|  |
| --- |
| Batería para máquina de tatuar: Rover2k |
| Documento de Software |
| MUSOTOKU |

|  |
| --- |
| Javier Rodriguez  30-3-2022 |

Índice

[1 Introducción 2](#_Toc100313107)

[1.1 Descripción Del Sistema. 2](#_Toc100313108)

[1.2 Entorno de Programación. 2](#_Toc100313109)

[1.2.1 IDEs 2](#_Toc100313110)

[1.2.2 Proceso de compilación. 2](#_Toc100313111)

[1.2.3 Proceso de programación de la CPU. 4](#_Toc100313112)

[2 Código Principal 10](#_Toc100313113)

[2.1 Estructura General. 10](#_Toc100313114)

[2.2 Descripción De Los Distintos Bloques. 11](#_Toc100313115)

[2.2.1 Inicialización 11](#_Toc100313116)

[2.2.2 Estado “START-UP” 13](#_Toc100313117)

[2.2.3 Estado “WORK” (RUN / STOP): 14](#_Toc100313118)

[2.2.4 Estado “SLEEP”. 19](#_Toc100313119)

[2.2.5 Estado “ERROR”. 21](#_Toc100313120)

[2.2.6 Estado “USB”. 21](#_Toc100313121)

[2.2.7 Gestor de Cambios de Estado. 23](#_Toc100313122)

[2.2.8 Rutinas Cíclicas. 25](#_Toc100313123)

[2.3 ProFiling Temporal 26](#_Toc100313124)

[3 Librerías 28](#_Toc100313125)

[3.1 Librerías Propias 28](#_Toc100313126)

[3.1.1 BOTONERA 28](#_Toc100313127)

[3.1.2 PANTALLA LED 29](#_Toc100313128)

[3.1.3 BARRA DE POTENCIA 31](#_Toc100313129)

[3.1.4 BUZZER 32](#_Toc100313130)

[3.1.5 DCDC 34](#_Toc100313131)

[3.1.6 SENSADO 36](#_Toc100313132)

[3.1.7 LOGGING 38](#_Toc100313133)

[3.2 Librerias Ajenas 40](#_Toc100313134)

# Introducción

## Descripción Del Sistema.

La batería Rover 2k es un sistema de alimentación inalámbrico para máquinas de tatuar.

En el siguiente diagrama se representan todos los componentes que conforman la batería:

(Diagrama)

## Entorno de Programación.

### IDEs

Arduino 1.8.13.([Descarga](https://downloads.arduino.cc/arduino-1.8.13-windows.exe))

### Proceso de compilación.

* Configuración de la carpeta del Proyecto.

Arduino para compilar un archivo necesita una carpeta raíz de la que extraer o consultar la presencia de las librerías utilizadas en el Sketch a compilar.

Por defecto Arduino utiliza la dirección “*C:\Users\[User]\Documents\Arduino\libraries”(En Windows)* como carpeta de proyecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteSin embargo, esta dirección se puede cambiar en la pestaña de Archivo -> Preferencias:

De esta manera, podemos configurar la carpeta raíz para compilar nuestro proyecto con las librerías específicas que necesitamos.

En el caso del proyecto del “Rover 2k”, la ubicación del proyecto esta direccionada desde la ubicación del guardado de:

* La carpeta del Team Dropbox.

*“…\Team Dropbox\JRODRIGUEZ\Workspace\2-PROYECTOS\Bitbucket\03-Battery Pack\bat\_sw”*

* Carpeta del repositorio Git del proyecto:

*“…\bat\_sw”*

* Configuracion de los ficheros de compilación.

Arduino es un IDE que esta pensado de manera cerrada. Codificar, Compilar y Programar, todo sin salir del IDE. Sin embargo, las características del proyecto no se ajustan a esta funcionalidad.

Es necesario tener un control de los archivos de compilación. Por defecto, los archivos generados de una compilación son almacenados en la sección de archivos temporales del sistema operativo. De tal manera que cuando se cierra el IDE, esos archivos se pierden.

Para cambiar la carpeta para los archivos obtenidos de la compilación del IDE de Arduino accedemos nuevamente a la pestaña Archivo -> Preferencias:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Accedemos a la ubicación del archivo *preferences.txt, ubicado en la parte inferior de la ventana.*

Una vez tengamos la ubicación abierta:

1. Cerramos el IDE.
2. Abrimos el archivo *preferencies.txt*
3. Insertamos el siguiente texto:

*build.path=./.build*

de esta manera, cuando compilemos por primera vez, se nos genera una carpeta llamada *.build* en la carpeta del fichero compilado, con todos los archivos obtenidos de la compilación.

NOTA: Si se quiere se puede sustituir *./.build* por una dirección completa a otra localización.

### Proceso de programación de la CPU.

Para la programación del SAMD21, se utilizará la plataforma de programación proporcionada por Segger J-LINK ([Descarga](https://www.segger.com/downloads/jlink/#ESforRISCV)). En concreto, se usa el programa J-Flash v7.20.

* Configuración del proyecto dentro de J-FLASH.

1. Iniciamos el programa J-Flash V7.20.
2. Interfaz de usuario gráfica, Aplicación

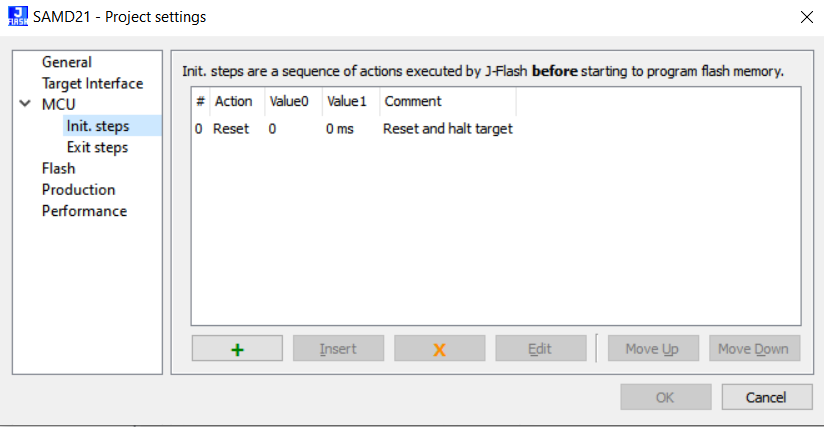
   Descripción generada automáticamenteCreamos nuevo proyecto.
3. Interfaz de usuario gráfica

   Descripción generada automáticamenteTabla

   Descripción generada automáticamenteSeleccionamos el dispositivo SAMD21G18.
4. Una vez dentro del programa vamos a configurar el proyecto en Options ->Project settings… y completamos la configuracion como en las imágenes:
   * Interfaz de usuario gráfica, Aplicación

     Descripción generada automáticamenteGeneral:
   * Interfaz de usuario gráfica

     Descripción generada automáticamenteTarget Interface:
   * Interfaz de usuario gráfica, Aplicación

     Descripción generada automáticamenteMCU:
   * Init Steps:
   * Interfaz de usuario gráfica, Texto

     Descripción generada automáticamenteExit steps:
   * Interfaz de usuario gráfica, Aplicación

     Descripción generada automáticamenteFlash:
   * Interfaz de usuario gráfica, Aplicación

     Descripción generada automáticamenteProduction:
   * Interfaz de usuario gráfica, Aplicación, Word

     Descripción generada automáticamentePerformance:

* Programación del sistema:

Interfaz de usuario gráfica, Texto, Aplicación, Tabla

Descripción generada automáticamenteClicamos en File->Open Data File y seleccionamos el fichero .hex del programa compilado.

Una vez tenemos cargado el archivo . hex procedemos a la programación:

1. Conectar el dispositivo J-Link al ordenador y al puerto de programación del sistema.
2. Asegurarse que la placa tenga alimentación externa ya que el J-Link no la proporciona.
3. Target->Connect.
   1. Texto

      Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

      Descripción generada automáticamenteEn caso de que alguna de las 2 cosas previas falle:
   2. Texto

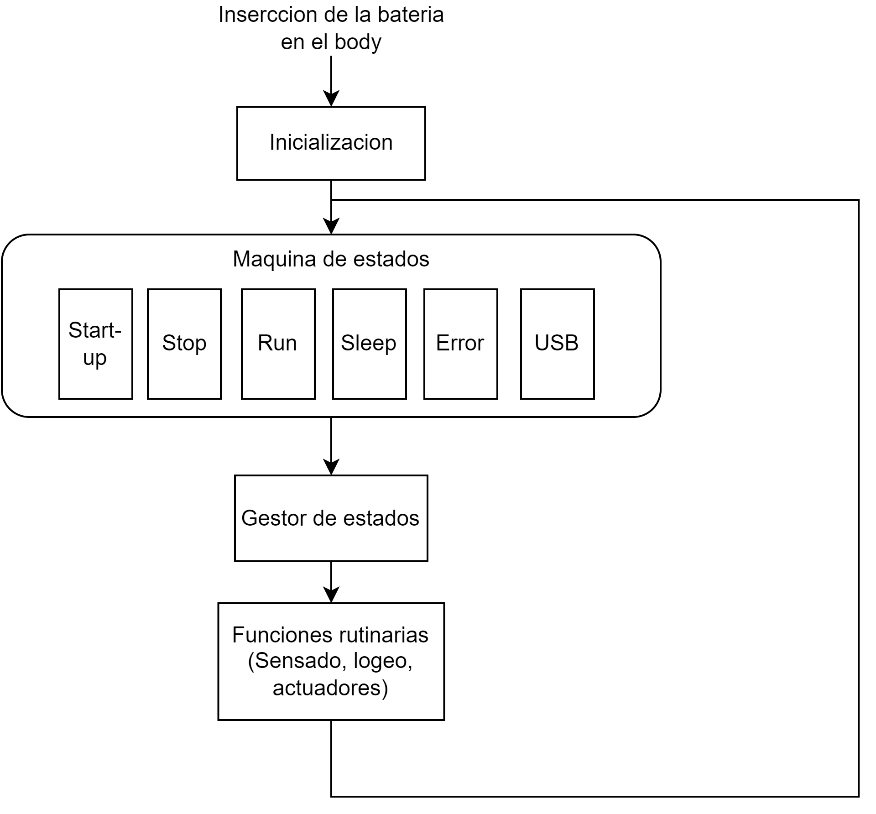
      Descripción generada automáticamenteEn caso de que no haya ningún error:
4. Con la conexión establecida, Target ->Prodution Programming (F7)

Y tras completarse correctamente tendríamos programado el sistema.

# Código Principal

## Estructura General.

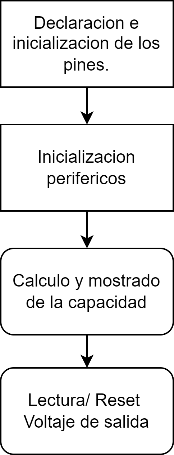
El código consta principalmente de una maquina de estados, un gestor de estados y funciones rutinarias como sensado o actuación.

 El siguiente diagrama ejemplifica y resume la distribución del código:

## Descripción De Los Distintos Bloques.

### Inicialización

La inicialización o set-up, se lleva a cabo cada vez que se inserta la batería y la CPU recibe alimentación, iniciándose. Esta parte del código, previa al bucle de control, se ejecuta una sola vez.

El siguiente diagrama muestra los pasos seguidos en la secuencia de encendido:

* Declaración e inicialización de los pines:

Configuración de todos los pines usados del SAMD según su uso: Entrada o salida.

Adicionalmente se inicializan aquellos pines que son de salida.

* Inicialización de los periféricos:

Algunos periféricos utilizados como el display, o el buzzer, requieren de una inicialización.

En el caso del display el controlador necesita inicializar sus registros. En el caso del buzzer se inicializan ciertas variables internas según lo almacenado en la eeprom del body.

En esta parte también es donde se realiza un volcado de los datos de las eeproms internas (tanto del body como de la batería) que sirven para situar el sistema en un punto de trabajo lo mas parecido al que se encontraba antes de su ultimo apagado.

En caso de que la lectura de las eeproms no se haya producido adecuadamente se registran dichos errores en variables en RAM. Mas adelante en el ciclo de control del programa, la próxima vez que se intente acceder a las EEPROMS para actualizarlas, en su lugar se volver a intentar la inicialización/volcado de los datos.

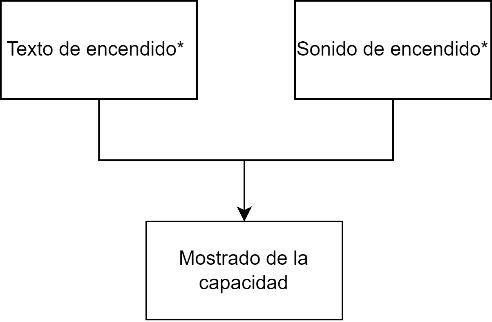
* Calculo y mostrado de la capacidad:

Debido a que esta parte de código se ejecuta tras la inserción de la batería, por decisión de diseño, se realiza una medida de la capacidad de la batería insertada y dicha capacidad se muestra por el display, sirviendo como método de consulta del porcentaje de la batería para el usuario.

### Estado “START-UP”

El estado de START-UP tiene lugar cada vez que se “encienda” la batería y es un estado transitorio, es decir, se ejecutara una vez y sirve de puente entre 2 estados permanentes (SLEEP y STOP).

En función de la configuración almacenada en el chasis, el proceso de encendido puede variar.

* (OPCIONAL) Se muestra un texto en el display.
* (OPCIONAL) Se escucha una música reproducida por el buzzer.
* Se muestra el porcentaje de capacidad restante de la batería.

\**Es necesario que la feature se encuentre activada en la configuración interna de la batería. Para activarla/desactivarla accede al modo diagnostico con la batería conectada por USB a un ordenador.*

* Texto de encendido

Al comienzo de la inicialización se muestra en el display un mensaje prefijado. Por defecto, el mensaje de encendido se encuentra desactivado.

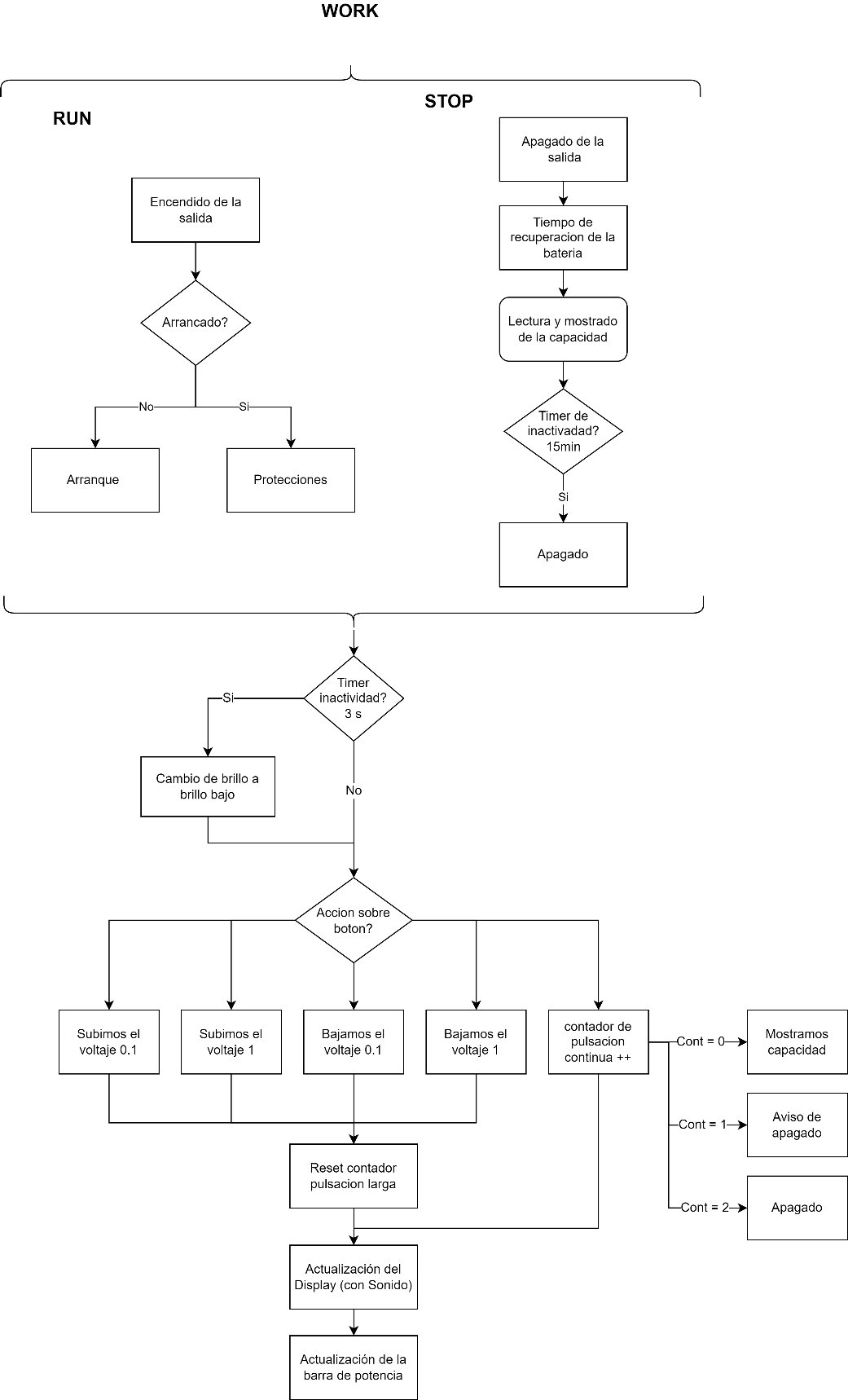
* Sonido de encendido

Al comienzo de la inicialización se reproduce en el buzzer un sonido prefijado. Por defecto, el sonido de encendido se encuentra desactivado.

* Mostrado de la capacidad

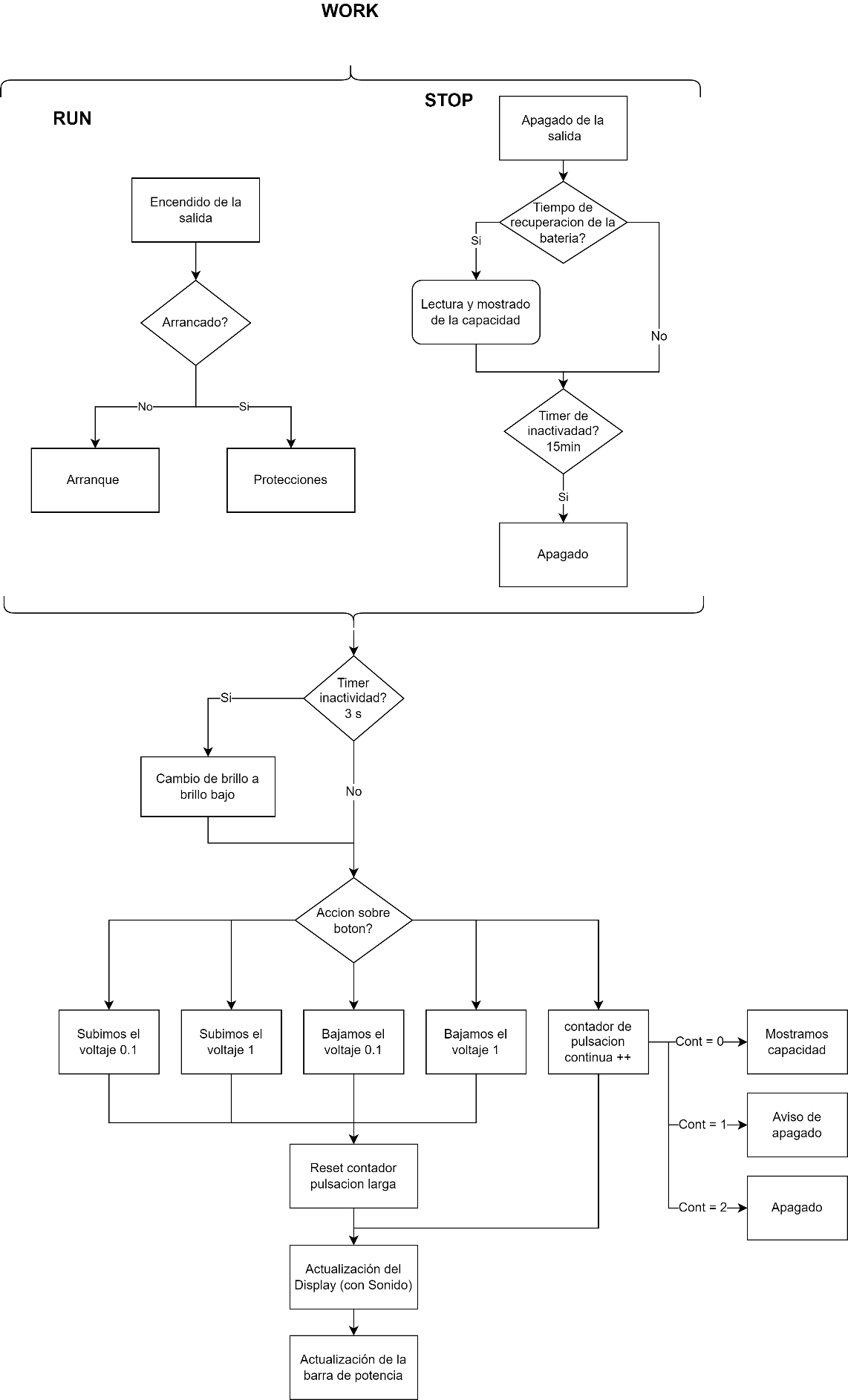
Una vez que se ha mostrado el mensaje y ha terminado el sonido. Para terminar la secuencia de encendido se muestra durante un breve tiempo la capacidad de la batería.

### Estado “WORK” (RUN / STOP):

Dentro del estado de WORK podemos diferenciar dos “sub-estados”, RUN y STOP, que a efectos prácticos se van a tratar como estados independientes. Sin embargo, ciertas comprobaciones y actuaciones se llevan a cabo independiente del sub-estado, siempre que el sistema se encuentre en WORK.

#### Estado “STOP”

El estado STOP es un estado permanente donde la alimentación de salida se encuentra desconectada.

El siguiente diagrama representa las acciones o comprobaciones que se realizan en este estado frente a distintos eventos y/o interacciones con la batería:

Por decisión de diseño, la actualización del valor de la capacidad de la batería se realiza durante el estado de STOP. Sin embargo, para realizar la medida hay tomar ciertas precauciones.

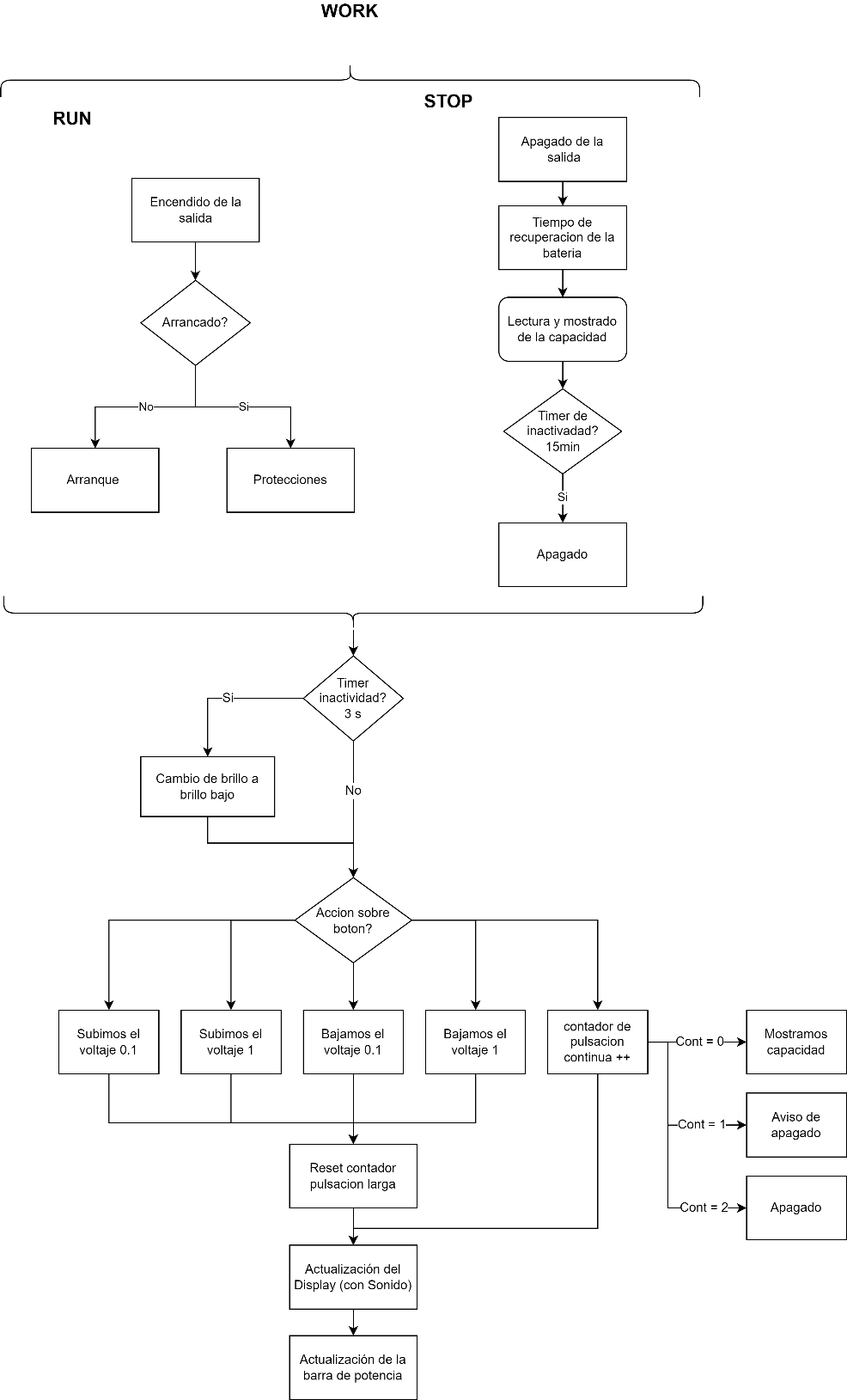
Debido a que la medida de la capacidad se realiza a través del voltaje observado en la batería, es necesario que este se encuentre en un estado libre de carga, por ello antes de leer se abre un abanico de tiempo en el que se espera que el voltaje de la batería se recupere en el caso de que el sistema venga del estado de RUN donde ante una carga como es una maquina de tatuar, la batería se arruga, disminuyendo su voltaje.

Una vez el timer de recuperación haya terminado, se puede realizar el cálculo de la capacidad.

Adicionalmente, en el estado de stop se monitorea si el sistema no recibe ninguna interacción durante un tiempo determinado (15 min aprox), en cuyo caso se considera que el sistema esta inactivo y se procede al apagado del mismo (SLEEP).

#### Estado “RUN”

El estado RUN es un estado en el que se permanece hasta que el usuario decide parar la máquina y por tanto la alimentación de salida se encuentra conectada.

El siguiente diagrama representa las acciones o comprobaciones que se realizan en este estado frente a distintos eventos y/o interacciones con la batería:

El estado RUN se corresponde con la activación de la salida, esto quiere decir que es cuando alimentamos a la maquina de tatuar conectada. Por ello hay que identificar 2 escenarios: el arrancado y el régimen permanente.

Durante el arrancado, hay que tener en cuenta que estamos alimentando máquinas de tatuar que pueden demandar potencias muy altas en las fases iniciales (hasta x10 veces el consumo estacionario), por ello, puede darse la situación que nos encontremos con picos de potencia que, al llegar a la batería, su voltaje caiga llegando, en algunos casos, a niveles peligrosos para la alimentación general del sistema, es decir que puedan:

1. Alcanzar un voltaje insuficiente para el funcionamiento de la CPU
2. Alcanzar un voltaje que dispare el protector de la batería y bloquee su salida permanentemente

Para prevenir este efecto, se ha tomado la decisión de implementar una secuencia de arranque gradual más benévola con la batería que el arranque directo.

La secuencia comienza por, independientemente del voltaje de salida objetivo, durante un tiempo (milisegundos) se pone la salida a un voltaje mínimo (4-5v). Tras ello y de manera gradual se implementa una escalera de voltajes en incrementos pequeños hasta llegar al voltaje objetivo marcado. De esta manera se pretende activar las maquinas siempre a voltajes donde no tengan picos de arrancados muy brutos que no perjudiquen a la batería y por tanto el voltaje de alimentación no caiga.

A pesar de esto, se ha comprobado que existen maquinas (como las de bobinas) que independientemente de la secuencia de arrancado, pueden provocar la caída del voltaje de la batería. Aunque el sistema no está pensado para usar con ese tipo de máquinas, es necesario que nos protejamos en esas situaciones. Para estos casos se ha introducido una interrupción externa que avisa de que el voltaje ha caído por debajo de un umbral de protección. Esta interrupción avisa del error y por tanto se puede proteger a la batería desconectando la salida en ese instante y evitando que el voltaje de la batería siga cayendo.

Una vez, la maquina se encuentre arrancada, consideramos régimen estacionario. En este régimen existen unas protecciones de corriente, voltaje de salida y potencia, que permitirán detectar cualquier mal funcionamiento y actuar en consecuencia.

Las protecciones funcionan con la librería HealthMonitor.h, dicha librería implementa un algoritmo de temporizadores de rampas de subida y bajada en función de la persistencia o no de los valores por encima del umbral fijado.

Los umbrales de dichas protecciones son:

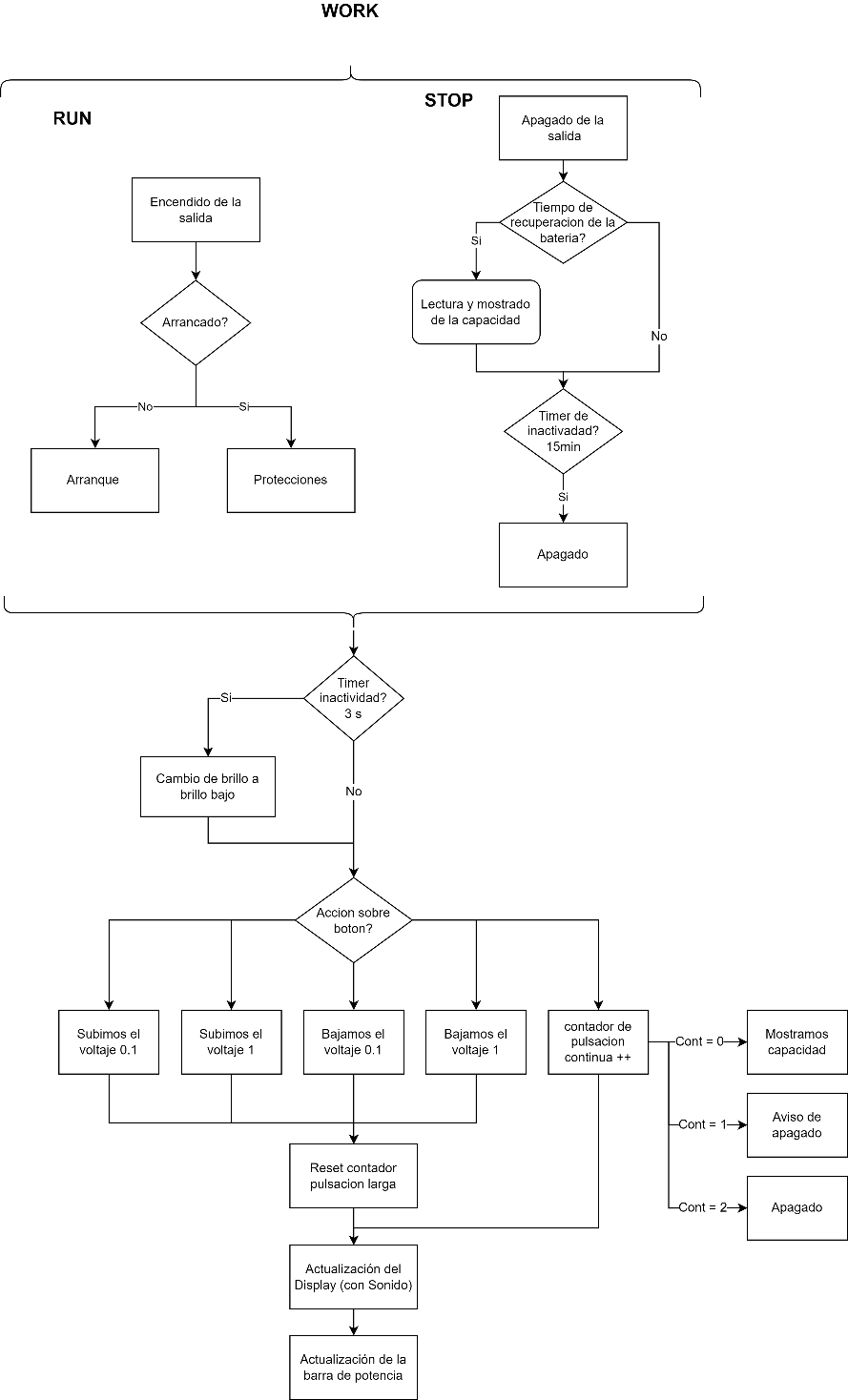
* Protección de Cortocircuito: Salta ante corrientes superiores a 450mA.

Medir corrientes de estos valores, puede deberse a fallos en el arranque la maquina y que estas se encuentren atascadas o directamente cortocircuitos en el conector de salida.

* Protección de UnderVoltage: Salta cuando el voltaje de salida medido y el teórico presentan una diferencia de -2 voltios (Vmedido <= Vteorico – 2).
* Protección de Potencia: Salta cuando la potencia de salida es superior a 4w.

Este umbral se debe a que la eficacia del DCDC cae enormemente al acercarse a 5 w de potencia de trabajo.

#### Rutinas del modo Work

Procesos que se llevan a cabo tanto el estado de Run como en el de Stop:

* Timer de Inactividad.

Al cabo de 3 segundo sin que se haya interactuado con la batería, el brillo se actualiza a un nivel más bajo para evitar el agotamiento prematuro de la batería y el deslumbramiento al usuario por intensidad de la luz.

* Botones:

Dependiendo de las acciones que se lleven acabo sobre los distintos botones del sistema se producen unos cambios u otros, si se produce:

* + Una pulsación corta sobre el Up-Button, se sube el voltaje 0.1v.
  + Una pulsación corta sobre el Down-Button, se baja el voltaje 0.1v.
  + Una pulsación larga sobre el Up-Button, se sube el voltaje 1v desde el voltaje actual (+1V).
  + Una pulsación larga sobre el Down-Button, se baja el voltaje 1v desde el voltaje actual (-1V).

Si se mantiene pulsado cualquiera de los 2 botones, se continúa aumentando/disminuyendo el voltaje de manera repetida hasta que se suelte el botón o se alcanzan los límites permitidos.

* + Una pulsación larga sobre el Power-Button, se inicia o continua la secuencia de apagado. Para pagar el sistema es necesario mantener presionado el botón durante al menos 3 pulsaciones largas. En cada pulsación se lleva a cabo una acción distinta:
    - La primera muestra la capacidad.
    - La segunda muestra un aviso de que se va a apagar la batería.
    - La tercera activa el cambio de estado para apagar el sistema.

Si no se completa de menara consecutiva las pulsaciones largas, el contador se resetea y se muestra de nuevo el voltaje actual.

* Actualización de la pantalla.

En el caso de que se haya modificado el valor del voltaje de salida, se haya producido un cambio de estado o se haya interrumpido la secuencia de apagado, la pantalla muestra el voltaje teórico en ese instante.

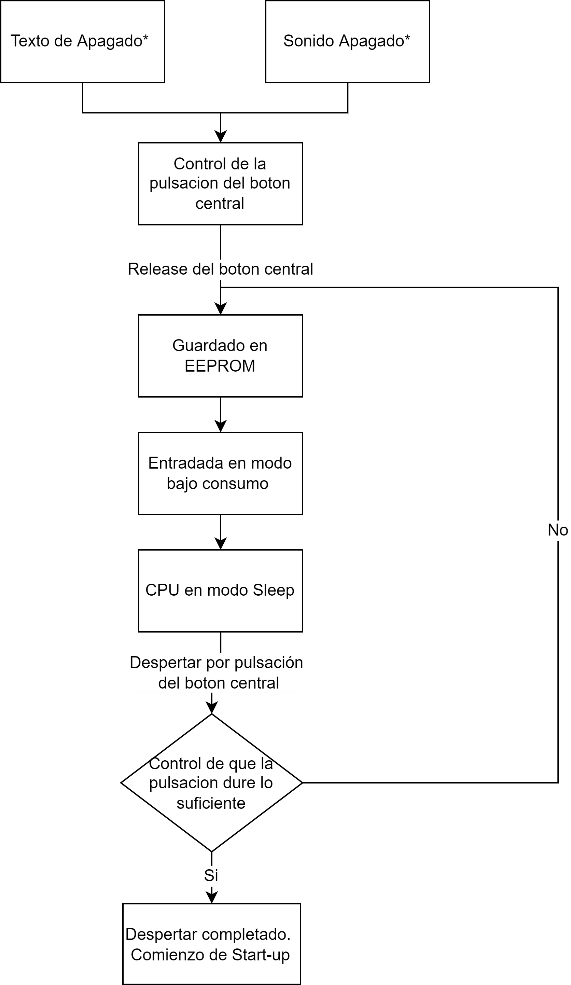
* Actualización de la barra de potencia.

Se manda actualizar la cantidad de puntos encendidos y apagados de la barra de potencia para reflejar la medida de potencia instantánea. En el momento de pasar al estado RUN siempre se activa al menos el primero de los leds para indicar que la salida está activada. El resto de los leds se encienden en función de la potencia instantánea medida (vout \* iout).

### Estado “SLEEP”.

El estado de SLEEP es un estado permanente en el que el sistema se encuentra alimentado (la batería esta insertada en el chasis, y por tanto la CPU esta funcionando) pero se considera “apagado”.

Durante este estado, el objetivo es reducir al mínimo el consumo.

El siguiente diagrama muestra los procesos de “apagado” y de “despertar” del sistema:

\**Es necesario que la feature se encuentre activada en la configuración interna de la batería. Para activarla/desactivarla accede al modo diagnostico con la batería conectada por USB a un ordenador.*

* Texto de encendido

Al principio de la secuencia de apagado se muestra en el display un mensaje prefijado. Por defecto, el mensaje de apagado se encuentra desactivado.

* Sonido de encendido

Al principio de la secuencia de apagado se reproduce en el buzzer un sonido prefijado. Por defecto, el sonido de apagado se encuentra desactivado.

* Control de la pulsación del botón central o Power Button.

Se percibió que existía la posibilidad que si se mantenía el botón apretado se podía generar un ciclo de encendido y apagado constante.

Por ello, antes de continuar con la secuencia de apagado se realiza una comprobación continua del estado del botón hasta que este se haya dejado de pulsar.

* Guardado en EEPROM.

Actualización de las memorias eeprom tanto de la batería como del Body.

* Entrada en modo Bajo consumo.

Se desactiva la alimentación alternativa que sirve de alimentación de periféricos como el display, el dcdc, el buzzer etc. De esta manera se reduce considerablemente el consumo general del sistema.

* CPU en modo Sleep.

El SAMD tiene un modo Sleep donde el consumo se reduce drásticamente. Al entrar en este modo, el ciclo de programa se congela. Es necesario setear una interrupción de Wake-Up para despertar al procesador. Una vez el procesador se despierta reanuda la ejecución del programa en la instrucción siguiente a la que le pone a dormir.

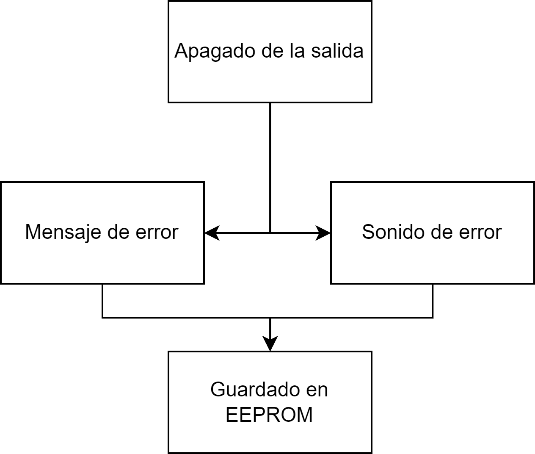
En este caso la interrupción queda seteada al pin del botón central o Power Button. En cuanto se pulsa el botón el procesador se despierta y continua con el programa.

* Control de pulsación del botón central.

Con el fin de prevenir activaciones involuntarias, una vez que el procesador se despierta, se procede a evaluar si el botón esta pulsado y si es así durante cuanto tiempo lleva pulsado. En este caso es necesario detectar una pulsación larga para que se entienda como una activación voluntaria. Si en algún momento se deja de pulsar el botón antes de cumplir el tiempo, el sistema volvería a iniciar la secuencia de dormir desde el guardado en eeprom para volver a meter al procesador en modo sleep o bajo consumo.

### Estado “ERROR”.

El estado de “ERROR” es un estado transitorio donde se llevan a cabo todos los procesos de atención a errores.

El siguiente diagrama muestra el proceso de atención a errores que se lleva a cabo:

El estado error es un estado en el que la salida queda inmediatamente deshabilitada.

Cualquiera de las razones por las que se accede a este modo tiene que ver con la salida, por lo que es importante que se corte en cuanto se detecta un error.

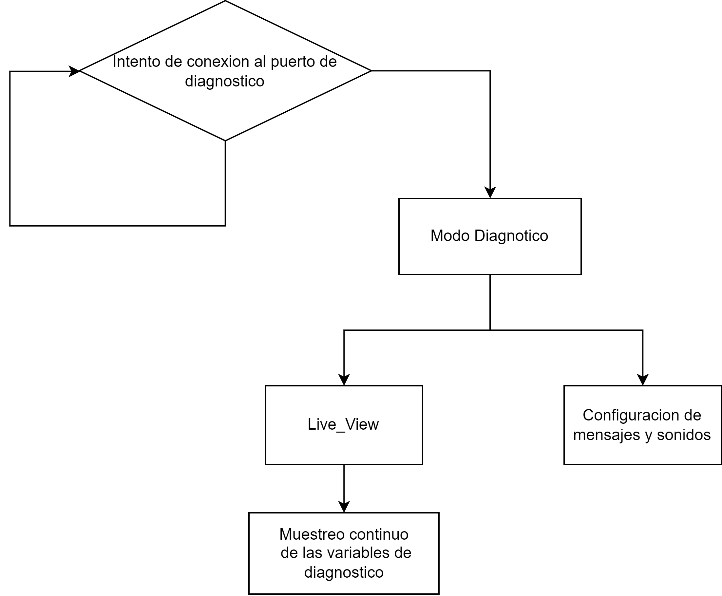
Actualmente todas las atenciones de errores pasan por mostrar un mensaje de aviso del error y de un sonido que lo acompaña.

Tras ello se realiza un guardado en eeprom para que quede registro de los problemas que ha sufrido el sistema.

### Estado “USB”.

El estado de “USB” es un estado permanente donde se ha detectado la conexión de un USB al puerto de la batería y se ha iniciado la carga de la batería mientras esta se encuentra insertada en el chasis.

Existe la posibilidad de acceder a un sistema de diagnóstico a través del puerto serie. *Esta funcionalidad vendrá condicionada a la versión HW del sistema.*

El siguiente diagrama muestra las acciones llevadas a cabo en este estado:

El modo USB es el también el modo carga, durante este estado únicamente se mostrará una imagen que indica que se ha conectado un cargador en el puerto USB-C.

En el caso que se haya conectado a un ordenador, es posible, mediante un puerto serie, entrar a un modo diagnóstico. El cual tiene 2 variantes: Live View o Configuración.

Mediante una comunicación por el puerto serie puede activarse el Live View. Con este modo activado, se procede a enviar cada segundo el valor de una serie de variables denominadas “variables de diagnóstico”. Algún ejemplo de estas variables es: Voltaje instantáneo, Corriente instantánea, Potencia instantánea o Voltaje de salida teórico.

También es posible configurar algunas de las características de la interfaz de usuario de la batería como son:

* + Activación/Desactivación del mensaje de encendido.
  + Activación/Desactivación del mensaje de apagado.
  + Activación/Desactivación del sonido de encendido.
  + Activación/Desactivación del sonido de apagado.
  + Activación/Desactivación del sonido de carga.

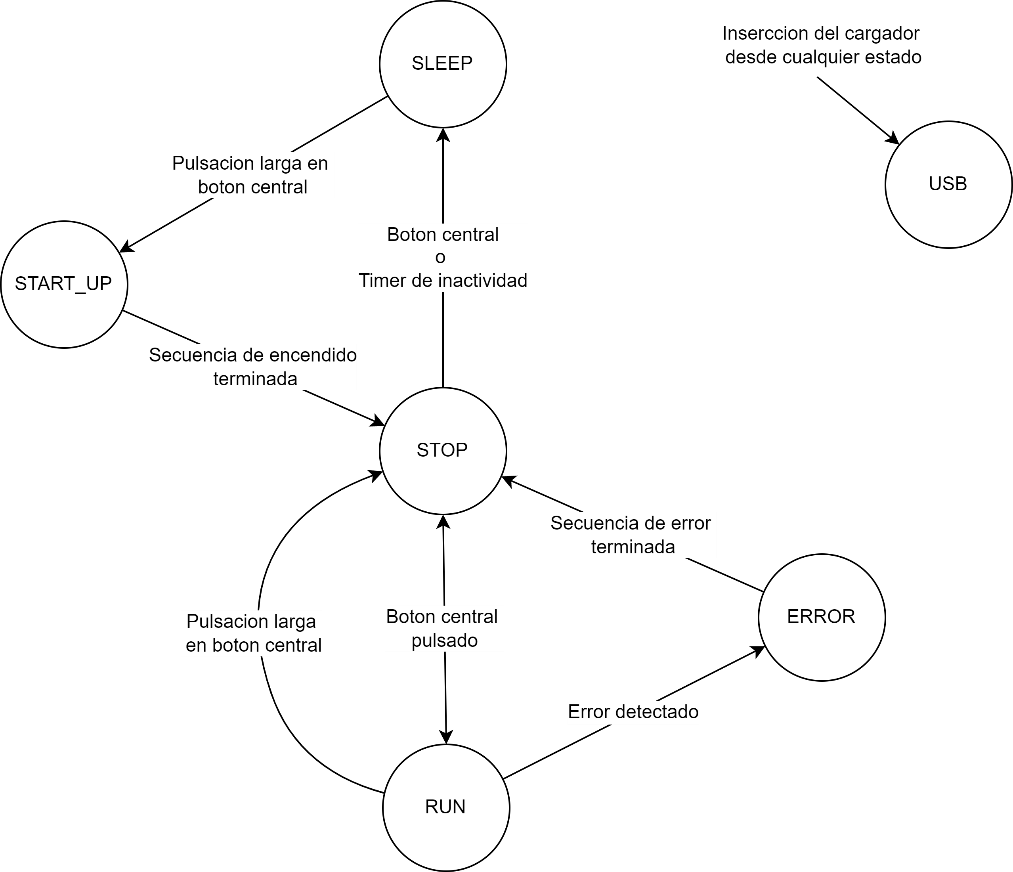
### Gestor de Cambios de Estado.

Este bloque es el encargado de detectar las condiciones de cambio entre los distintos estados y de ejecutar las acciones necesarias que tienen lugar en esos cambios.

A continuación, se enumeran las condiciones de cambio entre los distintos estados y las acciones, si las hubiera, consecuentes a dichos cambios:

|  |  |  |  |
| --- | --- | --- | --- |
| ORIGEN | DESTINO | CONDICION | ACCION |
| START\_UP | STOP | Secuencia de encendido terminada | * Inicio de timer de inactividad. * Inicio de timer de recuperación de voltaje. |
| USB | Detección de la conexión de un cargador en el puerto USB-C mediante interrupción. | * Animación de conexión USB (Imagen y sonido) |
| RUN | STOP | Pulsación del power button.  Pulsación larga del power button | * Inicio del timer de inactividad. * Inicio del timer de recuperación. * Sonido de “desactivación de la salida”. |
| USB | Detección de la conexión de un cargador en el puerto USB-C mediante interrupción. | * Animación de conexión USB (Imagen y sonido) |
| ERROR | Activación de Flag de Error. | * Desactivación de la salida. * Sonido de error. * Texto del tipo de error en el Display. * Apagado de la barra de potencia. |
| STOP | RUN | Pulsación del power button. | * Sonido de “Activación de la salida”. |
| USB | Detección de la conexión de un cargador en el puerto USB-C mediante interrupción. | * Animación de conexión USB (Imagen y sonido) |
| SLEEP | Secuencia de apagado finalizada. (pulsación continuada del Power button) | - |

|  |  |  |  |
| --- | --- | --- | --- |
| Sleep | START\_UP | Despertar completado (pulsación continuada del Power button). | * Inicialización de la pantalla. (Pantalla en negro) * Cálculo de la capacidad. |
| USB | Detección de la conexión de un cargador en el puerto USB-C mediante interrupción. | * Animación de conexión USB (Imagen y sonido) |
| ERROR | STOP | Secuencia de atención a errores terminada. | * Inicio del timer de inactividad. * Inicio del timer de recuperación. |
| USB | SLEEP | Detección de la desconexión del cargador en el puerto USB-C. | * Animación de desconexión USB (Imagen y sonido) * Desactivación de la Salida. |

El siguiente diagrama es un resumen de todos los cambios de estado enumerados:

### Rutinas Cíclicas.

En este bloque se agrupan aquellas funcionalidades que se ejecutan en cada ciclo de control y que son inherentes al estado en el que se encuentre el sistema.

* Sensado: Toma de medidas analógicas de las variables de: Tensión de salida, corriente de Salida y calculo de la Potencia de salida.
* Control Boost/No Boost (¿): En el caso que se detecte corriente a la salida se activa el modo boost. En caso negativo, se pasa a modo no-boost.
* Control de la salida: En función de dos señales (software o usuario) se activa o desactiva el voltaje de salida.
* Logging: Guardado de las variables dinámicas de las variables de diagnóstico y comprobación del timer que actualiza la eeprom.
* Lectura de los botones: comprobación de si ha ocurrido algún evento en los botones del sistema
* Comprobación del estado de la conexión USB: En caso de interrupción por conexión o desconexión, se actualiza el estado del usb.
* Control del periodo del ciclo de programa: Control del periodo mínimo del ciclo de programa, 5 milisegundos.

## ProFiling Temporal

|  |  |
| --- | --- |
| Inicialización |  |
| * Pines | 73us |
| * Pantalla | 60ms |
| * Buzzer | 80us |
| * Capacidad | 20,94 ms |
| * Volcado eeprom | 8,82 ms |
| Estado START-UP |  |
| * Capacidad | 1.019s |
| * Texto encendido |  |
| * Sonido encendido |  |
| Estado WORK |  |
| * Chequeo Capacidad | 124us |
| * Chequeo Botones | 3us |
| * Secuencia de apagado |  |
| * Fase 1 | 1s |
| * Fase 2 | 1s |
| * Fase de apagado | 1s |
| * Actualización pantalla |  |
| * Numérico | 60ms |
| * Brillo | 20ms |
| * Sonido | 40ms |
| * Arrancado | 400ms |
| * Protecciones | 6us |
| * Actualización barra de potencia | 6us |

|  |  |
| --- | --- |
| Estado SLEEP |  |
| * Texto apagado |  |
| * Sonido Apagado |  |
| * Espera del botón Central |  |
| * Release rápido | 600/1200ms |
| * Se mantiene apretado | x600ms |
| * Despertar | 1081ms |
| Gesto de Estados |  |
| * START-UP to STOP | 5us |
| * RUN to STOP | 300ms |
| * STOP to RUN | 300ms |
| * STOP to SLEEP | 3us |
| * SLEEP to START-UP | 60ms |
| Rutinas Ciclicas |  |
| * USB Check | 2us |
| * OUTPUT | Off 8us  ON 273 us |
| * LOGGING | 70us |
| * Comprobación BOTONES | 11us |
| * Sensado | 243us |
| * Boost Check | 3us |

# Librerías

## Librerías Propias

### BOTONERA

El sistema tiene 3 botones sobre los que se puede actuar. 2 en la parte frontal del body y uno en la parte superior de la batería.

La función de estos botones será: Control del voltaje de salida, activación y desactivación de la salida de voltaje y Apagado y encendido de la batería.

* Dpad.h

Esta librería contiene las funciones necesarias para la detección de pulsaciones cortas (clicks) y pulsaciones largas (Longpress) sobre cualquiera de los 3 botones del sistema

* + ReadDirPad ():

Esta función comprueba el estado de las señales que indican el estado de los pulsadores. Devuelve una variable que indica el evento que ha tenido lugar, pudiendo ser ninguno.

Una dinámica que se suele presentar en los pulsadores son los rebotes, por lo que es necesario implementar un temporizador que filtre estos rebotes. Durante el tiempo que esta activo el temporizador se ignorara cualquier cambio que se produzca en la línea del pulsador. Este temporizador se activará tanto en la detección de los flancos de subida como los de bajada.

* Click:

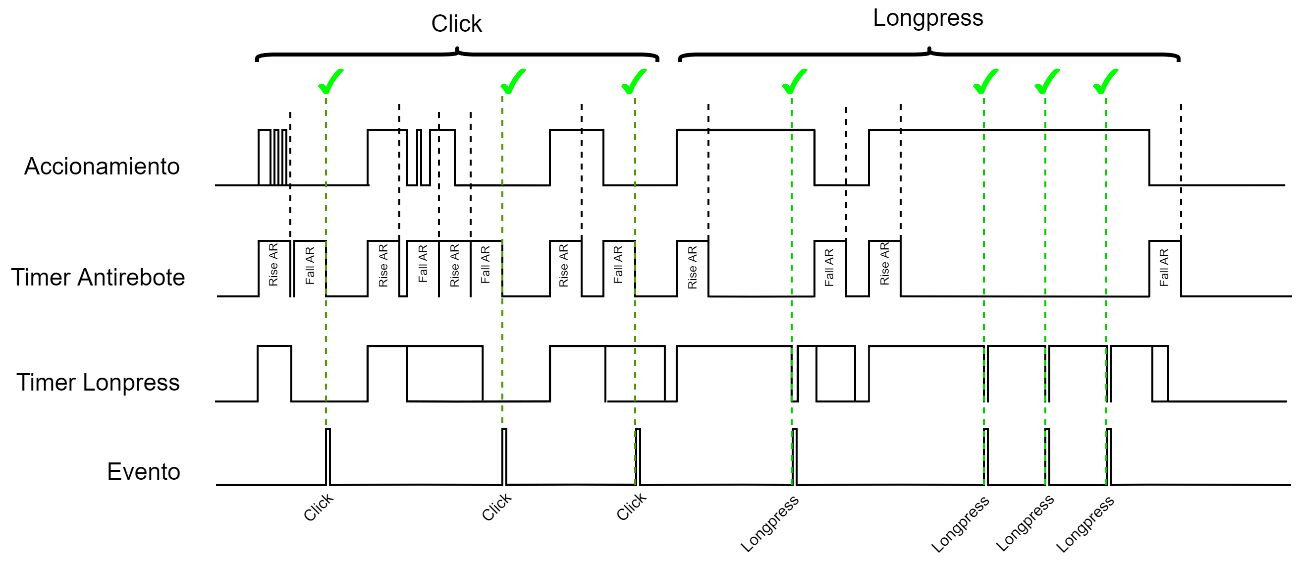
1. Se detecta el flanco de subida
2. Inicio del timer de anti-rebote y del timer que detecta un Longpress
3. Tras finalizar el timer de anti-rebote, se detecta un flanco de bajada antes de que el timer de longpress haya terminado.
4. Inicio del timer de anti-rebote.
5. Finaliza el timer de anti-rebote.
6. Se da el aviso de que se ha dado un Click event.

* Longpress:

1. Se detecta el flanco de subida
2. Inicio del timer de anti-rebote y del timer que detecta un Longpress.
3. Finaliza el timer anti-rebote.
4. Finaliza el timer de Longpress.
5. Se actualiza el evento a evento de Longpress del boton asociado.
6. Inicio del timer de longpress con otro tiempo más corto.
7. Finaliza el timer de Longpress.
8. Se actualiza el evento a evento de Longpress del boton asociado.

Se repite los eventos 6-7-8 hasta que se detecte un flanco de bajada.

Los eventos son One-Shot, es decir, solo se devuelven de la función en el momento que ocurren, el resto de tiempo la función devuelve NO\_EVENT.

En el siguiente diagrama de tiempo se puede observar el funcionamiento y detección de los eventos de click y longpress en cualquiera de los 3 botones.

Los tiempos establecidos para los timers son los siguientes:

* Anti-rebote: 50ms.
* Longpress: 1000 ms + T anti-rebote.
* Lonpress corto: 500ms

### PANTALLA LED

Diplay led de tamaño 15x7, de 105 leds blancos controlados por el IS31FL3731. Este display se "parte" en dos, de manera que queda una pantalla de 15x6 que se utiliza para la interfaz de la batería con el uso de caracteres ASCII y/o bitmaps, y además queda una "barra" de leds de 15x1.

La comunicación con el integrado es I2C y existen previamente librerías que permiten escribir caracteres y bitmaps. Por ello esta librería actúa de wraper de esas librerías y construye las herramientas necesarias para el sistema.

* Bitmap.h

En este archivo, se encuentran todos los bitmaps necesarios usados para la visualización en el display.

Entre estos bitmaps se encuentran:

* Caracteres ASCII.
* Bitmaps de la animación de carga.
* Bitmap de la animación de error.
* Bitmaps que actualmente no se encuentran en uso.

Todos los bitmaps cuentan con 2 versiones, tamaño de 7 leds de alto y de 6 leds de alto, para el caso que se está librería se pueda usar en otros proyectos.

Además de los bitmaps, en el .h se encuentra la función *Char2Bitmap()* que recibe como argumento un carácter y devuelve el bitmap correspondiente a ese carácter.

* Display.h

Esta librería agrupa las principales funciones de control de la visualización en el display.

* + *SwitchScreenOff()*

Apagado de la pantalla.

* + *DisplayText(cadena)*

Esta función muestra en el display una cadena de caracteres en un scroll lateral der-izq siempre que el ancho de los caracteres de la cadena (en pixeles/leds) supere el ancho de la pantalla (15px).

* + *DisplayVolt(volt)*

Esta función imprime la pantalla de voltaje, que muestra le voltaje al que se encuentra la salida de la batería, con un decimal.

* + *DisplayCap(capacity)*

Esta función muestra la pantalla de porcentaje de capacidad de batería. si la capacidad es 100% muestra la pantalla FULL.

* + *DisplayBattCharging(capacity)*

Esta función muestra el marco de una batería, se encarga de ir rellenando e manera solida la capacidad recibida, y deja parpadeando la que sería el próximo 10% que se encuentra cargando.

* + *DisplayUSBIn()*

Animación de Entrada del USB. A los 2 segundos disminuye su brillo.

* + *DisplayUSBOut()*

Animación de salida del USB.

* + *DisplayDiagnosticMode()*

Animación del modo Diagnostico.

* + *DisplayLowBattery()*

Animación Batería Baja.

* + *DisplayError()*

Animación de Error

* + *ScreenON(bright)*

Encender toda la pantalla con una cantidad de brillo determinada.

### BARRA DE POTENCIA

La barra de potencia consiste en la una fila de 15 leds que se usa como medidor visual del consumo/potencia instantánea de la batería.

El objetivo de la “barra de potencia” es la representación visual de la dinámica de la maquina de tatuar durante su funcionamiento.

Dentro de la “barra de potencia” se usa el primer led como identificador de la salida. Si la salida se encuentra activa, el led se enciende, mientras que, si la salida está desactivada, el led se apaga.

* Power\_bar.h

Además de funciones que interactúan con los leds de la barra de potencia, este fichero almacena las características de la dinámica de la barra de potencia como:

* Potencia máxima visualizada: 2.5KW.
* Posición del led que representa la salida: 1
* Potencia representada por cada Led: 2.5KW/15 = 167W
  + *PowerBar(leds)*

Esta función enciende, de derecha a izquierda, una cantidad *leds* de pixeles y apaga el resto de los pixeles de la barra de potencia.

* + *UpdatePowerBar(power sample)*

Calcula la cantidad de pixels que tiene que haber encendidos en la barra de potencia en función del valor de potencia que se pasa como argumento de la función. Tras el cálculo, actualiza la barra de potencia.

* + *LedWork(State\_Led)*

función que permite encender y apagar el pixel que representa el estado de la salida del voltaje.

### BUZZER

El sistema cuenta con un zumbador(buzzer), que actúa como indicador/confirmador sonoro de las interacciones que se tienen sobre la batería.

A través de la librería nativa de Arduino de *tone* se generan las distintas notas.

* tone(pin, frecuencia,[opcional] duracion)

Genera una onda cuadrada de la frecuencia especificada (y un ciclo de trabajo del 50%) en un pin. Se puede especificar una duración, de lo contrario la onda continúa hasta una llamada a noTone(). El pin puede conectarse a un zumbador piezoeléctrico u otro altavoz para reproducir tonos.

Sólo se puede generar un tono a la vez. Si ya se está reproduciendo un tono en un pin diferente, la llamada a tone() no tendrá efecto. Si el tono se está reproduciendo en el mismo pin, la llamada establecerá su frecuencia.

* Notes.h

Archivo que contiene una LUT con las frecuencias asociadas a cada nota musical.

* Buzzer.h

Para poder generar un efecto de sonido (SF) es necesario crear una “melodía” con las notas y los tiempos. Cada melodía se graba en un array con el siguiente formato:

MELODIA[] = {*NOTA, tiempo de reproducción, espacio entre notas, NOTA, tiempo de rep…}*

Las melodías del sistema son:

* Sonido de Encendido.
* Sonido de Apagado
* Sonido de Bajada de voltaje.
* Sonido de Subida de voltaje.
* Sonido de Activación de la Salida.
* Sonido de Desactivación de la Salida.
* Sonido de Conexión de USB.
* Sonido de Desconexión de USB
* Sonido de Error.
* Sonido de Batería baja.
  + *InitBuzzer(pin,mode)*

Función que configura el pin asociado al buzzer y configura los sonidos del sistema. Es posible que existan varios modos (o grupos de melodias) del sistema, en función del argumento *mode*, se seleccionan un grupo de melodías u otras. Existen un modo por defecto en el caso que no se especifique en el argumento.

* + *playsound(num)*

Función encargada de reproducir la melodía seleccionada por el argumento *num*.

1. La función copia en una variable local el array correspondiente a la melodía. (función *getSound()* )

De manera bloqueante la función recorre el array.

1. En cada iteración reproduce el sonido usando la función *tone()* con la frecuencia y el tiempo de reproducción como argumentos.
2. Se lanza un timer con la duración de la nota más el tiempo entre notas.
3. Una vez finaliza el timer, se continua con la siguiente nota.

Este proceso se repite hasta que se reproducen todas las notas de la melodía.

* + *getSound(sound\_id)*

Esta función devuelve un array con la melodía correspondiente al *sound\_id*.

### DCDC

Diagrama, Esquemático

Descripción generada automáticamenteLa conversión de voltaje llevada a cabo por el integrado DCDC funciona según la tensión que existe en el pin de realimentación de dicho integrado.

Este voltaje va en función del divisor resistivo que se forma entre R43 y la resistencia inferior.

Diagrama, Esquemático

Descripción generada automáticamentePara poder ofrecer un voltaje a la salida variable, es necesario variar dicho divisor resistivo. En el sistema se ha implementado una resistencia variable utilizando un controlador diodos LED, el TPIC2810D. Este controlador funciona a modo de interruptor entre la salida comandada y, en este caso, GND. De esta manera es posible generar una escalera de resistencias en paralelo y por tanto un rango de voltajes a la salida del DCDC.

La comunicación con el TPIC2810 es una comunicación I2C. Es necesario la obtención de un Look Up Table, con la correspondencia entre voltaje obtenido y comando a enviar al TPIC2810.

Esta tabla se obtiene mediante calculo externo en una tabla de Excel.

* Modo Boost/No boost.

Como decisión de diseño, se decide crear otra LUT cuya correspondencia sea asignar a un voltaje el comando correspondiente a 0.5v por encima. De esta manera se podrá crear una sensación de mayor potencia por parte de la batería. Esta feature será comandada de manera externa a la librería, pudiendo usar cualquiera da las dos LUT de manera independiente y ajustándose a los requisitos del sistema.

* DCDC.h

Es librería implementa todas las funcionalidades necesarias en relación con el control del voltaje de salida actuando sobre el integrado TPIC2810, mediante comunicación I2C, y por tanto sobre el divisor resistivo de la realimentación del DCDC.

Se declara una clase *dcdc\_controler* y se crean los siguientes métodos asociados:

* + *dcdc\_controler(pin)*

Creación de un objeto del tipo dcdc controler. Cada objeto estará liga a un DCDC cuyo enable se encuentra en el pin*.*

* + *EnableDCDC(bool)*

Permite activar o desactivar el enable del DCDC.

* + *SetVoltage (volt, mode)*

Comunicación I2C que fija el voltaje deseado. La decisión de que LUT utilizar viene determinada por el valor de la variable mode.

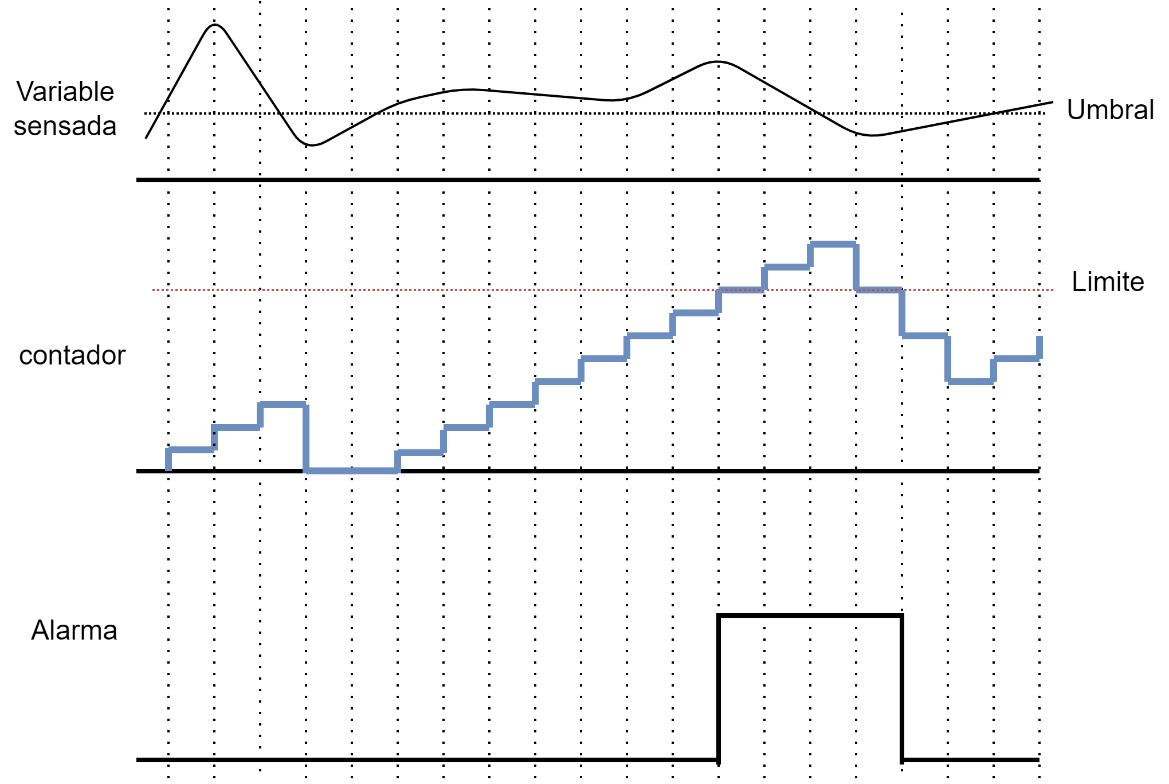
El sistema cuenta con un rango de 121 valores que van desde:

* Los 4.0v a los 16.0 v, en incrementos de 0.1 aproximadamente en el caso de la LUT No Boost.
* Los 4.5v a los 16.5v, en incrementos de 0.1 aproximadamente en el caso de la LUT Boost.

### SENSADO

El sistema implementa un algoritmo propio para la gestión de las protecciones. Este algoritmo consiste en la creación de rampas de subida o bajada en función de si la variable sensada se encuentra por encima o por debajo de un umbral.

Cada vez que se realiza una comprobación de la variable sensada, y esta se encuentre por encima del umbral, un contador se incrementara una cantidad determinada. Mientras que siempre que se realice la comprobación y la variable sensada se encuentre por debajo del umbral, el mismo contador se decrementa una cantidad.

 Si el contador alcanza un valor limite, el algoritmo da la alarma para que se puedan tomar las acciones correspondientes.

De esta manera se crea una protección que se protege frente a momento puntuales que provocarían el salto de la alarma de funcionar simplemente como umbral.

Los parámetros configurables de este algoritmo son:

* Umbral.
* Incremento.
* Decremento.
* Límite de cuenta.
* HealthMonitor.h

Se declara una clase de *HealthMonitor* y se crean los siguientes métodos:

* + *HealthMonitor (threshold, inc, dec, lim).*

Creador de la clase con las variables que fijan el umbral de comparación, el incremento por cada vez que se encuentre por encima, el decremento por cada vez que se encuentre por debajo del umbre y el limite de cuenta.

A la hora de crear el *HealthMonitor* es muy importante tener en cuenta el tiempo de muestreo.

Como aproximación se utiliza la siguiente función:

El decremento marcara la independencia a los picos que pueda sufrir la señal sensada.

* + *setCounter ()*

Existe la posibilidad de tras saltar una alarma se quiera resetear el contador del HealthMonitor, o simplemente fijar un valor de inicio.

Esta función permite dar un valor al contador.

* + *getCounter ()*

Devuelve el valor del contador del HealthMonitor en ese instante.

* + *Check(sample)*

función que realiza la comparación del *sample* con el umbral, e incrementa/decrementa el contador en función del resultado.

Si el contador alcanza el límite, esta función devuelve un aviso de alarma.

* + *GetSample (ADCpin)*

Para la adquisición de un sample se utiliza un filtrado por desplazamiento de media de 8 posiciones.

Se dispone de un array FIFO de 8 posiciones donde se van almacenando los simples previos. Cada vez que se realiza una muestra se hace la media de dicha muestra con los 8 valores almacenados. El resultado obtenido es el valor que devuelve esta función y el que se almacena en el array como nuevo sample.

### LOGGING

Por decisión de diseño, al tratarse de un sistema de tiempo real, es necesario implementar alguna funcionalidad de seguimiento de aquellas variables que permitan definir 2 cosas:

* Punto de trabajo del sistema en tiempo real.
* Estado del sistema con el paso del tiempo.

Debido a la dinámica del sistema (un sistema que pierde alimentación y por tanto resetea las variables en el inicio) es necesario tener 2 tipos de almacenamiento: RAM y ROM.

El almacenamiento RAM estará destinado a almacenar aquellas variables ligadas al ciclo de programa actual. (Voltaje instantáneo, estado de la salida, etc)

El almacenamiento Rom estará destinado a almacenar aquellas variables que tienen información relevante de manera permanente (núm. errores, valor del voltaje antes del apagado, etc). En línea con este conceto, se decide implementar 2 memorias ROM independientes, una en la batería y otra en el body.

* Almacenamiento en el BODY.

El micro-controlador SAMD21G18 tendremos una memoria de 256Kbytes. La memoria de programa esta divida en columnas (rows) y paginas (pages). Cada columna contiene 4 páginas. El tamaño de estas páginas está predefinidas a 64 bytes.

Si deseamos almacenar un tamaño de 32 bits (4 bytes) por variable, tenemos que en cada página podemos almacenar 16 variables. Sin embargo cada pagina tiene una cabecera que ocupa 4 bytes, por lo que se almacenan 15 variables.

Sin embargo, el SAMD21G18 no posee una memoria no volátil dedicada al usuario. Es necesario configurarla de la parte de memoria Flash. Por decisión de diseño se decide generar un espacio de 256 bytes, es decir, 4 paginas de 64 bytes, lo que hace un total de 60 variables almacenables.

Estas variables se almacenan en la parte no volatil de la memoria. Sin embargo, esta memoria tiene una vida útil limitada (10.000 escrituras), por lo que es necesario que se controle muy bien el momento de la escritura.

La emulación de la EEPROM en el SAMD se realiza con una función externa. El almacenamiento de los datos se realiza en un array de datos de tamaño 16bits por dato. Es necesario tener funciones destinadas al indexado de los datos en los procesos de lectura/escritura.

BATERIA

La batería cuenta con una memoria EEPROM de 128 paginas de 16 bytes por páginas. El acceso se realiza byte a byte tanto para escritura como lectura. Son necesarias funciones que se encarguen de la fragmentación y fragmentación de los datos almacenados.

El almacenamiento volátil se realiza en una estructura con los atributos siguientes:

* Value: Valor volátil de la variable
* Category: 2 categorías que indican la dinámica de la variable: Muestreada en el modo diagnostico o no y otra categoría que indica si se guarda en eeprom, batería o ninguna.
* Name: Cadena que se muestra en el modo diagnóstico.
* EEPROM\_LUT.h

Archivo donde se almacena las posiciones de las eeprom de la batería y del body.

* EEPROMBitbang.h

Librería de las funciones que implementan la comunicación I2C por bitbang y las funciones de fragmentación y desfragmentación de los datos de la batería.

* Diagnostic.h
  + *ResetBateriaEeprom()*

Resetea todas las posiciones de la tabla de memoria de la eeprom de la batería.

* + *Init\_diagnostic\_elements()*

Inicializa el array volátil. Algunos valores se inicializan a 0 y otros son consultados en las correspondientes memorias EEPROM.

* + *LogDiagnosticData(data,address)*

Actualiza el valor de una variable del array volátil comprobando que el dato es distinto, en cuyo caso se activa un flag de que ha habido un cambio en la memoria volátil.

* + *ReadDiagnosticData(address)*

La funcion devuelve el valor de la posición  *address* del array volátil de la variables de diagnostico

* + *IncrementDiagnosticData(in,address)*

La funcion incrementa en la cantidad indicada en *inc* el valor de la variable almacenada en la posición *address* del array volátil.

* + *SaveEepromChasis()*

Salvado en la eeprom del body todas aquella variables del array volátil que tengan la categoría 1 de “chasis [C]”. Previamente se comprueba el flag indicador de que ha habido cambios en el array de volátil. En caso positivo se procede al volcado de datos.

* + isValid

Devuelve true si la memoria del body contiene datos y false si la memoria no contiene ningún dato.

* + UpdateEepromBatery()

Actualización de los valores cuya categoría 1 es “Battery [B]”. Para evitar realizar escrituras en balde, antes de sobrescribir una posición, se lee el valor de dicha posición y se compara con el valor a guardar. Solo si el valor es distinto al almacenado, se produce la escritura.

* + *PrintStaticData()*

Impresión por el puerto serie de las variables del array volátil que tengan su categoría 2 igual a “Serie [S]”.

## Librerias Ajenas

* FlashStorage.

Esta librería permite la emulación de una porción de memoria EEPROM sobre la memoria Flash del SAMD.

* Adafruit\_IS31FL3131\_Library + Adafruit\_GFX\_LIBRARY.

Las librerías de adafruit que proporcionan las clases y métodos que actúan de wrapper sobre la comunicación I2C del controlador de la pantalla led.

* MilliTimer

Creación de temporizadores no bloqueantes.

* SAMD\_AnalogCorrection

Librería que suministra funciones para el calibrado del ADC del SAMD.

* SlowSoftI2CMaster

Librería que suministra las funciones de nivel bajo de comunicación I2C utilizadas en las librerías de BitBang.

* ArduinoLowPower + RTCZero

Conjunto de librerías de bajo consumo que suministran las funciones que permiten inducir al SAMD en modo Sleep y despertarlo.