

Técnicas Digitales II

Informe Técnico

Año 2021

Integrantes:

- Barreras Fauvety, Guillermo.

- Lopez, Luciano.

- Saldivia, Luciano.

- Stahl, Francisco.

Índice:

1. Descripción General.

1.1 Introducción.

1.2 Interfaz con el usuario.

1.3 Operación.

2. Hardware.

2.1 Diagrama en bloques.

2.2 Circuito esquemático.

2.3 Circuito Board.

2.4 Descripción del circuito.

2.5 Circuito impreso.

3. Software.

3.1 Entorno de desarrollo.

3.2 Sistema operativo.

3.3 Bibliotecas.

3.4 Programa principal.

3.5 Máquina de estados ¿?????

3.5 Rutinas de interrupción.

3.6 Rutinas generales.

4. Referencias.

**1. Descripción General**

1.1 Introducción

El proyecto trata sobre un emulador de chip 8. Es capaz de ejecutar ROMs desde una SD y ejecutarlas. Consta básicamente de dos dispositivos de entrada, una tarjeta SD por donde se cargarán las ROMs y un teclado matricial para ingresar los comandos por teclado, por último, un dispositivo de salida que será un display grafico monocromático de 128\*64 pixeles.

Se emula el sistema completo, incluyendo stack pointer, registros de propósito general, memoria de video, memoria de teclado y set de instrucciones completo.

1.2 Interfaz con el usuario

El usuario tendrá como método de ingreso de información el teclado matricial, la bluepill será la encargada de interpretar qué teclas fueron presionadas y actuar en consecuencia. La presentación al usuario de la información procesada será a través del display hallado en la parte superior del dispositivo.

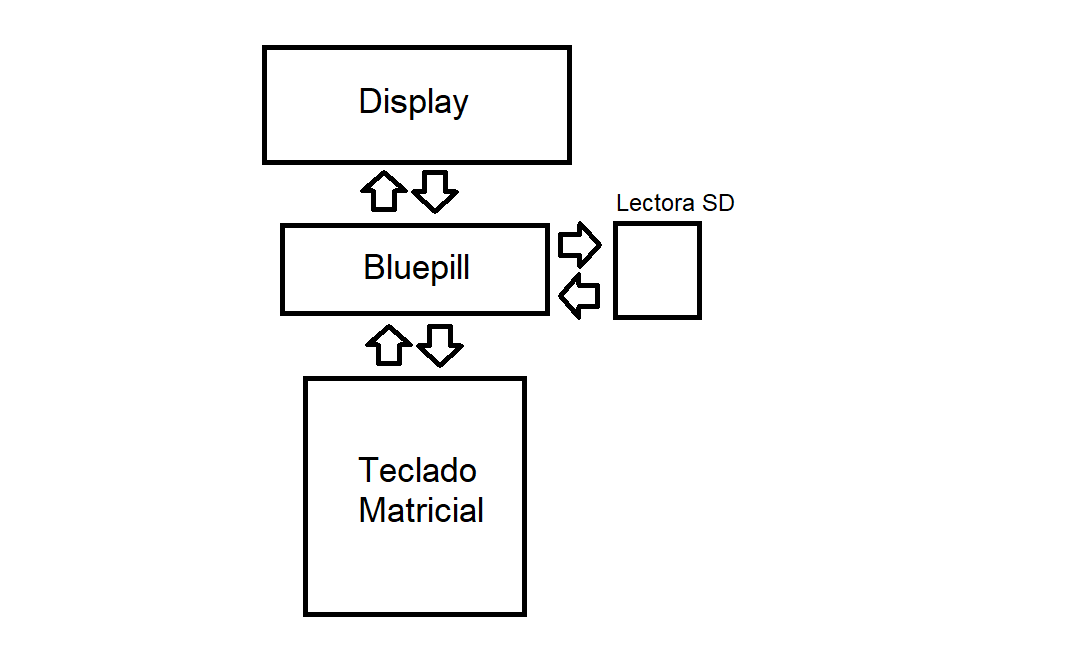
Además de las mencionadas interfaces, se encuentran en la parte inferior un switch de ON/OFF y un botón de reset.

1.3 Operación

La operación es sencilla, la interfaz de usuario también. A penas el equipo es alimentado, el directorio aparece en la pantalla. Con la tecla ***2***, el usuario recorrerá el directorio raíz y, por último, si quiere elegir una ROM simplemente debe oprimir el botón ***5*** y esperar a que la misma cargue. El dispositivo cuenta con un teclado matricial de 16 teclas (4x4) completamente funcionales mapeadas según la implementación original de chip 8. Las ROMs utilizan las teclas ***2*** (mover arriba), ***4*** (mover izquierda), ***8*** (mover abajo) y ***6*** (mover derecha) al igual que las flechas del teclado de la PC para jugar un juego).

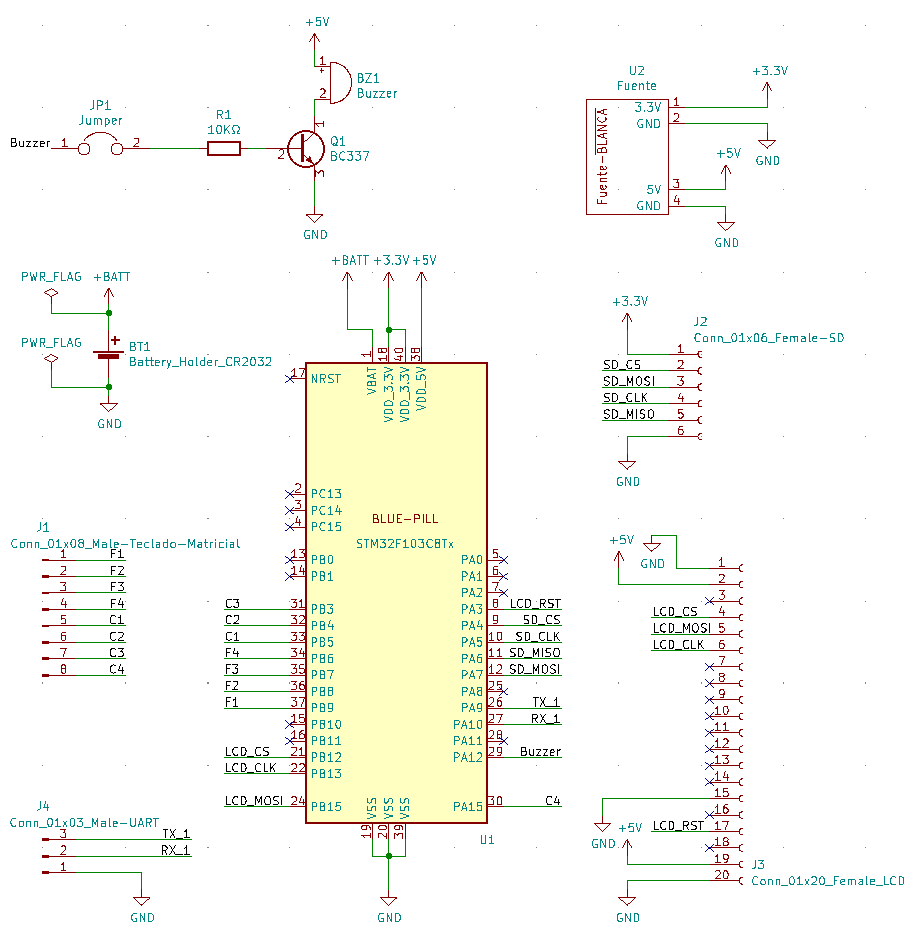
**2. Hardware**

2.1 Diagrama en bloques



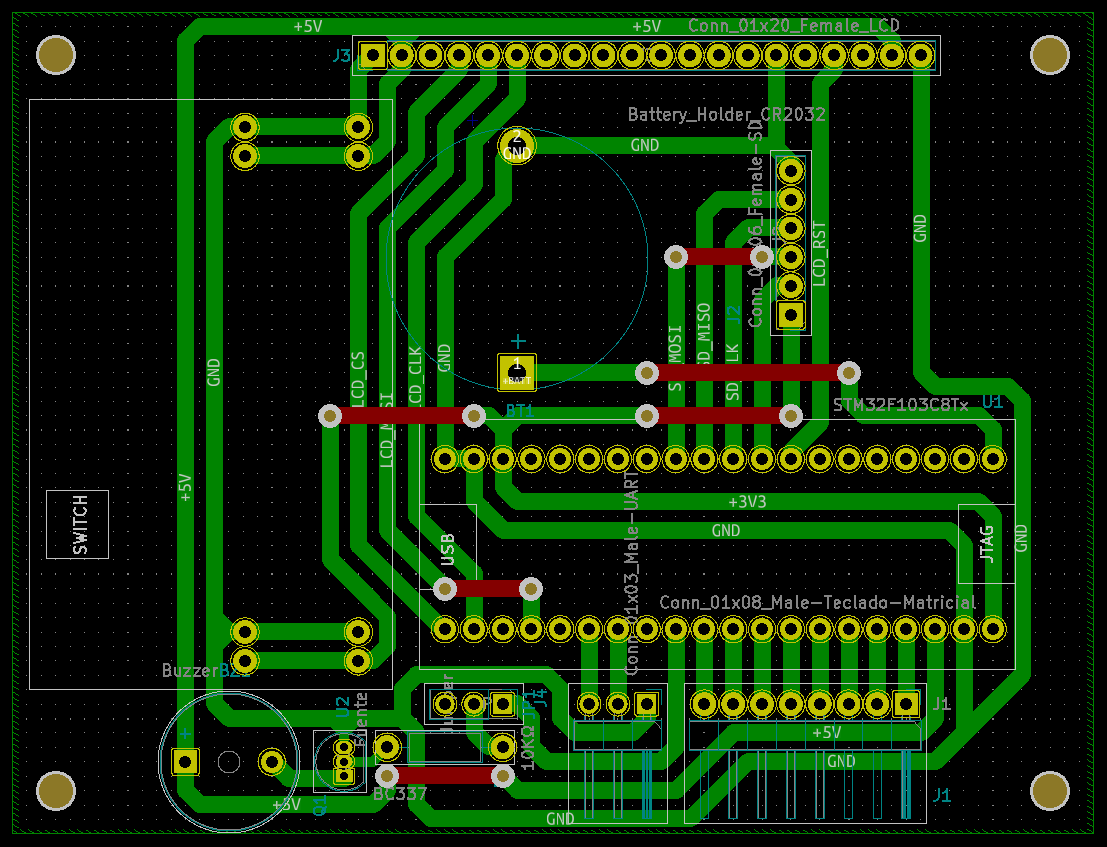
Tanto el display como la tarjeta SD utilizan el protocolo de comunicación SPI. El teclado matricial es controlado por una secuencia de barrido utilizando pines de propósito general.

2.2 Circuito esquemático

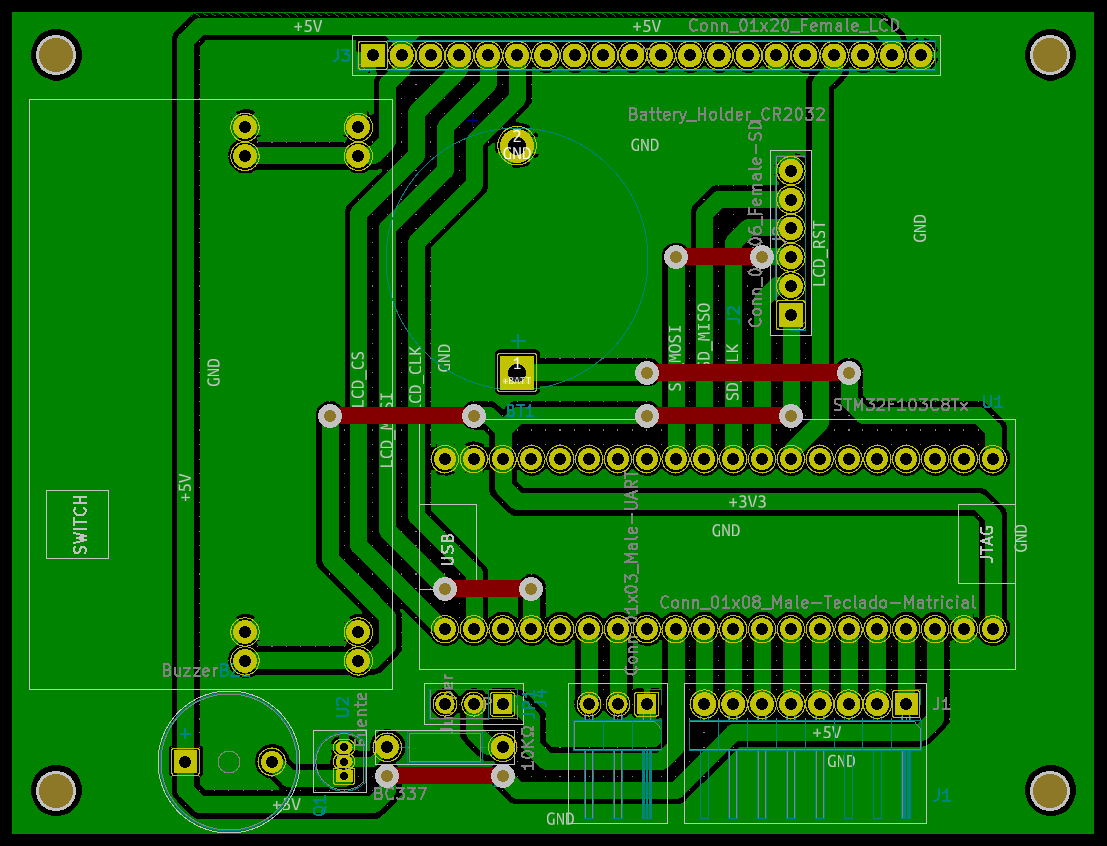


2.3 Circuito board

Sin plano de masa:



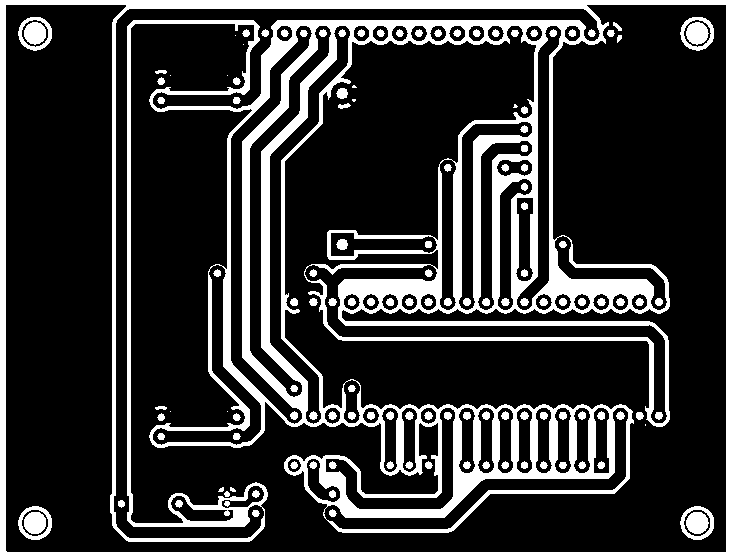
Con plano de masa:



2.4 Descripción del circuito

El circuito consta de partes:

1. BLUEPILL: Microcontrolador principal.
2. Modulo SD: Placa secundaria conectada por una tira de pines a la placa principal.
3. PILA para RTC: Se utiliza para obtener un correcto funcionamiento de la librería FATFS al momento de escribir archivos para que luego tengan un correcto timestamp.
4. Conector teclado matricial y display: tira de pines hembra.
5. Regulador de tensión: Es la encargada de suministrar 5V y 3.3V para todos los componentes del proyecto.
6. Buzzer: Accionado a través de un transistor funcionando en conmutación, el mismo está configurado como emisor común. Se le colocó un jumper en serie con la resistencia de base para tener un mejor control del mismo y evitar sonidos molestos al momento de realizar pruebas.

2.5 Circuito impreso

**3. Software**

3.1 Entorno de desarrollo: Se utilizo el IDE provisto por la catedra STM32Cube IDE. Para configurar el hardware se utilizó la herramienta de configuración de hardware para setear la mayoría de los periféricos.

3.2 Sistema operativo: Hay 3 tareas, de las cuales ***execute\_instruction*** (wcet = 27 ms), es la que más carga impone al procesamiento del sistema, dado que, entre otras cosas, actualiza el display. Luego las otras tareas son ***actualizar\_buzzer*** (wcet = 2 us) y ***process\_user\_input*** (wcet = 50 us).

Para hacer estas pruebas, se copiaron las tareas y se las adaptaron para ser incluidas en un scheduler, logrando así implementar la máquina de estado en un sistema TDS. Las primeras pruebas fueron hechas con un tick de sistema (o período secundario) de 10 ms, y era perceptible, en ciertos momentos, una falta de respuesta por parte del teclado (es decir que no se ejecutaba correctamente o a tiempo la tarea ***process\_user\_input***), al tomar las entradas, en ciertas ocasiones se congelaba el display por un instante y luego continuaba, lo que mostraba que estábamos sobrecargando al scheduler con los tiempos de las tareas.

Repensando el análisis y las pruebas hechas, y conociendo el código, fue fácil darse cuenta de que el problema estaba en la variabilidad de la tarea ***execute\_instruction***, que, si bien en el peor caso tomaba unos 27 ms, la mayoría de las veces que esta tarea era ejecutada, el tiempo que tomaba era mucho menor, dado que solamente algunas veces la instrucción a ejecutar por el chip8 emulado era la de actualizar el display, y la mayoría de las veces eran instrucciones muchísimo más simples. Al haber medido esto, se observó que el tiempo de ejecución en cada tick (de los que vimos simulando) era de 1ms, 400 us, y lo más frecuente, menos de 20 us, pero la menos frecuente y más relevante era al actualizar el display. Por lo que se tenían dos soluciones posibles para lograr que este sistema funcione como TDS.

Una de las soluciones, sería modificar el programa, fraccionando el código de actualización del display en varias partes, lo suficientemente pequeñas para no sobrecargar la CPU en el tick de sistema, y que modificadas para puedan ser ejecutadas otras tareas entre dichas fracciones. Esto no sólo implicaría muchísimo cambio en el código, que a su vez complicaría al mismo, sino también implicaría alejarnos del chip8. Dado que el mismo no funcionaba como TDS, y dado que la intención del proyecto es emularlo tan cercano como sea posible, esta solución fue descartada.

La otra solución, era ampliar el tiempo de tick del sistema, dado que para 10 ms sobrecargábamos al procesador. Haciendo las cuentas en un Excel, se vio que el CPU estaba con una carga de 270%. Se agrandó el tick de sistema, usando un nuevo tick de 30 ms, para dar un margen sobre esos 27 ms de wcet medidos de la tarea execute\_instruction. El problema era que al correr el programa con este nuevo tick, se trababa más seguido, las entradas se leían muy pocas veces, y cuando se leían, el programa se congelaba, tomando entre uno y dos segundos para continuar.

Concluimos entonces que el sistema no se puede planificar como TDS, a menos que se modifique como fue explicado, dado que por debajo de los 27 ms de tick de sistema, el sistema estaría sobrecargado, y por arriba de los 10 ms, el sistema tiene serios problemas para responder correctamente, dado que el mismo no está preparado para ser corrido como TDS.

3.3 Bibliotecas:

Se utilizaron cuatro bibliotecas:

* FATFS: Encarga del manejo de archivos.
* ST7920\_SERIAL: Biblioteca adaptada a las necesidades del proyecto, encargada de la comunicación del display gráfico.
* TECLADO: Encargada de realizar el algoritmo de barrido, debounce y llenar un buffer con el valor de la tecla presionada.
* HAL: Encargada de manejar todos los periféricos del microcontrolador.

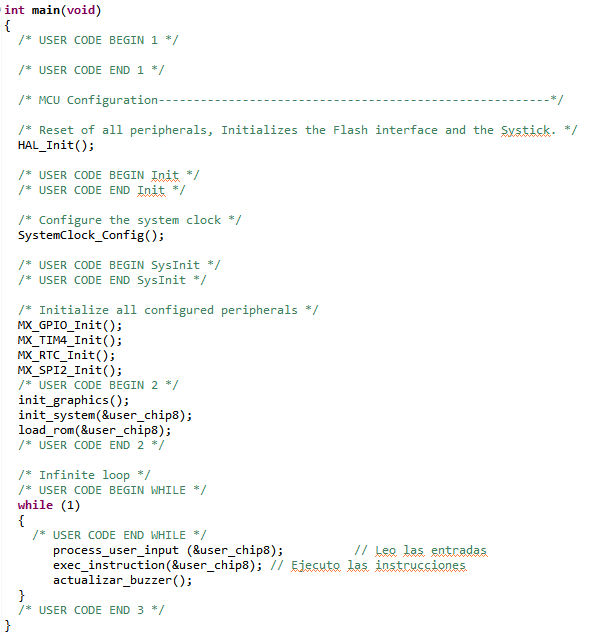
3.4 Programa principal:

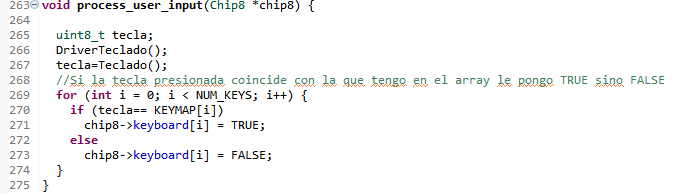
El programa principal cuenta con 3 funciones de inicialización y carga de ROM en la memoria RAM. La función ***init\_graphics(),*** es la encarga de inicializar el display gráfico***, init\_system()***, es la encargada de poner a 0 todos los registros, memorias, teclado, STACK, TIMER de sonido y pantalla por último **load\_rom()**, es la encargada de armar el menú mostrando el directorio en pantalla y seleccionar la ROM leída desde la tarjeta SD y cargarla en RAM.

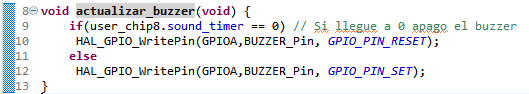
Una vez pasada la inicialización, entramos en el ***while(1)***, que solamente tiene 3 funciones:

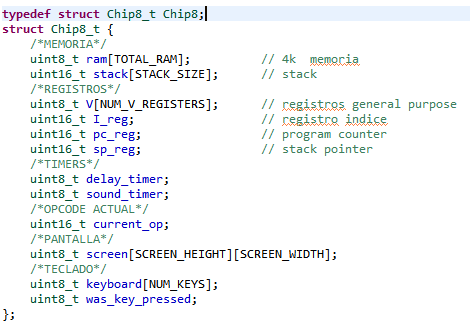
* ***process\_user\_input()***: Lee el teclado matricial y si una tecla es presionada la pone en estado ***TRUE***  sino en estado ***FALSE.***
* ***exec\_instruction()***: Se encarga de hacer las operaciones del core, es decir, fectch, decode y execute. La misma cuenta con dos máquinas de estado que serán descriptas más adelante.
* ***actualizar\_buzzer():*** Encargada de leer si el flag para activar o desactivar el buzzer está en ‘1’ o ‘0’. Prendiéndolo o apagándolo respectivamente.

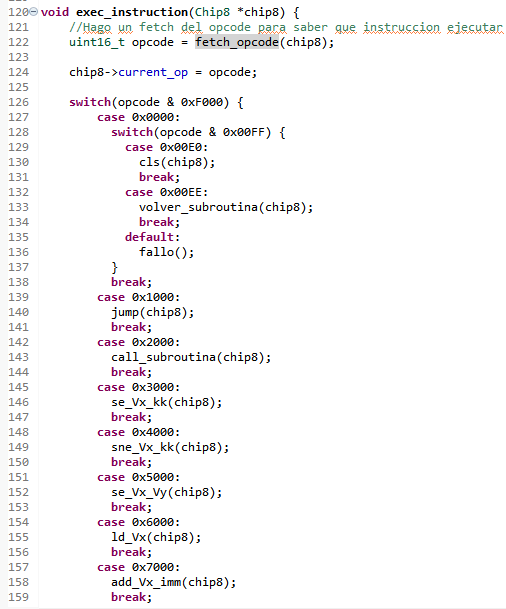
La mayoría de las funciones tienen como parámetro la dirección de una estructura tipo Chip8 llamada user\_chip8. La misma contiene todos los registros y flags de sistema necesarios para el funcionamiento del core.

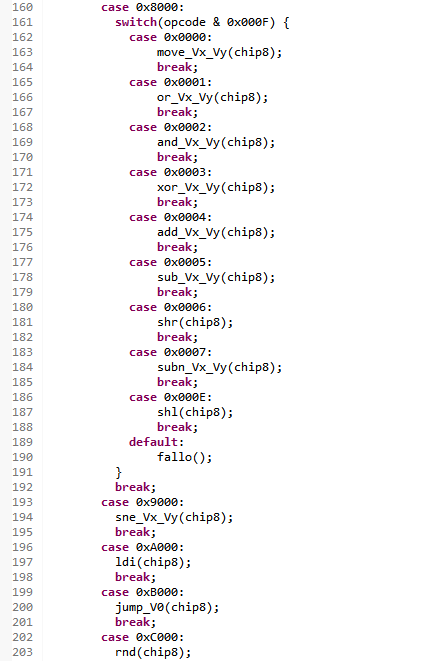


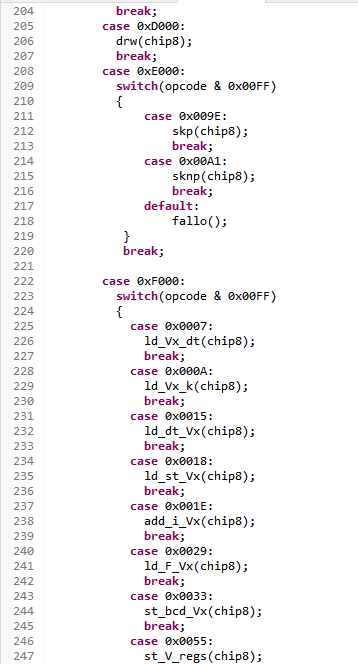
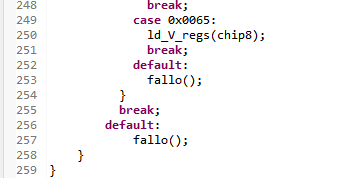






3.5 Máquina de estados:



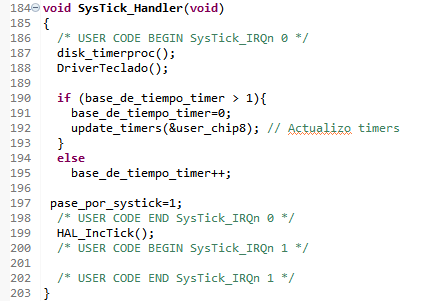
La función ***exec\_instruction()***, posee un switch a modo de máquina de encargado de decodificar el opcode actual. Los códigos de operación pueden ser encontrados fácilmente .



La función ***fetch\_opcode()***, es la encargada de buscar en RAM, (cargada previamente desde la SD) la parte alta y baja de la instrucción, componerla en una sola variable que luego será decodificada. Chip 8 utiliza instrucciones de 16 bits.

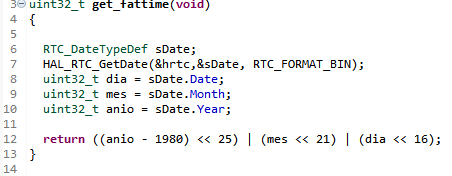
El switch principal es el encargado de recuperar los 4 bits más altos de la variable de 16 bits, para identificar el tipo de instrucción, luego según el tipo de instrucción se desglosa en distintos swichts tomando los 8 o 4 bits más pequeños decodificando por completo la instrucción.

3.5 Rutinas de interrupción:

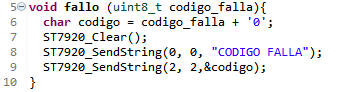


La única interrupción que es utilizada es la de Systick, la cual es llamada cada 1ms. En la misma se tienen coladas las interrupciones del timer de sonido y delay del chip8, lectura del teclado y rutinas de funcionamiento interno de la biblioteca FATFS para la lectura de la tarjeta SD.

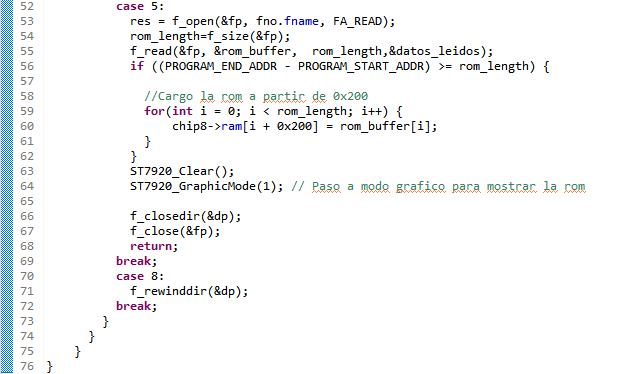
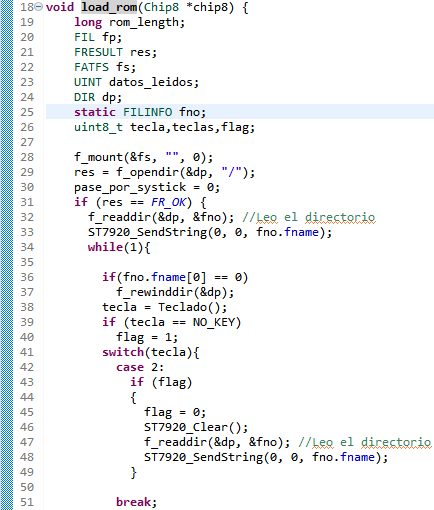
3.6 Rutinas generales



La rutina ***get\_fattime()***, esta implementada mediante un RTC alimentado a través de una pila CR2032 colocada en el PCB. De este modo cada vez que se escriba un archivo en la Bluepill el timestamp será el correcto.



En caso de que exista algún opcode no encontrado, automáticamente se imprimirá un código de error en la pantalla mediante la función ***fallo()*** a modo de alerta para el usuario. Esto se puede deber a un error en la ROM o en la SD.

****

La función ***load\_rom()*** como ya fue explicado anteriormente, es la encargada de leer el directorio y cargar la ROM en RAM. Como se puede ver en el código, utilizamos la biblioteca FATFS para hacer el manejo de archivos. Con el botón ***2*** incrementamos el índice del directorio y con el botón ***5*** cargamos la ROM.

**4. Referencias**

* <https://es.wikipedia.org/wiki/CHIP-8>.
* <http://devernay.free.fr/hacks/chip8/C8TECH10.HTM>.
* <https://multigesture.net/articles/how-to-write-an-emulator-chip-8-interpreter/>.
* <https://en.bmstu.wiki/index.php?title=CHIP-8_(programming_language)&mobileaction=toggle_view_mobile>.
* <https://es.wikipedia.org/wiki/Sprite_(videojuegos)>.
* <https://controllerstech.com/glcd-128x64-st7920-interfacing-with-stm32/>.
* <http://elm-chan.org/fsw/ff/doc/open.html>.
* <http://elm-chan.org/fsw/ff/doc/readdir.html>.
* <https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal-and-ll-drivers-stmicroelectronics.pdf>.
* <https://tobiasvl.github.io/blog/write-a-chip-8-emulator>.
* <https://chip-8.github.io/links/>.
* <https://github.com/corax89/chip8-test-rom>.
* <https://www.zophar.net/pdroms/chip8/chip-8-games-pack.html>.
* Diapositivas de clase Informática 2.
* Diapositivas de clase Técnicas Digitales 2.