UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Diseño de una autorradio con pantalla táctil mediante Raspberry Pi

Presentado por Rafael Fernández Flores
en Universidad de Burgos — 8 de junio de 2017
Tutor: César Represa Pérez

Resumen

El propósito del trabajo es diseñar un sistema, atendiendo tanto al apartado del hardware como al apartado del software, que implementa las funcionalidades de una autorradio, utilizando una pantalla táctil y teniendo como núcleo principal de procesamiento una Raspberry Pi. El sistema implementa funcionalidades tales como lectura y reproducción de archivos de música, sintonización de emisoras de radio, etc. Se trata de un sistema multimedia táctil destinado a vehículos reales.

El diseño de la interfaz y las aplicaciones deben ser adecuados para su uso en un automóvil.

La idea es realizarlo bajo hardware de bajo coste y con componentes que posteriormente pudieran ser insertados en un automóvil real.

Descriptores

Raspberry Pi, Linux, hardware de bajo coste, QT, Python, Unix, multimedia, radio, audio, pantalla táctil

Abstract

The point of this work is to design a system, considering hardware and software, that implements the functionalities of a car radio, using a touch screen and using as main processor core a Raspberry Pi board. The system implements functionalities such as reading and playing audio files, tunning radio stations, etc. It's a tactile multimedia system focused on vehicles.

The design of the interface and the applications should be suitable for the use in a car.

The idea is to do all over low cost hardware, using components that could be inserted later in a real car.

Keywords

Raspberry Pi, Linux, low cost hardware, QT, Python, Unix, multimedia, radio, audio, touchscreen

Índice general

Índice de figuras

Introducción

1.1. Automóviles, entretenimiento multimedia y hardware de bajo coste

Desde hace varios años, es habitual dotar a los automóviles de ciertos sistemas de entretenimiento. Antiguamente, los automóviles generalmente solo tenían una radio normal, donde solamente podían sintonizar emisoras de radio. El fin perseguido desde entonces ha sido el de amenizar el viaje tanto al conductor como a sus acompañantes.

El sector del automóvil siempre ha ido creciendo tanto en tecnología como en usuarios y automóviles circulando, por lo que estas necesidades de ocio y entretenimiento para los ocupantes del vehículo han ido creciendo, sobretodo de cara a largos trayectos.

Así pues, de estas primitivas radios para coches se han ido sucediendo varias mejoras, como primeramente radios con opción de cintas de casete, o ya, más modernas y actuales, radios con lectores de CD.

Posteriormente a esto, y en un camino de evolución constante del concepto de entretenimiento dentro del automóvil, se sucedieron también autorradios con puertos USB para poder conectar pendrives con canciones para reproducirlas dentro del coche. También se fue mejorando la calidad de los equipos de audio.

Todo esto ha seguido avanzando y evolucionando, y actualmente, tenemos un nuevo concepto que poco a poco se deja ver en automóviles relativamente nuevos, que son las autorradios táctiles.

Figura 1.1: Autorradio táctil para vehículo.

Este tipo de autorradios llevan este concepto del entretenimiento a otro nivel. Se basan en el concepto de tener un sistema multimedia completo dentro del propio automóvil. Hay varios tipos y muchas marcas que comercializan este tipo de autorradios, pero generalmente comparten una serie de rasgos comunes, como son la pantalla táctil como principal medio de interacción y las posibilidades multimedia.

Las posibilidades multimedia que nos ofrecen estos dispositivos son muy variadas, suelen permitir la reproducción de archivos de audio en los formatos más comunes, como mp3 o wav, o la sintonización de emisoras de radio. Según el modelo, sobretodo las gamas más altas, pueden incorporar muchísimas funcionalidades más, como reproducción de vídeo, visor de fotografías, sintonización de TV, sistemas de navegación GPS... Todo integrado en un solo dispositivo.

Por otra parte, estos sistemas suelen ser **bastante caros** económicamente, la cual es una de las principales razones por las que de momento no han proliferado demasiado. Así pues uno de los objetivos de este proyecto es intentar realizar el sistema bajo la premisa del **bajo coste económico**.

Así pues, siguiendo esta premisa, se ha decidido realizar el proyecto sobre la placa **Raspberry Pi**, ya que es una placa de bajo coste, con mucha potencia

para lo que es, y con un gran soporte por detrás, y cuyo software es open source.

Figura 1.2: Implantación de una autorradio en un vehículo real.

1.2. Estructura de la memoria

Esta memoria ha sido estructurada de la siguiente manera:

- **Introducción:** Breve introducción donde se plantea un poco cuáles son las bases del proyecto, de dónde surge, y trata muy por encima qué objetivos básicos pretende conseguir y de qué manera.
- **Objetivos del proyecto:** En esta sección se pretende hablar más detenida y pausadamente de los objetivos **concretos** del proyecto, haciendo un análisis más exhaustivo de los **requisitos hardware y software que han de cumplirse**.
- **Conceptos teóricos:** Aquí se explicarán todos los conceptos teóricos que se han utilizado en el desarrollo de este TFG, y que son necesarios conocer para comprender algunos aspectos importantes del proyecto.
- **Técnicas y herramientas:** Sección dirigida a tratar las diferentes técnicas de planificación y herramientas de trabajo utilizadas en este proyecto.
- **Aspectos relevantes del desarrollo del proyecto:** Aquí se va a detallar la evolución del proyecto, desde sus inicios hasta el resultado final, explicando detalladamente todas las fases del mismo.
- **Trabajos relacionados:** En esta sección se hablará sobre otros trabajos similares que hay por la red sobre sistemas de autorradio, ya sean o no comerciales, siempre que tengan relación con este proyecto, bien por utilizar una Raspberry o bien por ser proyectos libres o de código abierto.
- **Conclusiones y líneas de trabajo futuras:** Sección para tratar las conclusiones de la realización del trabajo y para plantear cómo podría ser ampliado este proyecto.

1.3. Materiales entregados

En los DVD's presentados en la secretaría de la EPS nos encontramos:

- **Carpeta TouchCarPi:** Contiene el proyecto de PyCharm junto al código y las librerías fake para abrirlo directamente con PyCharm y ejecutarlo sobre Windows. Se recomienda, no obstante, consultar los *anexos*, donde se detalla el proceso en profundidad.
- **Memoria.pdf:** Esta misma memoria en formato PDF.
- **Anexos.pdf:** Contiene los anexos en formato PDF.
- **TouchCarPiVideo.mp4:** Archivo MP4 con un vídeo del sistema en funcionamiento.

Objetivos del proyecto

2.1. Objetivos generales

El objetivo de este proyecto es, como su título indica, el de **diseñar una autorradio con pantalla táctil mediante Raspberry Pi**. No obstante este es un enunciado muy genérico, por lo que vamos a pasar a detallar con exactitud todos los objetivos.

El objetivo principal de este proyecto va a ser el diseño de un sistema **similar** al mostrado en la **figura 1.1**, **teniendo en cuenta tanto la parte hardware como la parte software**, y pensando en todo momento en su futura hipotética implantación real en un automóvil.

Este proyecto se compone de dos partes claramente diferenciadas, la parte referente al hardware y la comunicación con el mismo, y la parte referente al software, que sería nuestro programa que nos daría la función de autorradio.

2.2. Objetivos con respecto al hardware

En cuanto a los objetivos con respecto al hardware, tenemos los siguientes:

- El sistema debe de diseñarse sobre hardware de bajo coste (Raspberry Pi).
- El sistema debe de controlarse a través de una pantalla táctil.
- El sistema debe de comunicarse con un módulo hardware externo de radio para sintonizar emisoras de radio.
- Los componentes hardware del sistema tienen que ser elegidos cuidadosamente a propósito para que sea viable su implantación en un automóvil real.

- El sistema debe tener una salida de jack que pueda ser conectado a un amplificador de automóvil.

2.3. Objetivos con respecto al software

En cuanto a los objetivos con respecto al software, tenemos los siguientes:

- La interfaz gráfica del software debe de estar diseñada para ser utilizada a través de una pantalla táctil.
- El software debe de permitir la reproducción de archivos de audio.
- El software debe de establecer la conexión lógica a bajo nivel con el módulo de radio.
- El software debe de permitir escuchar la radio y sintonizar diferentes emisoras.
- El software debe de permitir memorizar nuevas emisoras.

Conceptos teóricos

3.1. Hardware libre

Se conoce como hardware libre a los dispositivos y placas hardware cuyas especificaciones y diagramas están al acceso de cualquiera, ya sea gratis o pagando según se de el caso. El concepto del software libre es similar, con pequeñas variaciones, al del hardware libre. Especialmente varían en el concepto de que para la creación de hardware libre hay que hacer una inversión monetaria que no ocurre con el software. [25]

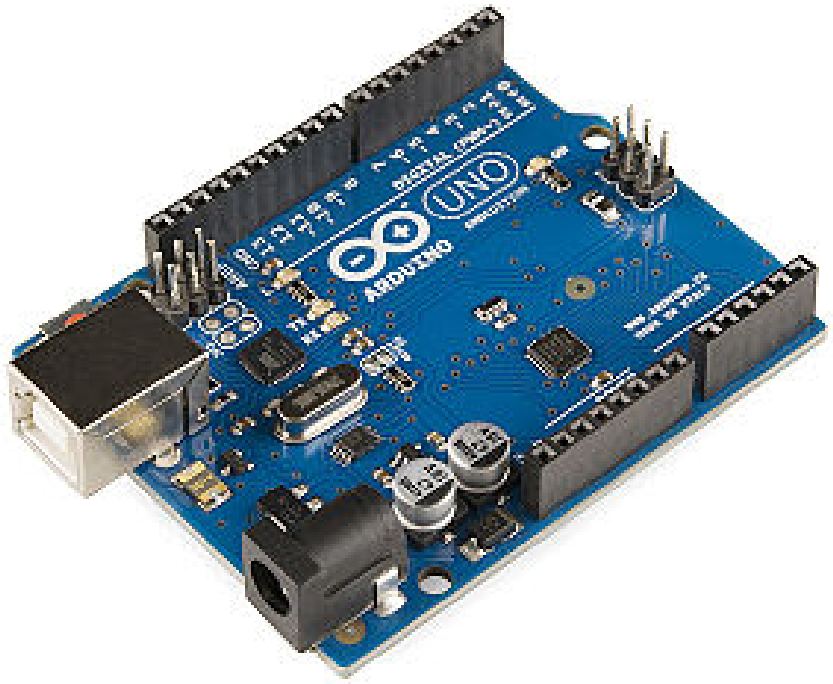


Figura 3.3: La placa arduino es un claro caso y uno de los referentes de hardware libre.

Este concepto tiene una amplia relación con la placa Raspberry Pi, utilizada en este proyecto, aunque esta misma no sea hardware libre al 100 %. Sin embargo, la placa *Arduino* mostrada en la **figura 3.3** sí que se considera hardware libre.

3.2. Raspberry Pi

La Raspberry Pi es una placa de hardware, que vendría a ser un computador del tamaño de una tarjeta de crédito, de bajo coste, el cual ha sido diseñado y producido en Reino Unido por la Fundación Raspberry Pi.

Esta fundación es de carácter benéfico, y su objetivo con esta placa es estimular el aprendizaje de estos temas al proveer una placa potente y de bajo coste.

No se indica expresamente si es hardware libre o con derechos, inicialmente

no entra en la definición de hardware libre, ya que aunque hay partes de su diseño que sí que están disponibles al público, hay otras que dependen de empresas externas a la Fundación Raspberry Pi y que son diseños de hardware cerrado, como por ejemplo el diseño del procesador que utiliza.

No obstante, la plataforma sí permite su uso libre a nivel educativo, particular y comercial.[30]

Figura 3.4: La placa Raspberry Pi es una de las placas de hardware de bajo coste más famosas, reconocidas y en auge. Hay infinidad de proyectos disponibles para ella.

3.3. GPIO

GPIO (General Purpose Input/Output, Entrada/Salida de Propósito General) es un pin o un conjunto de pines que pueden ser programados por el usuario.

Los pines GPIO no tienen ningún propósito predefinido, sirven para proveer de líneas de control adicionales a un chip o circuito integrado. Así pues, no hace falta recurrir a tener que crear circuitos externos adicionales para proporcionar estas líneas extra.[23]

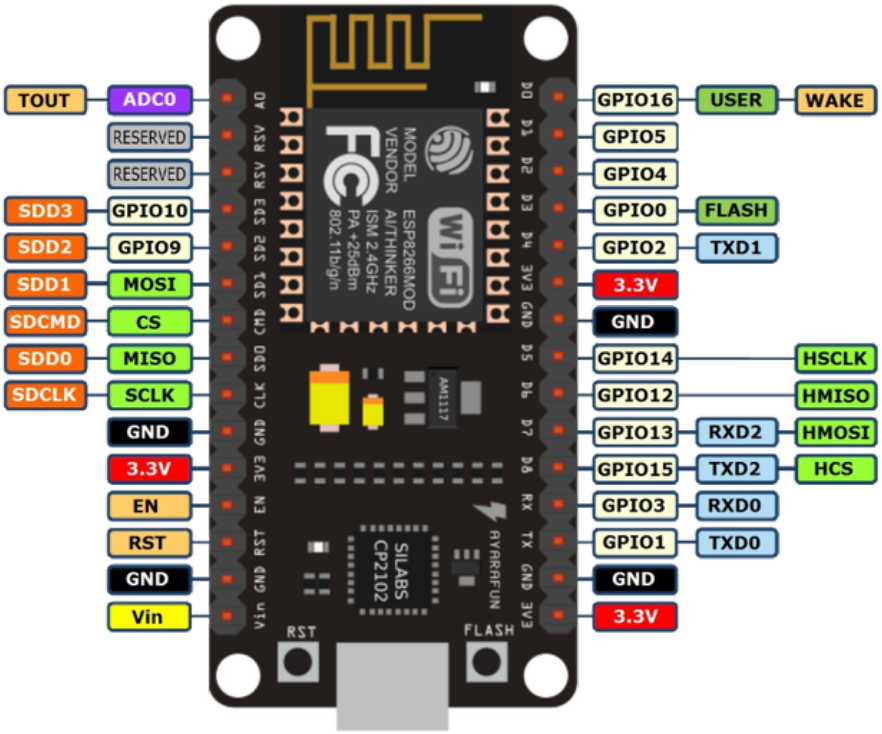


Figura 3.5: Detalle de los puertos GPIO de una mini placa NodeMCU.

3.3.1. Raspberry Pi GPIO

Sobre las Raspberry Pi, los GPIO (General Purpose Input/Output) son una serie de pines o conexiones de E/S (Entrada/Salida) de propósito general, programables por el usuario, a los que se les puede dar diversidad de usos. Estos pines están incluidos en todos los modelos de Raspberry Pi, aunque en función de qué modelo y revisión podría cambiar la funcionalidad de cada pin concreto.

Todos los pines son de tipo "unbuffered", no disponen de buffers de protección, lo cual quiere decir que hay que vigilar valores como voltaje o intensidad de lo que conectamos a estos pines, ya que en caso contrario podríamos dañar la placa.[5]

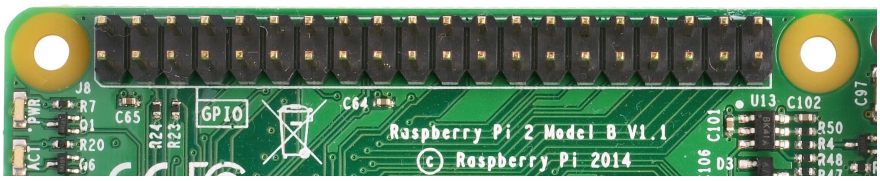


Figura 3.6: Detalle de los puertos GPIO de una Raspberry Pi Modelo 2B.

3.4. I2C

Un circuito interintegrado (I2C, del inglés Inter-Integrated Circuit) es un bus serie de datos desarrollado en 1982 por Philips Semiconductors. Es utilizado para la comunicación entre diferentes partes de un circuito, generalmente para conectar a un controlador ciertos periféricos compatibles con el protocolo.

El sistema inicialmente fue diseñado por Philips para controlar varios chips de sus televisores más fácilmente. No obstante, el protocolo fue adoptado por varios competidores que aseguraban compatibilidad con el sistema I2C de Philips. Hoy en día hay un total de más de mil circuitos integrados diferentes de más de 50 fabricantes compatibles con I2C, según datos de 2014.

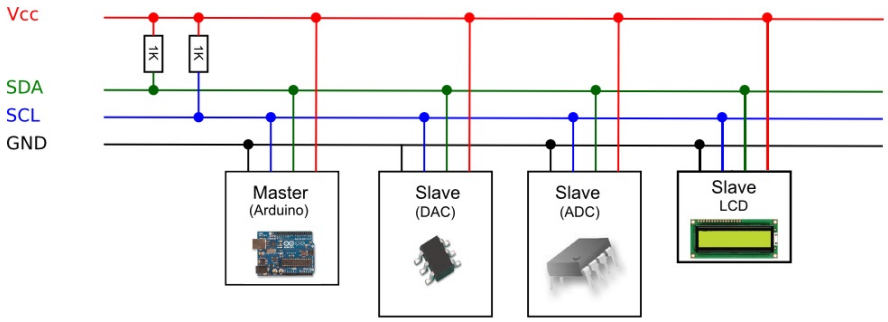


Figura 3.7: Ejemplo del conexionado del bus maestro-esclavo.

El protocolo I2C es un protocolo que contiene un bus maestro-esclavo. El maestro inicia siempre la transferencia de datos y el esclavo responde. Se pueden tener varios maestros, aunque lo común suele ser tener un maestro y varios esclavos, a los que se puede acceder mediante direcciones lógicas.[26]

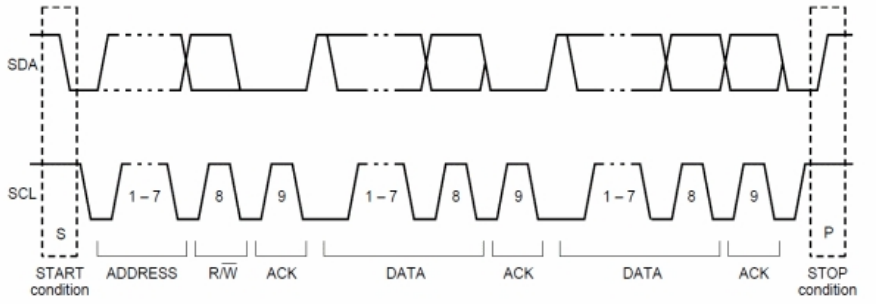


Figura 3.8: Trama de comunicación del protocolo I2C.

3.5. SSH

SSH es un protocolo que permite acceder a máquinas remotas vía consola de comandos. Su nombre viene del acrónimo Secure SHell.

SSH permite manejar por completo un ordenador remotamente, además de permitirnos copiar archivos de forma segura.

SSH ha sido utilizado en este proyecto para enviar archivos ejecutables desde el ordenador donde he desarrollado con Windows hasta la Raspberry Pi con Linux.[31]

Técnicas y herramientas

En este apartado voy a exponer las diferentes herramientas y utilidades, ya sean online o programas instalados que he utilizado para la realización de este proyecto.

4.1. Herramientas de planificación

Aquí englobaremos todas las herramientas que sirven para realizar la planificación del proyecto.

4.1.1. GitHub

GitHub es una plataforma basada en el control de versiones de Git, que implementa además del control de versiones, diferentes utilidades para la planificación de proyectos basándose en metodologías ágiles.

A parte de todo esto, ofrece un sistema de tracking mediante issues para llevar un seguimiento de tareas y bugs del proyecto entre otras cosas. Aunque inicialmente GitHub está pensado para trabajar en equipo con metodologías ágiles (por ejemplo Scrum), también da buen resultado para proyectos individuales, y por eso ha sido elegido.[3]

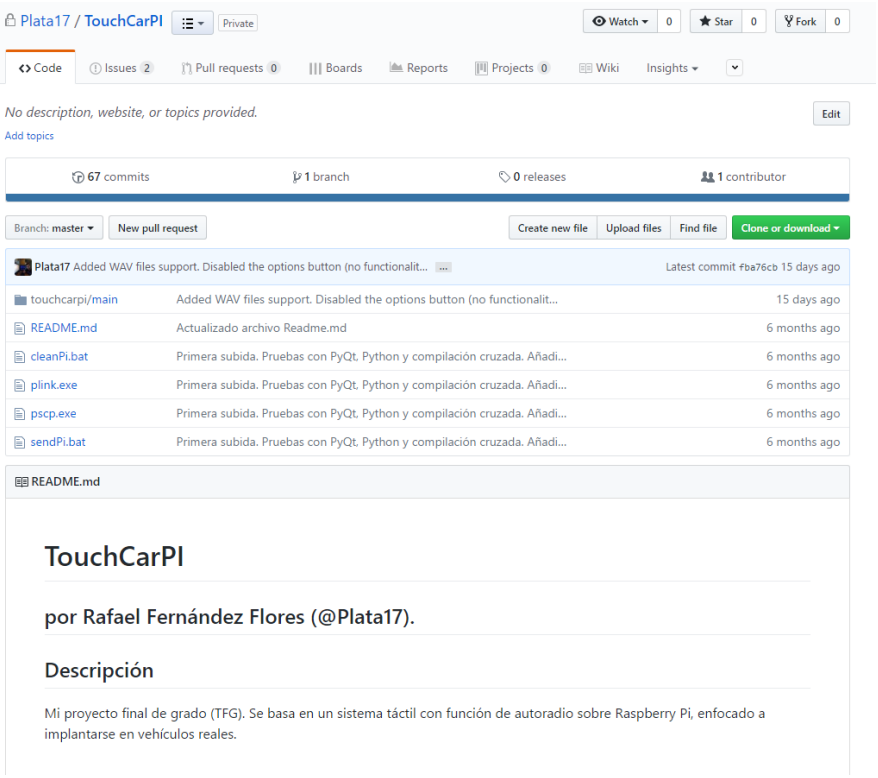


Figura 4.9: Captura del repositorio de este TFG en GitHub.

4.1.2. ZenHub

ZenHub es una extensión para el navegador que añade funcionalidades a GitHub. Principalmente, nos proporciona un tablero al estilo de Trello en nuestro repositorio de GitHub donde podemos organizar todas nuestras issues de GitHub en diferentes cajones para poder trabajar mejor y obtener de un vistazo qué cosas quedan por hacer en el proyecto.

Permite también la representación de varios tipos de gráficos relacionados con métodos ágiles, nos permite crear milestones en nuestro proyecto, asignar story points a las issues en función del tiempo estimado que va a consumir completarlas, junto a muchas funcionalidades más.[33]

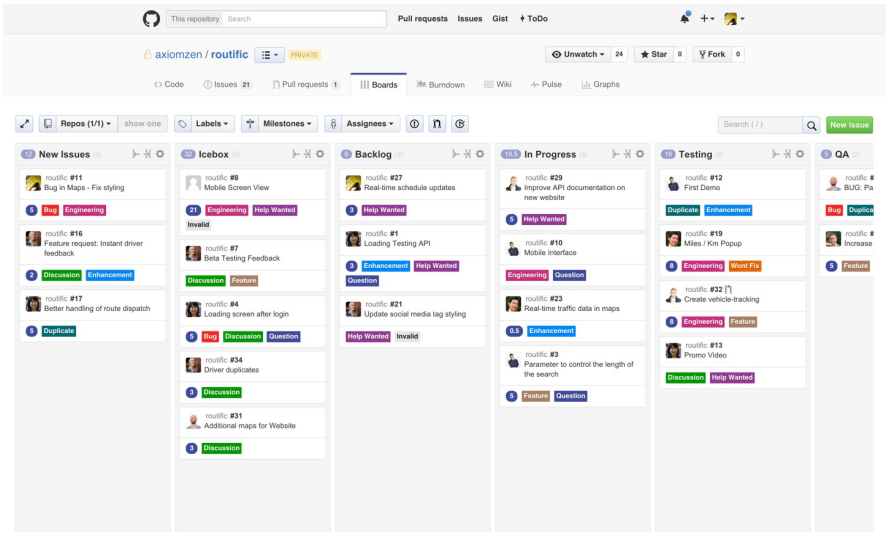


Figura 4.10: Captura del panel tipo Kanban de ZenHub.

4.1.3. TexMaker

TexMaker es un editor de LaTeXgratuito Y multiplataforma disponible para Windows, Linux y Mac.

TexMaker incluye todo el paquete de herramientas necesario para crear documentos en LaTeX, como herramientas para confeccionar el propio documento o herramientas para confeccionar las bibliografías.

Soporta características como el auto completado de comandos y corrector ortográfico. Además exporta el resultado final a PDF, permitiéndote visualizar el PDF creado mientras trabajas a la vez.[20]

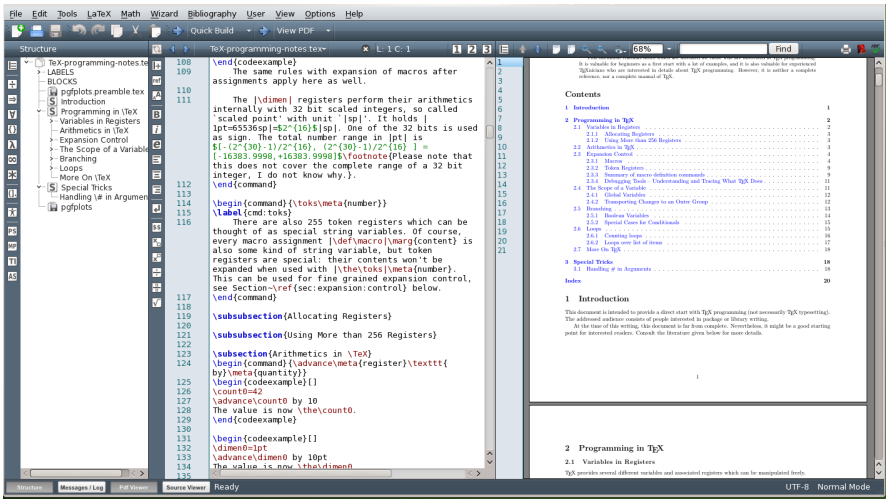


Figura 4.11: Captura del editor TexMaker en funcionamiento.

4.1.4. StarUML

StarUML es un programa que permite la creación de diagramas UML.

Entre las características de este software, las que nos interesan son básicamente la parte de diagramas UML que nos sirven para definir diagramas para programación orientada a objetos, como por ejemplo diagramas de clases. También permite crear otros tipos de diagramas, como diagramas de actividad, de secuencia, etc...

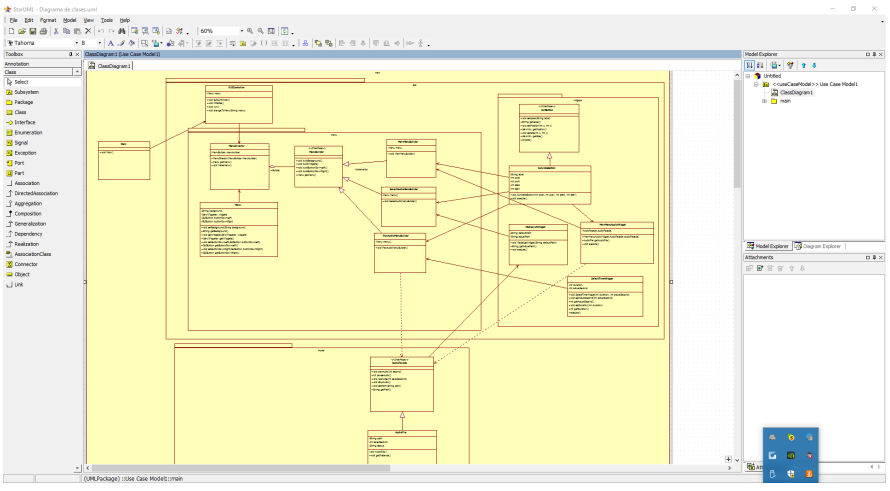


Figura 4.12: Captura del programa StarUML con un diagrama de clases antiguo de este proyecto abierto.

He trabajado con la versión original de StarUML a pesar de que hay versiones más recientes, ya que las últimas versiones son de pago mientras que la original es gratuita. A pesar de que en la versión original la interfaz luce mucho más anticuada, es igual de potente y útil que la última versión.[32]

4.2. Herramientas de desarrollo

Aquí englobaremos todas las herramientas que sirven para realizar o que están involucradas en el desarrollo del proyecto.

4.2.1. Python

Python es un lenguaje de programación, el cual es interpretado sobre una máquina virtual. Su objetivo principal es que el código sea fácilmente entendible.

Para este desarrollo, ha sido utilizada la versión 3 de Python.

Python ha sido elegido por varias de sus características. La primera es que es orientado a objetos, algo elemental para este desarrollo. Otra es que es multiplataforma, lo cual nos va a permitir ejecutar sobre cualquier plataforma que soporte Python la misma aplicación. Por ejemplo, este proyecto se puede ejecutar tanto sobre Windows en PC con arquitectura x86 que es donde se desarrolla como sobre la Raspberry Pi sobre Linux con arquitectura ARM.[28]

4.2.2. PyCharm

PyCharm es un IDE perteneciente a la empresa JetBrains[1], cuyo propósito es desarrollar aplicaciones en Python.

Entre sus características, nos encontramos con que provee análisis de código, utilidades de debug, soporta control de versiones y conectividad con GitHub.

Es multiplataforma, pudiéndolo encontrar para Windows, Linux y MacOS X.[27]

[1]Página oficial de JetBrains: https://www.jetbrains.com/

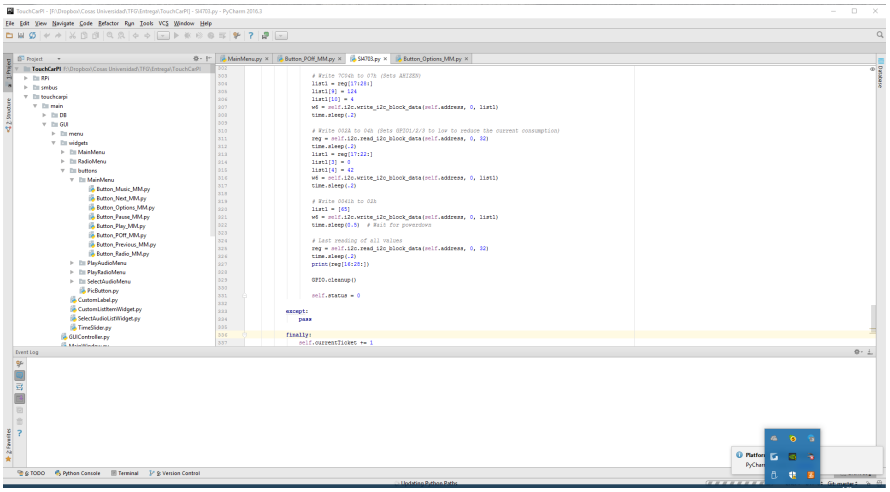


Figura 4.13: Captura del IDE PyCharm con el proyecto abierto.

4.2.3. LibVLC

LibVLC es una librería, propia del famoso reproductor multimedia VLC[2], la cual nos provee funcionalidades para la reproducción de archivos multimedia, tanto archivos de audio como archivos de vídeo.

Oficialmente se provee soporte a esta librería, la cual tiene diferentes implementaciones para diferentes lenguajes, contando con una implementación para Python.[22]

4.2.4. QT y PyQT

PyQT es la implementación oficial sobre Python de la famosa librería gráfica QT[3].

Qt es un framework multiplataforma utilizado para desarrollar programas que necesiten de una interfaz gráfica. Es una librería gráfica de código abierto, y es utilizada en infinidad de aplicaciones.[29]

[2]Página oficial del reproductor VLC: http://www.videolan.org/vlc/
[3]Más información sobre la librería QT: https://www.qt.io/es/

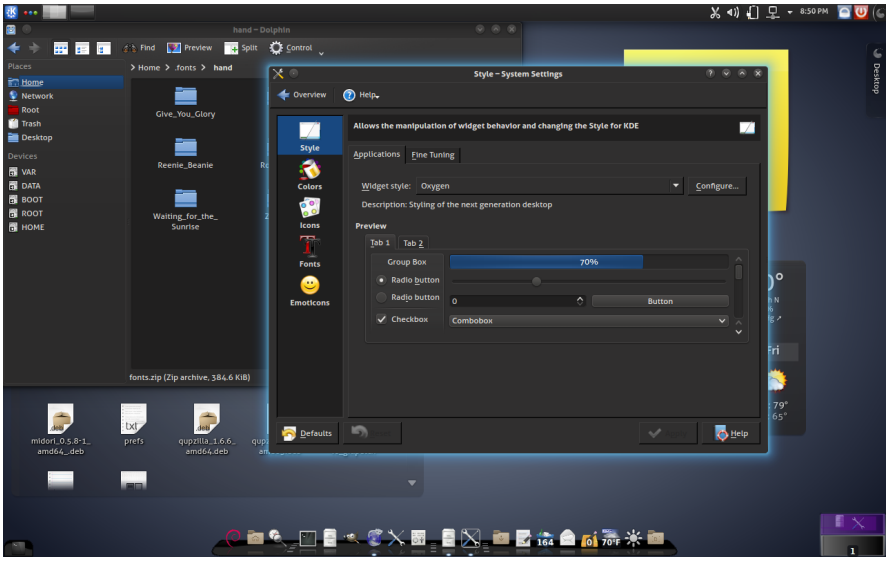


Figura 4.14: Captura del escritorio de Linux KDE, el cual utiliza QT como librería gráfica.

4.2.5. Otras librerías a destacar

Aquí se tratará sobre otra serie de librerías que han sido utilizadas en este proyecto. Estas tienen la misma relevancia o más que las anteriores, pero al ser librerías que han sido utilizadas para detalles muy concretos del proyecto, se pueden tratar todas por encima en un pequeño apartado.

Hay dos librerías utilizadas en el proyecto sobre Python que me gustaría destacar en este apartado:

- **RPi.GPIO:** Esta librería provee control sobre los puertos GPIO de la Raspberry Pi. En este proyecto es de suma importancia, dado que utilizamos y manipulamos a bajo nivel los puertos GPIO para establecer conexión con el hardware de la radio.[9]
- **SMBUS:** La librería SMBUS nos permite realizar la comunicación con hardware externo a través de los pines destinados a I2C en la Raspberry Pi, es utilizada en el proyecto, siendo una pieza clave del mismo.[10]

4.2.6. VNC Viewer

VNC, a modo genérico, es un programa que nos permite controlar un ordenador remoto, viendo su escritorio y pudiendo interaccionar como si fuese nuestro propio ordenador desde un ordenador cliente.

Una de las características más importantes es que es indiferente qué sistema operativo se ejecute en cada una de las máquinas (servidor y cliente), va a permitir compartir el escritorio igualmente.[24]

VNC Viewer[4] es simplemente un cliente de VNC que funciona sobre múltiples plataformas. En mi caso, lo he utilizado como cliente en Windows para controlar el escritorio de la Raspberry Pi.

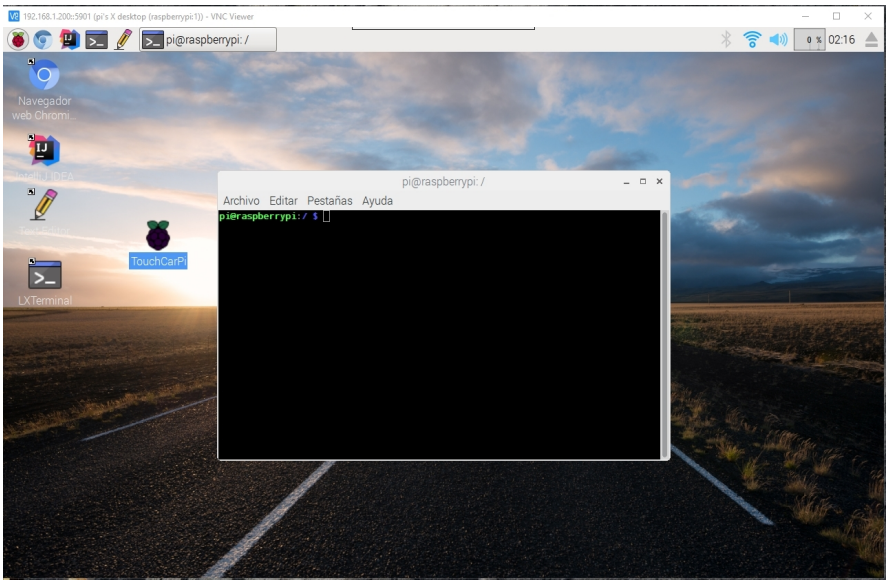


Figura 4.15: Captura del escritorio de mi Raspberry Pi a través de VNC Viewer en mi PC con Windows.

[4]Página oficial de VNC Viewer: https://www.realvnc.com/download/viewer/

Aspectos relevantes del desarrollo del proyecto

5.1. Introducción

En esta sección es donde se recogen realmente los puntos fuertes del desarrollo, y donde se va a tratar en profundidad aspectos del desarrollo.

Este proyecto toca varios palos, es decir, tiene una parte de hardware y un peso en ello muy importante, y otra parte de software con un peso también importante. Se van a separar claramente estas dos partes.

Vamos a empezar por la parte correspondiente al hardware, viendo todo el conexionado y módulos. Posteriormente vamos a ver el software base (Linux) que se ejecuta sobre el hardware, para ya finalmente ver el proceso de desarrollo del software, tanto la funcionalidad en sí como la comunicación con el software a través del hardware.

5.2. Sección de hardware

En esta sección vamos a listar todo el hardware empleado para este proyecto, junto a una breve explicación del mismo.

5.2.1. Raspberry Pi 3

Ya hemos hablado bastante sobre la Raspberry Pi en términos generales a lo largo de este documento, no obstante, a modo de pequeño recordatorio, la Raspberry Pi es una placa de hardware de bajo coste, potente para su tamaño y que puede ejecutar cosas relativamente pesadas, como por ejemplo Linux.

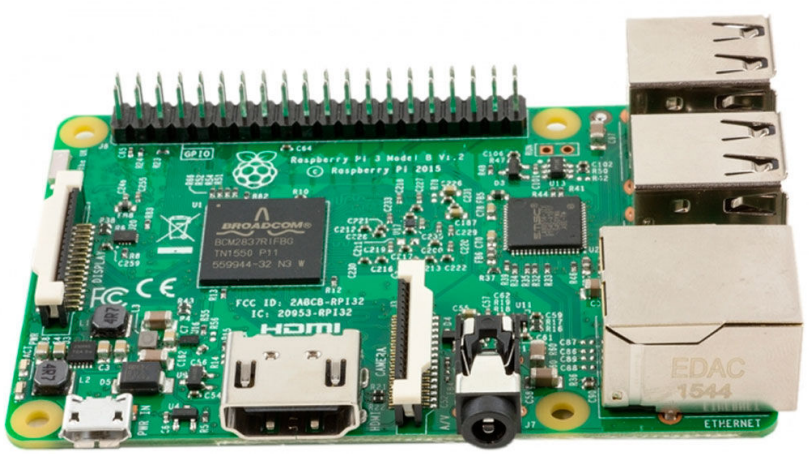


Figura 5.16: Raspberry Pi 3, la placa de hardware utilizada para este proyecto.

De entre todas las versiones, he optado por utilizar la Raspberry Pi 3, la cual es la última versión de esta placa. Las diferencias principales es que esta versión trae WiFi y Bluetooth integrado en la propia placa.

Estas son las características técnicas de la placa:

- Procesador Broadcom BCM2387, 1,2Ghz Quad-core ARM.
- GPU VideoCore IV Dual-core, con soporte OpenGL.
- 1GB RAM LPDDR2.
- Ethernet 10/100
- 802.11 b / g / n WiFi, Bluetooth 4.1
- Salida de vídeo HDMI.
- Salida de audio jack 3,5mm.
- x4 USB 2.0
- Puerto GPIO de 27 pines.
- Conector de cámara (CSI-2).
- Interfaz de conexión de display (DSI).
- Ranura para MicroSD.

Sobre esta placa, con este hardware es sobre la que vamos a mover toda nuestra aplicación, con unos resultados en cuanto a velocidad más que satisfactorios, y a un coste bastante bajo.

5.2.2. Pantalla táctil

Respecto a la pantalla táctil, podría haber utilizado cualquiera, ya que hay un montón de modelos que se pueden adaptar. No obstante, como esto sería un prototipo del sistema en sí, he utilizado la **pantalla oficial para la Raspberry Pi**, dado que me parecía la más adecuada para poder llevar el prototipo de una manera segura a cualquier parte. Es probable que a la hora de implantarlo en un automóvil escogiese otro tipo de pantalla, pero esto es lo de menos porque no supone grandes diferencias a la hora de su uso.

Figura 5.17: Pantalla táctil oficial para la Raspberry Pi 3 utilizada para este prototipo.

Esta pantalla táctil, junto a la carcasa, permiten la perfecta integración de la pantalla y la Raspberry en uno solo, haciendo el sistema en sí mucho más transportable para demostraciones, y dejando dentro de la carcasa espacio para posteriormente añadir más hardware, como por ejemplo el módulo de radio, el cual he albergado también dentro de la carcasa.

La pantalla se compone del LCD más un controlador que va alimentado por la Raspberry y un cable de display que va conectado a la placa, pero esto lo veremos posteriormente en el apartado correspondiente.

5.2.3. Módulo de radio SI4703

El módulo de radio es un pequeño IC (Integrated Circuit) o circuito integrado, que nos permite la sintonización de la radio.

Su funcionamiento es **externo** a nuestra placa de hardware, es decir, la sincronización entre ambas deberemos de proveerla nosotros vía software ya que en sí son dos placas autónomas e independientes. La comunicación con este hardware se hace mediante el protocolo I2C explicado más arriba, escribiendo a bajo nivel bits en las direcciones de memoria adecuadas para que el módulo obedezca nuestras órdenes.

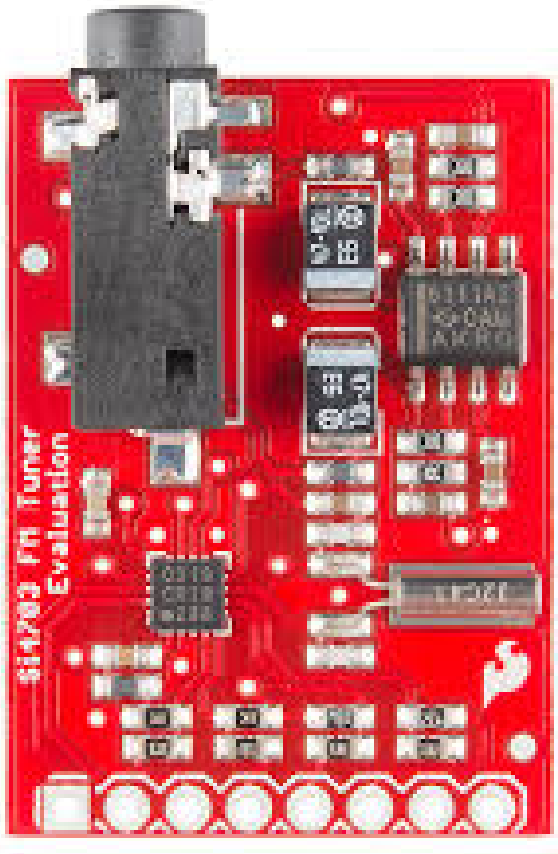


Figura 5.18: Imagen del IC SI4703 utilizado en este proyecto para sintonizar la radio.

Las características y posibilidades que nos ofrece este IC son las siguientes:

- Comunicación mediante el protocolo I2C.
- Sintonización de canales de radio FM.

- Opción de búsqueda automática de canal.
- Soporte para el protocolo RDS.
- Salida de audio mediante jack de 3,5mm.

Gracias a su pequeño tamaño, ha sido posible integrarlo dentro de la misma carcasa donde está contenido el resto del sistema para hacerlo todo más compacto y transportable. Dispone de 8 conexiones, de las cuales en este proyecto se utilizan 5, y que posteriormente veremos para qué sirven y cómo van conectadas.

Uno de los problemas, es que al ser un módulo independiente, tiene su propia salida de jack, por lo que deberemos de mezclar esta salida con la de la Raspberry Pi para poder escuchar tanto música como radio por la misma salida de auriculares.

5.2.4. Mezclador de audio

Como ya se ha indicado en el apartado anterior, el módulo de radio SI4703 tiene su propia salida de jack. Por lo tanto, la música reproducida desde nuestra aplicación saldría por el jack de audio de la Raspberry Pi, y el sonido de la radio saldría por el jack del SI4703. Esto es un gran problema ya que tenemos el audio en salidas separadas.

Para paliar esto, se ha añadido al proyecto un pequeño mezclador de audio que funciona a USB.

Figura 5.19: Ejemplo de una tarjeta de sonido/mezclador de audio USB como la usada en el proyecto.

Este mezclador es una tarjeta de sonido que se conecta por USB a la Raspberry, y nos permite por un jack conectar la salida de audio del módulo de radio, y por el otro jack nos va a sacar **juntas** las dos señales de audio,

teniendo así una única salida de audio. Además, esta tarjeta nos mejora la calidad del audio reproducirdo por la propia Raspberry, ya que el jack de audio de la placa no es de muy buena calidad y deja bastante que desear, sobretodo para un proyecto como este en el que se busca una calidad de reproducción de audio aceptable.

5.3. Conexionado del hardware

Ya hemos presentado todos los componentes hardware, ahora lo que toca es explicar cómo hacer para que éstos trabajen juntos.

A lo largo de toda la explicación, tenemos que tener como referencia la siguiente imagen, que es un esquema de los pines GPIO de la Raspberry Pi 3.

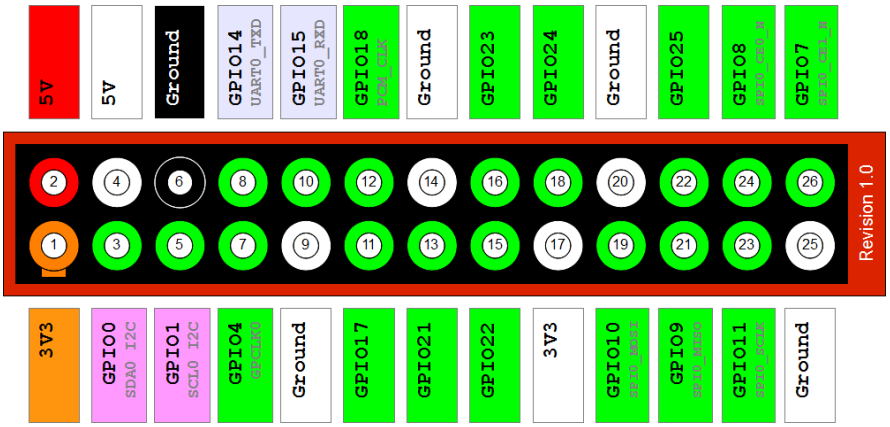


Figura 5.20: Esquema de los pines GPIO de la Raspberry Pi 3.

Haré referencia a este esquema de pines varias veces a lo largo de esta sección.

5.3.1. Conexionado de la pantalla

Primeramente vamos a empezar con la pantalla. La pantalla incluye también un controlador hardware que es el que va a permitir la comunicación entre nuestra Raspberry Pi y la pantalla táctil.

Aquí se puede ver el conexionado:

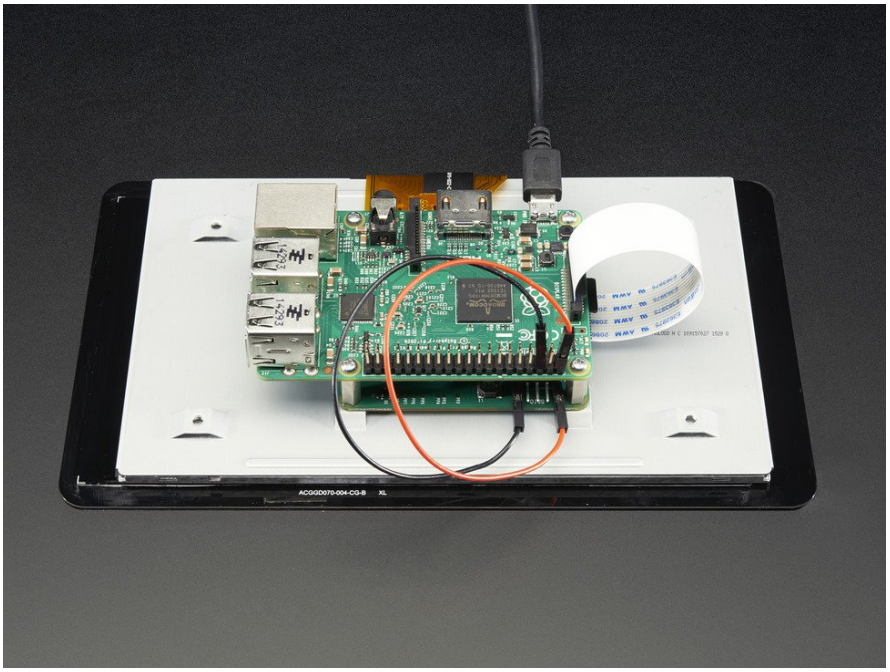


Figura 5.21: Conexionado de la pantalla táctil con la Raspberry Pi.

La faja de datos que sale de la pantalla táctil va conectada a la placa controladora. La placa controladora (la de abajo en la imagen) tiene para empezar una conexión de 4 pines.

De estos 4 pines, tendremos que conectar 2 al GPIO de la Raspberry, para alimentar la pantalla y la controladora. Los pines marcados como +5V y GND en la placa, tienen que ir respectivamente, al pin 2 del GPIO (5V), y el otro, que es el negativo, a cualquiera de las masas que tiene la Raspberry. En mi caso lo he conectado al pin 6, el cual se puede ver etiquetado como Ground (negativo o masa).

Posteriormente, de la placa controladora sale una faja de datos que tiene que ir conectada al puerto de interfaz de conexión de display de la Raspberry Pi (DSI), como se puede apreciar en la imagen.

Esta faja DSI va a encargarse tanto de la imagen como de controlar las pulsaciones de la pantalla táctil. Con esto estaría todo el conexionado de la pantalla realizado.

5.3.2. Conexionado del módulo de radio

Ahora lo que toca es conectar el módulo de radio SI4703 a nuestra Raspberry.

Vamos a utilizar 5 de sus pines, que son 3V3 (positivo a 3,3V), GND (tierra), RST (reset), SDA y SCL (pines para el protocolo I2C).

Estos pines irían conectados de la siguiente manera:

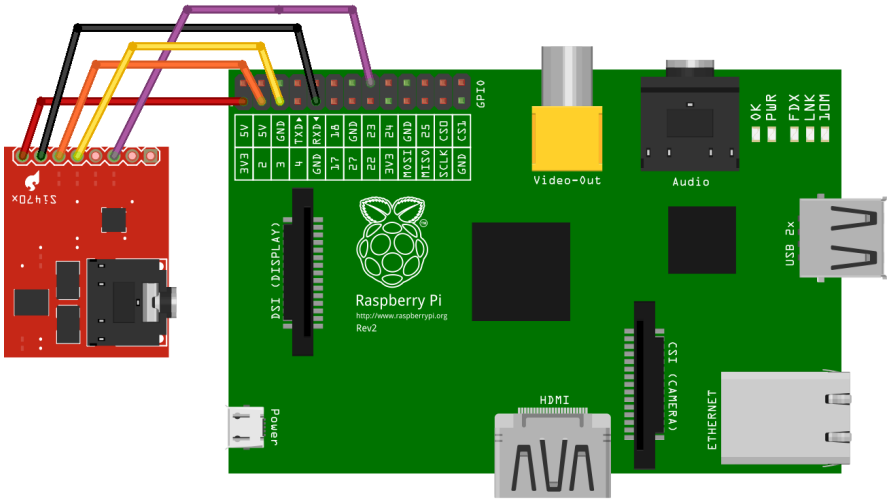


Figura 5.22: Conexionado del módulo SI4703 con la Raspberry Pi.

Con esto tendríamos la comunicación a nivel físico con el módulo de radio completada (más adelante en el apartado destinado al software veremos como hacemos la comunicación a nivel lógico).

5.3.3. Conexionado del mezclador de audio

Por último, nos quedaría la conexión del mixer o mezclador de audio. Se puede observar en el siguiente esquema:

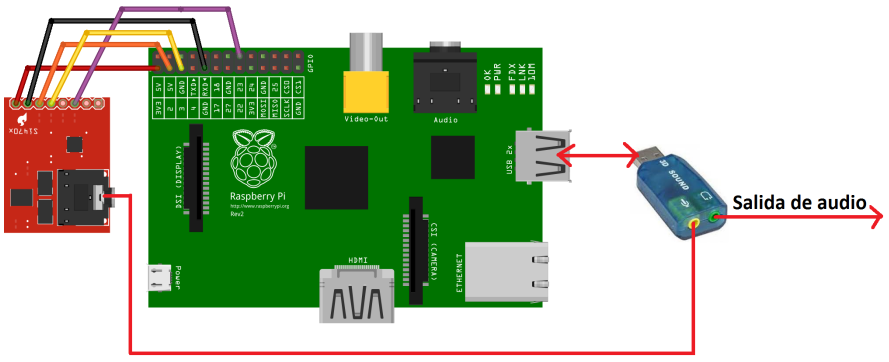


Figura 5.23: Conexionado del mezlador de audio con la Raspberry Pi.

Como se puede apreciar, la salida de audio del módulo de radio va a parar al mezclador, el cual recoge por USB la salida de audio de la Raspberry Pi, configurándolo como dispositivo predeterminado de salida de audio dentro del sistema operativo, y saca ambas señales juntas por el jack marcado como salida de audio.

Figura 5.24: Imagen de mi sistema completamente armado en funcionamiento.

5.4. Instalación del software y preparación del entorno de desarrollo

En esta sección pretendo hacer un breve resumen sin extenderme demasiado de cómo he montado el entorno de desarrollo.

5.4.1. Instalación de Python y PyCharm

Primeramente, se ha utilizado PyCharm para programar en Python sobre Windows, utilizando la distribución Anaconda[5]. Hay información detallada sobre PyCharm en la sección de herramientas utilizadas en este proyecto.

Se ha decidido utilizar Python por varias razones, entre las que están que es multiplataforma o que es un lenguaje que conozco bastante bien, pero la decisión real detrás de utilizar este lenguaje es que es el que más soporte tiene para Raspberry Pi y tiene soporte con QT (PyQT) oficial. Inicialmente se

[5]Anaconda es una distribución de Python que incluye el intérprete junto a otras librerías y añadidos: https://www.continuum.io/downloads

planteó hacer el proyecto sobre Java, pero tenía muy poco soporte, no había librerías gráficas decentes como QT que pudiesen soportar un proyecto como este que necesita tanto detalle y personalización de la interfaz gráfica, y otra serie de motivos como que programar en PC y enviarlo a la Raspberry se hacía más difícil. Así pues, se acabó escogiendo Python tanto por comodidad como por tener mejor soporte y mejores utilidades sobre Raspberry Pi.

Una vez tenemos PyCharm con Python montado y todas las librerías necesarias, he creado mis propios scripts para enviar el programa entero a la Raspberry Pi. Estos scripts funcionan utilizando Putty[6] y SSH, son dos archivos .bat que se ejecutan sobre Windows, uno llamado *sendPi.bat* que envía todos los archivos necesarios para ejecutar la aplicación a la Raspberry Pi, y otro llamado *cleanPi.bat*, que borra remotamente los archivos de la Raspberry. Estos archivos se pueden encontrar junto al código fuente del proyecto.

5.4.2. Instalación del software necesario en la Raspberry Pi

En cuanto a la Raspberry Pi, lo único que tenemos que hacer es instalar **Raspbian**[7]. Raspbian es una versión de la distribución de Linux Debian para la Raspberry Pi.

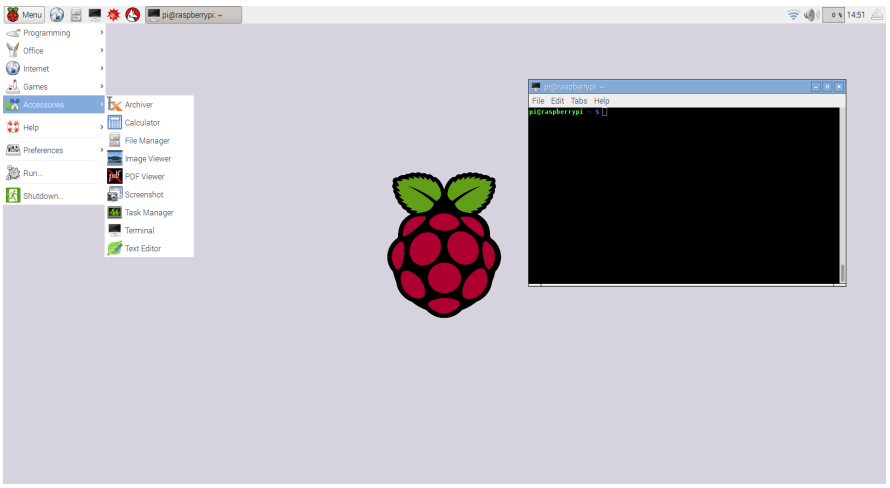


Figura 5.25: Raspbian es el sistema operativo utilizado sobre la Raspberry Pi en este proyecto.

Una vez instalado, se ha configurado SSH y VNC sobre el sistema operativo para poder enviar archivos mediante SSH y utilizar el escritorio remoto desde VNC.

[6]Putty es un cliente de SSH gratuito: http://www.putty.org/
[7]Página web oficial de Raspbian: https://www.raspberrypi.org/downloads/raspbian/

Con estas utilidades correctamente configuradas e instaladas, ya estaría el entorno de desarrollo listo para ser utilizado y comenzar a programar.

5.5. Desarrollo del software

Esta probablemente sea la sección más importante. Aquí se va a recoger toda la información a cerca del ciclo de vida del proyecto, las fases que ha seguido, las dificultades técnicas y problemas en cuanto a la programación del software se refiere que me he encontrado, y la soluciones que he decidido tomar para cada caso, que justifica todo lo anteriormente explicado en este documento, desde qué librerías o lenguajes se han utilizado hasta qué módulos de hardware han sido elegidos.

El ciclo de vida del proyecto es un ciclo en cascada iterativo, en el que se han realizado dos iteraciones (milestones) principales:

- Una primera iteración en la que se ha implementado toda la interfaz gráfica y la reproducción de audio.
- Una segunda iteración en la que se ha implantado la reproducción de la radio.

Las milestones que se han hecho son bastante largas, en contrapartida a otros proyectos que quizás hacen milestones mucho más cortas. La duración media de cada milestone ha rondado el mes y medio.

5.5.1. Milestone 1 - Interfaz gráfica y reproducción de audio

Creación de la interfaz gráfica y del sistema de menús

Como ya se ha indicado más arriba, la librería gráfica elegida para el proyecto es PyQT. Así pues, lo primero que se ha realizado en este proyecto es implementar la funcionalidad del menú en sí, el cambio de menús y la creación de botones personalizados.

La estructura básica de la aplicación desde el punto de la ingeniería del software, trata de ser algo así:

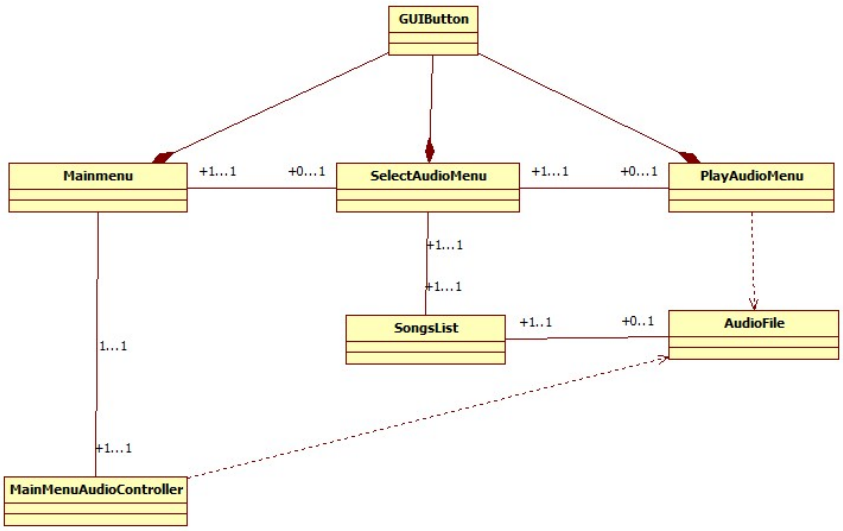


Figura 5.26: Modelo de dominio de la primera iteración del proyecto.

Se empezó implementando los botones personalizados con imagen, para ello fue necesario definir un tipo de botón personalizado reescribiendo parte del comportamiento del botón en PyQT.[14]

Este tipo de botón personalizado que reescribí, a parte de dejarme asociar un gráfico al botón, me permite asociarle al hacer click la acción que yo decida.

Cada vez que necesito un botón, creo una clase que es un **Factory Method** del botón personalizado, donde redefino el comportamiento del botón.

Figura 5.27: Comportamiento del botón personalizado.

Aquí se puede observar uno de los botones personalizados. Tiene asociado un gráfico para cuando está sin pulsar y otro para cuando está pulsado. En la clase que define el comportamiento del botón, defino sus parámetros, como el tamaño del botón, el texto que lleva encima, su comportamiento al hacer click (en este caso apagar nuestro sistema), o las imágenes que lo conforman.

El otro tema por abordar sobre la interfaz gráfica son los menús y los cambios entre ellos.

Esto se consigue de la siguiente manera: se define un widget en la ventana principal de tu aplicación QT que contiene todos los menús. Cada menú es un widget individual personalizado (se puede ver arriba en el modelo de dominio como cada menú pertenece a una clase separada). Luego lo que tienes que ir haciendo es añadir esos menús al widget central de la ventana, y elegir qué menú visualizarás de todos los que has insertado en el widget central con el método *setCurrentWidget* de QT.[17]

En el siguiente ejemplo extraído del proyecto se puede apreciar su uso:

```
self.centralWidget.addWidget(self.playAudioMenuWidget)
self.centralWidget.setCurrentWidget(self.playAudioMenuWidget)
```

Se puede observar cómo se añade a nuestro widget central, el cual contiene todos los menús, un menú concreto, y posteriormente se cambia la vista a ese menú. El resultado es que hemos conseguido cambiar de menú satisfactoriamente.

Reproducción de ficheros de audio con libVLC

Para la reproducción de audio, utilizo libVLC en mi proyecto. Esta librería me permite reproducir multitud de formatos de audio, aunque de momento solo lo utilizo para reproducir el formato MP3 que es el que me interesaba para este proyecto. Inicialmente con muy pocos cambios en el código puedo hacer que reproduzca cualquier otro formato que soporte la librería.

El funcionamiento vendría a ser de la siguiente manera:

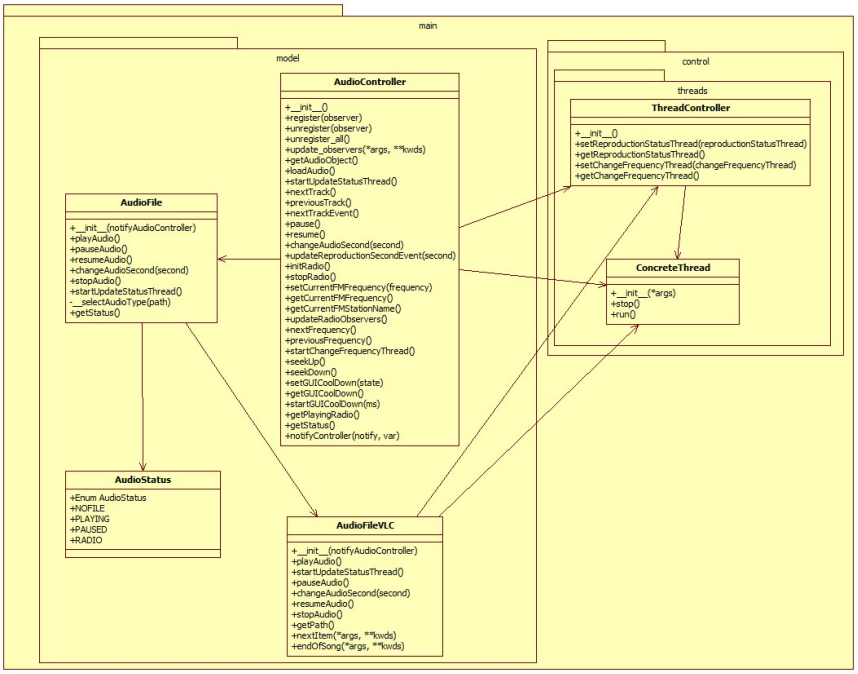


Figura 5.28: Diagrama de clases correspondiente a la parte de la reproducción de archivos de audio.

En este diagrama de clases se puede observar como funciona la estructura de la reproducción de ficheros de audio.

Tenemos una clase *AudioController* que va a ser la encargada de controlar todo lo que tenga que ver con el audio, tanto el proveniente de la radio como el proveniente de la librería de VLC. Posteriormente tenemos las clases *AudioFile* y *AudioFileVLC*, que básicamente vienen a ser un **patrón Facade**. *AudioFileVLC* implementa directamente las llamadas a la librería VLC para la reproducción de audio, mientras que *AudioFile* hace de intermediario, para que el código sea más fácil de mantener si en un futuro se decide cambiar de librería.

Los botones de la GUI de los que hemos hablado anteriormente, van a ser los encargados mediante sus acciones al hacer click de llamar a los métodos de *AudioController* para indicar qué acciones quieren que se realicen con los archivos de audio.

Otro aspecto importante es la clase *ThreadController* y los *ConcreteThread*. Un *ConcreteThread* es una clase para un hilo[21] concreto, pero como todas tienen la misma implementación en cuanto a métodos y solo cambia la funcionalidad interna, he preferido resumir el diagrama así.

Estas clases básicamente van a manejar una serie de hilos que son utilizados para diversas tareas. En el caso de la reproducción de audio, hay un hilo **muy importante**, que es el ***ReproductionStatusThread***. Este hilo se utiliza para obtener el segundo de reproducción actual de la canción y para actualizarlo en la GUI.

Para obtener el segundo de reproducción preciso de la canción, hay que ejecutar llamadas a uno de los métodos de la librería VLC, ya que no hay otra manera de averiguarlo. Así que la solución que se ha tomado es hacer **polling**[8] al método de la librería para obtener por qué segundo va la reproducción. Para hacer este polling, se crea un hilo a parte que va sondeando el dato de los segundos reproducidos cada poco tiempo, ya que si no lo hiciéramos en un hilo separado, bloquearíamos la interfaz con las esperas para consultar, ya que es un bucle infinito que va consultando el estado.

Este hilo pues, va a realizar la consulta y va a notificar a la GUI del nuevo valor del segundo de reproducción, para que le actualice el dato al usuario.

Cabe destacar, a modo anecdótico, el uso del **patrón Observer**[2] a lo largo de toda la aplicación para notificar a la GUI de nuevos cambios y de con qué datos tiene que actualizarle la vista al usuario.

Por último, en referente a la reproducción de audio, tenemos una última cosa de la que no se ha hablado. Los ficheros de audio, contienen una serie de **metadatos**, que nos van a proporcionar diversa información sobre la canción que se está reproduciendo.

Por poner algunos ejemplos, nos va a proporcionar información como:

- Nombre de la canción.
- Nombre del grupo.
- Nombre del álbum.
- Género musical.
- Portada del álbum (si viene dentro de los metadatos).
- Año de salida del álbum.

La librería nos va a proporcionar más información, pero estas son quizás las entradas de más relevancia y las que más nos interesan.

La estructura queda reflejada en el siguiente diagrama de clases:

[8] Se conoce como polling a realizar consultas constantes sobre un dato para saber cuál es su estado actual, o para crear una actividad síncrona sin utilizar interrupciones.

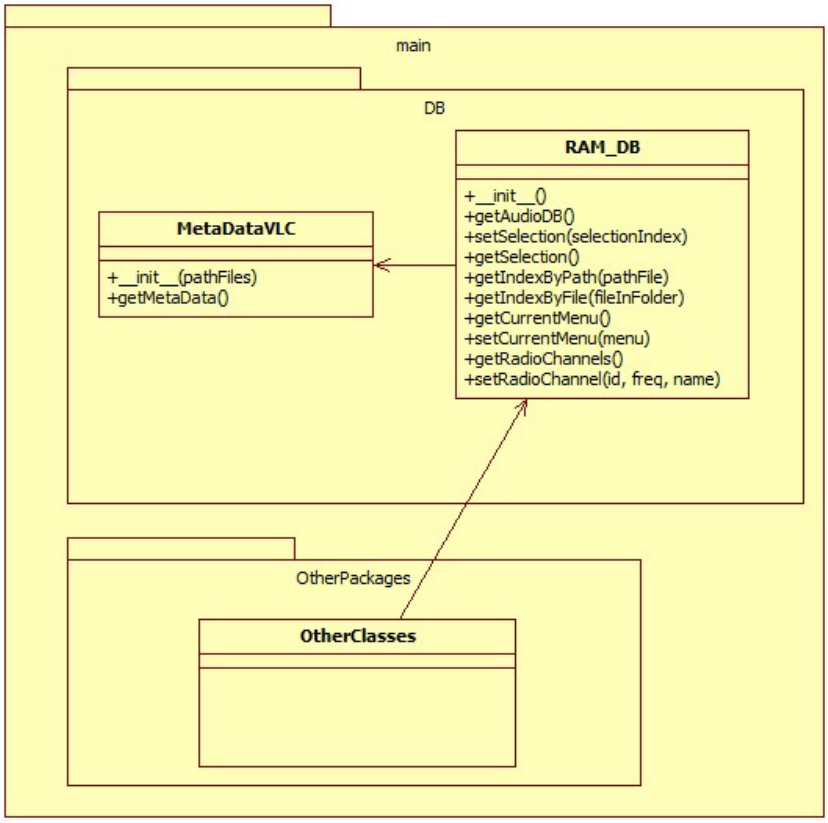


Figura 5.29: Diagrama de clases correspondiente a la parte de la estructura de datos principal de la aplicación que provee los metadatos.

Tenemos una clase *MetaDataVLC*, que es la encargada de la lectura de los metadatos de las canciones utilizando la librería VLC para ello. Posteriormente, tenemos la clase *RAM_DB*. Esta clase contiene un **patrón Singleton**[16], así puede ser instanciada desde cualquier lugar de la aplicación y siempre contendrá los mismos datos. Esta clase va a proveer a las demás clases que la instancien (como se ve en el diagrama), toda la información que necesitan a cerca de la reproducción actual, metadatos y otra serie de información.

De esta manera, los metadatos son leídos por la GUI, y se cambia la vista para que el usuario vea los datos de la canción que está reproduciendo en ese momento.

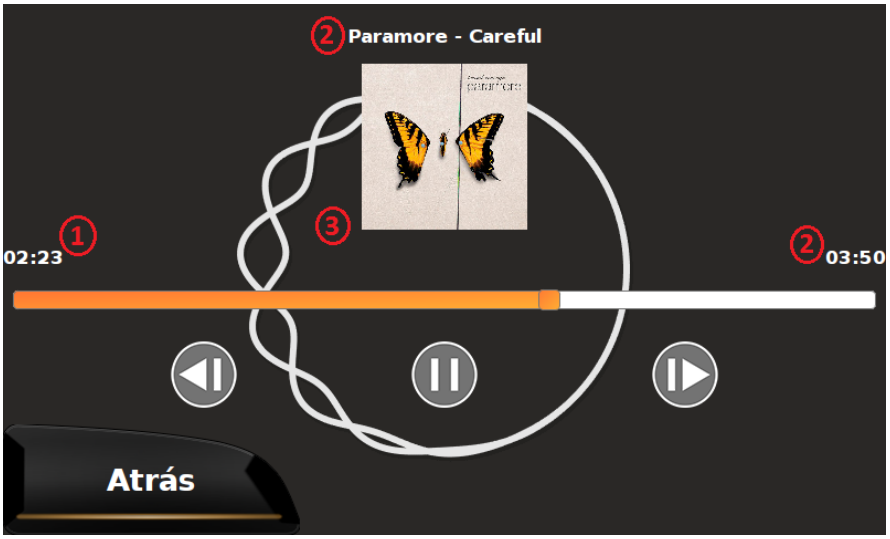


Figura 5.30: Vista de la aplicación en la que se puede observar el uso de los metadatos junto con la GUI.

En la imagen podemos observar:

- **1:** Lectura del segundo de reproducción actual de la canción, leída por el hilo correspondiente mediante polling, como se ha explicado anteriormente.
- **2:** Metadatos obtenidos del propio archivo MP3, como el nombre del grupo, el nombre de la canción o la duración de la misma.
- **3:** Imagen de la portada del disco, obtenida a través de los metadatos. Lo normal es que no se incluya. Si no encuentra ninguna portada de disco en los metadatos, en vez de aparecer la portada, aparece una imagen genérica.

Por último y no por ello menos importante, cabe destacar la clase *SelectAudioMenu*, la cual se puede observar en el modelo de dominio de los menús del apartado anterior. Esta clase implementa una lista de ítems[18] donde se va a mostrar todas las canciones en la lista, las cuales son leídas a través de los datos que nos proporciona la clase *RAM_DB*[15].

Con todo esto detallado, quedaría visto el funcionamiento de la reproducción de ficheros de audio con libVLC.

Reproducción en curso en el menú principal

Otra de las características que implementa el software es el poder modificar la reproducción en curso desde el menú principal. Esta opción se entiende más desde la perspectiva de que, a fin de cuentas, es una autorradio, y esta es una de esas características pensadas para el hecho de utilizarla desde el coche.

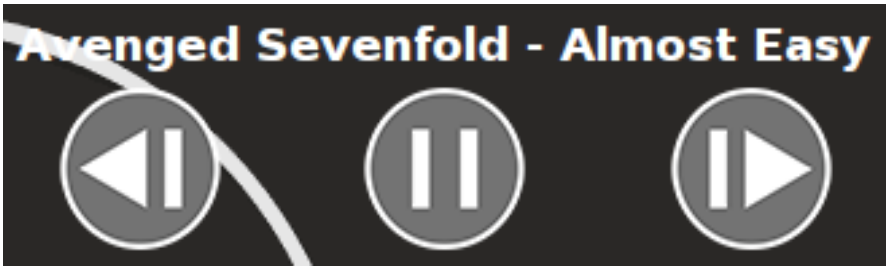


Figura 5.31: Vista del widget que permite modificar la reproducción en curso desde el menú principal.

El widget de la imagen es un widget personalizado creado con QT, que está contenido en la clase *MainMenuAudioWidget* del proyecto. Provee un área de texto y tres botones, cuya pulsación va a hacer que se modifique la reproducción actual.

Cabe destacar que, en la siguiente milestone, también se le ha añadido a este widget la funcionalidad de permitir **controlar la música o la radio**, en función de si lo que se está utilizando en ese momento es el reproductor de música o se tiene sintonizada la radio, adaptándose a cada perfil según lo requiera.

5.5.2. Milestone 2 - Reproducción de la radio

Funcionamiento general

La segunda iteración del proyecto se basa en implementar toda la funcionalidad de la radio, desde los menús hasta la comunicación con el módulo de hardware.

Como ya se ha explicado más arriba, la comunicación con la radio se realiza mediante el protocolo I2C.

A modo de resumen, he creado una librería propia para manejar el módulo y la conexión mediante I2C. La librería se corresponde con la clase *SI4703.py* dentro de la carpeta *lib* del proyecto.

Ahora bien, el funcionamiento de mi software para implementar el control y la interacción con la radio es el siguiente:

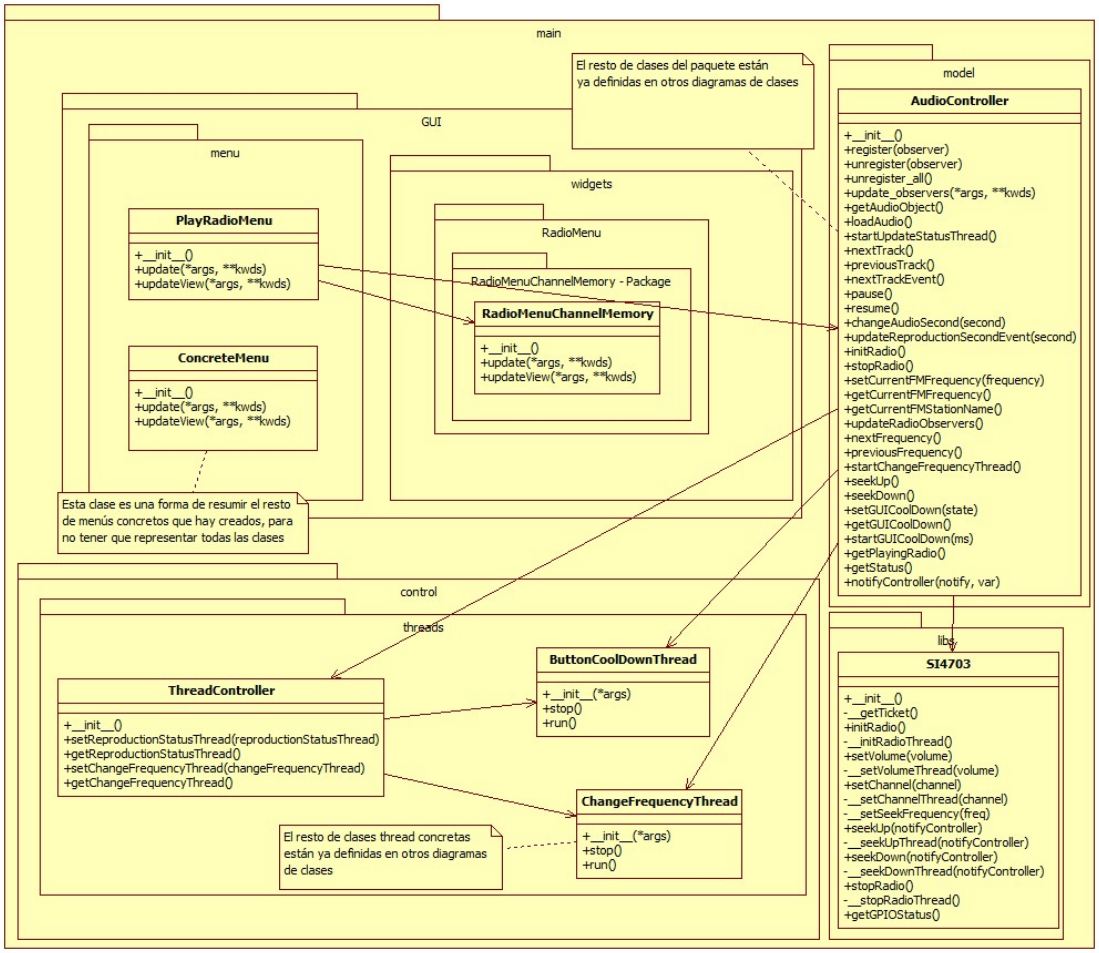


Figura 5.32: Diagrama de clases que refleja el funcionamiento de la radio y su interacción con la interfaz.

En este diagrama de clases, podemos ver las clases que implementan la funcionalidad de la radio.

Para empezar, tenemos el menú concreto *PlayRadioMenu*, que es una clase que implementa el menú de la radio. Dentro de este menú tenemos un widget que es el que almacena las emisoras en la memoria, que es la clase *RadioMenuChannelMemory*.

El menú de la radio, *PlayRadioMenu* va a interaccionar con la radio a través del *AudioController*, que va a ser nuestra pasarela entre la GUI y mi librería *SI4703*, que va a ser la encargada de comunicarse con la radio a **bajo nivel usando el protocolo I2C**[11].

La clase *AudioController* tiene también la responsabilidad de manejar un par de tareas extras, que son las de lanzar los hilos que implementan las clases *ButtonCoolDownThread* y *ChangeFrequencyThread*.

Cuando nosotros realizamos una acción en la radio (por ejemplo cambiar de emisora), al hardware le toma un tiempo realizar la acción. Tiene que escribir una serie de registros, y puede tardar tiempos considerables (mayores a un segundo) en dejar el módulo de radio en un estado estable y listo para ejecutar una nueva acción (por ejemplo sintonizar otra emisora diferente). Como estos retardos hay que respetarlos sí o sí por limitaciones del hardware, de cara al usuario se ha decidido implementar un cool down en los botones, que vendría a ser un tiempo en el que los botones se muestran como inactivos. Es decir, mientras el hardware de radio está ocupado y no puede realizar una nueva acción, los botones se van a desactivar. Aquí es cuando entra en juego el hilo que implementa la clase *ButtonCoolDownThread*, la cual se va a encargar de esperar el tiempo necesario paralelamente para no congelar la GUI, mostrando los botones como inactivos, y va a volver a activar los botones cuando la espera haya finalizado y el hardware esté listo para recibir una nueva orden.

Por último, tenemos la opción de cambiar entre emisoras subiendo la frecuencia de 0,1 en 0,1 MHz. Como ya hemos mencionado, cada vez que se cambia de emisora el hardware necesita un tiempo para realizar la acción, por lo que sería lentísimo ir por ejemplo de la frecuencia 89.7 a la 100.1. Para evitar este inconveniente, está el hilo implementado en la clase *ChangeFrequencyThread*, se encarga de realizar una cuenta atrás cuando nosotros cambiamos rápido entre canales, así hasta que nosotros no nos paramos en una frecuencia concreta, no va a sintonizarla, dándonos un margen de tiempo para llegar a nuestra frecuencia de destino sin que esté intentando sintonizar todas las que hay entre medias.

Por último, cabe destacar que el widget *RadioMenuChannelMemory* almacena las emisoras y las lee desde un fichero XML[19], llamado *RadioChannels.xml*, el cual se encuentra dentro de la carpeta *config* del proyecto. Para parsear el XML, se utiliza la librería **lxml**[6] de Python.

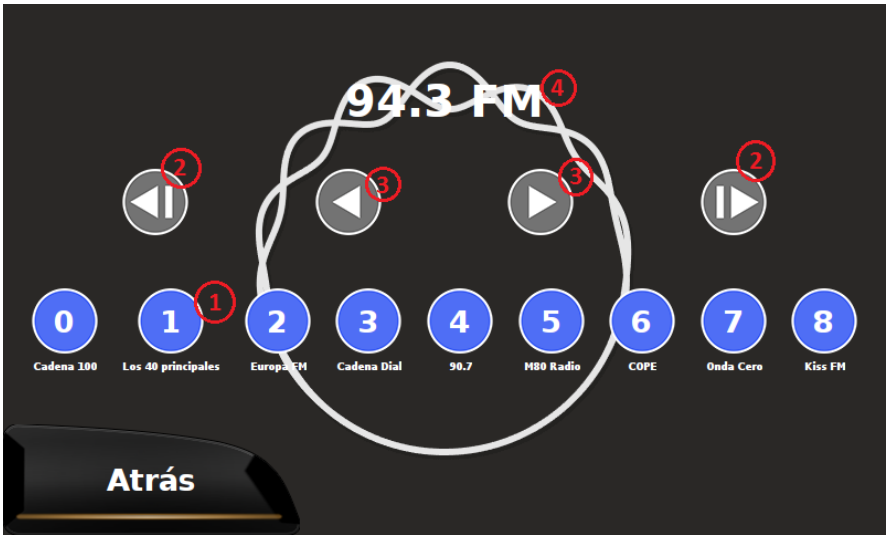


Figura 5.33: Vista de la aplicación en la que se puede observar la radio en funcionamiento.

En la imagen podemos observar:

- **1:** Emisoras almacenadas por el widget *RadioMenuChannelMemory*.
- **2:** Botones de búsqueda de emisoras automática.
- **3:** Botones para cambiar de emisora manualmente, cuyo tiempo de acción es controlado por el hilo implementado en la clase *ChangeFrequencyThread*.
- **4:** Emisora sintonizada actualmente.

Comunicación con el hardware de radio mediante el protocolo I2C

Como ya hemos dicho más arriba, nos comunicamos con el módulo de radio, el SI4703 mediante el protocolo I2C. Todo esto está implementado en la librería contenida en la clase *SI4703*, no obstante aquí vamos a dar una explicación más detallada de cómo comunicarse mediante I2C con un dispositivo.

Para ello, vamos a utilizar las librerías **smbus** y **RPi.GPIO**, de las cuales hemos hablado más en profundidad en apartados anteriores.

Para que el sistema reconozca el módulo de radio, es necesario realizar cierta configuración previa mediante los pines GPIO, a fin de poder establecer comunicación mediante el protocolo I2C. Vamos a analizar un pequeño trozo de código para ver cómo funciona la comunicación mediante I2C:

```
GPIO.setmode(GPIO.BCM)  # Forma de numerar los pines.
GPIO.setup(23, GPIO.OUT)
time.sleep(.2)
GPIO.setup(0, GPIO.OUT)  # SDA o SDIO
time.sleep(.2)

# Ponemos el integrado SI4703 en modo de dos cables (I2C)
GPIO.output(0, GPIO.LOW)
time.sleep(.2)
GPIO.output(23, GPIO.LOW)
time.sleep(.2)
GPIO.output(23, GPIO.HIGH)
time.sleep(.2)
```

Esta parte del código utiliza la librería **RPi.GPIO** para enviar señales binarias a través de los puertos GPIO. Concretamente, la función **GPIO.output** nos va a permitir seleccionar un pin (primer parámetro de la función), y enviar a través de él un bit (segundo parámetro de la función). El parámetro **GPIO.LOW** va a enviar un 0 binario, mientras que el parámetro **GPIO.HIGH** va a enviar un 1 binario.

En este trozo de código, enviamos esos bits acorde a lo que manda el datasheet del integrado SI4703 para activar la comunicación con él mediante I2C. Una vez está activada la comunicación, vamos a enviarle bloques de bytes a través de la librería **smbus**. En el siguiente extracto de código, podemos apreciar como leemos la memoria del integrado y como mandamos escribir en su memoria un bloque de bytes determinado:

```
reg = self.i2c.read_i2c_block_data(self.address, 0, 32)
time.sleep(.2)
# Escribimos el num. 8100h en la dir. 07h
# (Inicia el oscilador de cristal)
list1 = reg[17:28:]
list1[9] = 129
list1[10] = 0
w6 = self.i2c.write_i2c_block_data(self.address, 0, list1)
# Espera a que el oscilador de cristal se estabilice.
time.sleep(1)
```

En este trozo de código podemos apreciar las 2 funciones que vamos a utilizar de la librería *smbus*. La primera es *i2c.read_i2c_block_data*, la cual tiene 3 parámetros. El primero indica la dirección lógica dentro del bus I2C de nuestro dispositivo (siempre va a ser la misma para el mismo dispositivo), el segundo parámetro indica desde qué byte se va a empezar a leer, y el tercer

parámetro indica cuántos bytes se van a leer a partir del primer byte indicado (segundo parámetro).

La segunda función es *i2c.write_i2c_block_data*, la cual tiene también 3 parámetros. El primero es nuevamente la dirección lógica de nuestro dispositivo. El segundo parámetro es el primer byte **escrito en decimal** que va a escribir en la memoria, y el tercer parámetro, el cual es opcional, acepta que se le introduzca una lista con el resto de bytes **escritos en decimal** en el orden que se desea que sean escritos.

Por ejemplo, si nosotros ejecutamos lo siguiente:

```
self.i2c.write_i2c_block_data(self.address, 1)
```

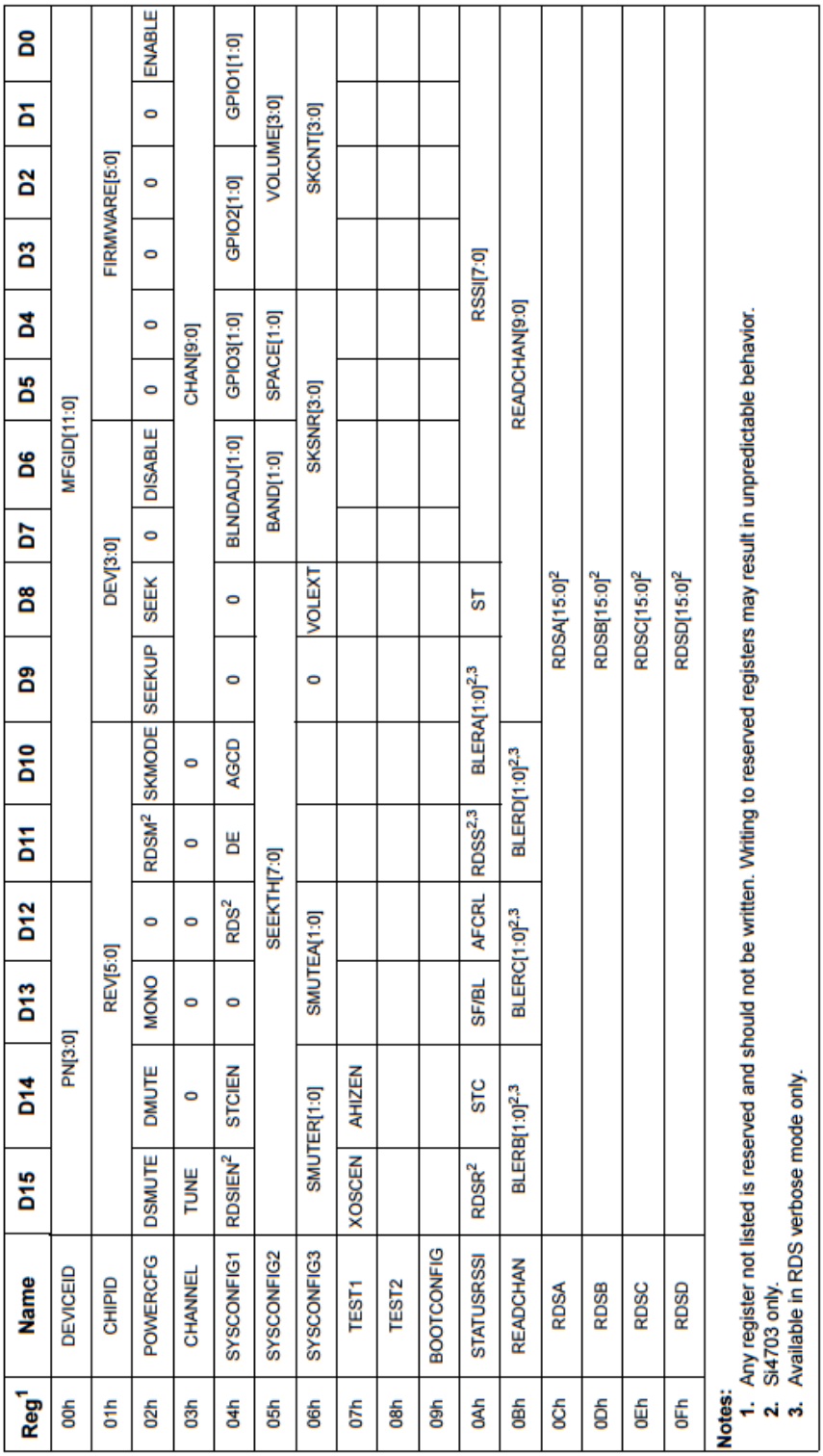
La librería *smbus* va a escribir el número **1 (decimal)** en forma de byte (binario). Por lo tanto, escribiría en la memoria el número **0000 0001**.

Manejando el módulo SI4703

Para manejar el módulo hardware de radio SI4703, y realizar todas las secuencias correctamente, como las secuencias de encendido, apagado, cambio de canal... no nos queda otra alternativa que consultar directamente el datasheet del componente[12] y la guía del programador[13] del componente. De cada documento nos interesan cosas distintas: De la guía del programador podemos sacar cómo ha de hacerse para realizar determinadas operaciones con el IC, es decir, en qué bancos de memoria debemos de escribir utilizando el protocolo I2C.

Del datasheet, lo que nos interesa en sí únicamente es la tabla donde viene mapeada toda la memoria del integrado, con información de todos los registros y su utilidad.

Concretamente ésta es la tabla que nos interesa, extraída del propio datasheet[12]:



Reg[1]	Name	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
00h	DEVICEID	PN[3:0]				MFGID[11:0]											
01h	CHIPID	REV[5:0]				DEV[3:0]					FIRMWARE[5:0]						
02h	POWERCFG	DSMUTE	DMUTE	MONO	0	RDSM[2]	SKMODE	SEEKUP	SEEK	0	DISABLE	0	0	0	0	0	ENABLE
03h	CHANNEL	TUNE	0	0	0	0	0	CHAN[9:0]									
04h	SYSCONFIG1	RDSIEN[2]	STCIEN	0	RDS[2]	DE	AGCD	0	0	BLNDADJ[1:0]		GPIO3[1:0]		GPIO2[1:0]		GPIO1[1:0]	
05h	SYSCONFIG2	SEEKTH[7:0]								BAND[1:0]		SPACE[1:0]		VOLUME[3:0]			
06h	SYSCONFIG3	SMUTER[1:0]		SMUTEA[1:0]				0	VOLEXT	SKSNR[3:0]				SKCNT[3:0]			
07h	TEST1	XOSCEN	AHIZEN														
08h	TEST2																
09h	BOOTCONFIG																
0Ah	STATUSRSSI	RDSR[2]	STC	SF/BL	AFCRL	RDSS[2,3]	BLERA[1:0][2,3]		ST	RSSI[7:0]							
0Bh	READCHAN	BLERB[1:0][2,3]		BLERC[1:0][2,3]		BLERD[1:0][2,3]		READCHAN[9:0]									
0Ch	RDSA	RDSA[15:0][2]															
0Dh	RDSB	RDSB[15:0][2]															
0Eh	RDSC	RDSC[15:0][2]															
0Fh	RDSD	RDSD[15:0][2]															

Notes:
1. Any register not listed is reserved and should not be written. Writing to reserved registers may result in unpredictable behavior.
2. Si4703 only.
3. Available in RDS verbose mode only.

Figura 5.34: Tabla con el mapeo de la memoria del IC SI4703.

Para controlar el módulo de radio, lo que vamos a hacer es escribir determinados bits en la memoria para que el módulo obedezca a nuestras órdenes. Ahora bien, si a simple vista parece sencillo, la lectura y escritura en la memoria del IC no es nada intuitiva.

Aviso: A partir de ahora, se van a dar explicaciones ateniéndonos a la tabla de memoria de la página anterior donde están todas las direcciones. Conviene tenerla a mano para seguir la explicación.

A la hora de hacer lecturas, va a empezar a leer a partir de la dirección **0Ah**. Como ya hablamos más arriba, las lecturas se hacen con la función *i2c.read_i2c_block_data(address, cmd, longitud)*. Pues bien, cada vez que hagamos una lectura, el número introducido en el parámetro *cmd*, que indica desde dónde empezar a leer, nos lo va a escribir en la posición **02h** en forma binaria como si de una entrada se tratase. Es decir, por ejemplo, si nosotros queremos leer un dato de la posición *07h*, podemos ejecutar el siguiente comando:

```
reg = self.i2c.read_i2c_block_data(self.address, 0, 32)
```

Esto nos va a devolver una lista de 32 elementos, siendo cada elemento **un byte diferente**, y conteniendo **cada registro dos bytes**. Por lo tanto, la dirección *07h* va a corresponderse con los elementos 26 y 27 de la lista devuelta (2 bytes cada registro, empezando por el 0Ah). Además, como el valor del parámetro *cmd* es **0**, en la posición de memoria **02h** va a escribirnos un byte a 0 en binario: **0000 0000**. Esto es así por cómo funciona el propio integrado internamente, y son decisiones del fabricante a las que nosotros no le podemos hacer mucho más que acatarlas.

Ahora le toca el turno a la escritura, como ya hemos visto más arriba, esta se hace con el comando *i2c.write_i2c_block_data(address, cmd, list)*. Al realizar una operación de escritura, va a empezar escribiendo en la posición **02h**, nuevamente por decisión del fabricante. Así pues, el número del parámetro *cmd* se va a corresponder con el primer byte del registro **02h**, y el resto de la lista a los bytes contiguos a esta dirección de memoria.

Por ejemplo, vamos a leer más detalladamente el código de ejemplo que he dejado sobre la escritura más arriba:

```
reg = self.i2c.read_i2c_block_data(self.address, 0, 32)
time.sleep(.2)
# Escribimos el num. 8100h en la dir. 07h
# (Inicia el oscilador de cristal)
list1 = reg[17:28:]
list1[9] = 129
list1[10] = 0
w6 = self.i2c.write_i2c_block_data(self.address, 0, list1)
# Espera a que el oscilador de cristal se estabilice
```

```
time.sleep(1)
```

Este código lo que hace básicamente es, primero lee los registros, ya que si quieres escribir como en el ejemplo en la dirección *07h*, va a escribir todo lo que hay entre la dirección *02h* (donde empieza a escribir), y la dirección *07h*, que es donde queremos realmente escribir. Machacar todos los datos que hay entre medias de esas dos direcciones es peligroso, ya que podría causar errores, por lo que debemos leerlos para volver a reescribir exactamente lo mismo que había antes, modificando solo los bytes pertenecientes a la dirección *07h*.

Según la guía del programador[13], uno de los pasos para iniciar la radio (al cual pertenece el extracto de código), es escribir el número hexadecimal *8100h* en la dirección *07h*, para arrancar el oscilador.

Para hacer esto, creamos una lista que contenga todos los valores desde la dirección *02h* hasta la *07h* leídos anteriormente, y modificamos justamente los dos valores que pertenecen a los dos bytes de la dirección *07h*. En este caso, va a escribir (en decimal), un **129** y un **0**, que en hexadecimal se corresponden con los números **81** y **00** (que juntos forman el **8100h** que nos manda escribir la guía del programador, y que a su vez en decimal forman los bytes **1000 0001** y **0000 0000**.

Si nos fijamos en la tabla del datasheet adjunta más arriba con el mapeo de memoria, vemos que al escribir ese primer byte (*1000 0001*), estamos poniendo el bit del registro **XOSCEN** a **1**, lo que va a activar el oscilador. El resto de valores que se escriben (en blanco en la tabla del mapeo de memoria), son bits que no nos incumben como programadores y que simplemente debemos de respetar y escribirlos con lo que la guía del programador nos ordene.

Esta sería básicamente la manera de leer y escribir, y al fin y al cabo de interaccionar con el módulo de radio a través del protocolo I2C. Es algo tedioso y complicado, además de que somos responsables de la sincronización entre el módulo y la Raspberry Pi (ya que, recordemos, el módulo trabaja de forma independiente y solamente obedece nuestras órdenes), y responsables de asegurarnos de hacer la secuencia de apagado en el módulo bajo cualquier circunstancia. En caso contrario, por ejemplo, podría darse el caso perfectamente de cerrar nuestra aplicación pero que el módulo de radio siga sonando y funcionando independientemente.

Trabajos relacionados

Estos son los trabajos relacionados más relevantes con los que me he encontrado por la red:

- **PyCar:** PyCar es un proyecto de media center desarrollado sobre Python, pensado para ejecutarse sobre un PC en un coche. Entre sus funcionalidades, nos encontramos con la reproducción de música en varios formatos, la posibilidad de visionar vídeo, crear playlists de audio, además de incluir otras cosas como interfaz con soporte táctil.[8]
- **Car PC Project:** Este proyecto, es un proyecto de autorradio orientado a Raspberry Pi. Principalmente, utiliza el reproductor multimedia *Kodi*[9] como base y le añade una serie de modificaciones para darle una interfaz más adecuada a un automóvil. Permite la reproducción de distintos formatos de audio y vídeo, uso de pantalla táctil, cámara trasera, radio y GPS.[1]
- **iCarus:** iCarus es otro proyecto de autorradio para Raspberry Pi, aunque a diferencia del resto, este es **comercial y de pago**. Al igual que los anteriores proyectos, incluye reproducción de audio, de vídeo, radio, GPS y alguna funcionalidad más, además de poder controlarlo con una pantalla táctil. También te lo ofrecen ya montado dentro de la carcasa necesaria para insertarlo en el coche, siendo también un proyecto comercial completo con hardware además del software.[4]

[9]Kodi es un media center de código abierto, se puede encontrar aquí: https://kodi.tv/

Conclusiones y líneas de trabajo futuras

7.1. Conclusiones finales

Este proyecto nació como una idea de unir dos cosas que me gustan, que son la informática y el mundo del automóvil.

En un inicio, pensé que iba a poder implementar muchas más funcionalidades de las que tiene hasta la fecha, ya que no calculé ni por asomo que iba a ser tan complicado realizar ciertas cosas. Por ejemplo, la comunicación con el módulo de radio fue algo que me comió muchísimo tiempo, aunque como se suele decir, de todo se aprende, y de este proyecto he aprendido mucho.

Tampoco había trabajado nunca con una librería gráfica para hacer algo serio, así que ha sido todo un reto meterme de cabeza con QT y el hecho de tratar de separar la lógica de la aplicación de la GUI convenientemente.

Como ya se ha visto anteriormente, el proyecto se ha separado en dos iteraciones principales, y ambas han sido un reto. La primera por el desconocimiento de la biblioteca QT y los problemas que me ha dado la librería de VLC con la reproducción, hasta el punto de que me planteé utilizar una librería distinta para la reproducción de audio (de hecho, no lo descarto en el futuro, ya que la librería tiene bugs y trabas considerables). También tuve que reescribir un pequeño trozo de la fachada en Python que proporciona la librería para poder acceder a algunos métodos que teóricamente estaban obsoletos pero de los que no encontraba equivalente. En definitiva, cada paso en este aspecto se ha hecho un mundo, y cada pequeña funcionalidad supone grandes horas de trabajo (por ejemplo, para algo que parece tan tonto como mostrar el segundo por el que va la reproducción de una canción he tenido que utilizar hilos y concurrencia).

En la segunda iteración, la correspondiente a la radio, ya contaba con la

experiencia de conocerme las librerías gráficas, pero implementar la comunicación con la radio ha sido muy difícil, ya que no había apenas información sobre como leer y escribir en ella, y los ejemplos de código que me encontraba parecían carentes de toda lógica hasta que di con la clave tras casi una semana atascado con ello.

En resumen, como digo, detrás de cada pequeña funcionalidad hay un montón de horas de trabajo ya que son cosas difíciles de implementar, y al final en cuanto a código ha quedado un proyecto de un tamaño considerable.

No obstante, he aprendido muchísimas cosas en el desarrollo del proyecto, y además, aunque quizás me hubiese gustado implementar alguna cosa extra, he alcanzado los objetivos que perseguía, además de haber conseguido un diseño visualmente atractivo, con lo cual estoy contento con el resultado final. La aplicación a la hora de utilizarla es simple, dado que se va a utilizar mientras estás conduciendo y hay que reducir las distracciones, pero funciona bastante bien y con bastante fluidez, por lo que estoy satisfecho con el resultado alcanzado.

7.2. Líneas de trabajo futuras

En cuanto a las líneas de trabajo futuras, básicamente este proyecto puede ser tan grande como quiera el que lo desarrolle. No obstante, hay varias funcionalidades más que quedarían muy bien, pero que en un TFG, con el tiempo limitado que hay, tampoco se pueden implementar.

Una de esas funcionalidades es la reproducción de vídeo, lo cual sería relativamente sencillo dado que la librería VLC da soporte a ello.

Otra funcionalidad que se puede realizar es integrar un navegador GPS. *Navit* es un navegador GPS de código abierto, el cual permite ser embebido en aplicaciones que utilicen QT, así pues se podría utilizar embebido en este proyecto, dando la sensación de que ambas aplicaciones son solamente una.[7]

Por último, también creo que estaría bien añadir soporte a la internacionalización, permitiendo elegir entre varios idiomas almacenados en diferentes documentos XML.

Creo que estas funcionalidades, junto a algunas mejoras en las actuales, como permitir organizar de otra manera las canciones en el modo de reproducción de ficheros de audio, harían del sistema uno muchísimo más completo. No obstante, como ya digo, por limitaciones obvias de tiempo es imposible implementar tantas funcionalidades, pero las que perseguía este proyecto, que son la reproducción de ficheros de audio y la sintonización de la radio, las cumple a la perfección.

Bibliografía

[1] Andrei Istodorescu. Car pc project, 2013. [Online; Consultado el 03 de Junio de 2017] http://engineering-diy.blogspot.com/2013/08/car-pc-projectaugust-2013-update.html.

[2] Chad Lung. The 10 minute guide to the observer pattern in python, 2014. [Online; Consultado el 13 de Marzo de 2017] http://www.giantflyingsaucer.com/blog/?p=5117.

[3] GitHub. Github features, 2017. [Online; Consultado el 02 de Junio de 2017] https://github.com/features.

[4] iCarus. icarus - intellectual car pc, 2015. [Online; Consultado el 03 de Junio de 2017] http://i-carus.com/.

[5] Isaac PE. Todo sobre los gpio raspberry pi, 2015. [Online; Consultado el 27 de Mayo de 2017] http://comohacer.eu/gpio-raspberry-pi/.

[6] lxml. lxml - processing xml and html with python, 2017. [Online; Consultado el 03 de Junio de 2017] http://lxml.de/.

[7] Navit. About navit, 2007. [Online; Consultado el 03 de Junio de 2017] http://www.navit-project.org/.

[8] PyCar. Pycar, 2004. [Online; Consultado el 03 de Junio de 2017] http://www.pymedia.org/pycar/index.html.

[9] Python Software Fundation. Rpi.gpio, 2017. [Online; Consultado el 02 de Junio de 2017] https://pypi.python.org/pypi/RPi.GPIO.

[10] Python Software Fundation. smbus2, 2017. [Online; Consultado el 02 de Junio de 2017] https://pypi.python.org/pypi/smbus2/0.1.2.

[11] Raspberry Pi Forums. Si4703 fm tuner eval board working with raspi, 2015. [Online; Consultado el 27 de Abril de 2017] https://www.raspberrypi.org/forums/viewtopic.php?t=28920&p=637590.

[12] Silicon Labs. Broadcast fm radio tuner for portable applications, 2009. [Online; Consultado el 27 de Abril de 2017] http://www.silabs.com/documents/public/data-sheets/Si4702-03-C19.pdf.

[13] Silicon Labs. Si4700/01/02/03 programming guide, 2009. [Online; Consultado el 27 de Abril de 2017] http://www.silabs.com/documents/public/application-notes/AN230.pdf.

[14] Stack Overflow. how code a image button in pyqt?, 2010. [Online; Consultado el 22 de Febrero de 2017] https://stackoverflow.com/questions/2711033/how-code-a-image-button-in-pyqt.

[15] Stack Overflow. How do i list all files of a directory?, 2010. [Online; Consultado el 08 de Marzo de 2017] http://stackoverflow.com/questions/3207219/how-to-list-all-files-of-a-directory.

[16] Stack Overflow. Creating a singleton in python, 2011. [Online; Consultado el 08 de Marzo de 2017] http://stackoverflow.com/questions/6760685/creating-a-singleton-in-python.

[17] Stack Overflow. How do i switch layouts in a window using pyqt?? (without closing/opening windows), 2014. [Online; Consultado el 07 de Marzo de 2017] https://goo.gl/Dak6YS.

[18] Stack Overflow. Pyqt qlistwidget custom items, 2014. [Online; Consultado el 08 de Marzo de 2017] http://stackoverflow.com/questions/25187444/pyqt-qlistwidget-custom-items.

[19] Stefan Behnel. The lxml.etree tutorial, 2017. [Online; Consultado el 15 de Mayo de 2017] http://lxml.de/tutorial.html#elements-carry-attributes-as-a-dict.

[20] TexMaker. Texmaker - the universal latex editor, 2017. [Online; Consultado el 02 de Junio de 2017] http://www.xm1math.net/texmaker/.

[21] tutorialspoint. Python multithreaded programming, 2009. [Online; Consultado el 21 de Marzo de 2017] https://www.tutorialspoint.com/python/python_multithreading.htm.

[22] VideoLan. libvlc, 2017. [Online; Consultado el 02 de Junio de 2017] https://wiki.videolan.org/LibVLC/.

[23] Wikipedia. Gpio, 2016. [Online; Consultado el 27 de Mayo de 2017] https://es.wikipedia.org/wiki/GPIO.

[24] Wikipedia. Vnc, 2016. [Online; Consultado el 02 de Junio de 2017] https://es.wikipedia.org/wiki/VNC.

[25] Wikipedia. Hardware libre, 2017. [Online; Consultado el 27 de Mayo de 2017] https://es.wikipedia.org/wiki/Hardware_libre.

[26] Wikipedia. I2c, 2017. [Online; Consultado el 27 de Mayo de 2017] https://es.wikipedia.org/wiki/I%C2%B2C.

[27] Wikipedia. Pycharm, 2017. [Online; Consultado el 02 de Junio de 2017] https://en.wikipedia.org/wiki/PyCharm.

[28] Wikipedia. Python, 2017. [Online; Consultado el 02 de Junio de 2017] https://es.wikipedia.org/wiki/Python.

[29] Wikipedia. Qt (biblioteca), 2017. [Online; Consultado el 02 de Junio de 2017] https://es.wikipedia.org/wiki/Qt_(biblioteca).

[30] Wikipedia. Raspberry pi, 2017. [Online; Consultado el 27 de Mayo de 2017] https://es.wikipedia.org/wiki/Raspberry_Pi.

[31] Wikipedia. Secure shell, 2017. [Online; Consultado el 03 de Junio de 2017] https://es.wikipedia.org/wiki/Secure_Shell.

[32] Wikipedia. Staruml, 2017. [Online; Consultado el 02 de Junio de 2017] https://en.wikipedia.org/wiki/StarUML.

[33] ZenHub. Zenhub - agile project management within github, 2017. [Online; Consultado el 02 de Junio de 2017] https://www.zenhub.com/.