Introducción

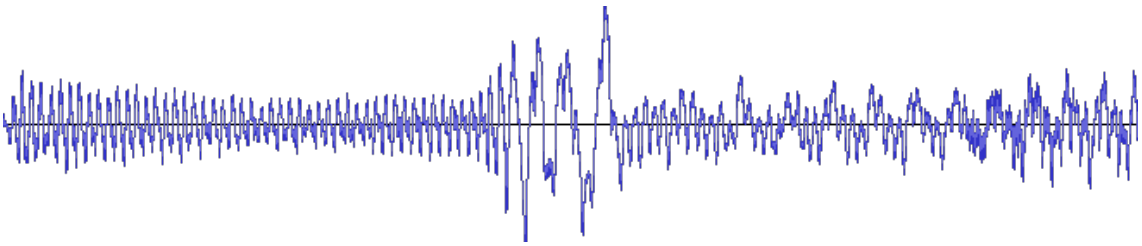
Este proyecto consiste en la realización de un software que realice la reproducción de un fichero de audio en formato raw de 8 bits sin signo, en mono y a 44.1Khz de muestreo.

La salida del audio se realiza mediante el uso de un Buzzer sin filtro, es importante que el Buzzer que utilicemos este limpio no lleve componentes que filtren la señal o no podremos reproducir el audio de forma correcta.

La idea del proyecto fue tener conocimiento de la parte de audio para incorporarlo a una mini demo realizada para Raspberry Pi Zero y sin OS, desarrollado todo íntegramente en ensamblador.

Para situarnos vamos a detallar un poco como se representa un sonido en formato digital y como podemos enfocar su reproducción.

Un sonido consiste en un conjunto de vibraciones que se propagan por un medio, por ejemplo, aire, agua, etc.



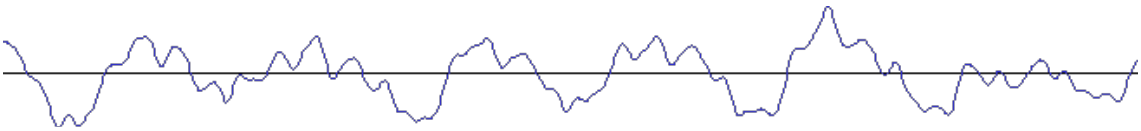
Este conjunto de vibraciones es algo continuo, no es digital es analógico, para llevarlo a nuestro mundo digital deberemos:

- Definir una frecuencia de muestreo que debe ser el doble de la frecuencia que queremos cubrir 22KHz 44KHz, etc.
- Establecer un rango de valores para determinar/cuantificar esa vibración en un momento dado 8 bits, 16 bits etc.
- Muestrear n canales que nos identificarán como se percibe ese sonido de manera espacial mono, estéreo, 2.1, 5.1, etc.

Para digitalizar la señal deberemos convertirla a un conjunto de valores 0 y 1 que nos representaran la señal en el tiempo.

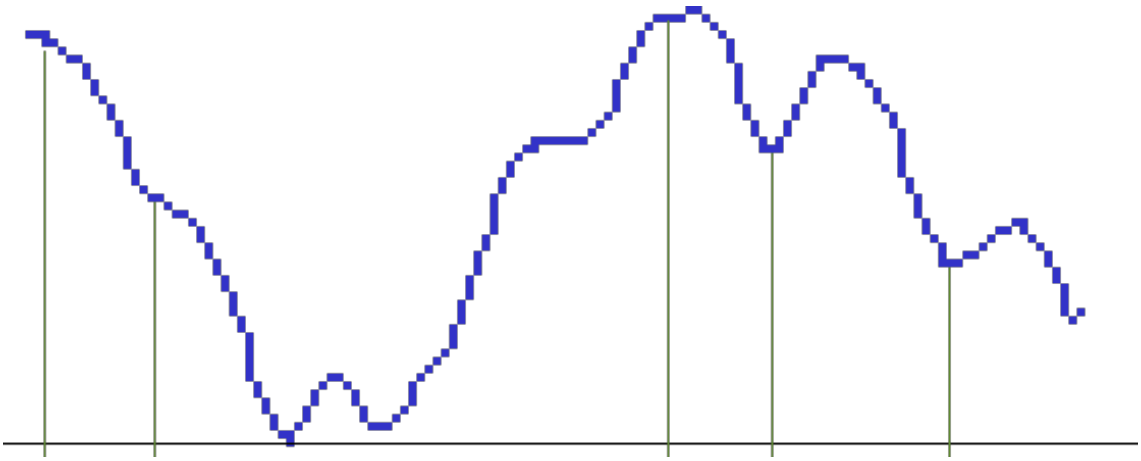
Copyright Iván Cerra De Castro 2021

Para ver el proceso primero ampliaremos un poco la señal



Si cogemos un rango de la señal

Ahora la ampliamos



Solo tenemos que ir cogiendo valores con la frecuencia deseada, por ejemplo, para 8 bits iríamos asignando un valor de vibración entre 0 y 255.

Muestreando la señal de esta manera tendríamos el sonido en formato digital, lo tendríamos digitalizado. Ahora tocaría reproducirlo por el buzzer. Pero tenemos un pequeño problema, el buzzer se conecta a la Raspberry en un puerto GPIO, que puede tener valor alto (1) o bajo (0).

¿Cómo lograremos pasarle valores, por ejemplo, entre 0 y 255 al buzzer?

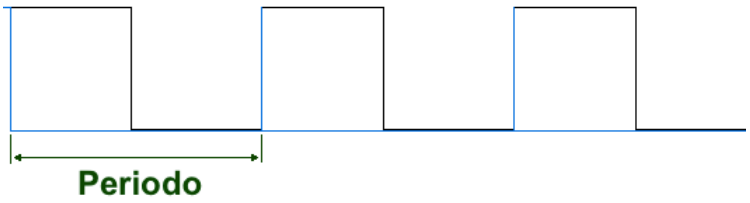
Hay varias formas para transmitir información por los pines de la Raspberry, en nuestro caso vamos a utilizar PWM (Pulse Width Modulation) modulación por ancho de pulso. Esto consiste en transmitir la información variando el tiempo que el pin está en alto (1) durante su periodo, para ello es necesario primero que la señal vaya acompasada a un reloj (frecuencia). Vamos a ver con más detalle PWM.

Copyright Iván Cerra De Castro 2021
https://www.ivancerra.com

PWM – Pulse Width Modulation

Como su propio nombre indica se trata de una técnica para enviar información, modulando el ancho del pulso en el periodo.

Una señal digital transmitida con una frecuencia (Periodo) sería un tren de ondas cuadradas de la siguiente forma.

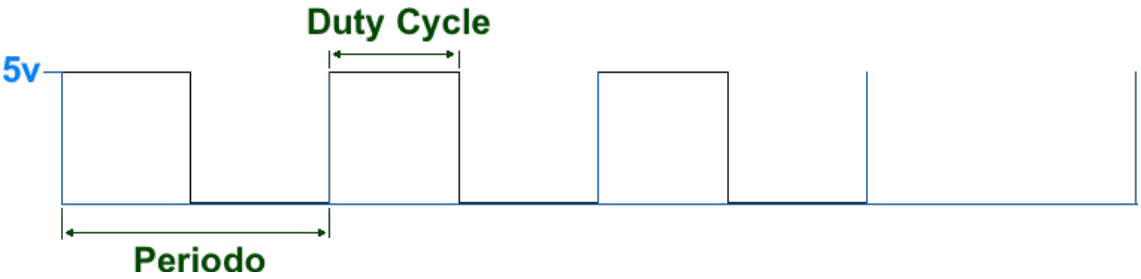


Para continuar con la explicación debemos definir unos conceptos más:

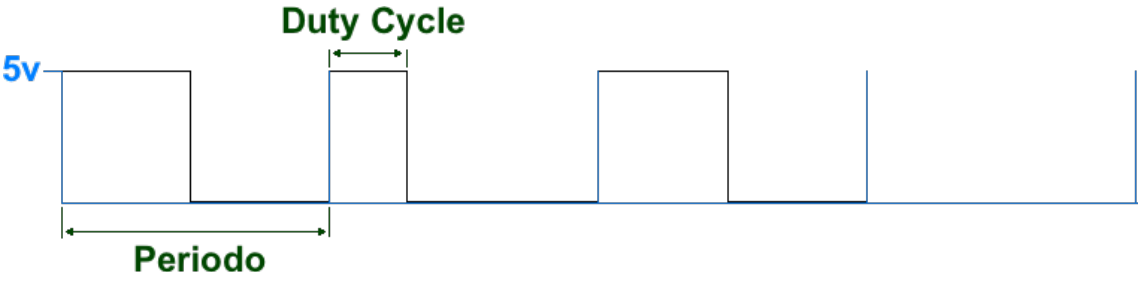
- Tiempo que la señal está en alto tON.
- Tiempo que la señal está en bajo tOFF.

Obviamente la suma tON + tOFF = Periodo. Si hacemos variar el tON lograremos variar el ancho del pulso, recordamos además que para que todo siga en orden tOFF = Periodo – tON.

Definidos estos dos conceptos, aparece uno nuevo, el porcentaje de tiempo o cociente entre el tON y el periodo, a este cociente se le denomina duty cycle, y será el portador de la información.



Variando el duty cycle en cada periodo podemos transmitir la información.



Copyright Iván Cerra De Castro 2021
https://www.ivancerra.com

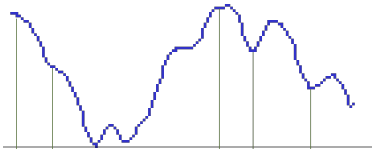
El problema del duty cycle es que sigue siendo un valor continuo en un intervalo, lo cual en el mundo digital no se lleva muy bien, así que toca cuantificarlo, a esa cuantificación se le denomina rango. Por ejemplo, para poder "transportar" 8 bits tendremos que aplicar un rango de 256 valores.

Como resumen podemos decir que los valores importantes para tener en cuenta en PWM son:

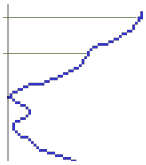
- Frecuencia o periodo.
- Rango.
- Duty cycle.

Con estos valores ya podemos transmitir la información que estimemos oportuna por el pin GPIO.

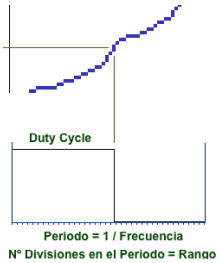
Una vez comprendido el funcionamiento de PWM, volvemos para ver como reproducir el sonido. Nos quedamos en la parte de tener los valores de la onda.



Si rotamos el grafico



Cogemos solo un valor



Vemos como sale casi de forma inmediata la forma de transmitir los valores al buzzer con PWM.

Copyright Iván Cerra De Castro 2021

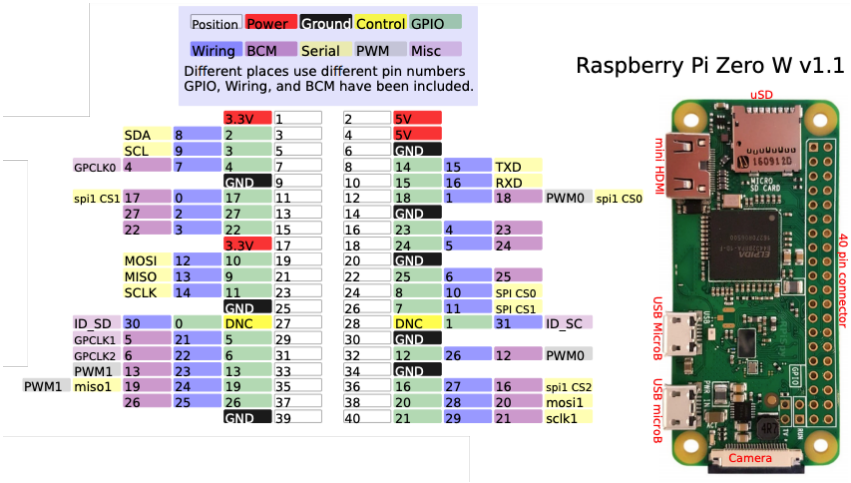
Realización del proyecto en Raspberry

Hardware

Para llevar a cabo el proyecto necesitaremos:

- Raspberry Pi, en mi caso he optado por el modelo más barato una Raspberry Pi Zero W (Para otros modelos los puertos pueden ser diferentes al igual que las direcciones de memoria, deberéis consultar el manual del SOC).
- Buzzer.
- Protoboard o Breadboard (Opcional, en mi caso he preferido montarlo en una protoboard por si luego se quieren añadir más componentes, pero no es necesario se puede conectar el Buzzer a los pines de salida directamente).

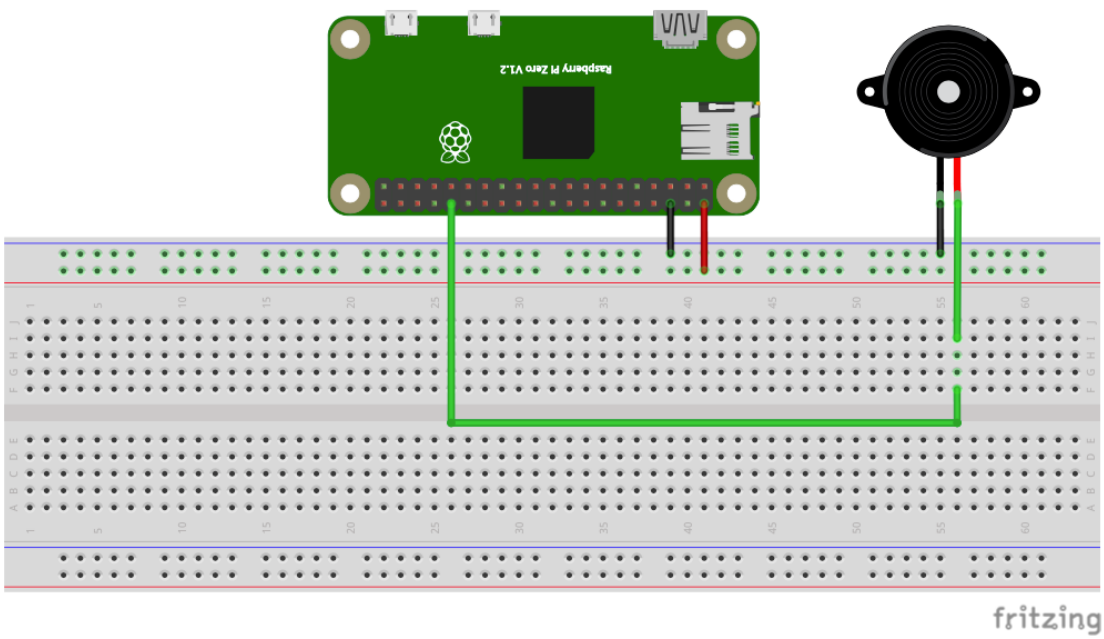
Antes de comentar un poco el código vamos a ver el montaje. Lo primero es ver un esquema de los pines de nuestra Raspberry para seleccionar un pin que tenga salida PWM por hardware.



En este caso el pin que vamos a seleccionar es el GPIO 12 que vemos tiene salida PWM por el canal 0 (Hay dos canales el 0 y el 1).

Una vez seleccionado el pin realizamos el montaje en la protoboard.

Copyright Iván Cerra De Castro 2021



Software

Después de elegir el hardware queda por tomar una importante decisión, lo vamos a realizar para la Raspberry con un OS, o bien lo hacemos de cero, sin OS (Bare metal).

Sobre Raspbian se nos dan más facilidades para programar, podemos usar el debugger, podemos usar las funciones del OS como printf, etc. Pero, para acceder al hardware deberemos hacerlo abriendo ficheros del sistema con permisos de root y pierde un poco el encanto de hacer algo de cero, además deberemos iniciar la Raspberry en el OS antes de ejecutar nuestra aplicación.

Por lo tanto, he optado en un desarrollo desde cero y en ensamblador,… Creo que lo siguiente sería en código máquina directamente. El código es muy sencillo y lo he intentado comentar al detalle.

La aplicación es muy sencilla y breve, la podemos dividir en estos puntos funcionales:

1. Adaptación del fichero de sonido Raw para gestionarlo con DMA.
2. Configuración de la fuente de reloj para el pin GPIO con PWM.
3. Configuración de PWM.
4. Configuración de DMA.
5. Activación de DMA.

Lo primero que haremos es adaptar el formato del fichero de audio, el fichero original viene en un byte por cada valor de vibración de sonido, pero DMA transfiere los datos en bloques de 32 bits, por lo tanto, tendremos que poner nuestros valores de 8 bits encapsulados en palabras de 32 bits.

Copyright Iván Cerra De Castro 2021

El paso siguiente es la configuración del pin GPIO para usar con PWM, también aprovecharemos para configurar el led de actividad (Ojo que el led de actividad se activa en bajo).

Para el pin 12 (PWM) deberemos ver en el registro de funciones para los pines, que valor necesitamos asignarle.

Las funciones disponibles para un pin GPIO.

Código (Binario)	Función del pin
000	GPIO Pin X es una entrada
001	GPIO Pin X es una salida
100	GPIO Pin X toma función alternativa 0
101	GPIO Pin X toma función alternativa 1
110	GPIO Pin X toma función alternativa 2
111	GPIO Pin X toma función alternativa 3
011	GPIO Pin X toma función alternativa 4
010	GPIO Pin X toma función alternativa 5

Y mirando en la guía del SOC Broadcom BCM2835

	PWM 0	PWM 1
GPIO 12	Alt Fun 0	-
GPIO 13	-	Alt Fun 0
GPIO 18	Alt Fun 5	-
GPIO 19	-	Alt Fun 5
GPIO 40	Alt Fun 0	-
GPIO 41	-	Alt Fun 0
GPIO 45	-	Alt Fun 0
GPIO 52	Alt Fun 1	-
GPIO 53	-	Alt Fun 1

También podemos observar en dicho manual que el DMA mapeado a PWM es el canal 5.

Configurados los pines GPIO, vamos a realizar la parametrización de la fuente de reloj para GPIO que usaremos para PWM, para PWM0 tenemos:

- Asignar una frecuencia de reloj.
- Definir el rango de PWM.

Tenemos que conseguir 44.1KHz a 8bits de resolución. De nuevo mirando la documentación del SOC vemos que podemos usar para los pines GPIO divisores sobre distintas entradas de reloj, de todas ellas nos vamos a centrar en dos, de las cuales conocemos su frecuencia:

- Clock Source Oscillator, INTOSC 19.2 MHz
- Clock Source PLLD, 500 MHz

Sobre estas fuentes (Hay más pero su frecuencia puede ser distinta en modelos de Raspberry y cogemos estas que ofrecen un buen rango) podemos aplicarle un divisor

Copyright Iván Cerra De Castro 2021
https://www.ivancerra.com

entero de 12 bits, es decir un número entero entre 1 y 4095. También podríamos añadirle a ese divisor una parte fraccional de 10 bits, pero es algo más complicado y en este caso con la parte entera nos arroja un buen resultado.

El calculo del divisor que hago yo, quizás es un poco extraño, pero no he visto en ningún sitio como hacer un buen calculo de estos datos, así que os pongo como yo lo hago.

Divido la frecuencia del reloj por la frecuencia que quiero tener en este caso (Luego explicaré porqué) cojo mi fuente de reloj PLLD de 100 MHz. Ahora calculo el ancho de banda que necesito 0.0441 MHz * 256 Valores = 11.2896 MHz esto seria mi ancho de banda en MHz, hago la división y sale 500 MHz / 11.2896 MHz = 44.28 voy a coger la parte entera superior es decir 45 que será mi divisor entero, ahora tengo que ver que rango me quedara con esa parte entera 500 MHz / 45 = 11.11 entonces mi rango será 11.11 / 0.0441 = 252 Valores posibles. Pierdo un poco con respecto a los 256.

Por tanto, seleccionamos la fuente del reloj PLLD y de divisor entero 45 ¿Por qué hemos seleccionado como fuente de reloj PLLD de 100 MHz y no hemos elegido INTOSC de 19.2 MHz (El cual suelen usar en la mayoría de ejemplos que he visto)? Si os fijáis bien si hubiéramos seleccionado 19.2 MHz encontrar un divisor entero con cuya división se aproxime más a nuestro ancho de banda es más difícil, lo normal podemos considerar la regla que cuanto mayor sea el dividendo más fácil es aproximarnos con un divisor a un resultado próximo.

Configurado el reloj interno para PWM, la siguiente configuración delicada a realizar es el rango de PWM, como vimos anteriormente nuestro rango será de 252. Como experimento os recomiendo que probéis a cambiar el rango por ejemplo poned 64, y en otro caso poned 512 ¿Veis que pasa? Es importante poner el rango exacto que nos ha resultado en nuestros cálculos, por eso debemos realizar siempre los cálculos para que quede lo más próximo el rango resultante a los valores cuantificados.

Ahora queda la parte más sencilla que es definir el bloque de control y el DMA que queremos usar, como punto a fijarnos es el enlace del siguiente bloque de control a nuestro mismo bloque de control para que se quede en un bucle sin fin el sonido.

Espero que os haya quedado claro este pequeño código que reproduce un audio raw con una Raspberry y un buzzer sin necesidad de OS.

Copyright Iván Cerra De Castro 2021
https://www.ivancerra.com

Bibliografía

Universidad de Málaga – Prácticas de ensamblador – Villena Godoy, Asenjo Plaza, Corbera
https://riuma.uma.es/xmlui/handle/10630/10214

University Of Cambridge - Raspberry OS
https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/introduction.html

Tutoriales de Peter Lemon
https://github.com/PeterLemon/RaspberryPi

Raspberry Pi como generador de frecuencias
http://electronicayciencia.blogspot.com/2017/05/raspberry-pi-como-generador-de.html

Además de estas fuentes he consultado y descargado la guía del SOC Broadcom 2835.

Copyright Iván Cerra De Castro 2021
https://www.ivancerra.com