EEE3096S: Practical 4 Report
JFFMAH001 NDKNIC001

Introduction

In this practical we were required to play an audio file provided to us using the MCP4911 DAC that communicated with the Pi using SPI communication protocol. The audio needed to be able to be played, paused and stop. This was achieved by using two buttons that interrupted the communication between the Pi and the DAC. One button was used to pause the playback of the audio – when the button is clicked the first time the playback stops and when pressed again the playback continues. The other button was used to completely stop the playback and end the program.

Since audio is hard time constrained, it was necessary to use circular buffering so that as one half of the buffer containing the audio file was played, the other half of the buffer would be loading. This was done to minimise delay.

SPI Communication using Wiring Pi

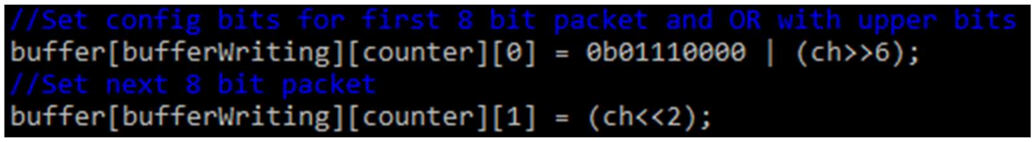
Initialisation, including clock speed calculation

The SPI communication was set up by calling `wiringPiSPISetup(SPI_CHAN, SPI_SPEED)`. SPI_CHAN is the channel that was used for the SPI communication which was set to 0. SPI_SPEED is the clock speed that was initialised to twice the speed of the audio file i.e 256kHz.

Send data

Sending data to the DAC over SPI was done by calling `wiringPiSPIDataRW (SPI_CHAN,data, 2).` This writes data that has a length of two over the channel indicated. The data in this case is the buffer array e.g **buffer[bufferReading][buffer_location]** that contains two words each having a length of 8 bits.

The DAC takes 10 bits but the data from the file given to us was 8 bits, therefore when the data was sent to the DAC we had to ensure that the data being written to the DAC contained the 8 bits of data from the buffer as well as the necessary control bits and the ignored bits to make up the two bytes. In order to do this, we manipulated the data by bit shifting the first byte of data 6 units to the left and we OR it with the control bits. The second byte of data was bit shifted to the right by 2. This can be seen in Figure 1 below.



```
//Set config bits for first 8 bit packet and OR with upper bits
buffer[bufferWriting][counter][0] = 0b01110000 | (ch>>6);
//Set next 8 bit packet
buffer[bufferWriting][counter][1] = (ch<<2);
```

Figure 1: Manipulation of data in buffer

The timing diagrams below show how the DAC reads data over SPI. In Figure 3 the values of A - H correspond to the values from the file.

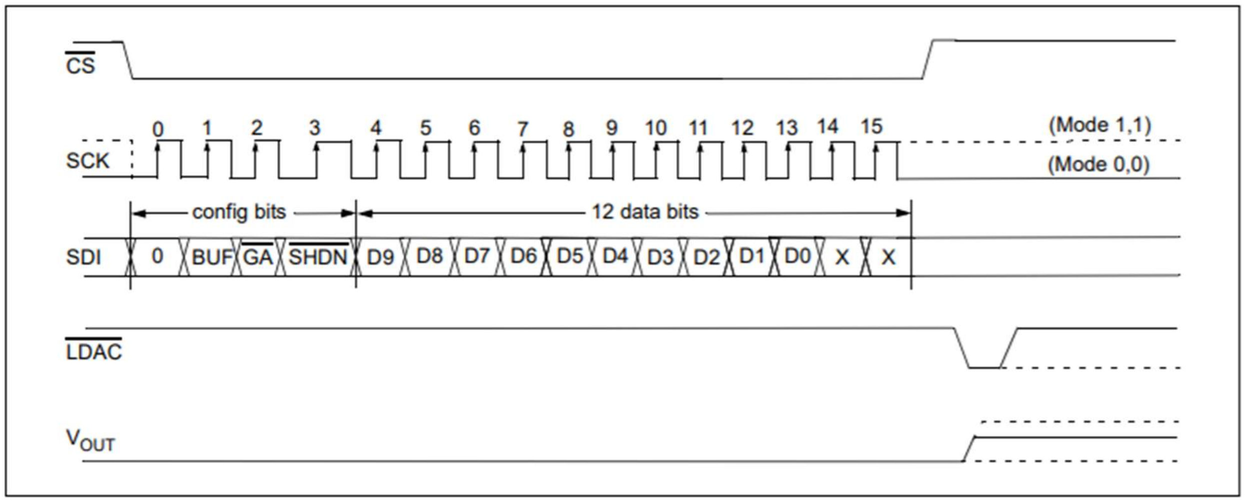


Figure 2: Timing diagram for the MCP4911 from the data sheet

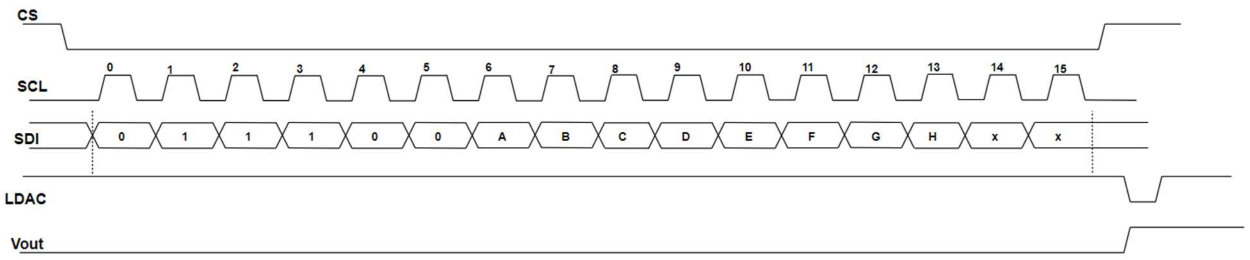


Figure 3: Timing diagram showing the data that was sent to the DAC from the Pi.

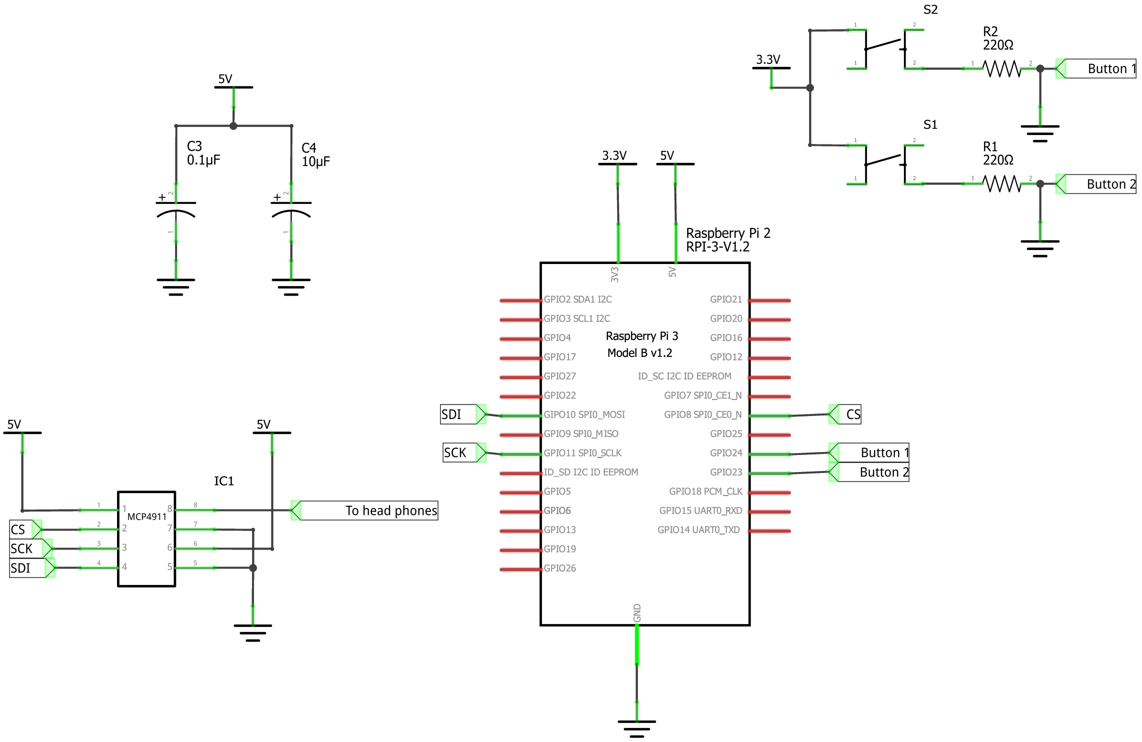
Importance of Real Time Constraints

Real time constraints are important because every real-time system has a set of timing constraints that it must meet. In the case of this assignment, since audio is hard time constrained, it was necessary to use “circular buffering” so that as one half of the buffer containing the audio file was played, the other half of the buffer would be loading. This was done to ensure that the data being sent to the DAC was continuous to ensure the smooth playing of the audio file.

The Raspberry Pi (under Raspbian) is unable to implement real time constraints since it uses the Network Time Daemon Protocol to get its time. This protocol relies on the internet to get its time and therefore the Raspberry Pi is unable to implement real time constraints.

Circuit Diagram

The circuit diagram can be found on the next page.



5V
C3
0.1µF
C4
10µF
3.3V
S2
R2
220Ω
Button 1
S1
R1
220Ω
Button 2
3.3V
5V
Raspberry Pi 2
RPI-3-V1.2
3V3
5V
GPIO2 SDA1 I2C
GPIO3 SCL1 I2C
GPIO4
GPIO17
GPIO27
GPIO22
GIPO10 SPI0_MOSI
GPIO9 SPI0_MISO
GPIO11 SPI0_SCLK
ID_SD I2C ID EEPROM
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
Raspberry Pi 3
Model B v1.2
GPIO21
GPIO20
GPIO16
GPIO12
ID_SC I2C ID EEPROM
GPIO7 SPI0_CE1_N
GPIO8 SPI0_CE0_N
GPIO25
GPIO24
GPIO23
GPIO18 PCM_CLK
GPIO15 UART0_RXD
GPIO14 UART0_TXD
GND
SDI
SCK
CS
Button 1
Button 2
5V
5V
IC1
MCP4911
CS
SCK
SDI
To head phones

fritzing