TESIS DE GRADO
EN INGENIERÍA ELECTRÓNICA

Data Logger EP

Autores:
Fernández Burnengo, Mariano
47366
Manchart, Darío Ruben
47127

RESUMEN

Se realizó el prototipo de un dispositivo que permite, mediante el uso de botones e indicadores leds, registrar la actividad de un paciente enfermo de Parkinson a lo largo del día. Se establece que el paciente debe indicar, cada cierto tiempo previamente establecido, el estado en el cual se encuentra. Dicho estado se cuantifica a partir de porcentajes, donde 0% indica malestar y 100%, bienestar.

Actualmente, se usa la misma metodología pero usando papel y lápiz, donde el paciente tiene que ir completando los datos mencionados. Estimamos que el dispositivo mejorará la efectividad del método. Así también, el médico puede descargar los datos mediante un ordenador, con el cual podrá procesar los datos. Por lo cual se simplificarán los procesos y habrá mayor precisión en el diagnóstico del paciente.

La tecnología ha avanzado en todos los ámbitos, y la medicina no es la excepción. Aspiramos a que el dispositivo sea una mejora en la calidad de vida de los ciudadanos que padecen esta enfermedad.

ABSTRACT

Here we developed the prototype of a device that allows, through the use of buttons and led indicators, record the activity of a patient suffering from Parkinson disease throughout the day. The patient must indicate his “state” in different periods of observation. This “state” is quantified from percentages where 0 % indicates discomfort and 100 %, wellness.

Currently, the same methodology is used but using pen and paper, where the patient has to write the above data manually. We estimate that the device will improve the effectiveness of the method. Also, the doctor can download the data using a computer, and process it. Therefore, the process will be simpler and more accurate for a better patient diagnosis.

Technology has advanced in all areas, and medicine is no exception. We hope that the device will be an improvement in the quality of life of citizens suffering from this disease.

ÍNDICE

RESUMEN 2
ABSTRACT 2
1. INTRODUCCIÓN 6
1.1 HISTORIA y ANTECEDENTES 6
1.2 JUSTIFICACIÓN del PROYECTO 6
2. OBJETIVOS 8
2.1 FINALIDAD del PROYECTO 8
2.2 PLANTEAMIENTO del PROBLEMA a RESOLVER 8
3. DEFINICIÓN del PRODUCTO 10
3.1 REQUERIMIENTOS 10
3.2 ESPECIFICACIONES FUNCIONALES y de DISEÑO 11
3.2.1 Especificaciones del Hard 11
3.2.2 Especificaciones del Soft 11
3.3 Casa de Calidad 12
4. ANÁLISIS de FACTIBILIDAD 14
4.1 FACTIBILIDAD TECNOLÓGICA 14
4.1.1 Propuesta de Alternativas de Diseño 14
4.1.2 Elección de una Solución 14
4.1.3 *DFMEA* 16
4.3 FACTIBILIDAD de TIEMPOS 17
4.2.1 *PERT* 17
4.2.2Simulación de Montecarlo 17
4.2.3 Gantt 19
4.3 FACTIBILIDAD ECONÓMICA 20
4.3.1 *Análisis de mercado* 20
4.3.2 *Tamaño y evolución del mercado* 20
4.3.3 *Análisis de la competencia* 20
4.3.4 *Proveedores y distribuidores* 21
4.3.5 *Canales comerciales* 21
4.3.6 *Marketing* 21
4.3.7 *Presupuesto de Marketing y ventas* 21
4.3.8 *Plan de ventas* 21
4.3.9 *Procesos internos* 22
4.3.10 *Personal* 22
4.3.11*Activos fijos* 22
4.3.12 *Gastos operativos y administrativos* 23
4.3.13 *Análisis económico financiero* 23

4.4 FACTIBILIDAD LEGAL 26
5. INGENIERÍA DE DETALLE 28
5.1 HARD 28
5.1.1 Diagrama de bloques del hard 28
5.1.2 Descripción detallada de cada bloque 28
5.1.3 Detalles de selección y cálculo de los elementos circuitales de cada bloque 30
5.1.4 *Plan de pruebas de cada módulo* 33
5.2 SOFT 35
5.2.1. Diagrama de estados y flujo gramas 35
5*.2.2. Análisis de complejidad* 38
5.2.3. Descripción de subrutinas 39
5.2.5. Plan de prueba de módulos y de depuración de soft 42
6. CONSTRUCCIÓN DEL PROTOTIPO 43
6.1 DEFINICIÓN DE LOS MÓDULOS 43
6.2 DISEÑO DE LOS CIRCUITOS IMPRESOS 43
6.3 DISEÑO MECÁNICO 44
6.4 DETALLES DE CONSTRUCCIÓN Y PRECAUCIONES ESPECIALES DE MONTAJE 45
7. VALIDACIÓN DEL PROTOTIPO 46
7.1 VALIDACIÓN DE HARD 46
7.1.1 *Plan y Protocolos especiales de medición* 46
7.1.2 *Medidas* 47
7.1.3 *Evaluación* 47
7.1.4 *Resultados* 47
7.2 VALIDACIÓN DE SOFT 48
8. ESTUDIOS DE CONFIABILIDAD DE HARD Y DE SOFT 50
8.1 CONFIABILIDAD DEL HARD 50
9. CONCLUSIONES 51
9.1 EXCELENCIAS. OBJETIVOS ALCANZADOS 51
9.2 FALLOS. RECOMENDACIONES PARA FUTUROS DISEÑOS 51
10. ANEXOS: TÉCNICOS. JUSTIFICATIVOS. DESCRIPTIVOS. DOCUMENTALES 52
10.1 GLOSARIO 53
10.2 PLANOS 54
Circuito Impreso 54
Gabinete 3D 55
Tapa del gabinete 55
Caja del gabinete 56
10.3 ESQUEMAS 57
Esquemático Microprocesador y conexiones eléctricas 57

Botoneras e indicadores lumínicos 58
Parlante 59
Dispositivo de almacenamiento 59
Sistema de alimentación 60
10.4 LISTADO DE PARTES 62
10.5 CÓDIGOS DE SOFT 64
10.6 HOJAS DE DATOS DE COMPONENTES 134
11. BIBLIOGRAFÍA 142

1. INTRODUCCIÓN

1.1 HISTORIA y ANTECEDENTES

La Enfermedad de Parkinson es un trastorno neurodegenerativo, que se traduce principalmente en dificultades motrices, alterando la capacidad del enfermo de controlar la contracción de los músculos. Los síntomas que suelen presentar los pacientes son rigidez muscular, temblor en reposo, alteración de la motricidad fina, alteración de la postura, entre otros. También hay una pérdida de expresión facial, disminución del parpadeo y del movimiento de los brazos al caminar.

Hasta el momento, la ciencia médica no ha encontrado una cura definitiva a la enfermedad, pero sí se han desarrollado tratamientos que permiten que el paciente pueda sobrellevar la enfermedad atenuando los síntomas más molestos. Existe un tratamiento farmacológico y, dependiendo el caso, se pueden realizar intervenciones quirúrgicas de tal forma de optimizar el estado de la enfermedad.

La dificultad motriz se produce por la ausencia en el cerebro de una sustancia llamada ‘dopamina’. El tratamiento farmacológico se lleva a cabo utilizando una droga llamada ‘Levodopa’, de gran eficacia. Esta debe ser consumida por el paciente durante el día, y la frecuencia de consumo dependerá de la respuesta del paciente a la misma. Hay que considerar que la sustancia tiene una cierta latencia hasta que comienza a hacer efecto. A su vez, puede suceder que el efecto caiga antes de que se tome la nueva dosis, momento en el cual el enfermo vuelve a su estado original de enfermedad, sufriendo los síntomas. En este estado, se dice que el paciente está en ‘OFF’. Bajo los efectos de la medicación, se dice que el estado es ‘ON’. Claro que para cada paciente el comportamiento y la respuesta ante el fármaco son diferentes, por lo que al comenzar el tratamiento, el médico le otorga a su paciente un diagrama de filas y columnas, que representan un total de 15 o 20 días, y las 24 horas diarias, subdivididas en intervalos de media hora, respectivamente. Entonces el paciente debe registrar en cada uno de los casilleros, indicando si en ese lapso ha estado en ‘ON’ o ha estado en ‘OFF’. Las horas en las que duerme no son evaluadas. Además, debe indicar con una flecha el momento en el cual toma la medicación.

Una vez finalizado el período de evaluación, el médico analiza los resultados, y en base a ello concluye el estado del paciente y su respuesta a la droga, entre otros factores.

1.2 JUSTIFICACIÓN del PROYECTO

El registro en papel descrito anteriormente se utiliza en la actualidad, y es útil para los médicos para extraer datos que permitan determinar el estado del paciente, su evolución, el tiempo entre ingesta de medicación óptimo, entre otras cosas. Pero en el uso cotidiano se han encontrado algunas dificultades para aplicar este método con mayor eficacia, a saber:

- ✓ La necesidad de estar constantemente llevando una lapicera y un papel.
- ✓ La información estará dada en valores discretos, tanto en referencia al horario (anotación cada media hora) como al estado de la persona (on/off).

- ✓ Por la enfermedad misma, es posible que el paciente muchas veces se encuentre en un estado en el cual no le es posible rellenar la hoja, debido a dificultad en los movimientos, tanto por temblor como por rigidez muscular.

- ✓ Muchas veces es una carga pedirle al paciente que cumpla con rellenar el registro y recordar cómo se sintió en cada momento del día, sobre todo teniendo en cuenta que los pacientes suelen ser de edad avanzada.

- ✓ No es posible marcar momentos de las disquinesias, ni cambios intermedios, por lo que se dificulta mucho más para el médico determinar las curvas de tiempo en que hacen efecto las dosis de medicación, u otras curvas de interés.

Por lo mencionado en el párrafo precedente es que la *Fundación Fleni* decidió pegar un salto de calidad en la información que puede ser suministrada al especialista por parte del paciente. Este salto de calidad se basa en implementar un dispositivo electrónico con distintos botones que el paciente pueda presionar y con esto indicar sus síntomas. Si este aparato es capaz de almacenar estos datos y ordenarlos cronológicamente, entonces podrá reemplazar al registro en papel. Además, pueden agregarse alarmas sonoras que avisen al usuario el momento de refrescar su estado o tomar su medicación y más botones para ampliar la información que el paciente pueda dar.

El mayor problema planteado por los médicos de la Fundación fue que el registro en papel no es lo suficientemente práctico para el paciente, y que estos no se preocupan por informar correctamente, se olvidan de hacerlo o no lo entienden. Si el dispositivo es lo suficientemente práctico y el paciente se siente cómodo con el mismo, probablemente le motive usarlo. Si el paciente está motivado para usarlo se preocupará por la fidelidad de los datos que ingrese, y así el registro de información será de mayor calidad y en mayor cantidad.

En principio, esto implicaría una mejora en la calidad de atención, ya que los médicos podrán hacer un mejor diagnóstico. A partir de esto, nuestra propuesta fue agregar la posibilidad de registrar no solo dos estados posibles (on/off), sino también estados intermedios, y así contar con información que antes ni siquiera era posible obtener con el registro manual. Finalmente, si la información es realmente abundante, de calidad y precisa, entonces se la podrá utilizar para investigación científica, potenciando mucho más el alcance del producto.

2. OBJETIVOS

2.1 FINALIDAD del PROYECTO

El proyecto tiene como principal alcance, reemplazar en todo caso posible al registro en papel. Si está disponible, debe ser indudable que se prefiera utilizar el dispositivo electrónico. Es decir, el paciente debe sentir que el cambio de metodología en el análisis, es positivo. Si no es mejor para el paciente, no sirve.

En segunda instancia, es muy importante que su implementación permita al médico un mejor tratamiento de los datos provistos por el enfermo, ya sea por una mejor organización al bajar los datos por software, o por precisión en los datos.

2.2 PLANTEAMIENTO del PROBLEMA a RESOLVER

En un principio, la idea era reemplazar el registro en papel por algún sistema que permita al paciente llevar el registro de su estado de una manera menos complicada. Se pensó que quizás esto sería posible de implementar a través de una aplicación de software que permita instalarse en, por ejemplo, celulares, de tal forma de que sea cómodo, y acoplarlo a algo ya utilizado. Pero esta idea rápidamente se descartó, dado que sería imposible para una persona con esta enfermedad la manipulación de un celular en alguno de sus estados más críticos. Por lo cual, a partir de esto surgió la idea de un dispositivo portable de fácil manipulación para pacientes de este tipo de enfermedad.

La idea principal es que el dispositivo tenga un clock interno con la hora y el día. Mediante la interfaz de botones, el usuario irá registrando eventos en determinados momentos, los cuales serán almacenados en una memoria, indicando el evento ocurrido y el momento en el cual ocurrió. De esta manera el paciente informa cada determinada cantidad de tiempo el estado en el cual se encuentra, simplemente presionando el botón de on/off.

Además de registrar el estado on/off, sería útil agregar un botón en el cual el paciente pueda indicar que se encuentra con disquinesias, lo cual aportaría un dato más de análisis al médico.

Siguiendo con las posibles implementaciones que ayuden al análisis clínico, y a través de las facilidades que nos aporta la tecnología, se pidió un registro analógico del estado del paciente. Esto quiere decir que en vez de colocar un simple botón de on/off, colocar una perilla que se deslice desde el peor estado, pasando por estados intermedios y llegando al mejor estado posible. Esto sería útil porque muchas veces el paciente se encuentra mejor por la mañana que por la tarde, o bien los mencionados estados de on/off se dan con distinta intensidad cada vez, con lo cual ya no habría dos estados posibles, sino más bien alrededor de cinco. Pero sobre todo, permite hacer un análisis mucho más detallado del estado del paciente, permitiendo generar curvas más precisas del comportamiento, viendo cómo es la transición por los distintos estados. Por lo tanto, se decidió que sea configurable por software la posibilidad de registrar estados intermedios entre ‘on’ y ‘off’, de manera que el especialista médico decida si para ese paciente en particular es conveniente su uso o no.

Además de los eventos indicados, el paciente también tiene definidos horarios de toma de medicación. Por este motivo, sería conveniente agregar otro botón que permita indicar al usuario que ha tomado la medicación, permitiendo al médico ver con mayor precisión el tiempo de efecto de la misma.

Como se mencionó al principio, se partió de la idea de que el usuario registre cada cierta cantidad de tiempo, su estado, tal como lo hace el diario. Pero a través del aparato se podría permitir que el usuario marque su estado en el momento que desee, sin que necesariamente haya pasado el tiempo estipulado desde que ocurrió el último evento. Este registro más detallado se puede dejar a voluntad, obligando a que registre *cómo mínimo* cada cierta cantidad de tiempo.

Para cumplir con ciertos requisitos, se pensó que sería buena la implementación de un sistema de alarma, en el cual el dispositivo *avise* que es momento de tomar la medicación, o que es momento de refrescar el estado en el que se encuentra, o cualquier otra indicación que se desee. Estas alarmas podrían ser programadas por el médico antes de entregar el dispositivo al paciente, y quedar registradas hasta que el *programador* (el médico en este caso) lo decida.

Para obtener los datos que el aparato almacena en memoria sería realmente práctico, no solo presentar los datos en estado puro, es decir, lista de relaciones *tipo de evento-instante del evento*, si no también ofrecer al médico una vasta variedad de curvas e índices calculados, permitiendo mucho más sencilla la interpretación de lo registrado por el aparato. Además, se podría programar el tiempo en que es necesario que se refresque el estado (media hora, una hora), programando una nueva alarma o simplemente haciendo titilar el botón de refresco de estado, y así también todas las utilidades que se ocurran.

Referido al software, también se puede establecer un bloqueo, de tal forma de que el dispositivo pueda ser accedido solamente por su dueño, haciéndolo de esta manera más seguro. Además, se puede personalizar el aparato para cada caso, registrando datos personales del paciente, simplemente como detalle.

En cuanto al hardware, tanto para la programación como para la descarga de datos será necesario que el aparato tenga un sistema de comunicación que permita conectarlo a una computadora. Se concluyó que lo más práctico sería comunicarse por puerto USB, dado que hoy día todas las máquinas tienen este puerto y es sencillo de utilizar.

Para cumplir con el requisito de botones de fácil manipulación, se colocarán botones grandes, para los cuales no se requiera mucha presión. Serían botones con leds, que indiquen con claridad su estado de encendido o de apagado.

Dado que el aparato debe ser autónomo, es necesario determinar qué tipo de batería lleva. En un principio se había pensado en la posibilidad de colocarle una pila, por su bajo consumo. Pero siguiendo una política de cuidado de medio ambiente, se ha decidido que lo mejor es utilizar una batería recargable, con lo cual debe incorporarse también un cargador para la misma.

3. DEFINICIÓN del PRODUCTO

3.1 REQUERIMIENTOS

Se pretende que el dispositivo a implementar cumpla ciertos requerimientos:

- ✓ Principalmente, se buscará que el dispositivo se use mediante unos pocos botones de muy sencilla manipulación, dada las dificultades motoras del usuario.
- ✓ Dispositivo portable, de un peso y tamaño considerablemente pequeño (Tamaño de un celular).
- ✓ Considerando el tipo de enfermedad que tiene el paciente que va a utilizarlo, se piensa en un dispositivo resistente a golpes, caídas, presión de botones indeseada, etc.
- ✓ Considerando que el paciente suele ser de edad avanzada, se piensa en un dispositivo absolutamente intuitivo, con botones bien visibles.
- ✓ Los datos recogidos, deben poder ser descargados a un ordenador.
- ✓ Los resultados se presentarán mediante gráficos e índices interpretables por el médico.
- ✓ Sistemas de alarmas totalmente programables por el médico. Tanto para aviso de ingesta de medicación, como de recordatorio para indicarle al paciente que debe informar su estado.
- ✓ Que el médico pueda programar el dispositivo para cada paciente en forma individual.

3.2 ESPECIFICACIONES FUNCIONALES y de DISEÑO

3.2.1 Especificaciones del Hard

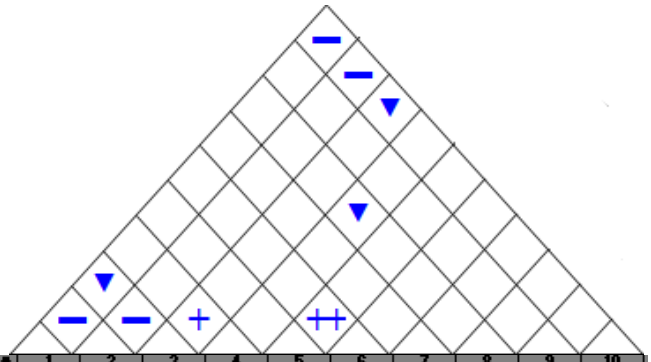
- Módulo de conexión con PC vía puerto USB
- Batería recargable por puerto USB
- Recarga de batería a 5V (+/- 1V)
- Consumo típico: 300mW (+/- 50%).
- Dimensión: 15cmx10cmx3cm. (+/- 25%)
- Peso: 500g (+/- 20%)
- Capacidad de almacenamiento: 256 kBytes (como mínimo 128, como máximo 512)
- Temperatura de trabajo -10°C a 45°C
- 1 a 3 Leds indicadores del estado de la batería
- Alarma sonora
- 4 a 10 botones para informe de estado (ON-OFF-DISQUINESIA-MEDICACIÓN)
- Tamaño de botones de mínimo: 1 cm de diámetro

3.2.2 Especificaciones del Soft

- Posibilidad de configuración de las alarmas del dispositivo
- Posibilidad de habilitación de los botones del potenciómetro
- Acceso al dispositivo mediante contraseña
- Ejecutable multiplataforma
- Visualización de datos en forma de gráficos e índices
- Exportación de datos a Excel

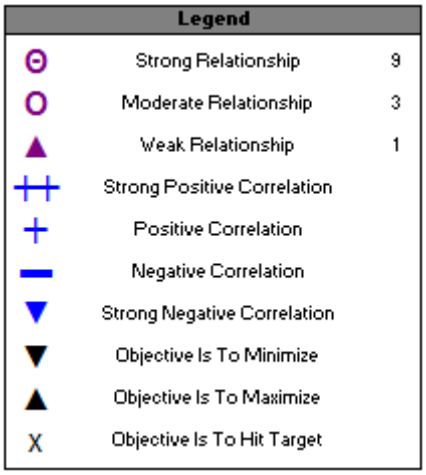
3.3 Casa de Calidad

A partir de los requerimientos anteriormente expuestos, se construyó la casa de calidad:



Row #	Max Relationship Value in Row	Relative Weight	Weight / Importance	Quality Characteristics (a.k.a. "Functional Requirements" or "Hows") / Demanded Quality (a.k.a. "Customer Requirements" or "Whats")	1 Tamaño de Botones	2 Cantidad de Botones	3 Dimensiones del equipo	4 Peso	5 Programabilidad	6 Software de configuración	7 Software de interpretación de resultados	8 Dureza	9 Alimentación	10 Tamaño de perilla de estado
				Column #	1	2	3	4	5	6	7	8	9	10
				Direction of Improvement: Minimize (▼), Maximize (▲), or Target (x)	▲	▼	▼	▼	▲	▲	▲	▲	x	▲
1	9	13,4	9,0	Sencilla manipulación	⊙	⊙	⊙	O						
2	9	11,9	8,0	Portable			⊙	⊙						
3	9	10,4	7,0	Resistente a golpes y caidas								⊙		
4	3	6,0	4,0	Bajada de datos al ordenador (USB)					O					
5	9	9,0	6,0	Alarmas programables					⊙	⊙				
6	3	7,5	5,0	Botones con indicador de estado	O									
7	9	11,9	8,0	Presentación de resultados en gráficos							⊙			
8	9	10,4	7,0	Perilla regulable para indicación de estado	▲	O	O							⊙
9	9	7,5	5,0	Alimentación por bateria recargable									⊙	
10	9	11,9	8,0	Ajustable para cada paciente individual					⊙	⊙				
				Target or Limit Value	1,5cm x 1,5cm	3	10cm x 5cm	60g	Máxima	Óptimo	Óptimo	Máxima	5v	radio > 1,5cm
				Difficulty (0=Easy to Accomplish, 10=Extremely Difficult)	4	5	3	7	7	8	8	6	3	5
				Max Relationship Value in Column	9	9	9	9	9	9	9	9	9	9
				Weight / Importance	153,7	152,2	259,7	147,8	206,0	188,1	107,5	94,0	67,2	94,0
				Relative Weight	10,5	10,4	17,7	10,1	14,0	12,8	7,3	6,4	4,6	6,4

Donde:


Legend
Strong Relationship 9
Moderate Relationship 3
Weak Relationship 1
Strong Positive Correlation
Positive Correlation
Negative Correlation
Strong Negative Correlation
Objective Is To Minimize
Objective Is To Maximize
Objective Is To Hit Target

4.1 FACTIBILIDAD TECNOLÓGICA

4.1.1 Propuesta de Alternativas de Diseño

Microprocesador: En primera instancia lo más relevante a definir es el microprocesador a utilizar para la aplicación. Dados los conocimientos y la experiencia en electrónica se abre un abanico de posibilidades relativamente grande. Las principales alternativas posibles serían las siguientes:

HC12 (Motorola): Fácil de usar, por conocimientos previos y por información disponible ya que es muy utilizado en el mundo académico. Bajo precio. Por otro lado, ha quedado un poco obsoleto dentro del mundo tecnológico.

ATM: Dentro de lo más moderno que se está usando y con un gran alcance. De todas maneras, tiene muchas más funciones de las que se necesitarán y eso se refleja en su precio.

Coldfire: De la misma línea del *HC12*, pero más nuevo. Es bastante básico pero cumple con los requerimientos que necesitamos. Por otro lado, al ser más nuevo, no se cuenta con tanta información acerca de su uso.

Diseño externo: Como ya se indicó en los requerimientos, es necesaria una cantidad limitada de botones, de fácil manejo y uso intuitivo. Es posible hacer simplemente dos botones (on-off), siendo igual al registro en papel. Ampliando, se pueden colocar hasta 4 botones que sean para ON, OFF, disquinesias y medicación. Aún más, en lugar de simplemente poner ON y OFF, se pueden colocar 5 botones con luces, simulando un potenciómetro digital, donde las luces que se prendan sean en función de cómo se siente el paciente. Los botones es recomendable que sean muy visibles y fáciles de accionar para facilitar el uso. El aparato debe ser lo suficientemente grande para que el paciente lo pueda utilizar a pesar de su enfermedad, pero no tanto como para que resulte molesto. Los botones deben estar bien separados y distinguibles. Las luces deben indicar claramente cuál es el estado anterior registrado.

Alimentación: Se puede colocar una pila de larga duración, de 3.3 o 5 Voltios, la cual es más fácil de colocar y no hay que estar pensando en recargarla o en hacer un cargador para el aparato. La otra opción es utilizar una batería recargable que es más cara pero también tiene mayor tiempo de vida y menor impacto ambiental.

Software: El software a utilizar puede diseñarse tanto en C#, Java, Visual Basic o también se pensó en Matlab, ya que simplificaría muchísimo la utilización de gráficos.

4.1.2 Elección de una Solución

Microprocesador: Finalmente, después de una ligera investigación, se obtuvieron las características deseadas en el modelo *MCF51JM128VLD* de Coldfire. Resultó práctico este modelo por su pequeño encapsulado de 44 pines, y su interfaz vía usb, que hoy en día es de uso común en la mayoría de los usuarios de computadora, por lo cual simplifica la utilización por parte del médico. Además, tiene bajo consumo (permite alimentación con 3.3 Voltios) y cumple con las características en cuestión de cantidad de periféricos y memoria disponible.

Diseño externo: Dada la baja complejidad del agregado de botones, se decidió colocar dos botones (disquinesias y medicación).

Para marcar el estado del paciente, una fila de botones simulando un potenciómetro digital, activables por software, en función de la capacidad del paciente.

Alimentación: Basando el criterio de decisión en el impacto ambiental, finalmente se decidió que se utilizará una batería recargable, más precisamente de ion-litio, cuyas características son superiores a otras como las de plomo-ácido o las de níquel-metal-hidruro, en cuanto a efectividad, peso, capacidad de almacenamiento de energía, durabilidad y seguridad, entre otras cuestiones. Si bien la batería se cargará mediante el puerto USB (5 Voltios), se alimentará el dispositivo con 3.3 Voltios para lograr un menor consumo.

Software: El software se programará con lenguaje Java SE, dada su característica de ser multiplataforma, con posibilidad de correrlo en cualquier sistema operativo (dependiendo solamente de la *Java Virtual Machine*). Además, la programación orientada a objetos, hace más simple la realización de la interfaz gráfica. De todas maneras, al usuario se le entregará el programa en dos versiones: *'.jar'* (ejecutable con la JVM) o *'.exe'*.

4.1.3 *DFMEA*

Ítem/Función	Batería		Botones	Microprocesador
Modo de falla	No entrega energía	Led indicador del estado de la batería indica de forma errónea	No recibe eventos	No almacena datos, o no lo hace de forma correcta
Efecto potencial de la falla	No funciona el dispositivo	Información incorrecta al usuario	Mal funcionamiento del dispositivo	No funciona el dispositivo
Severidad	Media	Baja	Media	Alta
Potenciales causas	Desgaste ; Problemas de desconexión del circuito ; problema con el cargador	Problemas de desconexión del circuito (por golpes)	Problema de desconexión del circuito (golpes) o falla del botón	Problema de desconexión del circuito (golpes) o falla del procesador por sobrecarga
Probabilidad de Ocurrencia	4	2	1	1
Controles de prevención	Cambio de la batería antes de que se agote su tiempo de vida	-	-	-
Controles de detección	Medición de la tensión de la batería, para saber si falla la batería o el circuito. Medición de la salida del cargador para saber si este funciona. Led indicador	Medición de la tensión de la batería (si es correcta, falla el led)	Encendido del led o no al momento de la utilización del botón	Descarga de datos
RPN	2	4	0	2
Acciones recomendadas	Colocación de un sistema mecánico para resistencia a los golpes	Colocación de un sistema mecánico para resistencia a los golpes	Colocación de un sistema mecánico para resistencia a los golpes	Colocación de un sistema mecánico para resistencia a los golpes

Tabla 1. *Tabla de análisis de fallas.*

4.3 FACTIBILIDAD de TIEMPOS

4.2.1 *PERT*

A continuación se presenta una tabla aplicando el método PERT (*Program Evaluation and Review Technique*).

Tiempo (días)	Definición de requerimientos	Definición de especificaciones	Diseño de hardware	Diseño de software	Implementación en hardware y software
Optimista	31	15	40	30	45
Probable	31	21	50	45	60
Pesimista	31	40	80	60	90
PERT	31	23	53	45	63
Desde 1/8 (PERT)	31-ago	23-sep	15-nov	07-nov	17-ene
Peor tiempo	31-ago	09-oct	28-dic	08-dic	09-feb
Mejor tiempo	31-ago	15-sep	24-oct	14-oct	29-nov
(Comienzos Tempranos)	-	31-ago	23-sep	23-sep	15-nov
(Comienzos Tardíos)	-	31-ago	23-sep	31-sep	15-nov

Tabla2. *Tabla con cálculo del PERT y otros tiempos dentro del plan de trabajo.*

4.2.2Simulación de Montecarlo

En el siguiente diagrama se indica el camino crítico para la fabricación del producto.

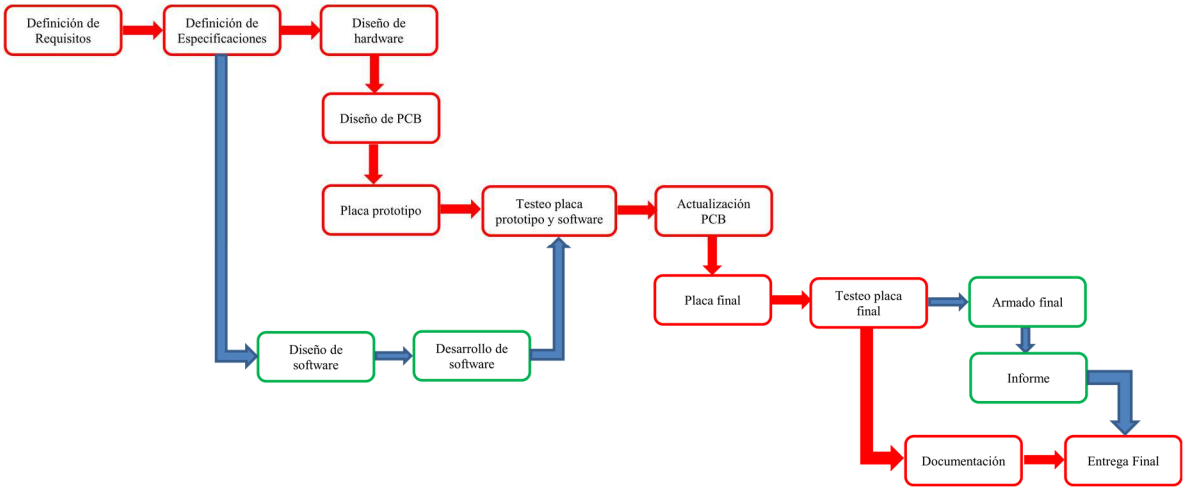


Gráfico 1. *Esquema sucesión de tareas. Camino crítico en rojo.*

Obtenido el camino crítico, se realiza la Simulación de Montecarlo, basado en los tiempos optimistas, probables y pesimistas. Para realizarlo, se utilizó el programa @Risk, que funciona como una herramienta de Excel.

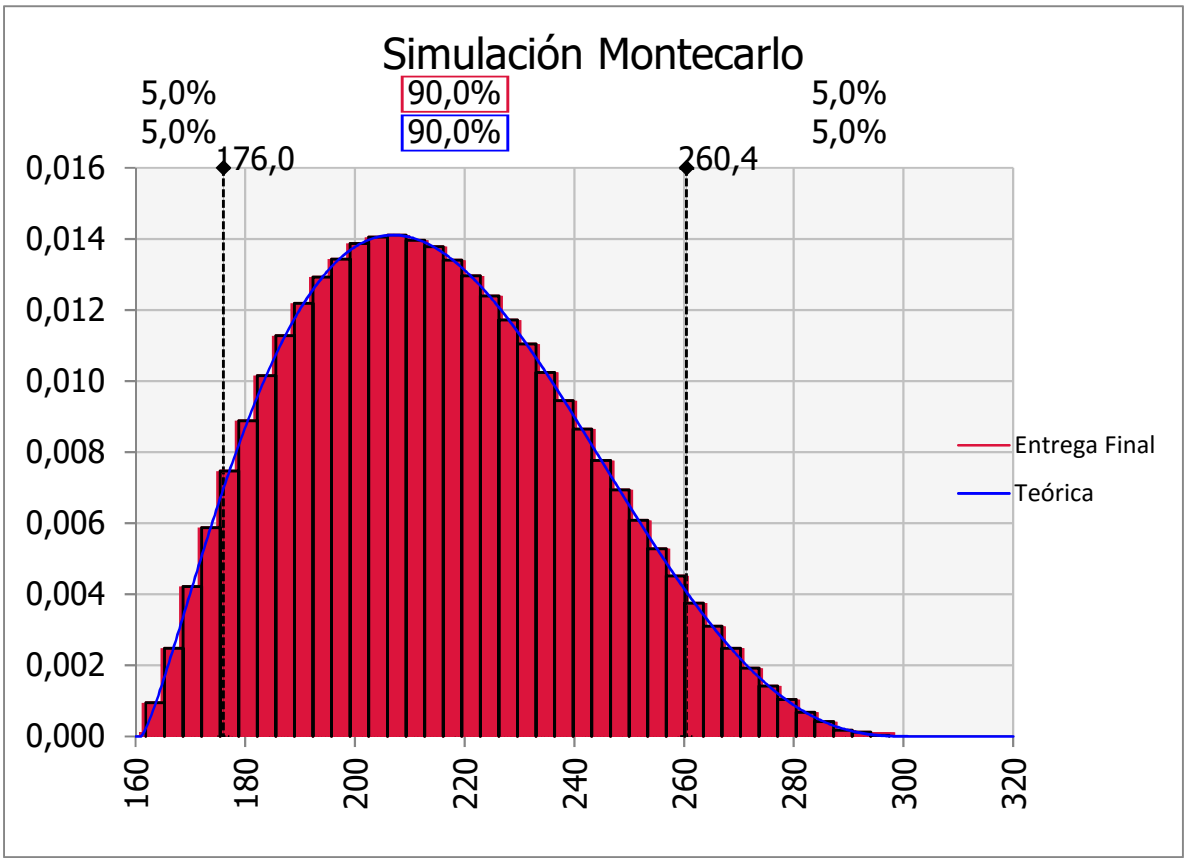


Gráfico 2. *Simulación de Montecarlo.*

Datos y resultados:

Mínimo = 167

Máximo = 276

Media = 215

Desvío estándar = 25

Iteraciones = 100

4.2.3 Gantt

Descripción más detallada del proceso de fabricación del producto utilizando el diagrama Gantt.

ACTIVIDAD		PRECEDENCIA	INICIO	DURACIÓN	FIN
A	Definición de requisitos		01-ago	30	31-ago
B	Definición de especificaciones	A	31-ago	30	30-sep
C	Diseño de hardware	B	30-sep	40	09-nov
D	Diseño de software	B	30-sep	30	30-oct
E	Desarrollo PCB del sistema completo	C	09-nov	5	14-nov
F	Desarrollo software	D	30-oct	20	19-nov
G	Armado placa prototipo	E	14-nov	5	19-nov
H	Testeo placa prototipo	F	19-nov	5	24-nov
I	Actualización PCB	G	24-nov	4	28-nov
J	Armado placa final	H	28-nov	10	08-dic
K	Testeo placa final	I	08-dic	10	18-dic
L	Armado final de dispositivo	K	18-dic	5	23-dic
M	Documentación	K	18-dic	11	29-dic
N	Informe	L	23-dic	5	28-dic
O	Entrega Final	M	29-dic	1	30-dic

Tabla 3. *Correlación de tareas y tiempos estimados para las mismas.*

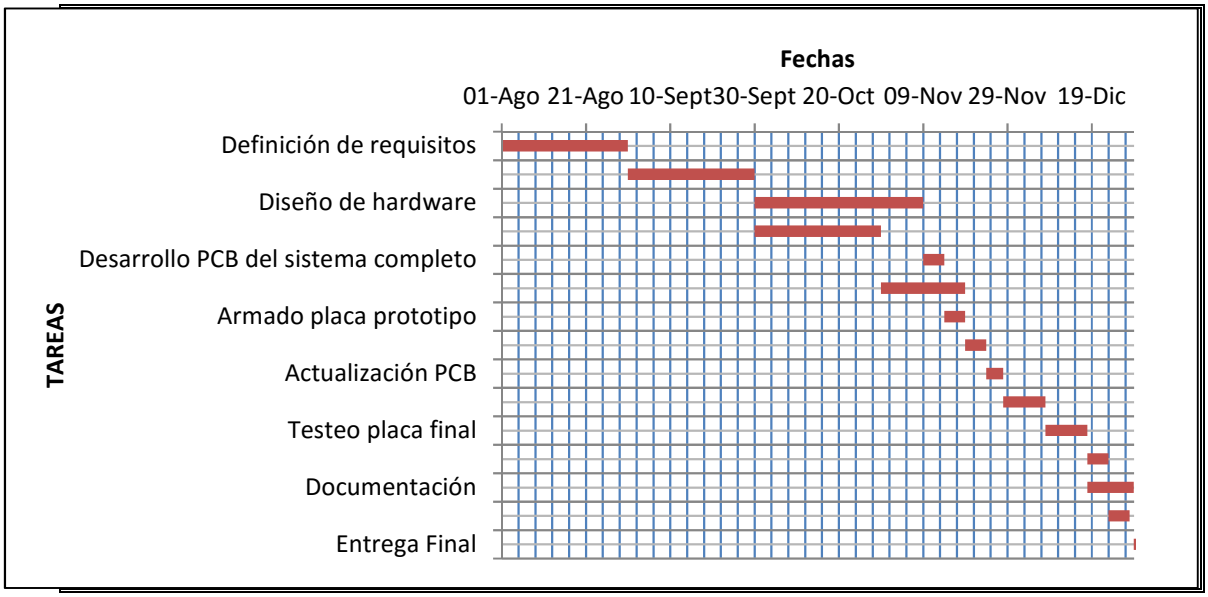


Gráfico 3. *Diagrama de Gantt para la fabricación del producto.*

4.3 FACTIBILIDAD ECONÓMICA

4.3.1 *Análisis de mercado*

Las circunstancias bajo las cuales se comienza a desarrollar el dispositivo coinciden con un momento del país en el cual la tecnología aplicada a la medicina es cada vez más aceptada y demandada, pero sobretodo, más útil. Por lo tanto, siempre será bien recibido un dispositivo que contribuya tanto a la labor del médico como a la vida del paciente. Y esto es precisamente lo que logra el innovador aparato diseñado exclusivamente para la correspondiente enfermedad. También se considera que una de las dificultades que se presentan a la hora de tratar a un paciente es la de dar un diagnóstico preciso de sus problemas. El paciente muchas veces no es confiable a la hora de registrar datos, y el uso actual de una planilla en papel resulta altamente impreciso. Se encuentra entonces la solución más práctica para llegar a ese diagnóstico, de forma que afecta lo menos posible al paciente en su vida diaria, dando un alto grado de confiabilidad al médico.

Siendo la medicina un campo que incide directamente en la calidad de vida humana, es lógico que no se escatime en gastos y que al ser los requerimientos mucho más precisos y definitorios que en cualquier otro ámbito, el valor del producto aumenta, pudiendo trabajar con un alto margen entre precio y costo.

Surge una leve desventaja al considerar que la naturaleza de la enfermedad lleva que aquel que la padece empieza a sufrir sus síntomas no antes de los 40 años, generalmente. Esto deriva que, hoy en día, muchos pacientes serán personas no tan *amigas* de la tecnología, y tal vez se sientan más cómodas con un papel que con un aparato. Sin embargo, hay cada vez una intromisión mayor de la tecnología en la sociedad, y por lo tanto, una mejor adaptación por parte de la población de la tercera edad.

4.3.2 *Tamaño y evolución del mercado*

Para empezar, se estima que alrededor del 1% de la población mundial tiene en sus genes esta enfermedad, por lo cual el mercado en el país, y en el mundo, es verdaderamente amplio, siendo un nicho a cubrir. Solamente en Argentina, hay cerca de 80.000 enfermos de Parkinson.

En un principio puede considerarse la posibilidad de que el médico responsable del tema sea quien adquiera varios aparatos, y los reparta a sus pacientes oportunamente durante la etapa de diagnóstico. Podría contar con unas veinte unidades para empezar. En una etapa más avanzada del proyecto, será posible que el paciente sea quien adquiera el aparato de forma individual y, considerando los posibles avances en la tecnología y las prestaciones del dispositivo, el paciente no tendrá que ir cada vez al consultorio a que el médico extraiga los datos, si no que se podrán transmitir los datos por algún medio como internet o GSM, por dar algún ejemplo. Esto habilitará a realizar una producción mayor aún, ampliando el mercado sin necesariamente ampliar los pacientes. Este mercado será tan grande como pacientes haya.

4.3.3 *Análisis de la competencia*

Es importante destacar que no existe un dispositivo similar que cumpla los requerimientos, por lo cual, en un principio, se contaría con una innovación total, revolucionando el mercado en este ámbito. Siempre es posible que se realicen *copias* del mismo, pero haber tomado la vanguardia en el asunto da garantías de ser dueño de un mercado que siempre irá detrás del productor más reconocido en el rubro, y siendo vanguardistas, ese lugar estará ganado. Además, no es menos el hecho de que al punto que nosotros

podamos estar produciendo en serie, optimizando costos, la competencia aún estará invirtiendo en investigación.

4.3.4 *Proveedores y distribuidores*

Actualmente, dado que en el país no se fabrican componentes electrónicos, los principales proveedores serían extranjeros. Inicialmente, el proyecto se abastecerá de proveedores pequeños como pueden ser Farnell o DigiKey y, si en un futuro las ventas se incrementan a gran escala, se desplazaría a fabricantes mayores como pueden ser Freescale o Atmel.

Considerando que el volumen de ventas será reducido, encarando el proyecto a nivel nacional, se podrá optar por fabricar el PCB, soldar y ensamblar los componentes en empresas nacionales, lo que permitirá reducir costos. Más adelante, si la cantidad de unidades vendidas aumenta, es importante analizar la posibilidad de adquirir una máquina que haga placas, siendo una inversión en maquinaria que ahorrará aún más los costos. Si se quisiera expandir a gran escala la venta, existen varias empresas de origen chino capacitadas para la fabricación y soldado de PCB, como pueden ser Honya Internacional, profesional en la fabricación y exportación de PCB, o Kong Electronics Company Limited, ambas líderes en el mercado oriental. El soldado puede realizarse localmente o a través de dichas empresas chinas. El gabinete es sencillo conseguir un proveedor nacional que le interese realizarlo, si bien será costoso al tratarse de bajas unidades. Dada la relación directa con el Hospital Neurológico Fleni, será sencilla la distribución, ya que en un principio constará de la entrega directa al médico.

4.3.5 *Canales comerciales*

Se utilizará un canal directo para comercializar el producto al cliente final. La publicidad se realizaría formalmente en exposiciones ante los médicos involucrados, en las cuales se presentaría el producto con todas sus características, ventajas e innovaciones. Finalmente, se garantizará la llegada al público y el conocimiento íntegro del producto con el diseño de un sitio web en donde se presenten todas las características. Evitando intermediarios, se minimizarán los costos.

4.3.6 *Marketing*

Dada la seriedad del asunto se entiende que no le cabe una campaña de *Marketing*, más que una presentación individual con los posibles interesados, y una página web que informe minuciosamente al interesado de una manera seria, y un centro de atención al cliente que le quite toda duda sobre la utilidad del aparato.

4.3.7 *Presupuesto de Marketing y ventas*

Por lo antes mencionado ha quedado claro que no habrá un presupuesto dedicado al *marketing* del producto, limitándose a evolucionar en las prestaciones del producto y que esto genere su propio marketing.

4.3.8 *Plan de ventas*

Considerando que la venta será en forma directa y que no habrá escases de clientes, únicamente resta tener en cuenta la capacidad de producción propia. Si se tiene en cuenta los tiempos requeridos para envío de mercadería, fabricación, soldadura y ensamble, y la programación y configuración de los equipos hasta el día

final de entrega, se estima una venta de aproximadamente diez equipos por mes, redondeando 100 para el primer año. En los siguientes años se irá aumentando las ventas en forma progresiva.

4.3.9 *Procesos internos*

La producción total, por lo tanto, se podrá dividir en dos: parte del trabajo realizado por terceros, y otra parte del trabajo realizado por nosotros.

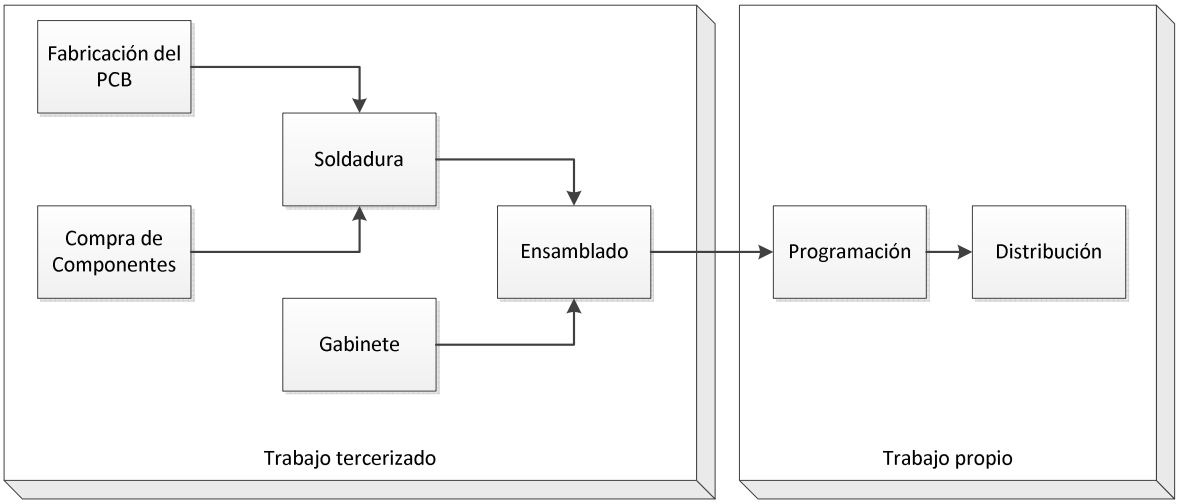


Gráfico 4. *Diagrama en bloques del proceso de producción.*

Se estima que, inicialmente, la fabricación de la placa impresa se realiza en Argentina y quince días hábiles más tarde se recibe el pedido. Se pueden realizar pedidos de 20 unidades para contar siempre con stock de reserva. A su vez, se deben comprar y pedir los componentes. El tiempo de entrega es de tres días. Puede realizarse el pedido del gabinete en simultáneo, lo que demanda aproximadamente una semana.

Una vez obtenido el PCB, se soldarán localmente los componentes. Se considera que un técnico es capaz de soldar diez placas por día aproximadamente. Luego se ensamblarán los equipos terminados, lo cual insume media hora por equipo por técnico, por lo cual se podrían ensamblar diez equipos por día, coincidiendo con el número de placas soldadas.

Finalmente, se programan los dispositivos y se realizan pruebas de funcionamiento y verificación de la calidad. Se entregarán al cliente junto con su correspondiente instalación, puesta a punto del dispositivo y manual de usuario.

4.3.10 *Personal*

Debido al volumen de unidades producidas se estima que no es necesario más personal que el dedicado para las tareas ya mencionadas.

4.3.11*Activos fijos*

Tal como se mencionó anteriormente, existe la posibilidad de la incorporación de una máquina dedicada a la confección de las placas, ahorrando un paso importante en la cadena de producción., que

implicará optimizar tiempos y costos. Esta máquina es posible conseguirla a un precio estimado de 100 US$, con un costo de mantenimiento mínimo.

4.3.12 *Gastos operativos y administrativos*

Gastos Administrativos (Costo fijo/mes)	Alquiler+Gastos	$ 2.000
	Servicio Tel+Int	$ 260
	Gastos generales	$ 500
	SUBTOTAL	$ 2.760
Gasto Operativo/unidad (Costo Variable)	Componentes	$ 200
	Fabricación	$ 500
	SUBTOTAL	$ 700
TOTAL		**$ 3.460**

Tabla 4. *Tabla de gastos operativos y administrativos.*

4.3.13 *Análisis económico financiero*

El objetivo será identificar los criterios necesarios para tomar las decisiones referentes a la ejecución del proyecto. El proyecto se evaluará desde la perspectiva económica y financiera para determinar la factibilidad del mismo.

Para ello se analizarán los flujos netos financieros y económicos a largo horizonte de planeamiento, tomando en cuenta la tasa de rendimiento esperada o TREMA. Para que el proyecto sea atractivo, el entorno de la inversión deberá ser mayor que la inversión realizada. Por tal motivo, se usará como instrumentos de medición:

- Valor Actual Neto (VAN)
- Tasa Interna de Retorno (TIR)
- Período de Recuperación (PR)

VAN: Es un método de evaluación para medir el valor presente Neto del proyecto a través de la actualización de sus beneficios o flujos netos y costos, el factor de actualización está dado por el costo de capital de la empresa.

- Si el VAN es cero, la inversión es indiferente, ya que el inversionista gana justo lo que esperaba obtener.

- Si el VAN es mayor que cero, la inversión es aceptable, ya que muestra cuanto más gana, por sobre lo que quería ganar.

- Si el VAN es menor que cero, la inversión se debe rechazar ya que aunque no indica pérdida, significa cuanto falto para que el inversionista ganara todo lo que quería ganar.

TIR: Es el método que introduce el valor del dinero en el tiempo; su tasa de descuento iguala al valor actual de los beneficios y al valor actual de los costos previstos.

- Si el TIR > TREMA, el proyecto puede ser aceptado debido a que la inversión ganará mas de el costo de los fondos utilizados para financiarlo.

- Si el TIR = TREMA, es indiferente aceptar o no el proyecto.

- Si el TIR < TREMA, el proyecto no se debe aceptar; se ganará menos que el costo de los fondos utilizados para financiarlo.

Costo de Capital Promedio Ponderado (TREMA): La tasa con la cual se evaluarán los flujos económicos netos a lo largo del horizonte de planeamiento, será la tasa de recuperación mínima atractiva de invertir en el proyecto.

Cálculo del Período de Recuperación: El período de recuperación se encuentra determinado por el año anterior a la recuperación total más el costo no recuperado al inicio de año dividido entre el flujo de efectivo durante el año.

En el siguiente cuadro se muestran los datos correspondientes a las unidades vendidas en los primeros cinco años del desarrollo del proyecto.

	Año 1	Año 2	Año 3	Año 4	Año 5
Unidades Vendidas	100	150	200	300	500
Precio por unidad (US$)	200	200	160	160	160
Costo variable por unidad (US$)	130	130	80	80	80

Tabla 5. *Tabla de ventas durante los primeros cinco años.*

En el próximo cuadro se presenta el estado de resultados correspondiente a cada uno de los 5 años detallando precisamente sus componentes. Se utilizaron valores en dólares porque los costos están asociados más que nada al costo de los componentes y de fabricación.

	Año 1	Año 2	Año 3	Año 4	Año 5
Ventas Netas (US$)	20000	30000	32000	48000	80000
Costo Variable (US$)	13000	19500	16000	24000	40000
Costo Fijo (US$)	1800	1800	6240	6240	6240
Ingresos Brutos (US$)	600	900	960	1440	2400
Margen Bruto (US$)	12400	18600	15040	22560	37600
Impuesto a las Ganancias (US$)	4340	6510	5264	7896	13160
Resultado Neto (US$)	**8060**	**12090**	**9776**	**14664**	**24440**

Tabla 6. *Estado de resultados.*

Analizando el primer año, se observa que la estimación de ventas es pequeña. La idea es incrementar las ventas con respecto al año anterior dado que el producto irá haciéndose conocido en el ambiente y su éxito llevará seguramente a nuevas ventas. El margen operativo es u$s200 – u$s130 = u$s70, representando un

porcentaje del 35%. Este valor representa el potencial y margen de error del proyecto. A partir del tercer año se piensa en alquilar un taller en el cual funcione la máquina para hacer las placas disminuyendo el costo variable. También aumenta el costo fijo pero este será prorrateado por la alta cantidad de unidades vendidas que habrá subido considerablemente.

Con respecto al flujo de caja del primer año, se consideró que los gastos fijos se reducirán a internet, teléfono y gastos generales dado que no se alquilará ni taller ni oficina en los primeros años. Como resultado, se obtuvo un tiempo de recupero aproximado de once meses, lo cual indica un alto potencial para la inversión.

Año 1	Mes 0	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5	Mes 6	Mes 7	Mes 8	Mes 9	Mes 10	Mes 11	Mes 12
Facturación	0	400	1000	2000	0	4000	0	0	6000	0	0	0	6600
Unidades vendidas	0	2	5	10	0	20	0	0	30	0	0	0	33
Inversión	3000				3000		4000			4000			
Costos fijos													
Teléfono + Internet		50	50	50	50	50	50	50	50	50	50	50	50
Gastos generales		95	95	95	95	95	95	95	95	95	95	95	95
Costos variables		260	650	1300	0	2600	0	0	3900	0	0	0	4290
Utilidades		255	855	1855	-3145	3855	-4145	-145	5855	-4145	-145	-145	6455
Flujo de caja	**-3000**	**-2745**	**-1890**	**-35**	**-3180**	**675**	**-3470**	**-3615**	**2240**	**-1905**	**-2050**	**-2195**	**4260**

Tabla 7. *Flujo de caja*

En la Tabla 7 se analiza el flujo de caja durante el primer año, haciendo las inversiones correspondientes a la producción de los siguientes meses. Se observa claramente que, de cumplirse las ventas programadas, el primer año podrían recuperarse las inversiones realizadas. Las cifras detalladas no incluyen IVA.

Finalmente, se llega al cálculo del Valor Actual Neto con una tasa de descuento del 10% en dólares, y se calcula la Tasa Interna de Retorno.

VAN = 19500 us$

TIR = 15%

4.4 FACTIBILIDAD LEGAL

El análisis legal se hace basado en la modificación de la ley 24.240, *Defensa al Consumidor*. Si bien dicha ley debe ser leída y cumplida en su totalidad, a continuación se describen solamente aquellos artículos en los cuales se hace referencia expresa a cuestiones técnicas y de documentación.

- Ley 26.361 - Artículo 4: *Información.*

El proveedor está obligado a suministrar al consumidor en forma cierta, clara y detallada todo lo relacionado con las características esenciales de los bienes y servicios que provee, y las condiciones de su comercialización. La información debe ser siempre gratuita para el consumidor y proporcionada con claridad necesaria que permita su comprensión.

- Ley 26.361 - Artículo 5: *Protección al Consumidor.*

Las cosas y servicios deben ser suministrados o prestados en forma tal que, utilizados en condiciones previsibles o normales de uso, no presenten peligro alguno para la salud o integridad física de los consumidores o usuarios.

El dispositivo que aquí presentamos se alimenta a 3,3 Voltios, con una batería de ion-litio, cuya carga se realiza desde la red eléctrica vía transformador. Dada esta descripción, será necesario que el dispositivo tenga homologación de seguridad eléctrica. Debe aclararse que la protección al consumidor será dada bajo el uso esperado del aparato y bajo las condiciones establecidas. Cualquier uso que se haga del aparato en formas no establecidas, fuera del objetivo del mismo, no serán responsabilidad del fabricante.

- Ley 26.361 - Artículo 7: *Contenido del documento de venta*.

En el documento que se extienda por la venta de cosas muebles o inmuebles, sin perjuicio de la información exigida por otras leyes o normas, deberá constar:
a) La descripción y especificación del bien.
b) Nombre y domicilio del vendedor.
c) Nombre y domicilio del fabricante, distribuidor o importador cuando correspondiere.
d) La mención de las características de la garantía conforme a lo establecido en esta ley.
e) Plazos y condiciones de entrega.
f) El precio y condiciones de pago.
g) Los costos adicionales, especificando precio final a pagar por el adquirente.

La redacción debe ser hecha en idioma castellano, en forma completa, clara y fácilmente legible, sin reenvíos a textos o documentos que no se entreguen previa o simultáneamente. Cuando se incluyan cláusulas adicionales a las aquí indicadas o exigibles en virtud de lo previsto en esta ley, aquellas deberán ser escritas en letra destacada y suscritas por ambas partes. Deben redactarse tantos ejemplares como partes integren la relación contractual y suscribirse a un solo efecto. Un ejemplar original debe ser entregado al consumidor. La reglamentación establecerá modalidades más simples cuando la índole del bien objeto de la contratación así lo determine, siempre que asegure la finalidad perseguida en esta ley.

- Ley 26.361 - Artículo 9: *Garantías.*

Cuando se comercialicen cosas muebles no consumibles conforme lo establece el artículo 2325 del Código Civil, el consumidor y los sucesivos adquirentes gozarán de garantía legal por los defectos o vicios de cualquier índole, aunque hayan sido ostensibles o manifiestos al tiempo del contrato, cuando afecten la identidad entre lo ofrecido y lo entregado, o su correcto funcionamiento. La garantía legal tendrá vigencia por TRES (3) meses cuando se trate de bienes muebles usados y por SEIS (6) meses en los demás casos a partir de la entrega, pudiendo las partes convenir un plazo mayor. En caso de que la cosa deba trasladarse a fábrica o taller habilitado el transporte será realizado por el responsable de la garantía, y serán a su cargo los gastos de flete y seguros y cualquier otro que deba realizarse para la ejecución del mismo.

- Ley 24.240 - Artículo 12: *Servicio Técnico.*

  Los fabricantes, importadores y vendedores de las cosas mencionadas en el artículo anterior, deben asegurar un servicio técnico adecuado y el suministro de partes y repuestos.

- Ley 24.999 - Artículo 3: *Certificado de Garantía*.

El certificado de garantía deberá constar por escrito en idioma nacional, con redacción de fácil comprensión en letra legible, y contendrá como mínimo:

a) La identificación del vendedor, fabricante, importador o distribuidor;
b) La identificación de la cosa con las especificaciones técnicas necesarias para su correcta individualización;
c) Las condiciones de uso, instalación y mantenimiento necesarias para su funcionamiento;
d) Las condiciones de validez de la garantía y su plazo de extensión;
e) Las condiciones de reparación de la cosa con especificación del lugar donde se hará efectiva.

El aparato debe ser entregado con ciertas pruebas realizadas, asegurando las condiciones y circunstancias bajo las cuales se garantiza que el producto no falla. Por esto es que es necesario aclarar que nuestra responsabilidad radica exclusivamente sobre los posibles daños causados a partir de una falla eléctrica del aparato y sus consecuencias inmediatas, o sobre alguna falla producida bajo las condiciones estrictamente definidas en las pruebas de funcionamiento. Cualquier daño causado a partir del incorrecto registro de los datos o de su mala interpretación es absoluta responsabilidad del médico, quien debe garantizar la coherencia de los mismos.

5. INGENIERÍA DE DETALLE

5.1 HARD

5.1.1 Diagrama de bloques del hard

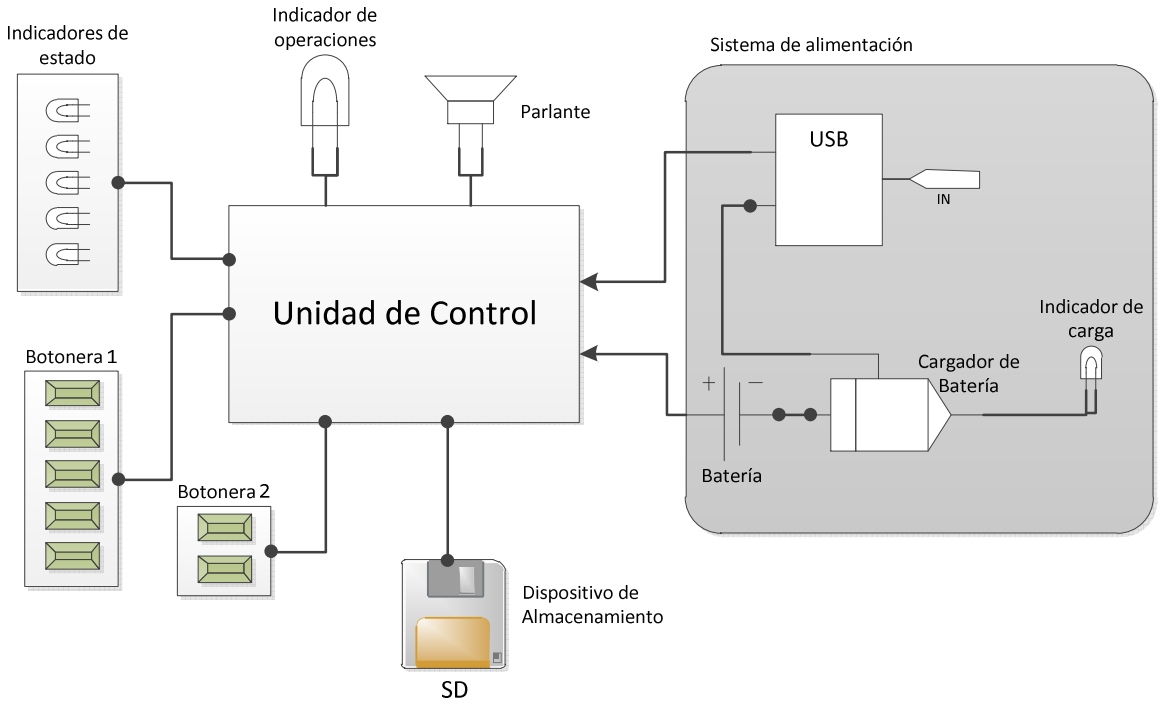


Gráfico 5. *Diagrama de bloques del hard.*

5.1.2 Descripción detallada de cada bloque

Unidad de Control

Como unidad de control del dispositivo se utiliza el Microprocesador de Motorola *MCF51JM128*. Este bloque es el que define todas las acciones realizadas por el aparato, y responde a los estímulos externos recibidos. En primer lugar, está pendiente constantemente del estado de las botoneras. En caso de detectar algún tipo de acción, ejecuta la reacción correspondiente, guardando en el dispositivo de almacenamiento el evento ocurrido e informando la recepción del dato mediante los indicadores de estado, el indicador de operaciones y/o el parlante, según corresponda. Este bloque se encarga de revisar los eventos programados y anunciarlos debidamente, señalizándolos mediante los indicadores ya mencionados. Estos eventos programados pueden ser el horario de toma de medicación o la necesidad de refrescar el estado del paciente. Por último, también es el microprocesador quien se encarga de revisar la carga de la batería, ejecutando el resguardo de los datos antes de que el dispositivo quede sin energía. Todo esto es establecido por el software cargado en la memoria no volátil del microprocesador. Físicamente, al microprocesador lo acompañan otros componentes, de los cuales se detalla su procedencia y sus valores en la próxima sección.

Indicadores de estado

Los indicadores de estado son cinco (5) leds que serán encendidos, desde 1 hasta los 5, indicando el estado de movilidad del paciente. Un (1) led encendido representa que el paciente está en un 20% de su capacidad motora. Dos (2) leds contiguos encendidos representarán el 40 % de su capacidad motora. Tres (3) leds, 60%. Cuatro (4) leds, 80%. Cinco (5) leds, 100%. Su funcionamiento es análogo al de un potenciómetro, dado que las luces se irán encendiendo contiguamente, formando una línea más larga o más corta, indicando mejor o peor estado, respectivamente.

Botoneras

El dispositivo consta de dos botoneras. La primera, con cinco (5) botones, los cuales corresponden cada uno a un led de los indicadores de estado. El paciente presionará el botón que corresponda a lo que considera es su estado en ese momento. La segunda botonera consiste de dos (2) botones, uno para indicar si el paciente tiene disquinesias o no, y el otro para confirmar la recepción de algún dato, como podría ser la alarma que indique el momento de toma de la medicación. Cada uno de estos botones están configurados de tal manera que es necesario presionar el botón durante dos segundos para que este considere la intención de hacerlo. Esto es para evitar el registro de eventos indeseados o el toque accidental de los botones.

Indicador de operaciones

Este indicador es una señal lumínica (led) cuya función es indicar si es posible ingresar un dato o no. Si el led está encendido, cualquier botón presionado será registrado. Si el led está apagado es porque ha ocurrido un evento recientemente y por lo tanto se está esperando un tiempo para volver a habilitar el registro de eventos.

Parlante

El Parlante funciona como un indicador más, utilizado principalmente para avisar la hora de la toma de la medicación y el momento de refrescar el estado del paciente.

Dispositivo de almacenamiento

El dispositivo de almacenamiento externo que se utiliza es una tarjeta de memoria tipo SD (*Secure Digital*). De esta forma, cuando se quieran revisar los datos, es tan simple como extraer esta tarjeta e insertarla en una computadora. Cada evento ocurrido será registrado en la misma bajo un formato predeterminado.

Sistema de alimentación

El sistema de alimentación del dispositivo consiste fundamentalmente en una batería de ion-litio de 3,3 Voltios. La misma es la que mantiene al dispositivo funcionando cuando no se encuentra enchufado a la corriente eléctrica. En el momento que la carga de la batería esté baja, un led dedicado para esto lo indicará, y se guardarán los datos en el dispositivo de almacenamiento. Cuando el aparato es enchufado por puerto mini USB (5 Voltios) un sistema de diodos desconecta a la batería del mismo y pasa a ser alimentado por la corriente que recibe de la corriente que entra por el USB. Al mismo tiempo, se activa el CMC (*Charge management controller*), elemento cuya función es regular la carga de la batería. Es decir, cargar la batería

con la corriente ingresada por USB, si es que está descargada, y simplemente no hacer nada, en caso de que la batería ya esté cargada.

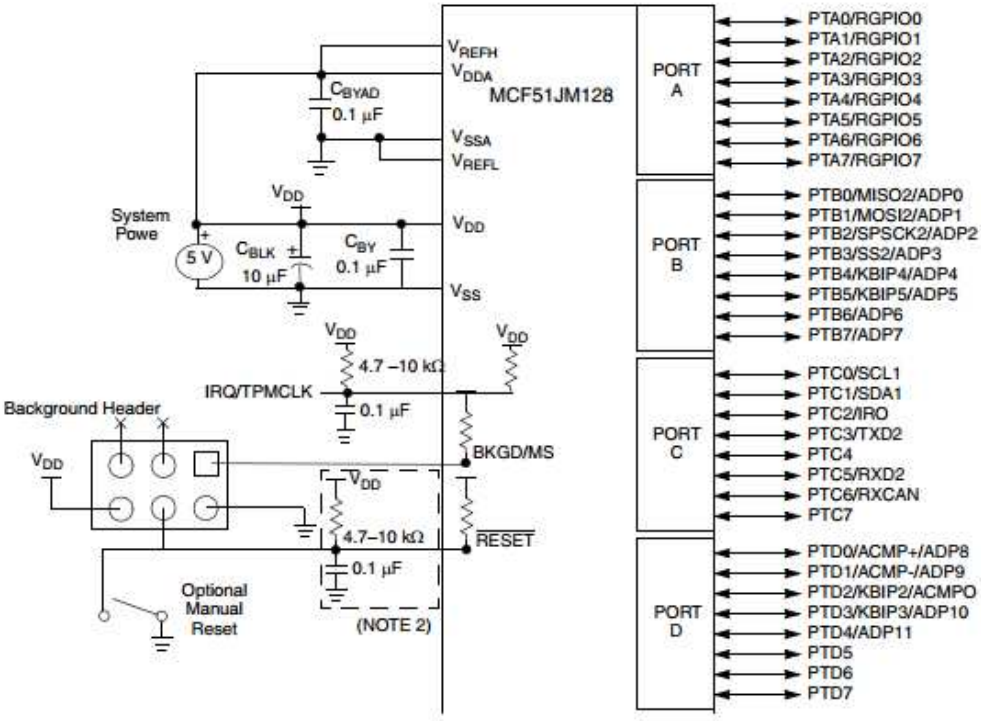
5.1.3 Detalles de selección y cálculo de los elementos circuitales de cada bloque

Unidad de Control

Como Unidad de Control del dispositivo se utilizó el microprocesador MCF51JM128, fabricado por *Freescale*. Con sus muchas características, fueron unas pocas específicas que nos ayudaron a determinar que este sería el indicado, a saber:

- Encapsulado pequeño, optimizando espacio y consumo.
- Gran número de puertos de salida, disponibles para las botoneras y los indicadores luminosos.
- Programación mediante puerto USB.
- Conocimiento previo sobre los procesadores de *Freescale*, optimizando el tiempo de aprendizaje.

De la misma hoja de datos del microprocesador se extrae el siguiente diagrama, el cual representa el conexionado y los componentes a colocar.



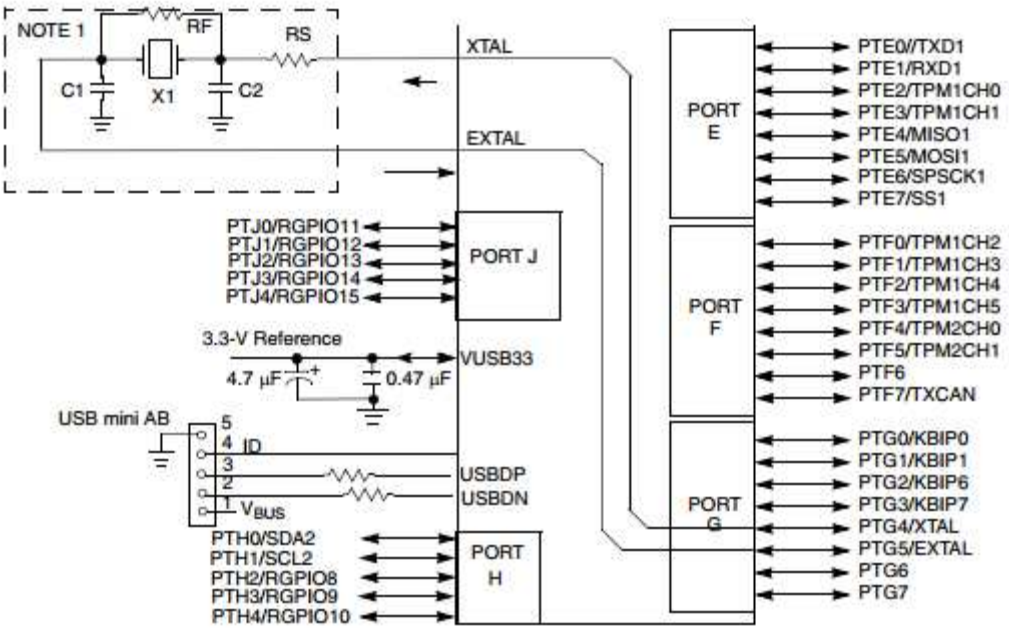


Gráfico 6. *Diagrama de esquemático del microprocesador.*

Indicadores de estado

Los indicadores de estado son implementados por leds, colocados linealmente simulando un potenciómetro digital, en el que la cantidad de luces prendidas representarán una línea más larga o más corta en función del porcentaje que se quiera representar.

Cada led está conectado con su terminal positivo a un puerto de salida del microprocesador y con su terminal negativo a una resistencia de *pull down*, que funciona como limitador de corriente. La resistencia colocada se calculó a partir de la corriente que se espera que pase por el led, con la siguiente fórmula:

$$R = \frac{V_{dd} - V_{led}}{I} = \frac{5V - 0{,}7V}{15mA} = 286\Omega$$

Para colocar la resistencia, se decidió tomar un margen de seguridad, y por lo tanto se optó por el siguiente valor comercial disponible, es decir **330Ω**.

Botoneras

La botonera principal se corresponde con cada uno de los leds del indicador de estado. Cada botón está conectado directamente por un lado a un puerto del microprocesador y por el otro a una resistencia de *pull down*. Para obtener el valor de la resistencia se procedió de la misma forma que para el indicador de estado, simplemente que no existe una caída de tensión considerable en el botón, y por lo tanto queda:

$$R = \frac{V_{dd}}{I} = \frac{5V}{15mA} = 333\Omega$$

Al igual que para los leds, se tomó un margen de seguridad, y se coloca una resistencia de un valor mayor, para este caso, **470Ω**.

Indicador de operaciones

El indicador de operaciones funciona del mismo modo que el indicador de estado pero, en este caso, con un solo led. La conexión y el cálculo de la resistencia es exactamente igual.

Parlante

El parlante es utilizado para el aviso de alarmas y fue implementado con un buzzer, el cual consiste de un electroimán y una lámina metálica de acero, donde cada pieza se corresponde a un terminal. Cuando se acciona, la corriente pasa por la bobina del electroimán y produce un campo magnético variable que hace vibrar la lámina de acero sobre la armadura. Para su conexión debe tener en cuenta que su resistencia es de tan solo 18 Ω, se decide poner una resistencia de 220Ω en serie para llevar su conexión a Vdd. Para satisfacer la demanda de corriente, se conecta el terminal negativo del buzzer al colector de un transistor bipolar cuya base se conecta al microprocesador con una resistencia en serie de 3,7 kΩ, y su emisor a masa.

Dispositivo de almacenamiento

La tarjeta de memoria se monta sobre la placa con un zócalo adecuado para la misma con un sistema de resorte muy similar al de la computadora tradicional. Los terminales irán conectados a los terminales correspondientes del microprocesador que se utilizan para realizar la transmisión de datos vía *SPI* (Serial Peripheral Interface). En el anexo se adjunta la hoja de datos correspondiente que especifica el modo en que deben hacerse las conexiones.

Sistema de alimentación

El sistema de alimentación se compone de tres elementos fundamentales: la batería, el controlador de carga y la entrada de alimentación externa vía mini USB.

La batería es de ion-litio de 3,3 Voltios, de 1 Ampere-hora de capacidad. Se conecta su terminal positivo a la salida correspondiente del controlador de carga. Esto permitirá que, en caso de ser necesario, la batería se cargue mediante el puerto mini USB. La misma se conecta a masa a través de dos capacitores de desacople, uno electrolítico de 10 µF y otro multicapa de 100 nF, por recomendación del fabricante.

El controlador de carga es el elemento MCP73861, del fabricante *Microchip*. Se conecta a masa a través de capacitores que funcionan como desacople, y también un divisor resistivo, por recomendaciones del fabricante, como se muestra en el diagrama siguiente.

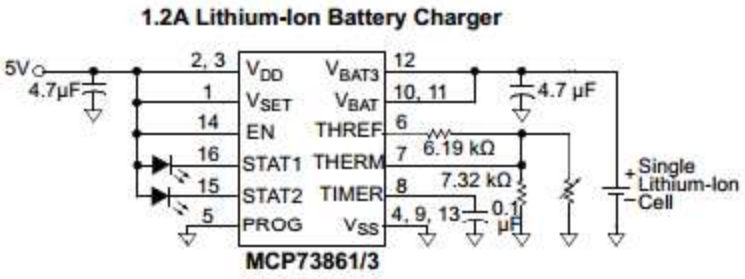


Gráfico 7. *Diagrama esquemático del Controlador de Carga.*

5.1.4 *Plan de pruebas de cada módulo*

Unidad de Control

Para evaluar el microprocesador, se hizo un análisis sobre cuáles son las funciones y los terminales utilizados del mismo, y se realizaron sucesivas pruebas que verifiquen el correcto funcionamiento del mismo.

Para cada uno de los puertos de salida utilizados se hizo un pequeño programa de prueba que haga titilar un led, verificando el funcionamiento eléctrico del *pin* del microprocesador y el funcionamiento correcto del *timer*, ya que se espera que el led titile a una frecuencia determinada.

Botoneras

Las botoneras necesitan la verificación de la continuidad de la corriente cuando está presionado el botón, y el valor de 0 Voltios cuando no está presionado. Esto se verifica con un *tester* común, evaluando la tensión entre el terminal que se conecta al micro y la tierra.

Indicadores de estado e indicador de operaciones

Para verificar el correcto funcionamiento de los leds, se arma el circuito con su correspondiente resistencia y se verifica que la corriente sea la esperada, midiéndolo con un tester. Si al colocar la tensión de alimentación el led tiene la intensidad lumínica y la corriente esperada, entonces estará corroborado su funcionamiento.

Parlante

Para probar este módulo, es necesario armar el circuito en *protoboard*, con el diodo y las resistencias correspondientes, y verificar, en primer lugar, que emita el sonido correspondiente, conectándolo a una fuente que limite la corriente. Luego, debe comprobarse que la corriente que demande no sea mayor a la que el microprocesador pueda entregar. Si con la corriente esperada el sonido es el adecuado, entonces se habrá comprobado su correcto funcionamiento.

Dispositivo de almacenamiento

Este módulo debe ser probado en conjunto con el procesador. En primer lugar, se puede verificar que la tarjeta de memoria funcione colocándola en una computadora común. Luego, debe realizarse un simple programa de lectura y escritura a través del microprocesador para verificar que la comunicación entre la tarjeta y el procesador sea la adecuada.

Sistema de alimentación

Tal como se mencionó previamente, este es el sistema de mayor complejidad eléctrica, por lo cual sería conveniente realizar un pequeño prototipo que involucre simplemente este sistema. Una vez armado, es importante medir la actividad eléctrica para los diversos casos posibles, a saber: salida de tensión en caso de no haber alimentación externa, para lo cual tiene que estar entregando corriente la batería; salida de tensión para el caso de alimentación externa, para lo cual la batería no debe estar entregando corriente y sí la entrada externa; debe comprobarse también que se cargue la batería en caso de estar descargada, cuando hay alimentación externa, y comprobarse que no se le entregue más corriente si ya está cargada. Si la actividad eléctrica es correcta en estos casos, entonces el funcionamiento será el correcto.

5.2.1. Diagrama de estados y flujo gramas

El flujo del software implementado en el microprocesador es el siguiente:

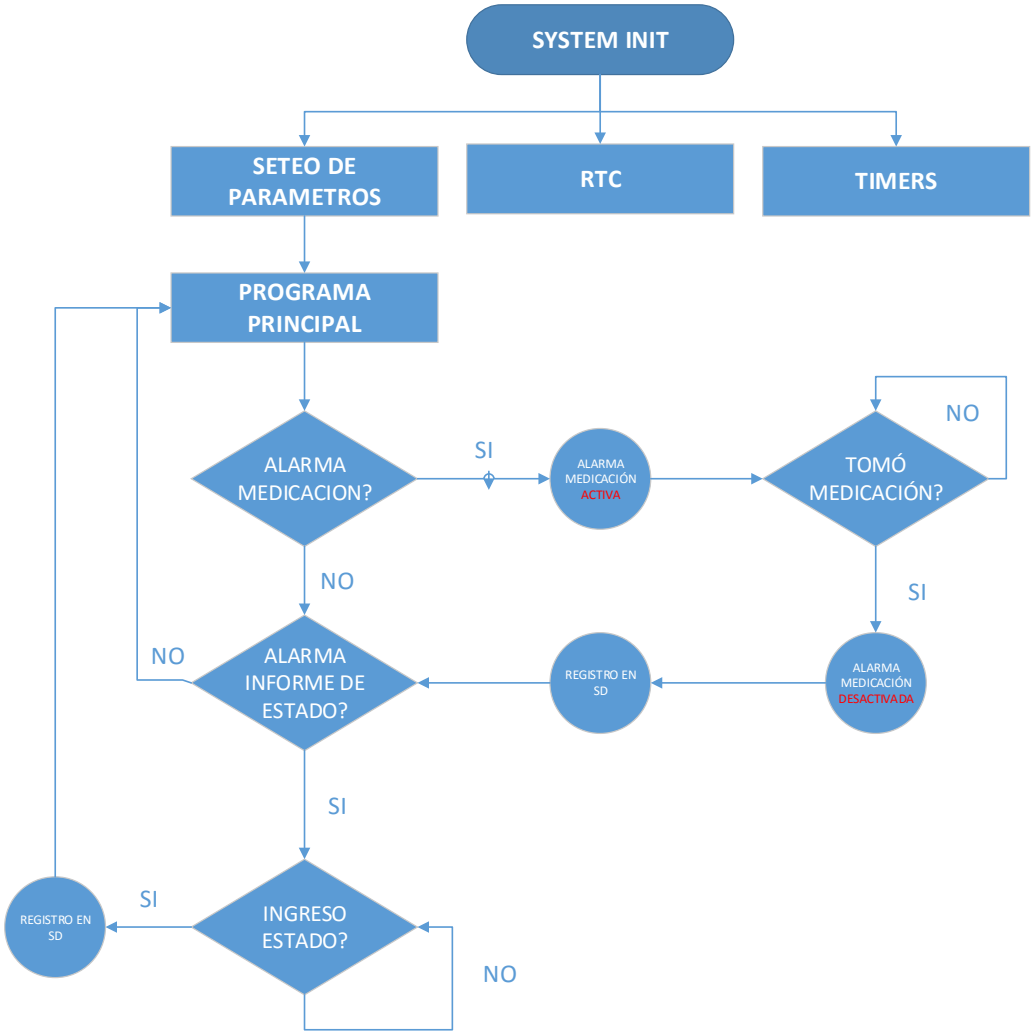


Fig. 5.2.1 – Flujo del software del proyecto

La primera vez que se encienda el equipo se va a llevar a cabo el “seteo inicial” del mismo, de acuerdo con los parámetros elegidos por el médico que realizó las configuraciones a través de la aplicación de JAVA utilizando la PC. En él se realizará el seteo de alarmas, nombre del paciente, historia clínica, etc. En breve, será explicado en detalle dicha configuración.

Una vez configurado todos los parámetros necesarios, inicia el flujo principal. Lo primero, corresponde a verificar si se cumplió el tiempo en que el paciente debe tomar su medicación (Verify_Alarm).

Hay cinco alarmas posibles. De haberse cumplido, se activará la alarma sonora/lumínica correspondiente, y se esperará a que el paciente ingrese mediante el botón de “MEDICACION”, que efectuó la toma de la misma y se registrará el suceso en la tarjeta SD. De no realizarse lo anterior, el dispositivo seguirá sumergido en el bucle alarma hasta que el paciente haga la toma de la medicación y la asiente.

A continuación, el programa pregunta si se llegó al tiempo indicado para hacer un registro de “ESTADO”. Si así fuera, se activará la alarma correspondiente para el ingreso del estado que el paciente desee. De no producirse el ingreso de un estado, el programa quedará en el bucle alarma. Una vez indicado el estado, se registrará el mismo en la SD y el programa podrá volver a su estado inicial.

Paralelamente al programa principal hay dos servicios más corriendo, RTC(Real Time Clock) y TIMERS. El primero maneja el horario y el calendario. El segundo setea timers necesarios para ejecutar la lectura del teclado, y retardos usados en el programa.

Seguido se hará una explicación en detalle de los procesos:

- *SYSTEM INIT:*

Es en esta sección donde se inicializan los servicios como el RTC, se definen los puertos del microprocesador que manejaran los distintos periféricos, se setean las interrupciones y se configura el uso de la SD.

```
system_init();// Se copia la tabla de interrupciones a la RAM. Puedo inicializar aca las interrupciones.

/* !! Start application code below here !! */
//------------------------------------------------------------//
SOPT1 = 0x00;           // disable COP
SOPT2 = 0x00;

RTC_Init();
hw_init();              /* MCU Initialization */

u8Error=SD_Init();      /* SD Card Init */
SPI_Init();
SCSI_MemoryInfo();
```

Fig. 5.2.2 – Inicialización de servicios

Luego, si es que es una nueva tarjeta SD o la misma se encuentra formateada, se realiza la creación del archivo .txt con el formato adecuado al programa:

```
/*******************************************************/
/*******CREO ARCHIVO LOG.TXT EN LA SD***************/
FAT_Read_Master_Block();                /* FAT Driver Init */

/* Creating the file if it doesn't exist */
if (FAT_FileOpen(FILE_NAME, MODIFY) == FILE_NOT_FOUND)
{
  u8Error=FAT_FileOpen(FILE_NAME,CREATE);
  FAT_FileClose();

  Print_Welcome(); //Imprime Titulo de Bienvenida
  Print_Name ();   //Imprime Nombre y alarmas de base
}
FAT_FileClose();
/*******************************************************/
```

Fig. 5.2.3 – Creación del archivo .txt y formato

- *SETEO DE PARAMETROS:*

En el caso del primer arranque del dispositivo y al encontrase la tarjeta SD formateada, se creará un archivo LOG.txt dentro de ella. En dicho archivo se generarán el siguiente header:

```
/*************************************************************/
/************ DATA LOGGER - 2013 - ITBA-FLENI ************/
/*************************************************************/
/*NOMBRE: Juan Perez                                         */
/*-----------------------------------------------------------*/
/*Fecha de Nacimiento: 1970              HC: 1234            */
/*-----------------------------------------------------------*/
/*FECHA DE INICIO: 10/12/2013            HORA: 00:00:00      */
/*-----------------------------------------------------------*/
/*ALARMAS:   #1   |   #2   |   #3   |   #4   |   #5   |      */
/*         000030| 000230| 000OFF| 000OFF| 000OFF|         */
/*-----------------------------------------------------------*/
/*Intervalo de Registros:  001 minutos                       */
/*************************************************************/
```

Fig. 5.2.4 – Parámetros seteados en la SD

Es en este archivo .txt donde se van a ir loggeando los diferentes sucesos con hora y fecha de aparición. Como se puede observar en la figura, hay variables que se deben configurar para que luego el dispositivo funcione adecuadamente. Todas estas configuraciones son realizadas mediante el ejecutable en JAVA, DLEP.

Primeramente se configuran los datos personales del paciente; Nombre, Fecha de nacimiento e Historia clínica (HC). Luego se debe setear la fecha de inicio, la cual se corresponde con el día que se comienza con el estudio del paciente y se le da el dispositivo para empezar con ello. Lo mismo para la configuración de la hora. Como este dispositivo no presenta un display visible, se debe realizar de este modo.

A continuación, existe la configuración de 5 alarmas posibles. Para lo cual se dispone de seis caracteres, de los cuales los dos primeros son para la “hora”, los dos segundos para los “minutos” y los restantes para los “segundos”. En este ejemplo, la alarma número uno se iniciará las 00:00:30 hs. En caso que se setee alguna alarma como en el caso número tres, dicha alarma estará apagada (OFF).

Ya con todos estos datos disponibles se comienza con la iniciación de los timers, hora, calendario, alarmas e intervalo de informe de estado:

```
//---------Habilito TIMER TPM1-----------------------//
start_mS_timer(10);
Init_Timers();
//---------------------------------------------------//
Init_Alarms();
Read_Alarms();//Leo las Alarmas a setear y las activo
Read_Date (); //Leo la hora y la fecha en la SD --> y lo modifico en mi RTC
Read_State_Interval(); // Leo intervalo de informe de estado y lo seted
```

Fig. 5.2.5 – Seteos iniciales

5.*2.2. Análisis de complejidad*

Para medir la complejidad del software del proyecto, se decidió medir la complejidad ciclomática del mismo. Esta métrica del software proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje.

Esta métrica, propuesta por Thomas McCabe en 1976, se basa en el diagrama de flujo determinado por las estructuras de control de un determinado código. Dicha métrica, permite calcular la complejidad ciclomática, según la siguiente fórmula:

$$\boldsymbol{M = E - N + 2P}$$

Siendo:

Letra	Significado
M	Complejidad ciclomática a calcular
E	Cantidad de conexiones del diagrama de estado
N	Cantidad de nodos
P	Cantidad de procesos independientes

Entonces:

$$\boldsymbol{M = 13 - 11 + 2 * 3 = 8}$$

Riesgo bajo-moderado.

5.2.3. Descripción de subrutinas

Driver SD	
UINT8 SD_Init(void)	
Descripción	**Función encargada de abrir la comunicación con la tarjeta de memoria SD.**
Parámetros	**Ninguno**
Retorno	**Resultado de la operación**
UINT8 SD_Write_Block(UINT32 u16SD_Block,UINT8 *pu8DataPointer)	
Descripción	**Función encargada de escribir un bloque en la memoria SD**
Parámetros	**El bloque de la memoria que se va a escribir, un puntero a los datos que se quieren escribir**
Retorno	**Resultado de la operación**
UINT8 SD_Read_Block(UINT32 u16SD_Block,UINT8 *pu8DataPointer)	
Descripción	**Función encargada de leer un bloque de memoria de la SD**
Parámetros	**El bloque de la memoria que se va a leer, un puntero a donde se copiará el bloque leído**
Retorno	**Resultado de la operación**
UINT8 SD_SendCommand(UINT8 u8SDCommand,UINT8 u8SDResponse)	
Descripción	**Función encargada de enviar los comandos necesarios para ejecutar la SD. En SD.h se encuentra la descripción de los mismos**
Parámetros	**Código del comando, respuesta esperada (ej. OK)**
Retorno	**Resultado de la operación**

Driver FAT	
void FAT_FileClose(void);	
Descripción	**Función encargada de cerrar un archivo que se está editando de la memoria SD.**
Parámetros	**Ninguno**
Retorno	**Ninguno**
UINT8 FAT_FileOpen(UINT8*,UINT8);	
Descripción	**Función encargada de abrir un archivo de la memoria SD**
Parámetros	**El nombre del archivo, acción(ej: READ-DELETE-OVERWRITE)**
Retorno	**Resultado de la operación**
void FAT_FileWrite(UINT8*,UINT32);	
Descripción	**Función encargada de escribir un archivo de la memoria SD**
Parámetros	**Un puntero a lo que se quiere escribir, el tamaño de dicho contenido**
Retorno	**Ninguno**
UINT16 FAT_FileRead(UINT8*);	
Descripción	**Función encargada de leer un archivo de la memoria SD**
Parámetros	**Un puntero al bloque que se va leyendo y almacenando**
Retorno	**Como se lee en bloque de 512, devuelve este valor cuando el buffer se encuentra lleno.**

Driver SPI	
void SPI_init(void);	
Descripción	**Función encargada de abrir la comunicación e inicializar los parámetros de SPI. Dichos parámetros son: Modo Lectura/Escritura; Cantidad de bits por palabra; Velocidad máxima.**
Parámetros	**Ninguno**
Retorno	**Ninguno**
UINT8 SPI_Receive_byte(void);	
Descripción	**Función encargada de recibir un byte por comunicación SPI**
Parámetros	**Ninguno**
Retorno	**Resultado de la operación**
void SPI_Send_byte(UINT8);	
Descripción	**Función encargada de enviar un byte por comunicación SPI**
Parámetros	**Byte a enviar**
Retorno	**Ninguno**

Driver Teclado	
void Leer_Tecla (void);	
Descripción	**Función encargada de leer el teclado**
Parámetros	**Ninguno**
Retorno	**Ninguno**
void Delay_Tecla (void);	
Descripción	**Función encargada de generar un retardo para no leer repetidamente la misma tecla**
Parámetros	**Ninguno**
Retorno	**Ninguno**
void Destello_potenciometro (char leds);	
Descripción	**Función encargada de hacer blinkear los leds del dispositivo**
Parámetros	**Led/s a hacer destellar**
Retorno	**Ninguno**
void Validate_State (char leds);	
Descripción	**Función encargada habilitar el teclado pasado un tiempo prudencial, y evitando imprimir en la SD más de un informe.**
Parámetros	**Led/s a hacer destellar**
Retorno	**Resultado de la operación**

Driver Alarma

void Init_Alarms(void);	
Descripción	**Función encargada inicializar las alarmas**
Parámetros	**Ninguno**
Retorno	**Ninguno**
void set_Alarm(UINT8 hora_Alm,UINT8 min_Alm,UINT8 seg_Alm,UINT8 index);	
Descripción	**Función encargada de setear la alarma correspondiente**
Parámetros	**Hora, minutos, segundos, número de alarma correspondiente**
Retorno	**Ninguno**
UINT8 Verify_Alarm (UINT8 index);	
Descripción	**Función encargada de verificar si se cumplió el tiempo de la alarma index**
Parámetros	**Alarma correspondiente**
Retorno	**Resultado de la operación**
void Encender_Alarma (UINT16 mseg);	
Descripción	**Función encargada encender la alarma según sea el caso**
Parámetros	**Tipo de alarma requerida**
Retorno	**Ninguno**
void Verify_Inform(UINT16 counter);	
Descripción	**Función encargada verificar si es necesario hacer un informe de estado**
Parámetros	**Tiempo seteado como parámetro de informe**
Retorno	**Ninguno**

Driver RTC	
void RTC_Init(void);	
Descripción	**Función encargada de inicializar el RTC**
Parámetros	**Ninguno**
Retorno	**Ninguno**
interrupt void RTC_ISR(void);	
Descripción	**Función encargada de generar la interrupción para actualizar el RTC, calendario y horario.**
Parámetros	**Ninguno**
Retorno	**Ninguno**
void OSUpdateTime(OSTime *Ptr_Hora);	
Descripción	**Función encargada de actualizar la hora real del dispositivo**
Parámetros	**Puntero a la clase Hora**
Retorno	**Ninguno**
void OSUpdateCalendar(void);	
Descripción	**Función encargada de actualizar el día real del dispositivo**
Parámetros	**Ninguno**
Retorno	**Ninguno**

5.2.5. Plan de prueba de módulos y de depuración de soft

Luego de haber realizado el soft implementado en el microprocesador, la depuración del mismo consistió en impresiones en la SD. Ya que este dispositivo no tiene una pantalla física donde probar cada módulo, la solución fue ir imprimiendo en la tarjeta de memoria cada medición realizada.

La **Fig. 5.2.4**, muestra el primer seteo básico utilizado para dicha depuración. Donde se puede ver que se pide un intervalo de registro de informe cada 1 minuto, pudiendo así recolectar en poco tiempo gran cantidad de datos necesarios para la ejecución de los gráficos. Así también las alarmas fueron seteadas al poco tiempo de encendido el aparato.

```
/*************************************************************/
/************ DATA LOGGER - 2012 - ITBA-FLENI ************/
/*************************************************************/
/*NOMBRE: Juan Perez                                         */
/*-----------------------------------------------------------*/
/*Fecha de Nacimiento: 1970              HC: 1234            */
/*-----------------------------------------------------------*/
/*FECHA DE INICIO: 10/12/2012            HORA: 00:00:00      */
/*-----------------------------------------------------------*/
/*ALARMAS:   #1   |   #2   |   #3   |   #4   |   #5   |      */
/*         000030| 000230| 000OFF| 000OFF| 000OFF|          */
/*-----------------------------------------------------------*/
/*Intervalo de Registros:  001 minutos                       */
/*************************************************************/
10/12/2012_00:00:32_MEDICACION
10/12/2012_00:01:05_ON_075%
```

Como puede observarse en esta última figura, el primer registro de la SD corresponde con la alarma de medicación seteada para los 30 segundos de iniciado el encendido. Y una vez que se tomó la medicación, se informó presionando el botón correspondiente.

El segundo informe se corresponde con un registro de estado, el cual había sido configurado para hacerlo cada 1 minuto.

Básicamente, se jugó con la edición de los datos de este header dispuestos en la tarjeta SD, y se buscó obtener los informes esperados para la correcta construcción de los gráficos buscados.

Principalmente es un dispositivo que va a ser utilizado por un paciente enfermo, con lo cual siempre la prioridad fue hacer que resulte sencillo de utilizar y siempre bien fácil de interpretar.

6.1 DEFINICIÓN DE LOS MÓDULOS

- **Unidad de control**: Consiste en el microprocesador con todos los componentes que garantizan su funcionamiento eléctrico, como los capacitores de desacople.

- **Indicadores de estado**: Compuesto por 5 leds colocados linealmente simulando un potenciómetro.

- **Botoneras**: Compuesto por 7 botones, 5 para indicar el estado (0%, 25%, 50%, 75% y 100%), 1 para indicar disquinesias y 1 para confirmar la acción realizada.

- **Indicador de operaciones**: Consiste en un led que está encendido cuando está disponible para realizar acciones o apagado indicando que no se puede realizar ninguna acción.

- **Dispositivo de almacenamiento**: Es una tarjeta de memoria tipo SD con su respectivo zócalo.

- **Sistema de alimentación**: Se compone de una batería, un controlador de carga, un sistema de diodos que controla un relé que determina si la alimentación del dispositivo es externa o desde la batería, y un led que encenderá cuando la carga de la batería esté próxima a agotarse.

6.2 DISEÑO DE LOS CIRCUITOS IMPRESOS

El diseño fue realizado con el programa *Altium*, con el cual se obtienen los circuitos impresos a partir de la confección del esquemático. A continuación se presenta el diagrama final de la placa.

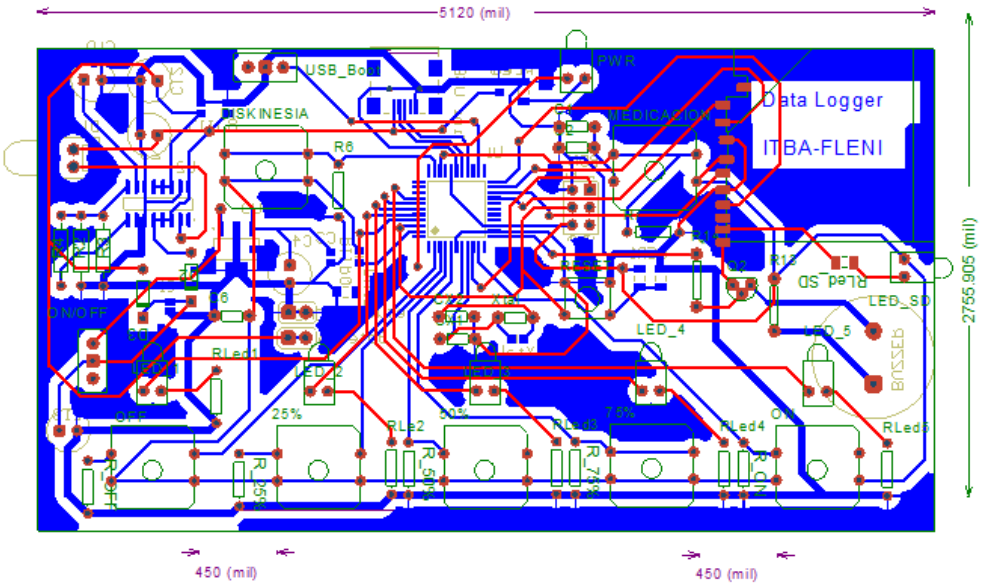


Gráfico 8. *Circuito impreso de la placa.*

6.3 DISEÑO MECÁNICO

Para realizar el diseño mecánico se pensó principalmente en las dimensiones y el peso deseados. Para lograrlo, teniendo en cuenta que el peso de la placa ya está previamente definido por los componentes que la conforman, se procedió a seleccionar el elemento mecánico faltante: el gabinete. Este se seleccionó de un material liviano y resistente como es el PVC. Para definir el tamaño, se hicieron la placa y el gabinete de forma tal que se cumpla con las medidas de las especificaciones previamente definidas.

A continuación se puede ver el diagrama del gabinete básico, sin los orificios correspondientes, con las medidas originales [milímetros].

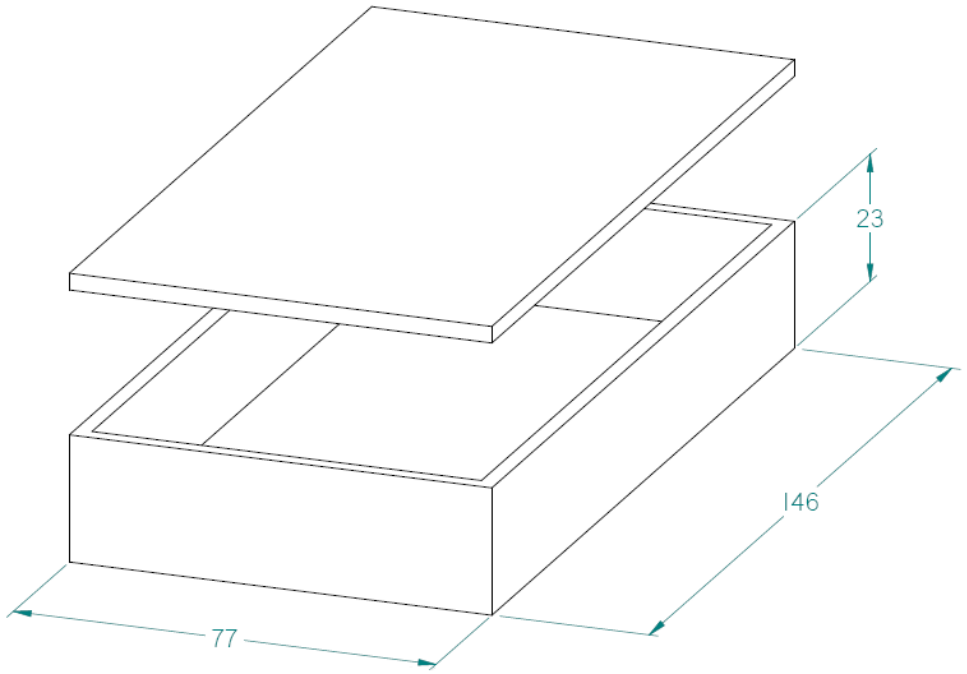


Gráfico 9. *Boceto del gabinete sin modificar. Medidas en milímetros.*

6.4 DETALLES DE CONSTRUCCIÓN Y PRECAUCIONES ESPECIALES DE MONTAJE

Como ya se mencionó en el punto anterior, el gabinete es precisamente del tamaño de la placa, y la placa es encastrada sobre este en forma ajustada, evitando desplazamientos al moverse el dispositivo. Luego, es necesario hacer los orificios correspondientes para permitir la conexión USB para realizar la carga de la batería, y para que se localicen los botones y los indicadores lumínicos.

Finalmente, el gabinete con todas las modificaciones realizadas quedará como se muestra en el esquema expuesto a continuación.

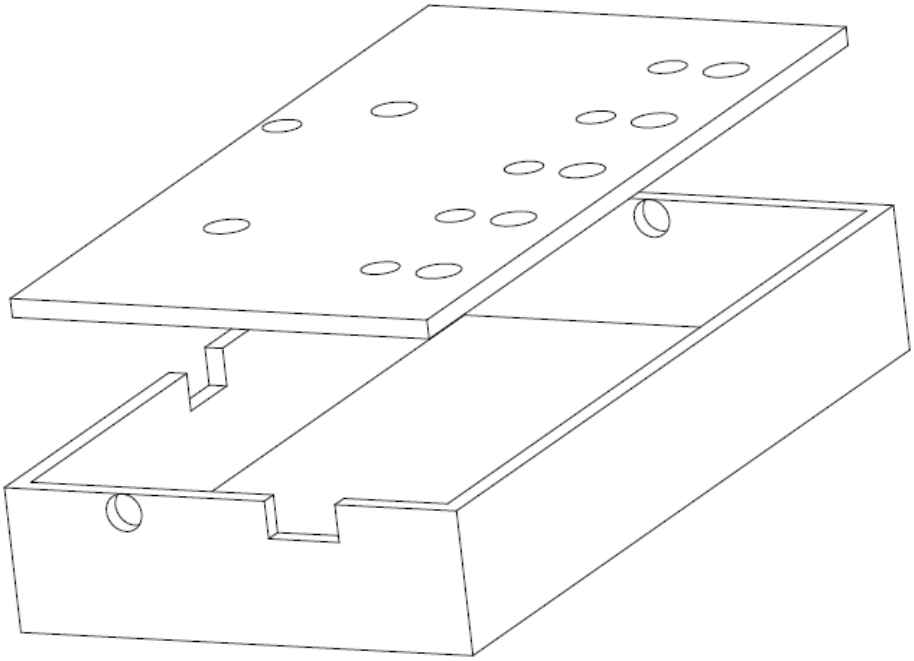


Gráfico 10. *Boceto del gabinete con las modificaciones correspondientes.*

7.1 VALIDACIÓN DE HARD

7.1.1 *Plan y Protocolos especiales de medición*

Con el objetivo de comprobar el correcto funcionamiento del dispositivo se procedió a realizar una serie de mediciones que evalúen el rendimiento eléctrico y el funcionamiento del software, y poder establecer que el prototipo está listo para usarse. A continuación se describirá paso a paso el plan de medición establecido. En los primeros cuatro pasos se describen los procedimientos necesarios para revisar el funcionamiento estrictamente eléctrico. A partir del quinto paso, se evalúa el funcionamiento integral, para lo cual es necesario que esté validado el funcionamiento del *software*, previamente.

1. **Alimentación con batería**: Debe encenderse el dispositivo con el *switch* indicado como *ON/OFF*. Si el led LED0 enciende, entonces este es indicador de que el dispositivo ha encendido correctamente.
2. **Alimentación externa sin batería**: Es necesario quitar la batería del dispositivo y enchufarlo por su puerto *mini USB*. Si LED0 enciende igual que antes, entonces la alimentación externa.
3. **Carga de la batería**: Para comprobar si el cargador de la batería funciona, es necesario realizar la prueba, en primer lugar, colocando la batería completamente descargada y verificando que el dispositivo no enciende. Luego, debe enchufarse el dispositivo y dejar cargando unos treinta minutos. Si luego de este tiempo, el dispositivo funciona desenchufado, entonces la batería se habrá cargado exitosamente.
4. **Consumo de corriente**: Para que el dispositivo tenga la autonomía deseada es necesario que el consumo del mismo no exceda los valores estimados. Esto se puede comprobar haciendo funcionar el dispositivo con alimentación externa, y más precisamente con una fuente de tensión de laboratorio, con la cual se pueda limitar y medir la entrega de corriente eléctrica realizada por la misma. Luego de esto, pueden realizarse algunas operaciones típicas con el dispositivo y comprobar que el consumo se mantenga por debajo de lo estimado. Es importante aclarar que este paso está supeditado al correcto funcionamiento del *software*, ya que si se comprobara que este no funciona correctamente en alguno de los siguientes pasos, este paso debería repetirse, ya que podría haber una influencia directa en el consumo una modificación de este tipo.
5. **Respuesta a estímulos**: Se comprobará el funcionamiento de todos los botones realizando una pequeña prueba en la cual se haga utilización de los mismos. Para cada uno de los estados posibles se hará una opción de ingresado. Los pasos y reacciones serían, en orden: presionar botón de estado o de disquinesias, por lo cual debe escucharse un pitido y encenderse el led contiguo; presionar el botón de confirmación de la operación, por lo cual los leds titilarán y el indicador de estado se apagará por unos segundos; luego de esos segundos el indicador de operaciones debe encenderse nuevamente, y el dispositivo estar disponible para cualquier nuevo evento. Si los leds no se encendieran o el pitido no se escuchara, entonces habría que revisar exclusivamente la alimentación de los mismos, para descartar fallas eléctricas. Si no las hubiere, entonces sería preciso hacer una revisión del *software*.
6. **Guardado de datos**: Luego de realizadas algunas operaciones, es necesario comprobar la correcta escritura en la tarjeta de memoria. Para ello, es posible extraerla y verificar en una computadora si los datos han sido correctamente registrados, leyendo el archivo correspondiente.

Esto implicará un correcto funcionamiento de la tarjeta de memoria y de las conexiones internas de la placa.

7. **Ejecución de alarmas programadas**: Tanto para los horarios de toma de medicación como para el momento del ingreso del estado del paciente, se programan distintas alarmas sonoras y luminosas. Para evaluar esto, se programan alarmas para los primeros cinco minutos, y poder hacer una rápida prueba, comprobando el funcionamiento de las dos alarmas diferentes. Esto comprobará no solo el funcionamiento de los indicadores sonoros y lumínicos, sino también el correcto funcionamiento del *timer* del microprocesador.

7.1.2 *Medidas*

Conforme a lo dispuesto en el protocolo del segmento anterior, se procedió a hacer las mediciones correspondientes para verificar el correcto funcionamiento tanto del *hardware* como del *software*. Solamente en uno de los pasos se explicita la necesidad de hacer *mediciones* como tales, más allá de la comprobación de ciertas reacciones del dispositivo ante determinadas acciones, cuestiones que tienen mayor relación con el desarrollo del *software*. La única medición eléctrica que se realizó es para evaluar el consumo de corriente, para los distintas operaciones que puede realizar el dispositivo. A continuación se expone la medición realizada.

OPERACIÓN	Consumo de corriente [mA]
Encendido	110
Ninguna	22
Ingreso nuevo estado	61
Confirmación de nuevo estado	61

Tabla 8. *Resultado de las mediciones de consumo del aparato.*

7.1.3 *Evaluación*

1. **Alimentación con batería**: Se encendió, y el dispositivo funciona correctamente.
2. **Alimentación externa sin batería**: Se encendió, y el dispositivo funciona correctamente.
3. **Carga de la batería**: Efectivamente, primero el dispositivo no funcionaba y la batería no funcionaba. Luego de 30 minutos con el dispositivo enchufado, el dispositivo funciona.
4. **Consumo de corriente**: El consumo de corriente es el que se presentó en el segmento anterior.
5. **Respuesta a estímulos**: El dispositivo responde a los estímulos adecuadamente.
6. **Guardado de datos**: La tarjeta de memoria fue leída en una computadora de escritorio y conserva los eventos ingresados.
7. **Ejecución de alarmas programadas**: Se han programado alarmas cada un minuto para refrescar el estado y cada minuto y medio para la toma de medicación, y ambas se activaron adecuadamente.

7.1.4 *Resultados*

Tal como se expuso en el punto anterior, se procedió a realizar el plan de evaluación del dispositivo, y el resultado fue óptimo. Cada uno de los pasos fue respetado cuidadosamente y se obtuvieron los resultados expuestos. De esta forma se verificó que todos los elementos del hardware funcionan separadamente y en conjunto, para lo que son los objetivos del proyecto.

7.2 VALIDACIÓN DE SOFT

Para realizar la validación definitiva del soft, se decidió probar el dispositivo tal cual funcionaría para lo que fue desarrollado. Para esto se modificaron los parámetros del archivo .txt de la tarjeta SD, de forma tal que concuerden con los que serán posiblemente seteados por el médico:

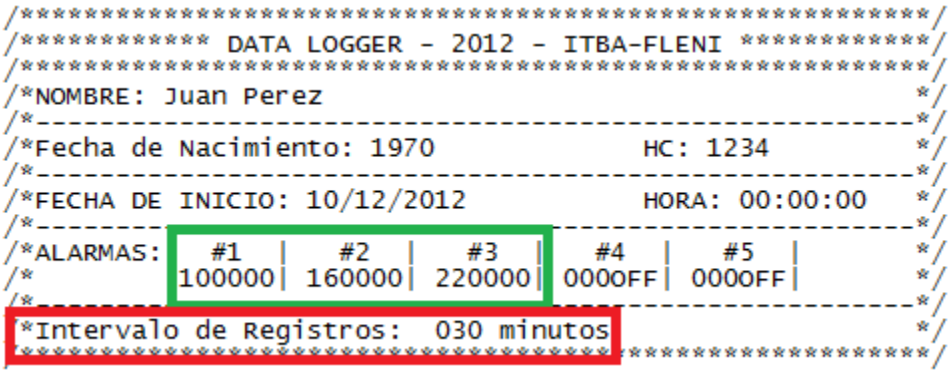


Fig. 8.2.1 – Parámetros seteados en la SD para validación

Con esto, se realizan informes de estado cada 30 minutos, y se debe tomar la medicación en las horas configuradas. El dispositivo se mantuvo encendido durante 5 días, y se lograron los informes necesarios para poder garantizar el correcto funcionamiento del mismo.

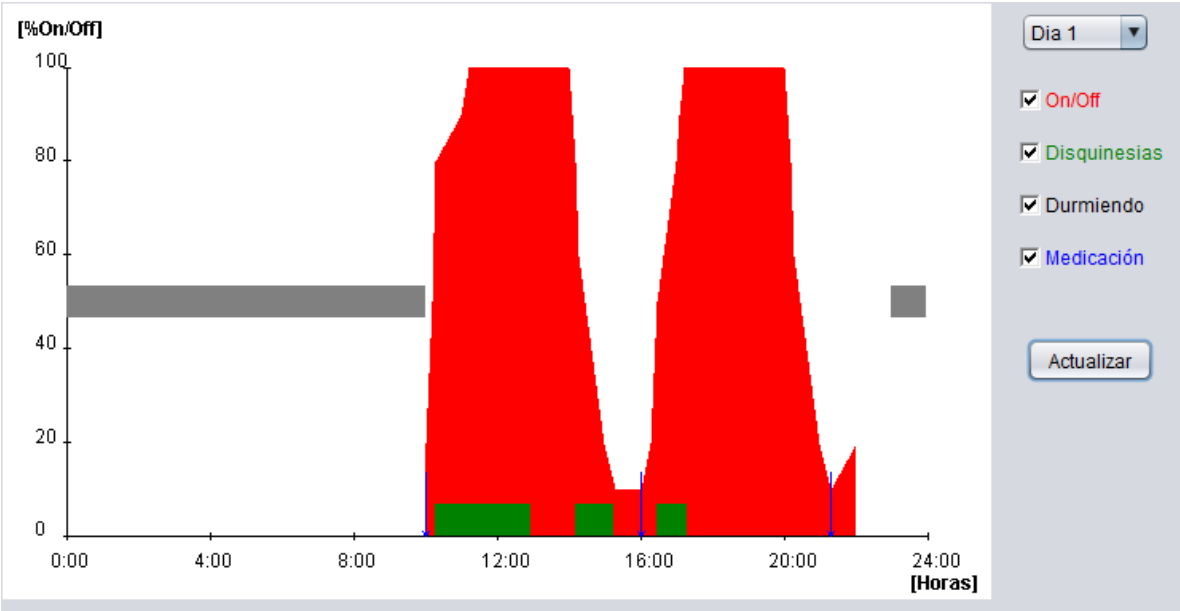


Fig. 8.2.2 – Resultados obtenidos con los datos seteados en Fig.8.2.1

8.1 CONFIABILIDAD DEL HARD

Para realizar el análisis de confiabilidad del Hard se tomó al dispositivo como un único módulo donde el fallo de uno de sus componentes implica el fallo total del sistema, es decir, la necesidad de una reparación. Esto implica calcular el tiempo que el dispositivo estará funcionando con alto rendimiento.

Componente	Cantidad	Material	Marca	MTBF
Resistencias	19	metal film	genérica	3×10^{7}
Capacitores	10	multicapa	genérica	$2{,}5\times10^{7}$
Capacitores	4	electrolítico	genérica	$2{,}5\times10^{7}$
Cristal	1	-	genérica	$1{,}5\times10^{7}$
Zócalo p/SD	1	-	genérica	3×10^{7}
Microprocesador	1	-	Freescale	5×10^{5}
Led	8	-	genérica	3×10^{7}
Switch	2	-	genérica	3×10^{7}
Botón s/ retención	7	-	genérica	3×10^{7}
Pulsador s/ retención	1	-	genérica	3×10^{7}
Buzzer	1	piezoeléctrico	genérica	1×10^{7}
Transistor Bipolar	1	-	genérica	1×10^{7}
Batería	1	ion-litio	genérica	1×10^{6}
Diodo	2	-	genérica	3×10^{7}
Preset	2	-	genérica	3×10^{7}
Controlador de carga	1	-	Microchip	1×10^{6}
Puerto Mini USB	1	-	genérica	3×10^{7}

Para obtener el tiempo medio para que falle el dispositivo se obtuvieron todos los tiempos medios de falla, y se sumaron las probabilidades de falla para cada uno, obteniendo así el tiempo aproximado de falla para el dispositivo íntegramente.

$$MTBF_{dispositivo} = \frac{1}{\sum {}^{1}/_{MTBF_i}} = 159744\ horas = 18\ años$$

9.1 EXCELENCIAS. OBJETIVOS ALCANZADOS

Se ha completado el diseño y construcción de un prototipo que ayudará al diagnóstico de pacientes con Enfermedad de Parkinson de un modo mucho más sencillo y preciso. El objetivo principal ha sido cumplido satisfactoriamente, ya que el dispositivo funciona y cumple con los requerimientos planteados inicialmente, tanto en dimensiones físicas como en funcionalidad. El diseño del mismo se considera absolutamente apto y adecuado para aquellos que serán sus usuarios, tanto los pacientes como los médicos.

Las soluciones y caminos posibles han ido variando en el transcurso del proceso, y finalmente se ha llegado al objetivo de algunas maneras distintas a lo inicialmente proyectado. De todas maneras, se cumplió satisfactoriamente con la demanda, y era de esperarse encontrar ciertas limitaciones en la proyección original.

Se estima que el dispositivo construido logrará una mejora en la calidad de vida en estos pacientes ya que será posible un mejor diagnóstico del comportamiento del mismo y por lo tanto se logrará un seguimiento más personalizado y preciso. El uso del aparato es absolutamente intuitivo y estamos convencidos de que no resultará una molestia si no todo una revolución en el ámbito.

9.2 FALLOS. RECOMENDACIONES PARA FUTUROS DISEÑOS

Lo primero que se puede considerar como un fallo en sí fue la decisión de plantear la necesidad de ciertos accesorios, como una comunicación vía USB para la descarga de datos, cuando el objetivo inicial era más elemental. El primer objetivo a cumplir debía ser la satisfacción del cliente. Luego, el resto sería accesorio. Tal vez en un futuro sí es importante plantear la evolución y hacer un diseño que incluya cuestiones que hagan a la comodidad del médico.

Inicialmente, el objetivo fue hacer todo de forma de darle toda la comodidad al paciente para que comenzar solucionando el sistema de recolección de datos. Para un futuro es importante comenzar a mejorar la interfaz entre el aparato y el médico, que será quien haga la descarga de esos datos. Algunas posibles mejoras son: la implementación de un puerto USB mediante el cual se descarguen los datos directamente, comunicación Ethernet con el médico para que no sea necesario ir a visitar al médico para que obtenga los datos, mejoramiento del software utilizado para la descarga de datos.

Otra deuda pendiente sería la implementación de un gabinete diseñado a medida de la placa, pero el cual tiene sentido cuando se vayan a producir varias unidades del mismo. Los planos del gabinete están incluidos en los anexos y están todos los datos como para que se pueda llevar a la práctica. El problema a la hora de realizar el prototipo fue el costo que implicaría realizar el gabinete para tan solo una unidad.

Finalmente, algo que no se pudo implementar es la inmunización a los golpes, haciendo un recubrimiento de silicona sobre la placa y otro al gabinete, de forma tal que un golpe no afecte el funcionamiento del mismo.

10.1 GLOSARIO

- ‘ON’. Hace referencia al estado del paciente en el cual su estado es óptimo, y los síntomas de la enfermedad están minimizados. Esto se da cuando está bajo el efecto de la medicación. Cabe destacar que el ‘ON’ para cada paciente es distinto, y es absolutamente subjetivo. El mismo paciente puede registrar distintos estados a lo largo del día. Suele pasar que el ‘ON’ a la mañana, luego de la primera medicación del día, es mejor que el de la tarde, y sin embargo, ambos son ‘ON’.

- ‘OFF’. Hace referencia al momento en el cual el paciente sufre los síntomas de la enfermedad, que será aquel en el cual la medicación no esté haciendo efecto. También en este estado existen distintos grados de afectación, como casos en que el efecto de la medicación se esté yendo, pero aún tenga cierta incidencia.

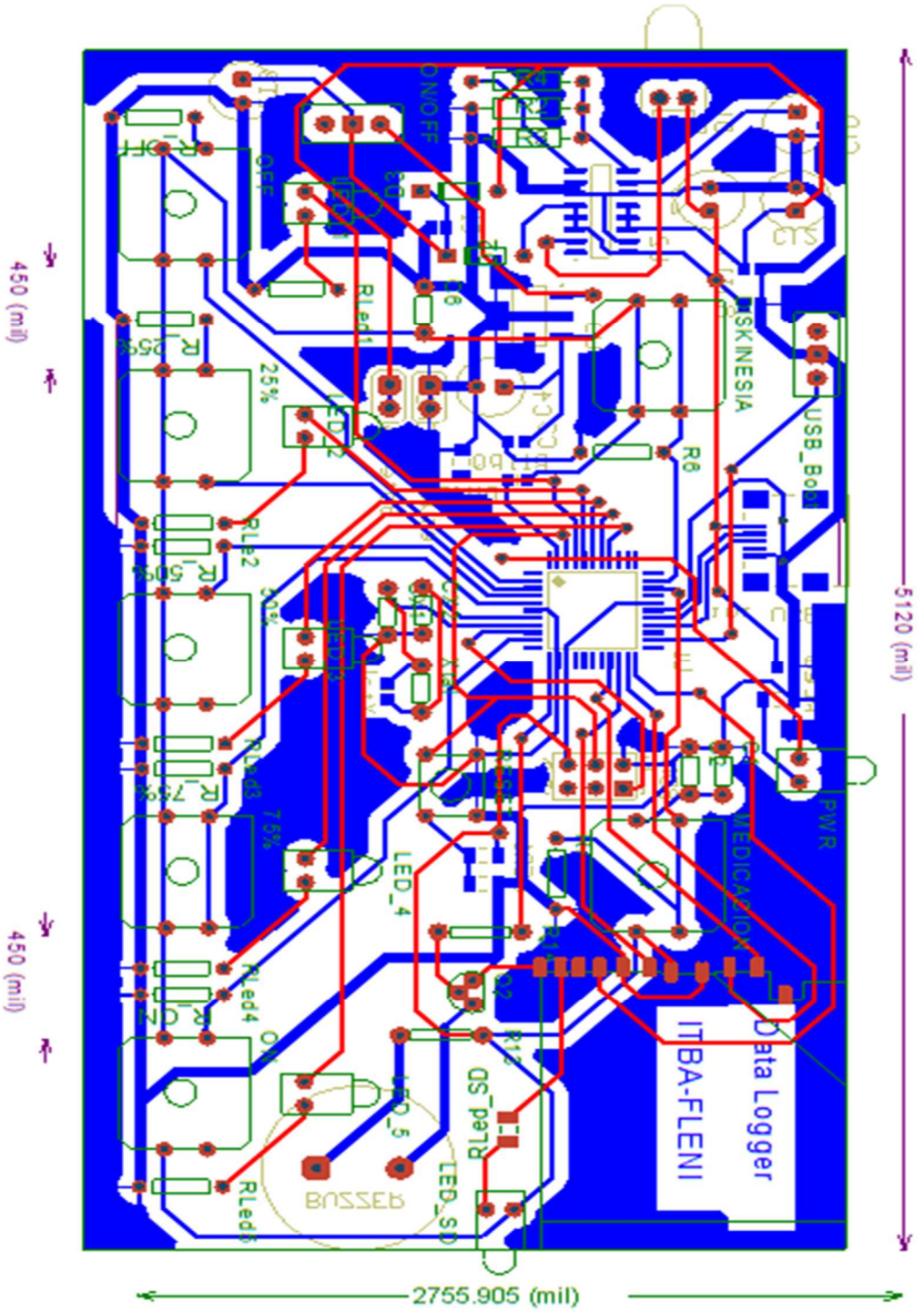
- ‘Levodopa’. Es el fármaco utilizado para contrarrestar los síntomas presentados por la enfermedad. Permite que al cerebro llegue la sustancia llamada dopamina, que es la que da al enfermo la capacidad de controlar sus músculos.

- ‘Disquinesia’. Así se denomina a los movimientos involuntarios que realiza el paciente. Estos suelen darse por una sobredosis de ‘Levodopa’, cuando el paciente pasa de dificultades para moverse a un estado en el cual se mueve constantemente. Generalmente este efecto se da cuando el paciente se encuentra en estado ‘ON’, pero también puede darse cuando está en ‘OFF’.

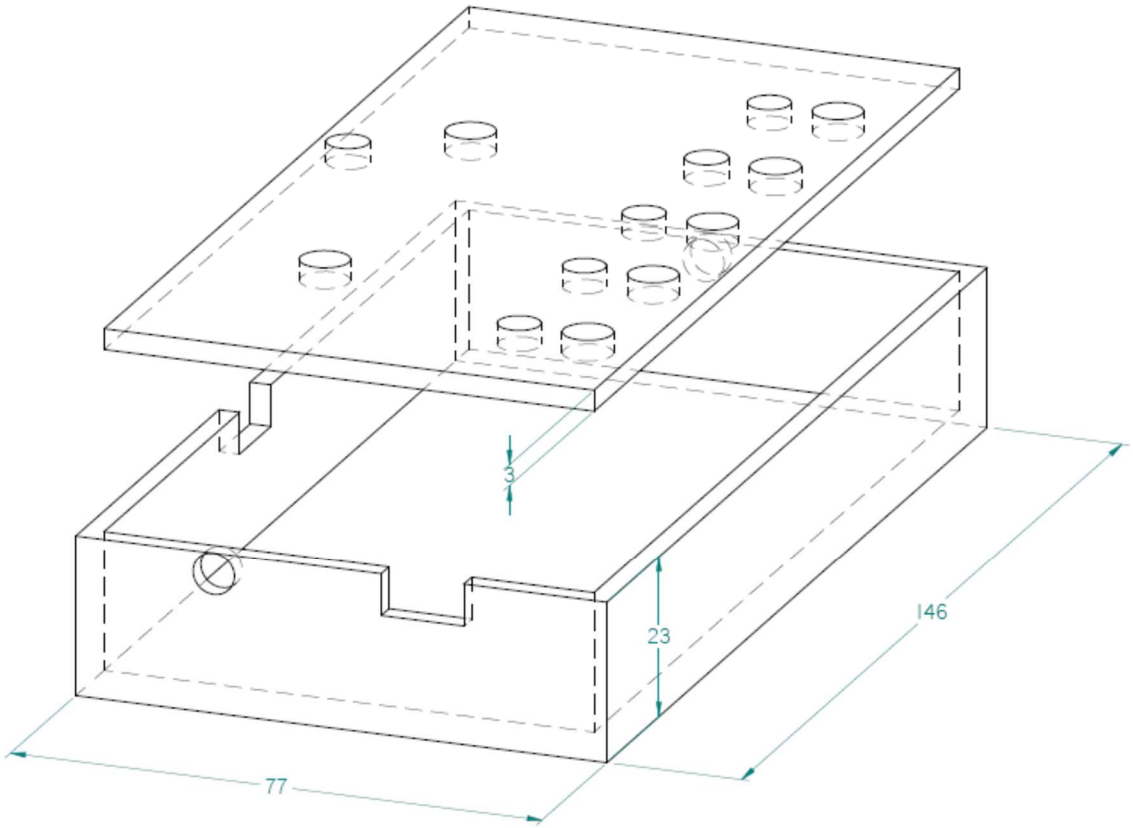
- ‘Pull up’. Término que en electrónica se aplica a la resistencia que conecta por un terminal a algún componente y por el otro a la alimentación. Se utiliza para que en ausencia de actividad el valor no quede indeterminado y quede estable en el valor de la alimentación.

- ‘Pull down’. Al igual que ‘pull up’, término utilizado en electrónica para definir la función de una resistencia que conecta el terminal de algún elemento a tierra, evitando que su valor quede indeterminado, y en ausencia de actividad eléctrica, quede establecido su valor en cero.

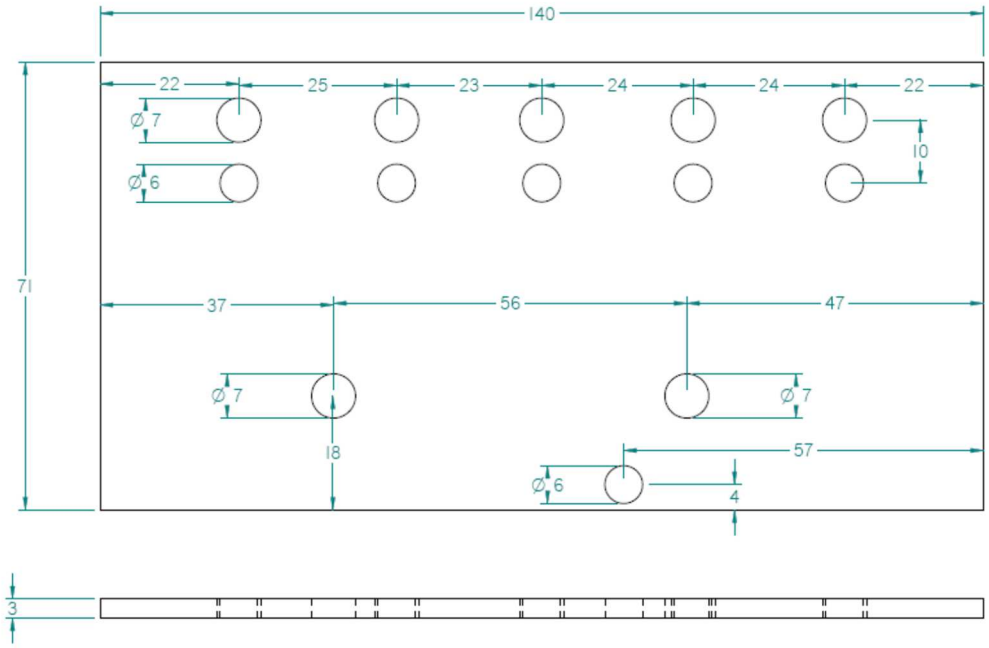
Circuito Impreso



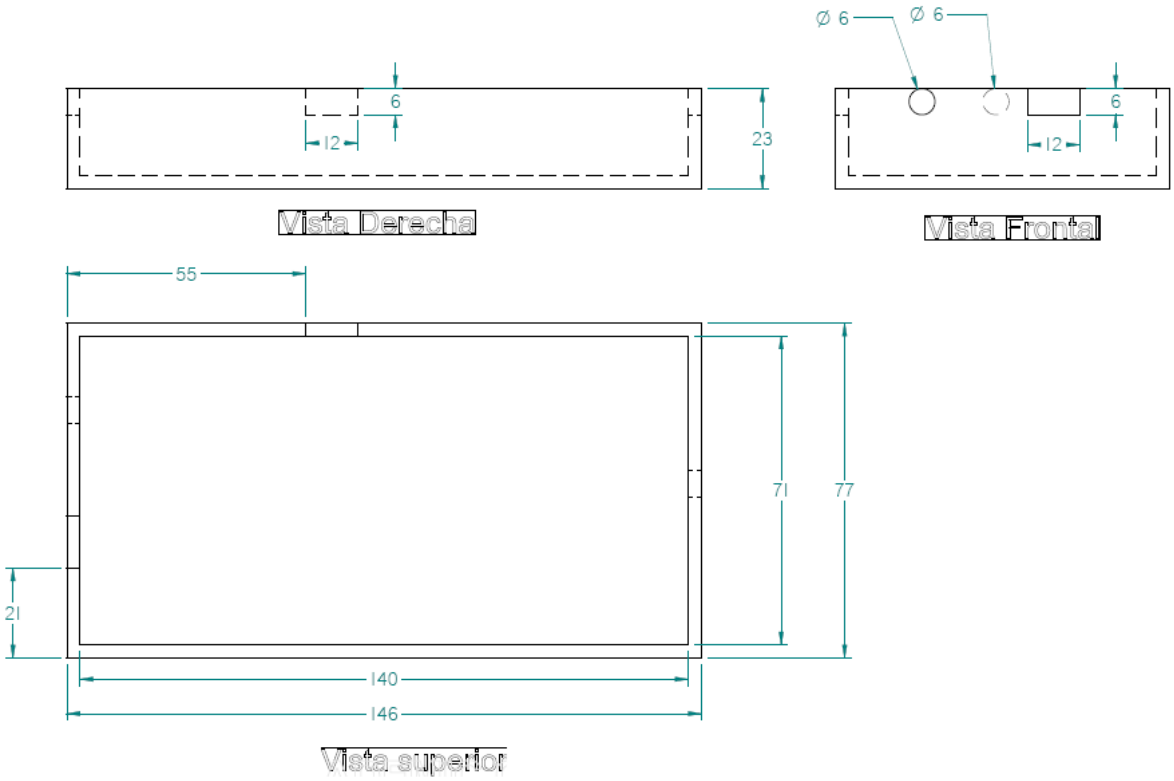
Gabinete 3D



Tapa del gabinete

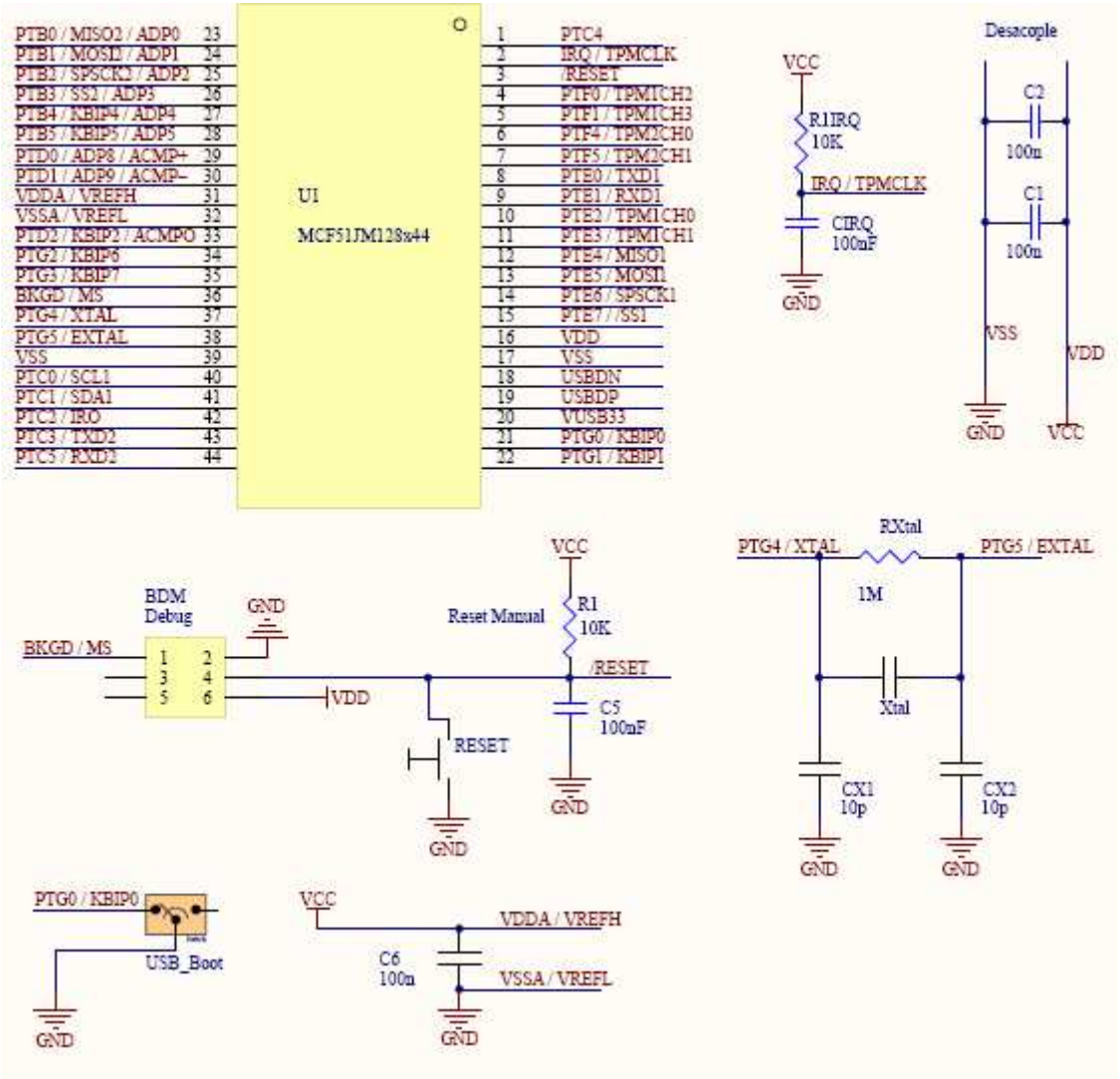


Caja del gabinete

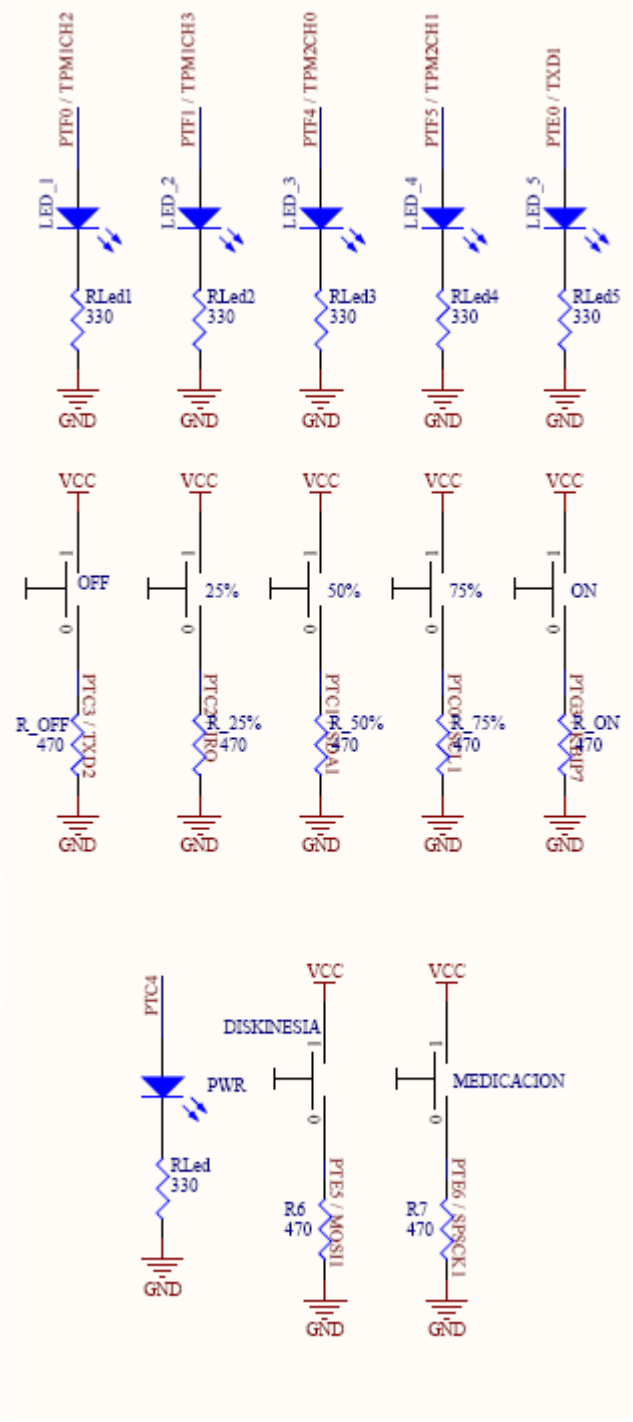


10.3 ESQUEMAS

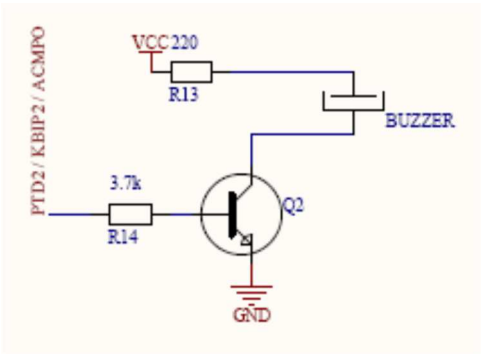
Esquemático Microprocesador y conexiones eléctricas



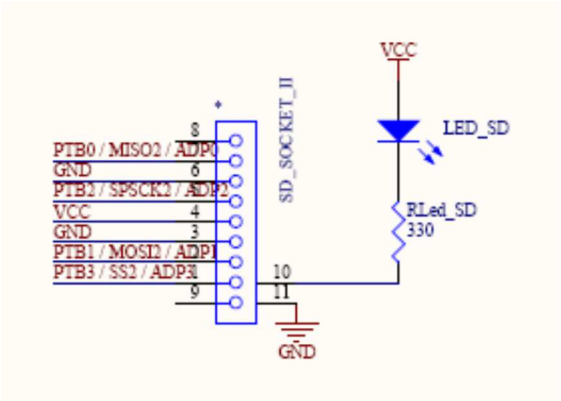
Botoneras e indicadores lumínicos



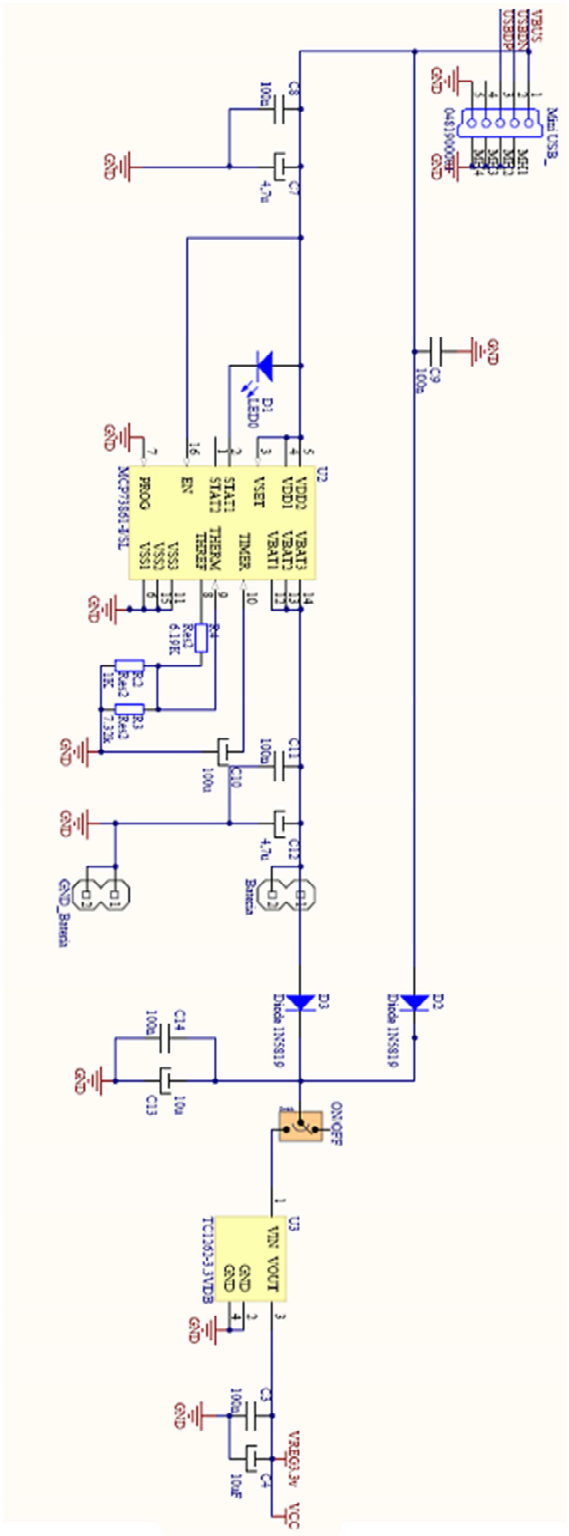
Parlante



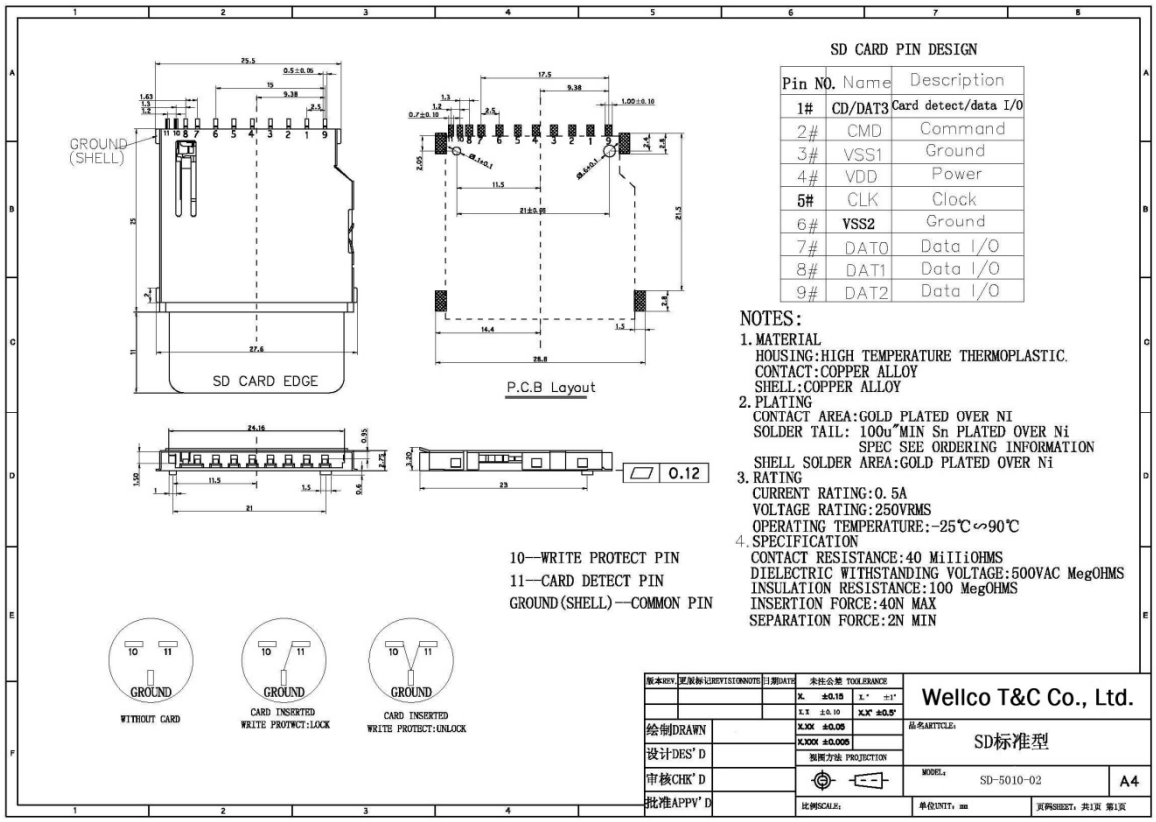
Dispositivo de almacenamiento



Sistema de alimentación



Zócalo SD:


SD CARD PIN DESIGN
Pin NO. Name Description
1# CD/DAT3 Card detect/data I/O
2# CMD Command
3# VSS1 Ground
4# VDD Power
5# CLK Clock
6# VSS2 Ground
7# DAT0 Data I/O
8# DAT1 Data I/O
9# DAT2 Data I/O
GROUND (SHELL)
SD CARD EDGE
P.C.B Layout
0.12
10—WRITE PROTECT PIN
11—CARD DETECT PIN
GROUND(SHELL)—COMMON PIN
WITHOUT CARD
CARD INSERTED WRITE PROTECT:LOCK
CARD INSERTED WRITE PROTECT:UNLOCK
NOTES:
1. MATERIAL
HOUSING:HIGH TEMPERATURE THERMOPLASTIC.
CONTACT:COPPER ALLOY
SHELL:COPPER ALLOY
2. PLATING
CONTACT AREA:GOLD PLATED OVER NI
SOLDER TAIL: 100u"MIN Sn PLATED OVER Ni
SPEC SEE ORDERING INFORMATION
SHELL SOLDER AREA:GOLD PLATED OVER Ni
3. RATING
CURRENT RATING:0.5A
VOLTAGE RATING:250VRMS
OPERATING TEMPERATURE:-25℃~90℃
4. SPECIFICATION
CONTACT RESISTANCE:40 MiIIiOHMS
DIELECTRIC WITHSTANDING VOLTAGE:500VAC MegOHMS
INSULATION RESISTANCE:100 MegOHMS
INSERTION FORCE:40N MAX
SEPARATION FORCE:2N MIN
Wellco T&C Co., Ltd.
SD标准型
SD-5010-02
A4

10.4 LISTADO DE PARTES

Componente	Valor	Material	Marca	Denominación en esquemático
Microprocesador	MCF51JM128	-	Freescale	U1
Resistencia	1 Mohm	Metal film	-	RXtal
Cristal	12 MHz	-	-	Xtal
Capacitor	10 pF	Multicapa	-	CX1
Capacitor	10 pF	Multicapa	-	CX2
Capacitor	100 nF	Multicapa	-	C6
Zócalo para SD	-	-	-	SD_Socket_II
Resistencia	330 ohm	Metal film	-	RLed_SD
Led	5 mm	-	-	Led_SD
Capacitor	100 nF	Multicapa	-	C1
Capacitor	100 nF	Multicapa	-	C2
Switch	-	-	-	USB_Boot
Pulsador	s/retención	-	-	RESET
Resistencia	10 kohm	Metal film	-	R1
Capacitor	100 nF	Multicapa	-	C5
Resistencia	330 ohm	Metal film	-	RLed
Led	5 mm	-	-	PWR
Botón	s/retención	-	-	DISKINESIA
Botón	s/retención	-		MEDICACION
Resistencia	470 ohm	Metal film	-	R6
Resistencia	470 ohm	Metal film	-	R7
Resistencia	330 ohm	Metal film	-	RLed1
Resistencia	330 ohm	Metal film	-	RLed2
Resistencia	330 ohm	Metal film	-	RLed3
Resistencia	330 ohm	Metal film	-	RLed4
Resistencia	330 ohm	Metal film	-	RLed5
Led	5 mm	-	-	LED_1
Led	5 mm	-	-	LED_2
Led	5 mm	-	-	LED_3
Led	5 mm	-	-	LED_4
Led	5 mm	-	-	LED_5
Resistencia	470 ohm	Metal film	-	R_OFF
Resistencia	470 ohm	Metal film	-	R_25%
Resistencia	470 ohm	Metal film	-	R_50%
Resistencia	470 ohm	Metal film	-	R_75%
Resistencia	470 ohm	Metal film	-	R_ON
Botón	s/retención	-	-	OFF
Botón	s/retención	-	-	25%
Botón	s/retención	-	-	50%

Botón	s/retención	-	-	75%
Botón	s/retención	-	-	ON
Buzzer	-	piezoeléctrico	-	BUZZER
Transistor Bipolar	BC548	-	-	Q2
Resistencia	220 ohm		-	R13
Resistencia	3,7 kohm	Metal film	-	R14
Capacitor	100 nF	Multicapa		C3
Capacitor	10 µF	Electrolítico		C4
Batería	TC1262-3.3VDB	Ion-Litio	-	U3
Switch		-	-	ON/OFF
Capacitor	100 nF	-	-	C14
Capacitor	10 µF	-	-	C13
Diodo	1N5819	-	-	D2
Diodo	1N5819	-	-	D3
Capacitor	4,7 µF	Electrolítico	-	C12
Capacitor	100 nF	Multicapa	-	C11
Capacitor	100 µF	Electrolítico	-	C10
Resistencia	1 kohm	Metal film	-	R2
Preset	(10k)7,32 kohm	Metal film	-	R3
Preset	(10k)6,19 kohm	Metal film	-	R4
Led	5 mm	-	-	LED0
Capacitor	4,7 µF	Electrolítico	-	C7
Capacitor	100 nF	Multicapa	-	C8
Capacitor	100 nF	Multicapa	-	C9
Controlador de carga	MCP73861	-	Microchip	MiniUSB_
Puerto Mini USB	Hembra	-	-	

Main.c

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "Bootloader_V1.h"

#include "FslTypes.h"
#include "target.h"
//#include "SPI.h"
#include "SD.h"
#include "Fat.h"
#include "Config.h"
#include "SCSI_Process.h"
#include "OS_RTC.h"
#include "timer.h"
#include "alarma.h"
#include "log.h"



// PROTOTIPOS DE FUNCIONES:
void system_init(void);


/*Variables*/
UINT16 State_Interval;//Intervalo para informe de estado. ej.30min
UINT16 INTERVAL;

/* extern Variables */
extern byte vCBWBuf_flag;
extern ALARMA_STRUCT Alarma[4]; //Estructura de 5 alarmas posibles


void main(void) {

  volatile UINT8 u8Error;


  system_init();// Se copia la tabla de interrupciones a la RAM. Puedo inicializar acá las interrupciones.


  /* !! Start application code below here !! */
  //------------------------------------------------------------//
  SOPT1 = 0x00;                  // disable COP
  SOPT2 = 0x00;

  RTC_Init();
  hw_init();                    /* MCU Initialization */
  //SPI_Init();                  /* SPI Module Init */

  u8Error=SD_Init();              /* SD Card Init */
  SPI_Init();
```

```
SCSI_MemoryInfo();


  /**********PROBAR FUNCIONAMIENDO DE SD***************/
  while(u8Error==NO_SD_CARD){
/* Wait until button clic */
    //PTBD_PTBD5 =0;
    LED_PIN1_OFF;
    u8Error=SD_Init();
  }

  if(u8Error==OK){
    //PTBD_PTBD5 =1; //SD ON
LED_PIN1_ON;
    }
  //-------------------------------------------------------//
  EnableInterrupts;


  /*****************************************************/
  /*******CREO ARCHIVO LOG.TXT EN LA SD***************/
FAT_Read_Master_Block();            /* FAT Driver Init */

  /* Creating the file if it doesn't exist */
  if (FAT_FileOpen(FILE_NAME, MODIFY) == FILE_NOT_FOUND)
  {
    u8Error=FAT_FileOpen(FILE_NAME,CREATE);
FAT_FileClose();

    Print_Welcome(); //Imprime Titulo de Bienvenida
    Print_Name ();   //Imprime Nombre y alarmas de base
}
  FAT_FileClose();
 /*********************************************************/

  //---------Habilito TIMER TPM1----------------------//
  start_mS_timer(10);
  Init_Timers();
//-------------------------------------------------//
  Init_Alarms();
  Read_Alarms();//Leo las Alarmas a setear y las activo
Read_Date ();
  Read_State_Interval();



//**************  PROGRAMA  *************************//

  for(;;) {

      char i=0;

      for (i=0 ; i<5 && Alarma[i].Alarm_State== ON ; i++)
```

```
(void)Verify_Alarm(i);

      Verify_Inform(State_Interval); // Verifico si es el tiempo en que hay
                                     // q pedirle al paciente que informe su Estado


} /* loop forever */
  /* please make sure that you never leave main */
}
```

SD.c

```
/* Include Files */
#include "SD.h"

/* Global Variables */
T32_8 gu8SD_Argument;

/************************************************/
/************************************************/
UINT8 SD_Init(void)
{
   _SD_PRESENT=_IN;

   /* Check for SD */
   if(SD_PRESENT==0)
      return(NO_SD_CARD);

   /* Initialize SPI Module */
   SPI_Init();


   /* Start SD card Init */
   SPI_SS=ENABLE;
   SD_CLKDelay(10);          // Send 80 clocks
   SPI_SS=DISABLE;

   gu8SD_Argument.lword=0;
   SD_CLKDelay(8);

   /* IDLE Command */

   SPI_SS=ENABLE;
   if(SD_SendCommand(SD_CMD0|0x40,SD_IDLE))
   {
      SPI_SS=DISABLE;
      return(INIT_FAILS);
   }
   SPI_SS=DISABLE;

   (void)SPI_Receive_byte();  // Dummy SPI cycle

   /*  Initialize SD Command */
   SPI_SS=ENABLE;
   while(SD_SendCommand(SD_CMD1|0x40,SD_OK));
   SPI_SS=DISABLE;

   (void)SPI_Receive_byte();  // Dummy SPI cycle

   /*  Block Length */
   SPI_SS=ENABLE;

   gu8SD_Argument.lword=SD_BLOCK_SIZE;
   if(SD_SendCommand(SD_CMD16|0x40,SD_OK))
```

```
    {
      SPI_SS=DISABLE;
      return(INIT_FAILS);
    }

    SPI_SS=DISABLE;

    SPI_High_rate();

    SPI_Send_byte(0x00);
    SPI_Send_byte(0x00);
    return(OK);
}

/*************************************************/
/*************************************************/
UINT8 SD_Write_Block(UINT32 u16SD_Block,UINT8 *pu8DataPointer)
{
    UINT16 u16Counter;

    SPI_SS=ENABLE;

    gu8SD_Argument.lword=u16SD_Block;
    gu8SD_Argument.lword=gu8SD_Argument.lword<< SD_BLOCK_SHIFT;

    if(SD_SendCommand(SD_CMD24|0x40,SD_OK))
    {
      SPI_SS=DISABLE;
      return(WRITE_COMMAND_FAILS);
    }

    SPI_Send_byte(0xFE);

    for(u16Counter=0;u16Counter<BLOCK_SIZE;u16Counter++)
      SPI_Send_byte(*pu8DataPointer++);

    SPI_Send_byte(0xFF);    // checksum Bytes not needed
    SPI_Send_byte(0xFF);

    for(u16Counter=0;u16Counter<BLOCK_SIZE;u16Counter++);

    if((SPI_Receive_byte() & 0x0F) != 0x05)
    {
      SPI_SS=DISABLE;
      return(WRITE_DATA_FAILS);
    }

    while(SPI_Receive_byte()==0x00);  // Dummy SPI cycle

    SPI_SS=DISABLE;
    return(OK);
}
/*************************************************/
/*************************************************/
```

```
UINT8 SD_Read_Block(UINT32 u16SD_Block,UINT8 *pu8DataPointer)
{
   UINT8 u8Temp=0;
   UINT16 u16Counter;

   SPI_SS=ENABLE;
   ////////////////////////////////////////////
   //while(PTBD_PTBD4==0);
   ////////////////////////////

   gu8SD_Argument.lword=u16SD_Block;
   gu8SD_Argument.lword=gu8SD_Argument.lword<< SD_BLOCK_SHIFT;

   if(SD_SendCommand(SD_CMD17|0x40,SD_OK))
   {
      SPI_SS=DISABLE;
      return(READ_COMMAND_FAILS);
   }

   while(u8Temp!=0xFE)
      u8Temp=SPI_Receive_byte();

   for(u16Counter=0;u16Counter<BLOCK_SIZE;u16Counter++)
      *pu8DataPointer++=SPI_Receive_byte();

   (void)SPI_Receive_byte();  // Dummy SPI cycle
   (void)SPI_Receive_byte();

   SPI_SS=DISABLE;

   (void)SPI_Receive_byte();  // Dummy SPI cycle

   return(OK);
}

/*************************************************/
/*************************************************/
UINT8 SD_SendCommand(UINT8 u8SDCommand,UINT8 u8SDResponse)
{
   UINT8 u8Counter;
   volatile UINT8 u8Temp=0;

   /* Send Start byte */
   SPI_Send_byte(u8SDCommand);

   /* Send Argument */
   for(u8Counter=0;u8Counter<4;u8Counter++)
      SPI_Send_byte(gu8SD_Argument.bytes[u8Counter]);

   /* Send CRC */
   SPI_Send_byte(0x95);

   /* Response RHandler */
   u8Counter=SD_WAIT_CYCLES;
```

```
   do
   {
      u8Temp=SPI_Receive_byte();
u8Counter--;
   }while((u8Temp != u8SDResponse) && u8Counter > 0);

if(u8Counter)   return(OK);
   else          return(COMMAND_FAILS);
}

/*************************************************/
/*************************************************/
void SD_CLKDelay(UINT8 u8Frames)
{
   while(u8Frames--)
      SPI_Send_byte(0xFF);
}
```

SD.h

```
#ifndef __SD__
#define __SD__

/* Includes */

#include "FslTypes.h"
#include "derivative.h"
#include "SPI.h"

/* User definitions */
#define SD_BLOCK_512
#define SD_WAIT_CYCLES 10

/* SD card Inserted detection Pin */
#define SD_PRESENT      PTED_PTED5
#define _SD_PRESENT     PTEDD_PTEDD5
//////////////////////////////////////////////////////
#define SD_WR     PTBD_PTBD4    // write protection
#define _SD_WR     PTBD_PTBD4    // write protection
#define kSD_WP    1
#define kSD_WnP   0
//////////////////////////////////////////////////////
//#define SD_PRESENT        PTGD_PTGD3
//#define _SD_PRESENT       PTGDD_PTGDD3

/* Error Codes */
enum
{
      OK,
      COMMAND_FAILS,
      INIT_FAILS,
      WRITE_COMMAND_FAILS,
      WRITE_DATA_FAILS,
      READ_COMMAND_FAILS,
      READ_DATA_FAILS,
      NO_SD_CARD
};

/* Status */
enum
{
   SD_OK,
   SD_IDLE
};

/* TypeDefs */
typedef union
{
      UINT8 bytes[4];
      UINT32 lword;
}T32_8;
```

```
typedef union
{
    UINT8  u8[2];
    UINT16 u16;
}T16_8;



/* Stardar Definitions */
#ifdef  SD_BLOCK_512
   #define SD_BLOCK_SIZE   (0x00000200)
   #define SD_BLOCK_SHIFT  (9)
   #define BLOCK_SIZE      512
#endif


/* Static Definitions */
/*********************************  SD  Card  Standard  Commands
************************************/
enum{
  SD_CMD0,  /* Resets the SD Memory Card */
  SD_CMD1,  /* Sends host capacity support information and activates the card's
          initialization process. HCS is effective when card receives SEND_IF_COND
          command. Reserved bits shall be set to '0'. */
  SD_CMD2,
  SD_CMD3,
  SD_CMD4,
  SD_CMD5,
  SD_CMD6,  /* Checks switchable function (mode 0) and switches card function (mode 1).*/
  SD_CMD7,
  SD_CMD8,  /* Sends SD Memory Card interface condition that includes host supply voltage
          information and asks the accessed card whether card can operate in supplied
          voltage range. Reserved bits shall be set to '0'.*/
  SD_CMD9,  /* Asks the selected card to send its cardspecific data (CSD)*/
  SD_CMD10, /* Asks the selected card to send its card identification (CID) */
  SD_CMD11,
  SD_CMD12, /* Forces the card to stop transmission in Multiple Block Read Operation */
  SD_CMD13, /* Asks the selected card to send its status register. */
  SD_CMD14,
  SD_CMD15,
  SD_CMD16, /* Sets a block length (in bytes) for all following block commands (read and
          write) of a Standard Capacity Card. Block length of the read and write
          commands are fixed to 512 bytes in a High Capacity Card. The length of
          LOCK_UNLOCK command is set by this command in both capacity cards.*/
  SD_CMD17, /* Reads a block of the size selected by the SET_BLOCKLEN command.*/
  SD_CMD18, /* Continuously transfers data blocks from card to host until interrupted by a
          STOP_TRANSMISSION command.*/
  SD_CMD19,
  SD_CMD20,
  SD_CMD21,
  SD_CMD22,
  SD_CMD23,
  SD_CMD24, /* Writes a block of the size selected by the SET_BLOCKLEN command. */
  SD_CMD25, /* Continuously writes blocks of data until 'Stop Tran' token is sent
```

```
            (instead 'Start Block').*/
        SD_CMD26,
        SD_CMD27, /* Programming of the programmable bits of the CSD. */
        SD_CMD28, /* If the card has write protection features, this command sets the write protection bit
            of the addressed group. The properties of write protection are coded in the card
            specific data (WP_GRP_SIZE). The High Capacity Card does not support this command.*/
        SD_CMD29, /* If the card has write protection features, this command clears the write protection
            bit of the addressed group. The High Capacity Card does not support this command. */
        SD_CMD30, /* If the card has write protection features, this command asks the card to send the
            status of the write protection bits.6 The High Capacity Card does not support this command. */
        SD_CMD31,
        SD_CMD32, /* Sets the address of the first write block to be erased.*/
        SD_CMD33, /* Sets the address of the last write block of the continuous range to be erased. */
        SD_CMD34,
        SD_CMD35,
        SD_CMD36,
        SD_CMD37,
        SD_CMD38, /* Erases all previously selected write blocks */
        SD_CMD39,
        SD_CMD40,
        SD_CMD41,
        SD_CMD42, /* Used to Set/Reset the Password or lock/unlock the card. A transferred data block
includes
            all the command details - refer to Chapter 4.3.7. The size of the Data Block is defined
            with SET_BLOCK_LEN command. Reserved bits in the argument and in Lock Card Data
Structure
            shall be set to 0. */
        SD_CMD43,
        SD_CMD44,
        SD_CMD45,
        SD_CMD46,
        SD_CMD47,
        SD_CMD48,
        SD_CMD49,
        SD_CMD50,
        SD_CMD51,
        SD_CMD52,
        SD_CMD53,
        SD_CMD54,
        SD_CMD55, /* Defines to the card that the next command is an application specific command
            rather than a standard command */
        SD_CMD56, /* Used either to transfer a Data Block to the card or to get a Data Block from the card
            for general purpose/application specific commands. In case of Standard Capacity SD
            Memory Card, the size of the Data Block shall be defined with SET_BLOCK_LEN command.
            Block length of this command is fixed to 512-byte in High Capacity Card. */
        SD_CMD57,
        SD_CMD58, /* Reads the OCR register of a card. CCS bit is assigned to OCR[30]. */
        SD_CMD59, /* Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on,
            a '0' will turn it off */
      SD_CMD60,
        SD_CMD61,
        SD_CMD62,
        SD_CMD63
      };
```

```
/* Prototypes */
UINT8 SD_Init(void);
UINT8 SD_SendCommand(UINT8,UINT8);
void SD_CLKDelay(UINT8);
UINT8 SD_Write_Block(UINT32,UINT8*);
UINT8 SD_Read_Block(UINT32,UINT8*);

#endif /* __SD__ */
```

SD_scsi.c

```
#include "derivative.h" /* include peripheral declarations */
#include "bootloader_V1.h"
#include "SCSI_Process.h"
#include "Fat16.h"
#include "USB_Handle.h"

////////////////////////////////////////////////SD_SCSI
UINT8  vSD_CSD[16];       // SD card CSD vaule
byte  vSD_Bnum;           // Block number
dword vSD_LBA;            // BLock address


T32_8 u32MaxBlocks;
T16_8 u16BlockSize;
////////////////////////////////////////////////

////////////////////////////////////SD_SCSI
//extern tBDT EP2Bo;
extern byte vEP2State;
//extern byte vUSB_Trf_State;
////////////////////////////////////



//NUEVO
/*****************************************************/
//////////////////   SD_FUCTIONS    ////////////////
void SD_SCSI_Init(void)
{
   (void)SD_Init();
   SCSI_MemoryInfo();
            //USB_Init();
}


/***********************************************************
* Name: SD_ReadCSD
* Desc: Read CSD vaule of SD card
* Parameter: None
* Return: Status of read -- Fail:04  Success:00
***********************************************************/

byte SD_ReadCSD(void)
{
   UINT8 i=0;

   SPI_SS=ENABLE;

   if(SD_SendCommand(SD_CMD9|0x40,SD_OK))
   {
      SPI_SS=DISABLE;
      return(4);
```

```
    }

    while(i!=0xFE)
      i=SPI_Receive_byte();


    for(i=0;i<16;i++)
      vSD_CSD[i]=SPI_Receive_byte();


    (void)SPI_Receive_byte();
    (void)SPI_Receive_byte();

    SPI_SS=DISABLE;

    (void)SPI_Receive_byte();



    return(0);
}


/*************************************************************
* Name: SCSI_MemoryInfo
* Desc: Storage information of SD Card
* Parameter: None
* Return: None
*************************************************************/
void SCSI_MemoryInfo(void)
{
    UINT16 u16Temp;

    (void)SD_ReadCSD();


    // Block Size
    u16BlockSize.u16= 1<<(vSD_CSD[5] & 0x0F);

    // Max Blocks
    u16Temp= (vSD_CSD[10]&0x80)>>7;
    u16Temp+=(vSD_CSD[9]&0x03)<<1;
    u32MaxBlocks.lword= vSD_CSD[6]&0x03;
    u32MaxBlocks.lword=u32MaxBlocks.lword<<8;
u32MaxBlocks.lword+=vSD_CSD[7];
    u32MaxBlocks.lword=u32MaxBlocks.lword<<2;
    u32MaxBlocks.lword+=(vSD_CSD[8]&0xC0)>>6;
    u32MaxBlocks.lword++;
    u32MaxBlocks.lword=u32MaxBlocks.lword<<(u16Temp+2);

// Patch for SD Cards of 2 GB
    if(u16BlockSize.u16 > 512)
    {
      u16BlockSize.u16=512;
```

```
u32MaxBlocks.lword=u32MaxBlocks.lword<<1;
    }


}
```

SPI.c

```
#include "SPI.h"

/***********************************************/
void SPI_Init(void)
{

  //SOPT2 = SOPT2_SPI1PS_MASK;  // Drive PTE as SPI port

SPI_SS = 1;
  _SPI_SS= 1;


  SPI2BR = 0x24; // 375 Khz
  SPI2C2 = 0x00;
  SPI2C1 = SPI2C1_SPE_MASK | SPI2C1_MSTR_MASK;
}

/***********************************************/
void SPI_Send_byte(UINT8 u8Data)
{
      (void)SPI2S;
      SPI2DL=u8Data;
      while(!SPI2S_SPTEF);
}

/***********************************************/
UINT8 SPI_Receive_byte(void)
{
      (void)SPI2DL;
      SPI2DL=0xFF;
      while(!SPI2S_SPRF);
      return(SPI2DL);
}

/***********************************************/
void SPI_High_rate(void)
{
  SPI2C1 = 0x00;
  SPI2BR = 0x11; // 375 Khz
  SPI2C1 = SPI2C1_SPE_MASK | SPI2C1_MSTR_MASK;

}
```

SPI.h

```
#ifndef __SPI__
#define __SPI__

/* Includes */

#include "FslTypes.h"
#include "derivative.h"


/* definitions */

#define SPI_SS    PTBD_PTBD3      /* Slave Select 2*/
#define _SPI_SS   PTBDD_PTBDD3

//#define SPI_SS    PTED_PTED7       /* Slave Select 1 */
//#define _SPI_SS   PTEDD_PTEDD7


#define ENABLE    0
#define DISABLE   1

/* Global Variables */

/* Prototypes */
void SPI_Init(void);
void SPI_Send_byte(UINT8 u8Data);
UINT8 SPI_Receive_byte(void);
void SPI_High_rate(void);


#endif /* __SPI__ */
```

RTC.c

```
#include "RTC.h"
#include "OS_RTC.h"
#include "Timer.h"
#include "Alarma.h"

extern UINT16 State_Interval;
extern UINT16 INTERVAL;
extern ALARMA_STRUCT Alarma[4]; //Estructura de 5 alarmas posibles

void Update_State_Interval(OSTime *Ptr_Hora);


/*********************************************************/
void RTC_Init(void)
{
  RTCSC = RTCSC_RTIE_MASK | SAMPLE_TIME;     // Enable RTC and set time interval
RTCMOD = 0;

}

/*********************************************************/
interrupt void RTC_ISR(void)
{


   RTCSC_RTIF = 1;              // Set RTC Flag
   //ADCSC1|=ADCSC1_ADCH4_MASK;  // Initialize ADC conversion
//PTDD_PTDD1^=1;

   OSUpdateTime( &Hora );     //Actualizo Fecha y Hora
OSUpdateCalendar();

   Update_State_Interval(&Hora);

   //-----------------------------------------------------//



}


void Update_State_Interval(OSTime *Ptr_Hora){

            if (Ptr_Hora -> RTC_Second == 0 && State_Interval!=0)
                State_Interval--;

            else if( State_Interval==0)
                State_Interval=INTERVAL;

}
```

RTC.h

```
#ifndef __RTC__
#define __RTC__

/* Includes */
#include "derivative.h" /* include peripheral declarations */
#include "FslTypes.h"
#include "Config.h"


/*** Conditional Compilation for Real Time interrupt */



//#ifdef SAMPLE_TIME_05_SEG
//# define SAMPLE_TIME
RTCSC_RTCPS3_MASK|RTCSC_RTCPS2_MASK|RTCSC_RTCPS1_MASK
//#endif

//#ifdef SAMPLE_TIME_1_SEG
  #define SAMPLE_TIME
RTCSC_RTCPS3_MASK|RTCSC_RTCPS2_MASK|RTCSC_RTCPS1_MASK|RTCSC_RTCPS0_MASK
//#endif
/*
#ifdef SAMPLE_TIME_2_SEG
  #define SAMPLE_TIME
RTCSC_RTCLKS1_MASK|RTCSC_RTCLKS0_MASK|RTCSC_RTCPS2_MASK|RTCSC_RTCPS1_MAS
K|RTCSC_RTCPS0_MASK
#endif

#ifdef SAMPLE_TIME_3_SEG
  #define SAMPLE_TIME    RTCSC_RTCLKS1_MASK|RTCSC_RTCLKS0_MASK|
                  RTCSC_RTCPS3_MASK|RTCSC_RTCPS2_MASK|RTCSC_RTCPS1_MASK
#endif

#ifdef SAMPLE_TIME_6_SEG
  #define SAMPLE_TIME    RTCSC_RTCLKS1_MASK|RTCSC_RTCLKS0_MASK|

RTCSC_RTCPS3_MASK|RTCSC_RTCPS2_MASK|RTCSC_RTCPS1_MASK|RTCSC_RTCPS0_MASK
#endif
*/


/* Prototypes */
void RTC_Init(void);
interrupt void RTC_ISR(void);



#endif /* __RTC__ */
```

OS_RTC.c

```
#include "FslTypes.h"
#include "OS_RTC.h"

// Estructura - Hora
  OSTime Hora;
  OSDate Data;
  OS_RTC OSRtc;

// Lookup table holding the length of each mont. The first element is a dummy.
  static const UINT8 MonthLength[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  static UINT8 LeapMonth;



////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/////      OS Update Time Function                  /////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

void OSUpdateTime(OSTime *Ptr_Hora)
{


  Ptr_Hora -> RTC_Second++;

      if (Ptr_Hora -> RTC_Second == 60){

                Ptr_Hora -> RTC_Second = 0;
                Ptr_Hora -> RTC_Minute++;

      if (Ptr_Hora -> RTC_Minute == 60){

                Ptr_Hora -> RTC_Minute = 0;
                Ptr_Hora -> RTC_Hour++;

      if (Ptr_Hora -> RTC_Hour == 24){

                Ptr_Hora -> RTC_Hour = 0;

      }}}


}

void OSUpdateCalendar(void)
{
   OSRtc.Sec++;              // increment second

   if (OSRtc.Sec == 60)
   {
      OSRtc.Sec = 0;
      OSRtc.Min++;
```

```
      if (OSRtc.Min > 59)
      {
         OSRtc.Min = 0;
         OSRtc.Hour++;

         if (OSRtc.Hour > 23)
         {
            OSRtc.Hour = 0;
            OSRtc.Day++;

            // Check for leap year if month == February
            if (OSRtc.Month == 2)
               if (!(OSRtc.Year & 0x0003))          // if (gYEAR%4 == 0)
                  if (OSRtc.Year%100 == 0)
                     if (OSRtc.Year%400 == 0)
                        LeapMonth = 1;
                     else
                        LeapMonth = 0;
                  else
                     LeapMonth = 1;
               else
                  LeapMonth = 0;
            else
               LeapMonth = 0;

            // Now, we can check for month length
            if (OSRtc.Day > (MonthLength[OSRtc.Month] + LeapMonth))
            {
               OSRtc.Day = 1;
               OSRtc.Month++;

               if (OSRtc.Month > 12)
               {
                  OSRtc.Month = 1;
                  OSRtc.Year++;
               }
            }
         }
      }
   }
}

/*
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
/////      Return Calendar Function                  /////
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

void GetCalendar(OS_RTC *rtc)
{
  UserEnterCritical();
  *rtc = OSRtc;
```

```
  UserExitCritical();
}


*/

////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/////      Set Calendar Function                  /////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

void SetCalendar(OS_RTC *rtc)
{

  OSRtc = *rtc;

}

////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////


void Init_Calendar(UINT8  Day, UINT8  Month,UINT16 Year,UINT8  Hour, UINT8  Min,UINT8  Sec)
{
  OS_RTC rtc;

  rtc.Year  = Year;
  rtc.Month = Month;
  rtc.Day   = Day;
  rtc.Hour  = Hour;
  rtc.Min   = Min;
  rtc.Sec   = Sec;

SetCalendar(&rtc);
}

void Init_Hora(void)
{

  Hora.RTC_Hour    = 0;
Hora.RTC_Minute  = 0;
  Hora.RTC_Second  = 0;

}
```

OS_RTC.h

```
#ifndef OS_RTC_H
#define OS_RTC_H

//#include "OS_types.h"
 #include "FslTypes.h"

////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/////    OS Time Structure                         /////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

/**
* \struct OSTime
* Real time clock - shows the current hours, minutes and seconds or the uptime info
*/

typedef struct _OSTime{

      UINT8 RTC_Second;        ///< Seconds of the clock
      UINT8 RTC_Minute;        ///< Minutes of the clock
      UINT8 RTC_Hour;                ///< Hours of the clock

} OSTime;


////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/////    OS Date Structure                         /////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

/**
* \struct OSDate
* Operating System Date - Shows the current day, month and year ou the uptime info
*/

typedef struct {

      UINT8       RTC_Day;         ///< Day of the date
      UINT8       RTC_Month;       ///< Month of the date
  UINT16      RTC_Year;        ///< Year of the date
} OSDate;


////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/////    OS Time and Date Structure                /////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

/**
* \struct OSTime_Date
```

```
    * Operating System Date and time - Shows the current time and date
    */

    typedef struct {


        OSTime  DC_Hora_Exata;
        OSDate  DC_Dia_Exato;

    } OSTime_Date;

    ////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////


    struct _OSRTC
    {
      UINT16 Year;
      UINT8  Month;
    UINT8  Day;
      UINT8  Hour;
      UINT8  Min;
      UINT8  Sec;
    };

    typedef struct _OSRTC OS_RTC;


    ////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////
    /////      Functions Prototypes                  /////
    ////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////

    /*****************************************************************************************
*****
    * \fn void OSUpdateTime(OSTime *Ptr_Hora)
    * \brief Update the system time
    *  Used to update the system time
    * \param Ptr_Hora - pointer to the current system time
    * \return None
    *****************************************************************************************
*****/

    void OSUpdateTime(OSTime *Ptr_Hora);

    void OSUpdateDate(OSDate *Ptr_Dia);

    void OSResetTime(OSTime *Ptr_Hora);

    void OSResetDate(OSDate *Ptr_Dia);
```

```
void OSUpdateUptime(OSTime *Ptr_Hora,OSDate *Ptr_Dia);

OSTime OSUptime(void);

OSDate OSUpDate(void);

// Calendar Functions
void OSUpdateCalendar(void);
void GetCalendar(OS_RTC *rtc);
void SetCalendar(OS_RTC *rtc);
void Init_Calendar(UINT8  Day, UINT8  Month,UINT16 Year,UINT8  Hour, UINT8  Min,UINT8  Sec);
void Init_Hora(void);


//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////
/////   Time and Date Variables Extern Declarations    /////
//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////

extern OSTime Hora;
extern OSDate Data;
extern OS_RTC OSRtc;
//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////


#endif
```

Log.h

```
void Print_Welcome (void);
void Print_Name(void);
void Print_Hour (void);
void Print_Date (void);

void Print_Estado(int estado, char u);

void Read_Date ( void );
void Read_Alarms(void);
```

log.c

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "Bootloader_V1.h"
#include "FslTypes.h"
#include "target.h"
#include "SD.h"
#include "Fat.h"
#include "Config.h"
#include "OS_RTC.h"
#include "alarma.h"

extern UINT16 State_Interval;
extern UINT16 INTERVAL;


void Print_Welcome (void){

char texto[62];
  volatile UINT8 u8Error;

u8Error=FAT_FileOpen(FILE_NAME,MODIFY);

texto = "/************************************************************/  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));
  texto = "/************ DATA LOGGER - 2012 - ITBA-FLENI ************/  ";
  texto[60] = 13;//CR
texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));
texto = "/************************************************************/  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));
FAT_FileClose();
}


void Print_Name(void){

char texto[62];
  volatile UINT8 u8Error;

u8Error=FAT_FileOpen(FILE_NAME,MODIFY);

texto = "/*NOMBRE: Juan Perez                                */  ";
  texto[60] = 13;//CR
texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

texto = "/*--------------------------------------------------------*/  ";
  texto[60] = 13;//CR
```

```
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  texto = "/*Fecha de Nacimiento: 1970            HC: 1234        */  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  texto = "/*-------------------------------------------------------*/  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  texto = "/*FECHA DE INICIO: 10/12/2012          HORA: 00:00:00   */  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  texto = "/*-------------------------------------------------------*/  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));


  texto = "/*ALARMAS:   #1  |  #2  |  #3  |  #4  |  #5  |      */  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  texto = "/*         000030| 000230| 000OFF| 000OFF| 000OFF|      */  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  texto = "/*-------------------------------------------------------*/  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  texto = "/*Intervalo de Registros:  001 minutos                 */  ";
  texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  texto = "/*************************************************************/  ";
texto[60] = 13;//CR
  texto[61] = 10;//LF
  FAT_FileWrite(texto,sizeof(texto));

  //FAT_FileClose();

}

void Print_Hour (void){
```

```
  UINT8 texto[9];
volatile UINT8 u8Error;

  texto[0] =  (Hora.RTC_Hour)/10 + '0';
  texto[1] =  (Hora.RTC_Hour)%10 + '0';
  texto[2] =  ':';
  texto[3] =  (Hora.RTC_Minute)/10 + '0';
  texto[4] =  (Hora.RTC_Minute)%10 + '0';
  texto[5] =  ':';
  texto[6] =  (Hora.RTC_Second)/10 + '0';
  texto[7] =  (Hora.RTC_Second)%10 + '0';
  texto[8] = '_';
  //texto[9] =  13;//CR
  //texto[10] = 10;//LF

u8Error=FAT_FileOpen(FILE_NAME,MODIFY);
  FAT_FileWrite(texto,sizeof(texto));
  FAT_FileClose();
}


void Print_Date (void){

UINT8 texto[11];
  volatile UINT8 u8Error;

  texto[0] =  (OSRtc.Day)/10 + '0';
  texto[1] =  (OSRtc.Day)%10 + '0';
  texto[2] =  '/';
  texto[3] =  (OSRtc.Month)/10 + '0';
  texto[4] =  (OSRtc.Month)%10 + '0';
  texto[5] =  '/';
  texto[6] =  (OSRtc.Year)/1000 + '0';       // 2012/1000=2  , resto=012
  texto[7] =  ((OSRtc.Year)%1000)/100 + '0'; // 012/100 = 0  , resto=12
  texto[8] =  (((OSRtc.Year)%1000)%100)/10 + '0'; //12/10=1 ,resto=2
  texto[9] =  (((OSRtc.Year)%1000)%100)%10 + '0';

  texto[10] = '_';


u8Error=FAT_FileOpen(FILE_NAME,MODIFY);
  FAT_FileWrite(texto,sizeof(texto));
FAT_FileClose();

}


void Print_Estado(int estado,char porcentaje){

  char texto[14];
  //char porcentaje=10;
volatile UINT8 u8Error;
```

```
      Print_Date();
      Print_Hour();

      switch (estado){

        case  ESTADO_ON:
    texto = "ON_          ";
         texto[3] = porcentaje/100 +'0';      //010/100= 0 , resto=10
         texto[4] = (porcentaje%100)/10 +'0'; // 10/10= 1  ,resto=0
         texto[5] = (porcentaje%100)%10 +'0'; //resto=0
    texto[6] = '%';
         break;

        case DISKINESIA:
    texto = "DISKINESIA   ";
         break;

        case MEDICACION:
         texto = "MEDICACION   ";
         break;

        default:
         texto = "Err de estado";
    break;
      }

        texto[12] = 13;//CR
        texto[13] = 10;//LF
        u8Error=FAT_FileOpen(FILE_NAME,MODIFY);
        FAT_FileWrite(texto,sizeof(texto));
    FAT_FileClose();

    }
    //*****************************************************************************************
***//
    // Subrutinas de Lectura de datos en la tarjeta SD para configuración de dispositivo    //
    //*****************************************************************************************
***//
    UINT8 Read_Text[512];

    void Read_Alarms(void){   // Subrutina que lee las alarmas escritas en la SD y las setea

    UINT16 u16BufferSize=1;
      UINT8 Renglon =0;
      UINT16 i=0;
    char j=0;
      UINT8 Alarm[5][7]; // Matriz de Alarmas, 5 posibles

    if ( FAT_FileOpen(FILE_NAME,READ)==FILE_FOUND){        // Abro el archivo de la SD
    // y busco las alarmas a setear
    while(u16BufferSize=FAT_FileRead(Read_Text)){

                for (i=0 ; i<512 ; i++) {
                    if( Read_Text[i]== 10 )
```

```
                Renglon++;
              else if (Renglon == 9){
                 for (j=0; j<6; j++){
                    Alarm[0][j] = Read_Text[i+j+13];//Alarma #1
                    Alarm[1][j] = Read_Text[i+j+21];
                    Alarm[2][j] = Read_Text[i+j+29];
                    Alarm[3][j] = Read_Text[i+j+37];
                    Alarm[4][j] = Read_Text[i+j+45];//Alarma #5
                 }
              }
           }
           //FAT_FileWrite(Read_Text,sizeof(Read_Text));
        }
  }
//   FAT_FileWrite(Alarm[0],sizeof(Alarm[0]));
  FAT_FileClose();
  FAT_FileClose();

  for(i=0; Alarm[i][5]!= 'F' && i<5; i++){              // Solo seteo las alarmas
set_Alarm( (Alarm[i][0]-'0')*10 + (Alarm[i][1]-'0'),// que no estan apagadas
           (Alarm[i][2]-'0')*10 + (Alarm[i][3]-'0'),// 000OFF = apagada
           (Alarm[i][4]-'0')*10 + (Alarm[i][5]-'0'),
           i );
  }
}


void Read_Date ( void ){

  UINT16 u16BufferSize=1;
  UINT8 Renglon =0;
  UINT16 i=0;
  char j=0;
  UINT8 Date[10];
  UINT8 Time[10];

  if ( FAT_FileOpen(FILE_NAME,READ)==FILE_FOUND){       // Abro el archivo de la SD
                                        // y busco la fecha inicial
       while(u16BufferSize=FAT_FileRead(Read_Text)){

           for (i=0 ; i<512 ; i++) {
              if( Read_Text[i]== 10 )
                Renglon++;
              else if (Renglon == 6){
                 for (j=0; j<12; j++){
                    Date[j] = Read_Text[i+j+21];//Fecha de Inicio
                    Time[j] = Read_Text[i+j+46];//Hora de Inicio
                 }
              }
           }
           //FAT_FileWrite(Read_Text,sizeof(Read_Text));
       }
  }
  //FAT_FileWrite(Time,sizeof(Time));
```

```
  FAT_FileClose();
  FAT_FileClose();


  OSRtc.Day  = (Date[0]-'0')*10 + (Date[1]-'0');          //Inicializo la Fecha
  OSRtc.Month= (Date[3]-'0')*10 + (Date[4]-'0');
  OSRtc.Year = (Date[6]-'0')*1000 + (Date[7]-'0')*100 + (Date[8]-'0')*10 + (Date[9]-'0');


  OSRtc.Hour= (Time[2]-'0')*10 + (Time[3]-'0');         //Inicializo la Hora
  OSRtc.Min = (Time[5]-'0')*10 + (Time[6]-'0');
  OSRtc.Sec = (Time[8]-'0')*10 + (Time[9]-'0');

  Hora.RTC_Hour    = (Time[2]-'0')*10 + (Time[3]-'0');
  Hora.RTC_Minute  = (Time[5]-'0')*10 + (Time[6]-'0');
  Hora.RTC_Second  = (Time[8]-'0')*10 + (Time[9]-'0');


}


void Read_State_Interval(void){   // Subrutina que lee el intervalo a setear para los informes de estados
UINT16 u16BufferSize=1;
  UINT8 Renglon =0;
  UINT16 i=0;
  char j=0;
  UINT8 _Interval[4]; // Intervalo

  if ( FAT_FileOpen(FILE_NAME,READ)==FILE_FOUND){        // Abro el archivo de la SD
// y busco las alarmas a setear
while(u16BufferSize=FAT_FileRead(Read_Text)){

            for (i=0 ; i<512 ; i++) {
                if( Read_Text[i]== 10 )
                  Renglon++;
                else if (Renglon == 11){
                    for (j=0; j<4; j++){

                        _Interval[j] = Read_Text[i+j+29];
                    }
                }
            }
        }
  }
  //FAT_FileWrite(_Interval,sizeof(_Interval));
  FAT_FileClose();
  FAT_FileClose();

  State_Interval = (_Interval[0]-'0')*100 + (_Interval[1]-'0')*10 +(_Interval[2]-'0');
  INTERVAL = State_Interval;

}
```

FAT.c

```
/* Includes */
#include "Fat.h"
#include "FAT_Debug.h"

/* File Handlers */
WriteRHandler WHandler;
ReadRHandler RHandler;

/* File Buffers */
UINT8 ag8FATReadBuffer[512];
UINT8 ag8FATWriteBuffer[512];

/* Global Variables */
UINT16 u16FAT_Sector_Size;
UINT16 u16FAT_Cluster_Size;
UINT16 u16FAT_FAT_BASE;
UINT16 u16FAT_Root_BASE;
UINT16 u16FAT_Data_BASE;
UINT16 u16Main_Offset=0;

/*****************************************************************************
***/
UINT32 LWordSwap(UINT32 u32DataSwap)
{
   UINT32 u32Temp;
u32Temp= (u32DataSwap & 0xFF000000) >> 24;
   u32Temp+=(u32DataSwap & 0xFF0000)   >> 8;
   u32Temp+=(u32DataSwap & 0xFF00)     << 8;
   u32Temp+=(u32DataSwap & 0xFF)       << 24;
return(u32Temp);
}

/*****************************************************************************
***/
void FAT_Read_Master_Block(void)
{
   MasterBoot_Entries *pMasterBoot;

   while(ag8FATReadBuffer[0]!= 0xEB && ag8FATReadBuffer[1]!=0x3C &&
ag8FATReadBuffer[2]!=0x90)
   {
      GetPhysicalBlock(u16Main_Offset++,&ag8FATReadBuffer[0]);
   }
   u16Main_Offset--;

   pMasterBoot=(MasterBoot_Entries*)ag8FATReadBuffer;
   u16FAT_Cluster_Size=pMasterBoot->SectorsPerCluster;
   u16FAT_Sector_Size=ByteSwap(pMasterBoot->BytesPerSector);
   u16FAT_FAT_BASE=  u16Main_Offset+ByteSwap(pMasterBoot->ReservedSectors);
   u16FAT_Root_BASE= (ByteSwap(pMasterBoot->SectorsPerFat)<<1)+u16FAT_FAT_BASE;
   u16FAT_Data_BASE= (ByteSwap(pMasterBoot->RootDirectoryEntries)
>>4)+u16FAT_Root_BASE;
}
```

```
    /*****************************************************************************
***/
    void FAT_LS(void)
    {
       UINT8 u8Counter;
       root_Entries *sFileStructure;

       GetPhysicalBlock(u16FAT_Root_BASE,ag8FATReadBuffer);
       sFileStructure = (root_Entries*)&ag8FATReadBuffer[RootEntrySize];
       while(sFileStructure->FileName[0]!=FILE_Clear)
       {
          if(sFileStructure->FileName[0]!=FILE_Erased)
          {
             Terminal_Send_String((UINT8*)"\r\n");
             for(u8Counter=0;u8Counter<8;u8Counter++)
                if(sFileStructure->FileName[u8Counter]!=' ')
                   Terminal_Send_Byte(sFileStructure->FileName[u8Counter]);
             Terminal_Send_Byte('.');
             for(u8Counter=0;u8Counter<3;u8Counter++)
                if(sFileStructure->Extension[u8Counter]!=' ')
                   Terminal_Send_Byte(sFileStructure->Extension[u8Counter]);
          }
          sFileStructure++;
       }
    }

    /*****************************************************************************
***/
    void FAT_FileClose(void)
    {
      root_Entries *sFileStructure;
      UINT16 *pu16FATPointer;
      UINT8 u8Counter;
      UINT32 u32Sector;
      UINT16 u16Offset;

       /* Directory Entry*/
       u32Sector=WHandler.Dir_Entry/(u16FAT_Sector_Size>>5);
       u16Offset=WHandler.Dir_Entry%(u16FAT_Sector_Size>>5);

       GetPhysicalBlock(u16FAT_Root_BASE+u32Sector,ag8FATReadBuffer);
       sFileStructure=(root_Entries*)ag8FATReadBuffer;
       sFileStructure+=u16Offset;

       // FileName
       for(u8Counter=0;u8Counter<8;u8Counter++)
          sFileStructure->FileName[u8Counter]=WHandler.FileName[u8Counter];

       // Entension
       for(u8Counter=0;u8Counter<3;u8Counter++)
          sFileStructure->Extension[u8Counter]=WHandler.Extension[u8Counter];


       // Attributes
```

```
        sFileStructure->Attributes=0x20;
        sFileStructure->_Case=0x18;
        sFileStructure->MiliSeconds=0xC6;

        // Date & Time Information
        sFileStructure->CreationTime=0x2008;
        sFileStructure->CreationDate=0x2136;
        sFileStructure->AccessDate=0x2136;
        sFileStructure->ModificationTime=0x2008;
        sFileStructure->ModificationDate=0x2136;

        // Fat entry and file Size
        sFileStructure->ClusterNumber=ByteSwap(WHandler.BaseFatEntry);

        sFileStructure->SizeofFile=LWordSwap(WHandler.File_Size);

        StorePhysicalBLock(u16FAT_Root_BASE+u32Sector,ag8FATReadBuffer)


        /* FAT Table */
        u32Sector=WHandler.CurrentFatEntry/(u16FAT_Sector_Size>>1);
        u16Offset=WHandler.CurrentFatEntry%(u16FAT_Sector_Size>>1);

        GetPhysicalBlock(u16FAT_FAT_BASE+u32Sector,ag8FATReadBuffer);

        pu16FATPointer=(UINT16*)ag8FATReadBuffer;
        pu16FATPointer+=u16Offset;
        *pu16FATPointer=0xFFFF;     // Write Final Cluster

        StorePhysicalBLock(u16FAT_FAT_BASE+u32Sector,ag8FATReadBuffer)
    }

    /*****************************************************************************
***/
    UINT16 FAT_SearchAvailableFAT(UINT16 u16CurrentFAT)
    {
        UINT16 *pu16DataPointer;
        UINT16 u16FatEntry=0;
        UINT16 u16Sector=0;
        UINT16 u16byteSector;
        UINT16 u16SectorSize=0;

        u16Sector=u16FAT_FAT_BASE;
        while(u16Sector < (((u16FAT_Root_BASE-u16FAT_FAT_BASE)>>1)+u16Main_Offset))
        {
            GetPhysicalBlock(u16Sector++,ag8FATReadBuffer);
            pu16DataPointer=(UINT16*)ag8FATReadBuffer;
            u16byteSector=0;
            u16SectorSize = ((u16FAT_Sector_Size)>> 1);

            while(u16byteSector<u16SectorSize)
            {
                if(*pu16DataPointer==0x0000)
                    if(u16FatEntry!=u16CurrentFAT)
```

```
                return(u16FatEntry);
            pu16DataPointer++;
            u16FatEntry++;
            u16byteSector++;
          }
        }
        return(0);  // Return 0 if no more FAT positions available
    }

    /*****************************************************************************************
***/
    UINT16 FAT_Entry(UINT16 u16FatEntry,UINT16 u16FatValue, UINT8 u8Function)
    {
        UINT16 *pu16DataPointer;

        UINT16 u16Block;
        UINT8 u8Offset;

        u16Block = u16FatEntry / (u16FAT_Sector_Size>>1);
        u8Offset = (UINT8)(u16FatEntry % (u16FAT_Sector_Size >>1));

        GetPhysicalBlock(u16FAT_FAT_BASE+u16Block,ag8FATReadBuffer);
        pu16DataPointer=(UINT16*)ag8FATReadBuffer;
        pu16DataPointer+=u8Offset;

        if(u8Function==NEXT_ENTRY)
          return(ByteSwap(*pu16DataPointer));

        if(u8Function==WRITE_ENTRY)
        {
          *pu16DataPointer=ByteSwap(u16FatValue);
          StorePhysicalBLock(u16FAT_FAT_BASE+u16Block,ag8FATReadBuffer);
          return(0x00);
        }

    }

    /*****************************************************************************************
***/
    void FAT_FileWrite(UINT8 *pu8DataPointer,UINT32 u32Size)
    {
        UINT32 u32SectorToWrite;
        UINT8 *pu8ArrayPointer;
        UINT16 u16TempFat;
        UINT8  u8ChangeSector=1;

        while(u32Size)
        {
          if(u8ChangeSector)
          {
            u32SectorToWrite= u16FAT_Data_BASE + WHandler.ClusterIndex +
(WHandler.CurrentFatEntry-2)*u16FAT_Cluster_Size;
            GetPhysicalBlock(u32SectorToWrite,ag8FATWriteBuffer);
            pu8ArrayPointer=ag8FATWriteBuffer+WHandler.SectorIndex;
```

```
            u8ChangeSector=0;
          }

          while(WHandler.SectorIndex<u16FAT_Sector_Size  &&  u32Size)
          {
            u32Size--;
            WHandler.SectorIndex++;
            WHandler.File_Size++;
            *pu8ArrayPointer++=*pu8DataPointer++;
          }

          StorePhysicalBLock(u32SectorToWrite,ag8FATWriteBuffer);     // Write Buffer to Sector

          /* Check Sector Size */
          if(WHandler.SectorIndex == u16FAT_Sector_Size)
          {
            WHandler.SectorIndex=0;
            WHandler.ClusterIndex++;
            u8ChangeSector=1;
          }

          /* Check Cluster Size */
          if(WHandler.ClusterIndex == u16FAT_Cluster_Size)
          {
            //_BGND;
            WHandler.ClusterIndex=0;
            u16TempFat=FAT_SearchAvailableFAT(WHandler.CurrentFatEntry);
            if(!u16TempFat)
              _NOP;//_BGND;
            (void)FAT_Entry(WHandler.CurrentFatEntry,u16TempFat,WRITE_ENTRY);
            WHandler.CurrentFatEntry=u16TempFat;
            u8ChangeSector=1;
          }
        }
      }

      /*****************************************************************************************
***/
      UINT16 FAT_FileRead(UINT8 *pu8UserBuffer)
      {
        UINT32 u32SectorToRead;
        UINT16 u16BufferSize;

        if(RHandler.File_Size==0)
          return(0);

        //_BGND;


        u32SectorToRead= u16FAT_Data_BASE + ((RHandler.FAT_Entry-
2)*u16FAT_Cluster_Size)+RHandler.SectorOffset;
        GetPhysicalBlock(u32SectorToRead,pu8UserBuffer);
```

```
        if(RHandler.File_Size > u16FAT_Sector_Size)
        {
           RHandler.File_Size-=u16FAT_Sector_Size;
           u16BufferSize=512;
        }
        else
        {
           u16BufferSize=(UINT16)RHandler.File_Size;
           RHandler.File_Size=0;
        }

        if(RHandler.SectorOffset < (u16FAT_Cluster_Size)-1)
           RHandler.SectorOffset++;
        else
        {
             RHandler.SectorOffset=0;
             RHandler.FAT_Entry = FAT_Entry(RHandler.FAT_Entry,0,NEXT_ENTRY); // Get Next FAT
Entry
        }
        return(u16BufferSize);
     }



     /*****************************************************************************************
***/
     void FAT_FileNameOrganizer(UINT8 *pu8FileName,UINT8 *pu8Destiny)
     {
        UINT8 u8Counter=0;

        while(u8Counter<12)
        {
           if(*pu8FileName != '.')
              *pu8Destiny++=*pu8FileName++;
           else
           {
              if(u8Counter<8)
                 *pu8Destiny++=0x20;
              else
                 pu8FileName++;
           }
           u8Counter++;
        }
     }

     /*****************************************************************************************
***/
     UINT8 FAT_FileOpen(UINT8 *pu8FileName,UINT8 u8Function)
     {

        UINT16 u16Temporal;
        UINT8  u8FileName[11];
        UINT8  u8Counter=0;
        UINT8  u8Flag=False;
```

```
        UINT16 u16Index;
        UINT16 u16Block;
        UINT16 u16BlockNum=u16FAT_Data_BASE-u16FAT_Root_BASE;
        UINT8  u8ErrorCode=ERROR_IDLE;
        UINT8  *pu8Pointer;
        UINT16 *pu16FATPointer;
        UINT32 u32Sector;
        UINT16 u16Offset;
        root_Entries *sFileStructure;

        FAT_FileNameOrganizer(pu8FileName,&u8FileName[0]);

        u16Block=0;

    while(u16Block < u16BlockNum && u8ErrorCode==ERROR_IDLE)
    {

        GetPhysicalBlock(u16FAT_Root_BASE+u16Block,ag8FATReadBuffer);
        sFileStructure = (root_Entries*)ag8FATReadBuffer;

        u16Index=0;
        while(u16Index<u16FAT_Sector_Size && u8ErrorCode==ERROR_IDLE)
        {
          /* If Read or Modify Function */
          if(u8Function==READ || u8Function==MODIFY|| u8Function==DELETE ||
u8Function==OVERWRITE)
          {
            if(sFileStructure->FileName[0]==FILE_Clear)
              u8ErrorCode=FILE_NOT_FOUND;

            if(sFileStructure->FileName[0] == u8FileName[0])
            {
              u8Flag=True;
              u8Counter=0;
              while(u8Flag==True && u8Counter < 10)
              {
                u8Counter++;
                if(sFileStructure->FileName[u8Counter] != u8FileName[u8Counter])
                  u8Flag=False;
              }
              if(u8Flag==True)
              {
                /* If Read Function */
                if(u8Function==READ)
                {
                  RHandler.Dir_Entry=(u16Block*EntriesPerBlock)+((u16Index)/RootEntrySize);
                  RHandler.File_Size=LWordSwap(sFileStructure->SizeofFile);
                  RHandler.FAT_Entry=ByteSwap(sFileStructure->ClusterNumber);
                  RHandler.SectorOffset=0;
                  u8ErrorCode=FILE_FOUND;
                }

                /*If Delete Function //Leandro Martinez*/
                else if(u8Function==DELETE)
```

```
                {
                   WHandler.FileName[0] = FILE_Erased;
                   if(WHandler.BaseFatEntry != 0)
                   {
                      u16Temporal=WHandler.BaseFatEntry;
                      do
                      {
                         WHandler.CurrentFatEntry=WHandler.BaseFatEntry;

WHandler.BaseFatEntry=FAT_Entry(WHandler.CurrentFatEntry,0,NEXT_ENTRY);
                         u32Sector=WHandler.CurrentFatEntry/(u16FAT_Sector_Size>>1);
                         u16Offset=WHandler.CurrentFatEntry%(u16FAT_Sector_Size>>1);

                         GetPhysicalBlock(u16FAT_FAT_BASE+u32Sector,ag8FATReadBuffer);

                         pu16FATPointer=(UINT16*)ag8FATReadBuffer;
                         pu16FATPointer+=u16Offset;
                         *pu16FATPointer=0x0000;     // clear FAT entry

                         StorePhysicalBLock(u16FAT_FAT_BASE+u32Sector,ag8FATReadBuffer)

                      }while(WHandler.BaseFatEntry!=0xFFFF);
                      WHandler.BaseFatEntry=u16Temporal;

                      u32Sector=WHandler.CurrentFatEntry/(u16FAT_Sector_Size>>1);
                      u16Offset=WHandler.CurrentFatEntry%(u16FAT_Sector_Size>>1);

                      GetPhysicalBlock(u16FAT_FAT_BASE+u32Sector,ag8FATReadBuffer);

                      pu16FATPointer=(UINT16*)ag8FATReadBuffer;
                      pu16FATPointer+=u16Offset;
                      *pu16FATPointer=0x0000;     // clear FAT entry

                      StorePhysicalBLock(u16FAT_FAT_BASE+u32Sector,ag8FATReadBuffer)

                   }

                    /*  u32Sector=WHandler.BaseFatEntry/(u16FAT_Sector_Size>>1);
                      u16Offset=WHandler.BaseFatEntry%(u16FAT_Sector_Size>>1);

                      GetPhysicalBlock(u16FAT_FAT_BASE+u32Sector,ag8FATReadBuffer);

                      pu16FATPointer=(UINT16*)ag8FATReadBuffer;
                      pu16FATPointer+=u16Offset;
                      *pu16FATPointer=0x0000;     // clear FAT entry

                      StorePhysicalBLock(u16FAT_FAT_BASE+u32Sector,ag8FATReadBuffer)

                      */

                   //WHandler.BaseFatEntry = FILE_Index_Clear;
                   //WHandler.CurrentFatEntry = FILE_Index_Clear;
                   //WHandler.ClusterIndex = FILE_Index_Clear;
```

```
                    }

                    /* If Overwrite Function */
                    else if (u8Function==OVERWRITE)
                    {
                       pu8Pointer=WHandler.FileName;
                       for(u8Counter=0;u8Counter<11;u8Counter++)
                          *pu8Pointer++=u8FileName[u8Counter];
                       WHandler.Dir_Entry=(u16Block*EntriesPerBlock)+((u16Index)/RootEntrySize);
                       WHandler.File_Size=LWordSwap(sFileStructure->SizeofFile);
                       WHandler.BaseFatEntry=ByteSwap(sFileStructure->ClusterNumber);

                       if(WHandler.BaseFatEntry != 0)
                       {
                          u16Temporal=WHandler.BaseFatEntry;
                          do
                          {
                             WHandler.CurrentFatEntry=WHandler.BaseFatEntry;

WHandler.BaseFatEntry=FAT_Entry(WHandler.CurrentFatEntry,0,NEXT_ENTRY);
                          }while(WHandler.BaseFatEntry!=0xFFFF);
                          WHandler.BaseFatEntry=u16Temporal;
                       }
                       else
                       {
                          WHandler.BaseFatEntry=FAT_SearchAvailableFAT(0);
                          WHandler.CurrentFatEntry=WHandler.BaseFatEntry;
                       }
                        /******************/
                        /* Double Check */
    /******************/

                       u8Counter=0;
                       u8ErrorCode=u16FAT_Cluster_Size;
                       while(u8ErrorCode != 0x01)
                       {
                          u8Counter++;
                          u8ErrorCode=u8ErrorCode>>1;
    }

                       u16Temporal=(UINT16)WHandler.File_Size % (u16FAT_Sector_Size<<u8Counter);
                       WHandler.ClusterIndex= u16Temporal/u16FAT_Sector_Size;
                       WHandler.SectorIndex=  0;//u16Temporal%u16FAT_Sector_Size;
                       u8ErrorCode=FILE_FOUND;
                    }

                   /* If Modify Function */
                    else
                    {
                       pu8Pointer=WHandler.FileName;
                       for(u8Counter=0;u8Counter<11;u8Counter++)
                          *pu8Pointer++=u8FileName[u8Counter];
                       WHandler.Dir_Entry=(u16Block*EntriesPerBlock)+((u16Index)/RootEntrySize);
```

```
                        WHandler.File_Size=LWordSwap(sFileStructure->SizeofFile);
                        WHandler.BaseFatEntry=ByteSwap(sFileStructure->ClusterNumber);

                        if(WHandler.BaseFatEntry != 0)
                        {
                           u16Temporal=WHandler.BaseFatEntry;
                           do
                           {
                              WHandler.CurrentFatEntry=WHandler.BaseFatEntry;

WHandler.BaseFatEntry=FAT_Entry(WHandler.CurrentFatEntry,0,NEXT_ENTRY);
                           }while(WHandler.BaseFatEntry!=0xFFFF);
                           WHandler.BaseFatEntry=u16Temporal;
                        }
                        else
                        {
                           WHandler.BaseFatEntry=FAT_SearchAvailableFAT(0);
                           WHandler.CurrentFatEntry=WHandler.BaseFatEntry;
                        }
                         /******************/
                         /* Double Check */
    /******************/

                        u8Counter=0;
                        u8ErrorCode=u16FAT_Cluster_Size;
                        while(u8ErrorCode != 0x01)
                        {
                           u8Counter++;
                           u8ErrorCode=u8ErrorCode>>1;
    }

                        u16Temporal=(UINT16)WHandler.File_Size % (u16FAT_Sector_Size<<u8Counter);
                        WHandler.ClusterIndex= u16Temporal/u16FAT_Sector_Size;
                        WHandler.SectorIndex= u16Temporal%u16FAT_Sector_Size;
                        u8ErrorCode=FILE_FOUND;
                     }

                  }
               }
            }

            /* If Write function */
            if(u8Function==CREATE)
            {
               if(sFileStructure->FileName[0]==FILE_Clear || sFileStructure->FileName[0]==FILE_Erased)
               {
                  pu8Pointer=WHandler.FileName;
                  for(u8Counter=0;u8Counter<11;u8Counter++)
                     *pu8Pointer++=u8FileName[u8Counter];

                  WHandler.Dir_Entry=(u16Block*EntriesPerBlock)+((u16Index)/RootEntrySize);
                  WHandler.File_Size=0;
                  WHandler.BaseFatEntry=FAT_SearchAvailableFAT(0);
                  WHandler.CurrentFatEntry=WHandler.BaseFatEntry;
```

```
                WHandler.ClusterIndex=0;
                WHandler.SectorIndex=0;

                if(WHandler.BaseFatEntry)
                   u8ErrorCode=FILE_CREATE_OK;
                else
                   u8ErrorCode=NO_FAT_ENTRY_AVAIlABLE;
             }
          }

          sFileStructure++;
          u16Index+=RootEntrySize;
       }
       u16Block++;
    }
    if(u16BlockNum==u16Block)
u8ErrorCode=NO_FILE_ENTRY_AVAILABLE;

    return(u8ErrorCode);

}
```

FAT.h

```
#ifndef __Fat__
#define __Fat__

/* Includes */
#include "FslTypes.h"

/************************** HIL ****************************/
/***********************************************************/
/* Includes */
#include "SD.h"     // SD Card Driver


/* Storage HIL */
#define GetPhysicalBlock(A,B)       (void)SD_Read_Block(A,B);
#define StorePhysicalBLock(A,B)     (void)SD_Write_Block(A,B);
/***********************************************************/
/***********************************************************/

/* Macros */
#define ByteSwap(A)     (A=(A<<8)+(A>>8))

/* definitions */
#define MASTER_BLOCK        0x00
#define RootEntrySize       32
#define EntriesPerBlock     16          //Block size / RootEntrySize
#define RHandler_FAT_ENTRIES 8



/*-- Directory Defines --*/
#define FILE_AVAILABLE      0x00
#define FILE_USER           0xFF

#define FILE_Erased         0xE5
#define FILE_Clear          0x00

#define AT_VOLUME           0x01
#define AT_DIRECTORY        0x02
#define AT_HIDDEN           0x04
#define AT_SYSTEM           0x08
#define AT_READONLY         0x10
#define AT_ARCHIVE          0x20
#define FILE_Index_Clear          0x0000


enum
{
   READ,
   CREATE,
   MODIFY,
   DELETE,
   NEXT_ENTRY,
```

```
   WRITE_ENTRY,
   OVERWRITE
};

enum
{
   FILE_FOUND,
   FILE_NOT_FOUND,
   FILE_CREATE_OK,
   NO_FILE_ENTRY_AVAILABLE,
   NO_FAT_ENTRY_AVAIlABLE,
   ERROR_IDLE
};


/* typedef */


typedef struct _ReadHandler
{
   UINT16  FAT_Entry;
   UINT16  SectorOffset;
   UINT16  Dir_Entry;
   UINT32  File_Size;
}ReadRHandler;


typedef struct _WriteRHandler
{
   UINT8   FileName[8];
   UINT8   Extension[3];
   UINT16  Dir_Entry;
   UINT32  File_Size;
   UINT16  BaseFatEntry;
   UINT16  CurrentFatEntry;
   UINT16  SectorIndex;
   UINT16  ClusterIndex;
}WriteRHandler;



/* Root Directory Structure */
typedef struct _root_Entries
{
   UINT8   FileName[8];
   UINT8   Extension[3];
   UINT8   Attributes;
   UINT8   _Case;
   UINT8   MiliSeconds;
   UINT16  CreationTime;
   UINT16  CreationDate;
   UINT16  AccessDate;
   UINT16  Reserved;
   UINT16  ModificationTime;
```

```
   UINT16  ModificationDate;
   UINT16  ClusterNumber;
   UINT32  SizeofFile;
}root_Entries;

/* Master Boot Record */
typedef struct _MasterBoot_Entries
{
   UINT8   JMP_NOP[3];
   UINT8   OEMName[8];
   UINT16  BytesPerSector;
   UINT8   SectorsPerCluster;
   UINT16  ReservedSectors;
   UINT8   FatCopies;
   UINT16  RootDirectoryEntries;
   UINT16  SectorsLess32MB;
   UINT8   MediaDescriptor;
   UINT16  SectorsPerFat;
   UINT16  SectorsPerTrack;
   UINT16  NumberOfHeads;
   UINT32  HiddenSectors;
   UINT32  SectorsInPartition;
   UINT16  LogicalNumberOfPartitions;
   UINT8   ExtendedSignature;
   UINT32  SerialNumber;
   UINT8   VolumeNumber[11];
   UINT8   FatName[8];
   UINT8   ExcecutableCode[448];
   UINT8   ExcecutableMarker[2];
}MasterBoot_Entries;

/*
void FAT_CreateFATLinks(UINT16);
*/
void FAT_LS(void);


/* Prototypes */
UINT32 LWordSwap(UINT32);
void FAT_FileClose(void);
void FAT_Read_Master_Block(void);
UINT8 FAT_FileOpen(UINT8*,UINT8);
void FAT_FileWrite(UINT8*,UINT32);
UINT16 FAT_FileRead(UINT8*);
UINT16 FAT_Entry(UINT16,UINT16,UINT8);
UINT16 FAT_SearchAvailableFAT(UINT16);

#endif /* __Fat__ */
```

usr_entry_V1.c

```
#include "derivative.h"
#include "Bootloader_V1.h"


void _Entry(void)
{
  byte i;
  dword* UserEntryCheck;

 PTGDD_PTGDD0 = 0;             // PTG0 is input
 PTGPE_PTGPE0 = 1;             // internal pullup for PTG0

 // delay some time for oscillator and GPIO stable
 for(i=0;i<3;i++) {

   __RESET_WATCHDOG();
}


 if(PTGD_PTGD0)
 {
    UserEntryCheck = (dword*)USER_ENTRY_ADDRESS;
      if(*UserEntryCheck == 0x4E714EF9)                          // check there is a valid jump op-code
      {
                UserEntryCheck++;                                                    // increment
pointer to next long word
                if(*UserEntryCheck != 0xFFFFFFFF)            // check there is a valid jump vector
                {
           asm (JMP USER_ENTRY_ADDRESS);    // jump to user entry
                }
      }

 }

 SOPT1 = 0x00;                // disable COP
 SOPT2 = 0x00;

 MCGC2 = 0x36;
 while(!(MCGSC & 0x02)){};                        //wait for the OSC stable
 MCGC1 = 0x98;
 while((MCGSC & 0x1C )!= 0x08){}; // external clock is selected
    MCGC3 = 0x48;
    while ((MCGSC & 0x48) != 0x48){};        //wait for the PLL is locked
    MCGC1 = 0x18;
 while((MCGSC & 0x6C) != 0x6C){};

 Bootloader_Main();
}
```

System_init.c

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "Bootloader_V1.h"
#include "usb.h"

void system_init (void);


typedef void (* pFun)(void);
extern asm void _startup(void);


__interrupt void dummy_ISR(void)
{
}

extern interrupt void RTC_ISR(void);
extern interrupt void Timer_Overflow(void);
extern interrupt void Timer_Overflow_TPM2 (void);

__interrupt void USB_ISR(void)
{
  //PollUSB();
  //while(PTDD_PTDD0==0);
  //PTCD_PTCD4 =1;

}
//extern interrupt void USB_ISR(void);

/*************************************************************
* Name: USB_ISR
* Desc: USB interrupt service routine
* Parameter: None
* Return: None
*************************************************************/
/*
__interrupt void USB_ISR(void)
{

   if(INT_STAT_USB_RST && INT_ENB_USB_RST_EN)
    {
       ICP_Reset_Handler();
       //USB_Reset_Handler();
    }

  if(INT_STAT_STALL && INT_ENB_STALL_EN )
      USB_Stall_Handler();


  if(INT_STAT_TOK_DNE && INT_ENB_TOK_DNE_EN)
    {
      //USB_Handler();
      ICP_In_Handler();
```

```
INT_STAT_TOK_DNE = 1;
   }
}
 */

/* Tabla de vectores de interrupción */

void  (* const RAM_vector[])()@REDIRECT_VECTORS= {

(pFun)&dummy_ISR,            // vector_0  INITSP
   (pFun)&dummy_ISR,           // vector_1  INITPC
   (pFun)&dummy_ISR,           // vector_2  Vaccerr
   (pFun)&dummy_ISR,           // vector_3  Vadderr
   (pFun)&dummy_ISR,           // vector_4  Viinstr
   (pFun)&dummy_ISR,           // vector_5  VReserved5
   (pFun)&dummy_ISR,           // vector_6  VReserved6
   (pFun)&dummy_ISR,           // vector_7  VReserved7
   (pFun)&dummy_ISR,           // vector_8  Vprviol
   (pFun)&dummy_ISR,           // vector_9  Vtrace
   (pFun)&dummy_ISR,           // vector_10 Vunilaop
   (pFun)&dummy_ISR,           // vector_11 Vunilfop
   (pFun)&dummy_ISR,           // vector_12 Vdbgi
   (pFun)&dummy_ISR,           // vector_13 VReserved13
   (pFun)&dummy_ISR,           // vector_14 Vferror
   (pFun)&dummy_ISR,           // vector_15 VReserved15
   (pFun)&dummy_ISR,           // vector_16 VReserved16
   (pFun)&dummy_ISR,           // vector_17 VReserved17
   (pFun)&dummy_ISR,           // vector_18 VReserved18
   (pFun)&dummy_ISR,           // vector_19 VReserved19
   (pFun)&dummy_ISR,           // vector_20 VReserved20
   (pFun)&dummy_ISR,           // vector_21 VReserved21
   (pFun)&dummy_ISR,           // vector_22 VReserved22
   (pFun)&dummy_ISR,           // vector_23 VReserved23
   (pFun)&dummy_ISR,           // vector_24 Vspuri
   (pFun)&dummy_ISR,           // vector_25 VReserved25
   (pFun)&dummy_ISR,           // vector_26 VReserved26
   (pFun)&dummy_ISR,           // vector_27 VReserved27
   (pFun)&dummy_ISR,           // vector_28 VReserved28
   (pFun)&dummy_ISR,           // vector_29 VReserved29
   (pFun)&dummy_ISR,           // vector_30 VReserved30
   (pFun)&dummy_ISR,           // vector_31 VReserved31
   (pFun)&dummy_ISR,           // vector_32 Vtrap0
   (pFun)&dummy_ISR,           // vector_33 Vtrap1
   (pFun)&dummy_ISR,           // vector_34 Vtrap2
   (pFun)&dummy_ISR,           // vector_35 Vtrap3
   (pFun)&dummy_ISR,           // vector_36 Vtrap4
   (pFun)&dummy_ISR,           // vector_37 Vtrap5
   (pFun)&dummy_ISR,           // vector_38 Vtrap6
   (pFun)&dummy_ISR,           // vector_39 Vtrap7
   (pFun)&dummy_ISR,           // vector_40 Vtrap8
   (pFun)&dummy_ISR,           // vector_41 Vtrap9
   (pFun)&dummy_ISR,           // vector_42 Vtrap10
   (pFun)&dummy_ISR,           // vector_43 Vtrap11
```

```
    (pFun)&dummy_ISR,           // vector_44 Vtrap12
    (pFun)&dummy_ISR,           // vector_45 Vtrap13
    (pFun)&dummy_ISR,           // vector_46 Vtrap14
    (pFun)&dummy_ISR,           // vector_47 Vtrap15
    (pFun)&dummy_ISR,           // vector_48 VReserved48
    (pFun)&dummy_ISR,           // vector_49 VReserved49
    (pFun)&dummy_ISR,           // vector_50 VReserved50
    (pFun)&dummy_ISR,           // vector_51 VReserved51
    (pFun)&dummy_ISR,           // vector_52 VReserved52
    (pFun)&dummy_ISR,           // vector_53 VReserved53
    (pFun)&dummy_ISR,           // vector_54 VReserved54
    (pFun)&dummy_ISR,           // vector_55 VReserved55
    (pFun)&dummy_ISR,           // vector_56 VReserved56
    (pFun)&dummy_ISR,           // vector_57 VReserved57
    (pFun)&dummy_ISR,           // vector_58 VReserved58
    (pFun)&dummy_ISR,           // vector_59 VReserved59
    (pFun)&dummy_ISR,           // vector_60 VReserved60
    (pFun)&dummy_ISR,           // vector_61 Vunsinstr
    (pFun)&dummy_ISR,           // vector_62 VReserved62
    (pFun)&dummy_ISR,           // vector_63 VReserved63
    (pFun)&dummy_ISR,           // vector_64 Virq
    (pFun)&dummy_ISR,           // vector_65 Vlvd
    (pFun)&dummy_ISR,           // vector_66 Vlol
    (pFun)&dummy_ISR,           // vector_67 Vspi1
    (pFun)&dummy_ISR,           // vector_68 Vspi2
    (pFun)&USB_ISR,           // vector_69 Vusb
    (pFun)&dummy_ISR,           // vector_70 VReserved70
    (pFun)&dummy_ISR,           // vector_71 Vtpm1ch0
    (pFun)&dummy_ISR,           // vector_72 Vtpm1ch1
    (pFun)&dummy_ISR,           // vector_73 Vtpm1ch2
    (pFun)&dummy_ISR,           // vector_74 Vtpm1ch3
    (pFun)&dummy_ISR,           // vector_75 Vtpm1ch4
(pFun)&dummy_ISR,           // vector_76 Vtpm1ch5
    (pFun)&Timer_Overflow,          // vector_77 Vtpm1ovf
    (pFun)&dummy_ISR,           // vector_78 Vtpm2ch0
    (pFun)&dummy_ISR,           // vector_79 Vtpm2ch1
    (pFun)&dummy_ISR,           // vector_80 Vtpm2ovf
    (pFun)&dummy_ISR,           // vector_81 Vsci1err
(pFun)&dummy_ISR,           // vector_82 Vsci1rx
    (pFun)&dummy_ISR,           // vector_83 Vsci1tx
    (pFun)&dummy_ISR,           // vector_84 Vsci2err
    (pFun)&dummy_ISR,           // vector_85 Vsci2rx
    (pFun)&dummy_ISR,           // vector_86 Vsci2tx
    (pFun)&dummy_ISR,           // vector_87 Vkeyboard
    (pFun)&dummy_ISR,           // vector_88 Vadc
    (pFun)&dummy_ISR,           // vector_89 Vacmpx
    (pFun)&dummy_ISR,           // vector_90 Viic1x
    (pFun)&RTC_ISR,           // vector_91 Vrtc
    (pFun)&dummy_ISR,           // vector_92 Viic2x
    (pFun)&dummy_ISR,           // vector_93 Vcmt
    (pFun)&dummy_ISR,           // vector_94 Vcanwu
    (pFun)&dummy_ISR,           // vector_95 Vcanerr
    (pFun)&dummy_ISR,           // vector_96 Vcanrx
    (pFun)&dummy_ISR,           // vector_97 Vcantx
```

```
   (pFun)&dummy_ISR,          // vector_98 Vrnga
   (pFun)&dummy_ISR,          // vector_99 VReserved99
   (pFun)&dummy_ISR,          // vector_100 VReserved100
   (pFun)&dummy_ISR,          // vector_101 VReserved101
   (pFun)&dummy_ISR,          // vector_102 VReserved102
   (pFun)&dummy_ISR,          // vector_103 VReserved103
   (pFun)&dummy_ISR,          // vector_104 VL7swi
   (pFun)&dummy_ISR,          // vector_105 VL6swi
   (pFun)&dummy_ISR,          // vector_106 VL5swi
   (pFun)&dummy_ISR,          // vector_107 VL4swi
   (pFun)&dummy_ISR,          // vector_108 VL3swi
   (pFun)&dummy_ISR,          // vector_109 VL2swi
   (pFun)&dummy_ISR,          // vector_110 VL1swi
};


const byte _UserEntry[] @ USER_ENTRY_ADDRESS = {
0x4E,
 0x71,
 0x4E,
 0xF9          //asm NOP(0x4E71), asm JMP(0x4EF9)
};

//const byte NVPROT_INIT @0x0000040D  = 0xFB;    // 0x00000000-0x00000FFF are protected

void  (* const _UserEntry2[])()@(USER_ENTRY_ADDRESS+4)=
{
 _startup,
};


// Inicializacion del sistema. Se copia la tabla de interrupciones a
// la RAM.
void system_init (void)
{
 // Aca se copia la tabla de interrupciones a la RAM.
/* !! This section needs to be here to redirect interrupt vectors !! */
 dword *pdst,*psrc;
 byte i;

 asm (move.l  #0x00800000,d0);
 asm (movec   d0,vbr);

 pdst=(dword*)0x00800000;
 psrc=(dword*)&RAM_vector;

 for (i=0;i<111;i++,pdst++,psrc++)
{
   *pdst=*psrc;
 }
}
```

FslTypes.h

```
#ifndef __Definitions__
#define __Definitions__


/* Typedefs */
typedef unsigned char   UINT8;                  /*unsigned 8 bit definition */
typedef unsigned short  UINT16;                 /*unsigned 16 bit definition*/
typedef unsigned long   UINT32;                 /*unsigned 32 bit definition*/
typedef signed char     INT8;           /*signed 8 bit definition */
typedef short                   INT16;                  /*signed 16 bit definition*/
typedef long int          INT32;                /*signed 32 bit definition*/

/* Definitions */
#define   _OUT      1
#define   _IN       0

#define   OUTPUT      1
#define   INPUT       0

//#define   ENABLE      1
//#define   DISABLE     0

enum ISR
{
   SCI_Flag,
   ADC_Flag
};

/* Macros */
#define FLAG_SET(BitNumber, Register)       (Register |=(1<<BitNumber))
#define FLAG_CLR(BitNumber, Register)       (Register &=~(1<<BitNumber))
#define FLAG_CHK(BitNumber, Register)       (Register & (1<<BitNumber))


#define    _BGND    //asm(BGND)
#define    _NOP     asm(NOP)
#define    _WAIT    asm(WAIT)
#define    False    0x00
#define    True     0x01


#endif /* __Definitions__ */
```

Target.c

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "target.h"
#include "hcc_types.h"
#include "derivative.h"
#include "FslTypes.h"

static void init_clock(void)
{
 /* Assume 12MHz external clock source connected. */

  /* In order to use the USB we need to enter PEE mode and MCGOUT set to 48 MHz.
     Out of reset MCG is in FEI mode. */

  /**** Moving from FEI (FLL engaged internal) to PEE (PLL engaged external) mode. */
  /* switch from FEI to FBE (FLL bypassed external) */
    /* enable external clock source */
  MCGC2 = MCGC2_HGO_MASK       /* oscillator in high gain mode */
        | MCGC2_EREFS_MASK   /* because crystal is being used */
        | MCGC2_RANGE_MASK   /* 12 MHz is in high freq range */
        | MCGC2_ERCLKEN_MASK;     /* activate external reference clock */
  while (MCGSC_OSCINIT == 0)
    ;
    /* select clock mode */
  MCGC1 = (2<<6)         /* CLKS = 10 -> external reference clock. */
        | (3<<3);      /* RDIV = 3 -> 12MHz/8=1.5 MHz */

    /* wait for mode change to be done */
  while (MCGSC_IREFST != 0)
    ;
  while (MCGSC_CLKST != 2)
    ;

  /* switch from FBE to PBE (PLL bypassed internal) mode */
  MCGC3=MCGC3_PLLS_MASK
       | (8<<0);     /* VDIV=6 -> multiply by 32 -> 1.5MHz * 32 = 48MHz */
  while(MCGSC_PLLST != 1)
    ;
  while(MCGSC_LOCK != 1)
    ;
  /* finally switch from PBE to PEE (PLL enabled external mode) */
  MCGC1 = (0<<6)         /* CLKS = 0 -> PLL or FLL output clock. */
        | (3<<3);      /* RDIV = 3 -> 12MHz/8=1.5 MHz */
  while(MCGSC_CLKST!=3)
    ;

  /* Now MCGOUT=48MHz, BUS_CLOCK=24MHz */
}


void init_board(void)
{
  /* Configure LED io pins to be outputs */
/*  PTEDD=(1<<3) | (1<<2);
```

```
  PTFDD=(1<<5) | (1<<1) | (1<<0);
  PTCDD=(1<<4) | (1<<2);
  PTDDD=(1<<2);
*/
  /* Enable internal pull-ups on port E pins to get switches working. */
/*  PTGPE=0x0F;
  PTGDD=0x00;
*/

  ////////////////////////////////////////////////////
  PTBDD_PTBDD4 = _IN;   // Initialize Button in

  PTCDD_PTCDD4 = _OUT;  //Pin1
  PTCD_PTCD4 =0;

  PTEDD_PTEDD2 = _OUT;  //Testigo Flash RTC
  PTED_PTED2 = 0;

  PTBDD_PTBDD5 = _OUT;  //Testigo SD OK
  PTBD_PTBD2 = 0;

  PTDDD_PTDDD2 = _OUT;  //BUZZER
PTDD_PTDD2 =0;

  PTDDD_PTDDD1 = _OUT;  //Testigo Lectura invalida de SD
PTDD_PTDD1 =0;

  PTEDD_PTEDD3 = _OUT;  //Testigo Prueba
  PTED_PTED3 = 0;


  //LEDS del potenciometro//
  PTFDD_PTFDD0 = _OUT; //LED_1
  PTFDD_PTFDD1 = _OUT;
  PTFDD_PTFDD4 = _OUT;
  PTFDD_PTFDD5 = _OUT;
  PTEDD_PTEDD0 = _OUT;

  // Potenciometro //
  PTCDD_PTCDD3 = _IN; //POTE_1
  PTCDD_PTCDD2 = _IN;
  PTCDD_PTCDD1 = _IN;
  PTCDD_PTCDD0 = _IN;
  PTGDD_PTGDD3 = _IN;


  //----- TECLADO ------------//
  PTEDD_PTEDD4= _IN; //BOTON_ESTADO
  PTEDD_PTEDD5= _IN; //BOTON_DISKINESIA
  PTEDD_PTEDD6= _IN; //BOTON_MEDICACION

//////////////////////////////////////////////////////
```

```
}

void _irq_restore (hcc_imask ip)
{
  if(ip)
  {
    /* Disable interrupts */
//    asm ("sei");
      DisableInterrupts;
  }
  else
  {
    /* Enable interrupts */
//    asm("cli");
      EnableInterrupts;
  }
}



hcc_imask _irq_disable (void)
{
  hcc_u8 r;

  DisableInterrupts;

  //asm ("tpa;");     /* transfer CCR to A */
  //asm ("sei;");     /* disable interrupts */
  //asm ("and #8;");  /* mask I bit of CCR */
//asm ("sta r;");

  return(r);
}



/*****************************************************************************/
void hw_init(void)
{
  /* Disable watchdog. */
  SOPT1_COPT=0;

  init_clock();
  init_board();
  //_irq_restore(0);
}


void Buzzer_Beep (void){

byte j;
    word i;

    BUZZER_ON;
```

```
   for(i=0;i<500;i++){            // delay
      for(j=0;j<255;j++) {
         asm(nop);
         asm(nop);
         asm(nop);
      }
   }
   BUZZER_OFF;

}

/****************************** END OF FILE **********************************/
```

Target.h

```
#ifndef _TARGET_H_
#define _TARGET_H_

#include "hcc_types.h"
#include "derivative.h"
#include "FslTypes.h"

extern void _irq_restore (hcc_imask ip);
extern hcc_imask _irq_disable (void);
extern void hw_init(void);
void Buzzer_Beep (void);

#define SW1_ACTIVE()  (PTGD_PTGD0 == 0)
#define SW2_ACTIVE()  (PTGD_PTGD1 == 0)

//----- LEDS del potenciometro ------//
#define LED1_ON       (PTFD_PTFD0=1)
#define LED2_ON       (PTFD_PTFD1=1)
#define LED3_ON       (PTFD_PTFD4=1)
#define LED4_ON       (PTFD_PTFD5=1)
#define LED5_ON       (PTED_PTED0=1)

#define LED1_OFF      (PTFD_PTFD0=0)
#define LED2_OFF      (PTFD_PTFD1=0)
#define LED3_OFF      (PTFD_PTFD4=0)
#define LED4_OFF      (PTFD_PTFD5=0)
#define LED5_OFF      (PTED_PTED0=0)

#define LED1_TGL      (PTFD_PTFD0^=1)
#define LED2_TGL      (PTFD_PTFD1^=1)
#define LED3_TGL      (PTFD_PTFD4^=1)
#define LED4_TGL      (PTFD_PTFD5^=1)
#define LED5_TGL      (PTED_PTED0^=1)

//----- Potenciometro   ------------//
#define POTE_1      PTCD_PTCD3
#define POTE_2      PTCD_PTCD2
#define POTE_3      PTCD_PTCD1
#define POTE_4      PTCD_PTCD0
#define POTE_5      PTGD_PTGD3

//----- TECLADO   ------------------//
#define BOTON_ESTADO       PTED_PTED4
#define BOTON_DISKINESIA    PTED_PTED5
#define BOTON_MEDICACION    PTED_PTED6

//----- ESTADOS   ------------------//
#define ESTADO_ON   1
#define ESTADO_OFF  0
#define DISKINESIA  3
#define MEDICACION  2
```

```
//----- LED PIN1----------------------//
#define LED_PIN1_ON     (PTCD_PTCD4=1)
#define LED_PIN1_OFF    (PTCD_PTCD4=0)
#define LED_PIN1_TGL    (PTCD_PTCD4^=1)

//----- BUZZER-----------------------//
#define BUZZER_ON     (PTDD_PTDD2=1)
#define BUZZER_OFF    (PTDD_PTDD2=0)
#define BUZZER_TGL    (PTDD_PTDD2^=1)


typedef struct {
      char   Estado;
      char   Porcentaje;
      //UINT8   flag;
} ESTADO_STRUCT;

#endif
/******************************** END OF FILE ***********************************/
```

Config.h

```
#ifndef __Config__
#define __Config__


/****** User Defines ****************/
#define DATA_SAMPLES    2     /* How many samples will read before store the buffer
                        in the SD Card */

//#define SAMPLE_TIME_05_SEG      /* SAMPLE_TIME_05_SEG Sampling Interval 0.5 seg*/
                         /* SAMPLE_TIME_1_SEG Sampling Interval 1 seg*/
                         /* SAMPLE_TIME_2_SEG Sampling Interval 2 seg*/
                         /* SAMPLE_TIME_3_SEG Sampling Interval 3 seg*/
                         /* SAMPLE_TIME_6_SEG Sampling Interval 6 seg*/


#define FILE_NAME   "LOG.TXT"  /* LOG file name */


/******* Please Do not Modify ********/
#define SAMPLES_WRITE   DATA_SAMPLES *2


#endif /* __Config__ */
```

Tecaldo.c

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h"
#include "timer.h"
#include "hcc_types.h"
#include "target.h"
#include "alarma.h"

#define Validate_Counter 7   // Constante de validacion = 7*Tiempo lectura de teclado

/*Variables*/
static ESTADO_STRUCT Estado_Actual;
static UINT8 Hold; //Para validar un Estado hay que presionar el boton por lo menos por 700mseg


void Leer_Tecla (void);
void Delay_Tecla (void);
void Destello_potenciometro (char leds);
void Validate_State (char leds );

void Leer_Tecla(void){    //Funcion que lee el teclado

if(BOTON_ESTADO==1){
   BUZZER_ON;
LED_PIN1_OFF;
   Print_Estado(ESTADO_ON,Estado_Actual.Porcentaje);
   BUZZER_OFF;
                    //Delay de informe de estado,
                    //para evitar que se imprima en la SD
                    //mas de un informe necesario, por problema de parkinson
   Destello_potenciometro (5);
   LED_PIN1_ON;
   Alarm_Flag = OFF;    //Apago la alarma
}

 if(BOTON_DISKINESIA==1 && Hold ==Validate_Counter){
   BUZZER_ON;
   LED_PIN1_OFF;
Print_Estado(DISKINESIA,0);
   BUZZER_OFF;              // Delay de imforme de estado
Destello_potenciometro (5);
   LED_PIN1_ON;
 }else if(BOTON_DISKINESIA==1 && Hold <Validate_Counter)
   Hold++;

 if(BOTON_MEDICACION==1&& Hold ==Validate_Counter){
   BUZZER_ON;
   LED_PIN1_OFF;
Print_Estado(MEDICACION,0);
     Alarm_Flag = OFF;    //Apago la alarma
BUZZER_OFF;
   Delay_Tecla();
   LED_PIN1_ON;
 }else if(BOTON_MEDICACION==1 && Hold <Validate_Counter)
```

```
    Hold++;

  if(POTE_1==1&& Hold ==Validate_Counter){
    Buzzer_Beep();
    LED1_ON,LED2_OFF,LED3_OFF,LED4_OFF,LED5_OFF;
Estado_Actual.Estado = ESTADO_ON;
Estado_Actual.Porcentaje = 0;
    Hold =0;
    Validate_State (1);
  }else if(POTE_1==1 && Hold <Validate_Counter)
    Hold++;


  if(POTE_2==1 && Hold ==Validate_Counter){
    Buzzer_Beep();
    LED1_ON,LED2_ON,LED3_OFF,LED4_OFF,LED5_OFF;
Estado_Actual.Estado = ESTADO_ON;
Estado_Actual.Porcentaje=25;
    Hold =0;
    Validate_State (2);
  }else if(POTE_2==1 && Hold <Validate_Counter)
    Hold++;


  if(POTE_3==1&& Hold ==Validate_Counter){
    Buzzer_Beep();
    LED1_ON,LED2_ON,LED3_ON,LED4_OFF,LED5_OFF;
Estado_Actual.Estado = ESTADO_ON;
    Estado_Actual.Porcentaje=50;
    Validate_State (3);
}else if(POTE_3==1 && Hold <Validate_Counter)
    Hold++;


  if(POTE_4==1 && Hold ==Validate_Counter){
    Buzzer_Beep();
    LED1_ON,LED2_ON,LED3_ON,LED4_ON,LED5_OFF;
Estado_Actual.Estado = ESTADO_ON;
    Estado_Actual.Porcentaje=75;
    Validate_State (4);
}else if(POTE_4==1 && Hold <Validate_Counter)
    Hold++;


  if(POTE_5==1&& Hold ==Validate_Counter){
    Buzzer_Beep();
    //LED1_ON,LED2_ON,LED3_ON,LED4_ON,LED5_ON;
Estado_Actual.Estado = ESTADO_ON;
    Estado_Actual.Porcentaje=100;
    Validate_State (5);
}else if(POTE_5==1 && Hold <Validate_Counter)
    Hold++;
```

```
  else if(!(BOTON_MEDICACION ^= BOTON_DISKINESIA ^= POTE_1 ^= POTE_2 ^= POTE_3 ^=
POTE_4 ^= POTE_5))Hold=0;

}


void Delay_Tecla (void){

   byte j;
   word i;

   for(i=0;i<25000;i++){            // delay
      for(j=0;j<255;j++) {
         asm(nop);
         asm(nop);
         asm(nop);
      }
   }
}

void Destello_potenciometro (char leds){

   byte j;
   word i;
   char index=3;


   while (index--){

    switch (leds){
      case 1:
        LED1_ON,LED2_OFF,LED3_OFF,LED4_OFF,LED5_OFF;
        break;
      case 2:
        LED1_ON,LED2_ON,LED3_OFF,LED4_OFF,LED5_OFF;
        break;
      case 3:
        LED1_ON,LED2_ON,LED3_ON,LED4_OFF,LED5_OFF;
        break;
      case 4:
        LED1_ON,LED2_ON,LED3_ON,LED4_ON,LED5_OFF;
        break;
      case 5:
        LED1_ON,LED2_ON,LED3_ON,LED4_ON,LED5_ON;
        break;
    }
      for(i=0;i<10000;i++){            // delay
         for(j=0;j<255;j++) {
            asm(nop);
            asm(nop);
            asm(nop);
         }
      }
      LED1_OFF,LED2_OFF,LED3_OFF,LED4_OFF,LED5_OFF;
```

```
      for(i=0;i<10000;i++){            // delay
         for(j=0;j<255;j++) {
            asm(nop);
            asm(nop);
            asm(nop);
         }
      }
   }
   LED1_OFF,LED2_OFF,LED3_OFF,LED4_OFF,LED5_OFF;
}


void Validate_State (char leds ){
   BUZZER_ON;
   LED_PIN1_OFF;
Print_Estado(ESTADO_ON,Estado_Actual.Porcentaje);
   BUZZER_OFF;
                  //Delay de informe de estado,
                  //para evitar que se imprima en la SD
                  //mas de un informe necesario, por problema de parkinson
   Destello_potenciometro (leds);
   LED_PIN1_ON;
   Alarm_Flag = OFF;    //Apago la alarma
 }
```

Alarma.h

```
#ifndef _Alarma_H_
#define _Alarma_H_


typedef struct _ALARMA_STRUCT{

        UINT8 RTC_Second;       ///< Seconds of the clock
        UINT8 RTC_Minute;       ///< Minutes of the clock
        UINT8 RTC_Hour;             ///< Hours of the clock
  UINT8 Alarm_State;

} ALARMA_STRUCT;

extern UINT8 Alarm_Flag;
void Init_Alarms(void);
void set_Alarm(UINT8 hora_Alm,UINT8 min_Alm,UINT8 seg_Alm,UINT8 index);
//index = cantidad de alarmas a crear

UINT8 Verify_Alarm (UINT8 index );
void Encender_Alarma (UINT16 mseg);
void Delay_Alarm (void);

void Verify_Inform(UINT16 counter);

#endif
```

Alarma.c

```
#include "RTC.h"
#include "OS_RTC.h"
#include "timer.h"
#include "alarma.h"
#include "target.h"

ALARMA_STRUCT Alarma[4]; //Estructura de 5 alarmas posibles
UINT8 Alarm_Flag =OFF;

/*********************************************************/
void Init_Alarms(void)
{
   char i=0;
  for(i=0; i<5 ; i++){

     Alarma[i].RTC_Hour   = 0;
     Alarma[i].RTC_Minute = 0;
     Alarma[i].RTC_Second = 0;
     Alarma[i].Alarm_State= OFF;
  }

}

void set_Alarm(UINT8 hora_Alm,UINT8 min_Alm,UINT8 seg_Alm,UINT8 index)
{

     Alarma[index].RTC_Hour   = hora_Alm;
     Alarma[index].RTC_Minute = min_Alm;
Alarma[index].RTC_Second = seg_Alm;
     Alarma[index].Alarm_State= ON;

}


UINT8 Verify_Alarm (UINT8 index ){

  if( Alarma[index].RTC_Hour   == Hora.RTC_Hour   &&
Alarma[index].RTC_Minute == Hora.RTC_Minute &&
Alarma[index].RTC_Second == Hora.RTC_Second)
  {
     Alarma[index].Alarm_State= ON;
Alarm_Flag = ON;
     Encender_Alarma (500);   //Alarmas de 500mseg
}
  return(Alarma[index].Alarm_State);
}

void Encender_Alarma (UINT16 mseg){ //Enciendo la Alarma. y hasta que no se tome
                        //la medicación, NO se continua
    while (Alarm_Flag == ON){

LED_PIN1_TGL;
     BUZZER_TGL;
```

```
Delay_mS(mseg); //Delay de 500 mSeg
     //Delay_Alarm();
   }
   LED_PIN1_ON;
   BUZZER_OFF;
}

void Delay_Alarm (void){
   byte j;
   word i;

   for(i=0;i<1000;i++){            // delay
      for(j=0;j<255;j++) {
         asm(nop);
asm(nop);
         asm(nop);
      }
   }
}

void Verify_Inform(UINT16 counter){   //Verifica si se cumplio el tiempo necesario
                              //para pedirle imforme de estado al paciente
if (counter==0){
     Alarm_Flag =ON;
     Encender_Alarma (100);
   }
}



/*********************************************************/
```

Timer.c

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h"
#include "timer.h"
#include "hcc_types.h"
#include "target.h"
#include "Alarma.h"


//Variables Staticas para Timers
static hcc_u16 Timer_Leer_Teclado;
static hcc_u16 Timer_Toggle_Led1;
static hcc_u16 Timer_Delay;
static hcc_u16 Delay_Flag;

hcc_u16 Delay_TPM;

void Leer_Tecla (void);

void start_mS_timer(hcc_u16 delay)
{

  TPM1MOD = (delay*6);
  TPM1C0V = (delay*6);
  TPM1CNT = 0;
  // TPM1SC_TOIE  = 1;       /* enable the timer overflow interrupt */
  // TPM1C0SC_CH0IE = 1;

  //TPM1SC_CPWMS = 0;
  //TPM1C0SC_MS0x = 1;
  //TPM1C0SC_ELS0x = 1;    /* toggle output */

  TPM1SC_PS =   7;        /* prescalor 128 */
  TPM1SC_CLKSA = 0;
  TPM1SC_CLKSB = 1;       /* select Fixed clock as source clock */


  TPM1SC_TOIE = 1;      // enable the timer overflow interrupt */

  Delay_TPM = delay;
  Init_Timers();
//-------------------------------------------------------//

}

interrupt void Timer_Overflow(void)
{
   TPM1SC_TOF = 0;     // clear flag

   Decrement_Timers();
}

/*
 * FUNCTION NAME: check_mS_timer:
```

```
 * DESCRIPTION:   to check the timer if it is expired.
 * INPUT PARAMETER:
 *    none
 * OUTPUT PARAMETER:
 * RETURN:
 *    0                  -          timer is not expired
 *    1                  -          timer is expired
 */
hcc_u8 check_mS_timer(void)
{
  if (!TPM1SC_TOF) {
    return (0);
  }
  TPM1SC_TOF = 0;
  //TPM1SC_CLKSx = 0;   /* disable the timer */
  return(1);

}

void Init_Timers (void){

Timer_Leer_Teclado = 100/Delay_TPM;
Timer_Toggle_Led1 =  500/Delay_TPM;

}


void Decrement_Timers( void ){

     //-------------------------------------------------------------//
    if(Timer_Leer_Teclado ==0){
Leer_Tecla();            //Leo El teclado cada 100mSeg
      Timer_Leer_Teclado = 100/Delay_TPM;
}else{
      Timer_Leer_Teclado--;
    }
    //-------------------------------------------------------------//
    if(Timer_Toggle_Led1 ==0){
      PTED_PTED2 ^=1;        //Toggle Led
      Timer_Toggle_Led1 = 500/Delay_TPM;
    }else{
      Timer_Toggle_Led1--;
    }
    //-------------------------------------------------------------//

   if( (Delay_Flag == ON) && (Timer_Delay!=0)){
        Timer_Delay--;

    }else if( Delay_Flag == ON && Timer_Delay==0){
Delay_Flag = OFF;
    }
   //-----------------------------------------------//

}
```

```
void Delay_mS(hcc_u16 delay){

    Delay_Flag= ON;
    Timer_Delay = delay/Delay_TPM;

    while(Delay_Flag == ON){
          asm(nop);
    }
}
```

/****************************** END OF FILE ***********************************/

Timer.h

```
/*****************************************************************************/
#ifndef _TIMER_H_
#define _TIMER_H_

#include "hcc_types.h"

#define ON 1
#define OFF 0

void start_mS_timer(hcc_u16 delay);
hcc_u8 check_mS_timer(void);
void Init_Timers (void);
void Decrement_Timers(void);
void Delay_mS(hcc_u16 delay);

#endif

/******************************** END OF FILE ********************************/
```

- **MCF51JM128 ColdFire Integrated Microcontroller Reference Manual, Rev. 2**

1.2 Block Diagram

Figure 1 shows the connections between the MCF51JM128 series pins and modules.

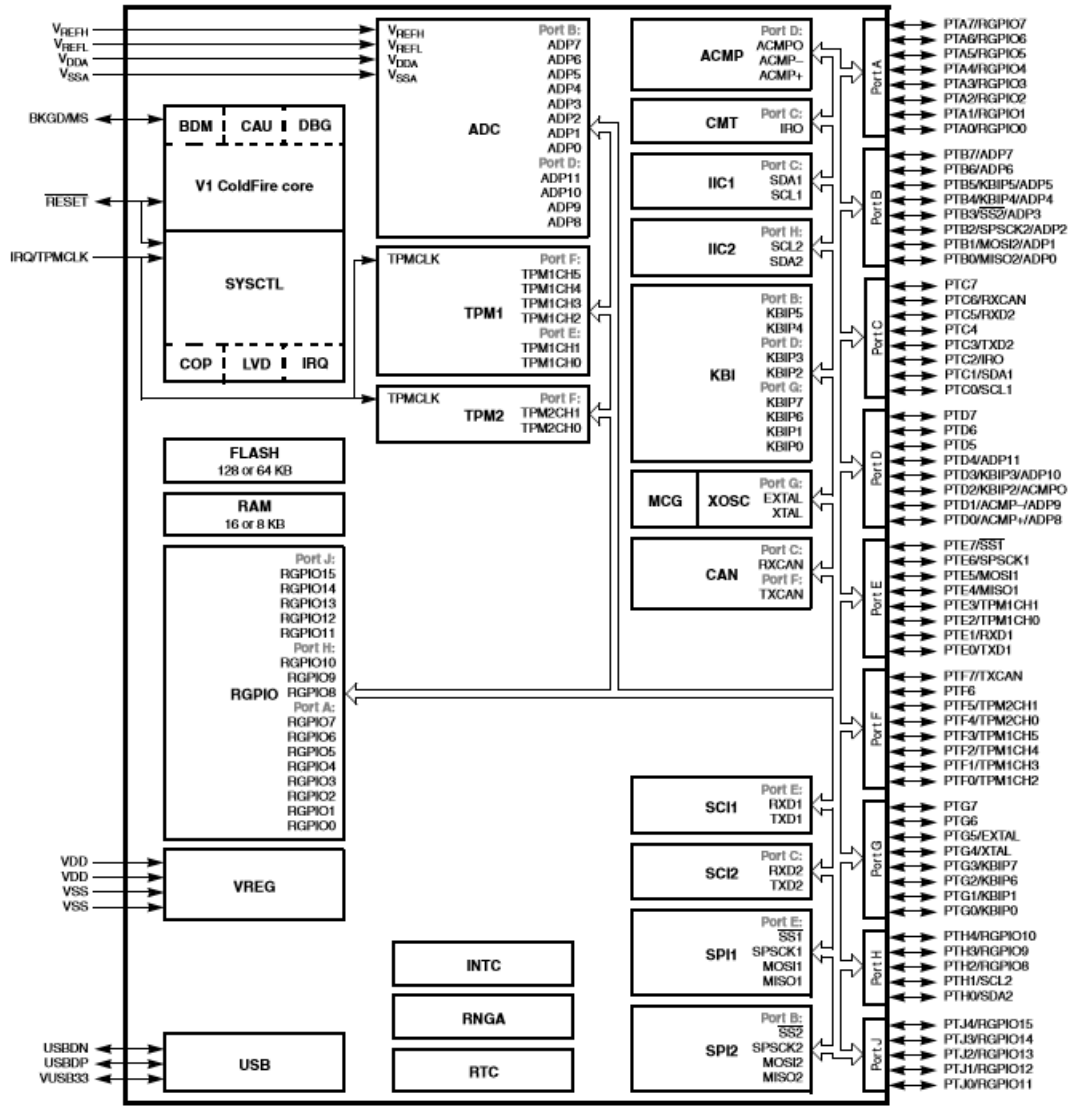
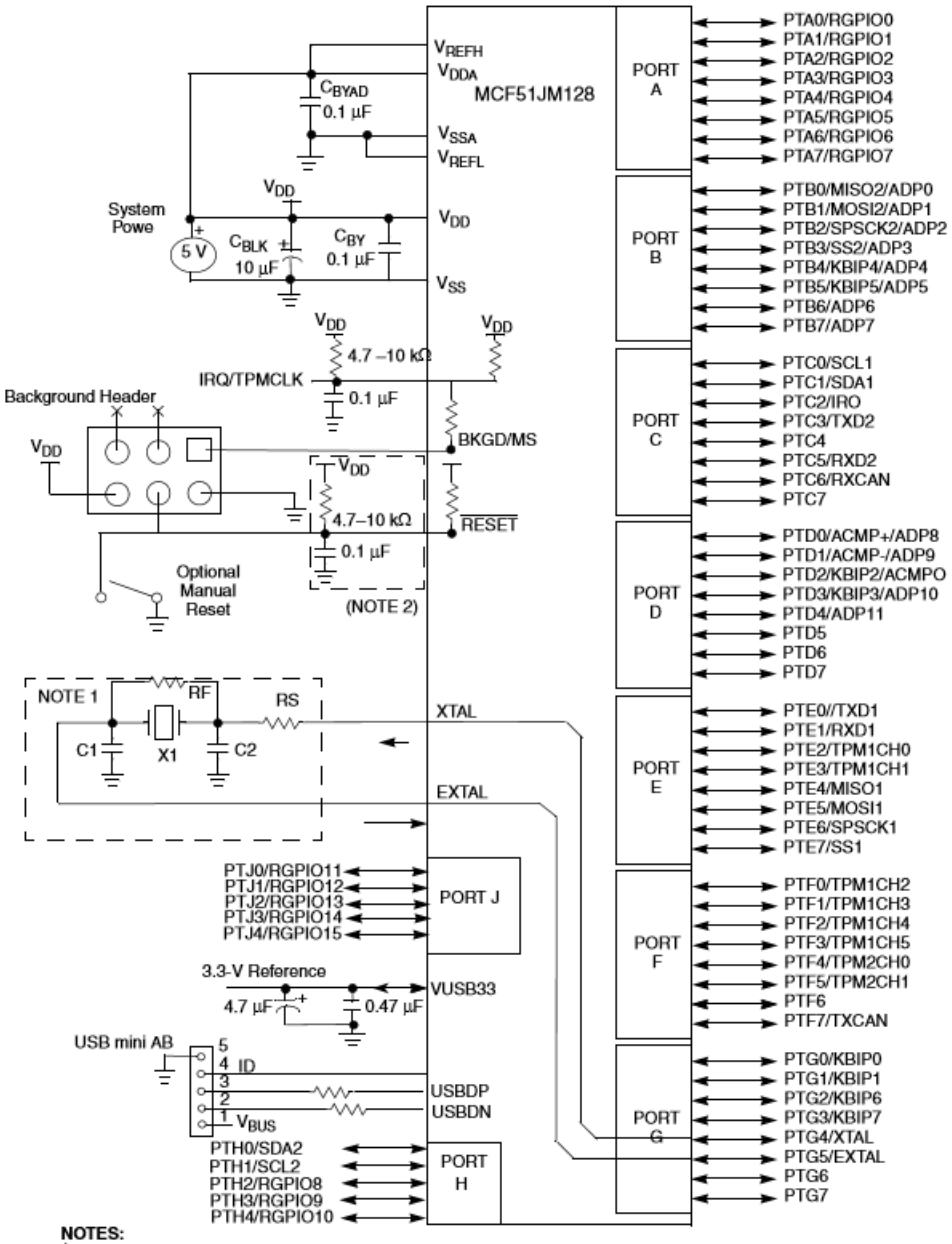


Figure 1. MCF51JM128 Block Diagram



NOTES:

1 $\overline{\text{RESET}}$ features have optional internal pullup device.

2 RC filter on $\overline{\text{RESET}}$ and IRQ pins recommended for noisy environments.

Figure 2-4. Basic System Connections

MCF51JM128 Family Configurations

Figure 4 shows the pinout of the 44-pin LQFP.

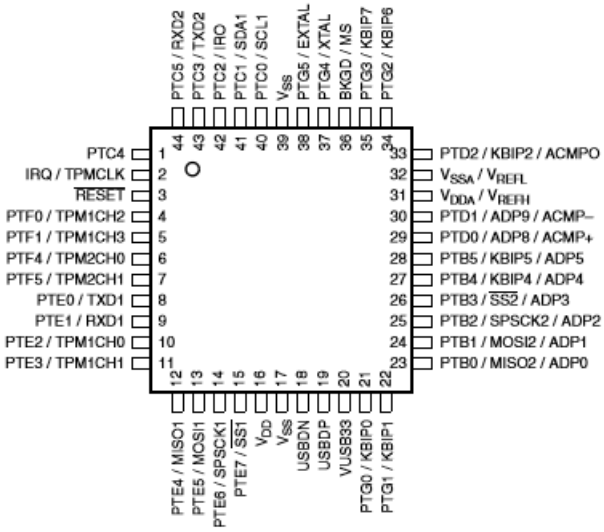


Figure 4. 44-pin LQFP

Table 4 shows the package pin assignments.

Table 4. Pin Assignments by Package and Pin Sharing Priority

Pin Number			<-- Lowest **Priority** --> Highest		
80	**64**	**44**	**Port Pin**	**Alt 1**	**Alt 2**
1	1	1	PTC4		—
2	2	2	—	IRQ	TPMCLK
3	3	3	—	$\overline{\text{RESET}}$	—
4	4	4	PTF0	TPM1CH2	—
5	5	5	PTF1	TPM1CH3	—
6	6	—	PTF2	TPM1CH4	—
7	7	—	PTF3	TPM1CH5	—
8	8	6	PTF4	TPM2CH0	BUSCLK_OUT
9	9	—	PTC6	RXCAN	—
10	10	—	PTF7	TXCAN	—
11	11	7	PTF5	TPM2CH1	—
12	12	—	PTF6	—	—
13	13	8	PTE0	TXD1	—
14	14	9	PTE1	RXD1	—
15	15	10	PTE2	TPM1CH0	—

Table 4. Pin Assignments by Package and Pin Sharing Priority (continued)

Pin Number			<-- Lowest **Priority** --> Highest		
80	**64**	**44**	**Port Pin**	**Alt 1**	**Alt 2**
16	16	11	PTE3	TPM1CH1	—
17	—	—	PTC7	—	—
18	—	—	PTH0	SDA2	—
19	—	—	PTH1	SCL2	—
20	—	—	PTH2	RGPIO8	—
21	—	—	PTH3	RGPIO9	—
22	—	—	PTH4	RGPIO10	—
23	17	12	PTE4	MISO1	—
24	18	13	PTE5	MOSI1	—
25	19	14	PTE6	SPSCK1	—
26	20	15	PTE7	$\overline{SS1}$	—
27	21	16	—	—	V_{DD}
28	22	17	—	—	V_{SS}
29	23	18	—	—	USBDN
30	24	19	—	—	USBDP
31	25	20	—	—	VUSB33
32	26	21	PTG0	KBIP0	USB_ALT_CLK
33	27	22	PTG1	KBIP1	—
34	28	—	PTA0	RGPIO0	USB_SESSVLD
35	29	—	PTA1	RGPIO1	USB_SESSEND
36	30	—	PTA2	RGPIO2	USB_VBUSVLD
37	31	—	PTA3	RGPIO3	USB_PULLUP(D+)
38	32	—	PTA4	RGPIO4	USB_DM_DOWN
39	33	—	PTA5	RGPIO5	USB_DP_DOWN
40	—	—	PTA6	RGPIO6	USB_ID
41	—	—	PTA7	RGPIO7	—
42	34	23	PTB0	MISO2	ADP0
43	35	24	PTB1	MOSI2	ADP1
44	36	25	PTB2	SPSCK2	ADP2
45	37	26	PTB3	$\overline{SS2}$	ADP3
46	38	27	PTB4	KBIP4	ADP4
47	39	28	PTB5	KBIP5	ADP5
48	40	—	PTB6	ADP6	—

Pin Number			<-- Lowest **Priority** --> Highest		
80	**64**	**44**	**Port Pin**	**Alt 1**	**Alt 2**
49	41	—	PTB7	ADP7	—
50	42	29	PTD0	ADP8	ACMP+
51	43	30	PTD1	ADP9	ACMP–
52	44	31	—	—	V_{DDA}
53	45		—	—	V_{REFH}
54	46	32	—	—	V_{REFL}
55	47		—	—	V_{SSA}
56	48	33	PTD2	KBIP2	ACMPO
57	—	—	PTJ0	RGPIO11	—
58	—	—	PTJ1	RGPIO12	—
59	—	—	PTJ2	RGPIO13	—
60	—	—	PTJ3	RGPIO14	—
61	—	—	PTJ4	RGPIO15	—
62	49	—	PTD3	KBIP3	ADP10
63	50	—	PTD4	ADP11	—
64	51	—	PTD5	—	—
65	52	—	PTD6	—	—
66	53	—	PTD7	—	—
67	54	34	PTG2	KBIP6	—
68	55	35	PTG3	KBIP7	—
69	56	36	—	BKGD	MS
70	57	37	PTG4	XTAL	
71	58	38	PTG5	EXTAL	
72	59	39	—	—	V_{SS}
73	—	—	—	—	V_{DD}
74	—	—	PTG6	—	—
75	—	—	PTG7	—	—
76	60	40	PTC0	SCL1	—
77	61	41	PTC1	SDA1	—
78	62	42	PTC2	IRO	—
79	63	43	PTC3	TXD2	—
80	64	44	PTC5	RXD2	—

Table 6. Absolute Maximum Ratings

Rating	Symbol	Value	Unit
Supply voltage	V_{DD}	–0.3 to 5.8	V
Input voltage	V_{In}	–0.3 to V_{DD} + 0.3	V
Instantaneous maximum current Single pin limit (applies to all port pins)[1], [2], [3]	I_D	±25	mA
Maximum current into V_{DD}	I_{DD}	120	mA
Storage temperature	T_{stg}	–55 to 150	°C
Maximum junction temperature	T_J	150	°C

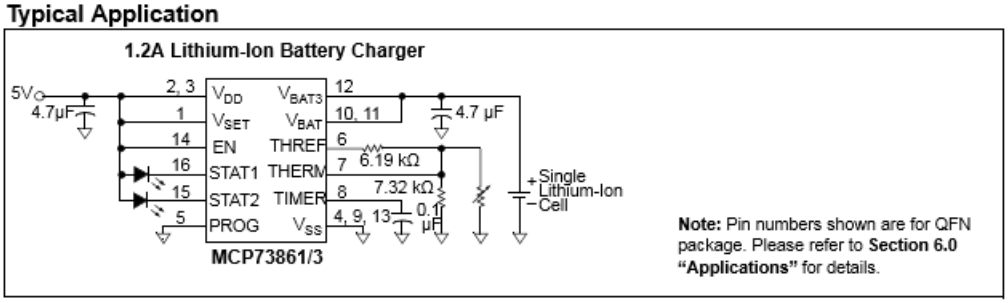
1 Input must be current limited to the value specified. To determine the value of the required current-limiting resistor, calculate resistance values for positive (V_{DD}) and negative (V_{SS}) clamp voltages, then use the larger of the two resistance values.

2 All functional non-supply pins are internally clamped to V_{SS} and V_{DD}.

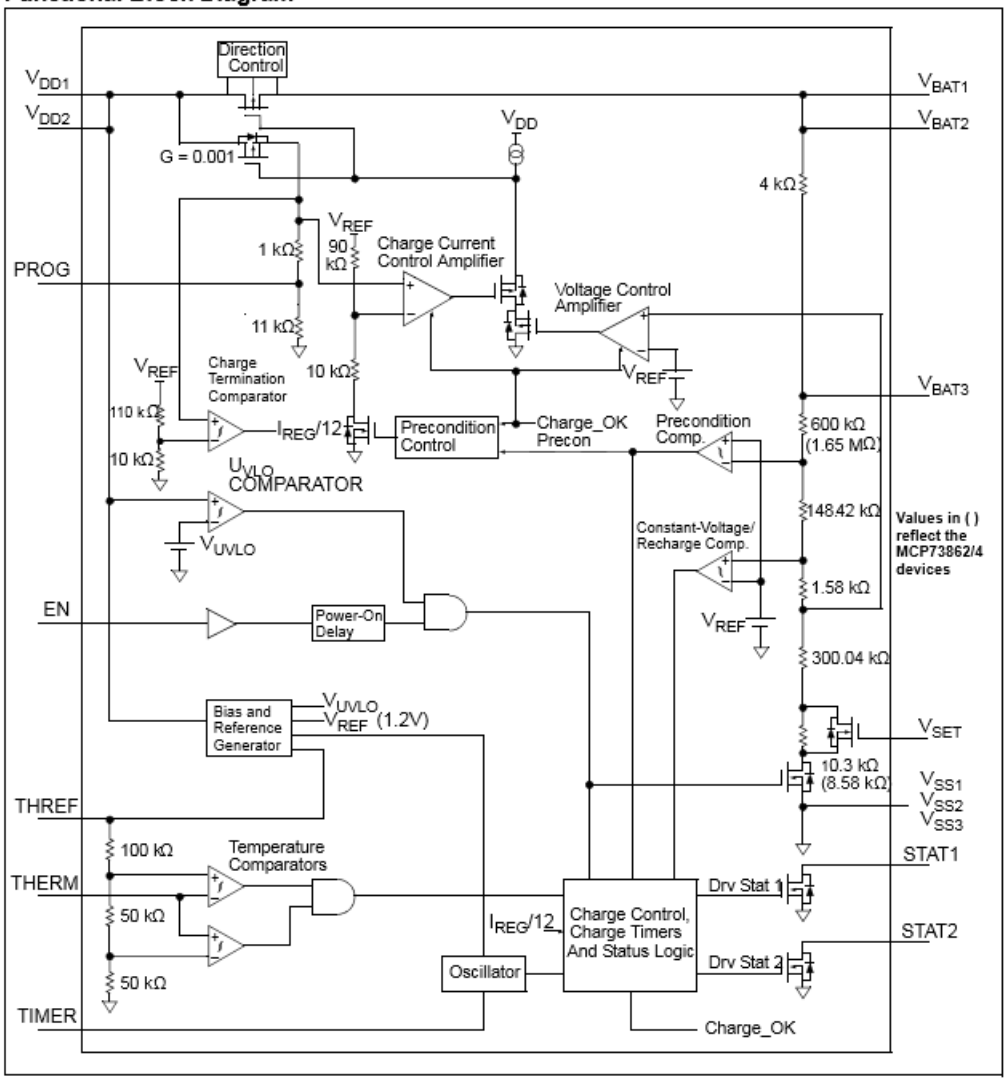
3 Power supply must maintain regulation within operating V_{DD} range during instantaneous and operating maximum current conditions. If positive injection current ($V_{In} > V_{DD}$) is greater than I_{DD}, the injection current may flow out of V_{DD} and could result in external power supply going out of regulation. Ensure external V_{DD} load shunt current is greater than maximum injection current. This is the greatest risk when the MCU is not consuming power. Examples: if no system clock is present or if the clock rate is low, which would reduce overall power consumption.

- **MCP7386X - Charge Management Controller**

Typical Application



Functional Block Diagram



- **TC1262-33VDB**

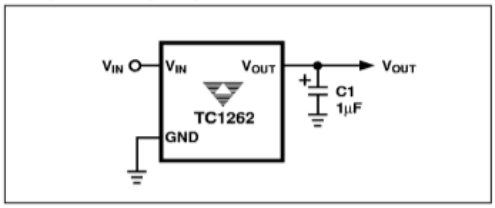
FEATURES

- **Very Low Dropout Voltage**
- **Guaranteed 500 mA Output**
- **High Output Voltage Accuracy**
- **Standard or Custom Output Voltages**
- **Over-Current and Over-Temperature Protection**

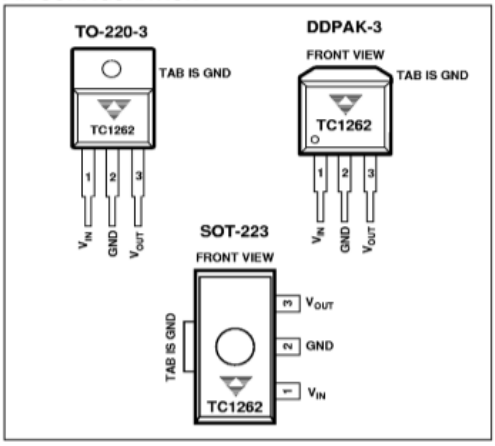
APPLICATIONS

- **Battery-Operated Systems**
- **Portable Computers**
- **Medical Instruments**
- **Instrumentation**
- **Cellular / GSM / PHS Phones**
- **Linear Post-Regulator for SMPS**
- **Pagers**

TYPICAL APPLICATION



PIN CONFIGURATION



GENERAL DESCRIPTION

The TC1262 is a fixed output, high accuracy (typically ±0.5%) CMOS low dropout regulator. Designed specifically for battery-operated systems, the TC1262's CMOS construction eliminates wasted ground current, significantly extending battery life. Total supply current is typically 80 μA at full load *(20 to 60 times lower than in bipolar regulators*!).

TC1262 key features include ultra low noise, very low dropout voltage (typically 350 mV at full load), and fast response to step changes in load. The TC1262 incorporates both over-temperature and over-current protection. The TC1262 is stable with an output capacitor of only 1 μF and has a maximum output current of 500 mA. It is available in SOT-223, TO-220-3, and 3-pin DDPAK packages.

ORDERING INFORMATION

Part Number	Output* Voltage (V)	Package	Junction Temperature Range
TC1262-2.5VDB	2.5	SOT-223	– 40°C to +125°C
TC1262-3.0VDB	3.0	SOT-223	– 40°C to +125°C
TC1262-3.3VDB	3.3	SOT-223	– 40°C to +125°C
TC1262-5.0VDB	5.0	SOT-223	– 40°C to +125°C
TC1262-2.5VAB	2.5	TO-220-3	– 40°C to +125°C
TC1262-3.0VAB	3.0	TO-220-3	– 40°C to +125°C
TC1262-3.3VAB	3.3	TO-220-3	– 40°C to +125°C
TC1262-5.0VAB	5.0	TO-220-3	– 40°C to +125°C
TC1262-2.5VEB	2.5	DDPAK-3	– 40°C to +125°C
TC1262-3.0VEB	3.0	DDPAK-3	– 40°C to +125°C
TC1262-3.3VEB	3.3	DDPAK-3	– 40°C to +125°C
TC1262-5.0VEB	5.0	DDPAK-3	– 40°C to +125°C

*Other output voltages available. Please contact TelCom Semiconductor for details.

- Teoría y Diseño con Microcontroladores de Freescale, Antonio Díaz Estrella, 2008
- Enfermedad de Parkinson y Trastornos relacionados, Federico Micheli, 2006
- Circuitos Microelectrónicos, Sedra y Smith, 2007
- Economía para Ingenieros, Cepeda Gonzalez, 2004