UNIVERSITY OF CAPE TOWN

DEPARTMENT OF ELECTRICAL ENGINEERING

CYBER-PHYSICAL SYSTEMS (EEE3095S)

PRAC 4: SPI and Threading Report

Name Student 1: **WILLIE MACHARIA**
Name Student 2: **SANDILE SITHOLE**

Student Number 1: **MCHWIL006**
Student Number 2: **STHSAN007**

We did not demonstrate in the lab and we were instructed to do a YouTube video. This is the link: https://youtu.be/CL_fXIeDJU8

EEE3096S Prac 4 - SPI and Threading Report

Demonstration Youtube Video is here: https://youtu.be/CL_fXIeDJU8

Introduction

This practical serves to use SPI and threading to communicate between a Raspberry Pi and an DAC(Digital Analogue Converter) component namely the MCP4911. MCP4911 has 10 bits DAC resolution and one SPI Channel. Since audio jack output in raspberry pi is not good quality, the DAC tries to better the quality of the sound. The audio file is loaded in the raspberry pi and it is played and its output is listened through the output of the DAC. The application is written in C++, using a provided template as a base. WiringPi is used as a library for implementation for the SPI simulation in code.

SPI communication using Wiring Pi

Serial Peripheral Interface(SPI) is full duplex meaning both ends can transmit simultaneously, synchronous (uses a shared clock), serial (Sequential bit transfer), communication protocol (voltages and data format are defined).

WiringPi has a SPI Library to implement in coding. To implement the library initialisation is done first. During initialisation of the SPI Library the following happens;

1. First the SPI Clock is calculated . For this practical :

   SPI_CLOCK = SAMPLERAT E ∗ WIDTH ∗ No.CHANNELS ∗ 8/5
   Sample rate =16,000
   Width of the DAC=10
   NumChannel=1

   SPI_Clock=16000*10*1*8/5=256000Hz
2. Setting up the SPI with both speed and channel. The Raspberry Pi has two channels which is 0 and 1. This practical we chose channel 1.

   In Prac4.h file we have defined the value of the SPI_CHAN and SPI_SPEED as :

   #define SPI_CHAN 1
   #define SPI_SPEED 256000

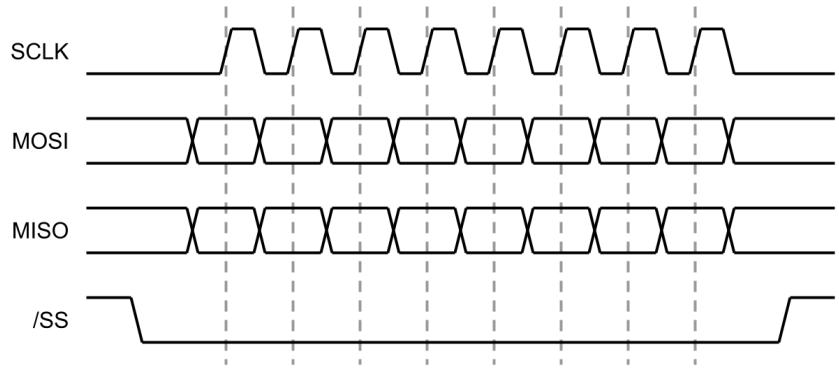
   In Prac4.cpp we called the setup function as shown below:

   wiringPiSPISetup(SPI_CHAN, SPI_SPEED);

   According with the WiringPi documentation, the return value of this function is an integer and if the value returned is -1; then an error has occurred. Thus our main method has an error handling statement to check this as shown below.

```
if(setup_gpio()==-1){
        return 0;
}
```

3. After initialising the SPI, then data needs to be read and written. This is done by calling the **wiringPiSPIDataRW (int channel, unsigned char *data, int len)** function. This function performs a simultaneous write/read transaction over the selected SPI bus. Data that was in your buffer is overwritten by data returned from the SPI bus. Devices such as ADC and DAC converters usually need to perform a concurrent write/read transaction to work since SPI is a full duplex interface where there is MISO(Master In Slave Out and MOSI(Master Out Slave In).
   A timing digram for the SPI Communication is shown below to show how data is sampled in SPI .



Importance of real time constraints

1. Embedded systems with implementation of real time constraints responds to an event or request within a strictly defined time and therefore they can perform deterministic tasks and operations hence predictability is higher. This is part of event responding constrain which means that a system which have implemented real time constraints is efficient in event responding as they can be entrusted to respond to some events as soon they occur while those which do not implement real-time constrains cannot be used in high risk embedded systems where event response is of high priority. This broadens to interrupt handling which needs to be real time as it is part of event responding. The
2. Embedded systems with implementation of real time constraints efficient for task scheduling. A task is a set of instructions that need to be run by the embedded system processor. Embedded systems designers will sometimes prefer to schedule some tasks in their software development hence these embedded systems need to repeatedly sense a number of inputs and then modify outputs based on this new information. Then these systems needs to have implemented real time constrains.
3. Embedded systems with implementation of real time constraints have low latency when responding to events. For example: Raspberry Pi since does not have real time response, the interrupt needs to be checked and debouncing need to be implemented hence there is high latency of event response. This could not be the case if the raspberry pi had the real time constraints implemented in it.

why the Raspberry Pi (under Raspbian) is unable to implement the Real-time Constraints.

The raspberry pi is unable to implement real time constrains since it does not have a real time clock. It uses and relies on NTPD (Network Time Protocol Daemon) to fetch, set and store the date and time. This is problematic as you may not always have an internet connection, and if your Pi doesn't have the correct localisation options, you may end up with the wrong

time due to time zone settings. Since the Raspberry Pi doesn't have an implementation of the real time clock, it therefore by default unable to implement real time constraints. However this challenge can be solved by adding a real time clock. But even after adding a real time clock, the raspberry Pi will cannot generate deterministic timing pulses which can be critical for some uses such as turning motors. But even after doing this , it is still difficult to run multithreaded programs in the pi using the Raspbian operating system. Hence to improve the Raspbian operating system to cater real time constraints is very difficult and so many overheads might come when adding more modules. Hence, it is efficient to use other open source operating systems on the pi to implement real time constrains such as RTEMS(Real Time Executive for Multiprocessor Systems).

Circuit Diagram with changes from the Preprac