ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA DE LA SALUD

MENCIÓN INGENIERÍA BIOMÉDICA

Diseño y montaje de un simulador de ECG

Design and development of ECG simulator

Realizado por

Sandro Hurtado Requena

Tutorizado por

Antonio Jesús Bandera Rubio

Departamento

Tecnología Electrónica

UNIVERSIDAD DE MÁLAGA

MÁLAGA, PONER FECHA 2017

Fecha de defensa:
El secretario del Tribunal

Resumen

Actualmente, uno de los retos de la ingeniería biomédica es ofrecer recursos para garantizar la calidad de vida de la población global, especialmente en los países que se encuentran en vías de exclusión y pobreza. Esta ingeniería recoge aspectos de la mayoría de las ciencias con el objetivo de ayudar a la mejora de la persona. En este entorno tan amplio, se encuadra este trabajo, que tiene como propósito el diseño y montaje de un simulador de electrocardiograma (ECG), que genere las principales formas de onda registradas en la actividad eléctrica del corazón. Se trata de un trabajo multidisciplinar, en el que abarcaremos diferentes ciencias, desde los fundamentos del corazón y su influencia directa en el ECG, como el desarrollo y montaje de circuitos electrónicos, y programación en distintos lenguajes. Este simulador será diseñado para fines académicos por lo general, con el que se pretende mejorar el aprendizaje de los estudiantes de medicina, pudiendo ser accesible a la mayoría de las personas, debido a su bajo coste en oposición al elevado precio de estos dispositivos en el mercado. Para el desarrollo de éste, se utiliza materiales y programas de código abierto y de fácil adquisición. Cabe destacar, que como base principal del simulador, se emplea el microcontrolador ATMega328P, integrado en la placa Adafruit Menta, el cual se encarga de gestionar el funcionamiento de todo el circuito.

Palabras clave: Simulador de Electrocardiograma, características multidisciplinares, circuitos electrónicos, programación en distintos lenguajes, bajo coste, código abierto, microcontrolador ATMega328P, Adafruit Menta.

Abstract

Nowadays, one of the biggest challenges in biomedical engineering (Bme) is to provide resources to enhance the healthcare of global population, especially in developing countries on the danger line. Bme gathers aspects of mosts sciences in order to improve life quality of people. In this wide enviroment, this work is framed, which purpose is the design and development of an electrocardiogram (ECG) simulator, which generates the main waveforms recorded in the electrical activity of a heart. This is a multidisciplinary work, in which we will study different sciences, from the basics of the heart and its direct influence on the ECG to the development and assembly of electronic circuits, going throught different languages of programming. This simulator has been genericly designed for academic purposes aiming to improve learning among medical students, as well as being accessible to most people because it's cheaper than any similar device. The development of this project, has been made with open source materials and software easy to acquire. Is highlight to say, the main base of this ECG simulator is a ATMega328P microcontroller integrated in a Adafruit menta plate, which is responsible for controlling the operation of the entire circuit.

Keywords: Electrocardiogram Simulator, multidisciplinary, electronics circuits, differents languages of programming, cheaper, open source, ATMega328P microcontroller, Adafruit Menta.

Índice general

1. Introducción **1**
- 1.1. Introducción al trabajo . . . 1
- 1.2. Motivación . . . 2
- 1.3. Objetivos . . . 3
- 1.4. Estructura de la memoria . . . 4
- 1.5. Tecnologías a utilizar . . . 4

2. Fundamentos anatómicos y fisiológicos **5**
- 2.1. Introducción al Electrocardiograma (ECG) . . . 5
  - 2.1.1. Anatomía del corazón . . . 5
  - 2.1.2. Potencial de reposo . . . 7
  - 2.1.3. Potenciales de acción cardíacos . . . 7
- 2.2. Electrocardiograma (ECG) . . . 8
  - 2.2.1. Representación del ECG . . . 9
  - 2.2.2. El corazón como fuente del ECG . . . 12

3. Desarrollo del hardware **13**
- 3.1. Introducción . . . 13
- 3.2. Interfaz Spi . . . 13
  - 3.2.1. Modos de operación de reloj . . . 15
- 3.3. Lista de materiales . . . 16
- 3.4. Diseño en EAGLE . . . 21
- 3.5. Ensamblaje y montaje . . . 22

4. Desarrollo del Software **25**
- 4.1. Introducción . . . 25
- 4.2. Digitalización de la imagen . . . 27
- 4.3. Phyton . . . 28
  - 4.3.1. Introducción . . . 28
  - 4.3.2. Desarrollo del programa Phyton . . . 29
- 4.4. Código de Arduino . . . 31
  - 4.4.1. Especificaciones del programa y funcionamiento de nuestro circuito 31
  - 4.4.2. Estructura modular del código. Organización por tareas . . . 31

5. Pruebas y resultados **37**

6. Conclusiones **41**

7. Anexos **45**
7.1. Código Arduino 45

Capítulo 1

Introducción

1.1. Introducción al trabajo

Las enfermedades del sistema circulatorio constituyen un problema de salud de primer orden en todo el mundo, ya que a su importancia capital en los países desarrollados se une su creciente relevancia en los países en vías de desarrollo. Además, muchas de las principales amenazas mundiales para la salud, como la hipertensión arterial, el tabaquismo, el consumo de alcohol, la hipocolesterolemia y la obesidad o el sobrepeso, están directa o indirectamente relacionadas con las enfermedades cardiovasculares (ECV) (Álvarez et al., 2003). Datos recogidos por la Organización Mundial de la salud (OMS) nos afirman que las ECV son la principal causa de muerte en todo el mundo, ya que en 2012 murieron por esta causa 17,5 millones de personas, lo cual representa un 31 % de todas las muertes registradas en el mundo. (OMS, 2015) Además, se tiene previsto que éstas aumentarán de 17.5 millones a 23.4 millones de muertes para el año 2030, por lo que las ECV junto con el cáncer representarán un 56 % del total de 67 millones de defunciones previstas para ese año (Ferrante et al., 2008).

A nivel nacional, las enfermedades del sistema cardiovascular representan una de las primeras causas de muerte. En el año 2014 la tasa bruta de mortalidad aumentó un 1.7 % y se situó en 852.1 fallecidos por cada 100.000 habitantes, de los cuales, 252.7 fallecimientos se produjeron por causa de enfermedades cardiovasculares, colocándose como primera causa de muerte en España, seguida de tumores y enfermedades respiratorias (de Cardiología (2009). Las enfermedades del sistema circulatorio fueron la primera causa de mortalidad femenina (270.2 muertes por cada 100.000) y la segunda entre los varones (234.6), como se muestra en la figura 1.1.

A nivel detallado, las principales enfermedades del sistema circulatorio son la enfermedad isquémica del corazón, que es la que ocasiona un mayor número de muertes, y la enfermedad cerebrovascular o ictus, que representa el 28 % de toda la mortalidad cardiovascular. En conjunto producen casi el 60 % de toda la mortalidad cardiovascular(Álvarez et al., 2003).

Las tasas de mortalidad cardiovascular se diferencian según el sexo y la edad de la persona. A medida que se incrementa la edad, la tasa de mortalidad aumenta exponencialmente, siendo sólo la primera causa de muerte a partir de los 75 años de edad. Por otro lado, en España mueren más mujeres que varones por enfermedades del sistema circulatorio.

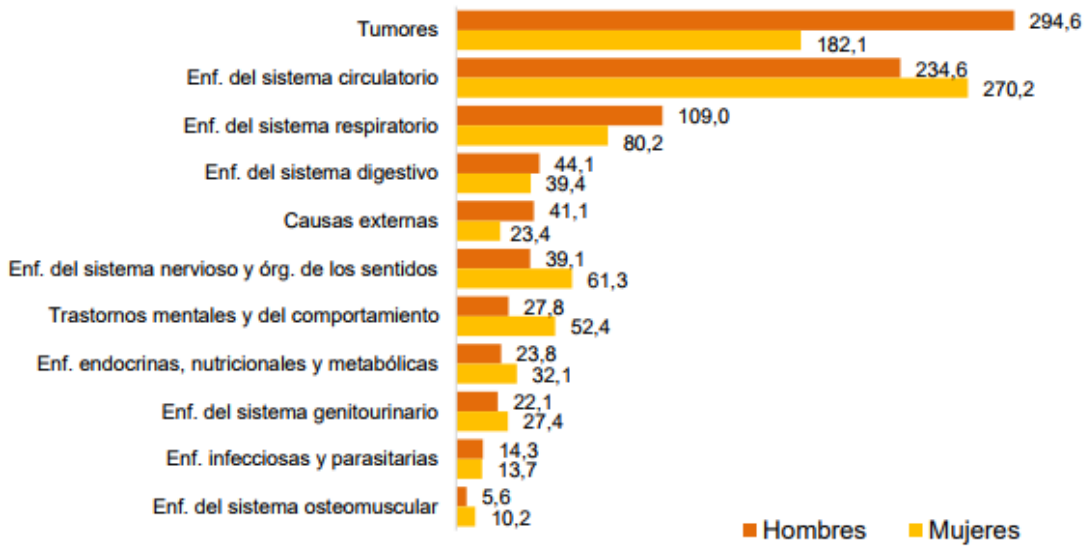


Figura 1.1: Tasas brutas de mortalidad por 100.000 habitantes según causa de muerte (de Cardiología (2009))

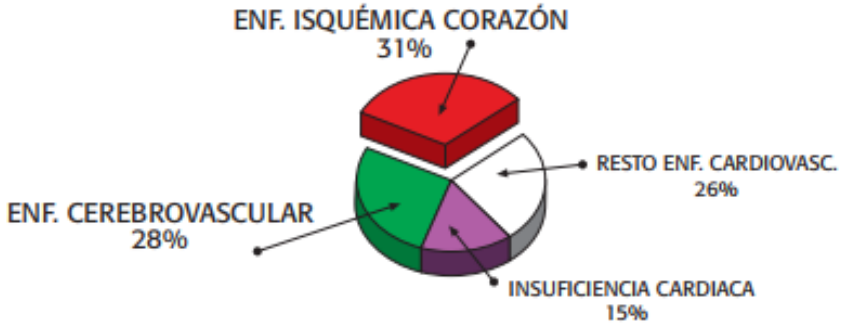


Figura 1.2: Mortalidad proporcional por las distintas enfermedades del sistema circulatorio en ambos sexos (Álvarez et al., 2003).

1.2. Motivación

Por los motivos anteriormente expuestos, resulta totalmente necesaria la adquisición de los conocimientos y competencias, para poder diagnosticar a tiempo las patologías cardíacas, para el futuro de la medicina profesional. No obstante esta labor resulta bastante complicada para los estudiantes de medicina que consideran el tema de gran complejidad para su estudio y compresión (Viñoles et al., 2015). De aquí surge la principal necesidad y motivación que fundamenta este proyecto, que tiene como propósito diseñar un simulador de electrocardiograma que genere las principales formas de onda registradas en la actividad eléctrica del corazón, permitiendo variar la frecuencia de visualización de las ondas. Estos dispositivos resultan muy útiles tanto en la vida laboral, para calibrar monitores de pacientes, como en lo académico, para mejorar el aprendizaje de los estudiantes en el ámbito de la medicina, ayudando al estudio y la compresión del Electrocardiograma (ECG) mediante la observación de la onda cardíaca y sus posibles variaciones.

Cabe destacar que existe una amplia variedad de equipos médicos que registran y miden la actividad eléctrica del corazón a partir del ECG, que son utilizados en diferentes áreas de urgencias, cardiología, quirófanos, salas de recuperación de pacientes, etc. Al igual que todos los equipos médicos, éstos necesitarán de mantenimientos, chequeos y calibraciones rutinarias que serán llevados a cabo por especialistas utilizando material de precisión, dada la importancia del correcto registro y medición de la actividad cardíaca. El principal problema que nos seguimos encontrando hoy día es que, a pesar de la amplia gama que existe en el mercado, los equipos de instrumentación para la simulación de señales de ECG tienen unos precios muy elevados, dando lugar a que la mayoría de los servicios de mantenimiento biomédico carezcan de ellos, y por lo tanto provocando que los respectivos mantenimientos de los equipos y sus calibraciones se realicen con instrumental inadecuado. Estas razones justifican, la propuesta del diseño y montaje de nuestro propio simulador de ECG, cuyas principales características son su gran versatilidad y bajo costo, que ronda entre los $60 - 100$ euros. Esto permitirá que sea accesible económicamente a todos los usuarios, frente a otros simuladores que se pueden comprar por cientos de euros (Banegas et al., 2006).

1.3. Objetivos

La idea de este proyecto será construir un simulador de ECG basado en el microcontrolador Arduino ATMega328P. La salida de la información se mostrará usando una pantalla numérica de 4 dígitos, que presentará la frecuencia cardíaca generada. También se deberán incluir en la interfaz del simulador un mando (potenciómetro) que permita ajustar la frecuencia y conectores tipo banana para poder enganchar las tres derivaciones del ECG.

La forma de onda se creará desde la digitalización de formas de onda representadas gráficamente. Dicha fuente, codificada en forma de ficheros de texto, deberá ser interpolada para generar la secuencia de muestras que genera el dispositivo (0,001 Hz). La emisión final de esta secuencia será gestionada por el microcontrolador. Como desafortunadamente el ATMega328P no dispone de convertidor digital-analógico, será necesario añadir un convertidor externo para generar la forma de onda del ECG analógica. Un divisor de tensión resistivo atenuará la señal analógica a los niveles requeridos de milivoltios, que es la amplitud típica de una señal de ECG humana capturada colocando los electrodos en el tórax.

1.4. Estructura de la memoria

La estructura de la memoria se ha elaborado en orden cronológico de las fases del diseño:

- **Capítulo 2, Fundamentos anatómicos y fisiológicos,** se realiza una primera búsqueda de información sobre los fundamentos anatómicos y fisiológicos del ECG, donde se adquieren las bases necesarias para el entendimiento de éste.
- **Capítulo 3, Desarrollo del hardware,** expone los distintos componentes que conforman el diseño del simulador de ECG, explicando la aportación de cada uno de ellos al funcionamiento del circuito.
- **Capítulo 4, Desarrollo del software,** presenta los programas utilizados para la creación de la forma de onda del ECG, detallando el funcionamiento interno de cada programa.
- **Capítulo 5, Pruebas y resultados,** se detallarán las pruebas experimentales que se han realizado para validar el correcto funcionamiento del sistema de simulación de ondas de ECG.
- **Capítulo 6, Conclusiones,** se tratarán los aspectos mas relevantes del trabajo. Asimismo, argumentaremos un apartado de conclusiones personales, donde se reflejan las competencias adquiridas durante este proceso.

1.5. Tecnologías a utilizar

Las principales tecnologías utilizadas para el diseño de nuestro simulador de ECG se expondrán a continuación, realizando una breve introducción a cada una de ellas:

- **Eagle**, se trata de un programa de diseño de diagramas electrónicos. Nos hemos servido de él para desarrollar un esquemático con las conexiones necesarias para el montaje del circuito.
- **Engauge Digitizer**, es un programa comúnmente utilizado para extraer puntos de datos de imágenes de gráficos. Este programa es ideal para la digitalización de la imagen de este trabajo.
- **Phyton**, es un lenguaje de programación, el cual tiene sintaxis de fácil interpretación. Con la ayuda de este, se creó un código cuyo objetivo es realizar una interpolación lineal de los puntos obtenidos con el programa Encauge Digitizer, y la creación de una C-matriz para su posterior utilización en Arduino.
- **Arduino**, es una plataforma de desarrollo computacional que consta de hardware y software. Lleva a cabo una filosofía de código abierto, que fomenta una comunidad para compartir sus conocimientos (Banzi and Shiloh (2014)). En el presente trabajo, se escogió una plataforma hardware compatible con el entorno de desarrollo de Arduino, llamada Adafruit Menta. Ésta se basa en el ya citado microcontrolador ATMega328P, el cual será el encargado de organizar el funcionamiento del simulador de ECG, mediante su previa programación.

Capítulo 2

Fundamentos anatómicos y fisiológicos

2.1. Introducción al Electrocardiograma (ECG)

Antes de explicar lo que es un ECG, debemos introducir algunos conceptos previos para el entendimiento del funcionamiento de éste. Este capítulo describe, en primer lugar del corazón y de los elementos que componen su estructura. Posteriormente, introduce el concepto de potencial de acción, el cual será muy importante para la explicación del ECG.

2.1.1. Anatomía del corazón

El corazón humano es un órgano muscular, que compone uno de los elementos del sistema circulatorio, junto con los vasos sanguíneos y la sangre. Éste consta de cuatro cámaras o cavidades, dos cámaras superiores (aurículas) y dos inferiores (ventrículos), separadas por unos tabiques denominados tabiques interauricular e interventricular. Entre las aurículas y los ventrículos se forma un surco auriculoventricular, por el que discurren las ramas de las arterias coronarias derecha e izquierda que irrigan el corazón (Flórez, 2014); (Mulroney and Myers, 2016).

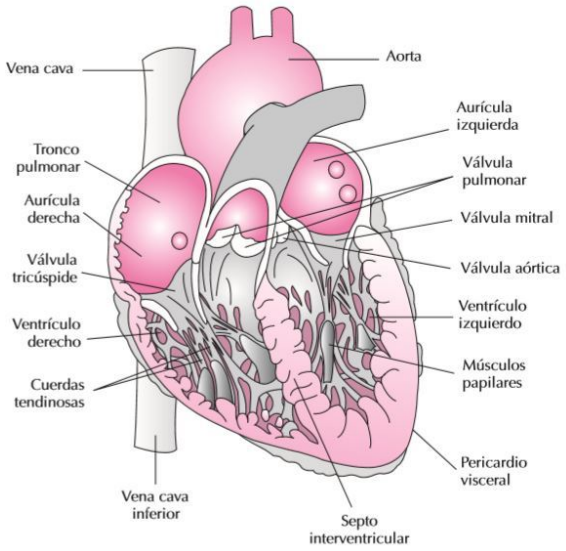


Figura 2.1: Anatomía del corazón. Localización de las aurículas y los ventrículos (Flórez, 2014).

Las aurículas son cavidades que presentan una pared delgada y presiones bajas. Su función es almacenar la sangre que procede del territorio venoso sistémico (aurícula derecha) y pulmonar (aurícula izquierda) durante la sístole ventricular. La sangre llega a la aurícula derecha a través de 3 venas: la cava superior, la cava inferior y el seno coronario. Por otro lado, la aurícula izquierda recibe la sangre procedente de los pulmones a través de las venas pulmonares (Flórez, 2014).

Los ventrículos proveen la fuerza necesaria para bombear la sangre a través de la arteria pulmonar (ventrículo derecho) y de la aorta (ventrículo izquierdo), razón por la que sus paredes son mucho más gruesas que las de las aurículas. Dado que la sangre sólo fluye desde las zonas de presión alta a las de presión baja, para poder expulsar la sangre es necesario que la presión generada en los ventrículos supere la presión existente en las arterias (Flórez, 2014).

La circulación de la sangre va desde las aurículas a los ventrículos y para que no se produzca una conducción de sangre de forma retrograda hacia las aurículas existen las denominadas válvulas cardíacas. Es necesario que existan dos pares de válvulas que ocupen los orificios de entrada (válvulas auriculoventriculares) y de salida (válvulas semilunares) de los ventrículos (Flórez, 2014). Las válvulas auriculoventriculares (AV) permiten que la sangre fluya de las aurículas a los ventrículos. La válvula AV derecha presenta tres valvas (anterior, media o septal y posterior), por lo que se denomina tricúspide, mientras que la situada entre la aurícula izquierda y el ventrículo izquierdo, que presenta dos valvas (anterior o aórtica y posterior), se denomina bicúspide o mitral, que es la encargada de impedir el flujo retrógrado a las aurículas en el corazón izquierdo (Flórez, 2014).

Las válvulas semilunares, pulmonares o derechas y aórticas o izquierda, presentan 3 valvas, una derecha, una izquierda y una posterior en el caso de la válvula aórtica, y una anterior, una derecha y una izquierda en el caso de la válvula pulmonar. Cuando se produce la eyección de sangre, las válvulas aórticas y pulmonares se encuentran abiertas, cuando se cierran están impiden el flujo de vuelta hacia los ventrículos durante el llenado cardiaco. Tanto la abertura, y el cierre de ambas son consecuencia de mecanismos pasivos por gradientes de presión (Mulroney and Myers, 2016).

Podemos explicar en resumidas palabras lo expuesto hasta ahora, ya que el funcionamiento básico de la estructura del corazón consiste en que, los ventrículos son los encargados de bombear la sangre, puesto que el ventrículo izquierdo conduce la sangre hacia la circulación sistémica de alta presión y el ventrículo derecho se encarga de bombear la sangre hacia la circulación pulmonar. Las válvulas se encargarán de evitar el flujo retrogrado de la sangre, así las válvulas mitral y tricúspide impiden el retorno sanguíneo hacia las aurículas, mientras que las válvulas aórticas y pulmonares impedirán dicho retorno hacia los ventrículos.

2.1.2. Potencial de reposo

A ambos lados de la membrana lipoproteica que separa los medios intra y extracelular existe una diferencia de potencial, a la que denominamos potencial de membrana (Em). Cuando introducimos un micro electrodo se puede medir el valor de este Em, que oscila entre –80 y –90 mV en las células musculares auriculares y ventriculares. Cuando el potencial de membrana de estas células no es excitado reciben el nombre de potencial de reposo. El potencial de reposo está determinado por el equilibrio entre la capacidad de distintos iones para atravesar la membrana a favor de su gradiente electroquímico (permeabilidad de la membrana para cada ion) y los sistemas de transporte que movilizan estos iones en contra de su gradiente de concentración. El sistema de denomina bomba sodio/potasio (Na^+/K^+) (Flórez, 2014).

2.1.3. Potenciales de acción cardíacos

Este concepto lo podemos introducir de la siguiente manera, si aplicamos una cierta intensidad de corriente a una célula cardíaca en reposo, ésta al incrementar la intensidad de dicha corriente aumentará la amplitud de la respuesta generada, y si ésta alcanza un determinado nivel, denominado potencial umbral, se produce una respuesta denominada potencial de acción (Flórez, 2014).

El nodo sinoauricular (SA) se comporta como un marcapasos que produce una frecuencia cardíaca de 70 pulsaciones por minuto en un corazón normal. El potencial de acción que se inicia en el nodo SA se esparce excitando las fibras musculares auriculares y ventriculares, denominadas fibras contráctiles, como se muestra en la figura 2.2.

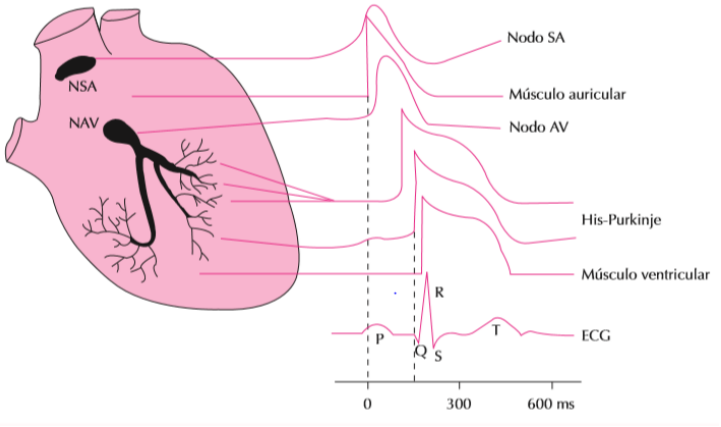


Figura 2.2: Representación esquemática de los potenciales de acción registrados en diversas zonas cardíacas y su correlación con el electrocardiograma. NSA: nodo sinoauricular. NAV: nodo auriculoventricular. (Flórez, 2014).

A continuación explicaremos como se produce un potencial de acción en una fibra contráctil en sus distintas fases, como se muestra en la figura 2.3

Fase 0 (despolarización) Como ya sabemos el potencial de membrana en reposo de las fibras contráctiles suelen ser de -90 mV, pues en esta fase se produce una despolarización rápida que tiene lugar cuando las células alcanzan el umbral y seguidamente se abren los canales rápidos de Na^+ (Sodio). Esta abertura se acompaña del descenso de la

conductancia de la corriente de entrada de K^+ (potasio). Esta pendiente rápida da lugar a la rápida dispersión de la despolarización hacia la mayor parte del sistema de conducción del corazón (Mulroney and Myers, 2016).

Fase 1 (Repolarización rápida hacia la meseta) En este momento de desactivan los canales de Na^+ y se abren los canales de K^+ sensibles al voltaje, produciéndose una corriente de entrada transitoria de K^+.

Fase 2 (Meseta) Durante esta fase la membrana se mantiene despolarizada debido a la abertura de los canales lentos de Ca^+ (Calcio), de tipo L sensibles al voltaje, y a la corriente de entrada de Ca^+. Por otro lado, se da una corriente de salida de K^+ a través del canal de K^+, dependiente del voltaje.

Fase 3 (Repolarización) Esta fase es producida por el cierre de los canales de Ca^+ y la salida de K^+, que provocan la repolarización rápida debido principalmente a una corriente de entrada rectificada de K^+.

Fase 4 (Potencial de membrana en reposo) Las concentraciones de iones que se han alterado anteriormente con el potencial de acción se restauran hasta los niveles de reposo por efecto de la Na^+/K^+- ATPasa (bomba sodio-potasio) y de un intercambiador Na^+/Ca^{2+}, así como de una bomba de Ca^{2+} dependiente de ATP (trifosfato de adenosina) (Mulroney and Myers, 2016).

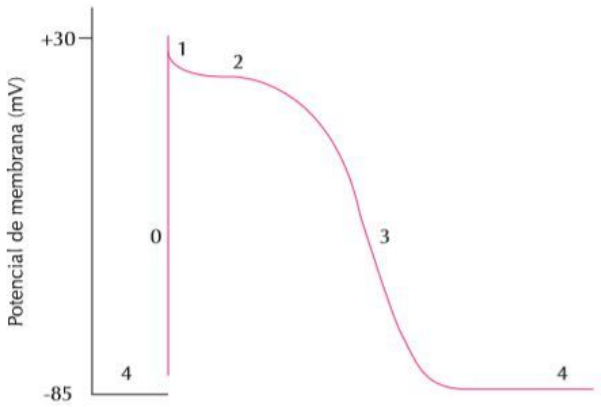


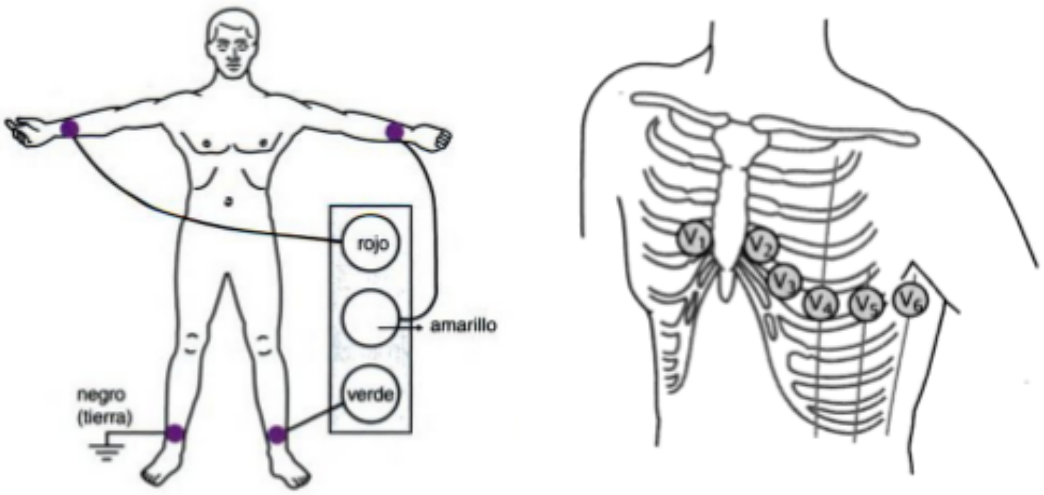
Figura 2.3: Potencial de Acción de una fibra contráctil (Flórez, 2014).

2.2. Electrocardiograma (ECG)

El electrocardiograma (ECG) es el registro de la actividad eléctrica del corazón, generada cuando éste es excitado por pequeños estímulos eléctricos periódicos, que provocan una diferencia de potencial entre varios puntos de la superficie del organismo por los fenómenos eléctricos que acompañan al latido cardíaco. El ECG lo podemos recoger mediante el uso de un electrocardiógrafo colocado sobre nuestro pecho (Tucci, 2007);(Flórez, 2014).

En la práctica, para realizar un ECG normal se colocan electrodos en los brazos y las piernas del paciente (Figura 2.4(a)). Por otro lado, podemos escoger seis ubicaciones a nivel

torácico (Figura 2.4(b)). El electrocardiógrafo amplifica estas señales para que puedan ser visualizadas y la representa en 12 trazados que representan las distintas combinaciones de las diferentes derivaciones de los miembros y precordiales. Cada uno de los electrodos recoge una actividad eléctrica del corazón según la posición que ocupa.



(a) Derivaciones de las extremidades. Disposición en "semáforo"

(b) Derivaciones precordiales

Figura 2.4: Colocación de los electrodos del ECG (Hamm and Willems, 2010)

.

2.2.1. Representación del ECG

En el registro del ECG podemos ver que aparecen una serie de deflexiones u ondas que vienen representadas en un plano de dos dimensiones, en el que el eje de ordenadas representa la magnitud del potencial en voltaje y en el eje de abscisas se representa el tiempo transcurrido. Tenemos que tener en cuenta que en el eje de ordenadas el voltaje se representará en el orden de mV, ya que los datos recogidos son de muy pequeña intensidad. El registro del ECG se recoge en un papel cuadriculado por líneas finas y gruesas para poder saber el voltaje y el tiempo de cada una de las ondas, como se muestra en la figura 2.5. Las líneas finas están separadas entre sí 1mm, mientras que las gruesas están separadas 5mm. Esto es equivalente a decir que las líneas finas de 1mm de separación en el eje de abscisas (tiempo), es equivalente a 0.04 segundos, mientras que las gruesas que están separadas 5 mm, son equivalentes a 0,2 segundos. Del mismo modo en el eje de coordenadas (voltaje), la separación de las líneas finas (1 mm), es equivalente a 1 mV y las gruesas (5 mm) a 0,5 mV (Flórez, 2014).
Una vez indicada las dimensiones de la cuadricula donde irá recogido nuestro ECG, podremos dar paso a la explicación de las distintas ondas que lo componen:

Onda P La primera onda que se produce es la onda P. Su origen es debido a la despolarización auricular, se trata de una pequeña deflexión positiva y despolarización auricular se propaga desde el nodo SA a través de las fibras contráctiles en ambas aurículas (Tortora and Derrickson, 2007).

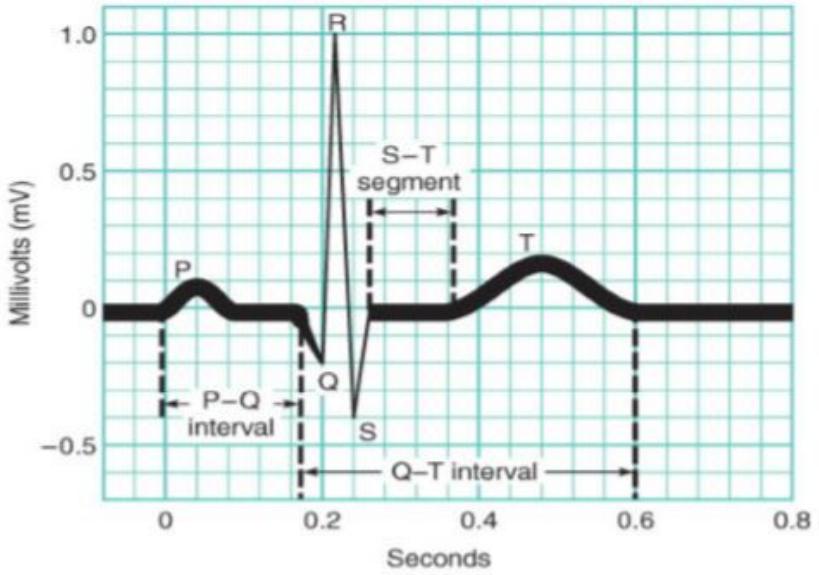


Figura 2.5: Representación de un ECG en papel cuadriculado (Tortora and Derrickson, 2007)

Ésta, para que se trate de una onda sinusal, tiene que tener unos valores normales que son por lo general, una amplitud de onda menor a 0,25 mV y una duración menor o igual a los 100 milisegundos, además de tener una forma redondeada y preceder al complejo QRS (Hamm and Willems, 2010).

Intervalo PQ Cada onda P esta seguida, de un intervalo PQ, que es el intervalo de tiempo desde el comienzo de la onda P hasta el complejo QRS. Representa el tiempo de conducción desde la excitación auricular hasta el inicio de la despolarización ventricular (Tortora and Derrickson, 2007).

El intervalo PQ tiene la característica de ser dependiente de la frecuencia, es decir, cuanto mayor sea la frecuencia cardíaca (taquicardia), más corto será este segmento PQ, de lo contrario a menor frecuencia cardíaca (bradicardia), más largo será este intervalo. Este intervalo, como ya hemos comentado, no suele tener un periodo de duración fijo, sino que depende de la frecuencia cardíaca, y suele variar en un intervalo de 120 ms a 200 ms (Hamm and Willems, 2010).

Complejo QRS La segunda onda originada, se denomina complejo QRS. Representa la despolarización ventricular rápida, a medida que el potencial de acción progresa a través de las fibras ventriculares contráctiles. Ésta comienza con una deflexión negativa (onda Q) y continúa con una importante onda triangular positiva (R), terminando con una onda negativa (S) (Tortora and Derrickson, 2007).

Los valores normales de éste complejo suele ser de una duración menor a 100 ms, donde:

- La onda Q suele durar un periodo menor de 30 ms y tiene un amplitud aproximada de 0.3mV
- La onda R suele tener un voltaje no mayor a los 3,5 mV.
- La onda S tiene un periodo menor a 60ms.

Segmento S-T Comienza al final de la onda S y termina en el inicio de la onda T, expuesto sobre la línea isoeléctrica donde no hay oscilaciones de potencial. Representa el tiempo en el que las fibras ventriculares contráctiles están despolarizadas en la fase de la meseta del potencial de acción, explicado anteriormente (Tortora and Derrickson, 2007).

Intervalo Q-T Éste se extiende desde el comienzo del complejo QRS hasta el final de la onda T. Representa el tiempo que transcurre desde el comienzo de la despolarización ventricular hasta la repolarización del dicho ventrículo (Tortora and Derrickson, 2007).

Onda T La tercera onda que se produce es la onda T, correspondiente a la repolarización ventricular y aparece justo cuando los ventrículos están comenzando a relajarse. Ésta se representa como una deflexión positiva abovedada, la cual es más pequeña y más ancha que el complejo QRS, ya que la repolarización se produce más lentamente que la despolarización (Tortora and Derrickson, 2007).

Onda U No la tendremos en cuenta ya que no se conoce aún a ciencia cierta el porqué se produce fisiológicamente. Además no es demasiado frecuente que se represente en un ECG, ya que es difícilmente detectable.

En la figura 2.6 se puede apreciar claramente la representación de un ECG normal con sus distintas ondas e intervalos que la componen.

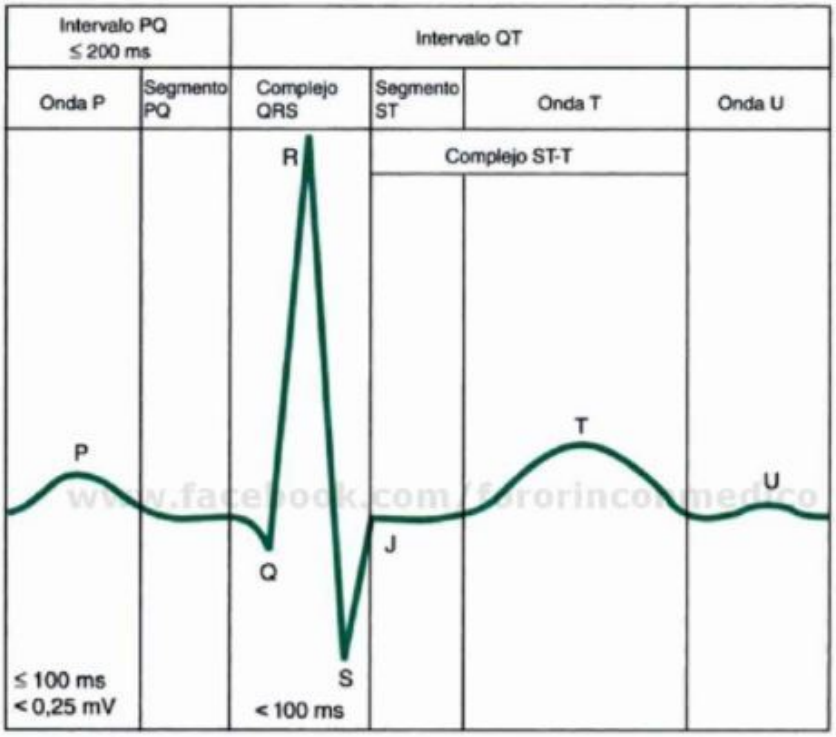


Figura 2.6: Definición de las diferentes porciones del ECG y sus valores normales (Hamm and Willems, 2010).

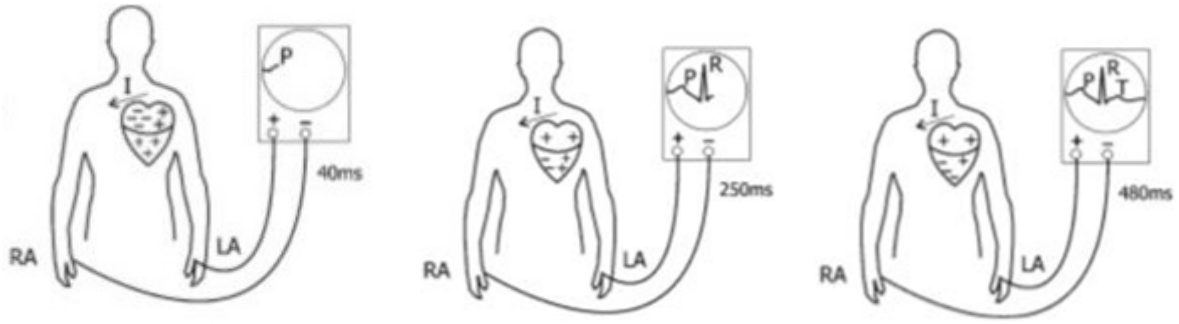
2.2.2. El corazón como fuente del ECG

Como ya sabemos, la diferencia de potencial entre los electrodos da origen al electrocardiograma, que es una representación gráfica de sus variaciones recogidas por los electrodos durante el ciclo cardíaco. A continuación se pretende explicar como el potencial eléctrico creado por un dipolo hace que circule una corriente del polo positivo (brazo izquierdo LA) al polo negativo (brazo derecho RA).

Se inicia la despolarización auricular creando la onda P. Después de unos 90 ms la aurícula está completamente despolarizada y la tensión cae a cero (Figura 2.7(a))

La despolarización de las aurículas se transmite al ventrículo derecho a través del nodo AV, por este motivo parte del músculo del ventrículo también comienza a despolarizarse. Una porción izquierda se vuelve negativa respecto a la todavía polarizada porción derecha. Se crea así otro dipolo que hace que el brazo izquierdo sea positivo respecto al derecho. De este modo se produce la onda R, que alcanza su valor máximo unos 250 ms después de iniciado el ciclo (Figura 2.7(b)) (Tucci, 2007).

La onda T se produce en forma similar al repolarizarse el músculo cardíaco (Figura 2.7(c)). Es positiva debido a que la repolarización se inicia en la superficie interna de las cavidades del corazón y se desplaza hacia la superficie externa (Tucci, 2007).



(a) Despolarización auricular (b) Despolarización ventricular (c) Repolarización del músculo cardíaco

Figura 2.7: Registro de la actividad eléctrica del corazón (Tucci, 2007).

Capítulo 3

Desarrollo del hardware

3.1. Introducción

En este capítulo explicaremos detalladamente cómo hemos realizado el diseño de nuestro simulador de ECG. Empezaremos mostrando una lista general con los materiales que hemos seleccionado para su creación y posteriormente iremos explicando el uso de cada uno de ellos con más detenimiento. El elemento clave del diseño será el ya citado microcontrolador ATMega328P, montado sobre la placa de Adafruit MENTA.

Sin embargo, en primer lugar será necesario explicar la interfaz de comunicación con la que se intercambiarán información los distintos componentes de nuestro circuito. Esta explicación será un punto clave para la compresión de nuestro diseño electrónico, ya que veremos en la forma en la que los elementos de nuestro circuito envían y reciben información al microcontrolador ATMEGA328P de nuestra placa Adafruit Menta, mediante el estándar de comunicación Spi.

3.2. Interfaz Spi

Spi (Serial Perifericial Interface) se define como un estándar de comunicaciones,empleado como un bus de interfaz de periféricos, a través del cual se pueden enviar paquetes de 1 Byte (8bits), a los dispositivos conectados usando esta interfaz. Los dispositivos conectados a esta interfaz tienen la posibilidad de enviar y recibir datos al mismo tiempo regulados por una señal de reloj, por lo que se trata de una comunicación full dúplex, ya que dos de las líneas del bus son utilizadas para transferir los datos. Excepcionalmente nos encontramos con algunos dispositivos que solo podrán transmitir datos o recibirlos. Una tercera línea será la encargada de regular la señal de reloj que estamos utilizando, y una última actuará como señal de selección, escogemos el dispositivo con el que queremos realizar el intercambio de información (López, 2011).

Para llegar a entender este estándar de comunicación hay que tener en cuenta que esta interfaz se rige a través de un método de transmisión de información Maestro-Esclavo, donde el maestro será el encargado de iniciar la transferencia de datos sobre el bus y generar las señales de reloj que vamos a emplear, mientras que el esclavo estará controlado por el maestro y solo podrá actuar cuando el maestro se lo permita. Esta selección del esclavo se hará mediante un chip de selección, por lo que como, ya hemos dicho antes, el esclavo solo actuará cuando el maestro se lo pida.

A continuación reflejamos un resumen de las especificaciones del bus, donde como ya hemos comentado se realiza la transmisión de la información y la sincronización de los datos por medio de 4 señales:

- Una señal de reloj (*SCLK*) que será generada por el maestro donde se envía un bit en cada uno de los pulsos de reloj.
- Una línea de transmisión MOSI (*Master Output Slave Input*) para enviar los datos desde el maestro hacia el esclavo.
- Una línea de transmisión MISO (*Master Input Slave Output*) para enviar los datos desde el esclavo hacia el maestro.
- Una señal de selección (*SS/Select*) para que el maestro pueda seleccionar el esclavo que se activa en cada caso.

La transmisión de datos está regida por la señal de reloj (SCLK) que genera el maestro, como se muestra en la figura 3.1. Se puede observar que ambas líneas de flujo de información, tanto MOSI como MISO, siguen la señal de reloj que el maestro ha preestablecido en un principio. (Sparkfun (2017))

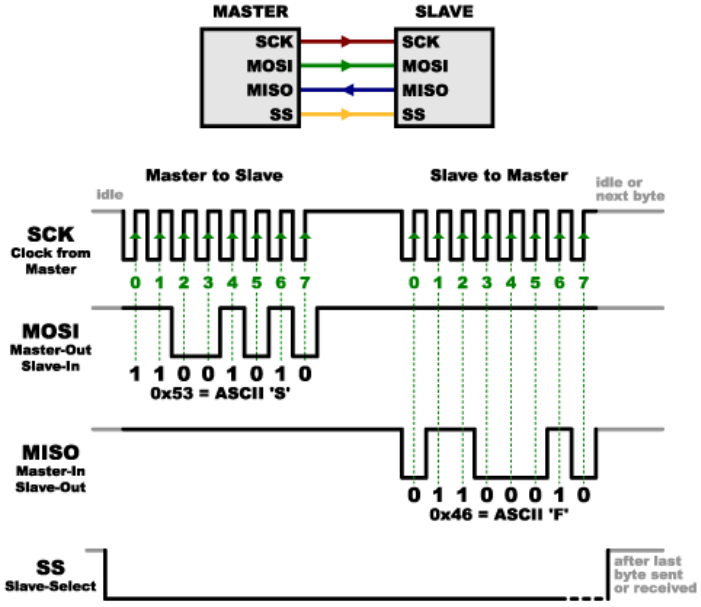


Figura 3.1: Transmisión de datos Maestro-Esclavo regida por una señal de reloj impuesta por el maestro (Sparkfun (2017))

Uno de los aspectos importante a tener en cuenta en este estándar de comunicación es que solo podrá haber un maestro. Generalmente éste será un microcontrolador, que se encargará de gestionar las tareas que tiene que realizar los distintos elementos que componen un dispositivo. Por otro lado, en lo referente a los esclavos, podremos disponer de más de uno, siempre y cuando cada uno de éstos disponga de su chip de selección.

Cuando tratamos con transmisión de información a muchos esclavos a la vez, se pueden dar dos casos principales. En el primer caso todos los dispositivos esclavos tendrán

su señal de selección conectada a la misma señal de selección del maestro, por lo que, cuando el maestro active esta señal de selección, todos los dispositivos se activarán (figura 3.2). De lo contrario en un segundo caso, del que podríamos decir que se trata del caso más general, cada uno de los esclavos están conectados a una señal de selección distinta proveniente del maestro, por lo que en este caso el maestro decidirá cuál es el dispositivo que quiere que se active en cada momento, como se muestra en la figura 3.3 (Sparkfun (2017))

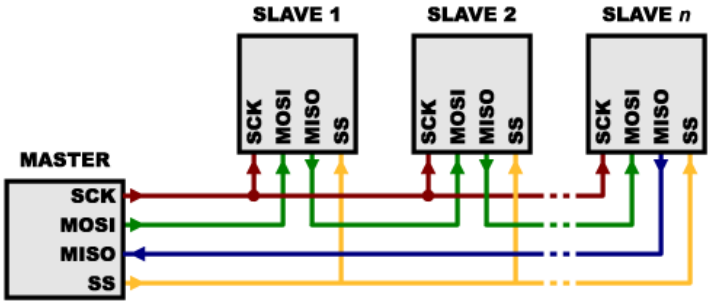


Figura 3.2: Dispositivos esclavos conectados a una misma señal de selección del maestro (Sparkfun (2017))

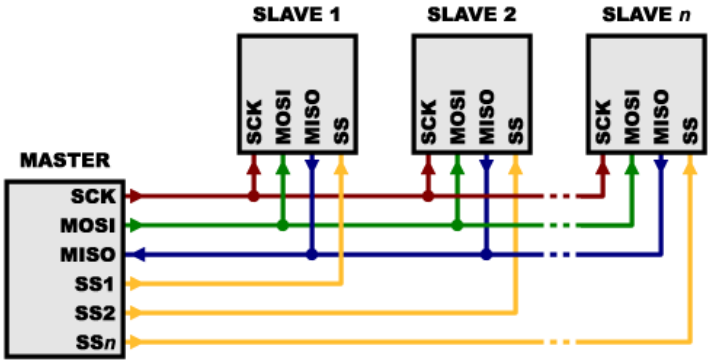


Figura 3.3: Dispositivos esclavos conectados a distintas señales de selección del maestro (Sparkfun (2017))

3.2.1. Modos de operación de reloj

La interfaz Spi cuenta con dos bits de configuración, los cuales nos indicarán como se realizará la transferencia de datos por la línea de reloj sincronizada de este bus, en la que en cada ciclo de reloj se enviará un bit. Estos dos bits se configuración son:

- CPHA (*Clock Phase*): nos indica en que flanco de reloj (subida o bajada) serán detectados los datos.
- CPOL (*Clock Polarity*): nos indica el estado del Idle.

Cuando CPOL es igual a cero (CPOL= 0) significa que el estado del Idle de la línea de reloj lo tenemos situado en bajo, mientras que si es igual a uno (CPOL=1) el estado del Idle estará situado a alto. Por otro lado, cuando CPHA es igual a cero (CPHA= 0) los datos serán detectados en cada flanco de bajada de la línea de MOSI y, en cada flanco de

subida serán detectados en la línea MISO. Cuando CPHA sea igual a uno (CPHA= 1) se dará el proceso inverso (López, 2011).

Al tener dos bits de configuración podremos tener varias posibles combinaciones de CPOL y CPHA, de manera que la transmisión de un bit podrá realizarse de los siguientes modos:

- Se podrá transmitir el bit en flanco de subida con y sin retraso.
- Se podrá transmitir el bit en flanco de bajada con y sin retraso.

Para que dos dispositivos puedan comunicarse mediante la interfaz SPI deben de tener el mismo Clock Polarity y Clock Phase, para que nos coincida en el mismo flanco de subida y bajada (López, 2011).

3.3. Lista de materiales

En este apartado realizaremos una lista con todos los materiales necesarios para el diseño de nuestro simulador, y realizaremos una breve explicación de los más importantes. Debemos hacer hincapié en la funcionalidad que tendrán en nuestro diseño y en la forma que trabajarán y se comunicarán unos de con otros para llevar a cabo el propósito de nuestro proyecto. En el diseño hay tres elementos fundamentales:

- ✓ Kit Menta de Adafruit
- ✓ Convertidor Digital/ Analógico: Microchip MCP4921-E/P-ND,
- ✓ Pantalla numérica: Sparkfun COM-11442 7-Segment Serial Display – Blue

También serán necesarios algunos dispositivos pasivos y conectores:

- ✓ Potenciómetro 5K
- ✓ Capacitor: 0.1 uF bypass (derivación) capacitor, 50 V, 20 %
- ✓ Resistencias: dos 10 K resistor $\frac{1}{4}$ watt y una 10 ohm resistor $\frac{1}{4}$ watt
- ✓ Tres receptáculos Banana

Adafruit Menta Para la realización de nuestro proyecto trabajaremos en una placa de circuito impreso muy parecida al Arduino UNO convencional llamada Adafruit Menta (Figura 3.4). Hemos elegido esta placa por distintos motivos, pero la razón principal por la que la escogimos es porque podremos soldar nuestro circuito en una protoboard que tiene incorporada y así evitaremos tener que utilizar alguna externa. Además incluye una caja metálica, que es donde alojaremos nuestro circuito y la utilizaremos como carcasa para nuestro simulador de ECG portátil.

Figura 3.4: Adafruit Menta Kit como base del montaje de nuestro simulador de ECG (Menta, 2017).

Una de las características que tiene esta placa es que trabaja con el mismo micro controlador (ATMEGA 328P) de Arduino UNO, y el entorno de desarrollo en la que la programaremos posteriormente es compatible con Arduino, por lo que contaremos con el soporte de una comunidad e usuarios amplia y abierta. Tenemos que tener en cuenta que para conectar nuestra placa para poder programarla con Arduino hemos necesitado un adaptador de la página oficial de Adafuit llamado FTDI Friend, para poder realizar la conexión USB (Figura3.5).

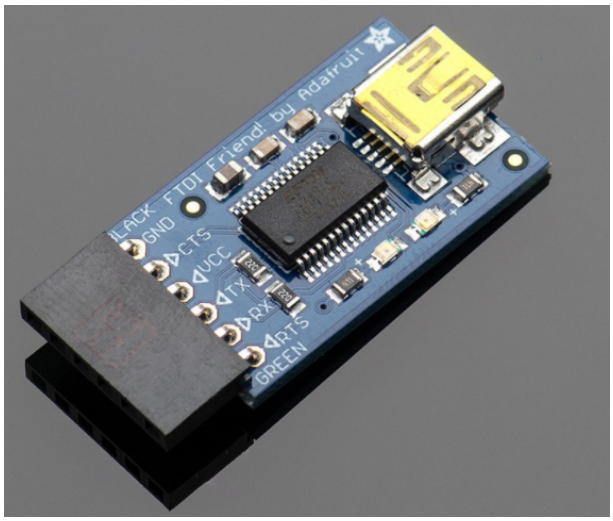


Figura 3.5: Adaptador FTDI Friend para la conexión USB de la placa (Menta, 2017).

Nuestra placa se comunica a través de la interfaz Spi con los otros dos componentes básicos de nuestro diseño:la pantalla de visualización numérica y el convertidor A/D (MPC4921). Para ello, la placa, tiene algunos pines predeterminados para este tipo de comunicación. Sin embargo, a la hora de buscar las conexiones de ésta no vienen como tal, y tendremos que utilizar las conexiones de la placa Duemilanove como nos indica en la página oficial de Adafruit. Los pines de conexión para la comunicación a través de la interfaz Spi con los siguientes:

- SCLK: PIN 13

- MISO: PIN 12
- MOSI: PIN 11
- SS: (este PIN puede ser variable)

Pantalla Numérica de 4 Dígitos Utilizaremos una pantalla de visualización numérica, concretamente la "SparkFun 7-Segment Serial Display – Blue", que combina un 4 dígitos de 7 segmentos con un ATMEGA328 para controlar cada uno de los segmentos de la pantalla. Ésta nos ofrece una gran variedad de posibilidades de escritura por pantalla, ya que nos permite mostrar números enteros y decimales, la mayoría de las letras y algunos caracteres especiales. Además nos permite controlar sus niveles de brillo, velocidad de transmisión, etc.

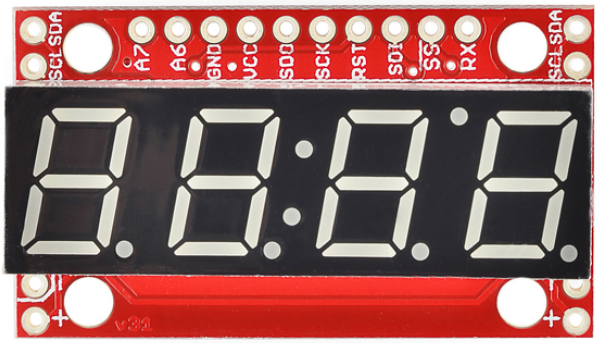


Figura 3.6: Pantalla numérica de 4 dígitos. Ésta se comunica por el estándar SPI (Sparkfun (2017))

La finalidad de ésta es recoger los datos de las pulsaciones que vamos a ir generando en cada momento con la variación del potenciómetro. Para ello, lo primero que tenemos que hacer es conectarla a los pines que le correspondan de la placa de Adafruit Menta. Esta pantalla se puede comunicar por distintos estándares de comunicación, como son TTL, Spi o I2C. En nuestro caso vamos a utilizar el estándar Spi, donde, como ya hemos visto anteriormente, tendremos una señal de reloj (SCLK), una señal de selección (SS), una línea de MOSI y una línea de MISO, (aunque en nuestro caso no necesitaremos una línea MISO porque no queremos que la pantalla le envié datos en ningún momento a nuestro micro controlador, solo queremos mostrar los datos por pantalla a través de la línea MOSI (Master Output Slave Input). Por lo tanto, los pines de conexión con la placa de Adafruit son los siguientes (podemos ver su correspondencia en la figura 3.6):

- SCLK: PIN 13
- MOSI: PIN 11
- SS: PIN 9

Convertidor (D/A) Como desafortunadamente el ATMega328P no dispone de convertidor digital-analógico (CDA), será necesario añadir un convertidor externo para generar la forma de onda del ECG analógica, por lo que utilizaremos el MCP4921. Así éste nos permitirá recoger las formas de onda digitales provenientes del micro controlador y convertirlas a analógica para generar la forma de ECG. Este componente también se comunicará

a través de la interfaz Spi, por lo que dispondrá de una señal de reloj, una señal de selección y una línea MOSI. No incluiremos la línea de MISO, ya que la función de éste (como yo ocurriera con el display) será únicamente recibir los datos provenientes del micro controlador y sacarlos por su pin de salida (Vout). Por lo tanto los pines utilizados en la placa de Adafruit serán los siguientes:

- SCLK: PIN 13
- MOSI: PIN 11
- SS: PIN 10

En la representación del microchip de la figura 3.7, se muestra la funcionalidad de cada una de sus pines (MCP4921 (2007)), necesario obviamente para saber cómo conectarlo posteriormente en nuestro circuito.

Figura 3.7: Microchip MPC4921 para la conversión (D/A)

- PIN1 (VDD): alimentación de energía. Lo conectamos a 5 V.
- PIN2 (CS): se trata del pin de selección del estándar Spi. Lo conectamos al PIN 10 de Adafruit.
- PIN3 (SCK): señal de reloj del estándar Spi. Lo conectamos al PIN13 de Adafruit
- PIN4 (SDI): señal MOSI (Master Output Slave Input) del estándar Spi. Lo conectamos al PIN 11 de Adafruit.
- PIN5 (LDAC):transfiere los datos del registro cuando está a nivel bajo.
- PIN6 (VREF): tensión de referencia.
- PIN7 (AVss): tierra analógica. Lo conectamos a GND
- PIN8 (Vout): salida analógica.

Además de estos tres elementos, el sistema incorpora un potenciómetro, para variar la frecuencia de las pulsaciones, y un divisor de tensión que permite atenuar la señal generada. Pese a su simplicidad, se describen brevemente también estos dos subsistemas para que así quede justificado claramente su posterior uso en el circuito

Potenciómetro Con la ayuda de este potenciómetro podremos variar la frecuencia de las pulsaciones que mostramos en la pantalla numérica y el periodo de reposo que se muestra en la visualización de la forma de onda de ECG, es decir, nos permitirá juntar o separar las formas de onda a medida que aumenta o disminuye la frecuencia cardíaca. Este concepto se explicará con mayor detenimiento en el desarrollo del software.

La conexión del potenciómetro en nuestro circuito será muy sencilla. Lo primero que tenemos que tener en cuenta es que éste consta de tres patas, las cuales dos de ellas serán fijas, y la pata central será variable, permitiendo así crear dos resistencias de valor controlado (según conectemos con uno u con otro terminal fijo). En la figura 3.8 se muestra la representación de un potenciómetro general y la conexión de sus terminales.

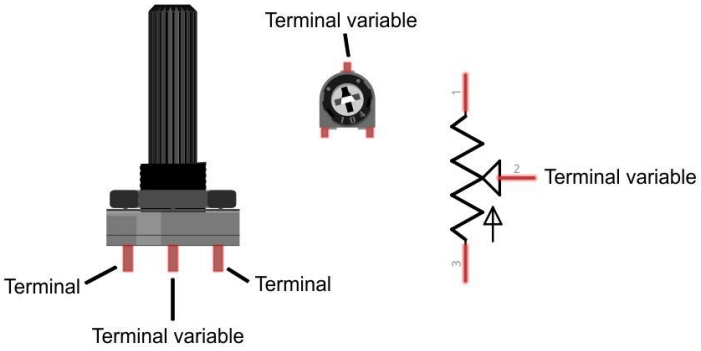


Figura 3.8: Potenciómetro con resistencia variable

Divisor de tensión Tenemos un conjunto de tres resistencia en serie para formar un divisor de tensión resistivo, el cual lo utilizaremos para atenuar la señal analógica, de la forma de onda de ECG que obtenemos finalmente, a los niveles requeridos de mili voltios que es la amplitud típica de una señal de ECG humana capturada con la colocación de los electrodos en el tórax.

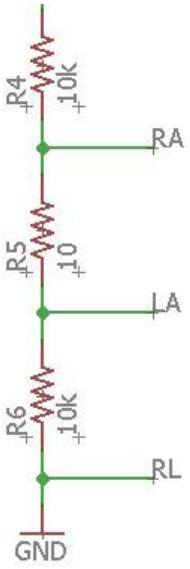


Figura 3.9: Divisor de tensión formado por tres resistencias

3.4. Diseño en EAGLE

Una vez conocida la función y el funcionamiento de cada uno de los componentes que utilizaremos en nuestro circuito, realizaremos un esquemático de nuestro diseño electrónico en EAGLE antes de dar comienzo al montaje en la placa de Adafruit.

Para la creación del diseño en Eagle (Figura 3.10), nos hemos servido de un esquemático previamente realizado de la placa de Adafruit menta, donde ya teníamos insertado el microcontrolador ATMEGA328p con las conexiones internas de la propia placa[1]. A continuación insertamos nuestro Display Sparkfun COM-11442[2] y el microchip MCP492, y procedemos con las conexiones.

El esquemático se observa en la figura 3.10. Se aprecia el elemento central, el microcontrolador, en torno al cual tenemos un cristal de cuarzo, un botón de reset y un conector de expasión. En el diagrama se han insertado el display y el convertidor, así como todos los elementos de conexión necesarios. Para realizar un esquemático claro y sencillo de interpretar, nos hemos servido del uso de etiquetas para realizar las conexiones. Las etiquetas nombradas por: SCK, CS y MOSI, hacen referencia a las líneas del estádar de comunicación SPI (como hemos visto anteriormente), por el que se comunican el display y el convertidor. El convertidor, tiene conectado a su salida (Vout), el divisor, ya comentado anteriormente, formado por tres resistencias en serie, que permitirá obtener la forma de onda del ECG reducida al orden de milivoltios.

[1] https://github.com/adafruit/Adafruit-Menta/blob/master/menta.sch
[2] https://www.sparkfun.com/products/11442

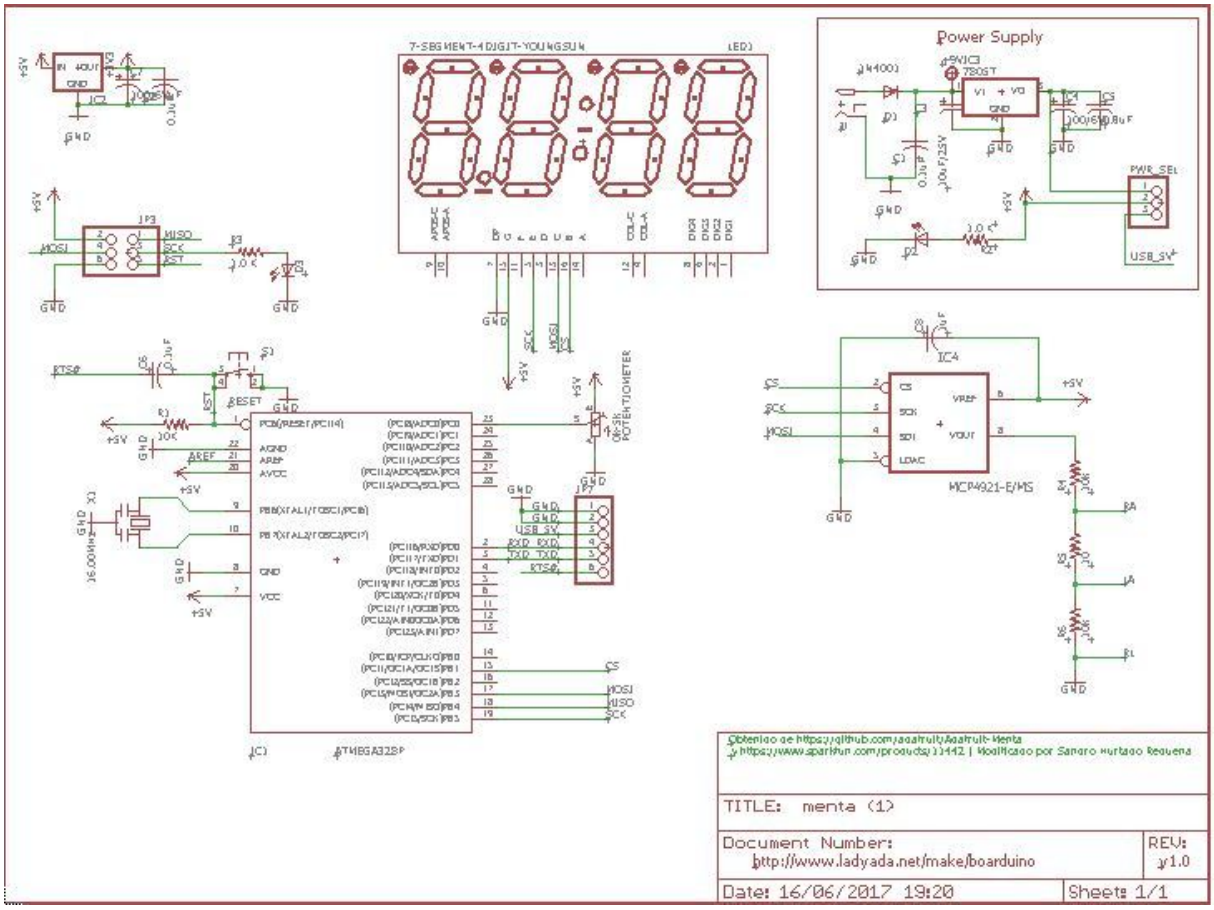


Figura 3.10: Diseño EAGLE

3.5. Ensamblaje y montaje

Desafortunadamente la placa de Adafruit Menta no viene previamente montada, por lo que hemos tenido que soldar todos sus componentes. Por suerte, hemos podido fijarnos en un tutorial de Adafruit[3] donde nos guiaba en el montaje de éste.

Los primeros montajes los hemos realizado en una protoboard auxiliar a la de nuestra placa Adafruit Menta, para comprobar el correcto funcionamiento de nuestro circuito, antes de proceder a su montaje final. Así, podemos observar que las conexiones no están realizadas de la forma más apropiada, ya que hay cables muy largos para conectar la placa a la protoboard pudiendo producirse ruido en la señal y algunas interferencias. Aunque no debemos preocuparnos, ya que éste solo es un diseño para realizar pruebas de que nuestro diseño electrónico funciona correctamente.

En la imagen 3.11 podemos observar que el circuito funciona perfectamente, mostrando a través de la pantalla numérica la frecuencia seleccionada con el potenciómetro. De esta forma, nos aseguramos que todas las conexiones están bien.

[3]https://learn.adafruit.com/adafruit-menta-kit-mint-tin-arduino-compatible

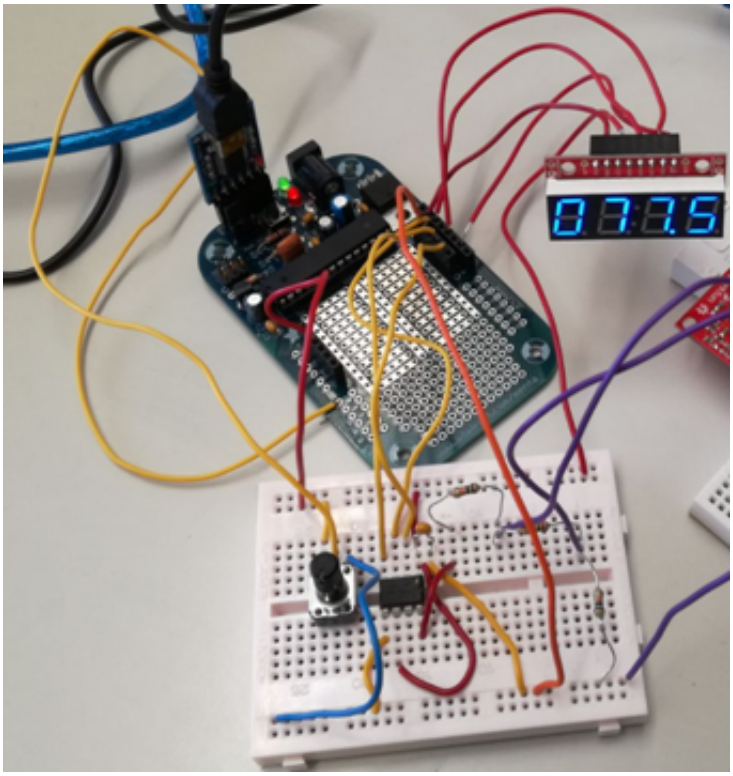


Figura 3.11: Pruebas Simulador de ECG

El siguiente paso, supone trasladar el circuito que actualmente teníamos montado a la placa de Adafruit. Por lo tanto, procedemos a soldar nuestro circuito en la protoboard que tiene integrada Adafruit Menta. La comunicación de nuestro circuito con la placa la hemos realizado a través de sus pines, tanto analógicos como digitales.

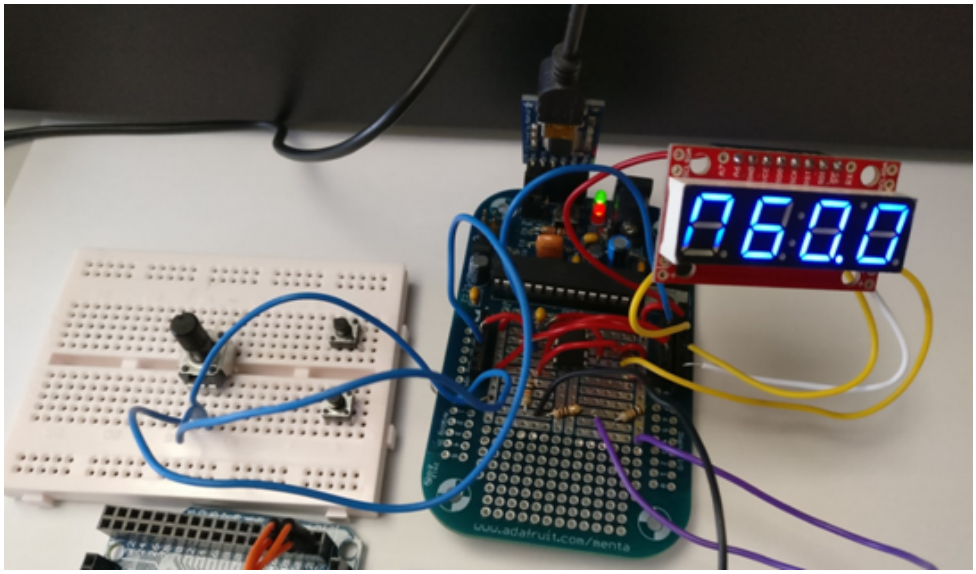


Figura 3.12: Placa Adafruit Menta con circuito Simulador de ECG

Ahora sí, tenemos nuestro circuito implementado directamente en nuestra placa de Adafruit Menta. En la imagen 3.12 se aprecian bastante mejor las conexiones del microchip MPC4921, la pantalla numérica y el potenciómetro. Es importante destacar que no todos los elementos se montarán sobre el grid qe ofrece la placa de Adafruit Menta, pues el display o el potenciómetro deberán quedar accesibles a la carcasa.

Se puede observar que los cables analógicos por donde obtendremos nuestra forma de onda de ECG (cable negro: RA, cable morado central: LA y cable morado en GND: RL), están colocados entre las resistencias de nuestro circuito, que actuarán como un divisor de tensión para reducir la intensidad de las señales al orden de milivoltios como en un ECG real.

Capítulo 4

Desarrollo del Software

Una vez tenemos nuestra placa montada, ya estará preparada para empezar a programar. En éste capítulo se detallarán los pasos necesarios para el desarrollo de nuestro software.

4.1. Introducción

Lo primero que tendremos que hacer para comenzar a trabajar, será escoger una imagen de ECG en condiciones normales que consistirá en una onda P, un complejo QRS y una onda T, como hemos explicado anteriormente en el capítulo de introducción al Electrocardiograma. Las propiedades que debemos de observar para escoger nuestra imagen es que cumplan las características principales de las formas de onda del ECG que se describieron en el Capítulo 2 de la presente memoria, y que se resume a continuación:

- La onda P, correspondiente a la despolarización auricular en condiciones normales, no debería de ser mayor a 0,25 mV, ni superar los 110 milisegundos de duración.

- El complejo QRS correspondiente a la despolarización ventricular, tiene una duración entre 60 y 100 milisegundos y un voltaje no superior a 3,5 mV. Además deberá tener una onda Q negativa, una onda R positiva, que es la de mayor tamaño, y por último una onda S negativa.

- La onda T correspondiente a la repolarización de los ventrículos, tendrá una duración de 0,20 segundos aproximadamente y un valor entre 0,2 y 0,3 mV.

En la siguiente figura 4.1, se muestra la imagen que hemos escogido, ya recortada y arreglada para su uso, que cumple con las características mencionadas anteriormente y además será perfecta para su posterior digitalización, ya que se encuentra cuadriculada y nos facilitará saber la amplitud y la duración de cada una de las formas de onda.

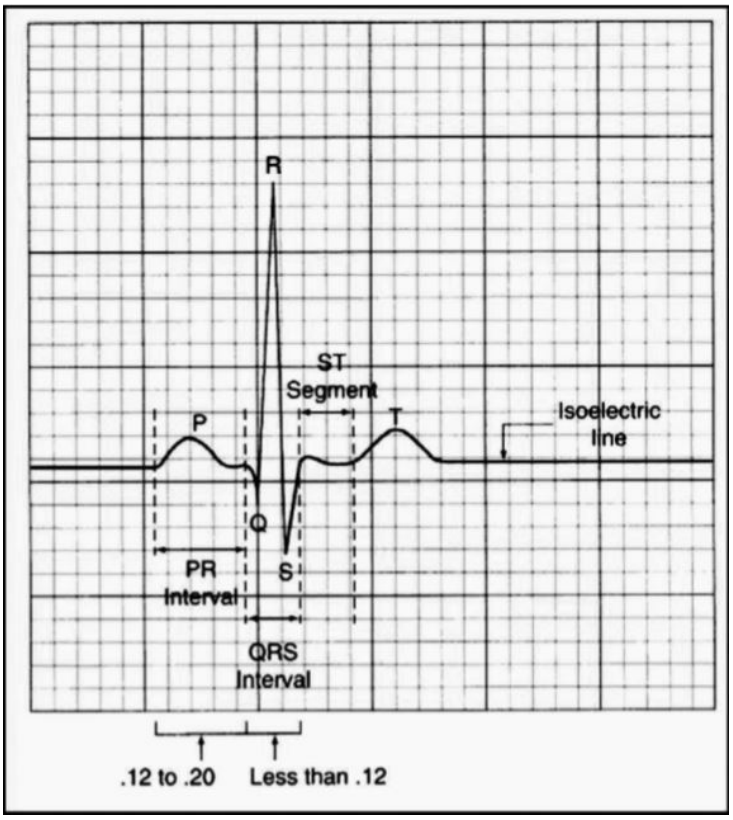


Figura 4.1: Imagen de un ECG en condiciones normales

En la figura 4.1 tenemos representado cuadriculadamente, la amplitud y el tiempo de la forma de onda del ECG. En el eje Y tenemos representado el voltaje y en el eje X el tiempo transcurrido. Cada uno de los cuadros grandes en el eje X representan 200 milisegundos y los cuadros pequeños representan 40 milisegundos, mientras que en el eje Y los cuadros grandes representan 1 mV y los pequeños 0,2 mV. Por lo tanto, se puede concluir, que esta imagen cumple con las especificaciones anteriormente mencionadas. Además debemos de tener en cuenta que después de la onda T, habrá un periodo de reposo, representado en la figura 4.1, en la línea isoeléctrica, hasta la siguiente forma de onda de PQRST.

Sabemos que las pulsaciones de una persona en condiciones normales pueden variar tipicalmente en un rango entre 60-100 pulsaciones por minuto aproximadamente, por lo tanto si conseguimos variar el periodo de reposo de nuestra gráfica aumentándolo o disminuyéndolo podremos variar el ritmo cardiaco. Sabemos que a medida que aumenta nuestra frecuencia cardíaca, en un ECG normal, podríamos observar como la forma de onda QRS se comprime muy ligeramente. Sin embargo nosotros simplificaremos esto manteniendo la forma de onda constante, es decir, solo variaremos su periodo de reposo.

4.2. Digitalización de la imagen

El siguiente paso de nuestro proyecto, consistirá en digitalizar la imagen obtenida de la forma de onda del ECG con el programa Engauge Digitizer, que es comúnmente utilizado para extraer puntos de datos de imágenes de gráficos. Este programa es ideal para nuestro proyecto, ya que se trata de un programa de código abierto y de fácil aprendizaje. Posteriormente utilizaremos esta imagen digitalizada en nuestro sistema.

Con el programa Engauge Digitizer obtendremos la forma de onda de ECG digitalizada en forma de coordenadas (X,Y), donde cada punto, nos indicará su posición en el tiempo (X) y su amplitud en mV (Y).

Para comenzar a trabajar en el programa insertaremos la imagen de la forma de onda de ECG. Una vez que hemos insertado la imagen, procedemos a declarar nuestro sistema de referencia (X, Y), en el que hemos tenido que seleccionar tres puntos para poder declararlo adecuadamente (cruces rojas de la figura 4.2): un punto que representa el origen de referencias $(0, 0)$, un punto para representar el eje X $(0{,}8, 0)$, y por ultimo un punto para representar el eje Y $(1{,}5, 0)$. Al igual que en la imagen 4.1, en el eje X se representará el periodo de la onda (ms), y en el eje Y su amplitud en milivoltios (mV). Los puntos que declaran los ejes (X, Y), los hemos escogido de tal forma que la forma de onda PQRST quede dentro de éstos. Una vez declarado los ejes, seleccionamos la herramienta de selección de puntos, y ya podemos empezar a seleccionar manualmente punto a punto la forma de onda PQRST, que es el segmento de la imagen que queremos digitalizar.

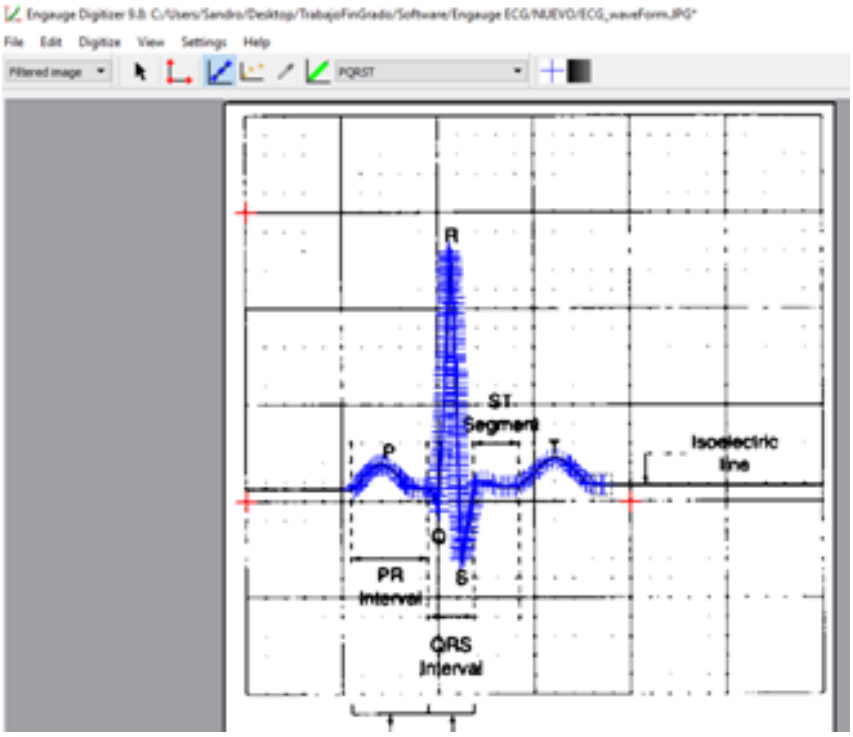


Figura 4.2: Selección de puntos de la onda PQRS para digitalizarla

La forma de onda PQRST ya está completamente digitalizada en parejas de puntos (X, Y), por lo que procederemos a extraer la información a un documento de texto, como se muestra en la figura 4.3, para su posterior utilización en el código de Arduino.

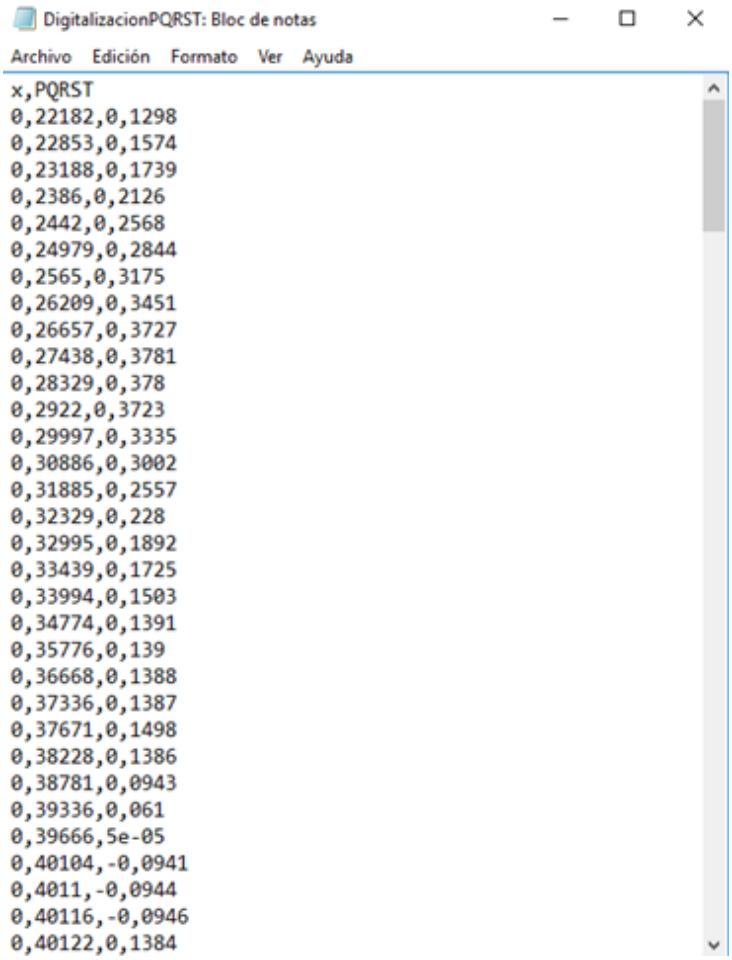


Figura 4.3: Coordenadas de puntos (x,y) previamente seleccionados, y extraídas en un archivo.txt

4.3. Phyton

4.3.1. Introducción

Los datos obtenidos en el programa anterior están almacenados en un documento de texto, de forma que nos muestra en cada línea, puntos de datos de coordenadas (x, y). Para poder trabajar con esta información en nuestro código Arduino, debemos realizar previamente una declaración de una constante C-matriz que contenga estos datos.

Vamos a desarrollar un programa en Phyton donde introduciremos los datos contenidos en el archivo de texto y nos devolverá una constante C-matriz de puntos de datos con un inicializador. Los valores obtenidos en la constante C-matriz, que corresponden con la forma de onda digitalizada, se pasan al convertidor D/A que es el encargado de convertir la forma de onda de digital a analógica, y la envía por su salida (Vout) para simular la actividad eléctrica del corazón. El Microchip MPC4921 es de 12bits, por lo que los valores de Y, correspondientes a la amplitud de la onda (mV), se escalarán en un rango de valores de 0 a 4095. Por otro lado, los valores de X, correspondientes al tiempo (ms), no harán falta introducirlos en Arduino, ya que nosotros declararemos que las muestras se enviarán cada 1 ms al convertidor (A/D), para que estén equiespaciadas.

4.3.2. Desarrollo del programa Phyton

Las bases fundamentales de este programa la podemos resumir a un único punto principal: interpolación lineal de los puntos (X,Y) del archivo de texto obtenido del programa anterior (Engauge DIgitizer). De esta forma integraremos el conjunto de valores en una única función F(x), que la representaremos como una constante C-matriz, que puede utilizarse posteriormente en Arduino.

Interpolación Lineal

Esta función será necesaria para la representación de nuestra forma de onda, ya que los valores obtenidos, procedentes del programa Engauge Digitizer proporcionan solo una aproximación a la forma de onda, al haber realizado una selección manual de los puntos.

Con la interpolación lineal podremos estimar un valor intermedio de los valores conocidos, siendo esta estimación proporcional a la proximidad de los datos que queremos interpolar. Por suerte, en la selección de puntos de la imagen con el programa Engauge Digitizer, tuvimos en cuenta este factor proximidad entre los puntos. Cogimos tantas muestras de puntos como pudimos, lo más cerca unas de otras.

La idea básica de esta función será, conectar los puntos dados en x_i, a través de una línea recta entre los puntos, como se muestra en la figura 4.4. De ésta forma obtendremos una imagen de la forma de onda de ECG.

Para cualquier punto entre los dos valores de x_0 y x_1, se debe seguir la ecuación de la línea que se puede derivar geométricamente.

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}$$

Debemos de calcular el valor de y, que representa el valor desconocido para x, quedando de la siguiente manera la ecuación:

$$y = (x - x_0) \cdot \frac{y_1 - y_0}{x_1 - x_0} + y_0$$

Sabemos que x estará comprendido entre $x_0 < x < x_1$. Éste es el resultado para ese punto, por lo que en nuestro caso que tenemos más de un punto, se concatenarán la interpolación lineal entre pares de puntos continuos.

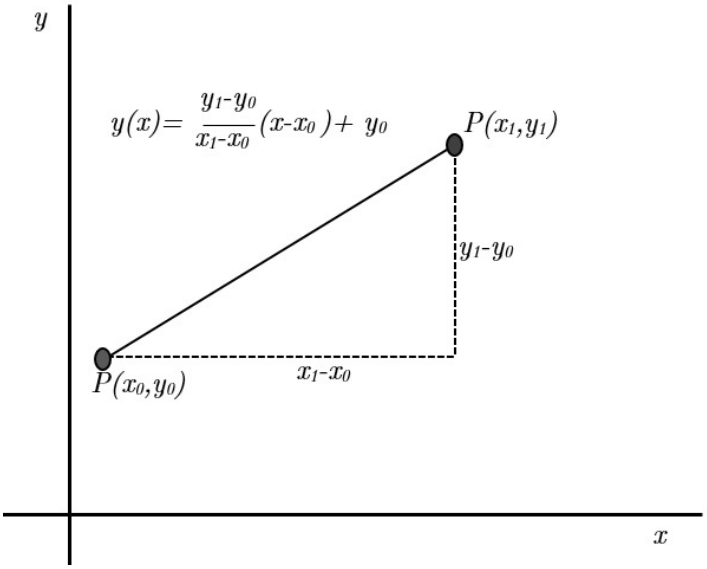


Figura 4.4: Interpolación lineal de dos puntos

Finalmente este programa nos proporcionará una función F(x), que la representaremos como una constante C-matriz, como se muestra en la figura 4.5. Esta función implementa el algoritmo de interpolación lineal, donde X es la variable de entrada a la función. La tabla de funciones se almacena en dos matrices llamadas x_values e y_values, entando x dentro de x_values.

DigitalizacionPQRSTArduino.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
const short  y_data[] = {
940, 945, 950, 956, 961, 966, 971, 977, 983, 990,
996, 1003, 1011, 1018, 1026, 1033, 1040, 1048, 1059, 1069,
1079, 1089, 1099, 1107, 1113, 1120, 1126, 1132, 1139, 1145,
1151, 1158, 1164, 1170, 1177, 1183, 1189, 1196, 1202, 1208,
1215, 1222, 1230, 1238, 1246, 1252, 1253, 1254, 1255, 1256,
1257, 1258, 1258, 1259, 1259, 1259, 1259, 1259, 1259, 1259,
1259, 1259, 1258, 1258, 1257, 1256, 1255, 1254, 1253, 1253,
1252, 1248, 1241, 1235, 1228, 1222, 1215, 1209, 1203, 1198,
1193, 1188, 1183, 1178, 1173, 1169, 1164, 1159, 1153, 1148,
1142, 1136, 1130, 1125, 1119, 1113, 1107, 1102, 1094, 1086,
1078, 1070, 1062, 1055, 1047, 1040, 1032, 1025, 1017, 1012,
1007, 1002, 997, 992, 987, 982, 977, 972, 967, 964,
963, 961, 959, 957, 955, 953, 952, 952, 952, 952,
952, 952, 952, 952, 952, 952, 952, 952, 952, 951,
951, 951, 951, 951, 951, 951, 951, 951, 951, 951,
951, 951, 953, 957, 962, 965, 963, 960, 957, 955,
952, 945, 935, 925, 915, 904, 894, 886, 879, 871,
863, 855, 840, 817, 793, 769, 741, 713, 686, 658,
991, 1307, 1332, 1358, 1401, 1493, 1591, 1670, 1766, 1910,
1954, 2187, 2532, 2548, 2782, 2923, 3247, 3263, 3909, 3899,
3746, 4021, 4080, 4058, 4037, 3974, 3859, 3870, 3533, 3468,
3404, 3329, 3251, 3095, 2715, 2625, 2513, 2486, 2198, 1690,
1268, 832, 572, 306, 346, 72, 67, 62, 57, 52,
47, 115, 226, 259, 272, 288, 313, 338, 363, 467,
532, 561, 577, 594, 618, 673, 687, 702, 717, 824,
922, 941, 960, 978, 988, 997, 1007, 1016, 1022, 1025,
1028, 1031, 1033, 1036, 1039, 1042, 1045, 1047, 1048, 1048,
1047, 1046, 1046, 1045, 1044, 1044, 1043, 1042, 1042, 1040,
1037, 1034, 1032, 1029, 1026, 1023, 1020, 1017, 1014, 1011,
1008, 1005, 1001, 998, 994, 990, 987, 983, 979, 977,
977, 977, 977, 977, 977, 977, 977, 977, 977, 977,
977, 977, 977, 977, 977, 977, 977, 977, 977, 977,
977, 977, 977, 977, 978, 979, 979, 980, 981, 981,
```

Figura 4.5: Constante C-matriz de datos de la forma de onda de ECG

4.4. Código de Arduino

En este apartado del desarrollo del software, se explicará el funcionamiento interno de nuestro simulador de ECG. Veremos todos y cada uno de los pasos necesarios para el desarrollo de nuestro código, desde la creación de nuestra forma de ECG, a partir de las muestras de la constante matriz-C de datos (*const short y_data*) hasta su envío por las salidas analógicas, a través de las derivaciones (RA, LA, RL).

4.4.1. Especificaciones del programa y funcionamiento de nuestro circuito

Nuestro programa seguirá un conjunto de especificaciones y requisitos que se exponen a continuación:

* La forma de onda se emitirá a una velocidad de 1000 muestras por segundo, o en otras palabras, 1 muestra por cada milisegundo. Las muestras se enviarán al microchip MPC4921, para convertir de digitales a analógicas.

* Cada 50 milisegundos, se visualizará en la pantalla de 4 dígitos la frecuencia cardíaca seleccionada por el usuario, a través del potenciómetro conectado como una entrada analógica.

* El código será modular.

* La forma de onda estará escalada en un intervalo de 0 a 4095, correspondiente a la escala completa del convertidor D/A de 12 bits.

* El almacenamiento de la forma de onda se hará en la memoria EPROM (*Erasable Programmable Read-Only*), mediante el uso de la constante C-matriz. Aquí nos encontramos con un problema, y es que esta memoria es muy limitada si queremos almacenar varias formas de ondas en nuestro programa, por lo que tendríamos que recurrir al uso de un módulo auxiliar de almacenamiento, por ejemplo una micro SD externa. Aunque finalmente en nuestro caso no hará falta, es una limitación importante para simular formas anormales de ECG más complejas.

* Tanto el convertidor D/A como la pantalla numérica de 4 dígitos se comunicarán a través del estándar Spi, por lo que tendremos que declarar sus pines correspondientes de señal de reloj (SCK), MOSI (Master Outout Slave Input) y su señal de selección (SS).

4.4.2. Estructura modular del código. Organización por tareas

Trabajaremos utilizando una estructuras por tareas, para poder realizar el envío de la forma de onda del convertidor D/A cada milisegundo, y poder visualizar la frecuencia cardíaca variada por el potenciómetro, en la pantalla numérica cada 50 milisegundos, de forma simultánea.

Utilizaremos un editor de texto, llamado Atom, que nos será muy útil para desarrollar las tareas, ya que nos facilitará la escritura del código reconociendo la declaración de variables, clases, funciones, etc.

Contaremos con tareas, a las cuales haremos llamadas desde nuestro programa principal de Arduino. Entre las tareas que hemos desarrollado se encuentran:

- ***ReadPot***, será la encargada de realizar la lectura analógica del potenciómetro y enviar los datos a la pantalla numérica de 4 dígitos para visualizarlos.
- ***SignalSender***, crearemos nuestra forma de onda a partir de la constante C-matriz donde tenemos almacenadas las muestras y la enviaremos al convertidor D/A, para que la transforme a analógica, y la transmita por su pin de salida (Vout) a las tres derivaciones analógicas.
- **Tarea Periódica**, se trata de una tarea auxiliar, llamada por otras dos tareas para poder ejecutarse de forma periódica, con un tiempo determinado.
- ***SignalSamples***, será necesaria para el almacenamiento de la constante C-matriz, donde se almacenan las muestras de la forma de onda del ECG que queremos representar.

Asimismo, explicaremos la funcionalidad de cada una de las tareas de forma mas detallada. En primer lugar se expondrán las dos tareas principales de nuestro programa, *ReadPot* y *SignalSender*, y posteriormente se detallarán *TareaPeriódica* y SiganlSamples, como tareas complementarias.

ReadPot ()

En esta tarea realizaremos la lectura del potenciómetro, y la mapearemos con las pulsaciones por minuto que realmente queremos mostrar por pantalla. Sabemos que el potenciómetro viene regulado en un intervalo de 0 a 1023 y nosotros queremos que éste nos muestre valores entre las pulsaciones mínimas (ppmMin) de una persona y las máximas (ppmMax), declaradas anteriormente.

- int Valor = analogRead(0);
- ppm = map(Valor, 0, 1023, ppmMin, ppmMax);

Tenemos que tener en cuenta que el intervalo de pulsaciones las declaramos nosotros, siendo las pulsaciones mínimas, un número entero el cual nos parezca razonable, como por ejemplo 60 pulsaciones por minuto, y las máximas dependerá del número de muestras de nuestra forma de onda, de la siguiente forma:

- ppmMax = $(60000/N^{o}deMuestras)$ (ms);

Las pulsaciones máximas las hemos calculado teniendo en cuenta que, en un minuto (60000 milisegundos), caben un número determinado de formas de onda. Por lo tanto si dividimos estos 60000 milisegundos entre el tamaño de nuestra forma de onda en milisegundos (las muestras están equiespaciadas cada un milisegundo), obtendremos el número total de formas de onda que podemos graficar sin que éstas se superpongan, y teniendo un periodo de reposo entre ellas igual a cero.

A continuación hacemos un llamamiento a la función que tenemos declarada para nuestra pantalla numérica de cuatro dígitos (Display7Seg_Send), para enviarle el valor de las pulsaciones que tiene que mostrar. Para mostrar datos en la pantalla numérica, debemos de ir escribiendo dígito a dígito desde el primero hasta el último, para que cuando le enviemos el valor de las pulsaciones, éste coloque cada número en su posición correspondiente.

Por último enviaremos a la pantalla numérica los datos de los dígitos creados anteriormente, a través del comando SPI.transfer (Número de dígito).

SignalSender ()

Se trata de la tarea principal, donde crearemos nuestra forma de onda de ECG a partir de la constante matriz-C donde tenemos almacenadas las muestras. La onda de ECG la enviaremos al convertidor D/A, para que la transforme a analógica, y la transmita por su pin de salida (Vout) a los cables que representan las tres derivaciones analógicas.

Declaración de la forma de onda de ECG

La forma de onda de ECG constará de dos estados:

- Un estado PQRS en el que la forma de onda se encuentra en su periodo activo, el cual consta de las ondas P, el complejo QRS y la onda T. Este estado consta de un número de muestras fijo.
- Un estado de reposo, que será el periodo de transición entre ambas formas de onda. Al contrario que el estado anterior, el número de muestras de éste depende de la frecuencia cardíaca. Por ejemplo, a mayor frecuencia, menor será el número de muestras, y viceversa.

Es importante entender que el periodo de reposo depende directamente de la lectura del potenciómetro, ya que a media que se aumentan las pulsaciones con el potenciómetro, el tiempo de reposo disminuye.

Para la declaración de la forma de onda, se ha creado una función llamada Estado_ECG(). En esta función se darán los dos estados del ECG (estado activo: PQRS y estado de reposo), en los cuales, la forma de onda se crea a partir de las muestras de la matriz-C. El procedimiento para la creación de la representación de la forma de onda en tiempo real, consiste en pasar continuamenente de un estado a otro. Esto lo haremos declarando dos contadores, uno para el número de muestras de la onda activa (PQRS) y otro para el estado de reposo (tenemos que tener en cuenta que cada muestra se da cada 1ms de forma equiespaciada). En consecuencia, cuando nos encontramos en el estado activo de la onda, si el contador de la onda PQRS es mayor o igual al número de muestras que tiene la onda PQRS (el número de muestras de PQRS lo sabemos previamente), pasaremos al siguiente estado: estado de reposo. Asimismo, cuando estamos en el estado de reposo, si el contador del número de muestras del estado de reposo es mayor o igual al tiempo de reposo (calculado anteriormente), pasaremos al estado activo (PQRS). Y así, este proceso se repetirá cíclicamente para representar la forma de onda del ECG.

Los datos del estado activo (PQRS) y del estado de reposo, en cada caso, serán enviados cada un milisegundo al microchip MPC4921, para que convierta los datos a analógicos y enviarlos a los 3 cables que representan las salidas de los electrodos (RL, LA, RA).

Tenemos que tener en cuenta una serie de requisitos a la hora de enviar datos con nuestro convertidor A/D (MPC4921), ya que éste tiene una resolución de 12 bits y se comunicará a través de la interfaz Spi, la cual se comunica por tramas de datos de 8 bits (1 Byte). Por lo tanto, los datos serán enviados en dos segmentos de 8 bits cada uno, que será del 15-8 bits y de 7-0 bits. Los primeros 4 bits serán bits de configuración (15-12), mientras que los 12 bits restantes contienen el valor de los datos de la salida de nuestro convertidor (11-0).

Bits de configuración (MCP4921 (2007)):

- Bit 15: selector de salida (este bit no se utiliza en este convertidor). Lo ponemos a 0.
- Bit 14: controla la memoria que le llega a la entrada. Lo ponemos a 0 para trabajar de 0 a 5V
- Bit 13: ganancia del bit de selección. Lo ponemos a 1, para no obtener ninguna ganancia.
- Bit 12: bit de control. Lo ponemos a 1, para que el envió de datos se haga cuando el bit de selección este a 0 (*chip select low*) El envío de datos de nuestro Convertidor A/D se realiza en dos segmentos de 8 bits cada uno, y quedaría expuesto de la siguiente forma:

Cuadro 4.1: Descripción del envio de bits del microchip MPC4921

0	0	1	1	bit 11	bit 10	bit 9	bit 8
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Tarea Periódica ()

Con el desarrollo de esta tarea permitiremos que *ReadPot* y *SignalSender* puedan ejecutarse de manera periódica con una simple llamada a ella. El funcionamiento es bastante simple de entender:

- *TareaPeriódica* tendrá una variable que declararemos en el constructor llamada "per", que nos indica un periodo que le pasaremos previamente (por ejemplo, para la tarea *ReadPot* declaramos un per=50, para que se ejecute cada 50 ms).
- Declaramos una función loop, que se ejecutará en bucle. Si el periodo actual menos el periodo que ha pasado, es mayor o igual al periodo que le hemos declarado nosotros previamente (en nuestro caso 50 ms), llamamos a la función que queramos implementar en ese momento.

SignalSamples ()

En esta tarea declararemos el número de muestras de nuestra forma de onda ECG, en una constante C-matriz. Con esto, permitiremos que otras tareas como, *ReadPot* y *SignalSender*, puedan utilizar esta C-matriz, sin tener que declararla en su clase. De esta forma, se conseguirá una programación mas eficiente y con menos consumo de memoria, ya que bastará con declararnos la C-matriz una sola vez, y realizar llamadas para usarla desde otras tareas.

Programa principal Arduino

En esta parte del código nos limitaremos a realizar llamadas a nuestras tareas implementadas. En la función *setup()*, declararemos dos variables del tipo *ReadPot* y *SignalSender*, y le pasaremos como parámetros el tiempo de muestreo al que se tienen que ejecutar.

Por último, le pasaremos estas variables a la función *loop()* y utilizaremos la función *millis()* para que nos cuente el número de milisegundos desde que se ejecuta el programa en la placa. De esta forma, cuando pase 1 milisegundo se ejecutará la tarea de *SignalSender* y cuando pasen 50 milisegundos se ejecutará la tarea de *ReadPot*.

Capítulo 5

Pruebas y resultados

Para la elaboración de las pruebas y resultados de nuestro proyecto y poder visualizar las formas de ondas de nuestro simulador de ECG, tuvimos que servirnos de la ayuda de un sistema de adquisición de señal de ECG funcional, que afortunadamente dos compañeros de laboratorio están realizando como trabajo de fin de grado (TFG). (Galeote Checa, Gabriel)(Arrabal Caro, Lorenzo)

La adquisición de las señales del ECG se recogieron en una placa de desarrollo Arduino a la que se le añadió un módulo de tarjeta SD externo para guardar los datos obtenidos.

El la figura 5.1 podemos a preciar la conexión de nuestro simulador de ECG con el sistema de adquisición de señales. Ésta es bastante simple, ya que lo único que tenemos que hacer es conectar la salida analógica de nuestro circuito, mediante el uso de tres cables, con sus tres entradas RA, LA y RL. El simulador de ECG actúa simulando la actividad eléctrica del corazón y el sistema de adquisición de señales recogerá la señal por sus entradas analógicas. Éste es el mismo proceso que colocarle a una persona unos electrodos en el tórax para medir su actividad cardíaca.

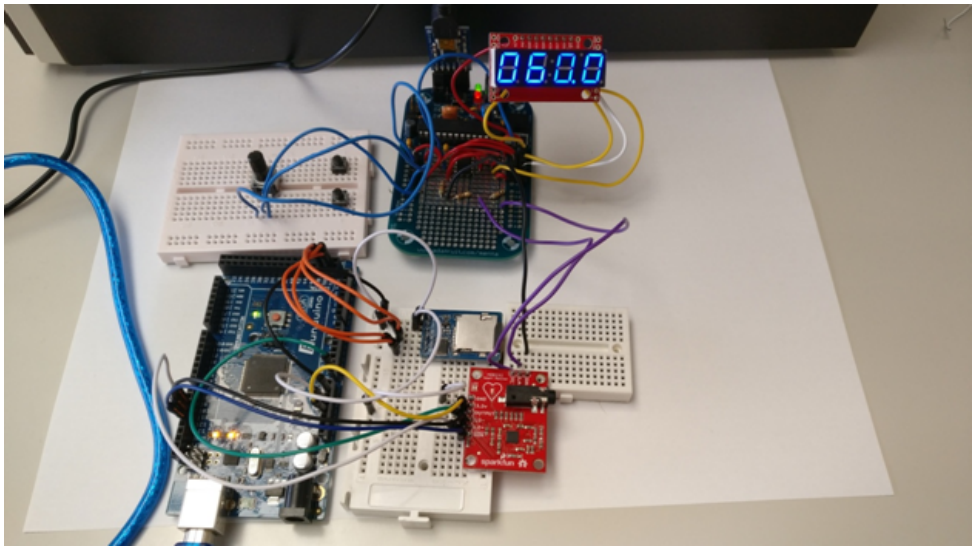
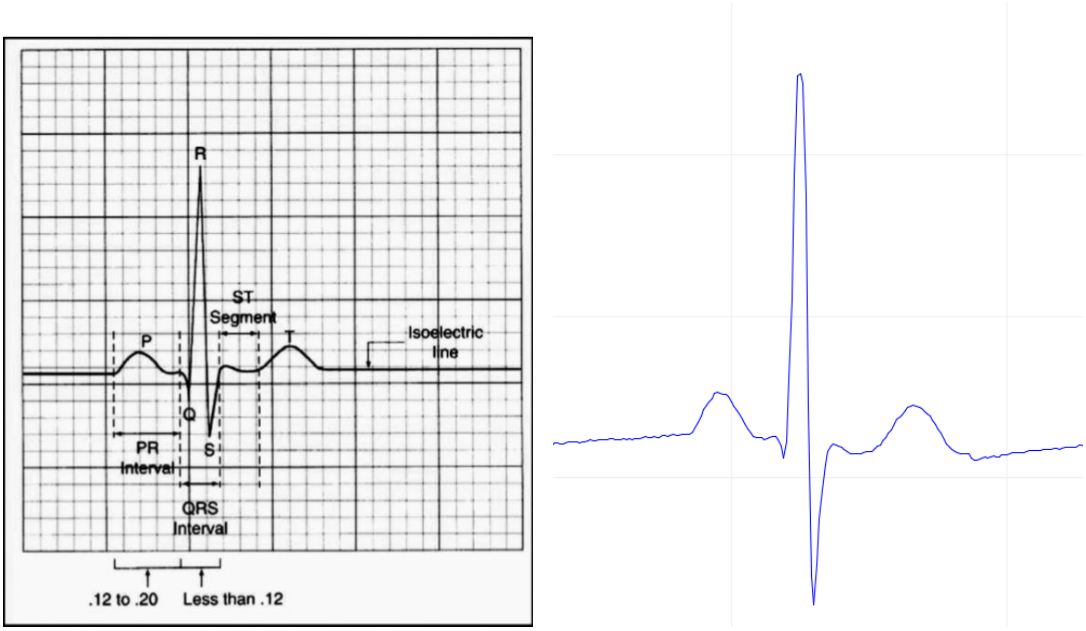


Figura 5.1: Conexión del Simulador de ondas de ECG con el sistema de adquisición de señales

Además, en la imagen de la figura 5.1, se observa que el display del simulador nos representa perfectamente la frecuencia cardíaca con un decimal, que le hemos enviado a través del potenciómetro. Cabe mencionar, que tuvimos que realizar algunos ajustes para la correcta visualización del display, ya que el último dígito de éste (el número decimal), presentaba una serie de interferencias y oscilaba de forma tediosa. Por ello, declaramos un "filtro", realizando una media de las últimas 32 muestras de pulsaciones que queríamos visualizar. De esta forma, logramos solucionar nuestro problema con éxito, y así representar las pulsaciones en el display, de forma más clara y precisa.

En la imagen 5.2 se realiza una comparativa de imagen original, escogida en un principio, (imagen 5.5(a)) y la imagen obtenida a partir de la representación de nuestro simulador de ECG (5.5(d)). Podemos observar a simple vista que la representación de la forma de onda se ha realizado con éxito, ya que es casi idéntica a la imagen original. La amplitud y la duración de las distintas ondas que representan el ECG son bastantes adecuadas.



(a) Imagen de ECG real

(b) ECG representado con el simulador

Figura 5.2: Comparación de la representación del ECG real con la simulación

Hemos podido realizar bastantes pruebas asegurando que nuestro simulador funciona correctamente. Para la forma de onda que hemos digitalizado, escogimos un intervalo de frecuencias de 60 a 114,7 pulsaciones por minuto (*ppm*), para visualizar la señal de ECG. El intervalo de frecuencias lo fijamos nosotros mismos, considerando razonable poner un mínimo de 60 *ppm*, ya que no es demasiado normal tener las pulsaciones por debajo de este punto. El máximo de éste intervalo fue fijado en 114,7 *ppm*, ya que este en ese punto el tiempo de reposo entre ondas es igual cero, pero si se supera ese valor las ondas se superpondrían unas a otras.

En la imagen 5.3 se muestra la forma de onda de ECG visualizada a través del sistema de adquisición de señales, a una frecuencia cardíaca de 60 *ppm*. Se puede distinguir perfectamente cómo se produce cada una de las ondas del ECG, observándose primeramente la onda P, precedida por el complejo QRS, la onda T y un periodo de reposo.

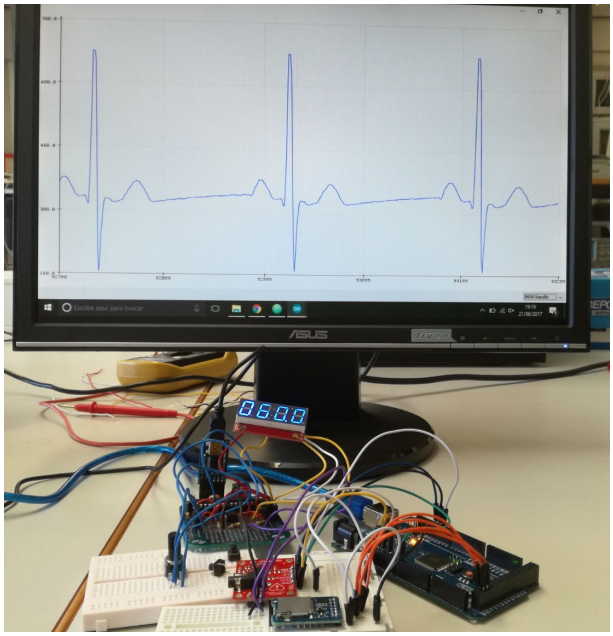


Figura 5.3: Simulación de ECG a una frecuencia cardíaca de 60ppm

Si nos fijamos con detenimiento, la imagen 5.4 es igual a la imagen 5.3, pero hemos aumentado la frecuencia cardíaca hasta nuestro máximo permitido de 114,7 *ppm*, provocando así que la representación de las formas de onda del ECG estén lo mas juntas posibles, teniendo un tiempo de reposo entre ondas igual a cero.

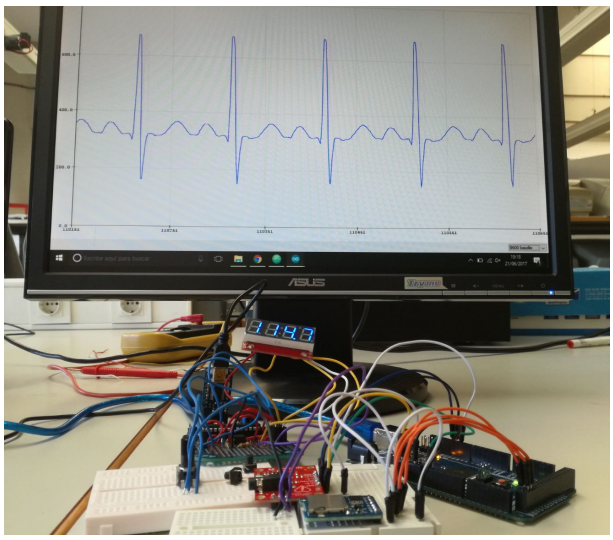


Figura 5.4: Simulación de ECG a una frecuencia cardíaca de 114,7 *ppm*

Ahora tomaremos distintas muestras a distintas frecuencias para poder observar como varía el tiempo de reposo de una a otra, siendo el tiempo de reposo menor cuanto mayor sea la frecuencia cardíaca y viceversa. Para apreciar detalladamente como varía el periodo de reposo, a continuación en la figura 5.5 se muestran 4 imágenes representando la forma de onda del ECG para distintas frecuencias, de 60, 75, 100 y 114,7 *ppm*.

En la imagen 5.5, notamos que al aumentar la frecuencia, el periodo de reposo entre

ondas se va acortando hasta llegar a cero. Sin embargo, la forma de onda del ECG sigue siendo la misma para los distintos valores de frecuencias.

(a) ECG a 60ppm

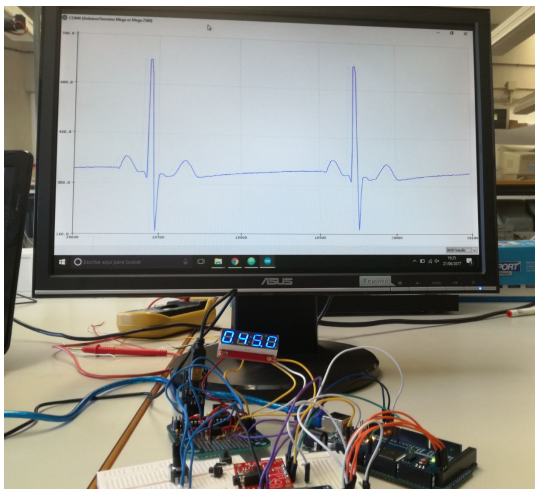
(b) ECG a 75ppm

(c) ECG a 100ppm

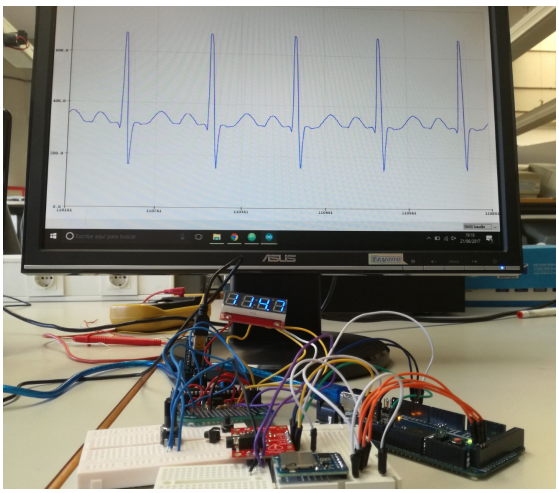
(d) ECG a 114.7ppm

Figura 5.5: Representación de la forma de onda de ECG a distintas frecuencias

Además, podríamos variar el intervalo de frecuencias para estudiar distintas patologías del ECG, como por ejemplo una simple bradicardia, que supone bajar las pulsaciones mínimas a $45 - 55$ *ppm* o simular una taquicardia, escogiendo una frecuencia mayor a las 100 *ppm*, como hemos hecho anteriormente.



(a) Bradicardia



(b) Taquicardia

Figura 5.6: Representación de una bradicardia y una taquicardia

Capítulo 6

Conclusiones

Finalmente en el trabajo expuesto se han conseguido los objetivos que se propusieron en un principio. Se ha logrado diseñar y montar un simulador de ECG, el cual es capaz de generar una forma de onda con características similares a la actividad cardíaca de un corazón normal. Además disponemos de un potenciómetro que nos permite variar su frecuencia en un determinado intervalo de pulsaciones mínimas y máximas, consiguiendo dos aspectos importantes de nuestro proyecto:

- Podemos variar el tiempo de reposo entre ondas.
- Podemos variar la frecuencia cardíaca que se muestra a través de una pantalla numérica de 4 dígitos.

El montaje de este trabajo se ha realizado sobre la protoboard de una placa de Adafruit Menta, utilizando materiales y componentes de bajo costo y fácilmente accesibles para la mayoría de personas que estén interesadas en montar su propio simulador de ECG. Ésta en una de las grandes ventajas de nuestro proyecto frente a otros simuladores de ECG disponibles en el mercado, cuyo precio ronda los cientos de euros, por lo que no todas las personas podrían permitírselo.

Por los motivos expuestos, podríamos afirmar que este dispositivo sería un ejemplar idóneo para uso académico, a pesar de no ser posible en su estado actual su venta comercial por diversos motivos de certificaciones y validaciones legales. Éste favorecería el aprendizaje de los estudiantes de medicina, y supondría una mejora en la comprensión del ECG, permitiendo su estudio mediante la observación directa de la onda cardíaca y sus posibles variaciones en frecuencia. Además, podría ser un dispositivo perfecto para realizar las pertinentes calibraciones de los monitores de hospitales que utilicen ECG. De esta forma, todos los centro médicos podrían asegurar un adecuado mantenimiento de sus aparatos por un precio más que asequible.

El diseño de nuestro trabajo ha sido implementado, de manera que, podríamos simular cualquier otra forma de onda conocida, siguiendo el patrón de trabajo que hemos realizado. Resulta muy interesante la idea de poder simular cualquier forma de onda procedente del cuerpo humano, como por ejemplo un Electromiograma (EMG) o Electroencefalograma (EEG). Para ello, tendríamos que saber previamente las características de la forma de onda que queremos simular, estudiando su patrón seguido y comportamientos en cada uno de los casos. En nuestro trabajo, decidimos la simulación de la actividad eléctrica del corazón porque consideramos que se trata de la bioseñal más fácil de reconocer, de las

producidas por el cuerpo humano. Se distinguen cada una de las ondas que la componen y además disponemos de infinidad de información sobre la amplitud y duración de cada una de sus partes.

Conclusiones personales

El desarrollo de este simulador nos ha llevado a estudiar muchas ciencias distintas, pero análogamente necesarias. A lo largo de estos cuatro años de estudio, hemos podido comprobar que la ingeniería de la salud es muy multidisciplinar, desde el estudio de la división celular a un protocolo de comunicación, por ejemplo. Es por ello por lo que este trabajo reúne en sí, el producto de muchas ciencias como la programación, comunicaciones, fisiología, electrónica. En otras palabras, se ha trabajado con programas como Eagle (desarrollo electrónico), Engauge Digitizer (digitalización de onda) y lenguajes como Phyton (para crear una C-matríz) y C (en el desarrollo del código Arduino).

Podemos concluir que es una propuesta con mucho futuro social. Actualmente, vivimos una época donde la población mayor, propensa a sufrir problemas cardiovasculares, aumenta exponencialmente. Cada vez, son más dispares las diferencias sociales entre países desarrollados y subdesarrollados, y por supuesto, económicamente lo son aún más. Uno de los objetivos actuales de la ingeniería biomédica es hacer llegar a toda la población global, recursos que mejoren la calidad de vida de estas personas en vías de exclusión y pobreza. Una de estas medidas podría ser la formación de personal sanitario, aportándoles cursos y material necesario para que vayan adquiriendo profesionalidad y conocimientos. Para ello, muchas compañías se han puesto manos a la obra con diseños de prótesis eficientes de bajo coste, campamentos con una alta resistencia y fácil montaje, etc. Como no podría ser menos, nosotros jugamos un papel muy importante. Es bien es sabido, que la ingeniería de la salud aúna a la mayoría de las ciencias con el objeto de que converjan en único punto, que no debe ser otro que la ayuda. Es por ello, por lo que este simulador tiene un papel muy importante como recurso didáctico. El poder simular una onda ECG, hará que se puedan formar a muchísimas personas con un simple aparato que ronda entre 60 y 100 euros.

En cuanto a los resultados, se ha obtenido lo que se quería y es por ello, por lo que personalmente estoy muy satisfecho. Si bien, me gustaría destacar, que no ha sido fácil trabajar con Adafruit Menta, por lo que sería recomendable para líneas futuras, hacer la conversión a Arduino, ya que es una placa mucho más conocida, estudiada y trabajada. También, podemos destacar que, económicamente, sería mucho más rentable trabajar con placas basadas en Arduino, ya que el precio se reduciría más del 50 %.

Bibliografía

Álvarez, F. V., Banegas, J. B., Campos, J. d. M. D., and Artalejo, F. R. (2003). Las enfermedades cardiovasculares y sus factores de riesgo en españa: hechos y cifras. *Informe Sea*, pages 43–52.

Banegas, J. R., Villar, F., Graciani, A., and Rodríguez-Artalejo, F. (2006). Epidemiología de las enfermedades cardiovasculares en españa. *Revista Española de Cardiología Suplementos*, 6(7):3G–12G.

Banzi, M. and Shiloh, M. (2014). *Getting Started with Arduino: The Open Source Electronics Prototyping Platform*. Maker Media, Inc.

de Cardiología, I. S. E. (2009). Fuente: Instituto nacional de estadística. defunciones según la causa de muerte 2007. madrid: Instituto nacional de estadística; 2009 [citado oct 2009].

Ferrante, D., Fitzpatrick, C., Dobo, M. G., Kinfu, Y., Fat, D. M., Mathers, C., Cibulskis, R., Fernandez-Vegas, K., Say, L., Bermedo, M. C. S., et al. (2008). Catalogación por la biblioteca de la oms.

Flórez, J. (2014). *Farmacología humana*. Elsevier.

Hamm, C. W. and Willems, S. (2010). *El electrocardiograma: su interpretación práctica; 32 cuadros*. Ed. Médica Panamericana.

López, E. (2011). Ingeniería en microcontroladores/protocolo spi (serial peripherical interface).

MCP4921, M. (2007). 4922 12-bit d/a converter datasheet.

Menta, A. (2017). Oficial web page adafruit menta.

Mulroney, S. E. and Myers, A. K. (2016). *Netter. Fundamentos de fisiología+ StudentConsult*. Elsevier España.

Sparkfun (2017). Spi.

Tortora, G. and Derrickson, B. (2007). *Principios de anatomía y fisiología*. Panamericana.

Tucci, A. (2007). *Instrumentacin Biomdica*. Lulu. com.

Viñoles, F. J. C., de Azevedo Guaura, R., Caporale, M. A., Montaño, P. R. P., and Guaura, R. (2015). Simulador de electrocardiogramas. *Venezolana de Medicina Interna*, 31(2):75–81.

Capítulo 7

Anexos

7.1. Código Arduino

En esta sección se incluye el código Arduino del programa. Éste consta de una estructura modular, que se organizará por tareas. Para cada tarea se generará un archivo .h, que se utiliza para la declaración de variables y métodos. Asimismo, tendremos un archivo .cpp, donde se implementará la función que llevará a cabo la tarea. Finalmente, en el programa principal de Arduino se ejecutarán todas las tareas. Se harán comentarios del código, representados por el color verde.

ReadPot()

```
#ifndef __ReadPot_h__
#define __ReadPot_h__

#include "Arduino.h"
#include "TareaPeriodica.h"
#include "SignalSamples.h"

class ReadPot: public TareaPeriodica
{
public:
  ReadPot(byte pin,unsigned long per);
  unsigned int  getppm();
protected:
  void tarea();

private:
  unsigned long ValoresPulsacion[32];
  byte Indice;
  const static int ppmMin = 300;  //Pulsaciones minimas
  int ppmMax;                     //Pulsaciones maximas
  unsigned int ppm;               //Numero de pulsaciones por minuto
  SignalSamples sigSamp;          //Variable de la clase SignalSamples


  void Display7Seg_Send(unsigned int  RitmoCardiaco);
};
#endif
```

Listing 7.1: ReadPot.h

```
#include "Arduino.h"
#include "ReadPot.h"
#include "SPI.h"    //Libreria SPI para comunicarnos con el Display

ReadPot::ReadPot(byte p,unsigned long per)
: TareaPeriodica( per)
{
  for (int i=0; i<32; i++){ ValoresPulsacion[i]= 0;}
  Indice=0;
  ppmMax = (60.0 / ((float)sigSamp.getNumOfSamples() * 0.001)) *10;
}

  void ReadPot::tarea ()
  {
    int Valor = analogRead(0); //Lectura del potenciometro
    unsigned long MediaPulsacion = 0;

   //Realizamos un mapeo de los valores recogidos por el potenciometro,
    entre ppmMin y ppmMax

    ppm = map(Valor, 0, 1023, ppmMin, ppmMax);

    //Creamos un filtro para evitar las oscilaciones en el display

    ValoresPulsacion[Indice++] = ppm;
    if (Indice == 32) {
        Indice = 0;
    }
    MediaPulsacion = 0;

    for (int  i = 0;  i < 32; i++) {
        MediaPulsacion += ValoresPulsacion[i];
    }
    MediaPulsacion >>= 5;

    //Le mandamos el valor de las pulsaciones para mostrarlo por pantalla

    Display7Seg_Send(MediaPulsacion);

  }

    //Metodo que devuelve las pulsaciones por minuto que recoge el
   potenciometro

    unsigned int ReadPot::getppm()
    {
         return ppm;
    }

    //Funcion de visualizacion del Display

    void  ReadPot:: Display7Seg_Send(unsigned int  RitmoCardiaco)
    {
        uint8_t       digito1, digito2, digito3, digito4;
        unsigned int  value;
```

```
57      //Escribimos los 4 digitos del display, colocando cada uno en su
        posicion
58
59          value = RitmoCardiaco;
60          digito1 = value / 1000;
61          digito2 = (value %10) / 100;
62          digito3 = (value %100) / 10;
63          digito4 =  (value %1000);
64
65          digitalWrite(9, LOW);     // Seleccion del Display
66          SPI.transfer(0x76);       // Reseteo del display
67          SPI.transfer(0x7A);       // Seleccion de brillo
68          SPI.transfer(200);        // Brillo entre [0-255]
69          SPI.transfer(digito1);    // Envio del primer digito (decimal)
70          SPI.transfer(digito2);    // Envio del segundo digito
71          SPI.transfer(digito3);    // Envio del tercer digito
72          SPI.transfer(digito4);    // Envio del cuarto digito
73          SPI.transfer(0x77);       // Seleccion de decimales
74          SPI.transfer(0x04);       // Posicion del punto decimal
75          digitalWrite(9, HIGH);    // Fin del envio
76      }
```

Listing 7.2: ReadPot.cpp

SignalSender()

```
#ifndef __SignalSender_h__
#define __SignalSender_h__

#include "Arduino.h"
#include "TareaPeriodica.h"
#include "ReadPot.h"
#include "SignalSamples.h"


class SignalSender: public TareaPeriodica
{
public:
    SignalSender(unsigned long ms, unsigned long per,ReadPot &Rp);
    void Estado_ECG();
    void DTOA_Send(unsigned short DtoAValue);

protected:
    void tarea();

private:
    ReadPot & rdpt;
    enum EnEstado  {
        PQRS,              // Estado activo de la forma de onda PQRS
        REPOSO};           // Estado de reposo

    EnEstado Estado;
    const float millisMin= 60000.0;
    unsigned int contPPQRS = 0;
    unsigned int contReposo = 0;
    unsigned int contDisplay = 0;
    unsigned long Treposo;
    SignalSamples  sigSamp;

    };
    #endif
```

Listing 7.3: SignalSender.h

```
#include "Arduino.h"
#include "SignalSender.h"
#include "SPI.h"        //Libreria SPI para comunicarnos con el MCP4921

SignalSender::SignalSender(unsigned long ms, unsigned long per, ReadPot &
    Rp)
: TareaPeriodica(per), rdpt(Rp)
{

  contPPQRS = 0;
  contReposo = 0;
  contDisplay = 0;

  Estado = PQRS; //Estado de la onda activa

}

//Esta tarea se ejecuta de forma periodica

void SignalSender::tarea(){

  Estado_ECG();

}

//Declaracion de la forma de onda de ECG

void SignalSender::Estado_ECG()
{
  float Pulsacionespm = (float)rdpt.getppm()/10;
  unsigned long Treposo;
  Treposo= (unsigned int)(millisMin / Pulsacionespm) - (float)sigSamp.
    getNumOfSamples();
  const int SampleOne=0;

  switch (Estado) {

    case PQRS:           //Estado: onda activa

//Envio de las muestras de la onda activa al MPC4921
    DTOA_Send((sigSamp.getSample(contPPQRS))*(-1));

//Contador para el numero de muestras que contiene la onda activa
     contPPQRS++;

// Cuando el contador de muestras es igual o mayor al numero de muestras de
    la onda activa, cambiamos al estado de reposo

    if (contPPQRS >= sigSamp.getNumOfSamples()){


      contPPQRS = 0;      //Ponemos el contador a cero
      Estado = REPOSO;
    }

    break;

    case REPOSO:         //Estado: reposo
```

```
56
57
58 //Contador para el numero de muestras que contiene la onda en estado de
       reposo
59     contReposo++;
60
61 // Cuando el contador de muestras es igual o mayor al numero de muestras de
       la onda en estado de reposo, cambiamos al estado de la onda activa
62
63     if (contReposo >= Treposo) {
64       contReposo = 0;
65       Estado = PQRS;
66     }
67     break;
68
69     default:
70     break;
71   }
72 }
73
74 //
75 void  SignalSender::DTOA_Send(unsigned short DtoAValue) {
76
77   byte Data = 0;
78
79 // Seleccionamos el convertidor a (low) para comenzar a trabajar
80
81   digitalWrite(10, 0);
82
83 //Enviamos la primera trama de 8 bits (4 bits de configuracion y 4 bits de
       datos)
84
85   Data = highByte(DtoAValue);
86   Data = 0b00001111 & Data;
87   Data = 0b00110000 | Data;
88   SPI.transfer(Data);
89   Serial.println(Data);
90
91 //Enviamos la segunda trama de 8 bits de datos
92   Data = lowByte(DtoAValue);
93   SPI.transfer(Data);
94   Serial.println(Data);
95
96 //Trabajo del convertidor realizado
97
98   digitalWrite(10, 1); //Seleccionamos el convertidor  a (high)
99 }
```

Listing 7.4: SignalSender.cpp

TareaPeriodica()

```
#ifndef TareaPeriodica_h
#define TareaPeriodica_h

#include "Arduino.h"

  class TareaPeriodica
  {
    public:
      TareaPeriodica(unsigned long per);
      void loop (unsigned long currTimeMs);

    protected:
      virtual void tarea () = 0;
      unsigned long _periodMs;
    private:
      unsigned long _lastMs;

  };
#endif
```

Listing 7.5: TareaPeriodica.h

```

#include "Arduino.h"
#include "TareaPeriodica.h"

  TareaPeriodica::TareaPeriodica(unsigned long per)
  :  _periodMs(per)
  {

  }

  void TareaPeriodica::loop (unsigned long currTimeMs)
  {
    if ( currTimeMs - _lastMs >= _periodMs ) {
      _lastMs +=_periodMs ;
      tarea();
    }

}
```

Listing 7.6: TareaPeriodica.cpp

SignalSamples()

```
1
2 #ifndef __SignalSamples_h__
3 #define __SignalSamples_h__
4 #include "Arduino.h"
5
6 class SignalSamples{
7 public:
8     SignalSamples();
9     int getNumOfSamples();
10    short getSample(int i);
11 private:
12     int NMuestras;
13
14 };
15  #endif
```

Listing 7.7: SignalSamples.h

```
1 #include "Arduino.h"
2 #include "SignalSamples.h"
3
4 //Declaramos la C-matriz de muestras de la forma de onda del ECG
5
6 const static short y_data[]= {
7   940, 945, 950, 956, 961, 966, 971, 977, 983, 990,
8   996, 1003, 1011, 1018, 1026, 1033, 1040, 1048, 1059, 1069,
9   1079, 1089, 1099, 1107, 1113, 1120, 1126, 1132, 1139, 1145,
10  1151, 1158, 1164, 1170, 1177, 1183, 1189, 1196, 1202, 1208,
11  1215, 1222, 1230, 1238, 1246, 1252, 1253, 1254, 1255, 1256,
12  1257, 1258, 1258, 1259, 1259, 1259, 1259, 1259, 1259, 1259,
13  1259, 1259, 1258, 1258, 1257, 1256, 1255, 1254, 1253, 1253,
14  1252, 1248, 1241, 1235, 1228, 1222, 1215, 1209, 1203, 1198,
15  1193, 1188, 1183, 1178, 1173, 1169, 1164, 1159, 1153, 1148,
16  1142, 1136, 1130, 1125, 1119, 1113, 1107, 1102, 1094, 1086,
17  1078, 1070, 1062, 1055, 1047, 1040, 1032, 1025, 1017, 1012,
18  1007, 1002, 997, 992, 987, 982, 977, 972, 967, 964,
19  963, 961, 959, 957, 955, 953, 952, 952, 952, 952,
20  952, 952, 952, 952, 952, 952, 952, 952, 952, 951,
21  951, 951, 951, 951, 951, 951, 951, 951, 951, 951,
22  951, 951, 953, 957, 962, 965, 963, 960, 957, 955,
23  952, 945, 935, 925, 915, 904, 894, 886, 879, 871,
24  863, 855, 840, 817, 793, 769, 741, 713, 686, 658,
25  991, 1307, 1332, 1358, 1401, 1493, 1591, 1670, 1766, 1910,
26  1954, 2187, 2532, 2548, 2782, 2923, 3247, 3263, 3909, 3899,
27  3746, 4021, 4080, 4058, 4037, 3974, 3859, 3870, 3533, 3468,
28  3404, 3329, 3251, 3095, 2715, 2625, 2513, 2486, 2198, 1690,
29  1268, 832, 572, 306, 346, 72, 67, 62, 57, 52,
30  47, 115, 226, 259, 272, 288, 313, 338, 363, 467,
31  532, 561, 577, 594, 618, 673, 687, 702, 717, 824,
32  922, 941, 960, 978, 988, 997, 1007, 1016, 1022, 1025,
33  1028, 1031, 1033, 1036, 1039, 1042, 1045, 1047, 1048, 1048,
34  1047, 1046, 1046, 1045, 1044, 1044, 1043, 1042, 1042, 1040,
35  1037, 1034, 1032, 1029, 1026, 1023, 1020, 1017, 1014, 1011,
36  1008, 1005, 1001, 998, 994, 990, 987, 983, 979, 977,
37  977, 977, 977, 977, 977, 977, 977, 977, 977, 977,
38  977, 977, 977, 977, 977, 977, 977, 977, 977, 977,
39  977, 977, 977, 977, 978, 979, 979, 980, 981, 981,
```

```
  982, 983, 984, 984, 984, 984, 984, 984, 983, 983,
  984, 986, 988, 990, 992, 994, 996, 999, 1001, 1003,
  1005, 1012, 1018, 1024, 1031, 1037, 1043, 1050, 1055, 1059,
  1063, 1067, 1071, 1074, 1078, 1082, 1086, 1090, 1093, 1098,
  1107, 1115, 1124, 1133, 1142, 1149, 1154, 1159, 1165, 1170,
  1175, 1180, 1185, 1191, 1196, 1201, 1206, 1211, 1216, 1221,
  1226, 1231, 1236, 1241, 1246, 1251, 1256, 1261, 1266, 1272,
  1277, 1283, 1289, 1294, 1300, 1305, 1310, 1313, 1315, 1318,
  1320, 1323, 1325, 1328, 1330, 1333, 1335, 1338, 1338, 1338,
  1338, 1338, 1338, 1338, 1338, 1338, 1338, 1338, 1338, 1334,
  1330, 1326, 1323, 1319, 1315, 1311, 1307, 1303, 1299, 1296,
  1292, 1288, 1285, 1281, 1278, 1274, 1271, 1267, 1263, 1260,
  1255, 1250, 1246, 1241, 1236, 1230, 1224, 1219, 1213, 1208,
  1202, 1196, 1191, 1184, 1177, 1170, 1162, 1155, 1147, 1140,
  1133, 1125, 1117, 1109, 1101, 1093, 1088, 1082, 1076, 1071,
  1065, 1059, 1053, 1048, 1042, 1037, 1034, 1032, 1029, 1026,
  1023, 1021, 1018, 1015, 1013, 1010, 1007, 1005, 1002, 1001,
  1000, 999, 998, 997, 997, 996, 995, 995, 995, 995,
  995, 995, 995, 995, 995, 995, 995, 995, 995, 994,
  994, 994, 994};

SignalSamples::SignalSamples()
{
  NMuestras = sizeof(y_data) / sizeof(short);
}
int SignalSamples::getNumOfSamples()
{
  return NMuestras;
}
short SignalSamples::getSample(int i)
{
  return y_data[i];
}
```

Listing 7.8: SignalSamples.cpp

Programa principal de Arduino

```
1
2 #include "SPI.h"
3 #include "ReadPot.h"
4 #include "SignalSender.h"
5
6 unsigned int SSMPC=10;       // Senal de Seleccion (SS) para el MPC4921
7 unsigned int SSD=9;          // Senal de Seleccion (SS) para el display
8
9 ReadPot rp(0,50);            // Variable del tipo ReadPot
10 SignalSender ss(0,1,rp);     // Variable del tipo SignalSender
11
12 void setup()   {
13
14 // Pines digitales que vamos a utilizar de nuestra placa
15
16 pinMode (13, OUTPUT);        // Senal de reloj (SCLK)
17 pinMode (11, OUTPUT);        // MOSI
18 pinMode (SSMPC, OUTPUT);     // Senal de Seleccion (SS) para el MPC4921
19 pinMode (SSD, OUTPUT);       // Senal de Seleccion (SS) para el display
20
21 // Inicio del estandar SPI
22 SPI.begin();
23 SPI.setDataMode(0);                        //Modo de operacion a low
24 SPI.setClockDivider(SPI_CLOCK_DIV64);     //Frec dividida por 64
25 SPI.setBitOrder(MSBFIRST);                //El primer bit, es el mas
                                                    significativo
26
27 }
28
29 void loop() {
30
31 rp.loop(millis()); //Llamamos a la tarea ReadPot
32 ss.loop(millis()); //Llamamos a la tarea SignalSender
33
34 }
```

Listing 7.9: Programa principal de Arduino