SD Card via SPI

rev1.0 24/03/2020

Manage files in a SD card via SPI: Read, Write and Remove files from SD cards

PREREQUISITES

Software needed:

- STM32IDE

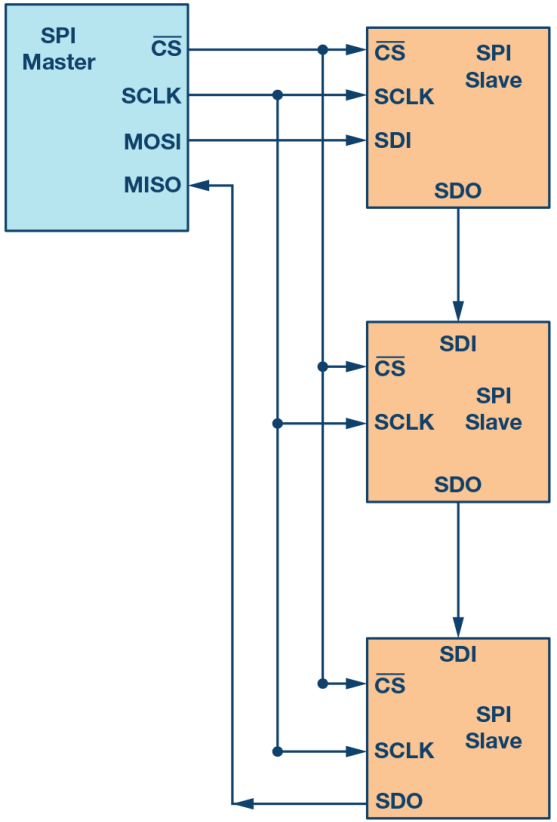
Hardware used in this example:

- **NUCLEO-F446ZE**
- **SPI SD CARD Module**

SPI (Serial Peripheral Interface)

The **Serial Peripheral Interface** (**SPI**) is a synchronous serial communication interface specification used for short-distance communication.

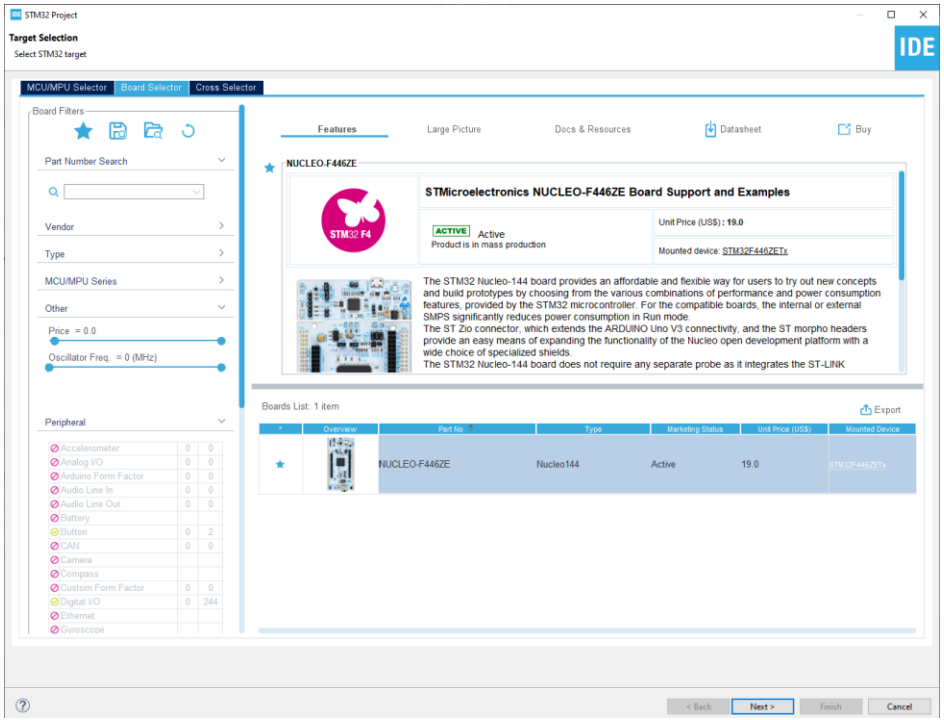
SPI devices communicate in full duplex mode using a master slave architecture with a single master. Multiple slave-devices are supported through selection with individual chip select (CS) lines.



Start a new project

From the stm32IDE software click on File -> New -> STM32 Project.

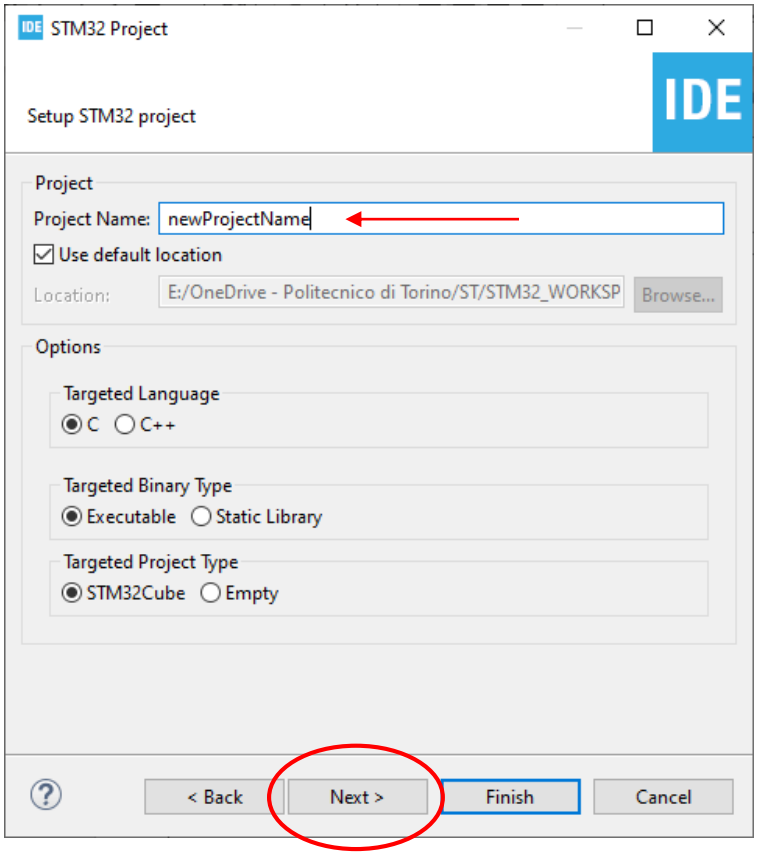
Select your board or your uC and click *next.*



Start a new project

Type the name of your project and click next.

By default the project will be created in the workspace folder.



Start a new project

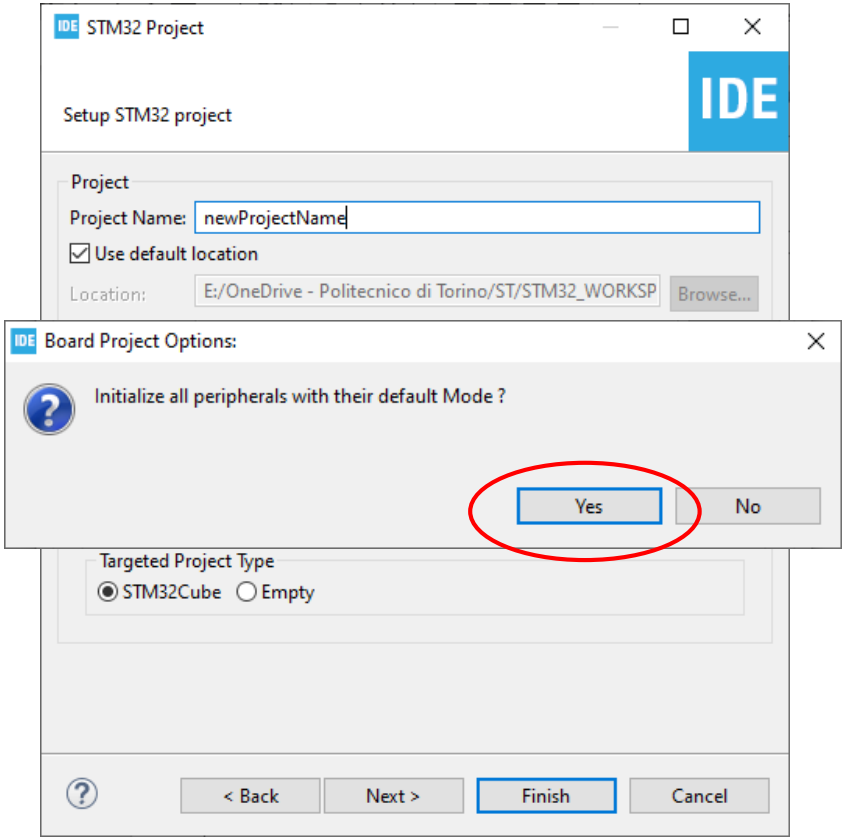
Type the name of your project and click next.

By default the project will be created in the *workspace* folder.

The *STM32IDE* has the option to initialize all the peripheral with their ***default*** mode:

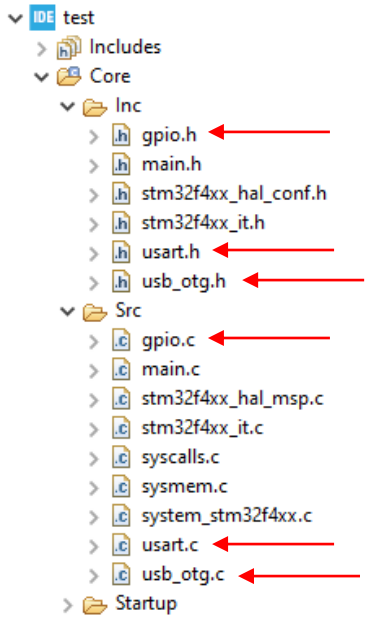
Clicking *Yes* the *USART3*, all the *LEDs* and the blue *UserButton* will be configured as default.

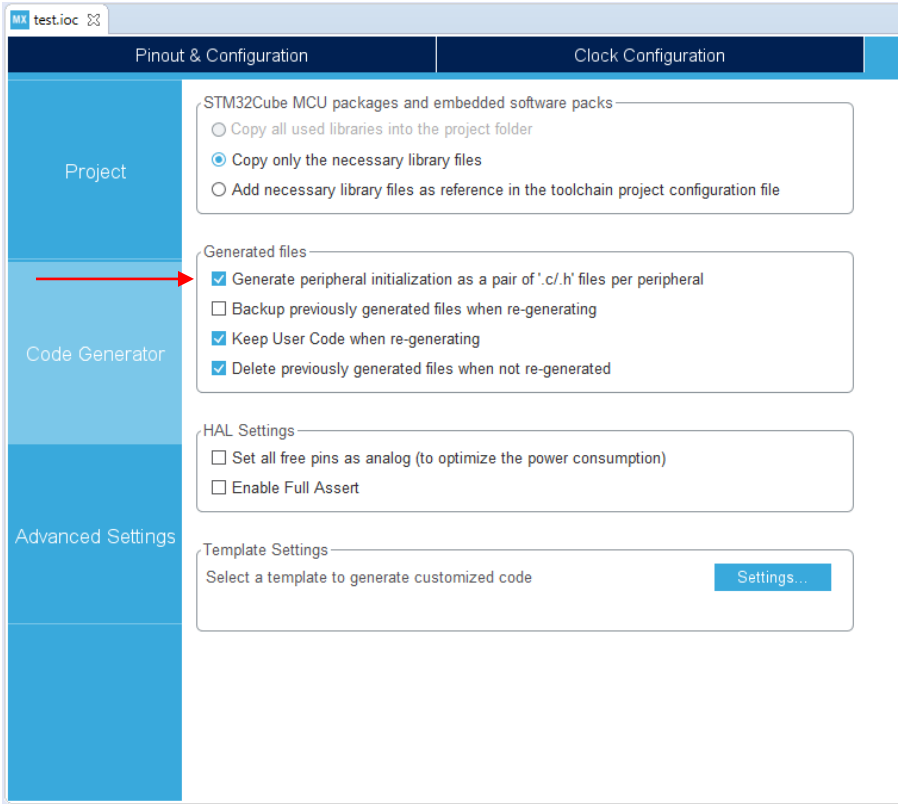
Click *Yes*.



Project Manager

In the Code Generator Tab check the ***Generate peripheral initialization […]*** box: each periperhal will have a disting *periph.c* and *periph.h* files.



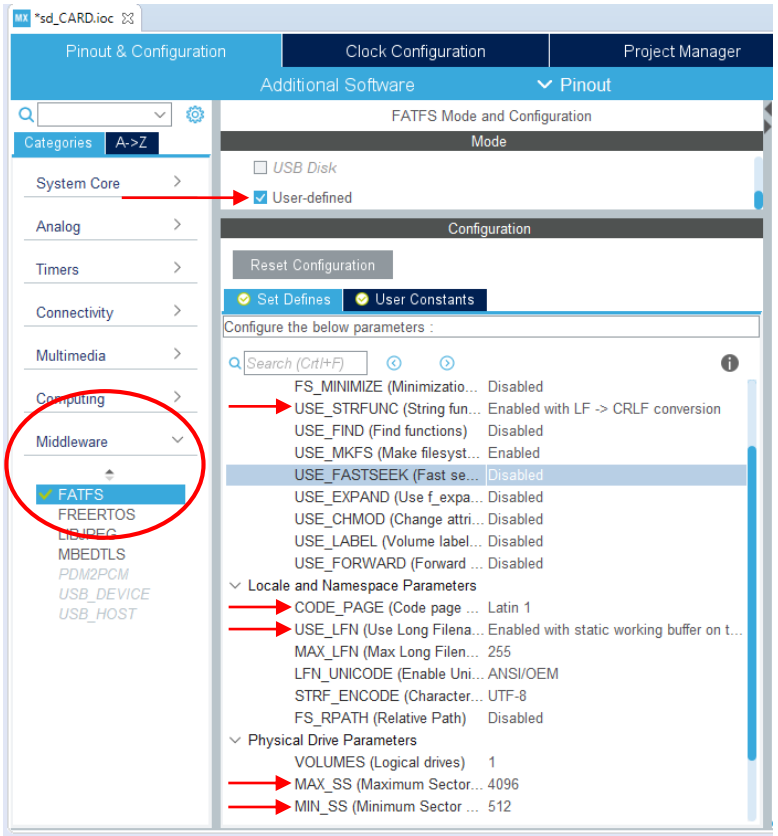


FATFS Configuration

Under the *middleware* section select **FATFS** then check *User-defined* Mode.

Pay attenction for all the configuration setting, in particular the ones indicated by the arrows:

- *USE_STRFUNC :* enabled with LF->CRLF conversion
- *CODE_PAGE :* Latin1
- *USE_LFN :* Enabled with static working buffer […]
- *MAX_SS :* 4096
- *MIN_SS :* 512

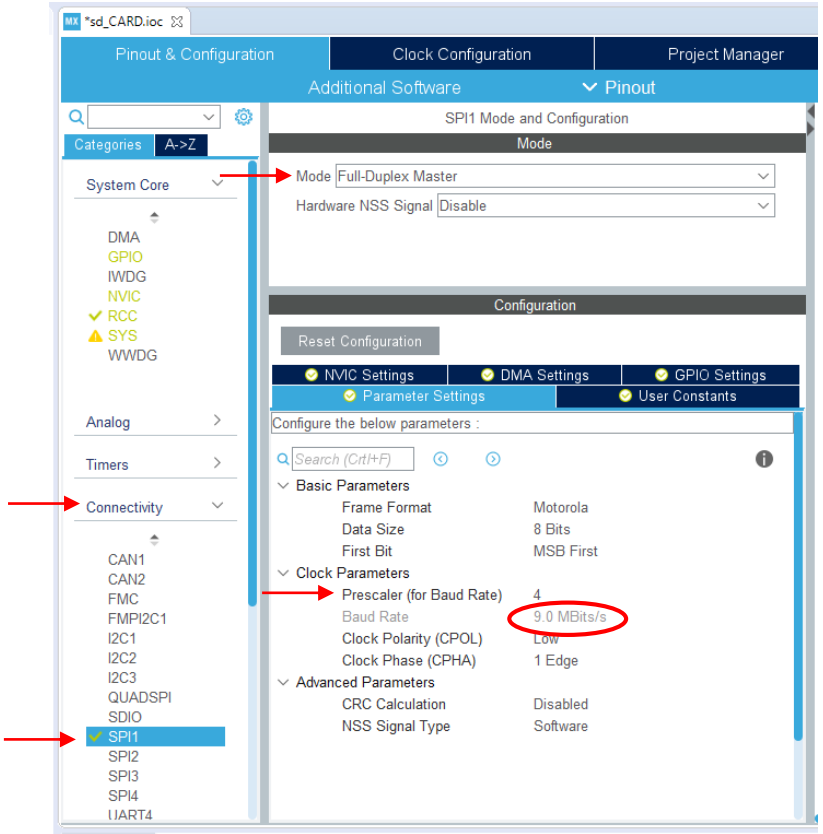


SPI Configuration

For this example we use the SPI1 bus.

Go under *connectivity -> SPI1* and enable it in ***Full-Duplex Master.***

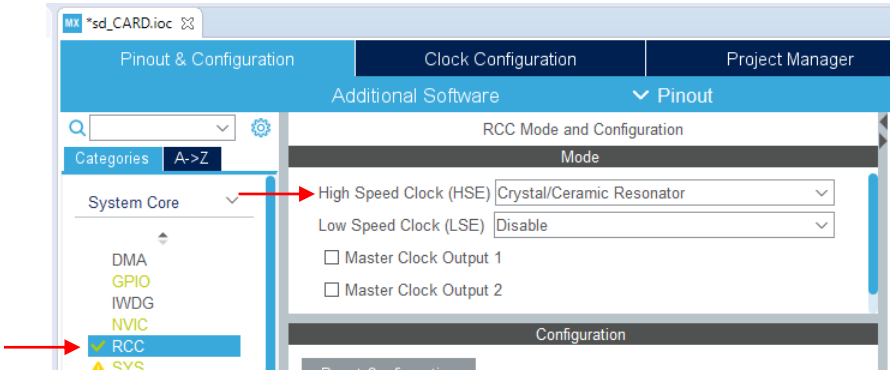
Set the prescaler in order to have a BaudRate around 9 MBits/s (7-12 Mbits/s are resonable values).



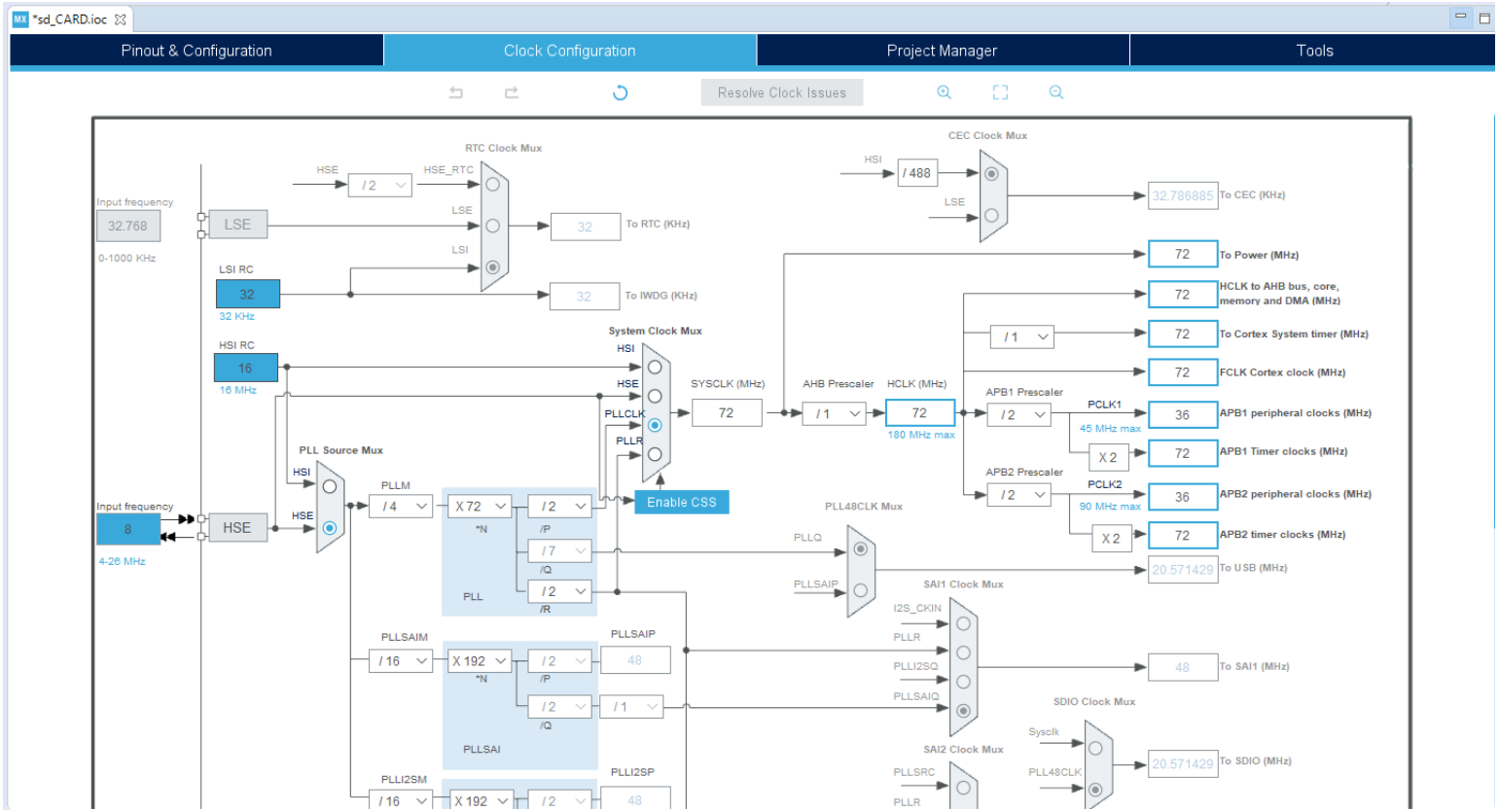
RCC configuration

In our case we need a prescaler of 4 since we have configured the RCC and the clock in this way:

- *High Speed Clock (HSE)* : Crystal/Ceramic Resonator
- *HCLK* (in the clock configuration tab) : 72MHz

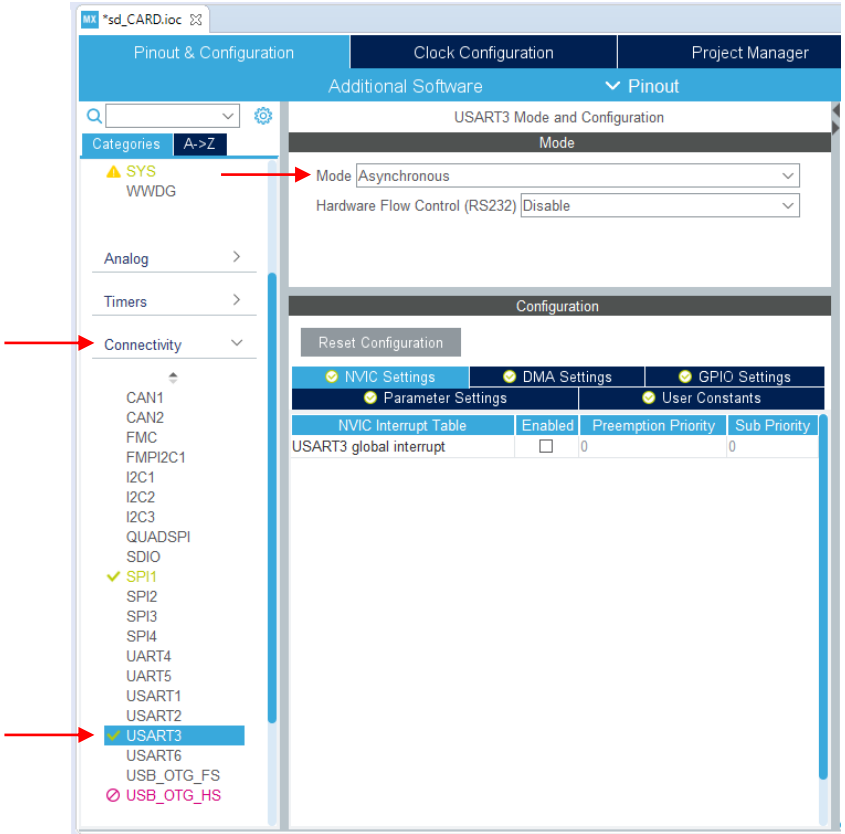


Clock Configuration



USART Configuration

For debugging purposes we use the USART bus, just go in the USART3 configuration tab and enable it in *Asynchronous* mode.

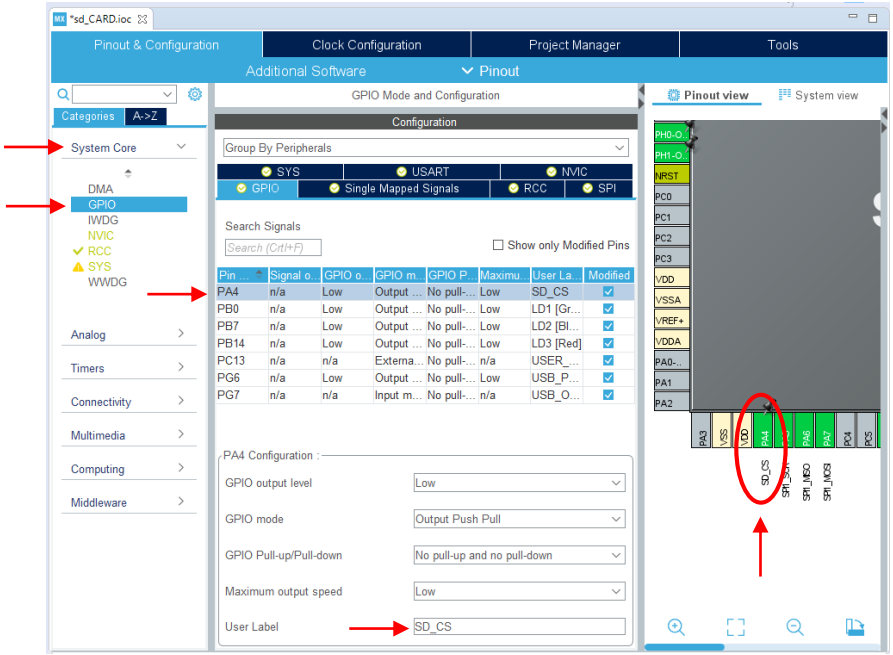


GPIO Configuration

As we described in the before, a CS pin is needed in order to communicate with the SD card.

Go in the *GPIO Tab* end enable the CS Pin as *GPIO OUTPUT* (in this case we used the pin PA4).

All the configuration ends here, so now you can generate the code (click on the generate icon).



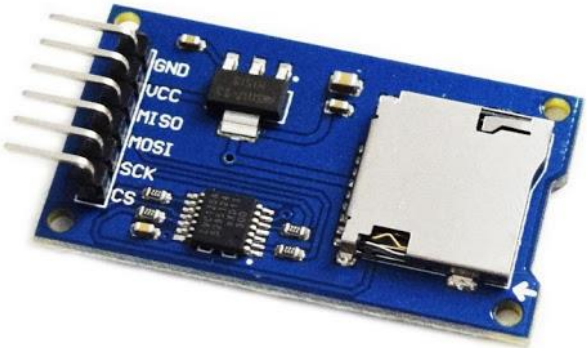
Connect the SD card module

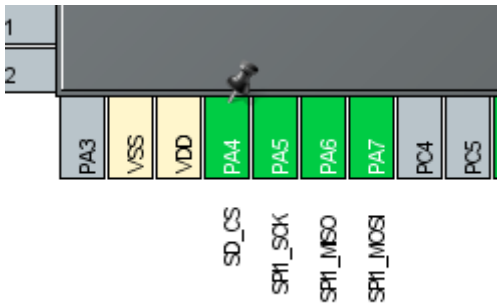
The module used for this example it the following one.

As you can see in the header we have the power pins (GND and VCC) and the SPI communication pins.

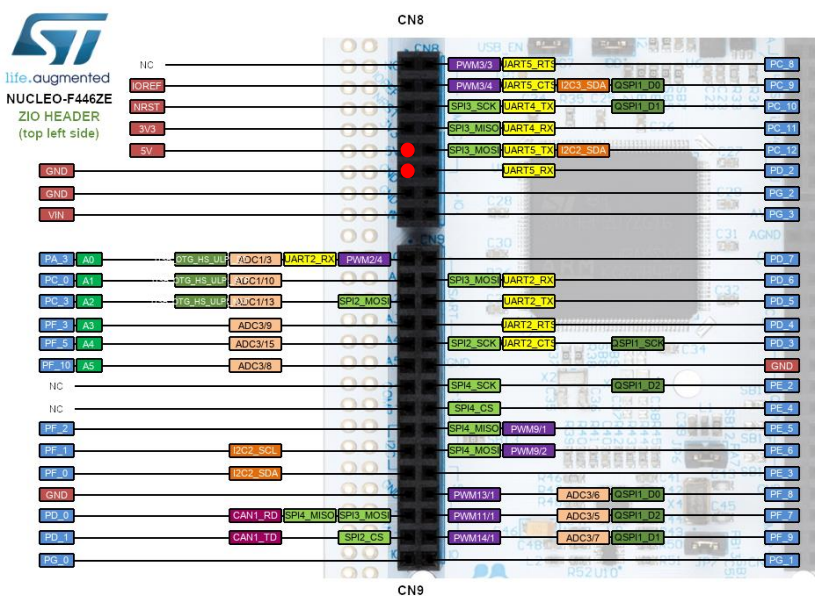
Simply connect the pins as :

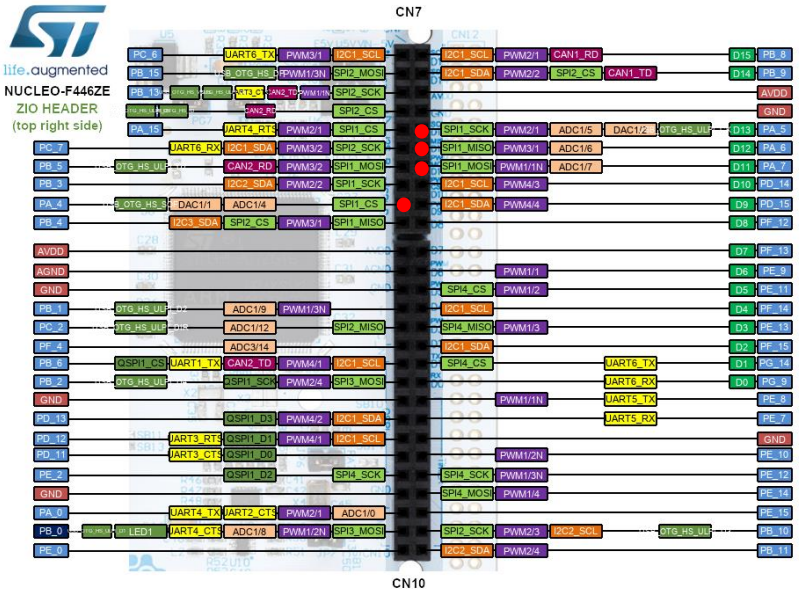
- *GND -> GND*
- *VCC -> 5V*
- *MISO -> PA6*
- *MOSI -> PA7*
- *SCK -> PA4*
- *CS -> PA5*





NUCLEO Pins





For more info go here.

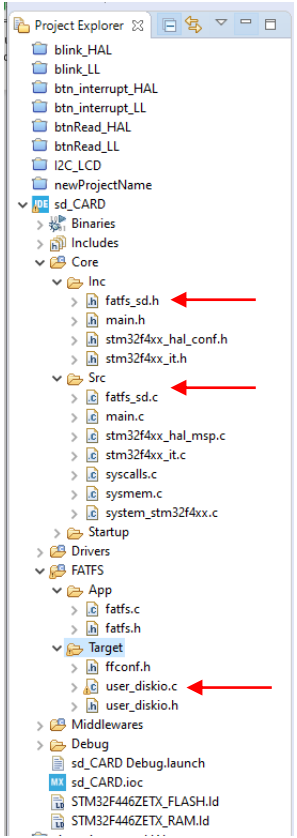
Including the libraries

By the project manager add the following files (just drag & drop them in the correct folder):

- *fatfs_sd.h*
- *fatfs_sd.c*

Replace the following files:

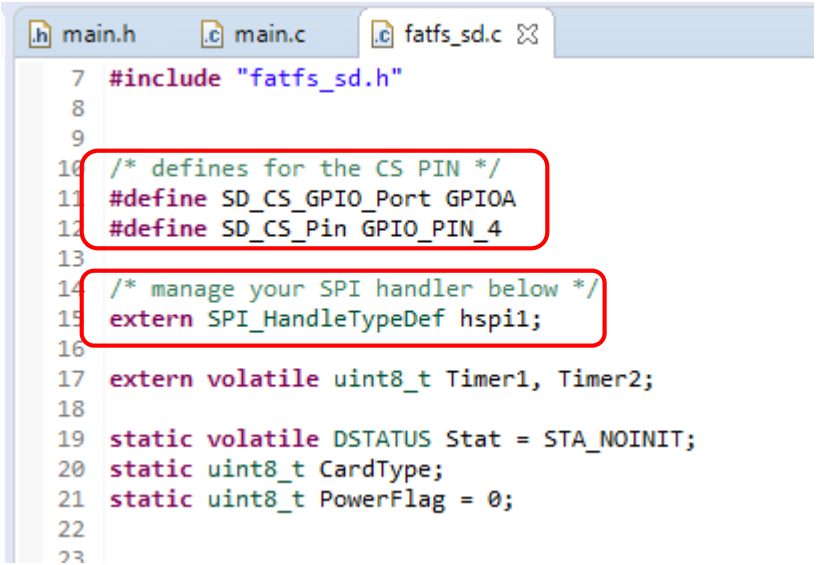
- *user_diskio.c*
- *stm32f4xx_it.c*



Check the correct CS pin

Open the file ***fatfs_sd.c*** and check if the CS pin is the correct one, in our case the CS pin is PA4.

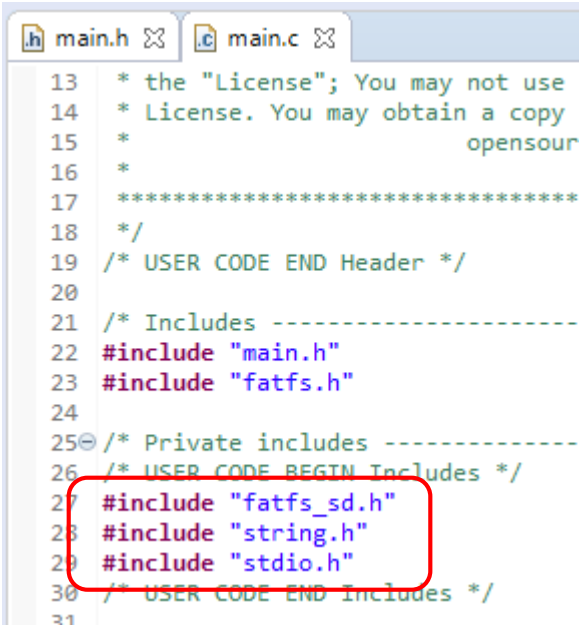
Check the SPI handler too, in our example we used the *SPI1*.



Including the libraries

Open the ***main.c*** file and include the following libraries:

- *fatfs_sd.h*
- *string.h*
- *stdio.h*



Create variables for SD card

Scroll down and declare the following variables in order to manage the SD Card, them will be useful for check the size of the SD, check for errors or tasks like open, close or remove files.

main.h main.c

```
43
44  /* USER CODE END PM */
45
46  /* Private variables ---------------------------------------------------------*/
47  SPI_HandleTypeDef hspi1;
48
49  UART_HandleTypeDef huart3;
50
51  /* USER CODE BEGIN PV */
52
53
54  /*************************** SD CARD VARIABLES ***************************/
55  FATFS fs;  // file system
56  FIL fil;  // file
57  FRESULT fresult;  // to store the result
58  char buffer[1024]; // to store data
59  char To_Open[1024];
60  UINT br, bw;   // file read/write count
61
62  /* capacity related variables */
63  FATFS *pfs;
64  DWORD fre_clust;
65  uint32_t total, free_space;
66
67
68
```

UART Send function

In this example we use the USART Bus to debug our application.

Write the ***send_uart()*** function to send string via UART easily, then write the ***bufsize()*** and ***bufclear()*** functions to manage the buffer.

main.h | main.c | fatfs_sd.c

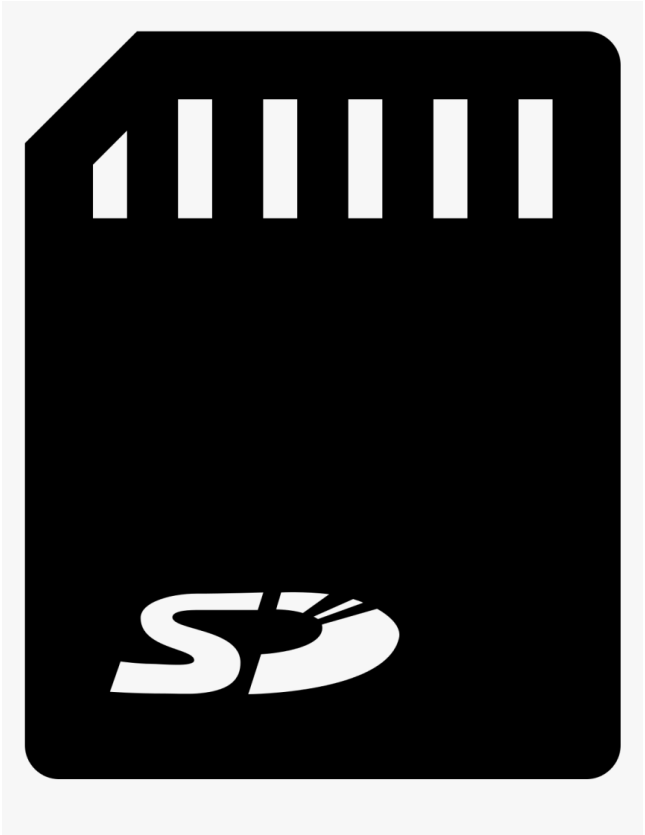
```
404 }
405
406 /* USER CODE BEGIN 4 */
407
408 /* to send the data to the uart */
409 void send_uart(char *string) {
410     uint8_t len = strlen(string);
411     HAL_UART_Transmit(&huart3, (uint8_t*) string, len, 2000); // transmit in blocking mode
412 }
413
414 /* to find the size of data in the buffer */
415 int bufsize(char *buf) {
416     int i = 0;
417     while (*buf++ != '\0')
418         i++;
419     return i;
420 }
421
422 /*Clear the buffer*/
423 void bufclear(void)  {
424     for (int i = 0; i < 1024; i++) {
425         buffer[i] = '\0';
426     }
427 }
428
```

SD Card functions

Our program should be able to;

- *Mount the SD card*
- *Check the size of the SD card*
- *Check the free space of the SD card*
- *Open or create a file*
- *Write on a file*
- *Close a file*

Eventually it’s possible to *delete* a file too.



Mount and check capacity

fresult stores the result of the operation, normally should be *FR_OK*, the other states are defined in ff.h file.

To check the freespace we use the ***f_getfree()*** function.

To check if the card has been read correctly we monitor the total capacity, since it should be under 32GB.

main.h | main.c | fatfs_sd.c

```
119
120     /* Mount SD Card */
121     fresult = f_mount(&fs, "", 0);
122     if (fresult != FR_OK)
123         send_uart("error in mounting SD CARD, check connection...\n");
124     else
125         send_uart("SD CARD mounted successfully...\n");
126
127     /************** Card capacity details ********************/
128     /* Check free space */
129     f_getfree("", &fre_clust, &pfs);
130     total = (uint32_t) ((pfs->n_fatent - 2) * pfs->csize * 0.5);
131     if (total < 32000000) {
132         sprintf(buffer, "SD CARD Total Size: \t%lu\n", total);
133         send_uart(buffer);
134         bufclear();
135         free_space = (uint32_t) (fre_clust * pfs->csize * 0.5);
136         sprintf(buffer, "SD CARD Free Space: \t%lu\n", free_space);
137         send_uart(buffer);
138     } else {
139         send_uart("ERROR: can't read SD CARD, check SD...\n");
140
141     }
```

main.h | main.c | fatfs_sd.c | ff.h

```
209
210
211
212 /* File function return code (FRESULT) */
213
214 typedef enum {
215     FR_OK = 0,              /* (0) Succeeded */
216     FR_DISK_ERR,            /* (1) A hard error occurred in the low level disk I/O layer */
217     FR_INT_ERR,             /* (2) Assertion failed */
218     FR_NOT_READY,           /* (3) The physical drive cannot work */
219     FR_NO_FILE,             /* (4) Could not find the file */
220     FR_NO_PATH,             /* (5) Could not find the path */
221     FR_INVALID_NAME,        /* (6) The path name format is invalid */
222     FR_DENIED,              /* (7) Access denied due to prohibited access or directory full */
223     FR_EXIST,               /* (8) Access denied due to prohibited access */
224     FR_INVALID_OBJECT,      /* (9) The file/directory object is invalid */
225     FR_WRITE_PROTECTED,     /* (10) The physical drive is write protected */
226     FR_INVALID_DRIVE,       /* (11) The logical drive number is invalid */
227     FR_NOT_ENABLED,         /* (12) The volume has no work area */
228     FR_NO_FILESYSTEM,       /* (13) There is no valid FAT volume */
229     FR_MKFS_ABORTED,        /* (14) The f_mkfs() aborted due to any problem */
230     FR_TIMEOUT,             /* (15) Could not get a grant to access the volume within defined period */
231     FR_LOCKED,              /* (16) The operation is rejected according to the file sharing policy */
232     FR_NOT_ENOUGH_CORE,     /* (17) LFN working buffer could not be allocated */
233     FR_TOO_MANY_OPEN_FILES, /* (18) Number of open files > _FS_LOCK */
234     FR_INVALID_PARAMETER    /* (19) Given parameter is invalid */
235 } FRESULT;
236
237
```

Write and Read from file

In order to open the file use the function ***f_open()***, as always we check for errors.

f_gets() read the entire file, line by line and put it in the buffer.

For write a string in the file, use the function ***f_puts().***

At the end do not forget to close the file and clear the buffer. Use the ***f_close()*** and ***bufclear()*** functions.

Remember that ***&fil*** is the address of our file.

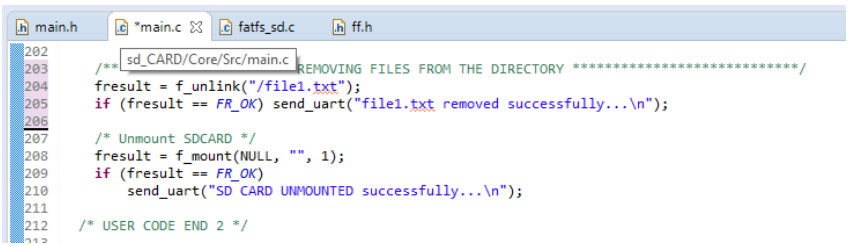
main.h | *main.c | fatfs_sd.c | ff.h

```
140
141     }
142
143     /************ The following operation is using PUTS and GETS *******************/
144
145     /* Open file to write/ create a file if it doesn't exist */
146     strcpy(To_Open,"file1.txt");
147     if( f_open(&fil, To_Open, FA_OPEN_ALWAYS | FA_READ | FA_WRITE) != FR_OK ) {
148         char error [1024];
149         strcpy(error,"ERROR: can't open ");
150         strcat(error, To_Open);
151         strcat(error, "\n\n");
152         send_uart(error);
153     }
154
155     /* Read string from the file */
156         while (f_gets(buffer, sizeof(buffer), &fil)) {
157             /* SWV output */
158             send_uart(buffer);
159             fflush(stdout);
160         }
161
162     /* Writing text */
163     fresult = f_lseek(&fil, fil.fptr);
164     fresult = f_puts("Hello World!\n", &fil);
165
166     /* Close file */
167     f_close(&fil);
168
169     bufclear();
```

Remove a file

If you want to remove any file from the SD card use the ***f_unlink()*** function, check always for errors.

At the end of all operation the SD card should be *unmounted,* so use the ***f_mount()*** function but with the **NULL** parameter.



Useful links

SD card with stm32 using SPI**:**

https://www.youtube.com/watch?v=spVIZO-jbxE&t=519s

How spi works:

https://www.youtube.com/watch?v=AuhFr88mjt0

Quad spi for fast operations:

https://www.youtube.com/watch?v=_-FCtJphSRY