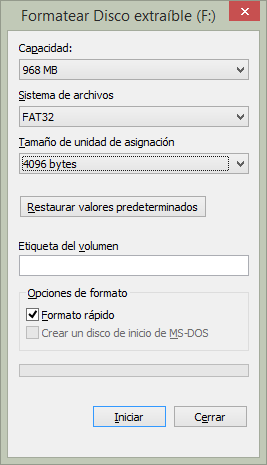
Laboratorio 3 – File Systems utilizando la librería FatFs y Cube MX

¿Qué es un filesystem?

Es un medio organizado para almacenar, recuperar, actualizar datos. Sin un sistema de archivos sería difícil encontrar en la memoria la información específica que buscamos.

Para crear un filesystem la memoria debe ser particionada. Existen varios formatos pero el más utilizado multiplataforma (o lo mismo que funciona tanto en Windows, Linux o Mac) es FAT, que viene de File Allocation Table.

1 – Conseguir una microSD y guardar la información relevante puesto que al formatear perderemos su contenido. En Windows entonces al insertar la microSD formatearla en FAT32. En mi caso es una memoria de 1Gb y los sectores de 4k. Con formato rápido es suficiente.

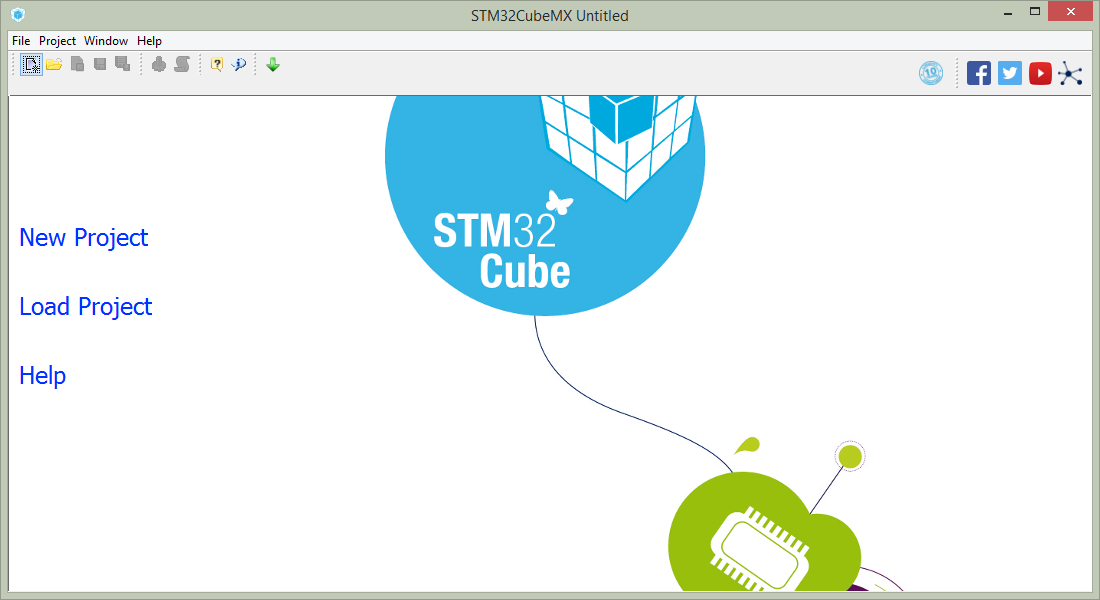


2 – Luego iniciaremos el proceso de creación de nuestro proyecto con Keil y CubeMX.

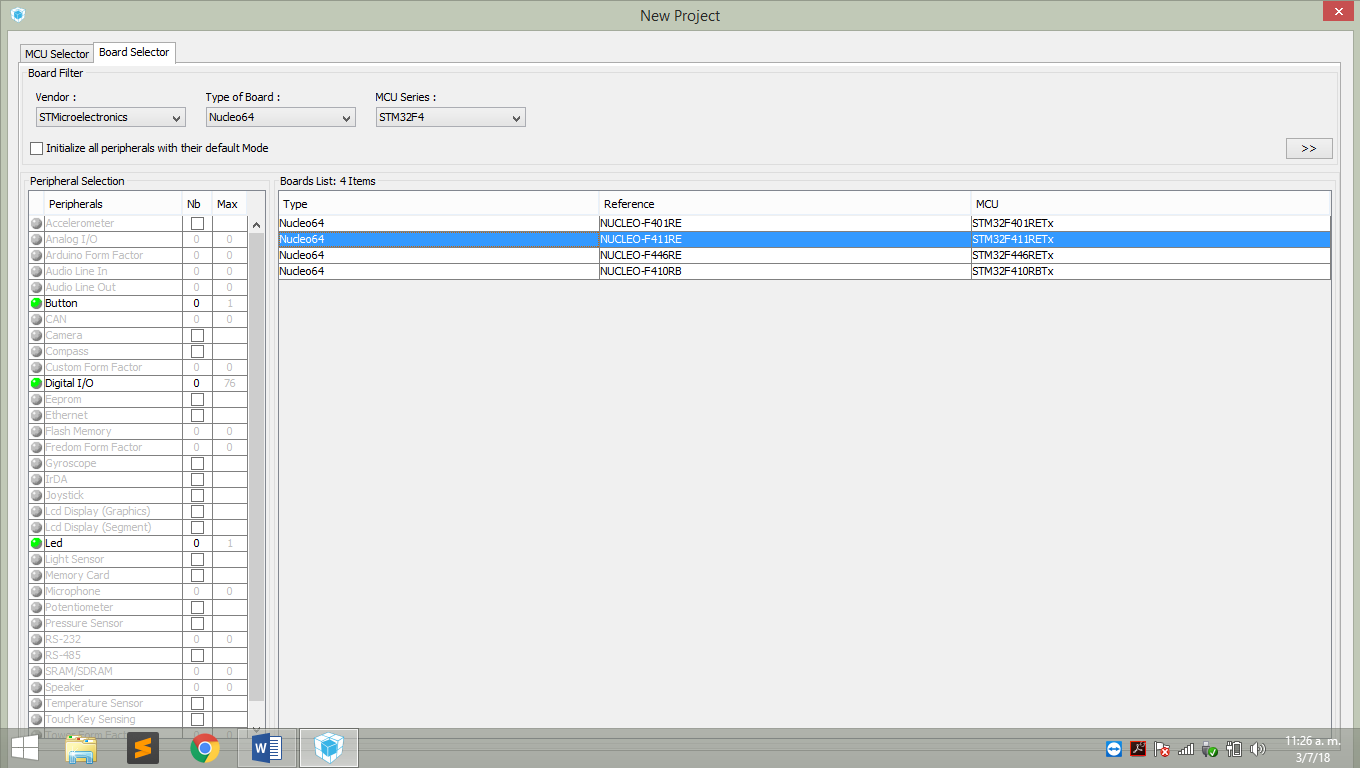
NOTA: Lamentablemente existe un problema a resolver con CubeMX 4.26 y versiones inferiores hasta 4.22. La versión estable es 4.21 y versión de firmware de F4 1.16. Por consiguiente para que este laboratorio funcione deben hacer un downgrade hasta que ST o el profesor solucione el incoveniente.

Por el momento trabaje con CubeMX 4.26, sigua los pasos de la guía, haga que el programa compile y diríjase al profesor con el main.c para generar el código en la laptop y poder ver el ejemplo.

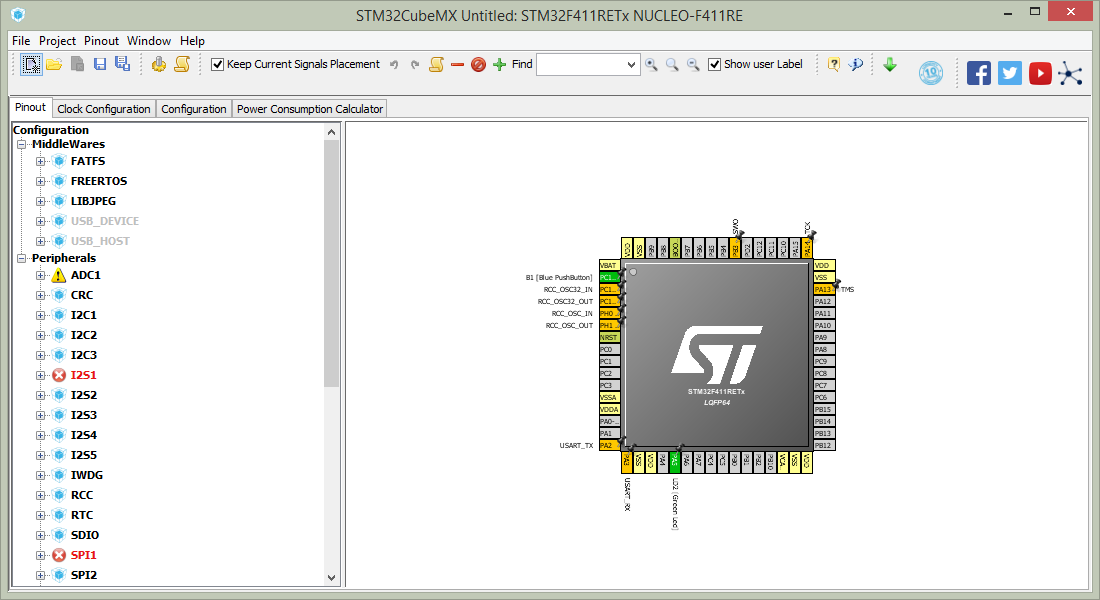
Lanzado CubeMX seleccione nuevo proyecto seguido de su plataforma de trabajo



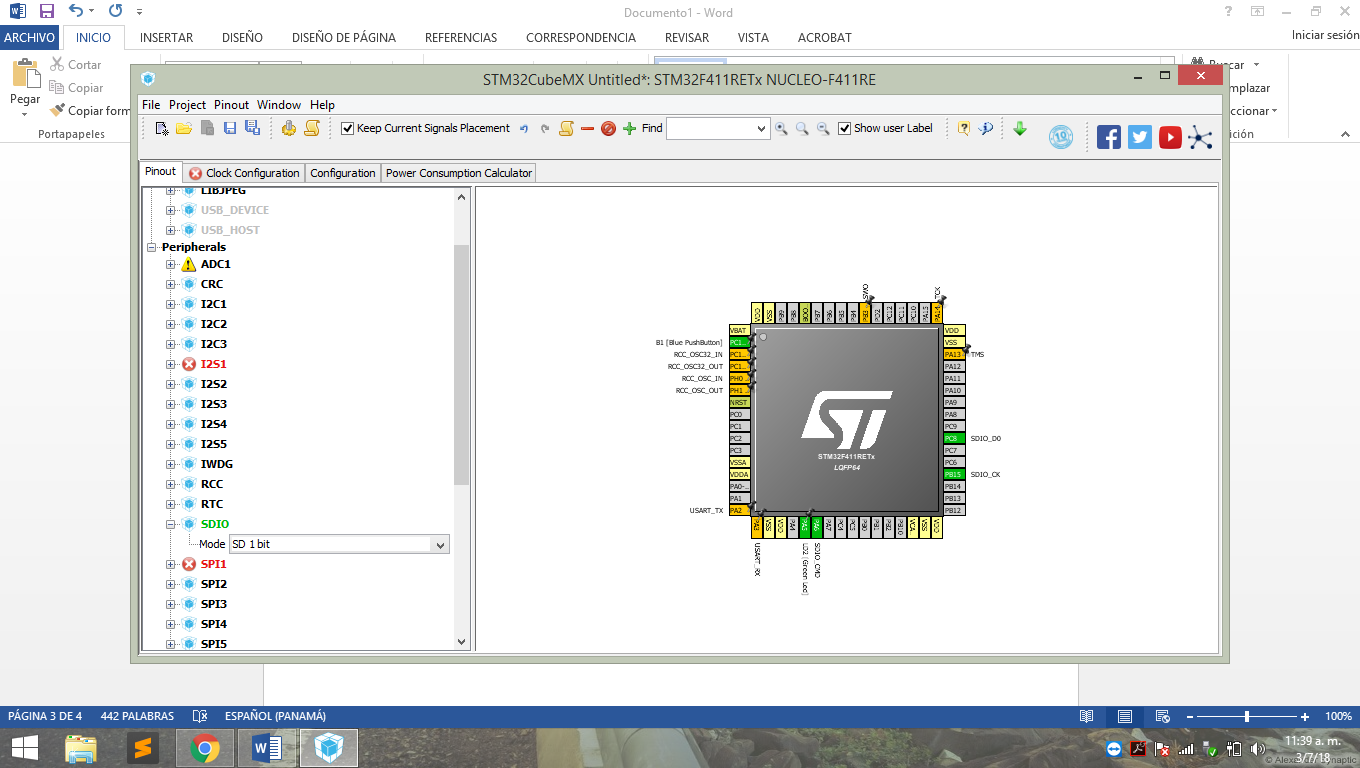
en este caso poseo una NUCLE-F411RE, que se puede conseguir de la pestaña “Board Selector”



3 – Luego de lanzado el proyecto creado se vera el pinout del microcontrolador, con todos los periféricos inicializados que utiliza la tarjeta NUCLEO-F411RE, del lado izquierdo están los posibles periféricos a hablitar en colores, rojo simboliza que probablemente haya un error de inicialización, verde que se está utilizando y bien configurado y amarillo simboliza que hay una configuración parcial



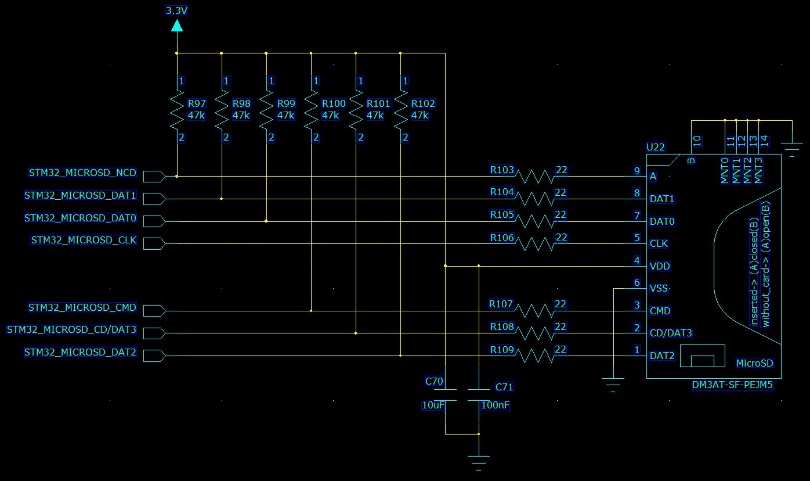
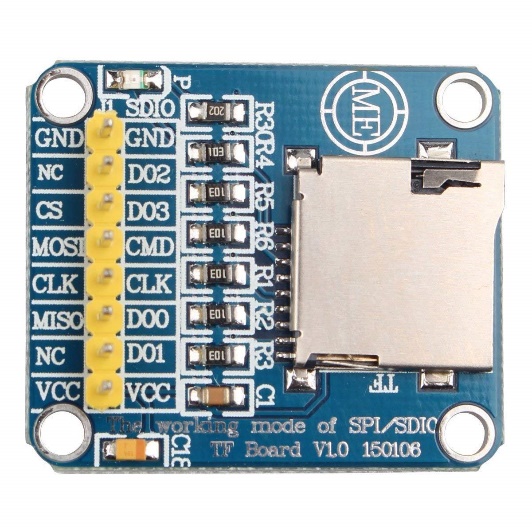
4 – Lo siguiente es habilitar dentro de los periféricos SDIO en modo de 1 bit, también llamado SPI. Notará que se han habilitado 3 pines asociados al periférico de SPI.



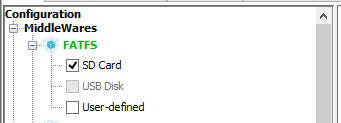
El SPI es un protocolo de comunicación de periféricos de bajo nivel (memorias, sensores, actuadores). Este protocolo consta de señales de reloj (CLK), Master Out Slave In (MOSI), Master In Slave Out (MISO), y el Chip Select (CS o SS) que generalmente se pone a bajo para interactuar con el dispositivo a controlar.



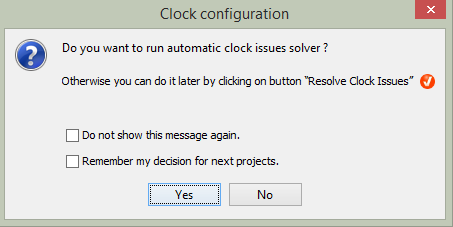
Para las tarjetas SD y microSD se puede poner en modo de 1 bit (SPI) como dijimos anteriormente así como de 4 bits, la diferencia es velocidad de transferencia. La equivalencia de estas señales de ve en la siguiente figura izquierda (pines izq. SPI o 1 bit mode, pines derecha 4 bit mode) y el esquemático recomendado se muestra debajo, sin embargo, con este hardware es suficiente para realizar nuestra demostración



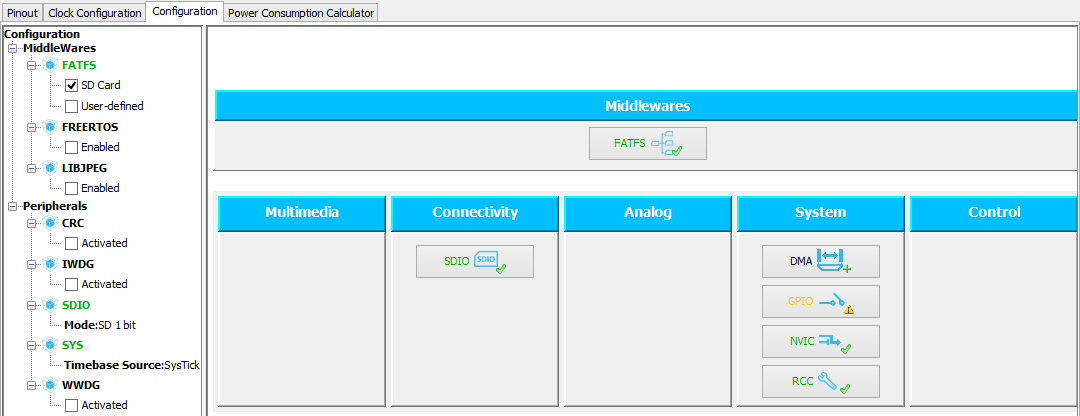
5 – Seguido habilitar la librería de FatFs que será el middleware que nos ayudará a realizar las transferencias de datos a nuestra memoria flash.



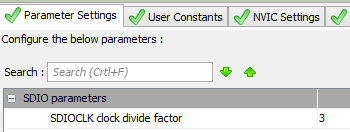
6 – Ya ajustado lo necesario pasamos a la ventana de “Clock configuration”. CubeMX lanzará una ventana a la cual daremos “Yes” para ajustar los inconvenientes causados por el reloj



7 – Pasamos ahora a la ventana de configuración en donde lista todo periférico necesario y configurado para iniciar nuestra aplicación.

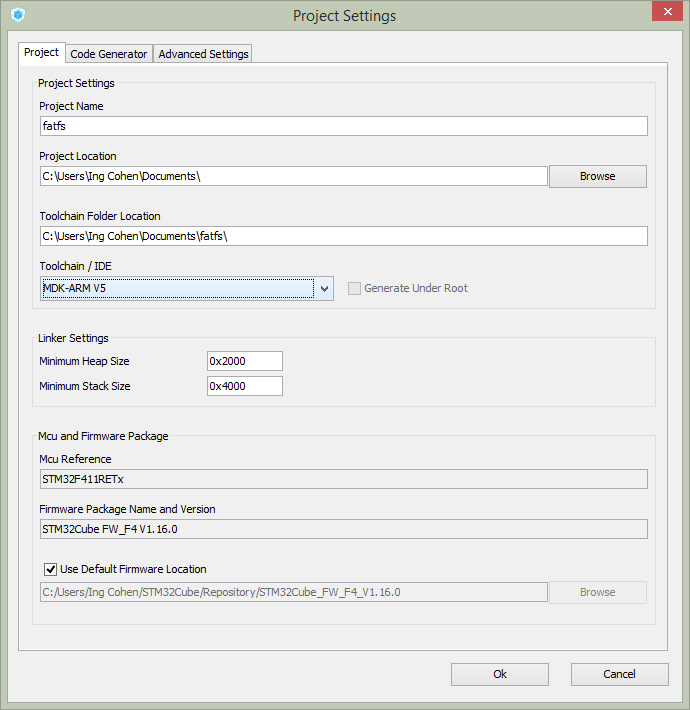


8 - Antes de empezar hay que saber que la SD la velocidad máxima es de 25 MHz, como nuestro bus está a 72MHz habrá que ajustar este valor en la sección de SDIO. Un valor de 3 (72MHz / 25 MHz) es más que suficiente en este caso.



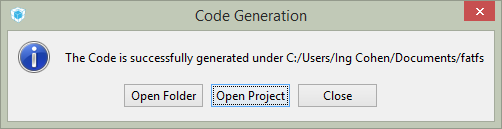
9 – Seguido es ajustar la generación del proyecto en Project/Settings. Aquí damos el nombre del proyecto, la ruta donde se guardará el proyecto.

Imprtante cambiar a Keil MDK-ARM y aumentar los Linker Settings de stack y heap a 0x2000 y 0x4000 aunque pensamos que con 1024 (0x1000) para ambos casos es suficiente.

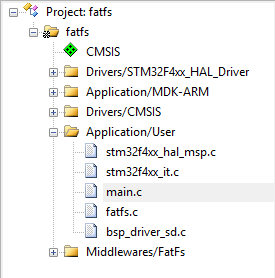


10 – Finalmente generamos el proyecto con el símbolo del engrane .

11 – Abrimos nuestro proyecto en Keil



12 – Seguidamente abrimos el archivo main.c generado para añadir el código para manejo de sistema de archivo



13 – Agregaremos dos variables globales para empezar llamadas fatfs y file.

/\* USER CODE BEGIN PV \*/

/\* Private variables ---------------------------------------------------------\*/

FATFS SDFatFs; /\* Sistema de Archivos de la SD \*/

FIL MyFile; /\* Manejador de Archivo (Estructura) \*/

char SDPath[4]; /\* Path Lógico de la SD \*/

/\* USER CODE END PV \*/

14 – Agregaremos ahora el código referente a filesystem para montar la unidad. La función f\_mount registra la unidad en nuestro sistema bajo el path designado, que también puede ser ingresado a mano, por ejemplo: “/” lo que significa que está en la raíz.

/\* USER CODE BEGIN 2 \*/

if(f\_mount(&SDFatFs, SDPath, 1) == FR\_OK) {

…

}

15 – Registrada la memoria ahora crearemos un archivo, pero antes añadiremos que se ilumine el LED si pasa estas partes, esto no es necesario, solo es un aviso visual para nosotros.

if(f\_mount(&SDFatFs, SDPath, 1) == FR\_OK) {

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(200);

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(200);

}

16 – Añadimos ahora la función de abrir un archivo, este no solo lo abrirá sino que lo creará si no existe en memoria. Ubicar debajo de la segunda línea de HAL\_Delay(200);

char myPath[] = "Text1.txt";

if(f\_open(&MyFile, (const TCHAR \*)myPath, FA\_WRITE | FA\_CREATE\_ALWAYS) == FR\_OK) {

El primer argumento es el puntero al manejador del archivo, el segundo es el path del archivo y el tercer argumento nos dice que lo cree si no existe. Si todo está correcto retornará FR\_OK

17 – Abierto el archivo ahora agreguemos algo. Los argumentos son el puntero del manejador de archivo, la data a escribir, la cantidad y el puntero al byte de error.

char data[] = "My data to write";

UINT testbytes;

if(f\_write(&MyFile, data, sizeof(data), &testbytes) == FR\_OK) {

}

18 – Lo ultimo es cerrar el archivo pues el sistema lo mantiene abierto. También agreguemos algo de LEDs para avisar que cumplio hasta este punto.

f\_close(&MyFile);

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(200);

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(200);

19 – El código se observará como sigue:

if(f\_mount(&SDFatFs, SDPath, 1) == FR\_OK) {

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(200);

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(200);

char myPath[] = "Text1.txt";

if(f\_open(&MyFile, (const TCHAR \*)myPath, FA\_WRITE | FA\_CREATE\_ALWAYS) == FR\_OK) {

char data[] = "My data to write";

UINT testbytes;

if(f\_write(&MyFile, data, sizeof(data), &testbytes) == FR\_OK) {

}

f\_close(&MyFile);

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(200);

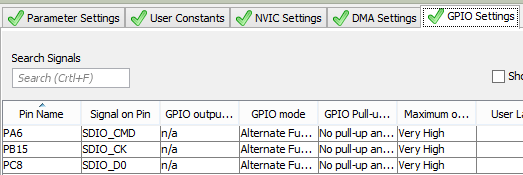
HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

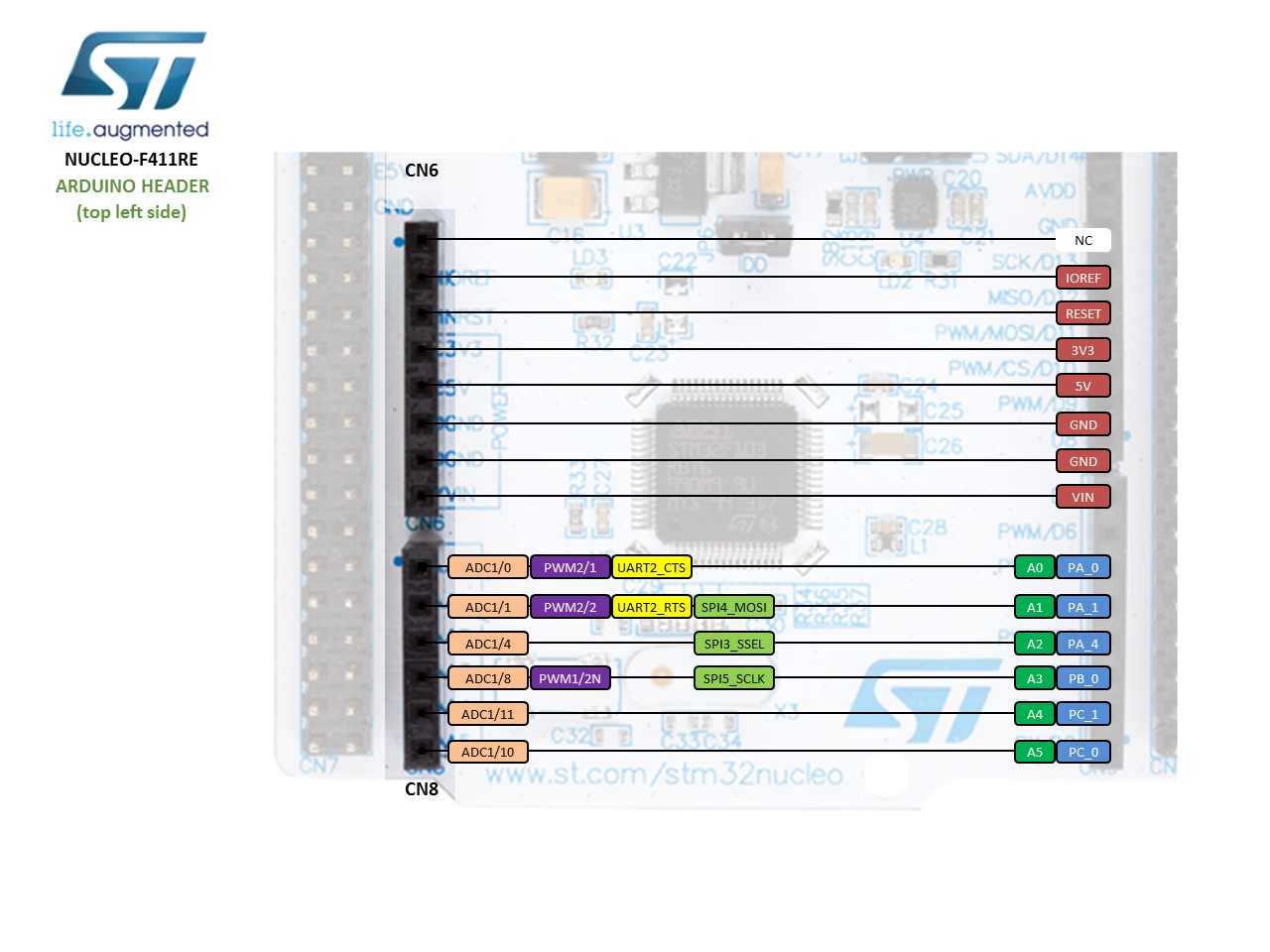
HAL\_Delay(200);

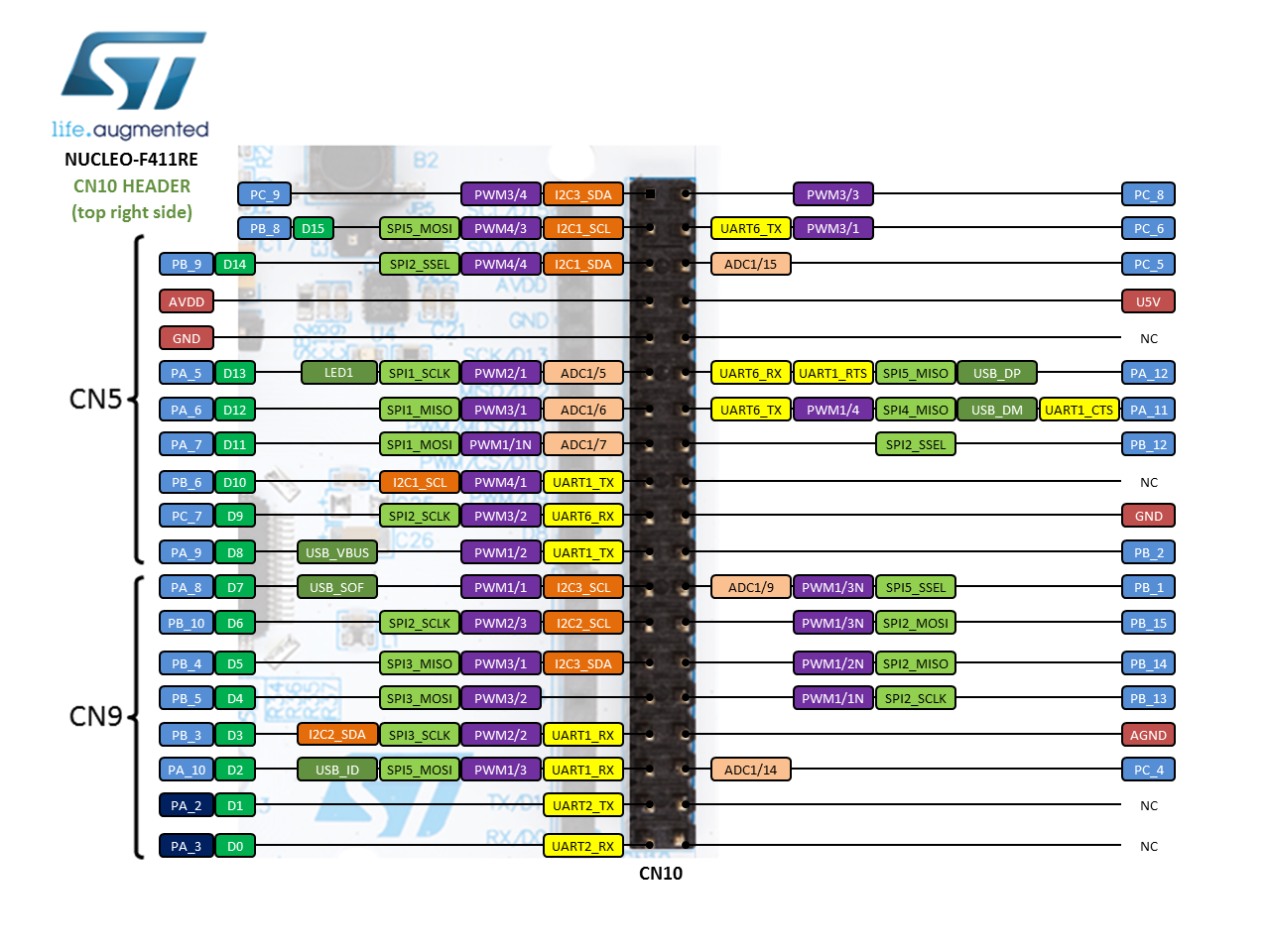
}

}

20 – Al compilar el código nos debe dar sin error. Conectemos la Nucleo, los pines necesarios a nuestra board de microSD. Usaremos 3.3V y GND del header izquierdo hembra y los pines de Morpho machos del lado derecho







21 – Finalmente al conectar, inserte la tarjeta, ejecute el programa (Reset). Saque la microSD y verifique que se ha escrito algo en la tarjeta SD.

22 – Solo como comentario. El código se puede mejorar un poco más, p.e., si la tarjeta no tiene el formato correcto que no sea FAT32 o falte formato podríamos añadir una función automática para formatear la tarjeta insertada.

if(f\_mkfs((TCHAR const\*)SDPath, FM\_ANY, 0, buffer, sizeof(buffer)) != FR\_OK) {

Error\_Handler();

}

else {

// open file

}

23 – Agreguemos este segmento de código debajo del último HAL\_Delay(200). Esto solo es para verificar que estaremos utilizando bien las funciones de lectura escritura

char dataread[sizeof(data)];

if(f\_open(&MyFile, (const TCHAR \*)myPath, FA\_READ | FA\_OPEN\_ALWAYS) == FR\_OK) {

if(f\_read(&MyFile, dataread, sizeof(dataread), &testbytes) == FR\_OK) {

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(200);

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

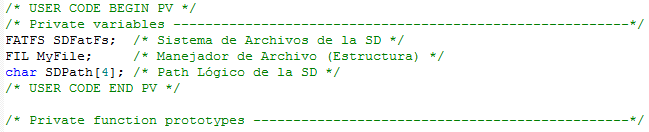
HAL\_Delay(200);

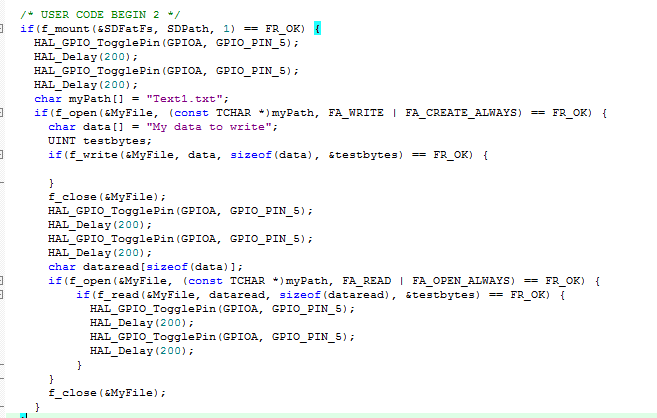
}

}

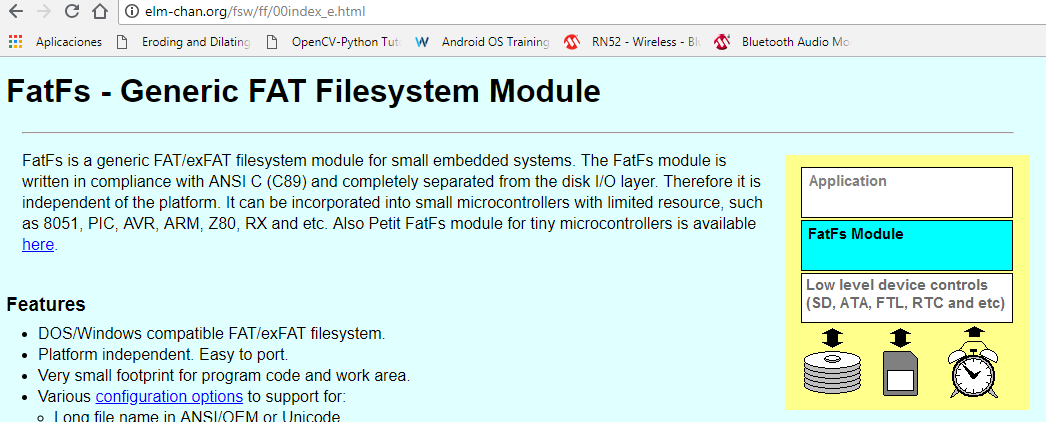
f\_close(&MyFile);

Al final el código debe verse como sigue para que todo funcione correctamente:





24 – Este es un archivo de ejemplo que le sirve como base para utilizar la librería FatFs. FatFs se puede conseguir del siguiente vínculo y es una librería de FAT para embedded systems:

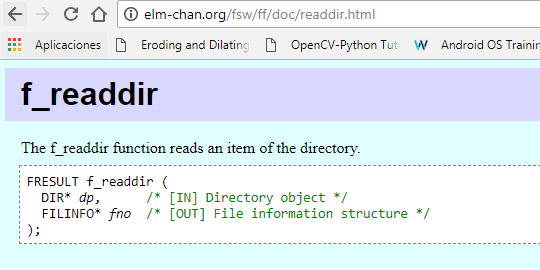


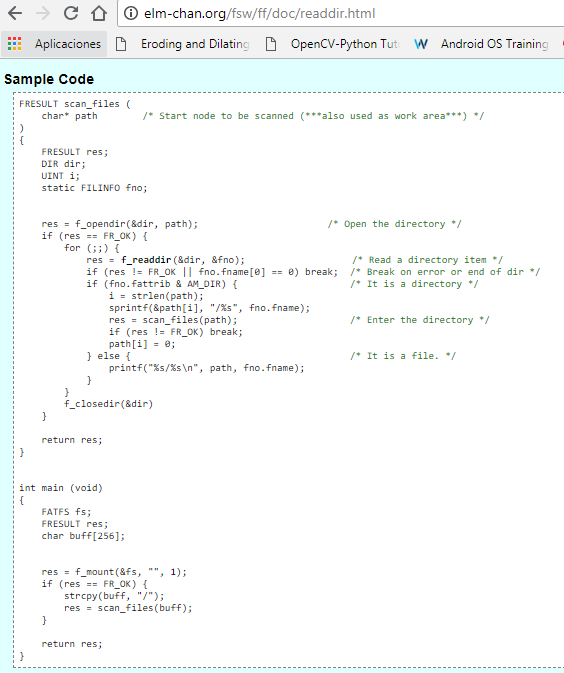
Su tarea será la siguiente…

24.1 - Crear una carpeta dentro de la microSD llamada “images” utilizando Windows (no programando sistemas embebidos).

24.2 – Arrojar varias imágenes con un nombre corto

24.3 – Utilizar la STM32 y la siguiente función (ver imagen inferior) para escanear la carpeta y mostrar el nombre de todos los archivos contenidos en esta carpeta.





24.4 – Puede utilizar el puerto serial para enviar el nombre o una variable interna que almacene el nombre y extensión del archivo (no abrir el archivo en el sistema embebido, solo deseo el nombre).