Universidad Zaragoza
1542

Proyecto Fin de Carrera

# Programación integrada para el rápido prototipado de sistemas de rehabilitación

**Autor:**

Pablo Torrecilla Maynar

**Director:**

Dr. Henrik Gollee

**Ponente:**

Prof. José Ramón Beltrán Blázquez

University *of* Glasgow / School of Engineering

Centre for Rehabilitation Engineering

2014

## *0. Resumen*

El propósito de este proyecto es desarrollar el prototipo de un sistema de control de un dispositivo de estimulación eléctrica. Partiendo de un sistema ya existente en ordenadores de sobre mesa, el controlador se implementara sobre una plataforma más flexible y económica, pero también más limitada: Raspberry Pi.

El proyecto incluye desde la configuración del protocolo de comunicación entre el dispositivo de estimulación y la Raspberry Pi, hasta el diseño de un interfaz sencillo que permita, a usuarios sin conocimientos técnicos, controlar la estimulación en tiempo real. Además el sistema será responsable de controlar la frecuencia de estimulación y habilitará opciones de recopilación de datos sobre la estimulación eléctrica.

# Indice

# 1. Introducción

## 1.1 Contexto y motivación

La investigación en el Centro de Ingeniería de Rehabilitación (CRE) de la Universidad de Glasgow se centra principalmente en el uso de la ingeniería para mejorar la salud y calidad de vida de las personas con lesión de médula espinal. Sus actividades van desde las investigaciones sobre los principios fundamentales del control de balance y el modelado de huesos y músculos, pasando por las aplicaciones clínicas tales como la estimulación de los músculos abdominales para la tos y la función respiratoria en tetraplejía, hasta el desarrollo y la evaluación de los sistemas para permitir el entrenamiento funcional y el ejercicio para las personas con lesión de médula espinal.

En la investigación de la estimulación muscular uno de los elementos de estudio es la respuesta eléctrica del cuerpo a un impulso eléctrico cuando el músculo es estimulado por un impulso eléctrico externo. Sin embargo, a la hora de leer las señales del cuerpo la propia señal estimulante oculta la respuesta eléctrica real del cuerpo debido a su gran amplitud, en comparación, saturando los sistemas de captación y grabación. Existen varios estudios acerca de cómo obtener una respuesta eléctrica limpia todos ellos dependen de los dispositivos que se utilizan para registrar el electromiograma (EMG).

En la estimulación mediante un pulso externo, es decir de manera no natural, se ha observado que al cabo de un tiempo la respuesta eléctrica se ve alterada debido a la fatiga del músculo. Por ello, otro punto a estudiar es cómo mejorar el tiempo antes de que el músculo se fatigue, y la respuesta eléctrica a la estimulación cambie. Una de las líneas de la investigación actual se centra en saber cómo una frecuencia aleatoria afectaría a este problema.

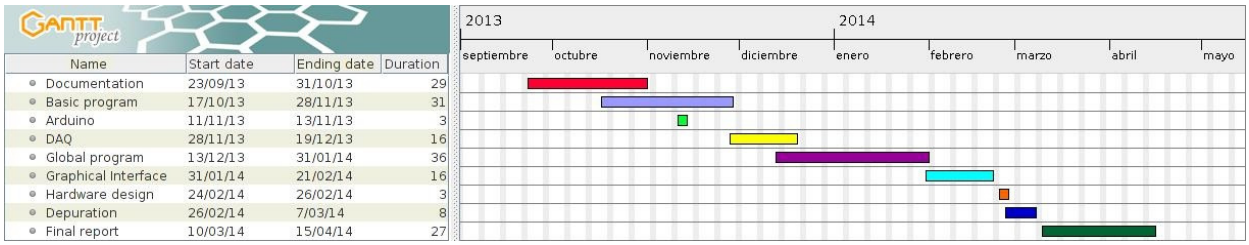
Además de dar solución a estos problemas, el objetivo principal de este proyecto es desarrollar un primer prototipo de un controlador de dispositivo de estimulación eléctrica en la plataforma Raspberry Pi. Hasta ahora, los controladores se ejecutan en los ordenadores y, la mayoría de ellos, debian ser manejados por un usuario técnico, este proyecto pretende abrir la puerta a una nueva plataforma más portátil y económica que podría ser utilizada por un personal sin conocimientos técnicos como, por ejemplo, personal médico.

## *1.2 Objetivos*

- Desarrollar un sistema integrado para el control de un dispositivo de estimulación RehaStimTM basado en la plataforma Raspberry Pi.
- Investigar la grabación la señal EMG de manera limpia para el estudio de la respuesta eléctrica del cuerpo a una estimulación externa.
- Habilitar una opción para estimular con una frecuencia aleatoria con el fin de poder llevar a cabo la investigación futura sobre la fatiga del músculo.
- Diseñar una interfaz de control que permita una interacción fácil entre el usuario y el sistema de estimulación.

## *1.3 Planning y tareas*

A lo largo siete meses, se han desarrollado las diferentes tareas del proyecto. El siguiente gráfico y la explicación posterior pretenden describir cada una de las funciones de las tareas y su temporalización.




| Name | Start date | Ending date | Duration |
|---|---|---|---|
| Documentation | 23/09/13 | 31/10/13 | 29 |
| Basic program | 17/10/13 | 28/11/13 | 31 |
| Arduino | 11/11/13 | 13/11/13 | 3 |
| DAQ | 28/11/13 | 19/12/13 | 16 |
| Global program | 13/12/13 | 31/01/14 | 36 |
| Graphical Interface | 31/01/14 | 21/02/14 | 16 |
| Hardware design | 24/02/14 | 26/02/14 | 3 |
| Depuration | 26/02/14 | 7/03/14 | 8 |
| Final report | 10/03/14 | 15/04/14 | 27 |

Fig. 1. Gantt Graphic with the task of the project and their duration

- **Documentación:** En primer lugar debía tomar contacto con la máquina y su entorno. Así que, sobre todo el primer mes lo dedique al estudio de manuales y la realización de tutoriales sobre Linux, la Raspberry Pi, sus puertos GPIO y el estimulador y la compresión de las tareas de las bibliotecas de C ++ de la versión para ordenadores de sobre mesa.
- **Programa Básico:** Con programas sencillos probé todas las funciones disponibles del estimulador. Estos programas se ejecutaban desde la línea de comandos, sólo para probar

una o dos tareas simples. En un principio funcionaban mediante simples bucles y al final incluyeron también el control de la temporización.

- **Arduino:** Durante unos días estuve trabajando con un estudiante de doctorado para enseñarle las ventajas que la Raspberry Pi aportaba frente al sistema Arduino, en el control de la estimulación en un sistema de ayuda en la respiración.

- **DAQ:** Para poder registrar y almacenar los datos de estimulación, desarrollé un diagrama de bloques en similink. Dicho diagrama utiliza una tarjeta DAQ de National Instruments y permite guardar los parámetros de estimulación, la señal de estimulación, la señal de blanqueo y después de grabarla, calcular el retardo entre ellas.

- **Global program:** En este punto diseñé un programa que abarcara todas las tareas de estimulación probadas antes así como los tres modos de estimulación. Todavía se ejecuta desde la línea de comandos, pero el usuario ya puede detener la estimulación, seleccionar otro modo diferente y continuar.

- **Graphical Interface:** Desarrollada en GTK+, este ha sido mi primer interfaz gráfico, por lo que necesité un trabajo previo de estudio sobre el lenguaje de programación antes de empezar a diseñarlo. Con la interfaz, no solo conseguimos un entorno más “agradable” sino que también facilitamos la interacción con el programa.

- **Hardware design:** Es el medio que el usuario utiliza para comunicarse con el programa, permitiéndole introducir parámetros, iniciar y detener la estimulación. Es un diseño sencillo con un interruptor, cuatro potenciómetros y un chip conversor de analógico a digital.

- **Depuration:** Una vez que la interfaz gráfica y el hardware fueron incluidos en el programa global y él programa funcionaba, dedique un tiempo a buscar y solucionar los posibles problemas que no habían aparecido hasta ahora.

- **Final report:** Se compone de dos documentos: el Manual para mis compañeros en Glasgow que quedaron el sistema y que van a trabajar con él, y el informe final para mi universidad de origen, que es, de hecho, este documento.

## *1.4 Estructura de la memoria*

En las siguientes secciones están expuestos siete meses de trabajo de manera que den una visión global del problema y de cómo se afrontó.

En primer lugar se expone el *Punto de partida*, explicando la forma en que se registra el EMG, las características de la onda-M y cómo se consigue una señal limpia. En este punto será también introducido el dispositivo de estimulación, sus funciones y sus diferentes modos, y la Raspberry Pi, el microordenador que se va a utilizar para controlar el dispositivo de estimulación.

En la sección tres, *Desarrollo del proyecto*, se explican los tres grandes ámbitos que comprenden el proyecto. *El control de la estimulación* incluye la secuencia de comandos para obtener la estimulación y la configuración de todos los parámetros necesarios a la misma. En ella, se resuelve el problema de controlar el momento de la estimulación y cómo ser capaz de hacer la estimulación con una frecuencia fija o aleatoria. Y se afrontará también el problema de crear un gran programa con los limitados recursos de la Raspberry Pi que no son suficientes para trabajar con dos procesos diferentes al mismo tiempo y controlar su correcta ejecución.

*Hardware del controlador* expone el control de los puertos GPIO de la Raspberry Pi y el diseño de un dispositivo de interacción externo con cable que permite al usuario controlar los cuatro parámetros principales de la estimulación sin la necesidad de teclado o ratón.

Y en *Interfaz gráfica sobre GTK+*, se describe el desarrollo de la interfaz gráfica, para un entorno de escritorio GNOME, que muestre el estado de la actual estimulación. Además se describe cómo se arreglaron los problemas de funcionamiento que aparecieron cuando el proceso gráfico funcionaba al mismo tiempo que el proceso de estimulación.

En el apartado cuarto, *Conclusión y trabajo futuro*, se evalúan los objetivos marcados en el inicio del proyecto, y se comentan cuáles serían, en mi punto de vista, los próximos pasos para mejorar este prototipo.

Después, en la *Bibliografía*, se comparte los libros, artículos y sitios web que han ayudado en el desarrollo del proyecto, y se incluyen varios anexos que son necesarios para comprender el trabajo y su resultado, la interfaz de control integrado para el dispositivo estimulación RehaStim™, Stimberry.

# 2. Punto de partida

## 2.1 EMG y la onda-M

La señal de electromiografía (EMG) es la manifestación de la actividad eléctrica producida por la contracción activa de unidades motoras. La monitorización de la EMG se utiliza ampliamente en la biomecánica y la investigación del control de movimiento para determinar cómo el sistema nervioso central (SNC) controla la contracción muscular para producir el movimiento. Más recientemente, la señal EMG producida por la estimulación eléctrica se ha utilizado para proporcionar información sobre el rendimiento muscular en la Estimulación Eléctrica Funcional del sistema (FES), el cual trata de restaurar la función de los miembros paralizados mediante la activación artificial de las neuronas motoras y de los músculos que se inervan.[1]

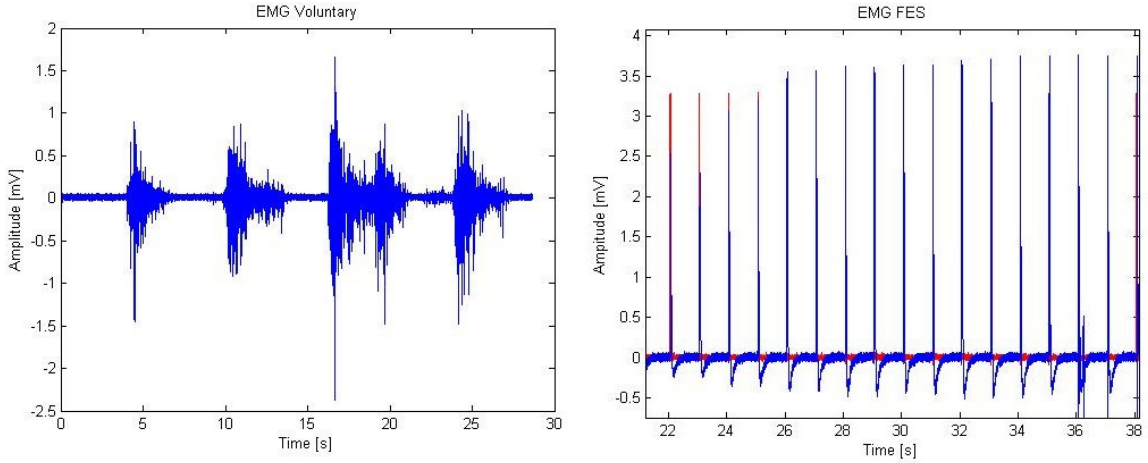



Fig. 1. Imagen comparativa entre la EMG resultante de un movimiento voluntario y una Estimulación Eléctrica Funcional (FES).

En la respuesta eléctrica del cuerpo para una FES se incluyen el reflejo de Hoffmann (reflejo-H) y la respuesta motora directa (onda-M). Estas dos señales se utilizan para estudiar la actividad eléctrica del cuerpo en la investigación para mejorar el movimiento en personas con lesión de la médula espinal. La onda-M, respuesta directa, generalmente tiene un umbral de activación más alto que el reflejo-H debido al tamaño relativamente más fino de los nervios motores en comparación con los husos musculares aferentes. La onda-M también se produce a una menor latencia (aproximadamente 5-8 ms en el sóleo) que el reflejo-H (aproximadamente 30-45 ms en el sóleo). Esto se debe a que la onda M sólo tiene que viajar a lo largo del axón motor mientras el reflejo-H se

produce en las fibras aferentes, realiza las sinapsis en las neuronas motoras de la médula espinal, y luego viaja a lo largo de las fibras nerviosas eferentes antes de provocar una respuesta en el músculo de la blanco.[2]

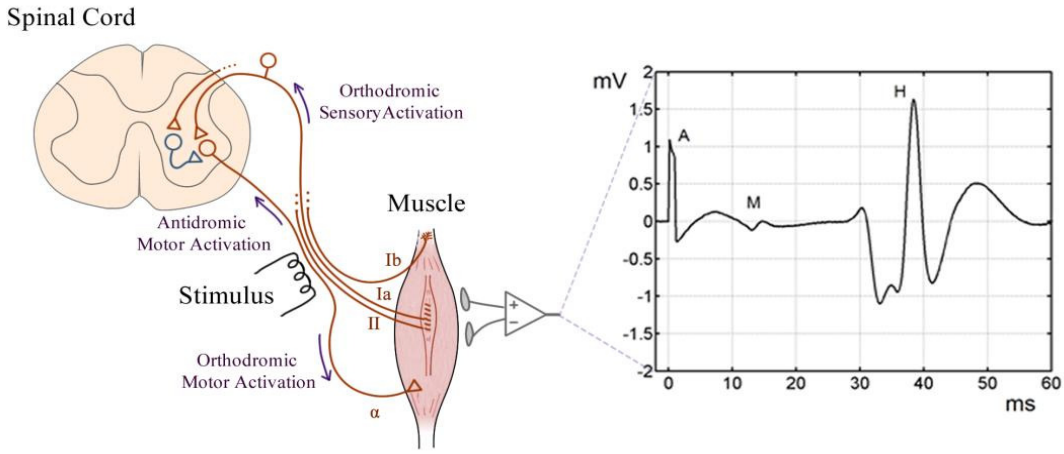



Fig. 2. Esquema de la estimulación y la respuesta eléctrica del cuerpo a ella. La onda A representa la estimulación externa, tras la estimulación aparecen la onda-M y unos pocos milisegundos después el reflejo-H.

En una situación de FES el resultado de la EMG, lo que debería ser la onda-M seguido del reflejo-H, es precedido por una señal de artefacto de estímulo mucho más fuerte que satura los dispositivos de medición ocultando la onda-M. La existencia del artefacto es el resultado de una diferencia de potencial desarrollado por la corriente estimulante entre los electrodos de registro del EMG. Debido a esta diferencia de potencial aparece como una señal diferencial, que no puede ser rechazada por un amplificador diferencial.

En la estimulación por electrodos de superficie, la magnitud del artefacto es especialmente alta por dos razones:

(a) las altas intensidades de estimulación que normalmente se requieren en este caso, para la activación de las extremidades.

(b) la ubicación de los electrodos de registro EMG, por lo general entre los electrodos de estimulación.

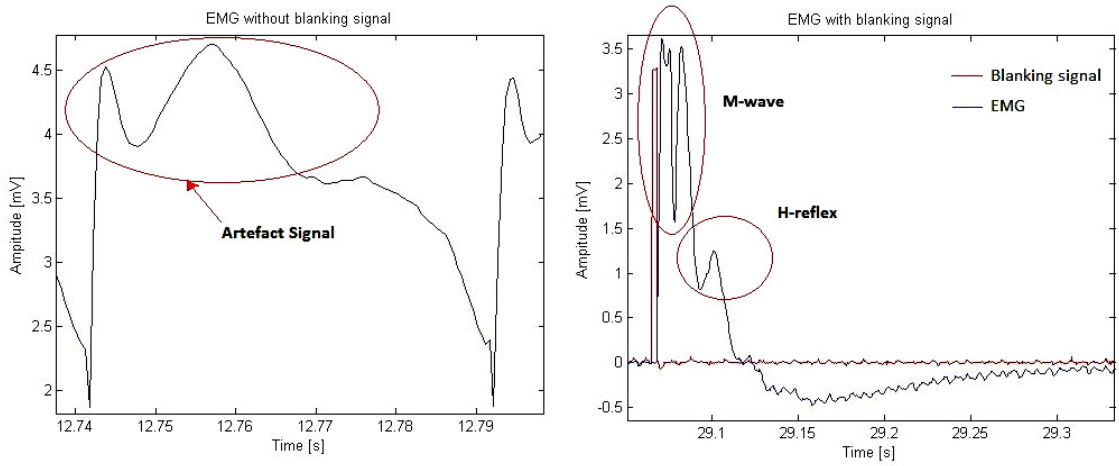



Fig. 3. Imagen comparativa entre la EMG obtenida utilizando una señal de blanqueo y sin utilizarla. Como muestra la imagen sin esta señal la señal obtenida está completamente saturada.

El tema de la eliminación de artefactos en la neurofisiología ha sido ampliamente discutido en la literatura[3] y su solución depende de la clase de dispositivo grabador EMG y el tipo de dispositivo de estimulación.

## *2.2 El dispositivo de grabación EMG y el dispositivo de estimulación*

### 2.2.1. El dispositivo de estimulación RehaStim™

El dispositivo de estimulación es un estimulador de 8 canales de corriente controlado, con una certificación de producto médico que posee dos fuentes de corriente independientes ambas multiplexadas a 4 salidas cada uno. Posee dos módulos de estimulación independientes alojados cada uno en una de las fuentes de corriente. El procesador principal del estimulador es un procesador de 16 bits de señal mixta RISC de ultra-baja potencia de Texas Instruments (MSP430). Cada módulo de estimulación posee uno de estos microprocesadores que es responsable de la temporización de la generación de impulsos. El módulo de estimulación A controla los canales de estimulación 1 a 4, módulo de estimulación B, los canales de estimulación 5 a 8. Dado que ambos módulos funcionan de forma independiente la generación de pulso simultánea en el módulo A y B es posible.[4]

La figura muestra la forma de un pulso emitido en el caso ideal de una carga resistiva. Como se puede ver se trata de un pulso bifásico con una pausa fija de 100 ms entre las dos fases del pulso.

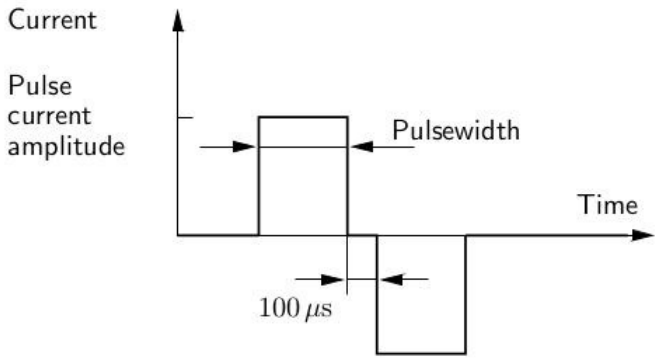



Fig. 4. Descripción del ancho de puso y de la corriente del puso bifásico.

El estimulador ofrece diferentes modos de generación de impulsos de estimulación:

**- Single Mode:** Este modo indica al dispositivo de estimulación que, en ese mismo instnate, tiene que estimular un canal específico con unos parámetros específicos.

**- CCL Mode:** Esta vez, el dispositivo de estimulación RehaStim™ se inicializa al principio con todos los parámetros que son necesarios para cada canal para estimular, incluso la frecuencia de la estimulación y el dispositivo RehaStim™ se encarga de controlar la temporización.

**- OSCL Mode:** Es un modo similar que la CCL, sólo hay una diferente entre ellos. Ahora la frecuencia no es controlada por el dispositivo de estimulador, sino que debe ser realizado por el programa.

### 2.2.2. Dispositivo de grabación de EMG

La grabadora EMG se compone de un pre-amplificador, un módulo de aislamiento y un módulo de filtro de paso bajo, todos ellos de Neurolog DIGIMETER.

El dispositivo fundamental para nosotros es el módulo de aislamiento. El NL820A es un módulo de aislamiento de la señal analógica de cuatro canales diseñado para conectar en el sistema de grabación de NeuroLog™. Tiene cuatro entradas de un solo terminal con un terminal aislado común que juntos proporcionan fuentes aisladas positivas y negativas. El modulo puede trabajar

con señales de entrada con frecuencias comprendidas desde el rango de CC hasta más de 10 kHz y amplitudes de hasta ± 1 voltio.

Como vimos en la sección de *EMG y onda-M*, la señal de artefacto, generada por la diferencia de potencial entre los electrodos de registro EMG, satura la grabadora y ocultar la onda-M. Solucionar este problema es el propósito de la entrada MUTE. El sistema utiliza esta entrada para bloquearse a sí mismo y lo protegerse de la señal de artefacto, siendo capaz de leer la EMG correctamente. La siguiente figura muestra cómo debe ser la señal de entrada y el comportamiento del sistema a la misma.

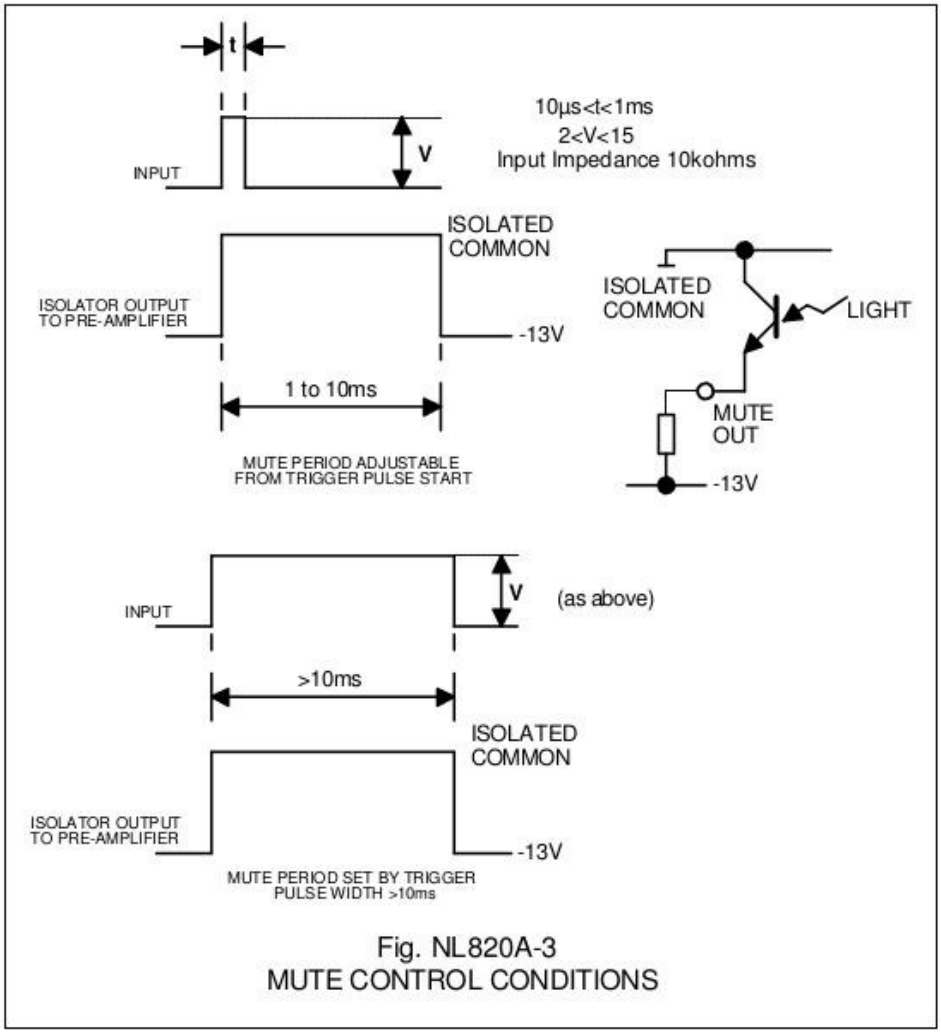



Fig. 5. Descripción de la señal MUTE y sus efectos en la grabadora de EMG.

## *2.3 Raspberry Pi*

La Raspberry Pi es una computadora de una sola placa del tamaño de una tarjeta de crédito. Fue desarrollada por la Fundación Raspberry Pi con la intención de promover la enseñanza de la informática básica en las escuelas. A día de hoy, no sólo ha cumplido con este propósito, sino que además ha revolucionado el mundo de microcontorladores.

La Raspberry Pi posee un sistema BCM2835 de Broadcom en un chip (SoC), que incluye un procesador ARM1176JZF-S de 700MHz, GPU VideoCore IV, y 512 megabytes de RAM. No incluye disco duro integrado o unidad de estado sólido, sino que utiliza una tarjeta SD para el arranque y el almacenamiento persistente.

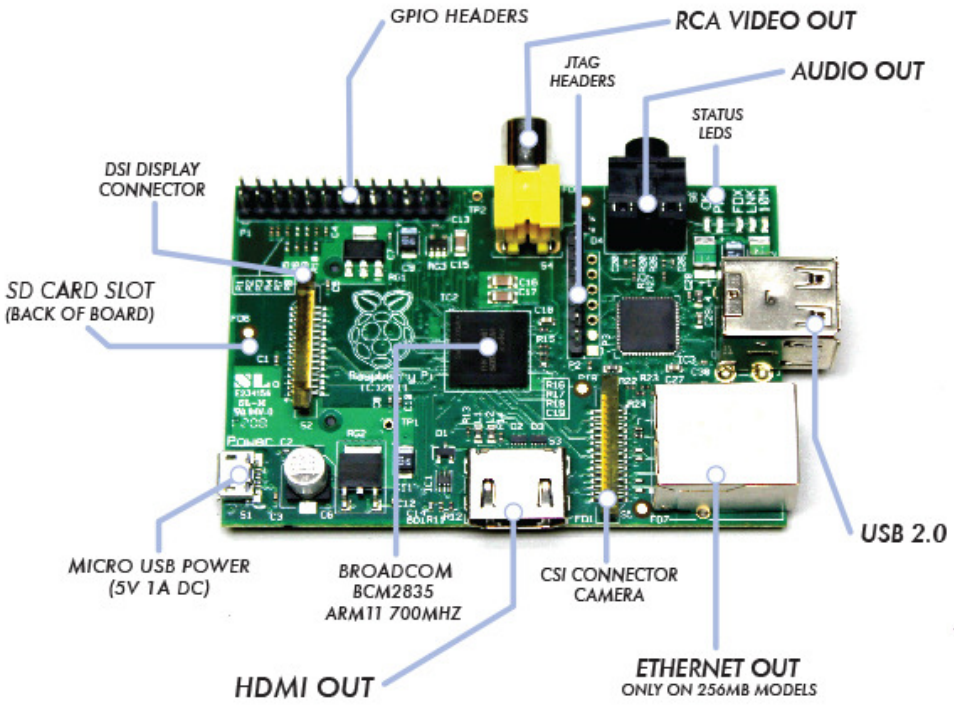



Fig. 6. Imagen descriptive de la Raspberry Pi

La Fundación ofrece seis sistemas operativos diferentes, 5 de ellos basados en Linux. La mayoría de ellas son versiones desarrolladas para un propósito específico, sin embargo el Raspbian, basado en Debian, fue creado para ser flexible y útil para todas las aplicaciones.

Además de contar con puertos USB y Ethernet, bus HDMI, vídeo RCA y jack de audio, la Raspberry Pi tiene 26 puertos GPIO para comunicarse con dispositivos externos. Algunos de ellos, además de ser puertos digitales de I/O, también pueden desarrollar otra función como SPI o puertos UART.

## *2.4 ScienStim. Versión para ordenador del controlador*

Como se explicó antes, existe una versión anterior del programa que hacen la misma tarea, pero para ordenador en lugar de para Raspberry Pi. Así que disponía de algunas bibliotecas que resultaron ser muy útiles en mi trabajo. Estas se pueden organizar en dos tipos:

- **Protocolo de comunicación:** La biblioteca "sciencemode.cpp" define la clase *estimulador* y todas las instrucciones necesarias para la comunicación con el estimulador. Estas instrucciones toman los datos de los parámetros, los empaquetan según describe el Manual de Protocolo de comunicación y envían el flujo de bits a través del puerto serie.

- **Puerto Serie:** La comunicación entre el estimulador y la Raspberry Pi se realiza a través del puerto USB, pero este puerto es manejado de manera diferente en Windows y Linux. De modo que la biblioteca "serialport.h" identifica el sistema y selecciona la biblioteca correcta ", serial_linux.cpp" o "serial_windows.cpp" en función del SO. Ambas definen y describen las instrucciones para manejar el puerto USB, como se abre, se configura, se envía una cadena, se lee, y como se cierra el puerto.

# 3. Desarrollo del proyecto

## 3.1 Controlar la estimulación

### 3.1.1 Modos Estimulación

El dispositivo de estimulación, RehaStimTM, aunque puede ser utilizado por sí solo, también tiene la opción de ser controlado por un dispositivo externo. Esta opción se activa con el modo *CienceMode* del menú principal. Aprovechando este punto es cómo la versión de sobremensa controlaba el estimulador y es cómo este sistema lo controlará.

Teniendo en cuenta que el sistema ya se había desarrollado para otra plataforma, el primer paso fue comprobar que partes del código anterior se podían utilizar en la versión para Raspberry Pi. Para esta tarea me serví de la ayuda del Manual de Protocolo[4] del dispositivo, para identificar las funciones que eran útiles en la comunicación con un controlador externo. Y así una vez comprendido el funcionamiento de las bibliotecas que disponía, explicadas en el apartado anterior, nacieron los algoritmos para la cada uno de los tres modos. Estos algoritmos se componían de un bucle finito simple, de manera que realizaran un par de estimulaciones con el fin de comprobar que la Raspberry Pi era capaz de controlar el estimulador.

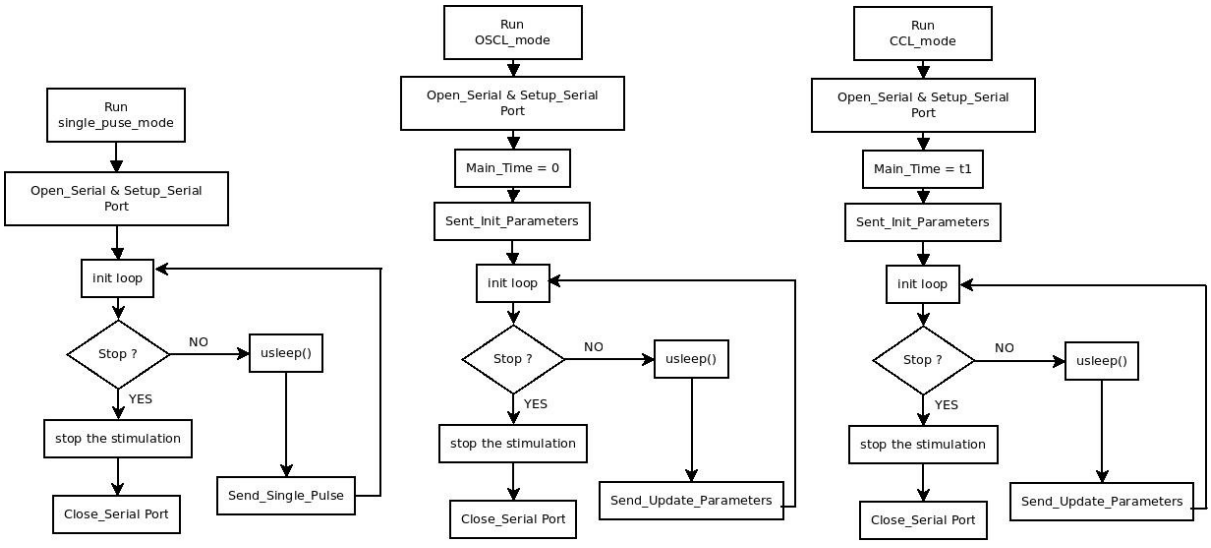



Fig. 6. Los tres modos de estimulación disponibles. La diferencia entre el modo OSCL y el CCL es que en el primero el control del tiempo lo realiza la Raspberry Pi, para ser capaces de esto Main Time debe ser igual a cero, de esta forma el estimulador entiende, en el proceso de inicialización, de qué modo se trata.

### 3.1.2 Timing

Una vez que todos los modos se ejecutaban correctamente en la Raspberry Pi, el siguiente paso fue ser capaz de controlar el momento de la estimulación. Esta fue la primera vez que me di cuenta de los límites de la CPU de la Raspberry Pi.

A sabiendas de que el uso de funciones como "usleep ()" no era una buena idea, ya que simplemente detenían la CPU y todos los procesos en curso, lo que lo que es un problema cuando se desea una información en tiempo real, la función lógica a usar parecía ser "setitimer ()". Esta función configura un temporizador que, cuando expira, envía una señal al proceso que lo crea, dicha señal puede ser atrapada para ejecutar una instrucción específica. Sin embargo, como la Raspberry Pi no puede ejecutar dos procesos en paralelo, no se controla el momento de captura de la señal, ni lo que la CPU está haciendo en ese momento ni el tiempo que tarda en terminarlo, por lo que se no puede garantizar que la estimulación ocurra en el mismo instante en cada iteración. Si la Raspberry Pi pudiera ejecutar dos procesos en paralelo el problema no existiría.

Una solución a este problema podría ser utilizar una señal externa. Entre los dispositivos que disponía allí había una tarjeta de expansión a través de los puertos GPIO conocida como Gertboard. Esta placa viene con una gran variedad de componentes, incluyendo un microcontrolador ATmel ATmega 328p AVR (Arduino) que puede ser utilizado para enviar un pulso a la Raspberry Pi en el momento de la estimulación. Sin embargo, un objetivo importante del proyecto es que el sistema sea lo más pequeño y portátil como sea posible, y el uso de un dispositivo externo sólo para una señal no parecía una buena opción. Aunque, la Gertboard no se incluyó en la versión final, resultó ser muy útil gracias a la cantidad de bibliotecas que existen para controlar el GPIO desarrolladas para ella.

De modo hubo que buscar otra solución que no necesitara un dispositivo externo y que pudiera garantizar el orden de la secuencia de instrucciones. Esta resulto ser una función de librería llamada "clock_gettime ()". Esta función lee uno de los relojes disponibles de forma que permite calcular cuánto falta para la siguiente estimulación, leyendo en cada iteración el reloj.

Esta opción, además de cumplir con la tarea de controlar la temporización, permite utilizar una frecuencia aleatoria fácilmente. Simplemente se necesita calcular un número aleatorio después de cada estimulación y usarlo como una variación de la frecuencia.

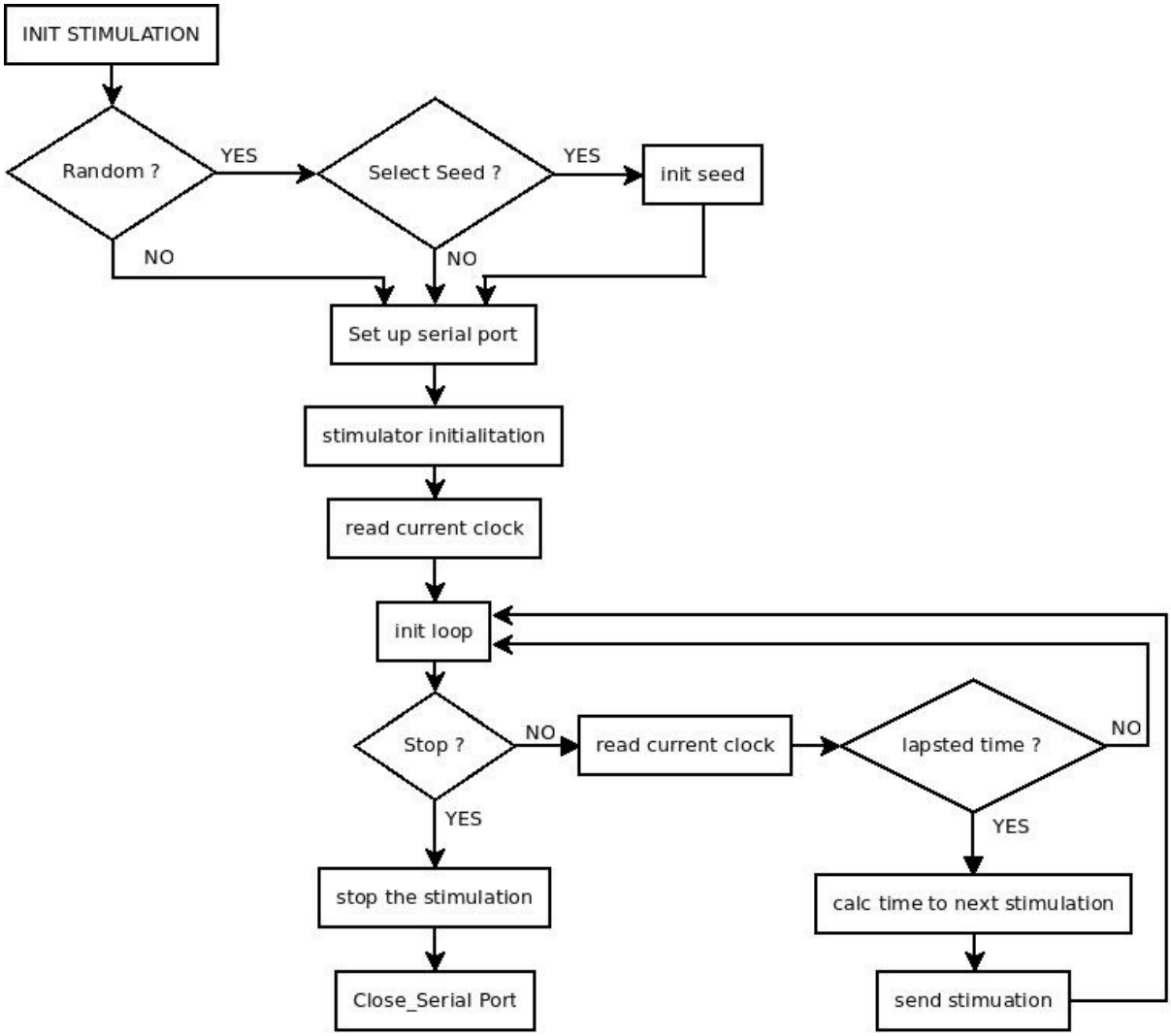



Fig. 7. Diagrama descriptivo del control del tiempo.

### 3.1.3 Englobando todas las tareas

En este punto, los tres modos de estimulación funcionaban correctamente, pero en diferentes programas. La versión final, en cambio, tenía que incluir todas las opciones para que el usuario pudiera seleccionar entre los tres modos de estimulación sin necesidad de salir del programa y tener que ejecutar otro. Este fue el siguiente paso.

La primera idea fue utilizar un pequeño programa como su menú principal. En el cual el usuario pudiera introducir los parámetros de la estimulación y seleccionar el modo, y, en el momento de iniciar la estimulación, se ejecutara el programa correspondiente en función del modo de estimulación. De esta forma, este programa sería el “proceso padre” del proceso de estimulación y estarían comunicados por una tubería. Sin embargo, una vez más me encontré con que el

Raspberry Pi no disponía de suficientes recursos para realizar dos procesos, el padre y el niño, al mismo tiempo. El orden de la secuencia de instrucciones no podía ser garantizado.

Al final, la solución fue utilizar cada modo como funciones del programa de menú principal, y los parámetros de la estimulación como variables globales. De esta forma, pueden ser modificados durante la estimulación y ser almacenados cuando la estimulación se ha terminado quedando disponibles para el próximo experimento.

## *3.2 Hardware del controlador*

Como ya he indicado varias veces antes, uno de los objetivos más importantes para este proyecto era conseguir un sistema lo más portátil como fuera posible. Esto supuso una gran restricción a la hora de diseñar un hardware que permite el cambio de algunos parámetros de la estimulación en tiempo real.

Los parámetros que el usuario tenía que ser capaz de modificar mientras la estimulación está sucediendo son:

- **Frecuencia** de la estimulación.
- **Corriente**, o intensidad del pulso.
- **Ancho** del pulso de estimulación.
- **Varianza**, para los escenarios con frecuencia aleatoria.

Para desarrollar un dispositivo controlador que pudiera comunicar estos valores a la Raspberry Pi en tiempo real, tuve que sacar partido a sus puertos GPIO.

### 3.2.1 Diseño del Hardware

Algunos puertos GPIO pueden ser configurados de un modo alternativo, en particular, cinco de ellos pueden ser configurados para desarrollar un BUS SPI (Serial Peripheral Intrerface). Este tipo de Bus permite la comunicación en serie con dispositivos periféricos como por ejemplo un conversor Analógico/Digital que lea los valores de los parámetros de cuatro potenciómetros. Además, dado que mucha gente ha trabajado con la Raspberry Pi y la Gertboard antes yo existen bibliotecas muy útiles diseñadas para sacar el máximo partido a estos puertos GPIO y sus funciones.[5]

El complemento de expansión GPIO Gertboard introducido ya en la sección *timing*, además de tener el chip ATmega, tenía también un chip de A/D con dos canales que podía ser utilizado para leer el valor de los cuatro potenciómetros con la ayuda de un multiplexor. Incluso, cuando al final decidí no utilizar la Gertboard, la idea continuaba siendo una buena opción, así que compre los componentes y los emcapsule en forma de mando de control.

Este es el esquema final del controlador: cuatro potenciómetros multiplexados, digitalizados conenctados a la Raspberry Pi a traves del Bus SPI.

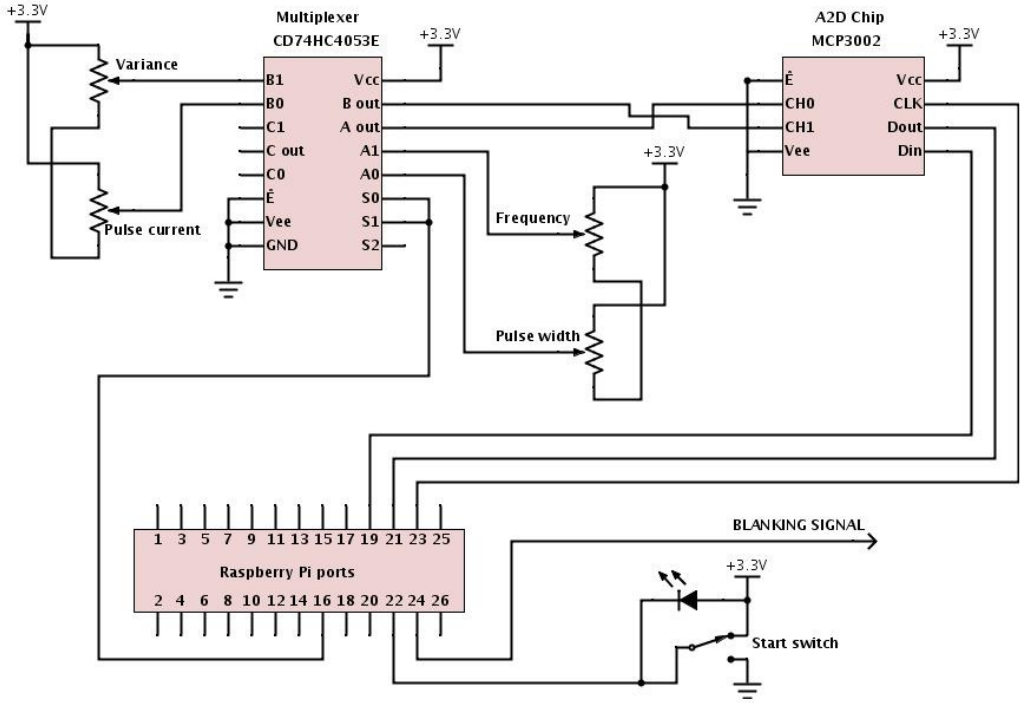



Fig. 8. Esquema final del Hardware

La siguiente imagen muestra el controlador encapsulado con sus cuatro potenciómetros para especificar los parámetros de estimulación: varianza, frecuencia, corriente de pulso y ancho de pulso, y un interruptor para iniciar y detener la estimulación. Como se ve en la imagen del controlador es tan pequeño como el Raspberry Pi y se puede unir a la parte posterior de la misma, ocupando el mínimo espacio.


Fig 9. Apariencia final del controlador con sus cuatro potenciómetros, el botón de inicio y las salidas en la parte trasera que facilitan la lectura de información en caso de que sea necesario.

En experimentos de investigación resulta útil ser capaz de almacenar los parámetros empleados. En caso de necesitar guardar el valor analógico, este se puede obtener en tiempo real por un puerto situado justo detrás de cada potenciómetro.

### 3.2.2 Señal de Blanqueo

La señal de supresión es una señal digital que indica, con un nivel alto, el momento de la estimulación. Como se explicó antes, la grabadora de EMG necesita una señal de disparo para bloquear el sistema y no saturarse con la señal de estimulación. Esta señal de supresión está disponible en la parte posterior del controlador marcada como *"blanking signal"* y debe ser conectado a la entrada MUTE de grabadora de EMG.

Esta señal sólo está disponible en modos Single Pulse Mode y OSCL Mode, donde el tiempo es controlado por el Raspberry Pi. Para obtener una mayor precisión, la señal de supresión se genera al mismo tiempo que es enviada la instrucción de estimular al estimulador. Esto sucede las instrucciones *Send_Update_Parameter* (para el modo OSCL) *y Send_Single_Pulse* (para el modo de un solo pulso) ambas definidas en el archivo *"sciencemode.h".*

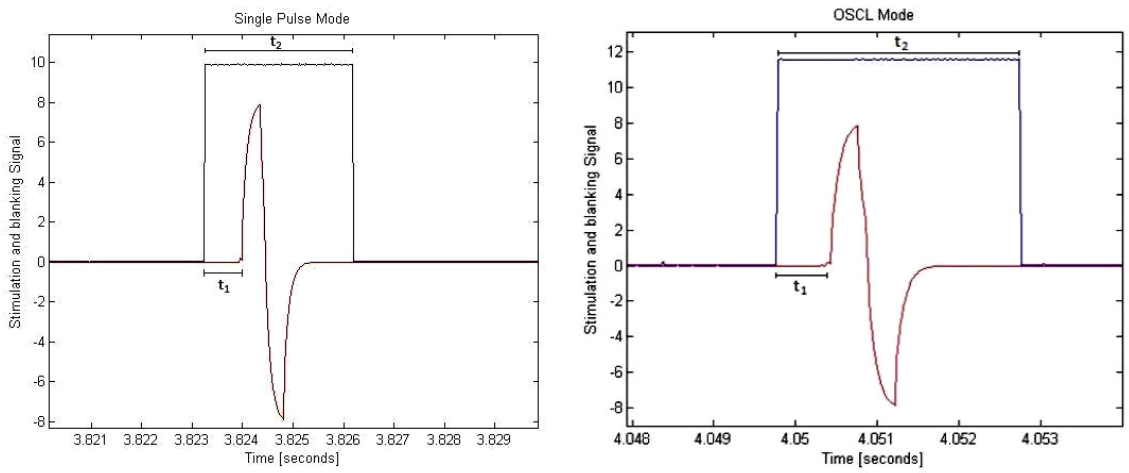



Fig. 10. La señal de blanqueo y la señal de estimulacion para Single Pulse and OSCL mode

La imagen de arriba muestra la señal de supresión y la estimulación en ambos modos. Esta señal, con 3,3 voltios de amplitud, está definida por dos parámetros de $t_1$ y $t_2$.

Experimentalmente el parámetro $t_1$ no siempre es el mismo, pues depende del tiempo de ejecución de las instrucciones, y oscila entre 460 a 860 microsegundos para Single Pulse Mode y entre 400 a 840 microsegundos para el modo OSCL. El otro parámetro, $t_2$, está definido por la suma de la anchura de pulso de la estimulación y la constante MUTE_PULSE (definida en *"PARAMETERS.h"*). Esta constante puede ser modificada según las especificaciones del dispositivo de grabación.

## *3.3 Interfaz gráfico sobre GTK+*

### 3.3.1 Interfaz Gráfico

Una interfaz gráfica, desarrollada en GTK para entornos de escritorio GNOME, hace más fácil el control de la estimulación para el usuario. Esta interfaz permite especificar y ver en tiempo real los parámetros de la estimulación.

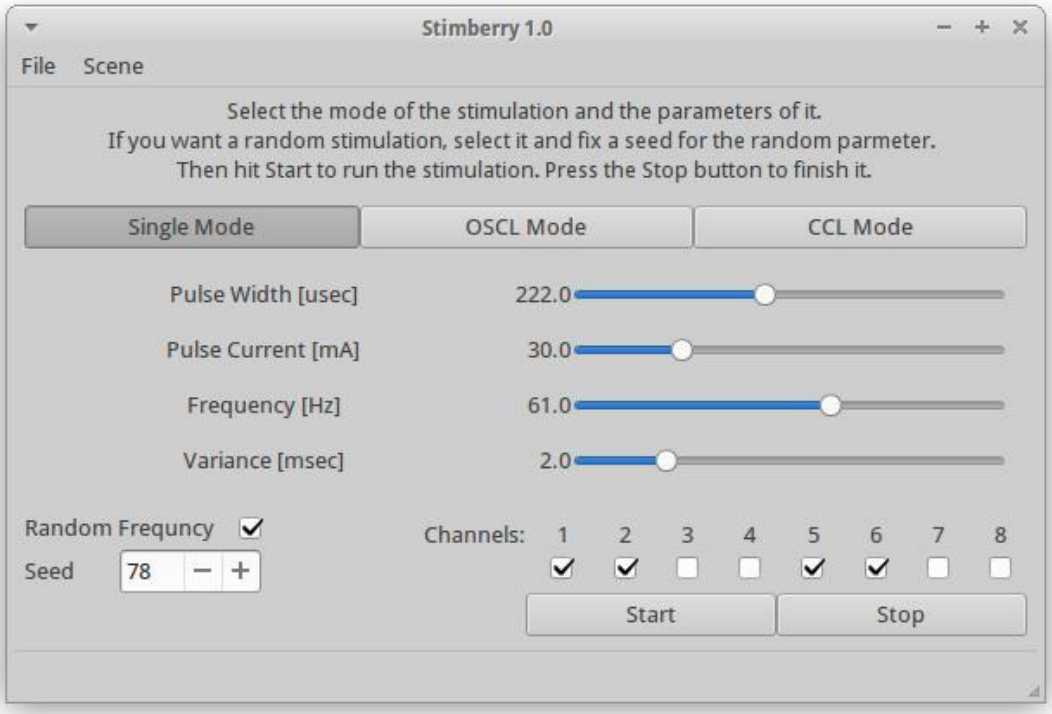



Fig 11. Interfaz gráfico

El programa principal, que hasta ahora era sólo un menú para fijar los parámetros y seleccionar el modo, ahora tiene que englobar además toda la configuración gráfica que permita leer, escribir y controlar todos objetos del interfaz. Para hacer esto posible, todos los objetos gráficos (escalas, botones...), sus atributos y las señales que generan cuando se interactúa con ellos, han sido definidos y especificados en el archivo *"interfaceStimberry.glade"*. De esta forma, cada señal generada es manejada por el programa principal ejecutando la función específica definida para esa señal.

Tras incluir la interfaz gráfica, el programa principal tiene la siguiente estructura:

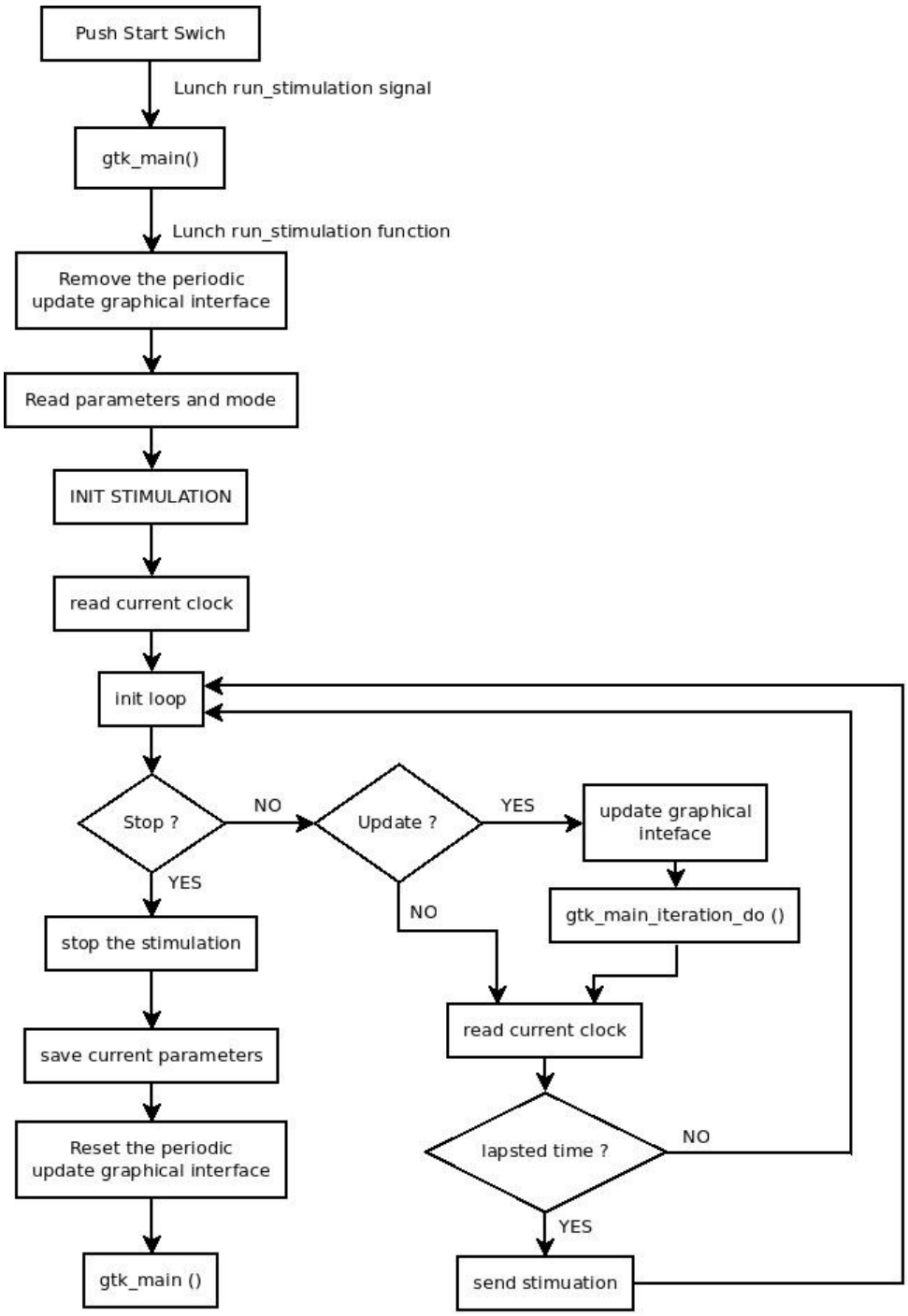



Fig. 12. Diagrama descriptive sobre el control de la interfaz gráfica.

Todas las interacciones entre el usuario y los objetos del interface son controladas por un bucle sin fin llamado *gtk_main()*. De manera que cuando pulsamos o modificamos el valor de algún dato, el objeto lanza una señal que es atrapada por esta función, y se ejecuta la secuencia

correspondiente a esa señal. Al terminal la secuencia de instrucciones el programa vuelve al estado de alerta a la espera de nuevas señales o eventos.

Sin embargo, el problema con el *bucle gtk_main()* es que cuando se captura la señal de inicio, y se comienza un modo de estimulación, el resto de señales o eventos quedan a la espera de que termine para ser tratados por la función *gtk_main()*. De modo que la interfaz gráfica se quedaba congelada y no se actualizaban los datos hasta que se detenía la estimulación y se devolvía el control al *gtk_main().* Este problema se pudo resolver este gracias a la función *gtk_main_iteration_do ().* El uso de este comando resuelve una iteración del *bucle gtk_main()*, por lo que utilizándolo en el código de la estimulación cada pocos milisegundos, se controlaban todos los posibles eventos.

### 3.3.2 Cinco escenarios de simulación

Para hacer más fácil la interacción con un usuario no-técnico y conseguir no utilizar nada más que lo esencial, plantee la opción de no necesitar utilizar ni el teclado ni el ratón.

Con este fin, el programa fue preparado para ser capaz de almacenar cinco escenas de estimulación con sus parámetros específicos que se pueden fijar en un paso previo de instalación. Para este paso sólo necesitamos un ratón para seleccionar las preferencias de la estimulación: Modo, Frecuencia Aleatoria, Canales,... y después guardar la escena en una de los cinco escenarios disponibles. Una vez realizado esto, la próxima vez que se inicie el programa, se cargará el último escenario guardado.

Estas opciones permiten no sólo guardar escenarios determinados para repetir los experimentos más tarde, sino también preparar el sistema para funcionar de forma automática con nada más, que el mando del controlador.

### 3.3.3 Autoarranque del sistema

Una vez desarrollados los escenarios sólo quedaba una cosa para conseguir no necesitar ni teclado ni ratón, y era poder ejecutar automáticamente el programa cuando la Raspberry Pi arrancase. Así, cuando el usuario encienda el Raspberry Pi el programa abrirá y cargará la última escena guardada y el usuario sólo necesita presionar el botón de inicio.

Este problema no resulto muy difícil de resolver. El Sistema Operativo Raspbian está basado en Linux por lo que cualquier script que este a la ruta "/home/pi/.config/autostart" se ejecutará en el momento del arranque. Guardando el siguiente script en dicha ruta solucionábamos el problema.

```
[Desktop Entry]
Type=Application
Exec=lxterminal --command sudo ./Stimberry/Stimberry_4.0
```

# *4. Conclusión y trabajo futuro*

El objetivo de este proyecto era desarrollar un primer prototipo de un controlador existente para una nueva plataforma, la Raspberry Pi. Este objetivo se ha superado con éxito, el sistema funciona de manera estable y se ha conseguido un interfaz sencillo de manejar que cualquier persona técnica o no puede utilizar, sinendo capaz de obtener información acerca de la estimulación en tiempo real.

El propósito fundamental de este proyecto es permitir el estudio de la onda-M en músculos con estimulación eléctrica funcional (FES) y su fatiga cuando utilizamos una frecuencia de estimación aleatoria. El resultado no solo permite estos estudios, sino que además el proyecto ha abierto la puerta a futuros controladores implementados sobre la Raspberry Pi. Durante el proyecto trabajé con otros estudiantes de doctorado y en otras aplicaciones, por ejemplo un controlador de un sistema de ayuda a la respiración que estimula el diafragma cuando un sensor detecta una exhalación. Con el fin de dar cabida a proyectos similares, el siguiente paso sería desarrollar un programa principal más grande que permite al usuario seleccionar entre diversas funciones de estimulación: estimulación con supresión de la señal de grabadora de EMG, la estimulación con sensor externo para ayudar en la respiración, estimulación para el estudio de sistemas de locomoción para personas con lesiones medulares, la estimulación de varios músculos para mejorar el movimiento del tronco inferior y/o superior,... De esta forma obtendríamos u producto más global y versátil.

La Raspberry Pi también nos ofrece otras opciones a considerar como por ejemplo utilizar un widget Wifi USB para conectarla a una LAN y ser capaz de controlar el sistema de forma remota, desde un ordenador o, incluso desde el teléfono móvil.

Este proyecto queda demostrado que la opción de la Raspberry Pi como controlador no sólo es viable sino que además es también muy rentable. La siguiente tabla muestra el coste de cada artículo que se necesita:

| Descripcion del artículo | Distribuidor | Precio |
|---|---|---|
| Raspberry Pi Model B + 8 Gb SD | www.element14.com | £27.98 |
| Multiplex CD74HC4053E | uk.farnell.com | £0.61 |
| Analog to Digital MCP3002-I/P | uk.farnell.com | £1.67 |
| BW® 7 inch TFT Color LCD | amazon.co.uk | £22.95 |
| Power Supply 12v/5A AC for LCD | amazon.co.uk | £10.00 |
| Prototyping Box | uk.farnell.com | £9.76 |
| GPIO Cable for Use with Raspberry Pi | amazon.co.uk | £4.99 |
| | | £77.95 |

A nivel personal, este proyecto me ha permitido aprender más sobre Linux y la Raspberry Pi, temas que en los que estaba realmente interesado. He visto las posibilidades que ofrece este pequeño ordenador y me gustaría seguir descubriendo más aplicaciones, es por ello que actualmente ya poseo mi propia Raspberry Pi. Además, he tenido la oportunidad de aprender nuevos lenguajes de programación, otras máquinas y otros Sistemas Operativos. Mientras estudiaba estos lenguajes pude darme cuenta de que, con los conocimientos adquiridos durante la carrera, el tiempo necesario para ser capaz de manejarme con rapidez era mucho menor del que en un principio planifique. Así fue el caso de la biblioteca GTK + que ha hecho posible que desarrollé mi primer programa con una interfaz para Linux.

# 5. Referencias y Bibliografía

### *Referencias:*

[1] Naomi C. Chesler, William K.Durfee: Surface EMG as a Fatigue Indicator During FES-induced Isometric Muscle Contractions. J.Electromyogr. Kinesiol. Vol. 7, No 1, pp. 27-37, 1997

[2] Kylie J.Tucker, Meltem Tuncer, Kemal S. Türker: A review of the H-reflex and M-wave in the human triceps surae. Human Movement Science No. 24, pp. 667-688, 2005

[3] J. Minzly, J. Mizrahi, N. Hakim, A. Liberson: Stimulus artefact suppressor for EMG recording during FES by a constant-current stimulator. *Med. & Biol. Eng. & Comput*. No 31, pp. 72-25, 1993

[4] Thomas Schauer, Nils-Otto Negaard, Carsten Behling: ScienceMode, RehaStim™ Stimulation Device. Description and Protocol. *www.hasomed.de,* 2009 **(see annex II.iii)**

[5] Gordon Henderson: *https://projects.drogon.net/raspberry-pi/gertboard/* 2012

### *Programación (C++ and gtk+):*

- Herbert Schildt*: C++ : the complete reference*, Osborne McGraw-Hill, c1998. 3rd ed.
- The C++ Resources Network: http://www.cplusplus.com
- Micah Carrick blog: http://www.micahcarrick.com/gtk-glade-tutorial-part-1.html
- GTK+ 3 library: https://developer.gnome.org/gtk3/3.7/
- Galde tutorials: https://wiki.gnome.org/Glade/Tutorials

### *Raspberry, Gertboard and Arduino:*

- Raspberry Fundation: *Raspberry: Quick start guide 2.1*
- Gert van Loo, Myra Vanlnwegen: *Gertboard user manual Rev. 2*, FEN LOGIC LTD 2012
- Gordon Henderson blog: https://www.projects.drogon.net
- The official Arduino support website: http://www.arduino.cc
- Raspberry pi projects: http://www.raspberry-projects.com/pi/pi-operating-systems/raspbian/scripts

***Dispositivo de estimulación y dispositivo de grabación de EMG:***

- Thomas Schauer, Nils-Otto Negaard, Carsten Behling: *ScienceMode, RehaStim™ Stimulation Device - Description and Protocol* - . ***www.hasomed.de****,* 2009

- NeuroLog™ System: *NL820A - Isolator ( 4-channel ) Users Manual*, 1996-8

# ***Anexo I.*** *Stimberry - Embedded Control Interface for RehaStim™ Stimulation Device - Description and Protocol*



## 1 Introduction

**Stimberry** is a embedded control interface for the RehaStim™ stimulation device from the company HASOMED. Stimberry runs on a single board computer (SBC) known as Raspberry Pi. This computer, which has its own dedicated version of Debian Wheezy, fits in the palm of your hand what makes to the system easily portable.

The Stimberry system fixes the parameters of the stimulation, controls the timing of it and, using the communication protocol of RehaStim™ device, sends the instructions to execute though the USB port to the stimulator.

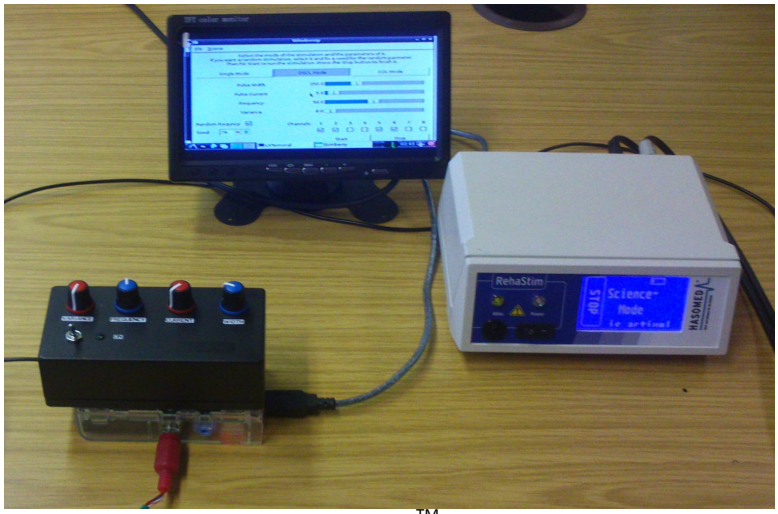



Fig 1. Stimberry Rehstim™ control Interface

## 2 Stimberry control Interface for Rehastim™

The current-controlled 8 channel stimulator RehaStim™ is a certified medical product and possesses two in-dependent current sources which are multiplexed to 4 outputs each. The main processor of the stimulator is an ultra-low-power 16-bit RISC mixed-signal processor from Texas Instruments (MSP430).

There are two independent stimulation modules hosting each one of the current sources. Each stimulation module owns another microprocessor (MSP430) which is responsible for the pulse generation timing. The stimulation module A has the stimulation channels 1 to 4, stimulation module B has the stimulation channels 5 to 8. Since both modules function independently simultaneous pulse generation on module A and B is possible.


Fig 2. RehaStimTM stimulation device

The control interface Stimberry allows select the stimulation mode, the channels to stimulate and the parameters of it: Frequency, Pulse width and Pulse current. However these parameters are the same for every channel.

There are three different modes of stimulation with they own characteristics:

**- Single Mode:** This mode indicates to the stimulation device that it has to stimulate, in that very moment, an specific channel with a specific parameters.

The commands to send from the CPU are simple and not require too much memory. However there is no control of the frequency by the simulator so, that means that the CPU must control the timing of the stimulation. Only channel #1 is available in this mode.

**- CCL Mode:** This time, the RehaStim™ stimulation device is initialised at the beginning, with all the parameters that are necessary for each channel to stimulate, even the frequency of the stimulation, and the RehaStim™ device take care of the timing control. If is necessary change the current or the width of any channel there is a command to updated them.

So, like time the control of the timing is at the stimulation device, the program just have to update the parameters when every few milliseconds, that get free the CPU much longer.

**- OSCL Mode:** Is a similar mode that CCL. There is just one different between them. Now the frequency is not controlled by the RehaStim™ stimulation device. It must be do by the

program, what, although spend more resources from the CPU, allows control the frequency and be able to do it, for example, ramdon.

Being able to select a random frequency could be useful in the researching about the EMG behaviour at the muscles. To make the frequency random, Stimberry use the fourth parameter: Variance. This parameters is used as a maximun value of a pseudo-random sequence with a normal distribution, which takes values between - Variance and Variance.

Besides with the purpose to be able to repeat a experiment with the same conditions, there is a special parameter called "seed" which start the generation of the sequence of pseudo-random. Thus these sequences are repeatable using same seed value.

***NOTE: There are more parameters which controls other features of the stimulation, like be able to use a fraction frequency at some channels or send pulses doubles or triples, but they are not interesting in our case.***

## 3 Graphical Interface

To make easier the stimulation control to the user, Stimberrry has a graphical interface developed on GTK for GNOME desktop environments. This interface allows specify and see on real time the parameters of the stimulation.

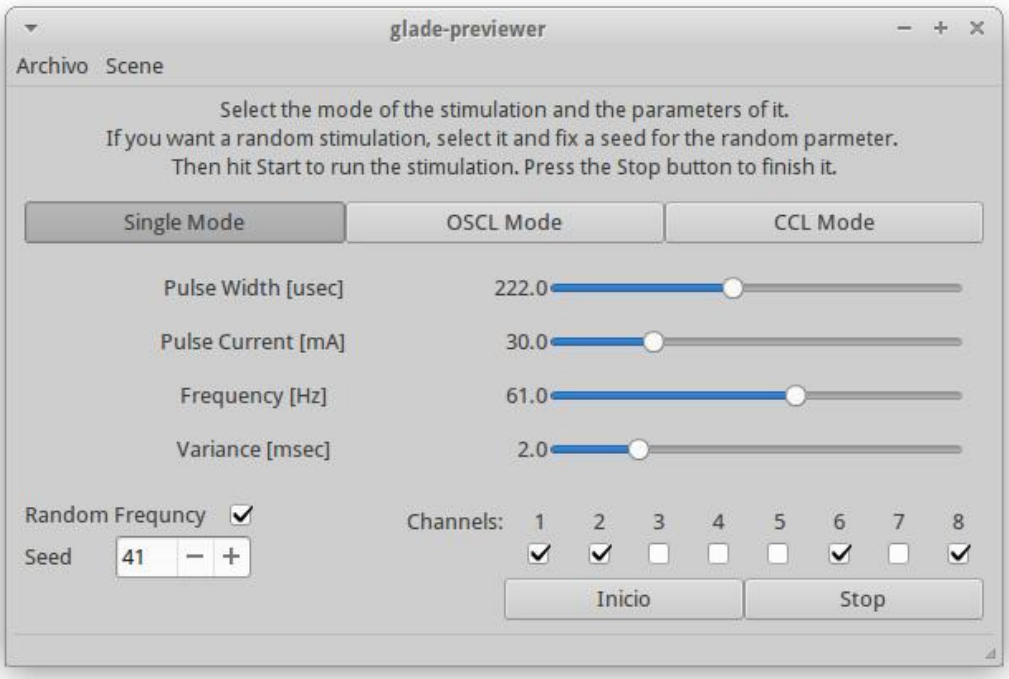



Fig 3. Graphical Interface

The main program can read, write and control the objects and their behaviour thanks to the gtk+ libraries. To do this possible all the objects, their attributes and the signals that they generate, are defined in "*interfaceStimberry*.glade". Every object generate a signal when something interact with it and this signal is handled by the main program which runs a specific function for each signal. In this way the main program has the next structure.

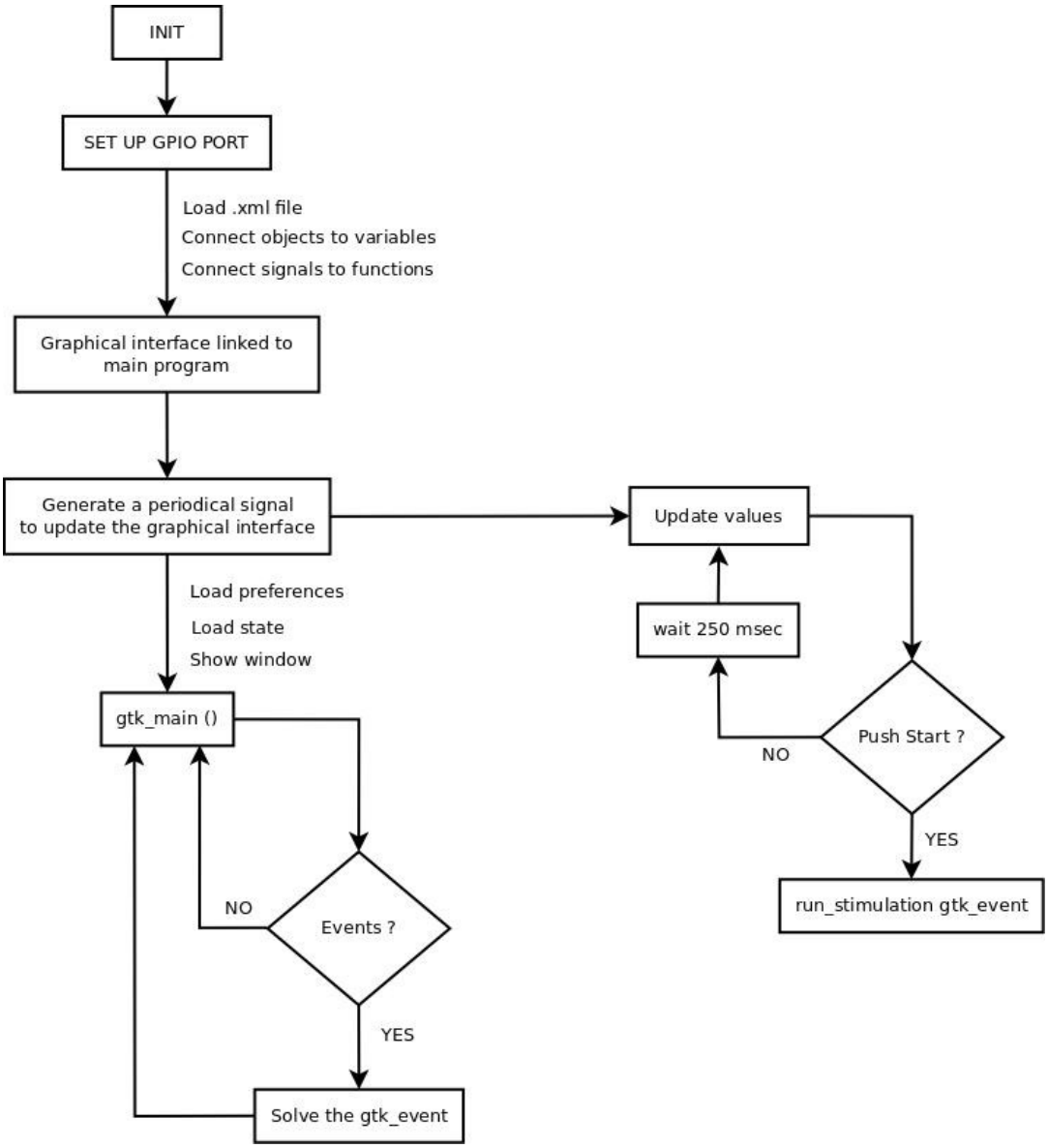



Fig 4. Main program chart

All the interactions between the main program and the objects are controlled by the endless loop gtk_main(). As we'll see at "Control Protocol" section if there is some event meanwhile it's happening other one, this second goes to the queue until the current event returns to the gtk_main.

## 4 Electronic Interface

An important target for the Stimberry Interface was being as embedded as possible. With this purpose, it was designed an specific wired control which allowed use the interface with nothing else that the Raspberry Pi and the controller (No keyboard nor mouse).

Fig 5. Stimberrry controller. Front potentiometers

As it sees at the picture the controller has four potentiometers to specify the stimulation parameters: variance, frequency, pulse current and pulse width, and a switch to start and stop the stimulation. The analog value of these parameters can be recorded on real time by the port behind of every potentiometer.

Fig 6. Stimberrry controller. Front potentiometers

The controller is connected to the Raspberry Pi through the GPIO ports. some of these ports, besides being I/O digital ports, have an alternative function which makes easier the communication.

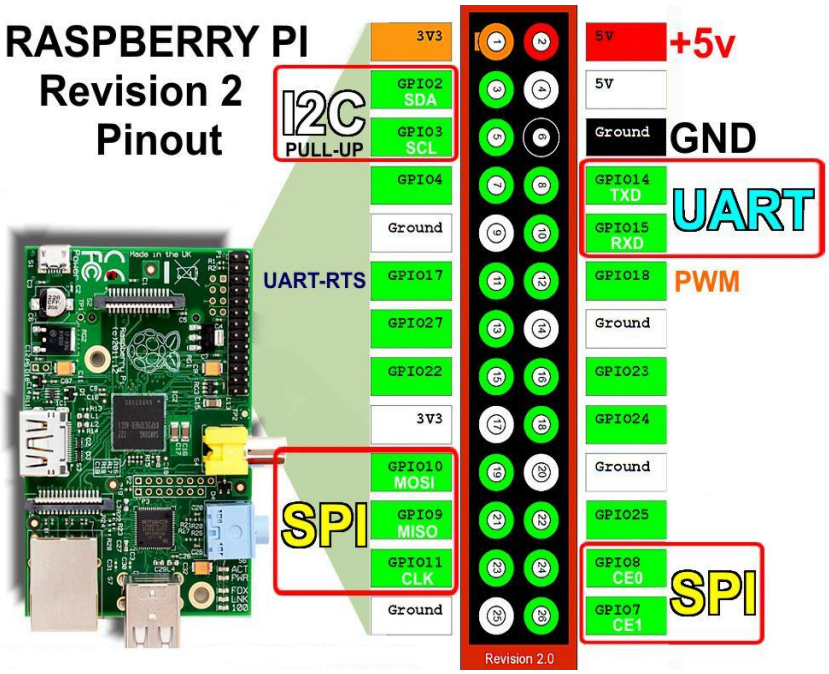



Fig 7. Raspberry Pi I/O ports configuration

The Serial Peripheral Interface or SPI bus is a synchronous serial data link which operates in full duplex mode and allows us read a analog values from the A/D chip. The next schematic shows the structure of the controller and its connections to the Raspberry Pi.

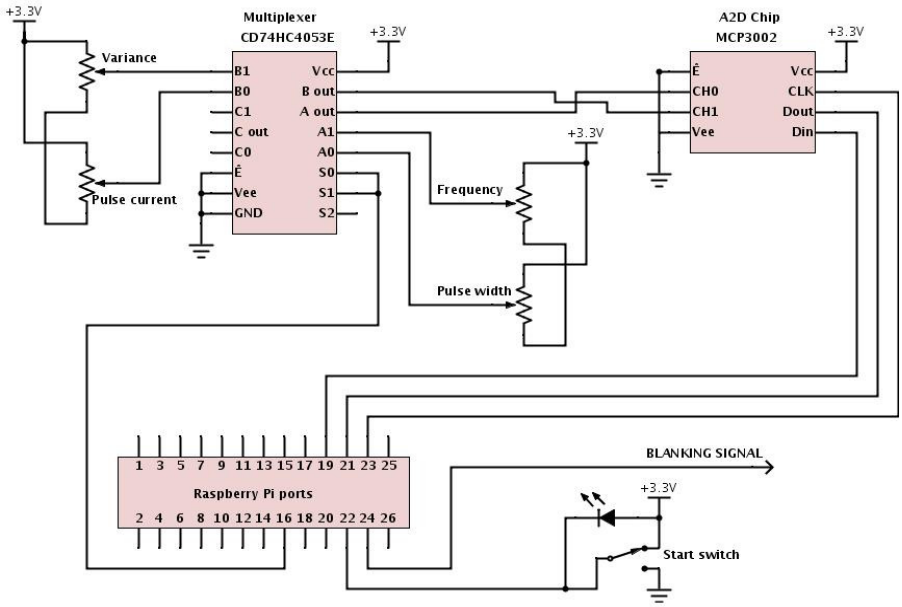



Fig 8. Hardware Schematic

## 5 Control Protocol

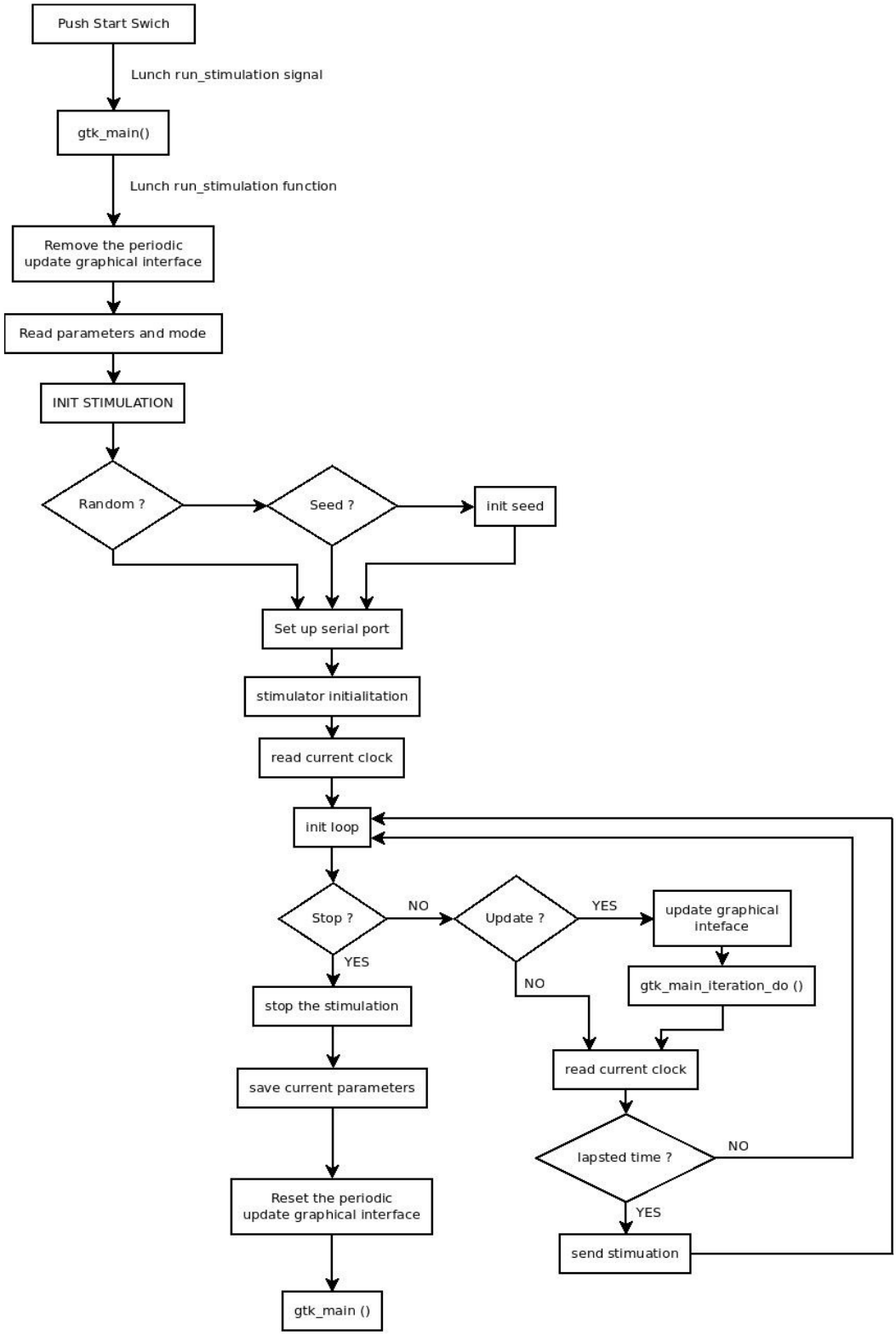



Fig 9. Stimulation Flow Chart

When the start button is pushed, a *gtk_event* is generated and in that moment, the *gtk_main* stops its endless loop and manage this event.

As while is running a stimulation, the *gtk_main* is stopped, the graphical interface is not being updated. So the simulation functions have to, besides control the stimulation timing and their

parameters, refresh the values of the graphical interface. Which makes it through the function *gtk_main_interantion_do,* which does one iteration of the loop, managing the pending events.

## 6 Scenes and Preferences

The file *"PARAMETERS.dat"* contains the values maximun, minumun and step of the parameters: frequency, variance, pulse width and pulse current. Defined here this parameters are used along the entire program. And it also define the width of the blanking signal pulse.

Stimberry Inteface can store five stimulation scenes what can be fixed in a setting up step. For this step we just need a mouse to select the preferences of the stimulation: Mode, Random Frequency, Channels,... and then save the scene in one of the five available stores. Once done this, the next time that the program be started, the last saved scene will be loaded.

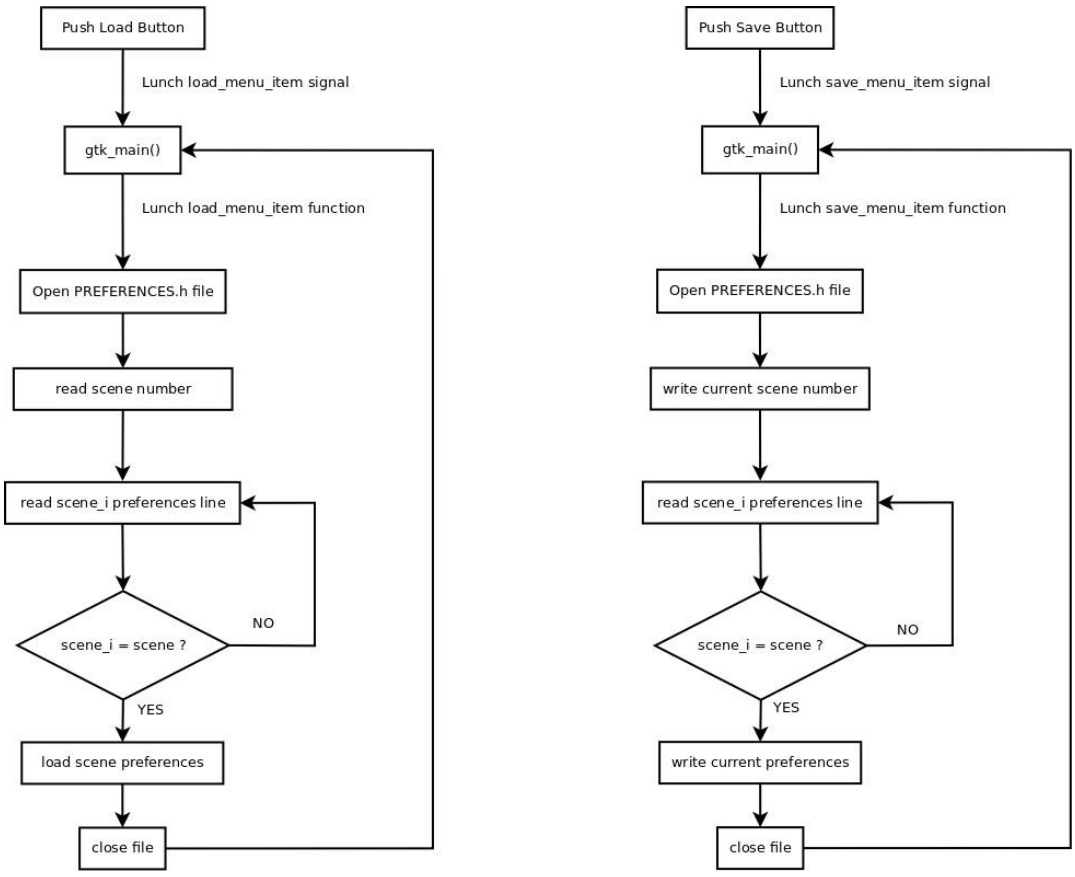



Fig 10. Load and save flow charts

These options allows not just save determinate states to repeat experiments later, if not it also prepare the system to run automatically with anything more but the controller.

## 7 Blanking Signal

Stimberry provides an output square signal to be used as a blanking signal. This signal is just available at the Single Pulse Mode and OSCL Mode, modes where the timing is controlled by the Raspberry Pi.

The blanking signal pulse is generated at the same time that the instruction to stimulate is sent to the stimulator. This happens in the instructions *Send_Update_Parameter* (for the OSCL mode) and *Send_Single_Pulse* (for the Single Pulse mode) both of them defined in the file *"sciencemode.h"*

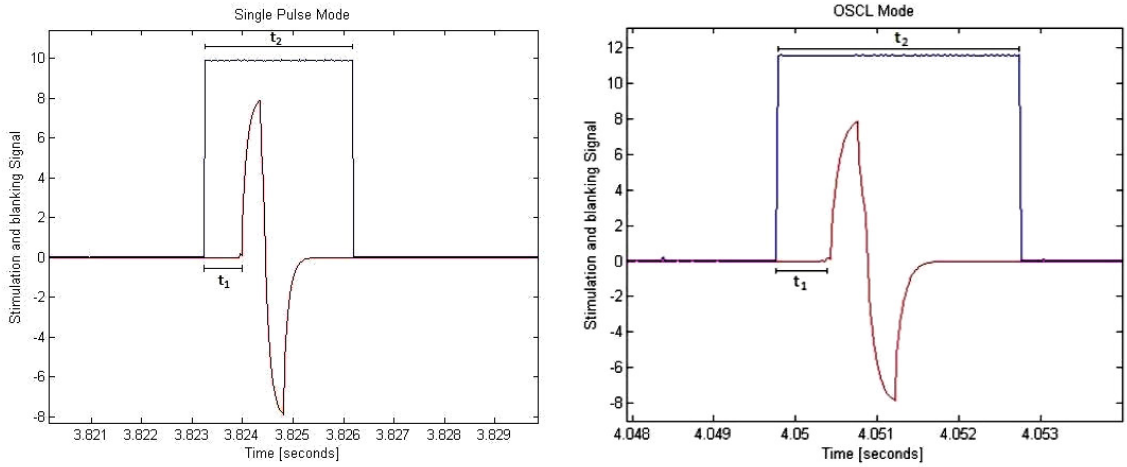



Fig 11. Blanking singnal for Single Pulse and OSCL mode

The image up shows the blanking signal and the stimulation in both modes. This signal, with 3.3 volt of amplitude, is defined by two parameters t1 and t2.

The parameters t1 is not always the same, it oscillate between 460-860 microseconds for the Single Pulse mode and between 400-840 microseconds for OSCL mode.

The other parameter, t2, goes defined by addition of the Pulse Width of the stimulation and the constant MUTE_PULSE (defined at "*PARAMETERS.h"*). This constant can be modified according the specification of the device which needs the blanking signal.

## 8 Libraries

Besides of the C libraries necessaries to use functions own to C, it's necessary include others which contains function and macros to communicate with the stimulator through the USB port and with the controller through the GPIO ports.

"sciencemode.h" defines the stimulator class, and define the flow bytes that compose the instructions for the stimulator.

"serial_linux.h" defines the SerialPort class, and its functions that open, set up and close the USB port, doing possible the communication

"gb_common.h" contains the macros and functions to control the Raspberry Pi I/O ports

"gb_spi.h" contains the macros and functions to control the SPI port which communicates the A/D chip to the Raspberry Pi.

"GPIOport.h" defines the GPIOport class and it used functions from gb_common and gb_spi to connect the controller with the program.

# ***Anexo II:*** *Datasheets*

## *i. Dual Channel 10-Bit A/D Converter*

MICROCHIP

# MCP3002

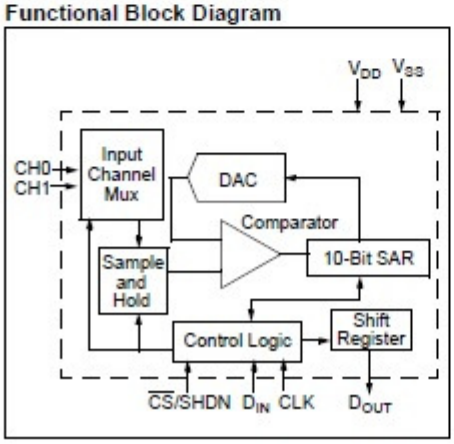
## 2.7V Dual Channel 10-Bit A/D Converter with SPI Serial Interface

- 10-bit resolution
- ±1 LSB maximum DNL
- ±1 LSB maximum INL
- Analog inputs programmable as single-ended or pseudo-differential pairs
- On-chip sample and hold
- SPI serial interface (modes 0,0 and 1,1)
- Single supply operation: 2.7V - 5.5V
- 200 ksps max sampling rate at $V_{DD}$ = 5V
- 75 ksps max sampling rate at $V_{DD}$ = 2.7V
- Low power CMOS technology:
  - 5 nA typical standby current, 2 µA maximum
  - 550 µA maximum active current at 5V
- Industrial temperature range: -40°C to +85°C
- 8-pin MSOP, PDIP, SOIC and TSSOP packages

### Applications

- Sensor Interface
- Process Control
- Data Acquisition
- Battery Operated Systems

### Functional Block Diagram




The MCP3002 is a successive approximation 10-bit analog-to-digital (A/D) converter with on-board sample and hold circuitry.

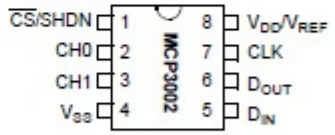
The MCP3002 is programmable to provide a single pseudo-differential input pair or dual single-ended inputs. Differential Nonlinearity (DNL) and Integral Nonlinearity (INL) are both specified at ±1 LSB. Communication with the device is done using a simple serial interface compatible with the SPI protocol. The device is capable of conversion rates of up to 200 ksps at 5V and 75 ksps at 2.7V.

The MCP3002 operates over a broad voltage range, 2.7V to 5.5V. Low-current design permits operation with a typical standby current of 5 nA and a typical active current of 375 µA.

The MCP3002 is offered in 8-pin MSOP, PDIP, TSSOP and 150 mil SOIC packages.

### Package Types

MSOP, PDIP, SOIC, TSSOP



## *ii. High-speed CMOS logic analog multiplexers*

TEXAS INSTRUMENTS

CD/74HC4051, CD54/74HCT4051, CD54/74HC4052,

www.ti.com

SCHS122J – NOVEMBER 1997 – REVISED FEBRUARY 2011

# CD74HCT4052, CD54/74HC4053, CD54/74HC54053
# HIGH-SPEED CMOS LOGIC ANALOG MULTIPLEXERS/DEMULTIPLEXERS

Check for Samples: CD/74HC4051, CD54/74HCT4051, CD54/74HC4052,

## FEATURES

- Wide Analog Input Voltage Range. . ±5 V Max
- Low ON Resistance
  - 70 Ω Typical ($V_{CC}$ – $V_{EE}$ = 4.5 V)
  - 40 Ω Typical ($V_{CC}$ – $V_{EE}$ = 9 V)
- Low Crosstalk Between Switches
- Fast Switching and Propagation Speeds
- Break-Before-Make Switching
- Wide Operating Temperature Range –55°C to 125°C
- CD54HC/CD74HC Types
  - Operation Control Voltage . . . . . 2 V to 6 V
  - Switch Voltage . . . . . 0 V to 10 V
- CD54HCT/CD74HCT Types
  - Operation Control Voltage . . . 4.5 V to 5.5 V
  - Switch Voltage . . . . . 0 V to 10 V
  - Direct LSTTL Input Logic Compatibility $V_{IL}$ = 0.8 V Max, $V_{IH}$ = 2 V Min
  - CMOS Input Compatibility $I_I \leq 1$ µA at $V_{OL}$, $V_{OH}$

## DESCRIPTION

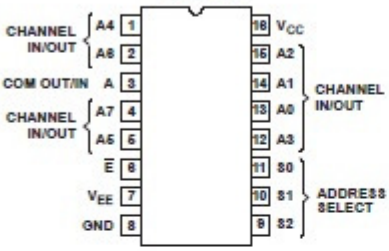
These devices are digitally controlled analog switches which utilize silicon gate CMOS technology to achieve operating speeds similar to LSTTL with the low power consumption of standard CMOS integrated circuits.

These analog multiplexers/demultiplexers control analog voltages that may vary across the voltage supply range (i.e., $V_{CC}$ to $V_{EE}$). They are bidirectional switches thus allowing any analog input to be used as an output and vice-versa. The switches have low ON resistance and low OFF leakages. In addition, all three devices have an enable control which, when high, disables all switches to their OFF state.

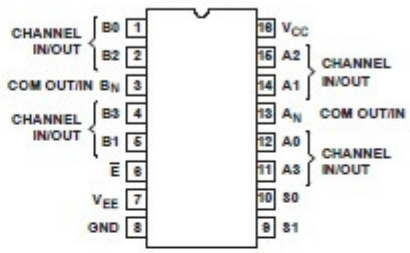
www.ti.com

SCHS122J – NOVEMBER 1997 – REVISED FEBRUARY 2011

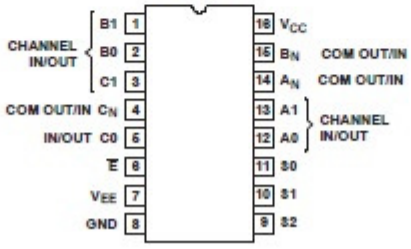
CD54HC4051, CD54HCT4051
(CERDIP)
CD74HC4051
(PDIP, SOIC, SOP, TSSOP)
CD74HCT4051
(PDIP, SOIC)
TOP VIEW




CD54HC4052
(CERDIP)
CD74HC4052
(PDIP, SOIC, SOP, TSSOP)
CD74HCT4052
(PDIP, SOIC)
TOP VIEW




CD54HC4053
(CERDIP)
CD74HC4053
(PDIP, SOIC, SOP, TSSOP)
CD74HCT4053
(PDIP, SOIC, TSSOP)
TOP VIEW



*iii. RehaStim Device - Description and protocol*

# – ScienceMode –
# RehaStim™ Stimulation Device

Description and Protocol

**Thomas Schauer[1], Nils-Otto Negaard, Carsten Behling[2]**

[1] Technische Universität Berlin, Control Systems Group, Germany
E-mail: schauer@control.tu-berlin.de
[2] HASOMED GmbH, Magdeburg, Germany
E-mail: carsten.behling@hasomed.de

21st September 2009

1

– ScienceMode – RehaStim™ Stimulation Device

## 1 Introduction

**ScienceMode** is a serial communication protocol to directly control the 8-channel stimulator RehaStim™ by the company HASOMED GmbH (see http://www.hasomed.de or http://www.rehastim.de/ ) from an external device, preferably a PC, via the standard USB interface. The connection between PC and stimulator corresponds to the USB 1.1 standard. Galvanic isolation is provided by the USB interface of the stimulator. The ScienceMode offers great flexibility for research applications.

## 2 Stimulation Device

The current-controlled 8 channel stimulator RehaStim™ is a certified medical product and possesses two independent current sources which are multiplexed to 4 outputs each. The main processor of the stimulator is an ultra-low-power 16-bit RISC mixed-signal processor from Texas Instruments (MSP430). There are two independent stimulation modules hosting each one of the current sources. Each stimulation module owns another microprocessor (MSP430) which is responsible for the pulse generation timing. The stimulation module A has the stimulation channels 1 to 4, stimulation module B has the stimulation channels 5 to 8. Since both modules function independently simultaneous pulse generation on module A and B is possible.
Figure 1 shows the stimulator device. The device is operated by a touch panel with illumination. The technical specifications are listed in the Table 1. Galvanic isolation between high voltage generation/electrodes and the rest of the stimulator electronics has been additionally realised for safety reasons.


Figure 1: Portable 8 channel stimulator RehaStim.

Table 1: Technical details of the RehaStim stimulation device.

| | |
|---|---|
| Current | 0 . . . 126 mA in 2 mA steps |
| Pulsewidth | 0, 20 . . . 500 $\mu$s in 1 $\mu s$ steps |
| Frequency | see Section 4 |
| Pulse form | biphasic |
| Channels | 8 (2 times 4 on two modules) |
| Serial ports | USB with galvanic isolation |

Figure 2 shows the form of a delivered bi-phasic pulse on an ideal resistive load. Current amplitude and

pulsewidth are defined in the figure. Notice that there is a fixed pause of 100 $\mu s$ between the two phases of the pulse. At the end of the pulse the remaining charge on the electrodes and skin is removed by an active shortcut (change of electrode polarity for 1 $\mu s$).

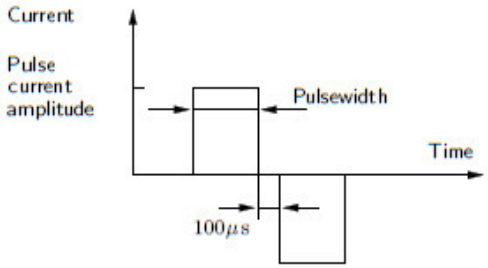



Figure 2: Definition of pulsewidth and current amplitude for a biphasic pulse.

The stimulator provides a skin resistance check for safety reasons. The resistance is determined by analysing the effect of a small test impulse which is send before each stimulation pulse. If the resistance is not inside normal ranges then a stimulation pulse will not be generated.

## 3 How to activate/deactivate the ScienceMode

The operating manual of the RehaStim device should give detailed instructions how to activate/deactivate the ScienceMode.

## 4 Pulse Generation Modes

The stimulator offers different modes of stimulation pulse generation.
The following modes of pulse generation are available:

- **Single Pulse Mode:** On an external command the stimulator generates a single pulse on a specified channel with desired current amplitude and pulsewidth. The stimulator will generate the pulse immediately after processing the command. Complex stimulation patterns may be generated by sending more than one command. The external device (PC) is responsible for controlling the stimulation timing, i.e. the stimulation inter-pulse interval.

- **Continuous Channel List (CCL) Mode:** Using this mode, the generation of complex patterns is greatly simplified. The main processor and the processors of the stimulation modules control the pulse generation by means of timer-interrupts. A list of stimulation channels has to be specified, on which pulses or even pulse groups (doublets or triplets) will repeatedly be generated. Figure 3 defines the main stimulation period $t_1$ and the inter-pulse time $t_2$ of doublets and triplets with the help of an example. The channel list is repeatedly processed with time period $t_1$. Pulse generation takes place on the selected channels, ordered by the channel numbers. For each selected channel a time slot of 1.5 ms is reserved, even if current or/and pulsewidth are zero for the channel. At least 1.5 ms pass between the stimulation of different channels of one module. The stimulation modules A and B process the channel list in parallel with a time offset of 0.6 ms. Module A generates pulses on the channels 1 to 4 if applicable, and module

B generates pulses on the channels 5 to 8 if applicable. The inter-pulse time $t_2$ of doublets and triplets is fixed for all channels and is set during an initialisation step. When doublets or triplets have to be generated the channel list will be processed two or three times with period $t_2$. Stimulation takes place only on the channels on which doublets or triplets have to be generated.

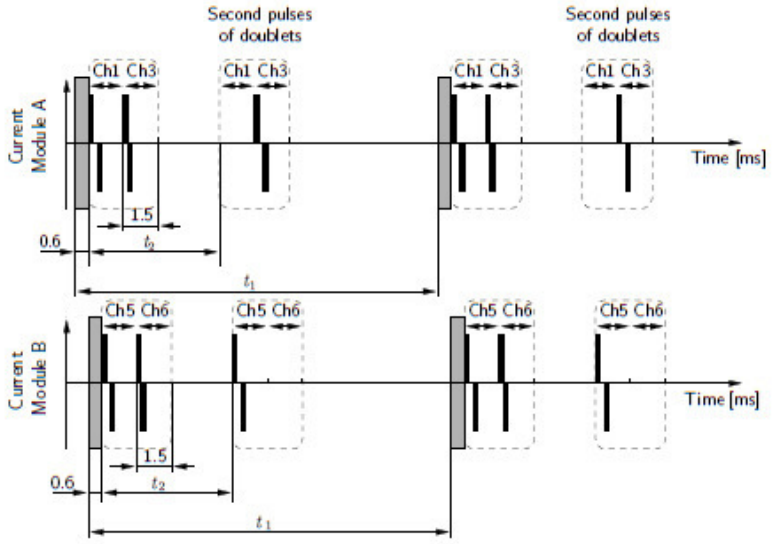



Figure 3: Example for the Continuous Channel List Mode of the stimulator. The channel list encloses the channels 1, 3, 5 and 6. The main stimulation frequency is $1/t_1$. Doublets are generated on the channels 3 and 5 with a frequency $1/t_2$. The grey bars indicate some communication periods in which the actual stimulation settings are transfered from the main controller to the stimulation modules. Stimulation pulses are represented by black bars where the same width and amplitude have been assumed for simplicity.

The main stimulation frequency is specified by the period $t_1$. Some channels of the channel list can be assigned to a lower frequency which has the period $nt_1$ where $n$ is a positive integer.

The CCL mode must be initialised by a command. Used channels, main time $t_1$, inter-pulse time $t_2$ and the maximal size of pulse groups must be chosen at this stage. The minimal possible inter-pulse time $t_2$ depends on the maximal number of channels assigned to the individual stimulation modules as follows

$$t_2 \geq 1.5\,\text{ms} \cdot \max(n_{ch_A}, n_{ch_B}) \tag{1}$$

where $n_{ch_A}$ and $n_{ch_B}$ are the number of selected channels of stimulation modules A and B respectively. The minimal possible period $t_1$ depends on $t_2$ and the maximal size $n_{pg}$ of pulse groups used ($n_{pg} = 1$ for single pulses, $n_{pg} = 2$ for doublets, $n_{pg} = 3$ for triplets). The constraint for $t_1$ is then

$$t_1 \geq n_{pg} \cdot t_2 + 1.5\,\text{ms} \tag{2}$$

where the 1.5 ms above results from a communication between the main processor and the stimulation modules as indicated in Figure 3. The period $t_1$ can be changed in the range $3 \ldots 1023.5$ ms in 0.5 ms steps subject to the constraint (2), and $t_2$ can be altered in the range $3 \ldots 16$ ms in 0.5 ms steps subject to the constraint (1). Using all 8 channels, a minimal group time $t_2$ of 6 ms can be achieved, i.e. a doublet or triplet frequency of 180 Hz. When only doublets and single pulses with a frequency of 180 Hz are

applied ($n_{pg} = 2$), a minimal value of 13.5 ms for period $t_1$ is obtained, i.e. a maximal possible main frequency of 74.1 Hz.

Another external command deactivates the CCL mode. When the CCL mode is active, the pulse parameters (pulsewidth, current amplitude and group mode (single pulse, doublet or triplet) of the selected channels can be altered by a corresponding external command. The new parameters will be used from the next processing of the channel list onwards.

The main processor controls the frequency $1/t_1$ by which the channel list is processed. Each time the stimulation cycle repeats, the main processor sends the actual stimulation settings to the two stimulation modules which then generates the individual pulses on the selected channels. The processors on the stimulation modules guarantee that specified pulsewidths and inter-pulse times of the doubles and triples are realised.

- **One Shot Channel List (OSCL) Mode:** Just as in the Continuous Channel List Mode a channel list is defined by an initialisation step. However, processing of the channel list is not automatically repeated so that the time period $t_1$ ceases to exist. Instead, the channel list will be processed once and pulses and/or pulse groups are generated, only if an external command is issued. Pulse parameters (pulsewidth, current amplitude and group mode of the selected channels) are specified by the external command.

  The OSCL mode offers the possibility to control the main stimulation frequency by an external device while the inter-pulse time of the doublets and triplets is realised by the stimulation modules.

# 5 Protocol

## 5.1 Serial Settings for FTDI driver

Table 2: Serial settings

| Parameter | Value |
|---|---|
| Baudrate | 115200 |
| parity | no |
| data bits | 8 |
| stop bit | 2 |
| flow control RTS/CTS | on |

## 5.2 Overview on commands

All information is contained in packets of one or more bytes conforming with some predefined sequence. In each packet all bytes other than the first have bit 7 clear; the starting byte has bit 7 set.

Table 3: Commands with their 2 bit identification number `Ident`

| No. | Command | Identification number `Ident` |
|---|---|---|
| 1 | Channel list mode initialisation | 00 |
| 2 | Channel list mode update | 01 |
| 3 | Channel list mode stop | 10 |
| 4 | Single pulse generation | 11 |

### 5.3 Channel list mode initialisation command

Table 4: Variables used for channel list mode initialisation command

| Variable | Bits | Value/Range | Description |
|---|---|---|---|
| `Ident` | 2 | 0 | Command identification number |
| `Check` | 3 | 0..7 | Checksum = sum of all logical variables Modulo 8 = (`N_Factor`+`Channel_Stim` +`Channel_Lf`+ `Group_Time`+`Main_Time`) modulo 8 |
| `N_Factor` | 3 | 0..7 | Defines how many times the stimulation is skipped for channels specified in `Channel_Lf` as well as in `Channel_Stim`. 0 = no skip 1 = skip once ... 7 = skip seven times |
| `Channel_Stim` | 8 | 0..255 | Defines the actives channels. Bit 0 corresponds to channel 1. ⋮ Bit 7 corresponds to channel 8. A bit set activates a channel. |
| `Channel_Lf` | 8 | 0..255 | Defines the low frequency channels. Bit 0 corresponds to channel 1. ⋮ Bit 7 corresponds to channel 8. A bit set activates a channel for low frequency stimulation but only if the same bit is also set in `Channel_Stim`. |
| `Group_Time` | 5 | 0..31 | Defines the interpulse-interval $t_{s_2}$ by $t_{s_2} = \texttt{Group\_Time} \cdot 0.5\,\text{ms} + 1.5\,\text{ms}$ |
| `Main_Time` | 11 | 0..2047 | Defines the main time period $t_{s_1}$ by $t_{s_1} = \texttt{Main\_Time} \cdot 0.5\,\text{ms} + 1\,\text{ms}$ |

– ScienceMode – RehaStim™ Stimulation Device

Table 5: Definition of the channel mode list initialisation command

| Byte | Bits | Value | Variable | Bit no. with respect to the variable |
|---|---|---|---|---|
| Byte 1 | 7 | 1 | | |
| | 6 | 0 | Ident | 1 |
| | 5 | 0 | Ident | 0 |
| | 4 | | Check | 2 |
| | 3 | | Check | 1 |
| | 2 | | Check | 0 |
| | 1 | | N_Factor | 2 |
| | 0 | | N_Factor | 1 |
| Byte 2 | 7 | 0 | | |
| | 6 | | N_Factor | 0 |
| | 5 | | Channel_Stim | 7 |
| | 4 | | Channel_Stim | 6 |
| | 3 | | Channel_Stim | 5 |
| | 2 | | Channel_Stim | 4 |
| | 1 | | Channel_Stim | 3 |
| | 0 | | Channel_Stim | 2 |
| Byte 3 | 7 | 0 | | |
| | 6 | | Channel_Stim | 1 |
| | 5 | | Channel_Stim | 0 |
| | 4 | | Channel_Lf | 7 |
| | 3 | | Channel_Lf | 6 |
| | 2 | | Channel_Lf | 5 |
| | 1 | | Channel_Lf | 4 |
| | 0 | | Channel_Lf | 3 |
| Byte 4 | 7 | 0 | | |
| | 6 | | Channel_Lf | 2 |
| | 5 | | Channel_Lf | 1 |
| | 4 | | Channel_Lf | 0 |
| | 3 | X | | |
| | 2 | X | | |
| | 1 | | Group_Time | 4 |
| | 0 | | Group_Time | 3 |
| Byte 5 | 7 | 0 | | |
| | 6 | | Group_Time | 2 |
| | 5 | | Group_Time | 1 |
| | 4 | | Group_Time | 0 |
| | 3 | | Main_Time | 10 |
| | 2 | | Main_Time | 9 |
| | 1 | | Main_Time | 8 |
| | 0 | | Main_Time | 7 |
| Byte 6 | 7 | 0 | | |
| | 6 | | Main_Time | 6 |
| | 5 | | Main_Time | 5 |
| | 4 | | Main_Time | 4 |
| | 3 | | Main_Time | 3 |
| | 2 | | Main_Time | 2 |
| | 1 | | Main_Time | 1 |
| | 0 | | Main_Time | 0 |

8

– ScienceMode – RehaStim™ Stimulation Device

### 5.4 Channel list mode update command

Table 6: Variables used for channel list mode update command

| Variable | Bits | Value/Range | Description |
|---|---|---|---|
| `Ident` | 2 | 1 | Command identification number |
| `Check` | 5 | 0..31 | Checksum = sum of all logical variables Modulo 32 = (`Mode`+`Pulse_Width`+`Pulse_Current`) modulo 32 |
| `Mode` | 2 | 0..2 | `Mode` = 0: generate single pulse<br>`Mode` = 1: generate doublet<br>`Mode` = 2: generate triplet |
| `Pulse_Width` | 9 | 0,10..500 | Pulse width in $\mu$s |
| `Pulse_Current` | 7 | 0..127 | Current in mA |

– ScienceMode – RehaStim™ Stimulation Device

Table 7: Definition of the channel mode update command

| Byte | Bits | Value | Variable | Bit no. with respect to the variable |
|---|---|---|---|---|
| Byte 1 | 7 | 1 | | |
| | 6 | 0 | Ident | 1 |
| | 5 | 1 | Ident | 0 |
| | 4 | | Check | 4 |
| | 3 | | Check | 3 |
| | 2 | | Check | 2 |
| | 1 | | Check | 1 |
| | 0 | | Check | 0 |
| For each channel activated in the channel list, the next three bytes are send in increasing order with respect to the channel number. | | | | |
| Byte 2 | 7 | 0 | | |
| | 6 | | Mode | 1 |
| | 5 | | Mode | 0 |
| | 4 | X | | |
| | 3 | X | | |
| | 2 | X | | |
| | 1 | | Pulse_Width | 8 |
| | 0 | | Pulse_Width | 7 |
| Byte 3 | 7 | 0 | | |
| | 6 | | Pulse_Width | 6 |
| | 5 | | Pulse_Width | 5 |
| | 4 | | Pulse_Width | 4 |
| | 3 | | Pulse_Width | 3 |
| | 2 | | Pulse_Width | 2 |
| | 1 | | Pulse_Width | 1 |
| | 0 | | Pulse_Width | 0 |
| Byte 4 | 7 | 0 | | |
| | 6 | | Pulse_Current | 6 |
| | 5 | | Pulse_Current | 5 |
| | 4 | | Pulse_Current | 4 |
| | 3 | | Pulse_Current | 3 |
| | 2 | | Pulse_Current | 2 |
| | 1 | | Pulse_Current | 1 |
| | 0 | | Pulse_Current | 0 |
| ⋮ | | | | |

### 5.5 Channel list mode stop command

Table 8: Variables used for channel list mode stop command

| Variable | Bits | Value/Range | Description |
|---|---|---|---|
| Ident | 2 | 2 | Command identification number |
| Check | 5 | 0..31 | Checksum =<br>sum of all logical variables Modulo 32 =<br>(0) modulo 32 = 0 |

Table 9: Definition of the channel mode stop command

| Byte | Bits | Value | Variable | Bit no. with respect to the variable |
|---|---|---|---|---|
| Byte 1 | 7 | 1 | | |
| | 6 | 1 | Ident | 1 |
| | 5 | 0 | Ident | 0 |
| | 4 | 0 | Check | 4 |
| | 3 | 0 | Check | 3 |
| | 2 | 0 | Check | 2 |
| | 1 | 0 | Check | 1 |
| | 0 | 0 | Check | 0 |

### 5.6 Single pulse generation command

Table 10: Variables used for single pulse generation command

| Variable | Bits | Value/Range | Description |
|---|---|---|---|
| Ident | 2 | 3 | Command identification number |
| Check | 5 | 0..31 | Checksum =<br>sum of all logical variables Modulo 32 =<br>(Channel_Number +<br>Pulse_Width + Pulse_Current)<br>modulo 32 |
| Channel_Number | 3 | 0..7 | Channel_Number = 0 is channel no. 1<br>:<br>Channel_Number = 7 is channel no. 8 |
| Pulse_Width | 9 | 0,10..500 | Pulse width in $\mu$s |
| Pulse_Current | 7 | 0..127 | Current in mA |

– ScienceMode – RehaStim™ Stimulation Device

Table 11: Definition of the single pulse generation command

| Byte | Bits | Value | Variable | Bit no. with respect to the variable |
|---|---|---|---|---|
| Byte 1 | 7 | 1 | | |
| | 6 | 1 | Ident | 1 |
| | 5 | 1 | Ident | 1 |
| | 4 | | Check | 4 |
| | 3 | | Check | 3 |
| | 2 | | Check | 2 |
| | 1 | | Check | 1 |
| | 0 | | Check | 0 |
| Byte 2 | 7 | 0 | | |
| | 6 | | Channel_Number | 2 |
| | 5 | | Channel_Number | 1 |
| | 4 | | Channel_Number | 0 |
| | 3 | X | | |
| | 2 | X | | |
| | 1 | | Pulse_Width | 8 |
| | 0 | | Pulse_Width | 7 |
| Byte 3 | 7 | 0 | | |
| | 6 | | Pulse_Width | 6 |
| | 5 | | Pulse_Width | 5 |
| | 4 | | Pulse_Width | 4 |
| | 3 | | Pulse_Width | 3 |
| | 2 | | Pulse_Width | 2 |
| | 1 | | Pulse_Width | 1 |
| | 0 | | Pulse_Width | 0 |
| Byte 4 | 7 | 0 | | |
| | 6 | | Pulse_Current | 6 |
| | 5 | | Pulse_Current | 5 |
| | 4 | | Pulse_Current | 4 |
| | 3 | | Pulse_Current | 3 |
| | 2 | | Pulse_Current | 2 |
| | 1 | | Pulse_Current | 1 |
| | 0 | | Pulse_Current | 0 |

### 5.7 Acknowledgement by the stimulator

Table 12: Acknowledgement byte

| Byte | Bits | Value | Variable | Bit no. with respect to the variable |
|---|---|---|---|---|
| Byte 1 | 7 | | `Ident` | 1 |
| | 6 | | `Ident` | 0 |
| | 5 | | | |
| | 4 | | | |
| | 3 | | | |
| | 2 | | | |
| | 1 | | | |
| | 0 | 1 = OK, 0 = error | `Error_Code` | 0 |

### 5.8 Examples

*Single pulse example 1*

Sending a pulse on channel 3 with a pulse width of 200 $\mu$s (binary 011001000) and a current of 120 mA (binary 1111000). Hence, `Channel_Number` is 2 (binary 010). In this case, the checksum is $(2 + 200 + 120)\,\text{modulo}\,32 = 2$ (binary 00010). In binary format, the modulo 32 operation can be easily performed by just taking the 5 LSB of the sum 2+200+120 (binary 101000010). The byte sequence for the command is:

Byte 0 – 11100010
Byte 1 – 0010XX01
Byte 2 – 01001000
Byte 3 – 01111000

The bits with XX do not have a meaning. In hex code the command is E2 21 48 78.
The return value would be in hex C1 (binary 11000001) if no error occurred else C0 (11000000).

*Single pulse example 2*

Sending a pulse on channel 6 with a pulse width of 221 $\mu$s (binary 011011101) and a current of 55 mA (binary 0110111). Hence, `Channel_Number` is 5 (binary 101). The checksum is in this case $(5 + 221 + 51)\,\text{modulo}\,32 = 25$ (binary 11001). In binary format, the modulo 32 operation can be easily performed by just taking the 5 LSB of the sum 5+221+55 (binary 100011001). The byte sequence for the command is:

Byte 0 – 11111001
Byte 1 – 0101XX01
Byte 2 – 01011101
Byte 3 – 00110111

The bits with XX do not have a meaning. In hex code the command is F9 51 5D 37.
The return value would be in hex C1 (binary 11000001) if no error occurred else C0 (11000000).

*Channel list mode initialisation example 1*

– ScienceMode – RehaStim™ Stimulation Device

The channel list mode shall be initialised with the following parameters:
`Main_Time`=98 (binary 00001100010) gives $t_{s1}$=50 ms
`Group_Time`=7 (binary 00111) gives $t_{s2}$=5 ms
`N_Factor`=1 (binary 001)
Channels to be activated are 1,2 and 5 whereas channel 5 runs with lower frequency. This leads to `Channel_Stim`= (binary 0001 0011) = 19 and `Channel_Lf`= (binary 0001 0000) = 16. The checksum is given as 98+7+1+19+16 modulo 8 = 5 (binary 101). In binary format, the modulo 8 operation can be easily performed by just taking the 3 LSB of the sum 98+7+1+19+16 = 141 (binary 10001101). The command byte sequence is then

Byte 1 – 10010100
Byte 2 – 01000100
Byte 3 – 01100010
Byte 4 – 0000XX00
Byte 5 – 01110000
Byte 6 – 01100010

In hex code the command is 94 44 62 0 70 62.

*Channel list mode initialisation example 2*

The channel list mode shall be initialised with the following parameters:
`Main_Time`=31 (binary 00000011111) gives $t_{s1}$=16.5 ms
`Group_Time`=9 (binary 01001) gives $t_{s2}$=6 ms
`N_Factor`=2 (binary 010)
Channels to be activated are 2,3,6 and 8 whereas the channel 2 and 3 run with lower frequency. This leads to `Channel_Stim`= (binary 1010 0110) = 166 and `Channel_Lf`= (binary 0000 0110) = 6. The checksum is given as 31+9+166+6+2 modulo 8 = 6 (binary 110). In binary format, the modulo 8 operation can be easily performed by just taking the 3 LSB of the sum 31+9+166+6+2 = 141 (binary 11010110). The command byte sequence is then

Byte 1 – 10011001
Byte 2 – 00101001
Byte 3 – 01000000
Byte 4 – 0110XX01
byte 5 – 00010000
byte 6 – 00011111

In hex code the command is 99 29 40 61 10 1F.

*Channel list mode update example*

This example is for the initialisation example 2 above.

Pulse width channel 2: 100 (001100100)
Pulse width channel 3: 200 (011001000)
Pulse width channel 6: 300 (100101100)
Pulse width channel 8: 400 (110010000)

Current amplitude channel 2: 52 (0110100)

– ScienceMode – RehaStim™ Stimulation Device

Current amplitude channel 3: 55 (0110111)
Current amplitude channel 6: 72 (1001000)
Current amplitude channel 8: 92 (1011100)

Mode channel 2: 0 (00)
Mode channel 3: 2 (10)
Mode channel 6: 1 (01)
Mode channel 8: 1 (01)

The checksum is given by 100+200+300+400+52+55+72+92+0+2+1+1 modulu 32 = 1+2+8+16 = 28 (binary 11011).

In binary format, the modulo 32 operation can be easily performed by just taking the 5 LSB of the sum 100+200+300+400+52+55+72+92+0+2+1+1 = 1275 (10011111011).

The command byte sequence is then

Byte 0 – 10111011
Byte 1 – 000XXX00
Byte 2 – 01100100
Byte 3 —00110100
Byte 4 – 010XXX01
Byte 5 – 01001000
Byte 6 – 00110111
Byte 7 – 001XXX10
Byte 8 – 00101100
Byte 9 – 01001000
Byte 10 – 001XXX11
Byte 11 – 0010000
Byte 12 – 01011100

In hex code the command is BB 00 64 34 41 48 37 22 2C 48 23 10 5C.

15

## *iv. Gertboard Overview*

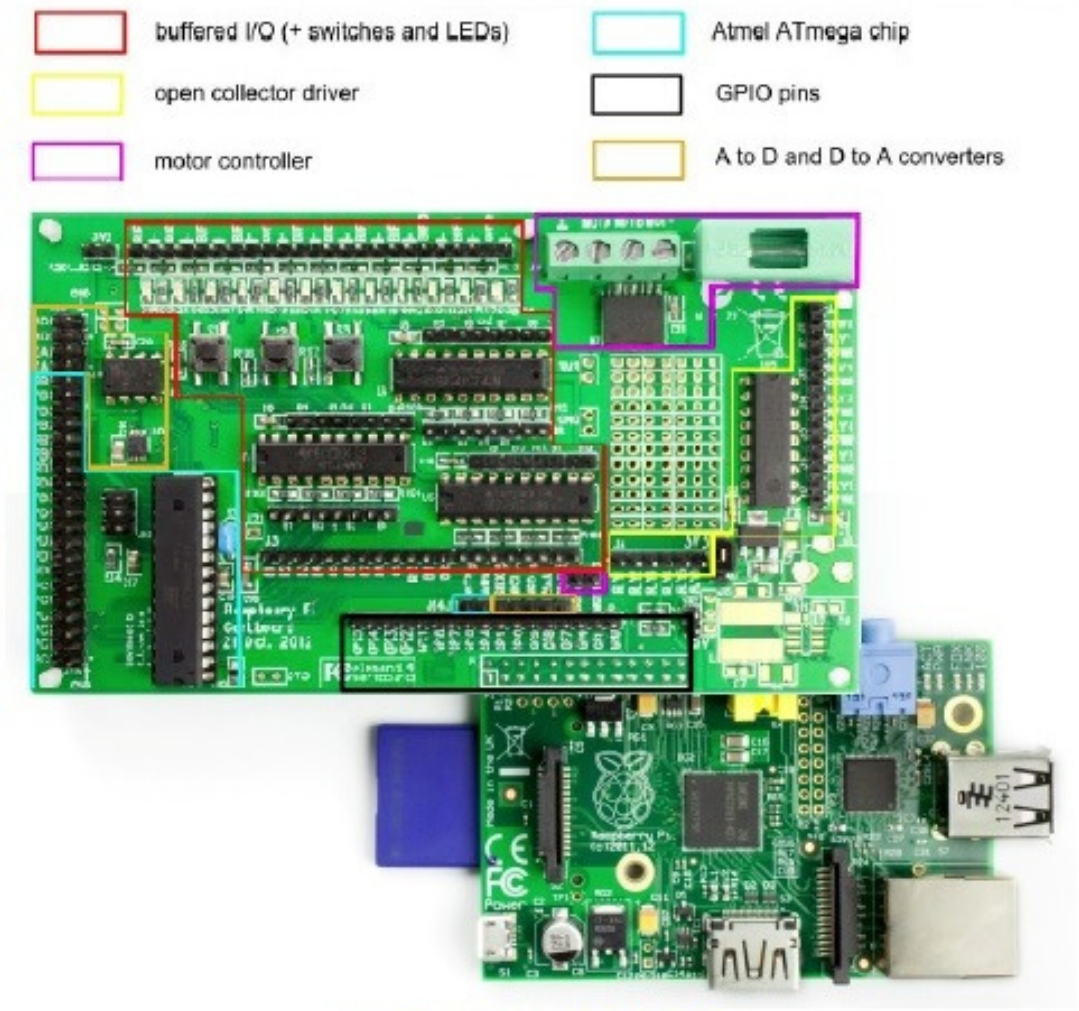



**Figure 1: Gertboard and Raspberry Pi**

The Gertboard is an input/output (I/O) extension board for the Raspberry Pi computer. It fits onto the GPIO (general purpose I/O) pins of the Raspberry Pi (the double row of pins on the upper left corner) via a socket on the back of the Gertboard. A bit of care is required when putting the two devices together. It is easy to insert just one row of pins into the socket, but all of the pins need to be connected. The Gertboard gets its power from those GPIO pins, so you will need a power supply for the Raspberry Pi (RPi) that is capable of supplying a current of at least 1A.

The Gertboard has collection of functional blocks (the major capabilities of the board) which can be connected together in a myriad of ways using header pins. The functional blocks are:

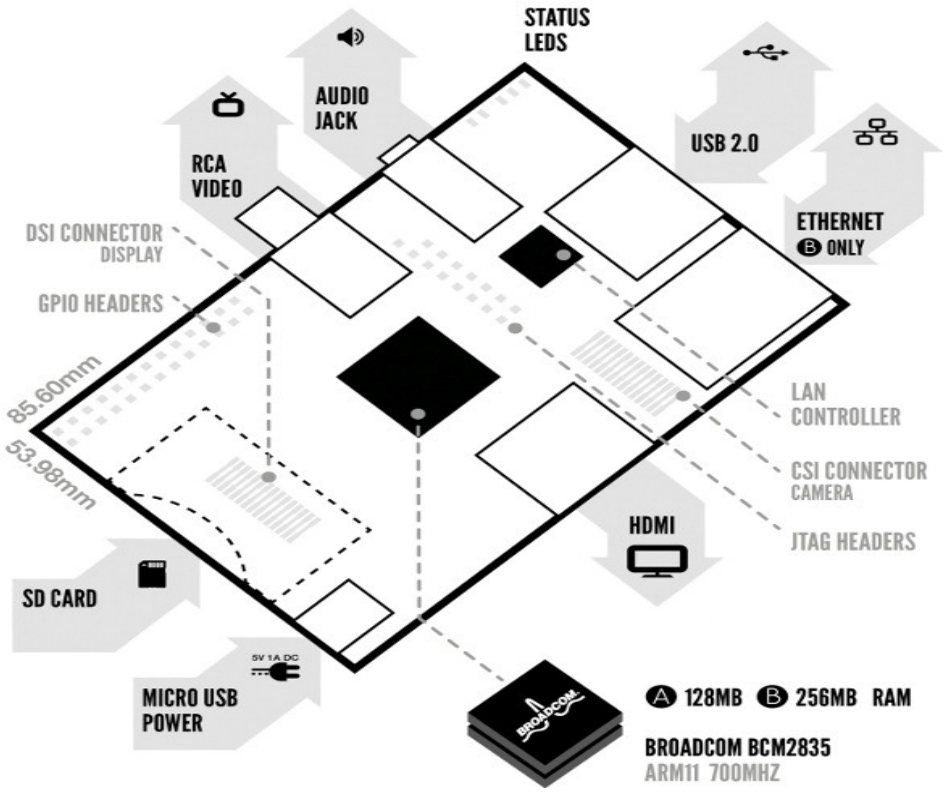
- 12x buffered I/O
- 3x pushbuttons
- 6x open collector drivers (50V, 0.5A)
- 18V, 2A motor controller
- 28-pin dual in line ATmega microcontroller
- 2-channel 8, 10, or 12 bit Digital to Analogue converter
- 2-channel 10 bit Analogue to Digital converter

*v. Raspberry Pi*

# Raspberry Pi

# Getting Started Guide

Raspberry Pi
Model A B




## What is it and how does it work?

The Raspberry Pi is a tiny, credit-card sized single board computer that is truly on the edge of innovation. It uses an HDMI cable to connect to a traditional monitor or TV. This capable little PC can be used for many of the things that your desktop PC does, like building spreadsheets, word-processing, and gaming. It even plays highdefinition video. Unlike a traditional PC, the design does not include a built-in hard disk or solid-state drive. Instead it relies on an SD card for booting and long-term storage. This tiny piece of bare-bones hardware was originally commissioned to teach basic computer skills to students.

## Where did it come from?

The idea behind a tiny and cheap computer for kids came in 2006, when Eben Upton, Rob Mullins, Jack Lang and Alan Mycroft, based at the University of Cambridge's Computer Laboratory, became concerned about the decline in the numbers and skills levels of the A Level students applying to read Computer Science. By 2008, processors designed for mobile devices were becoming more affordable, and powerful enough to provide excellent multimedia. The project started to look very realizable when the Raspberry Pi Foundation was formed. Three years later, the Raspberry Pi Model B entered mass production through licensed manufacture deals with element 14/Premier Farnell and RS Electronics, and within a year it had sold over one million units.

# 1. Raspberry Pi Basic Hardware Setup

## 1.1. Extra Hardware You Will Need

The Raspberry Pi board contains a processor and graphics chip, program memory (RAM) and various interfaces and connectors for external devices. Some of these devices are essential, others are optional. RPi operates in the same way as a standard PC, requiring a keyboard for command entry, a display unit and a power supply.

It also requires 'mass-storage', but a hard disk drive of the type found in a typical PC is not really in keeping with the miniature size of RPi. Instead we will use an SD Flash memory card normally used in digital cameras, configured in such a way to 'look like' a hard drive to RPi's processor. RPi will
'boot' (load the Operating System into RAM) from this card in the same way as a PC 'boots up' into
Windows from its hard disk.

The following are essential to get started:

- SD card containing Linux Operating system
- USB keyboard
- TV or monitor (with HDMI, DVI, Composite or SCART input)
- Power supply (see Section 1.6 below)
- Video cable to suit the TV or monitor used

Recommended optional extras include:

- USB mouse
- Internet connection, Model A or B: USB WiFi adaptor
- Internet connection, Model B only: LAN (Ethernet) cable
- Powered USB hub
- Case

## 1.2. Connecting Everything Together

1. Plug the preloaded SD Card into the RPi.
2. Plug the USB keyboard and mouse into the RPi, perhaps via a USB hub. Connect the Hub to power, if necessary.
3. Plug a video cable into the screen (TV or monitor) and into the RPi.
4. Plug your extras into the RPi (USB WiFi, Ethernet cable, external hard drive etc.). This is where you may really need a USB hub.
5. Ensure that your USB hub (if any) and screen are working.
6. Plug the power supply into the mains socket.
7. With your screen on, plug the power supply into the RPi microUSB socket.
8. The RPi should boot up and display messages on the screen.

It is always recommended to connect the MicroUSB power to the unit last (while most connections can be made live, it is best practice to connect items such as displays with the power turned off).

The RPi may take a long time to boot when powered-on for the first time, so be patient!

## 1.3. Operating System SD Card

As the RPi has no internal mass storage or built-in operating system it requires an SD card preloaded with a version of the Linux Operating System.

- You can create your own preloaded card using any suitable SD card (4GBytes or above) you have to hand. We suggest you use a new blank card to avoid arguments over lost pictures.
- Preloaded SD cards will be available from the RPi Shop.

## 1.4. Keyboard & Mouse

Most standard USB keyboards and mice will work with the RPi. Wireless keyboard/mice should also function, and only require a single USB port for an RF dongle. In order to use a Bluetooth keyboard or mouse you will need a Bluetooth USB dongle, which again uses a single port.

Remember that the Model A has a single USB port and the Model B has two (typically a keyboard and mouse will use a USB port each).

## 1.5. Display

There are two main connection options for the RPi display, *HDMI* (High Definition) and *Composite*
(Standard Definition).

- HD TVs and many LCD monitors can be connected using a full-size 'male' HDMI cable, and with an inexpensive adaptor if DVI is used. HDMI versions 1.3 and 1.4 are supported and a version 1.4 cable is recommended. The RPi outputs audio and video via HMDI, but does not support HDMI input.
- Older TVs can be connected using Composite video (a yellow-to-yellow RCA cable) or via SCART (using a Composite video to SCART adaptor). Both PAL and NTSC format TVs are supported.

When using a composite video connection, audio is available from the 3.5mm jack socket, and can be sent to your TV, headphones or an amplifier. To send audio to your TV, you will need a cable which adapts from 3.5mm to double (red and white) RCA connectors.

**Note: There is no analogue VGA output available. This is the connection required by many computer monitors, apart from the latest ones. If you have a monitor with only a D-shaped plug containing 15 pins, then it is unsuitable.**

## 1.6. Power Supply

The unit is powered via the microUSB connector (only the power pins are connected, so it will not transfer data over this connection). A standard modern phone charger with a microUSB connector will do, providing it can supply at least **700mA at +5Vdc**. Check your power supply's ratings carefully. Suitable mains adaptors will be available from the RPi Shop and are recommended if you are unsure what to use.

**Note: The individual USB ports on a powered hub or a PC are usually rated to provide 500mA maximum. If you wish to use either of these as a power source then you will need a special cable which plugs into two ports providing a combined current capability of 1000mA.**

## 1.7. Cables

You will need one or more cables to connect up your RPi system.

- Video cable alternatives:
  - HDMI-A cable
  - HDMI-A cable + DVI adapter
  - Composite video cable
  - Composite video cable + SCART adaptor
- Audio cable (not needed if you use the HDMI video connection to a TV)
- Ethernet/LAN cable (Model B only)

## 1.8. Additional Peripherals

You may decide you want to use various other devices with your RPi, such as Flash Drives/Portable, Hard Drives, Speakers etc.

### 1.8.1. Internet Connectivity

This may be via an Ethernet/LAN cable (standard RJ45 connector) or a USB WiFi adaptor. The RPi Model B Ethernet port is auto-sensing which means that it may be connected to a router or directly to another computer (without the need for a crossover cable).

### 1.8.2. USB hub

In order to connect additional devices to the RPi, you may want to obtain a USB hub, which will allow multiple devices to be used. It is recommended that a **powered** hub is used - this will provide any additional power to the devices without affecting the RPi itself. A USB 2.0 model is recommended. USB 1.1 is fine for keyboards and mice, but may not be fast enough for other accessories.

### 1.8.3. Case

Since the RPi is supplied without a case, it will be important to ensure that you do not use it in places where it will come into contact with conductive metal or liquids, unless suitably protected.

### 1.8.4. Expansion & Low-Level Peripherals

If you plan on making use of the low-level interfaces available on the RPi, then ensure you have a suitable plug for the GPIO header pins.

Also if you have a particular low-level project in mind, then ensure you design-in suitable protection circuits to keep your RPi safe.

# 2. Raspberry Pi Advanced Setup (Geeks only)

## 2.1. Finding hardware and setting up

You'll need a preloaded SD card, USB keyboard, TV/Monitor (with HDMI/ DVI/ Composite/SCART input), and power supply (USB charger or a USB port from a powered USB Hub or another computer). You'll likely also want a USB mouse, a case, and a USB hub (a necessity for Model A).

A powered USB hub will reduce the demand on the RPi. To connect to the Internet, you'll need either an Ethernet/LAN cable (Model B) or a USB WiFi adaptor (either model). When setting up, it is advisable to connect the power after everything else is ready.

## 2.2. Serial connection

The Serial Port is a simple and uncomplicated method to connect to the RPi. The communication depends on byte wise data transmission, is easy to setup and is generally available even before boot time.

### 2.2.1. First interaction with the board

Connect the serial cable to the COM port in the RPi, and connect the other end to the COM port or
USB Serial Adapter in the computer.

### 2.2.2. Serial Parameters

The following parameters are needed to connect to the RPi. All parameters except **Port_Name** and **Speed** are default values and may not need to be set.

**Port_Name**: Linux automatically assigns different names for different types of serial connectors. Choose your option:

- Standard Serial Port: ttyS0 ... ttySn
- USB Serial Port Adapter: ttyUSB0 ... ttyUSBn
- Speed: 115200
- Bits: 8
- Parity: None
- Stop Bits: 1
- Flow Control: None

The Serial Port is generally usable by the users in the group **dialout**. To add oneself to the group **dialout** the the following command needs to be executed with **root** privileges:

```
$useradd -G {dialout} your_name
```

**2.2.3. Super Easy Way using GNU Screen** Enter the command below into a terminal window `screen Port_Name 115200`

#### 2.2.4. Super Easy Way using Minicom

Run minicom with the following parameters:

```
minicom -b 115200 -o -D Port_Name
```

#### 2.2.5. GUI method with GtkTerm

Start *GtkTerm*, select Configuration->Port and enter the values above in the labelled fields.

#### 2.2.6. Windows Users

Windows 7 or Vista users must download putty or a comparable terminal program. Users of XP and below can choose between using *putty* and *Hyperterminal.*

#### 2.2.7. First Dialog

If you get the prompt below, you are connected to the Raspberry Pi shell!

```
prompt> #
```

First command you might want try is "help":

```
prompt> # help
```

If you get some output, you are correctly connected to the Raspberry Pi! Congratulations!

## 2.3. SD card setup

Now we want to install a GNU/Linux distro on an SD card and make space for our stuff. You can use either an SD or SDHC card. In the latter case of course take care that your PC card reader also supports SDHC. Be aware that you are not dealing with an x86 processor, but instead a completely different architecture called ARM, so don't forget to install the ARM port for the distro you are planning to use.

#### 2.3.1. Formatting the SD card via the mkcard.txt script

1. Download **mkcard.txt**.
2. $ chmod +x mkcard.txt
3. $ ./mkcard.txt /dev/sd*x*, where *x* is the letter of the card. You can find this by inserting your card and then running dmesg | tail.

You should see the messages about the device being mounted in the log. Mine mounts as **sdc**. Once run, your card should be formatted.

#### 2.3.2. Formatting the SD card via fdisk "Expert mode"

First, lets clear the partition table:

```
================================================================================

$ sudo fdisk /dev/sdb

Command (m for help): o
```

```
Building a new DOS disklabel. Changes will remain in memory only, until you
decide to write them. After that, of course, the previous content won't be
recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

================================================================================
```

Print card info:

```
================================================================================

Command (m for help): p

Disk /dev/sdb: 128 MB, 128450560 bytes
....

================================================================================
```

Make a note of the card size in bytes. You will need it later. Then go into "Expert mode":

```
================================================================================

Command (m for help): x

================================================================================
```

Now we want to set the geometry to 255 heads, 63 sectors and calculate the number of cylinders required for the particular SD/MMC card:

```
================================================================================

Expert command (m for help): h
Number of heads (1-256, default 4): 255

Expert command (m for help): s
Number of sectors (1-63, default 62): 63
Warning: setting sector offset for DOS compatiblity

================================================================================
```

NOTE: Be especially careful in the next step. First calculate the number of cylinders as follows: C

= B / 255 / 63 / 512 where

- B = Card size in bytes (The number you wrote down earlier.)
- C = Number of cylinders

When you get the number, round it DOWN. Thus, if you got C = 108.8 you'll be using 108 cylinders.

```
================================================================================

Expert command (m for help): c
Number of cylinders (1-1048576, default 1011): 15

================================================================================
```

In this case a 128MB card is used (reported as 128450560 bytes by *fdisk* above), thus 128450560 / 255 / 63 / 512 = 15.6 rounded down to 15 cylinders. There are 255 heads, 63 sectors, 512 bytes per sector.

So far so good, now we want to create two partitions: one for the boot image, one for our distro. Create the FAT32 partition for booting and transferring files from Windows. Mark it as bootable.

```
================================================================================

Expert command (m for help): r
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 1

First cylinder (1-245, default 1): (press Enter) Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-245, default 245): +50

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32 (LBA))

Command (m for help): a
Partition number (1-4): 1

================================================================================
```

Create the Linux partition for the root file system.

```
================================================================================

Command (m for help): n
Command action e extended
p primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (52-245, default 52): (press Enter) Using default value 52
Last cylinder or +size or +sizeM or +sizeK (52-245, default 245):(press
Enter)
Using default value 245

================================================================================
```

Print and save the new partition records.

```
================================================================================

Command (m for help): p

Disk /dev/sdc: 2021 MB, 2021654528 bytes
255 heads, 63 sectors/track, 245 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

```
Device Boot      Start         End      Blocks   Id  System
/dev/sdc1   *        1          51      409626    c  W95 FAT32 (LBA)
/dev/sdc2           52         245     1558305   83  Linux

Command (m for help): w
The partition table has been altered! Calling ioctl() to re-read partition
table.

WARNING: Re-reading the partition table failed with error 16: Device or
resource busy. The kernel still uses the old table. The new table will be used
at the next reboot.

WARNING: If you have created or modified any DOS 6.x partitions, please see
the fdisk manual page for additional information.
Syncing disks.
```

==========

Now we've got both partitions, next step is formatting them.
*NOTE*: If the partitions (/dev/sdc1 and /dev/sdc2) do not exist, you should unplug the card and plug it back in. Linux will now be able to detect the new partitions.

==========

```
$ sudo mkfs.msdos -F 32 /dev/sdc1 -n LABEL
mkfs.msdos 2.11 (12 Mar 2005)

$ sudo mkfs.ext3 /dev/sdc2 mke2fs 1.40-WIP (14-Nov-2006) Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
195072 inodes, 389576 blocks
19478 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=402653184
12 block groups
32768 blocks per group, 32768 fragments per group
16256 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information:
```

==========
All done!

*NOTE*: For convenience, you can add the -L option to the *mkfs.ext3* command to assign a volume label to the new ext3 filesystem. If you do that, the new (automatic) mount point under /media when you insert that SD card into some Linux hosts will be based on that label.
If there's no label, the new mount point will most likely be a long hstring, so assigning a label makes manual mounting on the host more convenient.

## 2.4. Setting up the boot partition

The boot partition must contain:

- bootcode.bin : 2nd stage bootloader, starts with SDRAM disabled
- loader.bin : 3rd stage bootloader, starts with SDRAM enabled
- start.elf: The GPU binary firmware image, provided by the foundation.
- kernel.img: The OS kernel to load on the ARM processor. Normally this is Linux -see instructions for compiling a kernel.
- cmdline.txt: Parameters passed to the kernel on boot. Optional files:

- config.txt: A configuration file read by the GPU. Use this to override set the video mode, alter system clock speeds, voltages, etc.
- vlls directory: Additional GPU code, e.g. extra CODECs. Not present in the initial release.

## 2.5. Additional files supplied by RPi Foundation

These files are also present on the SD cards supplied by the Foundation.

Additional kernels. Rename over kernel.img to use them (ensure you have a backup of the original kernel.img first!):

- kernel_emergency.img : kernel with busybox rootfs. You can use this to repair the main Linux partition using e2fsck if the Linux partition gets corrupted. Additional GPU firmware

images, rename over start.elf to use them:

- arm128_start.elf : 128MB ARM, 128MB GPU split (use this for heavy 3D work, possibly also required for some video decoding)
- arm192_start.elf : 192MB ARM, 64MB GPU split (this is the default)
- arm224_start.elf : 224MB ARM, 32MB GPU split (use this for Linux only with no 3D or video processing. It's enough for the 1080p frame buffer, but not much else)

## 2.6. Writing the image to the SDcard and booting GNU/Linux

The easiest way to do this is to use PiCard. It even saves you from some hassles explained above. You will need your SD card + reader and a Linux PC to use PiCard. After that, just plug the card into your Rpi.

## 2.7. Wire up your Raspberry Pi and power it up

As explained in Section 1

## 2.8. SD Card Cloning/Backup

*Note: Update these instructions if required once they've been tried.*

From windows you can copy the full SD card by using Win32DiskImager. Alternatively, you can use the following instructions;

*Note: Many built-in SD card readers do not work, so if you have problems use an external SD-USB adapter for this.*

## 2.9. Required Software Setup

- Download a windows utility *dd.exe* from *http://www.chrysocome.net/dd*
- Rename it: *windd.exe*

*(This executable can to write to your hard disk so exercise caution using it!)*

- Make a copy named*: dd-removable.exe*

*(That executable refuses to write to your hard disk as it is named dd-removable As long as you use dd-removable.exe you cannot lose your hard disk)*

- Connect an SD card to the computer
- Run: *"dd-removable –list"*

Should give something like this:

```
rawwrite dd for windows version 0.6beta3. Written by John Newbigin
<jn@it.swin.edu.au>
This program is covered by terms of the GPL Version 2.

NT Block Device Objects
\\?\Device\Harddisk1\Partition0 link to \\?\Device\Harddisk1\DR8
Removable media other than floppy. Block size = 512 size is 4075290624 bytes
```

This "\\?\Device\Harddisk1\Partition0" is the part you need.

## 2.10. Reading an image from the SD Card

**BEWARE: DO THIS WRONG AND YOU MIGHT CORRUPT YOUR HARD DISK!**

Obviously, you can NOT use 'dd-removable' to read an image as that executable refuses to write to your hard disk (so extra care is required here as you use 'windd').

- To **read** an SD-card image from the SD-card use:

```
windd bs=1M if=\\?\Device\Harddisk1\Partition0 of=THE_IMAGE_READ -size
Your disk name ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

## 2.11. Copying an image to the SD Card

**BEWARE: DO THIS WRONG AND YOU MIGHT CORRUPT YOUR HARD DISK!**

- To **copy** an image named "THEIMAGE" to the SD-card do this:

```
dd-removable bs=1M if=THEIMAGE of=\\?\Device\Harddisk1\Partition0
Your disk name ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

## 2.12. Software Development/Proving

A supported platform for the Raspberry is Qt , which is already being worked on. C/C++ is supported through a gcc cross-compiling tool chain.

After compiling, using QEMU and a Linux VM would be one way of testing your apps. This also works on Windows. Search the forum for the readymade ARM images. The choice of programming languages, IDEs and other tools on the RPi is only determined by:

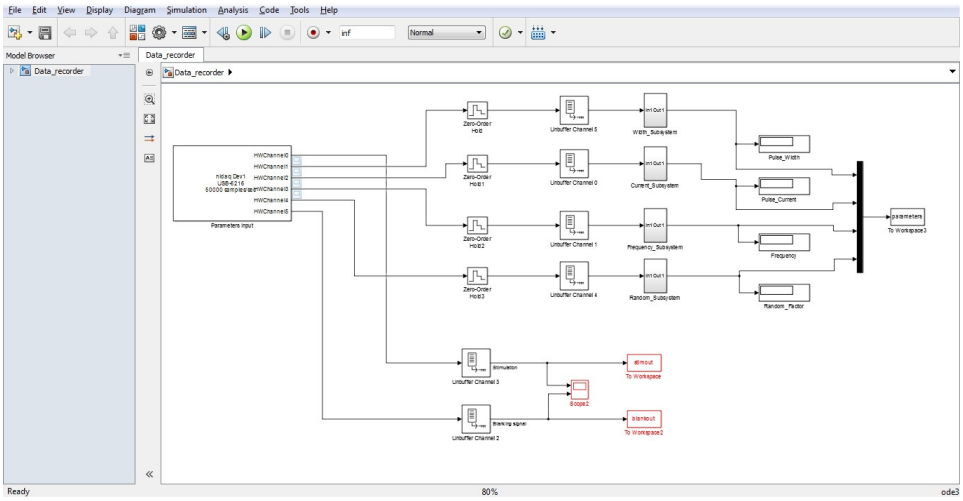
- The operating system compatibility (at the moment the specific Linux distro used)
- The status of the respective ARM package repositories and their binary compatibility
- The possibility to build other software + its dependencies for the RPi from sources.

**For more guides and projects involving the Raspberry Pi, see RPi Projects**

**http://elinux.org/RPi_Projects**

# **Annex III:** Simulink Model: data recorder

During the project, sometimes is useful can record the generated signal to analyze them and compare different results. The next diagram was programing on Simulink with this purpose, actually, some of the signal showed at the report comes from this diagram.




To acquire the signal I had a National Instruments USB DAQ board. This board can sample at 400 KSanmples per second in total, which was so enough for me.

## NI USB-621*x* Specifications

Specifications listed below are typical at 25 °C unless otherwise noted. Refer to the *NI USB-621x User Manual* for more information about USB-621x devices.

**Caution** The input/output ports of this device are not protected for electromagnetic interference due to functional reasons. As a result, this device may experience reduced measurement accuracy or other temporary performance degradation when connected cables are routed in an environment with radiated or conducted radio frequency electromagnetic interference.

To ensure that this device functions within specifications in its operational electromagnetic environment and to limit radiated emissions, care should be taken in the selection, design, and installation of measurement probes and cables.

Français Deutsch 日本語 한국어 简体中文
ni.com/manuals

### Analog Input

Number of channels
USB-6210/6211/6212/6215/6216 ..... 8 differential or 16 single ended
USB-6218 ..... 16 differential or 32 single ended

ADC resolution ..... 16 bits

DNL ..... No missing codes guaranteed

INL ..... Refer to the *AI Absolute Accuracy Tables*

Sampling rate
Maximum
USB-6210/6211/6215/6218 ... 250 kS/s single channel, 250 kS/s multichannel (aggregate)
USB-6212/6216 ..... 400 kS/s single channel, 400 kS/s multichannel (aggregate)
Minimum ..... 0 S/s
Timing accuracy ..... 50 ppm of sample rate
Timing resolution ..... 50 ns

Input coupling ..... DC

Input range ..... ±10 V, ±5 V, ±1 V, ±0.2 V

Maximum working voltage for analog inputs (signal + common mode) ..... ±10.4 V of AI GND

CMRR (DC to 60 Hz) ..... 100 dB

Input impedance
Device on
AI+ to AI GND ..... >10 GΩ in parallel with 100 pF
AI– to AI GND ..... >10 GΩ in parallel with 100 pF
Device off
AI+ to AI GND ..... 1200 Ω
AI– to AI GND ..... 1200 Ω

Input bias current ..... ±100 pA

Crosstalk (at 100 kHz)
Adjacent channels ..... –75 dB
Non-adjacent channels ..... –90 dB

Small signal bandwidth (–3 dB)
USB-6210/6211/6215/6218 ..... 450 kHz
USB-6212/6216 ..... 1.5 MHz

NATIONAL INSTRUMENTS

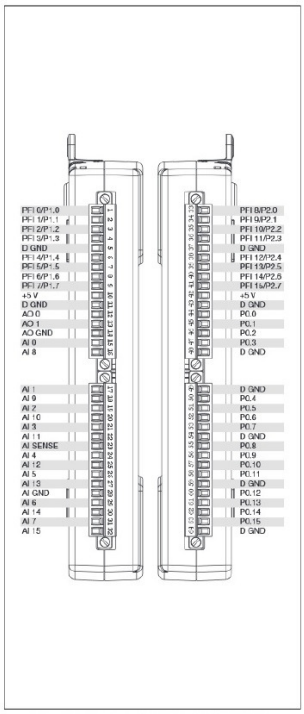


**Figure 4.** USB-6212/6216 Screw Terminal Pinout

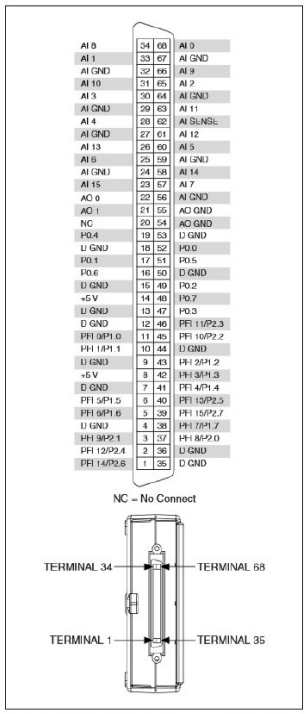



**Figure 5.** USB-6212/6216 Mass Termination Pinout