================================================================================================================================

* Copy this code add in main.c user\_code\_begin\_1 middle

========================================================================

int \_write(int file,char \*ptr,int len)

{

int i=0;

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

ITM\_SendChar((\*ptr++));

return len;

}

================================================================================================================================================

* Add same header in main.h

#include<stdio.h>

#include<stdbool.h>

#include<string.h>

========================================================================

void process\_SD\_card( void )

{

FATFS FatFs; //Fatfs handle

FIL fil; //File handle

FRESULT fres; //Result after operations

char buf[100];

do

{

//Mount the SD Card

fres = f\_mount(&FatFs, "", 1); //1=mount now

if (fres != FR\_OK)

{

printf("No SD Card found : (%i)\r\n", fres);

break;

}

printf("SD Card Mounted Successfully!!!\r\n");

//Read the SD Card Total size and Free Size

FATFS \*pfs;

DWORD fre\_clust;

uint32\_t totalSpace, freeSpace;

f\_getfree("", &fre\_clust, &pfs);

totalSpace = (uint32\_t)((pfs->n\_fatent - 2) \* pfs->csize \* 0.5);

freeSpace = (uint32\_t)(fre\_clust \* pfs->csize \* 0.5);

printf("TotalSpace : %lu bytes, FreeSpace = %lu bytes\n", totalSpace, freeSpace);

//Open the file

fres = f\_open(&fil, "logs.txt", FA\_WRITE | FA\_READ | FA\_CREATE\_ALWAYS);

if(fres != FR\_OK)

{

printf("File creation/open Error : (%i)\r\n", fres);

break;

}

printf("Writing data!!!\r\n");

//write the data

f\_puts("Welcome to Welcome Bitsilica", &fil);

//close your file

f\_close(&fil);

//Open the file

fres = f\_open(&fil, "logs.txt", FA\_READ);

if(fres != FR\_OK)

{

printf("File opening Error : (%i)\r\n", fres);

break;

}

//read the data

f\_gets(buf, sizeof(buf), &fil);

printf("Read Data : %s\n", buf);

//close your file

f\_close(&fil);

printf("Closing File!!!\r\n");

#if 0

//Delete the file.

fres = f\_unlink(logs.txt);

if (fres != FR\_OK)

{

printf("Cannot able to delete the file\n");

}

#endif

} while( false );

//We're done, so de-mount the drive

f\_mount(NULL, "", 0);

printf("SD Card Unmounted Successfully!!!\r\n");

}

================================================================================================================================================