Practical 4: SPI and Threading

Ronak Mehta[†] and Vikyle Naidoo[‡]
EEE3096S - 2019
University of Cape Town
South Africa
[†]MHTRON001 [‡]NDXVIK005

***Abstract*—This practical was aimed at understanding SPI (Serial Peripheral Interface) and Threading using the Wiring Pi configuration**

I. Introduction

It is usually impossible to perform hard time operations (i.e. events that need to happen at a very specific time with no variations) like audio processing in a Raspberry Pi as it uses a Linux based Operating System. This practical was hence designed to play audio over SPI and into the DAC (Digital to Analog Converter) to improve its audio quality. It made use of a single array split into two parts mimicking a circular buffering procedure (a technique which ensures samples are ready to play as soon as they are requested). This was implemented through playing one part of the single array while the other part loaded data.

This practical started by first determining the rate at which the audio was played by setting the SPI Clock using the SPIClock formula discussed later in the report. The audio file provided was of 16kHz and 8 bits, which was read in each 8 bit character and sent over to the DAC as a thread with high priority using the *pthread.h* and *sched.h* attributes. It also used two push buttons to act as interrupts to pause playback and stop playback. Finally, the DAC output was connected to a speaker for testing.

II. SPI using WiringPi

The SPI bus is a synchronous data bus which uses separate lines for data and clock. These lines are SCLK (Clock generated by master), MOSI (Master Out Slave In), MISO (Master In Slave Out) and !SS (Slave Select active low). Wiring Pi includes a library which makes it easier to use Raspberry Pis on-board SPI interface.

A. Initialization:

- To use the Wiring Pi GPIO library, *#include wiringPi.h* header file must be included in the program.
- To use the SPI library in Wiring Pi, *#include wiringPiSPI.h* header file must be included in the program.
- To initialize the SPI system with a given device identifier, this command is used: *int wiringPiSPISetup(int channel, int speed);*

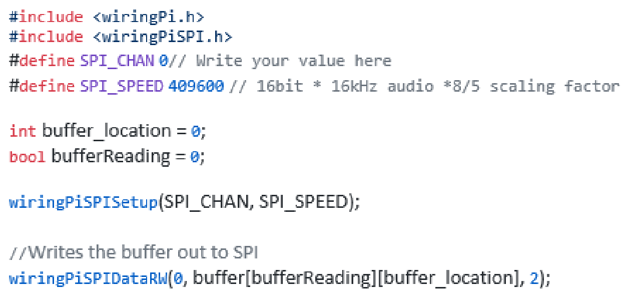
To set the SPI clock and get the SPI Clock speed, the following calculation was made:

$$SPIClock = SampleRate * Width * No.of.Channels * 8/5$$

$$SPIClock = 16kHz * 16bit * 1 * 8/5 = \textbf{409600 Hz}$$

B. Send Data:

To send data to other devices, it is required to use the Read/Write command. This performs a simultaneous read/write transaction over the selected SPI bus. Data that was in the buffer would be overwritten by data returned from the SPI bus. The command for this is:
*int wiringPiSPIDataRW (int channel, unsigned char*data, int len);*



```
#include <wiringPi.h>
#include <wiringPiSPI.h>
#define SPI_CHAN 0// Write your value here
#define SPI_SPEED 409600 // 16bit * 16kHz audio *8/5 scaling factor

int buffer_location = 0;
bool bufferReading = 0;

wiringPiSPISetup(SPI_CHAN, SPI_SPEED);

//Writes the buffer out to SPI
wiringPiSPIDataRW(0, buffer[bufferReading][buffer_location], 2);
```

Fig. 1: Some commands used for this practical indicating the initialization and sending of data

III. Real Time Constraints

Real-time constraints are restrictions on the timings of events, such that they should occur at a very specific time and with no variations. Some importances of real time constraints include:

1. Systems having real time constraints would want to stop an already running instruction in case of a rise of an immediate situation which needs to be taken care of. For example, if one is working with a temperature sensor and monitoring its behavior, it is important to call for an interrupt if there is a sudden temperature rise or fall which might affect the purpose of the system
2. It makes a system flexible and allows little time to transition between different tasks.
3. Real time constraints also allow maximum output because there is little or no downtime. Thus, systems having

real time are suitable for applications that need to run constantly.

The Raspberry Pi (under Raspbian) is unable to implement the real time constraint because it uses a Linux based Operating system and not a RTOS (Real Time Operating System). Raspbian does not support real time because it is connected to the Network Time Protocol (NTP) server and expects the device to be constantly connected to the internet and thus has no Real Time Clock (RTC) to monitor and implement real time.

IV. Circuit Diagram

Below is the schematic detailing the DAC connections with the Raspberry Pi. This practical also made use of push buttons as input peripherals.

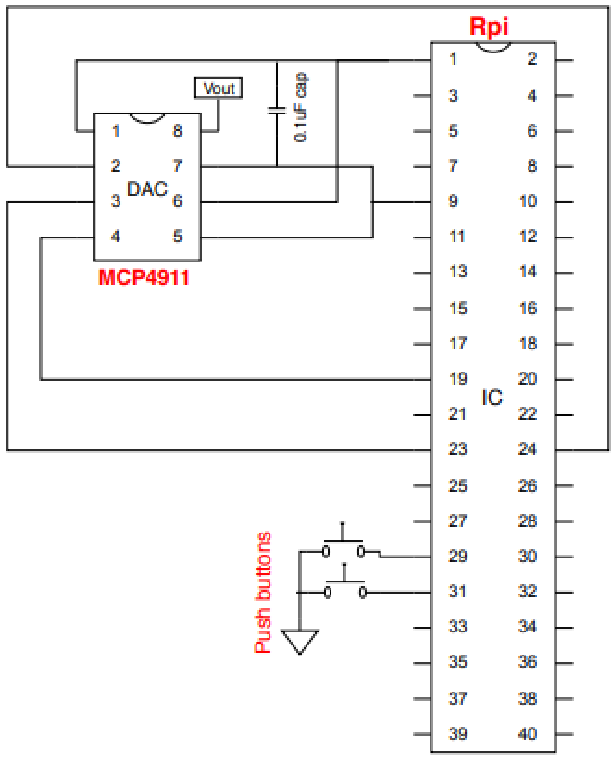


Fig. 2: Schematic detailing DAC connections with RPi

References

[1] SPI WiringPi Details
http://wiringpi.com/reference/spi-library/

[2] Vula briefing
Prac4 Document

[3] Keegan Github
https://github.com/kcranky/EEE3096S