Electronics and Computer Science  
Faculty of Physical Sciences and Engineering  
University of Southampton

Lukman Kazeem

William Hazelden  
Muhammad Ariif Rozali  
Fedor Selensky

PPL Pressure Switch Setting App and Communications

Project supervisor: Yoshi Tsuchiya  
Second examiner: Kirk Martinez

A Group Design Project Report submitted for the Award of  
< Degree Title >  
< Degree Title >  
< Degree Title >

< Degree Title >

**Abstract**

Contents

[1 Introduction 5](#_Toc61091902)

[2 System Design 6](#_Toc61091903)

[3 Pressure Switch Device Design 8](#_Toc61091904)

[3.1 Hardware Design 8](#_Toc61091905)

[3.2 Software Design 9](#_Toc61091906)

[4 Pressure Switch App Design 12](#_Toc61091907)

[4.1.1 Android App Design 12](#_Toc61091908)

[5 Pressure Switch Device Build 16](#_Toc61091909)

[5.1 Hardware Build 16](#_Toc61091910)

[5.1.1 Pressure Switch Circuit Overview 16](#_Toc61091911)

[5.1.2 Microcontroller 18](#_Toc61091912)

[5.1.3 Voltage Regulator 19](#_Toc61091913)

[5.1.4 Pressure Sensor Input Circuit 20](#_Toc61091914)

[5.1.5 Digital Output Circuit 25](#_Toc61091915)

[5.1.6 Analogue Output Circuit 26](#_Toc61091916)

[5.1.7 Integrated Pressure Switch Circuit 27](#_Toc61091917)

[5.1.8 Printed Circuit Board (PCB) Layout 28](#_Toc61091918)

[5.1.9 PCB Bill of Materials 29](#_Toc61091919)

[5.2 Software Build 29](#_Toc61091920)

[5.2.1 System Overview 30](#_Toc61091921)

[5.2.2 Setup 31](#_Toc61091922)

[5.2.3 Data Acquisition 32](#_Toc61091923)

[5.2.4 Data Processing 36](#_Toc61091924)

[5.2.5 Output 37](#_Toc61091925)

[5.2.6 Communication 39](#_Toc61091926)

[6 Pressure Switch App Build 45](#_Toc61091927)

[6.1 Android App Build 45](#_Toc61091928)

[6.1.1 The GUI 45](#_Toc61091929)

[6.1.2 NFC Functionality 45](#_Toc61091930)

[6.1.3 Bluetooth Functionality 46](#_Toc61091931)

[6.1.4 The Home Screen 47](#_Toc61091932)

[6.1.5 The Settings Screen 48](#_Toc61091933)

[6.1.6 Navigation 49](#_Toc61091934)

[7 Pressure Switch Device Testing 52](#_Toc61091935)

[7.1 Hardware Testing 52](#_Toc61091936)

[7.1.1 Development and Testing 52](#_Toc61091937)

[7.1.2 Circuit Simulation 53](#_Toc61091938)

[7.1.3 Breadboard Implementation and Testing 54](#_Toc61091939)

[7.1.4 PCB Circuit Testing 55](#_Toc61091940)

[7.2 Software Testing 59](#_Toc61091941)

[7.2.1 Read ADC/ Rolling Average 59](#_Toc61091942)

[7.2.2 Peak/Bottom Hold and Reset 60](#_Toc61091943)

[7.2.3 Zero Clear 61](#_Toc61091944)

[7.2.4 Hysteresis and Comparator Mode 63](#_Toc61091945)

[7.2.5 Bluetooth TX/RX 65](#_Toc61091946)

[7.2.6 Analogue Output 69](#_Toc61091947)

[8 Pressure Switch App Testing 70](#_Toc61091948)

[8.1 Android App Testing 70](#_Toc61091949)

[8.1.1 Unit Testing 70](#_Toc61091950)

[9 Integration 71](#_Toc61091951)

[9.1 PSD Hardware/Software Integration 71](#_Toc61091952)

[9.2 Android PSA & PSD Integration 72](#_Toc61091953)

[9.2.1 Integration Testing 72](#_Toc61091954)

[9.3 IOS PSA & PSD Integration 72](#_Toc61091955)

[10 Management 72](#_Toc61091956)

[11 Conclusion 72](#_Toc61091957)

[12 Further Work 72](#_Toc61091958)

[13 Bibliography 72](#_Toc61091959)

[Appendix A. Project Specification Document 73](#_Toc61091960)

[Appendix B. Gantt Chart 76](#_Toc61091961)

[Appendix C. Full PCB Schematics 77](#_Toc61091962)

[Appendix D. PCB Bill of Materials 78](#_Toc61091963)

[Appendix E. Bluetooth Command Table 80](#_Toc61091964)

# Introduction

 (i) what is the required product and how it will be applied. The name of the organisation of the external partner and names of the contact(s)

(ii) briefly why the product should be created (Societal, environmental, economic (competing technologies and advantages over these?)

(iii) the main field, technology area of your external partner and the rational as to why the external partner wishes for the product to be created and who will exploit it (non-expert/expert, other engineer/research area)

(iv) Clearly identify whether the output required by the external partner is a 'proof of concept', demonstrator or application of an existing technology (for a trade show for instance), a prototype or an elaboration or add-on of/intergration with/ superseeds an existing system

# System Design

The PSD aims to replicate the functionality of a digital pressure switch, using a microcontroller to measure and process 3 input channels (Pressure Sensors). The parameters used within the PSD will be controlled by the User via the PSA, communicating bi-directionally using Bluetooth.The PSA (Android only) and PSD will use NFC for initial pairing to enable communication. Each channel will have its own digital and analogue output signal, which will be dependent on the input pressure of each independent channel.



Figure 3.1.1 - PSD System Architecture

Figure 3.1.1 illustrates the PSD system architecture. The system contains all the requirements outlined within the specification document (Appendix x).

**Pressure Switch and App Specifications**

|  |  |
| --- | --- |
| Supply Voltage | 12-24V DC |
| Current Consumption | 100mA or less |
| Rated Pressure | 0.000 to 1.000MPa |
| Applicable Fluid | Air |
| Number of Pressure Channels | 3 (CH1, CH2 & CH3) |
| Pressure Type | CH1 – Gauge, CH2 – Gauge, CH3 - Absolute |
| **Configurable Switch Parameters for Each Channel** | |
| Switch Mode | Selectable Normally Open (NO), Normally Close (NC) |
| Operation Mode | Selectable Hysteresis & Window Comparator |
| Digital (Switch) Output | 1 PNP output, Maximum Source Current 40mA |
| LED Indicator Configuration | * Green: ON, Red: OFF * Green: OFF, Red: OFF * Always Red LED * Always Green LED |
| **Common Switch Setting for All Channels** | |
| Analogue Output | Selectable 1-5V and 0-10V |
| Response Time | Selectable 2ms, 20ms, 50ms, 100ms, 200ms & 500ms |
| **Communications** | |
| Bluetooth Classic | Range Up to 10m, Bluetooth class 2 |
| Bluetooth Low Energy | Bluetooth 5.0 |
| **App Setting and Features** | |
| Reading Refresh Rate | 200ms, 500ms & 500ms |
| Reading Resolution | 2 & 3 decimal places |
| Pressure Unit | MPa, kPa, Kgf/cm3, Bar, PSI, mmHg, cmHg, inHg |
| Min iOS version |  |
| Min Android version |  |

# Pressure Switch Device Design

## Hardware Design

The pressure switch device will be the physical hardware that is able to read pressure value that are input to the device. This device will also able to output digital and analogue signals based on the pressure it receives through the wired connections to different external device besides performing Bluetooth communication with the pressure switch app for the pressure monitoring and parameter settings. Unlike most of the pressure switch in the market, this pressure switch device can receive three pressure sources at once in one device hence has three pressure sensors channels.

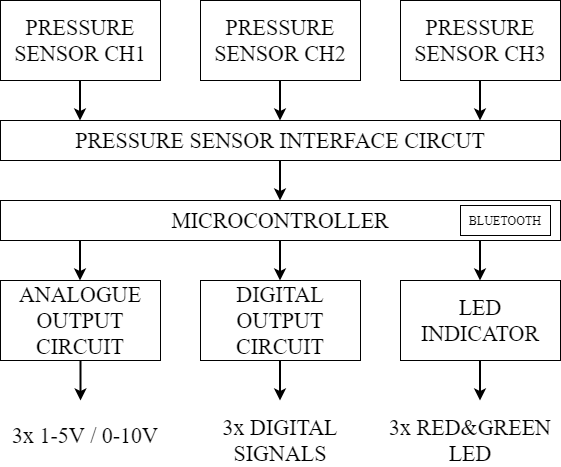


Figure 3.1.1 - PSD Circuit Components

To perform all the stated functions, the switch device consists of several type of circuits that are finally integrated into one Printed Circuit Board (PCB). The components of the circuit are shown in Figure 3.1.1 - PSD Circuit Components. The pressure sensors will output signals that are corresponding to the pressure value that it received. The signals are in small mV signals and come with noise and offset, therefore each of the pressure sensor signals will be amplify and conditioning in the pressure sensor interface circuit. The microcontroller will act as processor that will receive the pressure signals and output the required signals. The analogue and digital output signals generated by the microcontroller will pass through analogue and digital output circuit respectively for the switch to output required signals.

## Software Design

To aid the hardware design of the PSD to create the 3-Channel Pressure Switch product, the microcontroller must be programmed to perform the desired functionality set out in the specification (Appendix A). Table 3.2.1 outlines the functions that will be carried out on the microcontroller to meet the specification requirements. Some functions have been added to add reliability within the system. For example, a Rolling Average to smooth the ADC input.

Table 3.2.1 - PSD Software Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| Set Default | Resets PSD and PSA settings to their default. |
| Rolling Average | A time-moving average to smoothen ADC input values, increasing the accuracy of the pressure sensor values by removing erratic spikes. |
| Peak Hold Reset **[REQ\_110]** | Resets a channel’s saved peak pressure value so the last recorded peak can be cleared, and new peak can be recorded. |
| Bottom Hold Reset **[REQ\_110]** | Resets a channel’s saved bottom pressure value so the last bottom peak can be cleared, and new bottom can be recorded. |
| Zero Clear **[REQ\_120]** | Sets live pressure values and analogue outputs to zero, to compensate for the zero drift that naturally occurs within the pressure sensors. This is done by exposing the sensors to the atmosphere and using that value for offset calibration. Alongside this, we used the offset value to determine sensor wear/faulty operation. |
| Hysteresis Check **[REQ\_180]** | Hysteresis is an Operation Mode that checks Channel Pressure against Hi and Hysteresis setpoints. |
| Comparator Check **[Objective\_02]** | Comparator is an Operation Mode that checks Channel Pressure against Hi, Lo and Hysteresis setpoints. |
| Bluetooth TX/RX **[REQ\_010]** | TX: Send live pressure readings, peak pressure and bottom pressure at regular intervals, alongside parameters.  RX: Receive updated settings sent from PSA, process and update parameters. |
| Digital Output **[REQ\_160, REQ\_170**] | Set Digital Output ‘High’ or ‘Low’, indicated by the pertinent LED ‘High’ dependent on the LED State Variable and Switch Mode |
| Analogue Output **[REQ\_150, REQ\_190]** | Set Analogue Output to a voltage value relative to pressure reading in range set by the User (1-5V, 0-10V) |

Alongside the functions being performed, the microcontroller will store all parameter data associated with pressure switch functionality. The PSD stores all the parameter variables shown within Table 3.2.2. The PSD stores these variables in the event a new device connects to the PSD. The PSA can then initialise with the parameters used by the previous user.

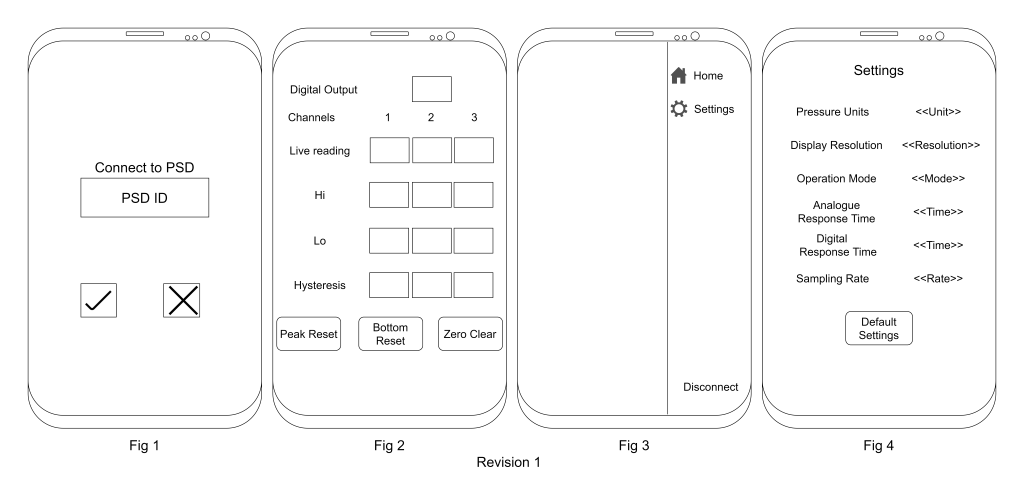
Table 3.2.2 - PSD Stored Parameter Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable/Setting** | **Description** | **Selection** | **Value** |
| CHX\_PRESSURE | Live pressure reading from channel X | - | 0.000 – 1.000MPa |
| CHX\_PEAK | Peak pressure recorded from channel X | - | 0.000 – 1.000MPa |
| CHX\_BOT | Bottom pressure recorded from channel X | - | 0.000 – 1.000MPa |
| CHX\_OUTY\_SM | Switch mode of digital output Y of channel X | 1. Normally Open  2. Normally Close | - |
| CHX\_OUTY\_OM | Operation mode of digital output Y of channel X | 1. Hysteresis  2. Window Comparator | - |
| CHX\_OUTY\_HI | High pressure setpoint value of digital output Y of channel X.  User can input the setpoint value | - | 0.000 – 1.000MPa |
| CHX\_OUTY\_HY | Hysteresis value of digital output Y of channel X.  User can input the Hysteresis value | - | 0.000 – 1.000MPa |
| CHX\_OUTY\_LO | Low pressure setpoint value of digital output Y of channel X.  User can input the setpoint value | - | 0.000 – 1.000MPa |
| CHX\_OUT1\_LED | LED setting of digital output of channel X | 1. ON: Green, OFF: Red  2. ON: Red, OFF: Green  3. Normally: Red  4. Normally: Green | - |
| RESP\_TIME | Response time of all digital/analogue outputs | 1. 2ms  2. 20ms  3. 50ms  4. 100ms  5. 200ms  6. 500ms | - |
| ANALOG\_VOLT | Analogue output voltage range | 1. 1-5V  2. 0-10V | - |
| USER\_PRESSURE  \_UNIT | Pressure unit used in App that can be set by the App user for the live pressure reading (CHX\_PRESSURE), peak recorded pressure (CHX\_PEAK), bottom recorded pressure (CHX\_BOT), also the pressure that is input by the user: Hi-setpoint (CHX\_OUTY\_HI), Hysteresis (CHX\_OUTY\_HY) and Lo-setpoint (CHX\_OUTY\_LO). | 1. MPa  2. kPa  3. kgf/cm2  4. Bar  5. PSI  6. mm Hg  7. cm Hg  8. inch Hg | - |
| USER\_DISP\_RESO | The decimal places shown by the live pressure reading, peak recorded pressure and bottom recorded pressure, can be set by the user. | 1. 0.001 (3 decimal places)  2. 0.01 (2 decimal places) | - |
| USER\_REFRESH  \_RATE | The time taken for the display of the live pressure reading to update, this can be set by the user. | 1. 200ms  2. 500ms  3. 1000ms | - |

# Pressure Switch App Design

### Android App Design

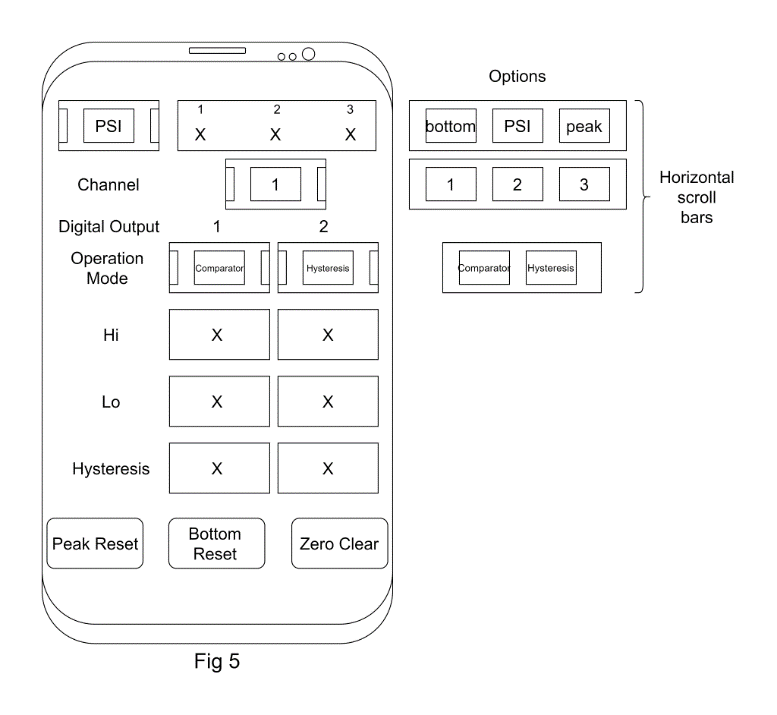
The customer suggested a design for the application containing all the desired elements, however it was not very mobile friendly. In order to make the app more suitable for mobile phones, I went through an iterative design process by communicating directly with the customer.



Above are 4 figures displaying the design of the 4 sections of the application.

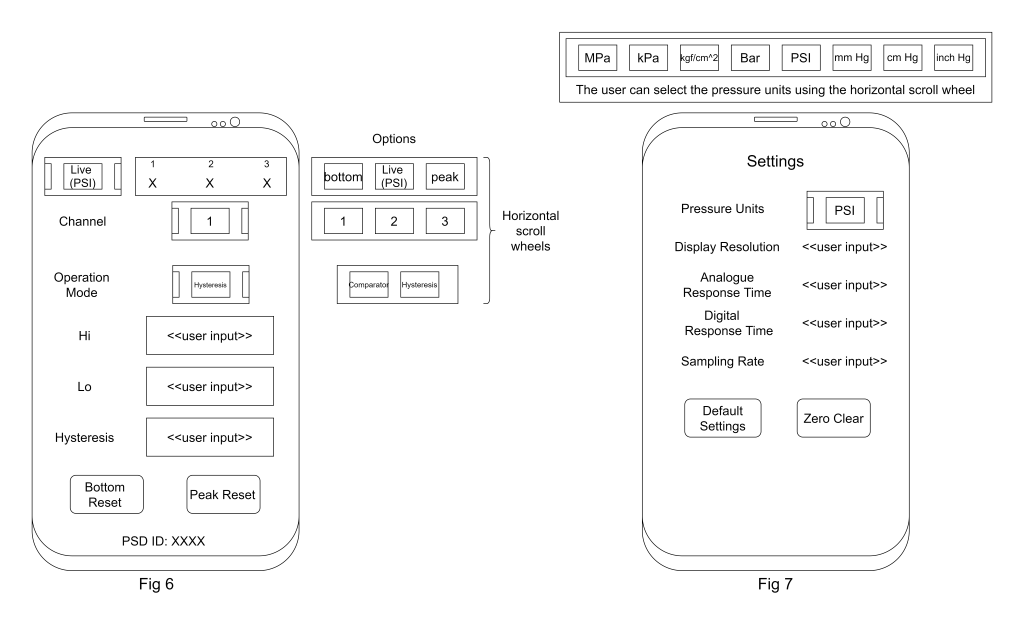
1. Fig 1: This is the initial design of the connection screen. This is the first screen the user is presented with when the app is loaded. A connection to a switch can be initiated by tapping a corresponding NFC tag.
2. Fig 2: This is the design of the home page the app takes the user to once a connection to the switch has been established. Here the user can access the basic settings of the switch. In order to make the application less cluttered, I decided to present one channel settings at a time, with the ability to select the channel at the top of the page.
3. Fig 3: This is the design of the navigation drawer, which is accessible by swiping from the edge of the screen.
4. Fig 4: This is the initial design of the settings page. Here the user can access additional settings that are not as frequently needed. The rationale behind splitting the app into two main screens (Fig 2 and Fig 4), was the limited screen space on phones.

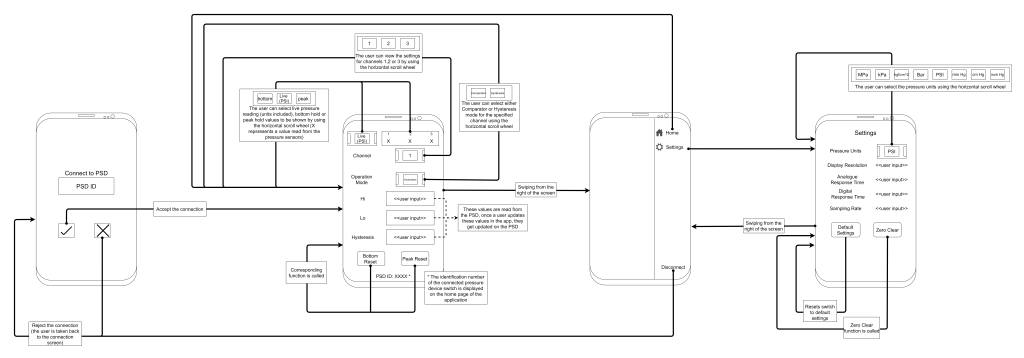
The customer provided feedback on the first design revision. Fig 1, 3 and 4 were found to be satisfactory, but Fig 2 (the Home screen) required some changes. The customer suggested incorporating a way to view bottom and peak hold values, and live readings, as well as including a way to change the operation mode of each one of the two outputs per channel. These changes are shown in Fig 5.



Further feedback was provided by the customer. Fig 6 shows the updated design of the Home screen. Due to time constraints of this project, incorporating functionality for changing the settings of both channel outputs became a stretch objective.

Fig 7 shows the updated design of the settings page.

  
Having completed the third iteration, we reached a design the customer was happy with. Below is an image showing the app flow of the final iteration of the design. The arrows shown are associated with a possible action, have a short description of what the action should accomplish, and points to the screen the user is taken to once the action is accomplished.



#### App Architecture

The app architecture used in this project is MVVM - Model, View, ViewModel.

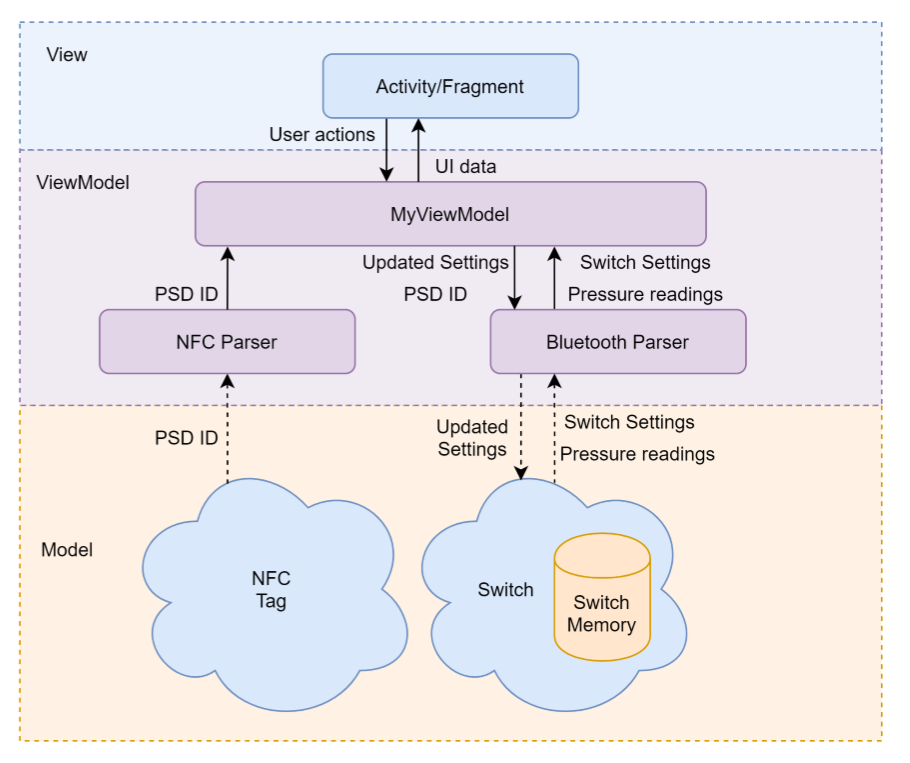
* Model: this is the data the application must work with.
* View: Like the MVC architecture, View refers to a visual aspect of the application, something the user can see on their screen.
* ViewModel: this is the part that manages and prepares the raw data to be presentable by the View.

An apparent benefit of this design architecture is the fact that the View is quite separate from the model, which means that changes in the Model will have no effect on the View (granted that the ViewModel is updated accordingly).

Android applications are built using the following building blocks:

* Activity: entry point for the user which contains the user interface.
* View: represents a visual component in the application. All visual components extend this class.
* Fragment: represents a collection of related Views. Combinations of Fragments can be used to construct a dynamic UI inside an Activity.

Below is a diagram showing the MVVM architecture of the application and its different components. One feature that the client wanted in the application was initiating Bluetooth communication between the app and the switch using NFC to simplify the pairing process.



For this to work, the app must know the Bluetooth name of the switch. We decided that the simplest way to approach this would be to have an NFC tag containing the name of the switch. Each switch device is given a unique name to make them uniquely identifiable. Once the application has read the name of the switch from the NFC tag, it can proceed to establish a Bluetooth connection with that switch.

The diagram shows an overview of the function of the application. Since the switch will be storing all the settings, and transmitting the pressure readings, there is no need for the application to store anything locally, hence why the Model is separate from the application.

The ViewModel has some classes that are used to parse data read from the NFC tag, shown as NFC Parser, and some other classes that are used for the Bluetooth connection, shown as Bluetooth parser. The ViewModel acts as a mediator between the switch and the application, by receiving the data from the switch and rendering it to a format used by the View, as well as processing user input from the View.

Finally, the View contains Activity and Fragment classes, which represent the visual components of the application and handle any user input and events.

# Pressure Switch Device Build

## Hardware Build

### Pressure Switch Circuit Overview

The circuit components can be divided into several sub-components, each performed different function as shown in Block Diagram in Figure 4.1.1.

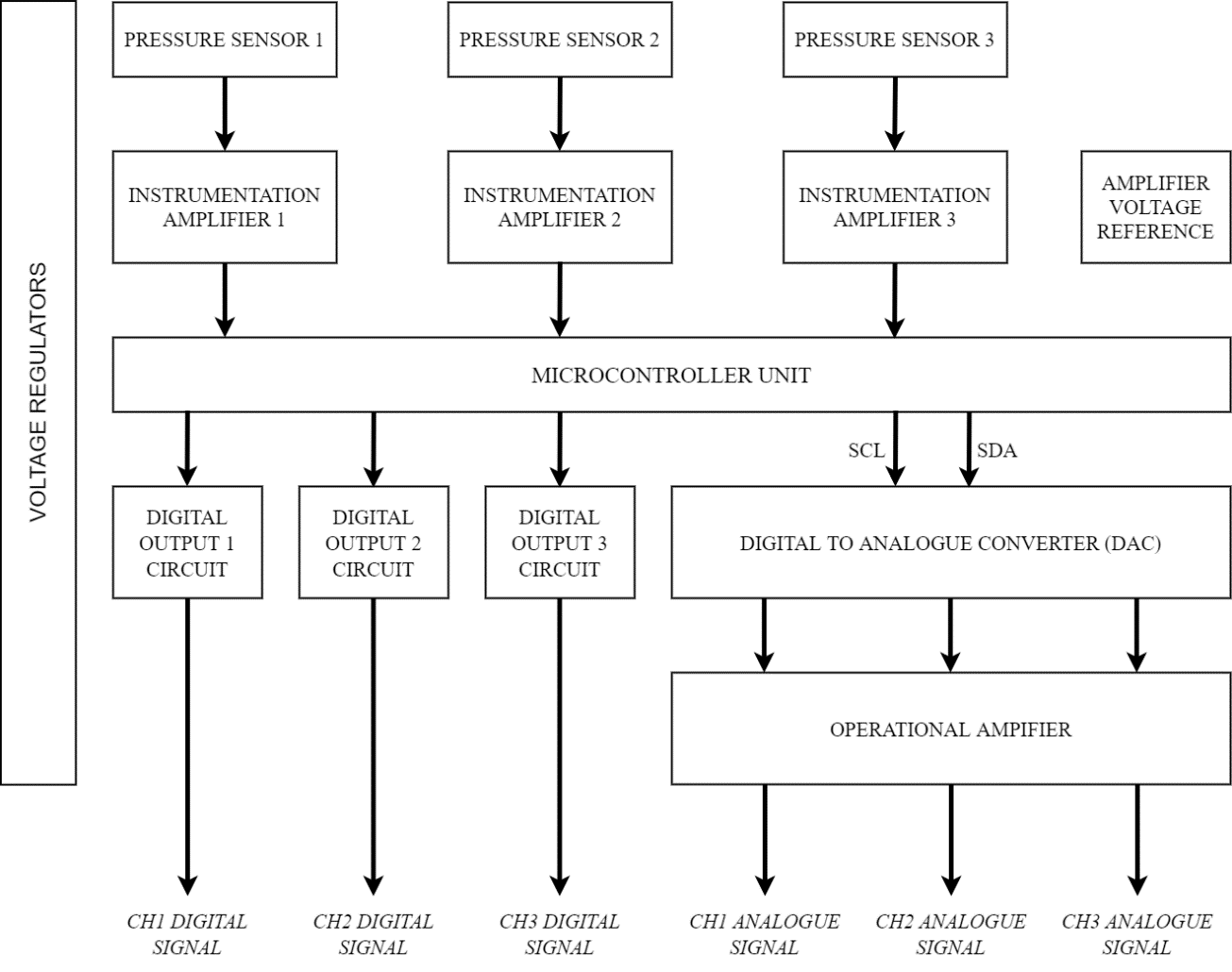


Figure 4.1.1 - Pressure Switch Circuit Block Diagram

The pressure switch circuit use two voltage regulators to step down the 12 to 24V supply voltage to a stable system voltage of 5V and 10V. These 5V and 10V regulators provide power to different components of the circuit. The 5V voltage regulator provide a stable voltage for the microcontroller board, digital to analogue converter (DAC) and voltage reference for the instrumentation amplifiers. The 10V voltage regulator provide voltage for the three sensors, instrumentation amplifiers and amplifier of the analogue output circuit.

There are two type of pressure sensors used in the pressure switch; gauge and absolute type pressure sensor. For this version of the pressure switch, channel 1 and channel 2 pressure sensor are configured with gauge type pressure sensor and channel 3 is configured with absolute type pressure sensor. The pressure sensor input circuit which includes the instrumentation amplifier and voltage reference are designed to interface with Honeywell XPC150AFSH absolute pressure sensor and Honeywell XPC150GFSH gauge pressure sensor. However, different type of pressure sensor that are based on bridge circuit and able to output two differential signals should work with the designed circuit with some modifications. An instrumentation amplifier is used for each pressure channel to amplify the differential signal of the corresponding pressure sensor before the signal is send to microcontroller unit for further processing and logic controls. To provide a stable and precise voltage with low temperature drift, a dedicated voltage reference IC is also used to provide a voltage reference for the three instrumentation amplifiers.

The pressure switch used a small Arduino Nano 33 BLE that act as microcontroller unit of the pressure switch and as Bluetooth Low Energy transceiver which are built into the microcontroller board. The board is based on nRF52840 microcontroller which used ARM Cortex M4 CPU with clock speed of 64MHz, as this is complete microcontroller board, there a built-in 3.3V switching regulator to supply power to all the components on the microcontroller board. The power to the microcontroller board is supply by the switch 5V LDO voltage regulator. The small size board and the built in BLE transceiver allowed the overall pressure switch circuit to be simplified by smaller microcontroller unit footprint and no additional BLE module is required. The board used stand-alone uBlox Nina-B306 BLE module.

The circuit for digital output or the switch output is designed as high side switching circuit which provide power to grounded load. The logic high voltage for the digital output is equal to the voltage supply to the pressure switch. This digital output circuit consist of several discreet components that includes NPN transistor, PNP transistor, Zener diode, current limiting resistor and decoupling capacitor. These components work together to switch the supply voltage of the pressure switch to the load based on the 3.3V signal it receive from the microcontroller.

The switch analogue output used digital to analogue converter (DAC) IC to convert the digital signals from the microcontroller unit to analogue signals to produce the required analogue signals. The communication between the microcontroller unit and DAC used I2C serial communication protocol hence the used of the two wire SDA and SCL signal link between the two. The DAC has maximum output signal voltage of 5V. The analogue circuit are designed to output two type of analogue voltage signal; 1-5V and 0-10V, this is achieve using the operational amplifier that act as 2x amplifier with 10V maximum output voltage and also the switch programming of the DAC to varies the analogue output signal between the two type analogue signals.

### Microcontroller

The selections of the Microcontroller were made based on factors of development time, required hardware interfaces and the Bluetooth communication.

Based on Figure 4.1.1 and Figure 3.1.1 , the circuit required microcontroller that has at least 3 ports that support ADC to receive pressure input from the sensors. The circuit also required 9 General Purpose Input and Output (GPIO) ports that can output digital signal for the 3 digital outputs circuit and 6 LEDs. The used of external DAC required I2C serial communication interface to communicate with the microcontroller, therefore two port are required for the I2C which are SDA and SCL. The port requirement for the circuit functionality are shown in Figure 4.1.1Table 4.1.1.

Table 4.1.1 - Microcontroller input and output ports requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Main Components** | **Hardware Interface** | **No of I/O Port(s)** | **Sub-Components** |
| Pressure Sensor | ADC | 3 | * Pressure Sensor CH1 * Pressure Sensor CH2 * Pressure Sensor CH3 |
| Digital Output | GPIO | 3 | * Digital Output CH1 * Digital Output CH2 * Digital Output CH3 |
| Digital to Analogue Converter (DAC) | I2C | 2 | * SCL * SDA |
| LED Indicators | GPIO | 6 | * CH1 Green LED * CH1 Red LED * CH2 Green LED * CH2 Red LED * CH3 Green LED * CH3 Red LED |

Considering the relatively short development time, the microcontroller that can support Arduino Integrated Development Environment (IDE) are chosen for the hardware development. The program written in the IDE can easily be change for another microcontroller that also support Arduino IDE. At the start of the project, Arduino Nano was chosen based on port requirements in Table 4.1.1 which later change to Arduino Nano 33 BLE when BLE communication is needed for iOS app. The same pins arrangement of the two boards facilitate this transition. Arduino Nano 33 BLE also has higher 12-bit ADC resolution which is needed to maintain the effective resolution of the reading when the pressure reading from the instrumentation amplifier is offset by the voltage reference. The high clock rate of 64MHz also enable the 2ms response time to be achieve which is difficult in standard Arduino Nano.

### Voltage Regulator

Texas Instruments LM2937 low dropout (LDO) regulator series is used for both 5V and 10V voltage regulators. This regulator has 500mA maximum load current, 26V maximum input voltage and 0.5V dropout voltage. The main reason for the use of voltage regulator instead of switching regulator is to avoid the switching noise which could affect performance of the mixed signals circuit. The whole switch circuits were designed to use the same voltage level of 10V and 5V to reduce the need for many voltage regulators hence reducing the overall circuit footprint and cost.

Compared to normal linear regulator, LDO regulator can regulate the output voltage even when supply voltage close to it. As with other voltage regulator, bypass capacitors are needed to reduce the voltage ripple output from the regulator. However, for LDO regulator, the parameters of the bypass capacitors specifically the equivalent series resistance (ESR) of the capacitors is critical to ensure the voltage stability of the regulators. The capacitances of the bypass capacitors are shown in Figure 4.1.2.

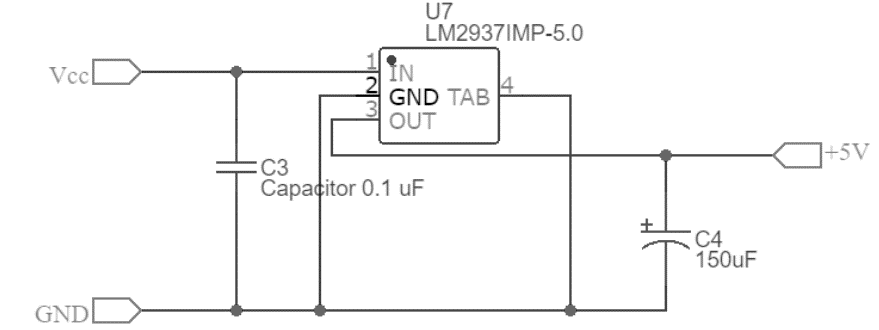


Figure 4.1.2 - Low dropout voltage regulator circuit diagram

As recommended in the datasheets of LM2937 [1], the output bypass capacitor must have capacitance value of at least 10uF with ESR of between 10mΩ to 3Ω. Electrolytic capacitor is chosen for both input and output bypass capacitor because of large case that can dissipate more heat and cheaper compare to Tantalum capacitor. The datasheet also recommended 0.1uF for input bypass capacitor for regulator that more than 3 inches from the supply capacitor which is applied in this circuit. The output of the bypass capacitor characteristics are shown in Table 4.1.2.

Table 4.1.2 - Voltage regulator bypass capacitors parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Capacitance | Manufacturer and part number | Max Voltage | ESR |
| 150uF | Panasonic - EEEFTV151XAP | 35V | 160mΩ |

### Pressure Sensor Input Circuit

#### Pressure Sensor

The two Honeywell Pressure Sensor XPC150AFSH and XPC150GFSH used in the pressure switch are bridge-type sensor which can be described as two parallel voltage dividers connected together as shown in the Figure 4.1.3 a), the measured pressure is equivalent to the voltage difference between the two outputs of the dividers which are -Vout and +Vout. Mechanically, when the pressure applied to this sensor, the diaphragm in the sensor will deflect and change the resistance of the circuit hence change the measured pressure. The pressure sensor diagram and connection are shown in Figure 4.1.3 b), the +Vs is connected to the supply voltage and -Vs connected to the ground.

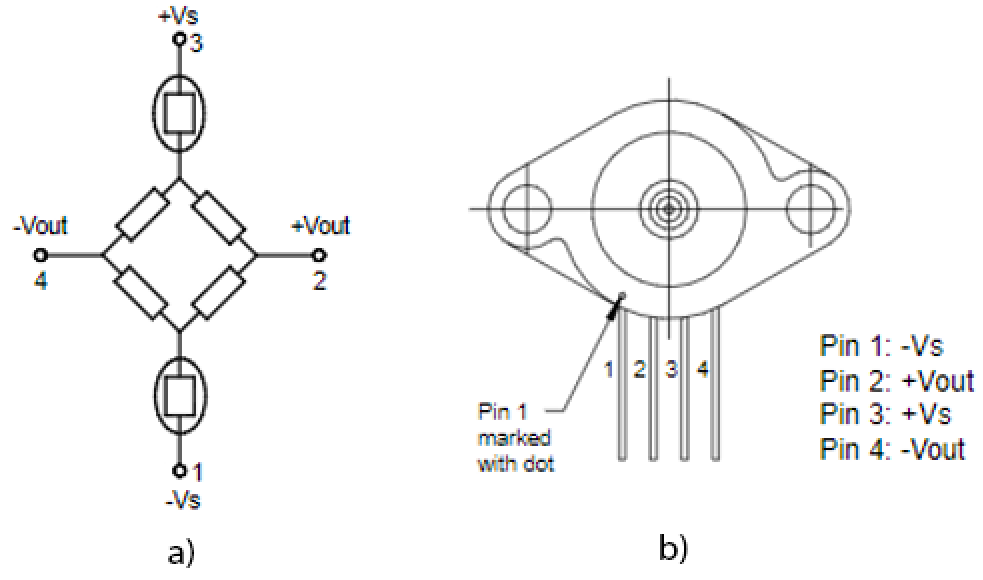


Figure 4.1.3 - a) Equivalent Pressure Sensor Circuit, b) Pressure sensor pins connection

The output of these sensors is in mV range and are ratio metric to the supply voltage. Hence, the sensor is supplied with 10V voltage to have a large full-scale voltage span. The full-scale span at supply voltage of 10V are shown in Table 4.1.3.

Table 4.1.3 - Full scale voltage span of the pressure sensor at supply voltage of 10V

|  |  |
| --- | --- |
| Characteristics | Value |
| Full scale span (Min.) | 70 mV |
| Full scale span (Typ.) | 75 mV |
| Full scale span (Max.) | 79 mV |

The pressure sensor parameters shown in Table 4.1.4 below was used for the instrumentation amplifier and voltage reference circuit.

Table 4.1.4 - Pressure sensor parameters

|  |  |
| --- | --- |
| Zero Pressure Offset | 1mV |
| Response Time | 100us |
| Common Mode Voltage | 5V |

#### Instrumentation Amplifier

Based on Table 4.1.3, the maximum different between the two signals produced from the pressure sensor is about 79mV, this signal will need to be amplify to a voltage level that can be read by the Analogue to Digital Converter (ADC) of the microcontroller to make the most of the ADC resolution for good precision reading. The ADC as the name suggest will convert the analogue signal into digital signal that can be process by the microcontroller. As with most of the bridge type sensor, the signal produced will has high DC offset (refer to Table 4.1.4) and large common-mode voltage [2] this result in produced signal that has high distortion. The large common mode voltage in the bridge circuit arise from the fact that the two output signals from the pressure sensor have almost the same voltage, in this case 5V which is the common mode voltage of the pressure sensor. Therefore, a special type of differential amplifier is required to amplify the two signals and reject the common mode voltage, this amplifier is called an instrumentation amplifier or in-amp.

An instrumentation amplifier is basically a type of differential amplifier that made up of three op-amp as illustrate in Figure 4.1.4. The two left op-amp act as a buffer for each input which provide matching impedance (refer impedance matching [3]) and the left op-amp act as differential amplifier that reject common mode component. The gain of the amplifier is set by resistor Rgain. A dedicated in-amp IC that has all this component and is designed for the bridge sensor reduced the circuit complexity. As there are 3 pressure sensors used in the switch, therefore 3 in-amp are required.



Figure 4.1.4 - Building block of basic instrumentation amplifier (Source [4])

There are few amplifier characteristics that must be considered when choosing a suitable instrumentation amplifier, first is the common mode rejection ratio (CMMR). High CMMR amplifier can reject most of the signal noise as result of common mode voltage. Second is the amplifier input impedance, the input impedance must be large “to avoid loading the circuit” [5]. The third characteristics is input offset which ideally need to be low.

Analog Devices AD623 in-amp was chosen because it is available in PDIP package which make it easier for breadboard testing and because of the stock availability at the time of testing. The parameters for AD623 and other alternative instrumentation amplifier that have an identical pins arrangement with AD623 are shown in Table 4.1.5

Table 4.1.5 - Comparison of different instrumentation amplifier parameters

|  |  |  |
| --- | --- | --- |
| Instrumentation Amplifier | Input Offset Voltage | Quiescent current |
| AD623ARM | 200uV | 550uA |
| AD8223 | 250uV | 500uA |
| INA128 | 50uV | 700uA |

The pins arrangements of AD623 are shown in Figure 4.1.5. As the three in-amp listed in Table 4.1.5 have identical pins arrangement, the same PCB layout can be used with all the three in-amp if there a need to change to other in-amp due to stock availability. Each of listed in-amp are available in SOIC surface mount package. However, each in-amp has different gain equation therefore may need a difference gain resistor for the same gain value, but the gain can easily set by one resistor. The gain resistor placed between pin 1 and pin 8 of the in-amp shown in Table 4.1.5.

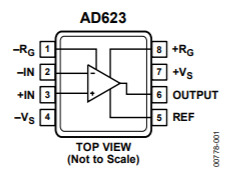


Figure 4.1.5 - AD623 In-Amp Pins Arrangements

From the pressure sensor voltage span data in **Error! Reference source not found.**, the suitable gain to set for the in-amp is around 10x this translate to maximum voltage span of 0.79V after amplifications which is lower than maximum input 3.3V of the microcontroller. The gain equation and the output voltage of the AD623 is shown in equation 1 and 2.

From the gain equation 1, a gain of 10 result in the calculated gain resistor value of 11kΩ but a more widely available 10kΩ resistor was chosen, which result in a gain of 11 and give a maximum voltage span of 0.869V. This is the maximum voltage range that the ADC of the microcontroller will receive.

The connections of the instrumentation amplifier pins with the pressure sensors pins are shown in Figure 4.1.6 - Connections between pressure sensor and instrumentation amplifier. The in-amp is supply with 10 V voltage at +Vs pin and -Vs pin connected to the ground.

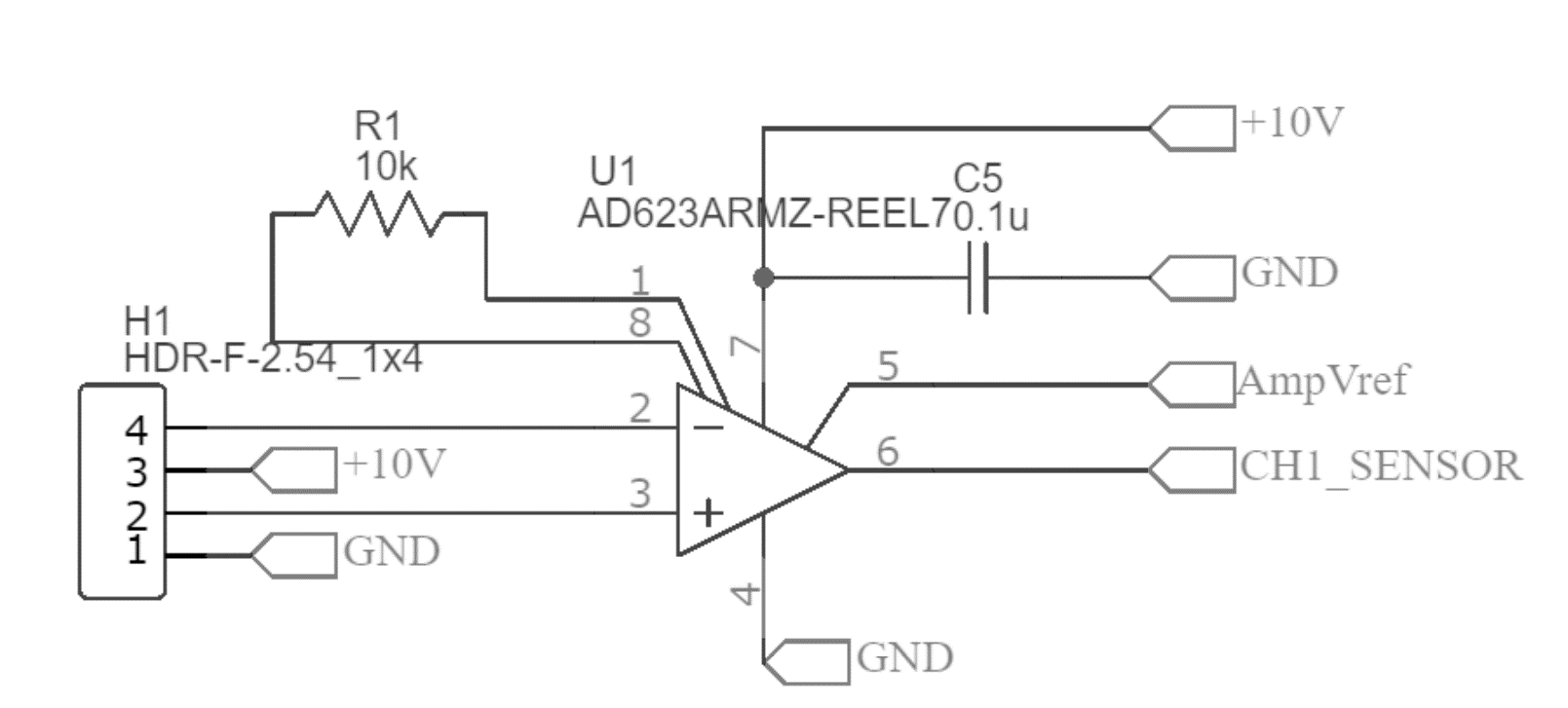


Figure 4.1.6 - Connections between pressure sensor and instrumentation amplifier

#### Amplifier Voltage Reference

With supply of +Vs at 10V and -Vs at ground, the testing of the instrumentation amplifier has found that the in-amp has offset error of 60mV which mean the amplifier cannot detect voltage below this voltage. Therefore, given that maximum voltage span is only 0.869V, this voltage can be bias with voltage above 1V at the reference pin of the in-amp so that the output voltage does not clip to 60mV. Instead of using voltage divider to generate this voltage at reference pin of the in-amp, voltage reference IC are used to give a precise voltage with low temperature drift and low impedance as recommended by the in-amp datasheets.

Series voltage reference like TI REF3012 has simpler implementation than shunt voltage reference without the need for the shunt resistor. However, this type of voltage reference IC required the used of low equivalent series resistance (ESR) capacitor at the input supply such as Tantalum capacitor. The output of the voltage reference is measured at 1.35V which become the minimum voltage receive by the ADC. Hence, assuming the typical voltage range of the pressure sensor after amplified is 0.8V the ADC range will be from 1.35V to 2.15V which translate to 1675 – 2668 ADC values for 0.0 - 1.0 MPa pressure range.

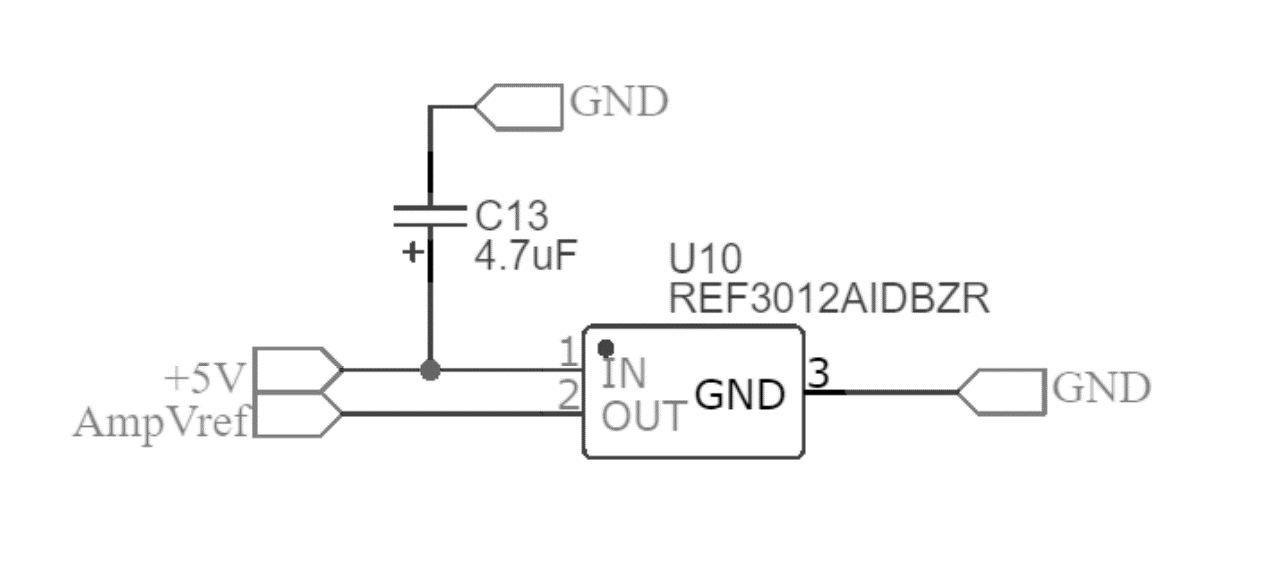


Figure 4.1.7 - REF3012 Voltage Reference Circuit Setup

### Digital Output Circuit

The PNP switch or high side digital output switching required the microcontroller to switch load that has higher voltage than its signal voltage. Therefore, the PNP transistor that switch the load need an NPN transistor to act as its driver to unbiased the PNP Vbe junction. Figure 4.1.9 shown the high side switching circuit diagram.

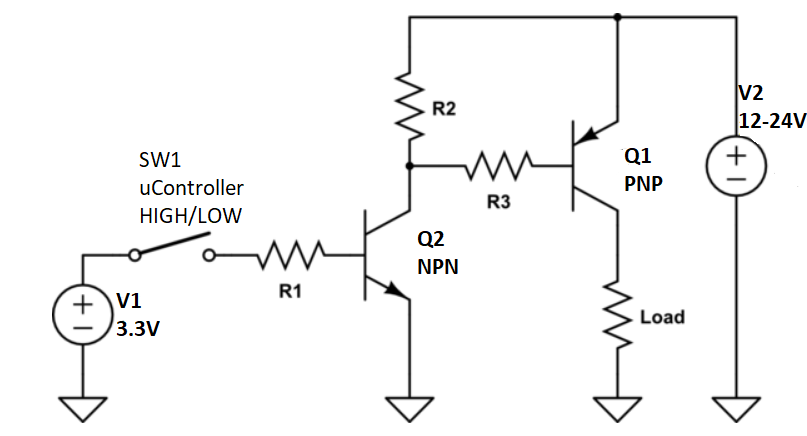


Figure 4.1.8 - PNP high side switching with NPN transistor driver

Zener diode with Zener breakdown voltage of 24V is added to the circuit to provide overvoltage protection by clipping the maximum voltage that can receive by the load. Current limiting resistor of 200Ω is added in series with the circuit to limit the current flow to around 100mA. The PCB circuit implementation is shown in Figure 4.1.9. Pre-biased NPN and PNP transistor IC reduced the overall circuit footprint.

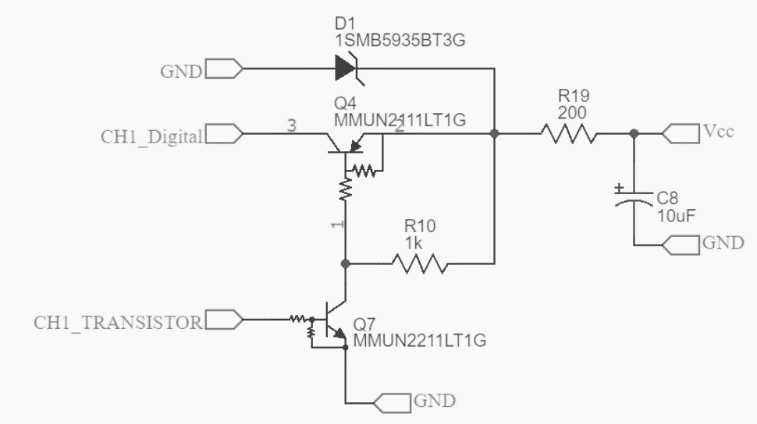


Figure 4.1.9 - Digital output circuit implementation

The characteristics of the digital output components are shown in Table 4.1.6.

Table 4.1.6 - Digital output circuit components part and characteristics

|  |  |  |  |
| --- | --- | --- | --- |
| **Components** | **Part** | **Characteristics** | **Value** |
| PNP Transistor | MMUN2111LT1G | Collector Current | 100mA |
| Pre-biased Resistance, R1 & R2 | 10kΩ |
| NPN Transistor | MMUN2211LT1G | Collector Current | 100mA |
| Pre-biased Resistance, R1 & R2 | 10kΩ |
| Zener Diode | 1SMB5935BT3G | Zener Voltage | 27V |
| Max Power Dissipation | 3W |

### Analogue Output Circuit

The MCP4728 Digital to Analogue Converter IC provide a fast settling time which fulfil the 2ms pressure reading response time. The DAC has 4 analogue output channels which reduced the need of daisy chaining multiple DAC IC. It also output 12-bit resolutions which equivalent to 4096-voltage step and use I2C communication protocol to communicate with the microcontroller. The use of external DAC reduced the microcontroller pins requirements and allow the switch to output a true analogue signal. The DAC circuit connection are shown in Figure 4.1.10. The chip is supply with 5V voltage and two 10kΩ pullup resistor that connected to 5V supply voltage for SCL and SDA pins, this to pull the two signals high as the bus driver of the DAC is open drain hence provide a known voltage level when the signal is high.

A picture containing diagram

Description automatically generated

Figure 4.1.10 - MCP4728 Digital to Analogue Converter circuit connections

The maximum voltage signal of the DAC is 5V as this is the voltage set as reference voltage for the DAC. Therefore, for the switch to output a 0-10V analogue signal, an operational amplifier with gain of 2x is added at the output of the DAC. This amplifier is setup as non-inverter amplifier, with gain equation 3.

Hence, the value of R1 and R2 are set to the same value as shown in Figure 4.1.11 - Non-inverting amplifier circuit connections.

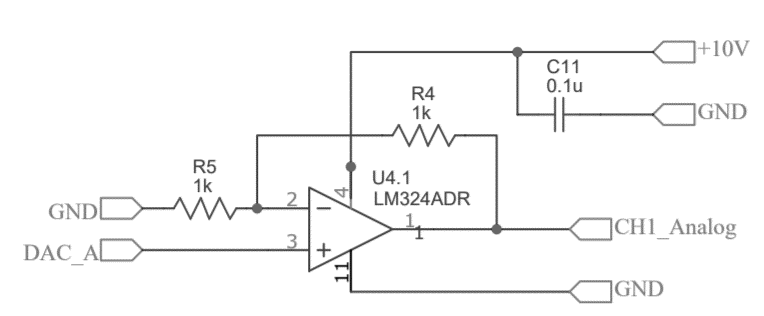
****

Figure 4.1.11 - Non-inverting amplifier circuit connections

### Integrated Pressure Switch Circuit

The schematics for the PCB board is shown Appendix C. Each integrated circuit chip is equipped with decoupling capacitor of 0.1uF to provide a stable voltage to the IC components which will reduce the effect of voltage drop and voltage spike as a result of noise that could come from other components of the board.

JST-XH male headers are chosen for the input and output ports for the board, these headers are used for the power and signal cables for the circuit. There are two types of headers pins use, the 3 pins type header and 7 pins type header. The Vcc and ground input cable to the board use 3 pins headers, with a Vcc supply cable and two ground cables. The digital and analogue output signals also used separate 3 pins type headers. The 7 pins header are used for the 6 LED cables and a ground cable. This LED cable is used for the 6 LED indicators that mounted on top of the pressure switch case.

Other type of header used on the board is a standard 2.54 female header. This header is used to mount the HC-05 Bluetooth Classics module to add Bluetooth classic functionality to the pressure switch. The 3 sensors also used the same type of headers.

The microcontroller has 3 extra pins which are pins A3, A6 and A7 that can be used in the future to add functionality to the circuit. These pins may be used for 3 additional digital output signals, to add another digital output for each channel.

### Printed Circuit Board (PCB) Layout

The PCB layout are shown in Figure 4.1.13. This is a two-layer PCB board. The design of the layout has taken into consideration the overall heat dissipation of the circuit and ease of assembly of the Surface Mount Device (SMD) components.

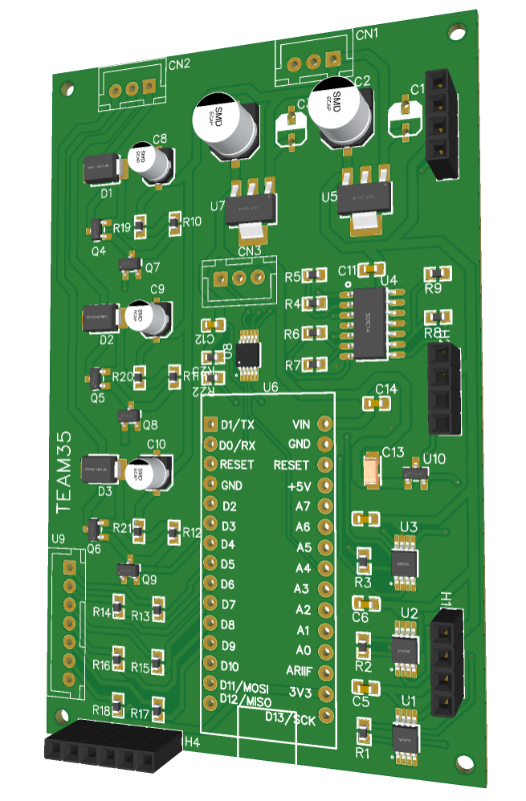
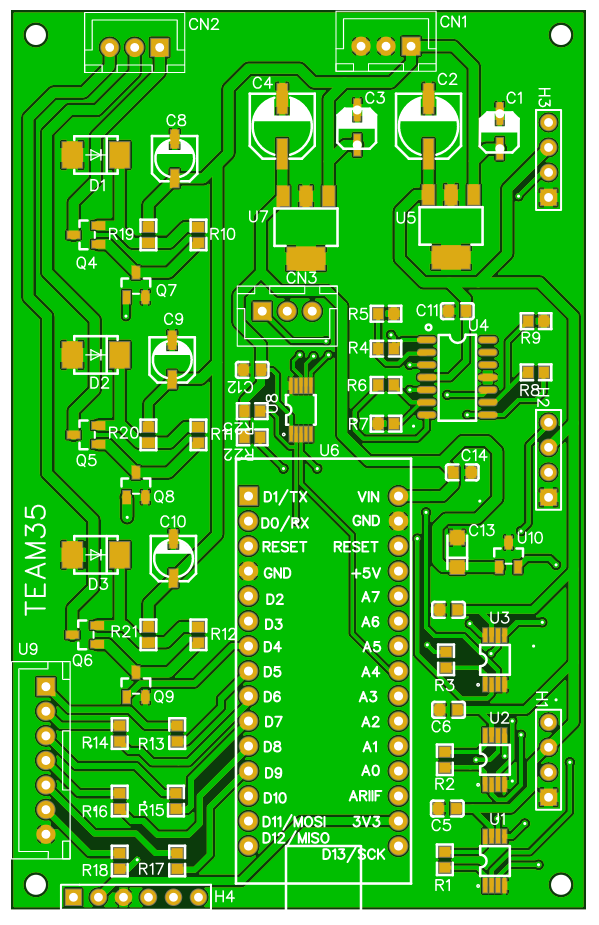


Figure 4.1.13 - Pressure Switch Circuit PCB Layout

The SMD components were placed in such way to make it easy for the assembly by allowing some space between each component. Each decoupling capacitors positioned close to the voltage supply of the corresponding IC to ensure its effective functionality in filtering the supply voltage. Large components such as the electrolytic capacitor for the voltage regulator and digital output were chosen to ensure the components dissipate less heat to overall circuit. Other components that are relatively large are the Zener diodes, npn and pnp transistor and the opamp.

The manually routing of all the traces allowed all the traces to bend 45º to ensure constant width of the traces. Large traces of 1mm are used where possible for the supply traces which includes Vcc, 10V and 5V across the board. The signal traces are varying in size dependent on the pad size of the components, the sizes of the signal traces are 0.30mm, 0.35mm, 0.4mm and 0.6mm. The final board dimension are 91 mm by 58 mm.

Four holes with diameter of 2mm at the edge were made which can be used to mount the board securely on non-conducting surface or inside a case. The board has ground plane for both layers to provide common ground to all the components on the board and to distribute the heat of the components evenly.

The designing of the PCB schematics and layout was done in EasyEDA which has large library of SMD components footprint and can be submit easily to be manufactured by JLPCB at very low cost.

### PCB Bill of Materials

The part number of the components and the price for each component unit are in Appendix D. The prices are based on Farnell and RS Components and not included the shipping cost. The shown prices are also based on the smallest quantity the components that can be purchase, the price of each unit will decrease when the quantity ordered is large which usually the case in large scale production. The Arduino Nano 33 BLE made up the large chunks of PCB cost, this is because the PCB board was designed to work with the whole Arduino board to accelerate the development duration, a more cost-effective solution is to use the uBlox Nina B306 in the Arduino which cost around £10 or an alternative Bluetooth equipped microcontroller.

## Software Build

Within this section, the PSD functions and operation will be outlined and explained. Alongside, any design decisions made in the process of building the PSD embedded software.

### System Overview

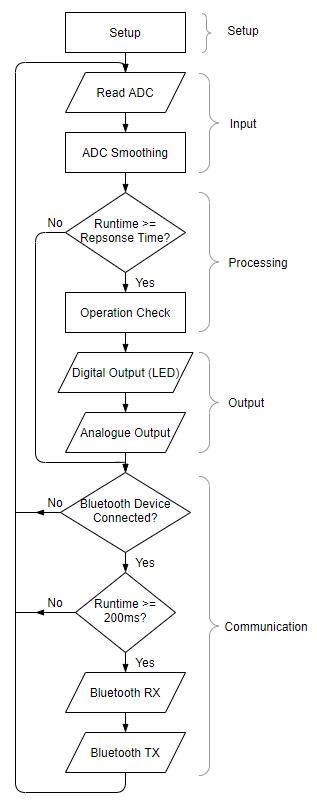


Figure 4.2.1 - PSD System Flowchart

The PSD is programmed to replicate the functionality of a digital Pressure Switch. The PSD has a sequential system flow, extracting, processing and outputting the pressure sensor values in the form of live pressure readings and digital/analogue signals. The PSD uses a built-in runtime function to perform CPU time-slicing. The timing aspect is used only for the response time of the Pressure Switch, as well as, executing periodic Bluetooth transmit/receive requests. This is done by using the Arduino *millis* function.

**Error! Reference source not found.** shows the sequential flow of the operation of the PSD system. The implementation of the PSD is using the C++ programming language, due to the use of the Arduino Nano 33 BLE microcontroller. A library has been developed to perform the functionalities outlined within the specification document, alongside additional requested functionality from the customer.

The library is encapsulated in a Switch class, containing all the PSD methods, attributes and objects. The choice of using a class was predominantly due to being able to have an organised code base, allowing it to be easily maintained which is crucial when passing it on to the customer for further development. Furthermore, the data structures lay the foundations of the Channel Objects, having their own attributes such as Pressure, Hi Setpoint, Lo Setpoint etc (see Table 3.2.2). Allowing for code modularity and reusability.

The following sections will discuss in detail, the operation of the functions implemented within the PSD library, to implement a successful, reliable Pressure Switch.

### Setup

Upon start-up of the PSD, it undergoes a setup phase, allowing it to be in a state ready for operation. This phase must be carried out under zero pressure, to allow for sensor calibration using the Zero Clear function.

Firstly, the pin configuration of the Arduino is set. Table 4.2.1 shows the configuration of each pin used by the PSD, along with its input/output configuration.

Table 4.2.1 - PSD Pin Configuration

|  |  |  |
| --- | --- | --- |
| **Pin No.** | **Input / Output** | **Pin Name:** |
| A0 | Input | CH1 Input |
| A1 | Input | CH2 Input |
| A2 | Input | CH3 Input |
| A4 | Output | DAC Data Line (SDA) |
| A5 | Output | DAC Clock Line (SCL) |
| D2 | Output | CH1 Digital Output |
| D3 | Output | CH2 Digital Output |
| D4 | Output | CH3 Digital Output |
| D5 | Output | CH1 Red LED |
| D6 | Output | CH1 Green LED |
| D7 | Output | CH2 Red LED |
| D8 | Output | CH2 Green LED |
| D9 | Output | CH2 Red LED |
| D10 | Output | CH2 Green LED |
| D11 | Output | Bluetooth TX |
| D12 | Input | Bluetooth RX |
| D13 | Input | Bluetooth State |

The next stage of Setup is to initialise the PSD with its default parameter values. The default parameter values are outlined in Table 4.2.2.Alongside being the values that are used to initialise the PSD, these are the same default values that will be applied when the Set Default function is called by the PSA. Lastly, the DAC is configured using the ‘mcp4728’ Arduino library [1].

Table 4.2.2 - PSD Default Parameter Values

|  |  |  |
| --- | --- | --- |
| **Variable Type** | **Variable** | **Default setting** |
| User preferences | USER\_PRESSURE\_UNIT | MPa |
| USER\_DISP\_RESO | 0.001 (3 decimal places) |
| USER\_SAMP\_RATE | 500ms |
| Digital Output | CHX\_OUTY\_SM | Normally Open |
| CHX\_OUTY\_OM | Hysteresis |
| CHX\_OUTY\_HI | 0.500 MPa |
| CHX\_OUTY\_HY | 0.050 MPa |
| CHX\_OUTY\_LO | - |
| CHX\_OUT1\_LED | ON: Green OFF: Red |
| DIGITAL\_RESP\_TIME | 2.0ms |
| Analog Output | ANALOG\_VOLT | 1-5V |
| ANALOG\_RESP\_TIME | 2.0ms |

### Data Acquisition

Within the input section of the PSD, the ADC readings are smoothened by a Rolling Average, the raw ADC readings are converted into the associated PSD working pressure unit – MPa, and the peak/bottom values are stored as shown in Figure 4.2.2.

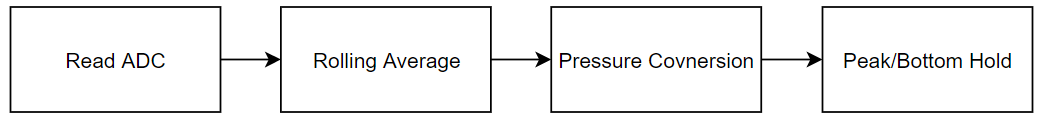


Figure 4.2.2 - PSD Input Flowchart

#### Read ADC

The Arduino reads a voltage produced by the pressure sensor input circuit via an on-board 12-bit ADC. The ADC is configured to 12-bits to allow for the maximum resolution of 4,096 (212) to be realised, maximising system accuracy.

The equation for extracting the ADC value can be seen in Equation 4:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

Equation 4 outlines how an accurate ADC value can be calculated. The pressure sensors are prone to zero drift, due to the mechanical properties of the sensors themselves. This has the undesired effect of the zero pressure ADC value to be offset. Within the Zero Clear function, a zero offset value is obtained (initially = 0) and subtracted from the raw ADC value. This is performed for each channel. The ADC measures samples at 200kHz allowing for a large dataset to be used for increased precision.

#### Rolling Average

When first testing the Pressure Sensors with the Arduino, the input readings were noisy and erratic. This was an issue as it didn’t produce a stable input stream and would cause false outputs to occur.

To solve this a Rolling Average has been implemented to smoothen the ADC readings. The Rolling Average uses a sliding window algorithm, storing previous readings within a fixed size time-moving window. The sensitivity and speed of the smoothing can be altered by altering the window size parameter.

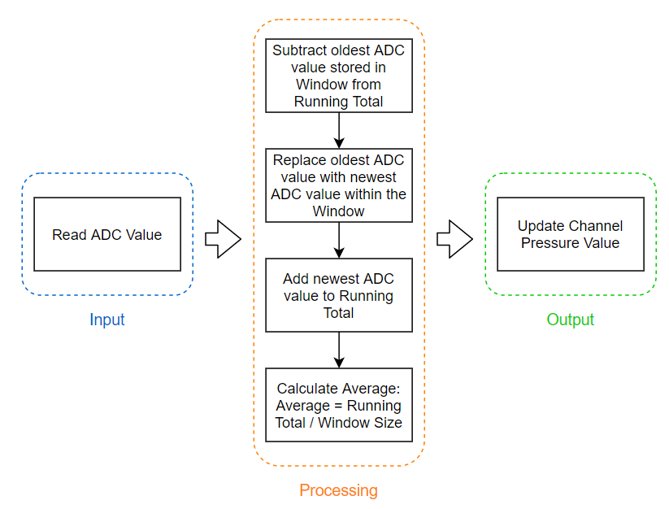


Figure 4.2.3 - PSD Rolling Average Flowchart

Figure 4.2.3 shows a simple outline of how the Rolling Average function works. A running total is kept, subtracting old values and adding new values of the window. This running total is then used to calculate the Rolling Average by dividing by the Window size (Equation 5). Before the Window is full, the Window size used is the number of readings within the window. The average will then be used as the pressure input for processing.

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Noticeably, a Rolling Average is efficient computationally due to its fixed size and also by it keeping a running total, therefore not reducing speed of the system. Alongside this, the sliding window avoids data which may distort the average from data that may not be relevant anymore by falling outside the window.

The Rolling Average is implemented using a FIFO (First-In First-Out) structure. This allows for the oldest value stored within the structure to be overwritten once the window element size has been met, allowing for efficient use of memory. A buffer using an array was created, pointing to the start element of the array once the last element of the array had been written to. When using the Rolling Average, the ADC values are smoothened, removing erratic behaviour and noise. Increasing the accuracy and reliability of the Pressure Readings, hence producing meaningful data used for decision making within the PSD.

#### Pressure Conversion

Within computation of the PSD, only one type of pressure unit is used – MPa. Therefore, when the ADC readings (a value between 1675 – 2668, discussed in section 4.1.4.3) have been obtained and smoothed, the reading is converted into MPa. The working pressure range of the PSD is 0-1MPa, this means to convert the ADC reading to MPa, the ADC reading is divided by 1000. This is due to the working resolution of the ADC from the input circuit by the PSD being 0 -1000, when divided by 1000 it will give a range of 0-1MPa, allowing for the input ADC readings to now be able to be used within the PSD processing functions and to view by the User.

#### Peak/Bottom Hold

Now a useable and reliable Pressure reading has been acquired, the PSD can perform the Peak/Bottom Hold function as stated in REQ\_110 (see Appendix A). The Peak/Bottom Hold function will store the highest (Peak) and lowest (Bottom) measured pressure readings (see **Error! Reference source not found.**). This is of particular importance for the User, allowing them to see the certain pressure which caused stress or damage after the event. The Peak/Bottom Hold readings are also able to be reset by the request of the User via the PSA. Once reset, the next measured pressure will be the new Peak and Bottom Hold Pressure.

#### Zero Clear

To further increase the accuracy of the input pressure readings, a sensor calibration function has been created for the pressure sensors – Zero Clear. The pressure sensors are prone to an effect called Zero Drift. This is due to after a long time of use or a pressure exceeding the maximum operating pressure of the sensor, the diaphragm used within the pressure sensors can be permanently deformed. This causes the deflection range of the strain gauge to be altered, causing the zero-pressure output value to drift, ultimately offsetting the pressure sensors characteristics (seen in Figure 4.2.4). Leading to incorrect readings to be measured and undesired operation of the PSD to occur.

To calibrate the sensors, this offset needs to be removed. The Set Zero function will be executed by the User when zero pressure is being applied. For Gauge pressure sensors, the PSD stores the ADC input value as an ADC zero offset. For Absolute pressure sensors, the PSD will subtract the Absolute ADC reading offset (100 or 0.1MPa) from the ADC input value at zero applied pressure. These ADC zero offset values will then be subtracted from the subsequent ADC pressure readings (see Equation 1), removing the Zero Drift. The functionality described can be seen below.

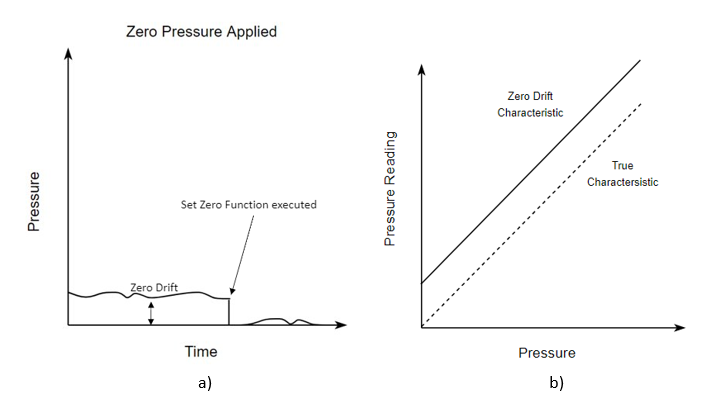


Figure 4.2.4 - a) Zero Clear Functionality, b) Zero Drift Sensor Characteristic

By measuring a parameter of the pressure sensors which indicates sensor wear in the Set Zero function – the Zero Offset, it can lead to the detection of faulty sensors and essentially perform predictive maintenance, this has been realised as an additional function. When the ADC Zero Offset exceeds a defined fault threshold, the PSD will notify the PSA, in turn notifying the User. The fault threshold defined within the PSD is 15% of the total range of the pressure sensors – an ADC value of 1861. Predictive maintenance is beneficial to the system and end-User as it will reduce downtime, increase reliability also increasing the life cycle of the system.

### Data Processing

Within the PSD, the pressure readings are used to produce useful analogue and digital outputs. The processing functions are the decision-making aspect of the PSD allowing for the correct pressure switch functionality to occur and produce a correct, useful output. The PSD operates either in Hysteresis mode or Comparator mode. These modes will determine the state of the digital output of the pressure switch.

Firstly, Hysteresis mode uses two parameter values, Hi and Hysteresis. The Hi setpoint represents the pressure the PSD will turn ‘ON’. The Hysteresis value is a value used to reduce output chattering; this is when the switch output is changing state at a high frequency due to pulsation [3, pp.11]. It achieves this by offsetting the ‘OFF’ point by the Hysteresis value.

Table 4.2.3 - Hysteresis Mode State Table

|  |  |  |
| --- | --- | --- |
| **Input** | **Digital Output State** | |
|  | **Normally Open** | **Normally Closed** |
| Pressure < Hi - Hysteresis | Low | High |
| Pressure > Hi | High | Low |

Table 4.2.3 outlines the digital output state whilst in Hysteresis mode. A graphical view of the operation in both Normally Open and Normally Closed switch mode can be seen below in Figure 4.2.5.

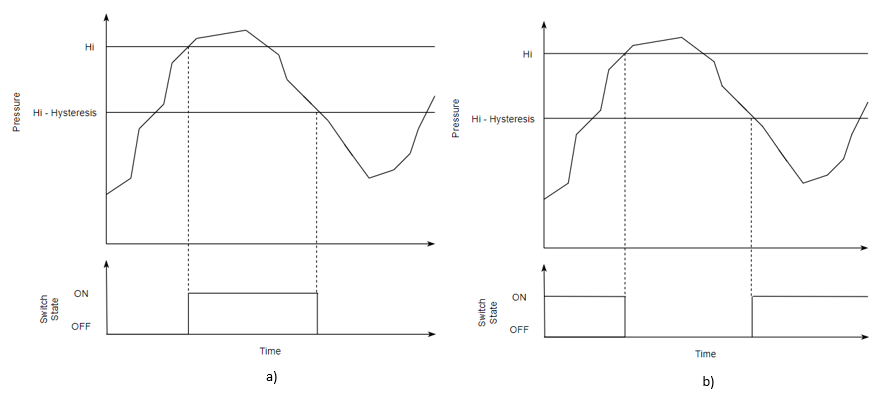


Figure 4.2.5 - Hysteresis Mode Functionality: a) Normally Open, b) Normally Closed

Comparator mode has a window-like functionality, turning the switch ’ON’ when between the Hi and Lo setpoints. Comparator mode uses three parameter values, Hi, Lo and Hysteresis. The Hysteresis value is used again to offset the ‘OFF’ value, to reduce output chattering.

Table 4.2.4 - Comparator Mode State Table

|  |  |  |
| --- | --- | --- |
| **Input** | **Digital Output State** | |
|  | **Normally Open** | **Normally Closed** |
| Pressure > Hi + Hysteresis | Low | High |
| Pressure < Hi | High | Low |
| Pressure > Lo | High | Low |
| Pressure < Lo - Hysteresis | Low | High |

Table 4.2.4 outlines the digital output state whilst in Comparator operation mode. A graphical view of the operation in both Normally Open and Normally Closed switch mode can be seen below in Figure 4.2.6Figure 4.2.5.

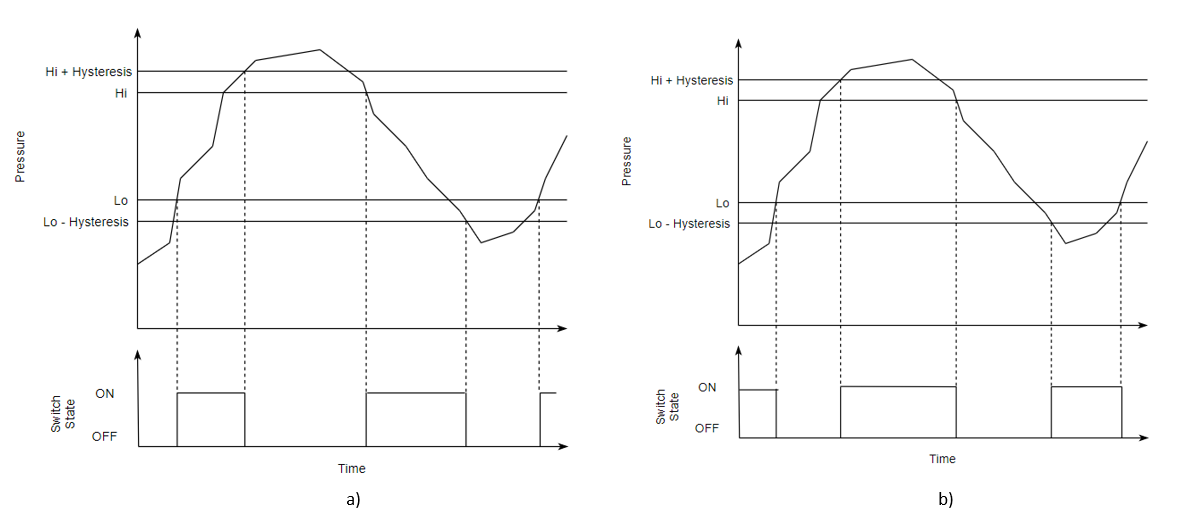


Figure 4.2.6 - Comparator Mode Functionality: a) Normally Open, b) Normally Closed

An important aspect of the PSD processing design is the timing of execution. The *response time* parameter is used to determine when the processing of the sensor inputs is executed, having periods of 2ms, 20ms, 50ms, 100ms, 200ms, 500ms (seen in Table 3.2.2). As seen in **Error! Reference source not found.**, the operation mode is executed only when the *runtime* (built-in program runtime since last execution), is greater than or equal to the *response time*. By choosing this type of design, it allows for reduced frequency of computation, improving power and time efficiency, as the alternative would be to compute the digital output state every sensor reading which would be much more computationally expensive.

### Output

Once the operation mode function (Hysteresis or Comparator) has been carried out, the outputs must be set. The output of the PSD comes in the form of a digital output for each channel, consisting of a digital signal, green and red LED; as well as an analogue output for each channel.

#### Digital Output

The digital output of the PSD is the ‘switch’ functionality, being either ‘High’ or ‘Low’ dependent on the operation mode and switch mode (normally open/normally closed) parameters. The digital output is set only upon a state change of the digital output, optimising the system. The physical output is lighting a red or green LED (dependent on the LED state parameter), to signify the switch state as well as a 3.3V voltage signal. This signal is then the input to the Output Circuitry for the digital output.

#### Analogue Output

The analogue output is a voltage signal which is equivalent to the ratio of the pressure to the operating pressure range of the PSD. Unlike the digital output, it is updated every *response time* period. The analogue output has two output ranges: 1-5V or 0-10V (seen in **Error! Reference source not found.**).

To produce an analogue output, the Arduino uses the output circuit containing a DAC discussed in section 4.1.6, the Arduino sends a value between 0 – 4096 via i2c communication. To convert the input pressure to a meaningful analogue output value (between 0-4096), the following equations are used:

For analogue output range, 1-5V:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | (3) |

Where:

1V Duty Offset = 819

New Duty Range = 3277

Max Duty = 4096

Equation 3 will output a duty cycle value in the range of 0 – 2048, allowing the DAC to output a range of 0.5-2.5V. Then passing through the operational-amplifier with a gain of 2, the range then becomes 1-5V. The equation offsets the duty cycle value by 1V duty cycle (819) and then ratios the input pressure with a new duty cycle range – 3277, which is the range between 819 - 4096. As discussed, due to the output circuit containing an amplifier of gain 2, the duty cycles are halved within equation 3 to account for this. Finally, the PSD works in MPa and therefore, the operating range is between 0 and 1MPa. The channel pressure can then be used directly to harness a duty cycle value relative to the input. The *min* function is used to control the maximum value sent from the PSD; in the event the PSD input pressure exceeds the maximum pressure (1MPa).

For analogue output range, 0-10V:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

Equation 4 outputs a duty cycle value between 0 – 4096, which will allow the DAC to output a voltage range of 0-5V. Once passed through the Operational-Amplifier with a gain of 2, the range then becomes 0-10V.

### Communication

For the PSD and PSA to work cohesively, a reliable stream of data must be communicated bi-directionally using a wireless communication protocol. Bluetooth is the wireless communication protocol chosen, due to its ideal range, connection-oriented nature, reliable transmission and low-energy implementation. Initially, Bluetooth Classic (Bluetooth 2.0) was chosen as the protocol of choice, however, for IOS to be implemented Bluetooth Low-Energy (Bluetooth 4.0) was needed. Therefore, both protocols have been functionally implemented.

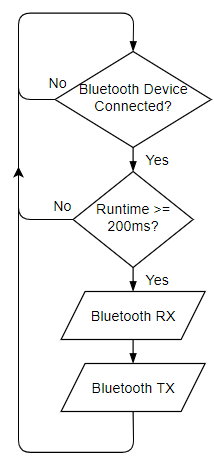


Figure 4.2.7 – Communication Function Flowchart

The PSD has several checks to preform before the Bluetooth TX and RX functions are executed (see Figure x). Firstly, it will check if a Bluetooth device is connected. With the classic Bluetooth implementation, the transceiver module has a state pin which if ‘High’, denotes a device is connected. For BLE, it checks if any central devices are connected to the Switch’s service. Once a device is known to be connected, periodic transmission and checking of the RX buffer is executed every 200ms.

The Bluetooth TX function will send the Live Pressure values for each independent channel. Peak and Bottom values will be transmitted upon value change, due to it happening less frequently than the TX period. The Bluetooth RX function will receive commands and update the parameters values stored on the PSD. A command table was created to allow cohesive development between the PSA and PSD – see Appendix A, allocating every transmission of data between the PSA and PSD a unique identifier used by both forms of communication. The way in which the TX and RX functions are realised is different for each protocol which will now be discussed.

#### Bluetooth Classic

For Bluetooth Classic, upon initial connection, the PSA will be sent the PSD’s stored parameter values to initialise with. This process is called App Parameter Setup, and it has its own command – 250 (see Appendix E).

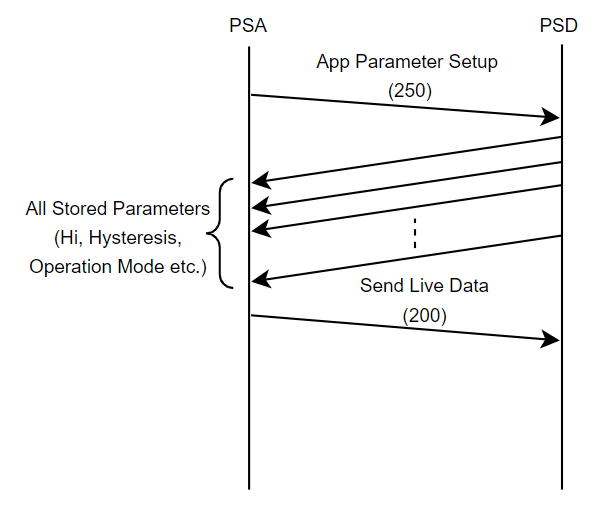


Figure 4.2.8 - App Parameter Setup Communication Diagram

The process starts with the PSA sending the App Parameter Setup packet, the PSD will then send all parameters stored to the PSA. Upon successful receipt of all parameters needed for initialisation, the PSA will acknowledge this with a ‘Send Live Data’ packet (seen in Figure 4.2.8 - App Parameter Setup Communication Diagram. This acknowledgment process is implemented to make sure that all parameters are received successfully. During integration the PSA would crash when receiving data on the settings screen, therefore a ‘Stop Live Data’ packet was also added to implement a form of flow control as seen in Figure 4.2.9 - Send-Stop Live Data Flow Control Diagram.

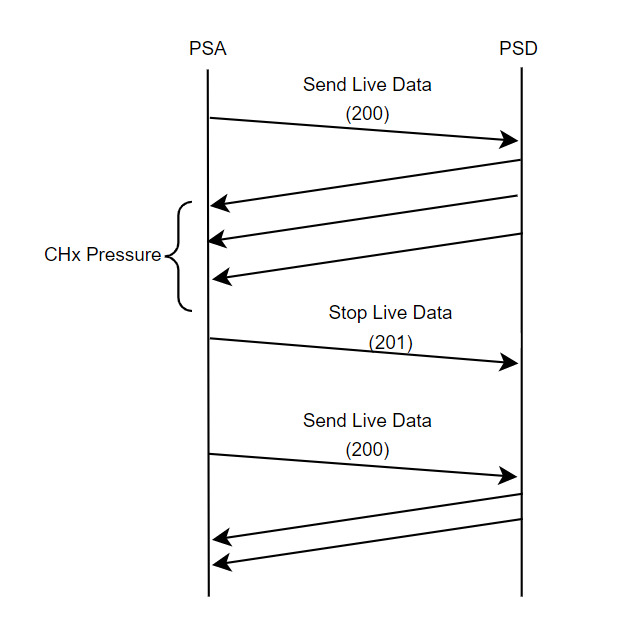


Figure 4.2.9 - Send-Stop Live Data Flow Control Diagram

When using Bluetooth Classic, the PSD communicates with the PSA using a defined fixed packet structure – seen in Figure 4.2.10.



Figure 4.2.10 - Bluetooth Classic Packet Payload Structure

The packet is of fixed length and is made up of 3 bytes, a command byte and 2 data bytes. The command byte will be a unique identifier, corresponding to a specific operation to carry out (see Appendix E), using the data. The data within the packet will be a 16-bit number, and during encapsulation of the packet it is split into its MSB and LSB forms due to Bluetooth using a byte-stream for communication. To manipulate the data the following bit shifting equations are used:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |
|  |  |  |
|  |  | (6) |

The data being sent needs to be 16-bits in length due to the use of floats within the PSD (e.g. Live Pressure = 0.234MPa). A float on the Arduino is stored as 4 bytes and has a precision of 6/7 decimal places. The maximum *display resolution* of the PSA is 3 decimal places (see Table 3.2.2 - PSD Stored Parameter Variables), therefore, a unique solution was created. Multiplying a float by 1000 and rounding will create an integer value of the float with 3 decimal place resolution. The maximum pressure the PSD operates at is 1MPa, meaning the maximum integer value of the float will be 1000 which is represented by 2-bytes (16-bits).

The Bluetooth Classic RX function checks the RX buffer of the Arduino and will then reconstruct a packet 3 bytes at a time. The RX buffer is read sequentially until empty, reducing the chance of packet de-synchronisation (e.g. identifying the LSB of data as the command byte). The data’s MSB and LSB are concatenated to create a 16-bit value. This value is then divided by 1000 to then get back to the original float value sent by the PSD, this process is only carried out on pressure values (e.g. Live Pressure, Peak Pressure, Hi Setpoint Pressure etc.).

#### Bluetooth Low-Energy

The Bluetooth Low-Energy structure is different to that of Bluetooth Classic. BLE advertises packets to any devices which connect to its Service using its Universally Unique Identifier (UUID). The BLE communication is structured so that the PSD is the ‘Peripheral’ device, advertising packets to its Service – Switch (seen in Figure 4.2.12).

The PSA will be the ‘Central’ device within the network, being able to read all the characteristics being advertised by the PSD; also, being able to update some characteristics with exclusive write access – see Appendix E. The structure of the UUID of the Service and Characteristics of the PSD can be seen in Figure 4.2.11 - PSD Service/Characteristic UUID Structure.

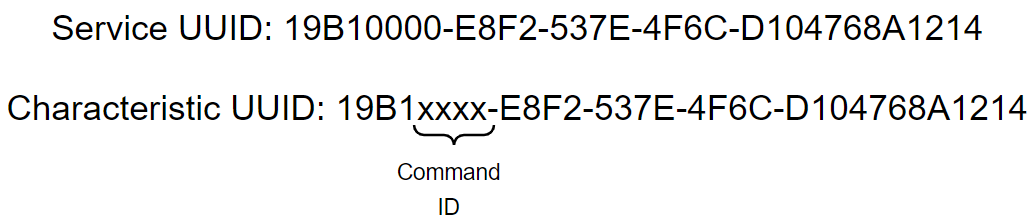


Figure 4.2.11 - PSD Service/Characteristic UUID Structure

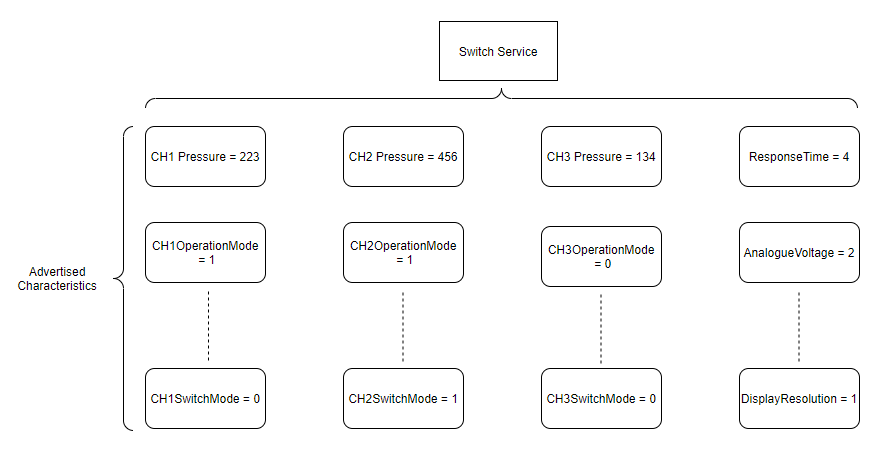


Figure 4.2.12 - PSD Service-Characteristics Structure

Within the Switch service, there are multiple characteristics with UUIDs corresponding to the command table in Appendix E, seen in Figure 4.2.12 - PSD Service-Characteristics Structure.

PSA initialisation is simpler using BLE, when the PSA connects to the PSD Service it will read the parameter characteristic values to initialise itself/get previous user parameters. Similar to Bluetooth Classic, the PSD transmits data every 200ms and only if a Central device (the PSA) is connected to the PSD Service UUID. The packet structure is handled by the ArduinoBLE library [4], therefore, the process is a simple write to the characteristic UUID to update any pressure values/parameters. The characteristics are advertised to the PSA connected to the PSD BLE Service and the PSA can pick and choose which characteristic data it will read, as well which characteristics to update (if write capability enabled).

The PSD checks periodically (see Figure x) whether parameter values have been updated by the PSA, if so, it will then update its stored parameter values to the new value. Otherwise, no action is needed, optimising the Bluetooth RX function.

# Pressure Switch App Build

## Android App Build

### The GUI

After finalising the design of the application, the next step was to create the GUI in Android Studio. Android GUI can be created using XML documents specifying the layout of the different components in the application, to which the event handling code could be attached at a later point.



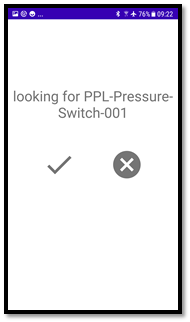
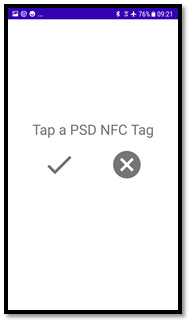
Image on the left is a screenshot of the layout of the connection page. No design alterations have been made.

Image in the centre shows the layout of the home screen. In order to conserve screen space, we opted to use a small sliding window in the top left corner of the screen. The user can slide horizontally to reveal the three options, “Bottom Hold”, “Live”, and “Peak Hold”, which will display the corresponding values just to the right of the sliding window.

Image on the right shows the settings page. Slight alterations have been made to the design. All the settings in the settings screen can be chosen from a dropdown menu containing all the available values. We chose this approach for simplicity, and to make the design consistent throughout the screen.

### NFC Functionality

Adapting the code from [<https://ssaurel.medium.com/create-a-nfc-reader-application-for-android-74cf24f38a6f>] allowed us to implement NFC functionality in the application. We then purchased some NFC tags to work with.



Leftmost picture shows a screenshot of the application when it is first loaded. The user is prompted to tap an NFC tag in order to connect to a pressure switch.

The center picture shows a screenshot of the application once an NFC tag has been successfully read, and the app is searching for a pressure switch with the specified name over Bluetooth. This particular switch was given the name “PPL-Pressure-Switch-001". The number at the end can be used to uniquely identify the pressure switches, although in this case it was included simply for demonstrative purposes.

The rightmost picture shows the prompt the user is presented with once a pressure switch with the specified name has been found. Pressing the check button would then take the user to the home page of the application, while the cross button will simply close the application.

### Bluetooth Functionality

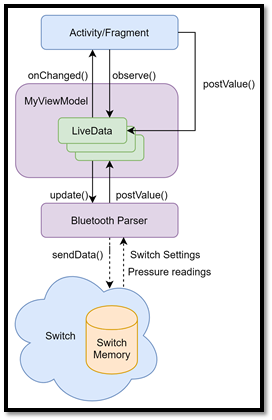
Following the guide from official Android documentation [<https://developer.android.com/guide/topics/connectivity/bluetooth>] allowed us to implement Bluetooth functionality in the application.

As the Bluetooth read and write methods are blocking, in order to make the experience of the application as smooth as possible, the Bluetooth functionality was implemented on a separate thread. This allows the data to be transferred asynchronously between the pressure switch and the application.

We decided to transmit data between the pressure switch and the application in the form of three-byte long data packets, with the first byte containing the op code, and the remaining two containing the value [described in detail in section...]. Furthermore, the pressure values are going to be transmitted from the pressure switch in megapascals, and any conversion between the different pressure units is to be done directly in the app. This seemed to be the most logical approach, as this would guarantee consistency.

### The Home Screen

Once a Bluetooth connection has been established, the pressure switch transmits all the current parameters [described in detail in section...] to the application, which then displays it. This is accomplished using LiveData, which is an observable, lifecycle-aware data holder class in Android [<https://developer.android.com/topic/libraries/architecture/livedata>].

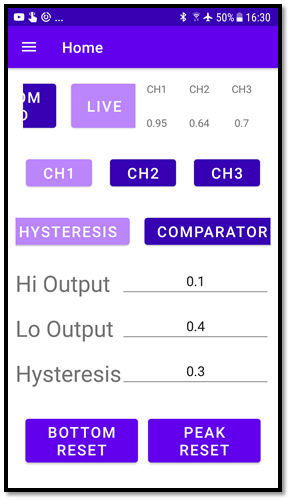


Above is a picture showing how LiveData is used in the application. The switch sends data to the application, which is parsed by the Bluetooth parser on a separate thread. The ViewModel contains LiveData holder objects, which are observed by the Activities and Fragments.

When a new value is parsed by the Bluetooth parser, for example for the live pressure reading of channel 1, the corresponding LiveData object is updated using the postValue() method. This updated notifies all the observers that the value has changed, triggering the onChanged() method of any observer. The onChanged() method inside the Activities and Fragments then updates the corresponding UI element to reflect the change. This process is done continuously, which allows the app to display updating values in real time.

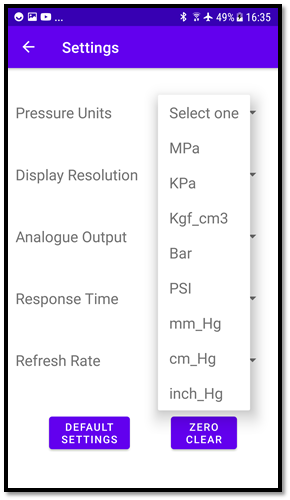
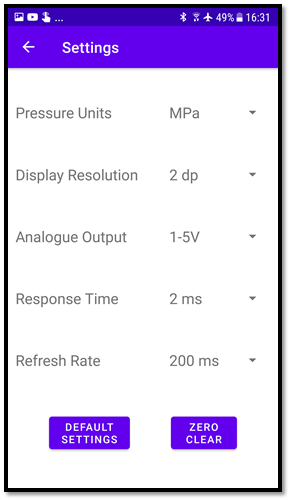
Similarly, when a user changes the value of a setting inside the application, for example the Hi Output of channel 1, the corresponding LiveData object is updated using postValue() method inside the ViewModel, and the sendData() method of the Bluetooth parser is called, which notifies the pressure switch of the updated value.

The image below shows a screenshot of the Home screen of the application, once it has been successfully connected to a pressure switch.



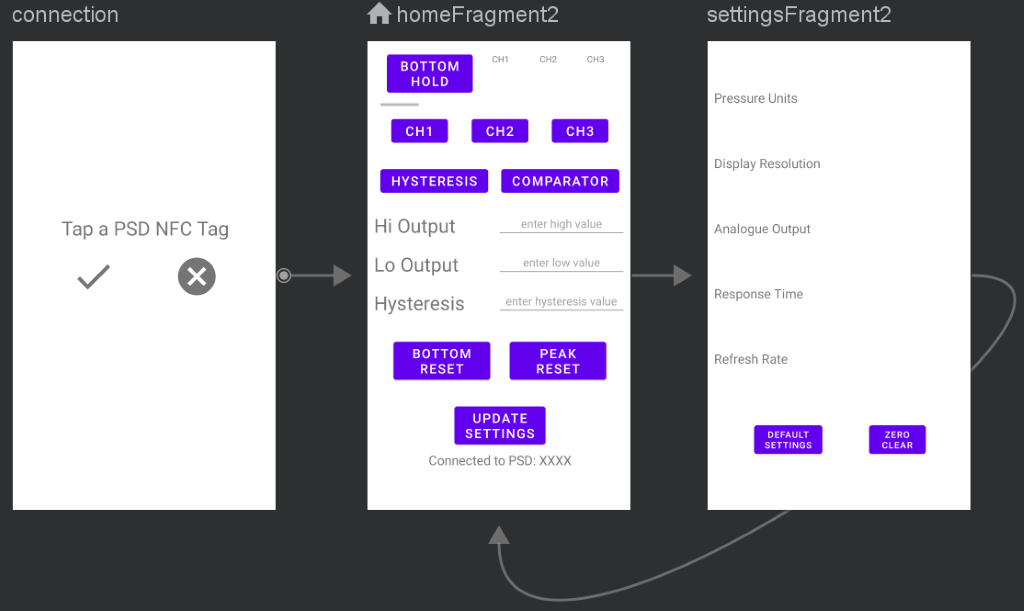
### The Settings Screen

The settings screen was implemented using the same approach. Below are screenshots of the Settings screen in the application. The user is presented with a drop-down menu, containing the pre-existing values that are available for the selected setting.



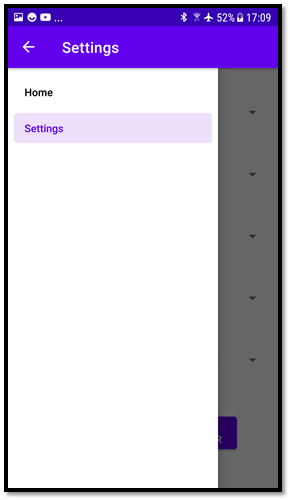
### Navigation

Once the Home and Settings screens have been implemented, the next step was to implement the navigation between them. This was achieved using the Android Navigation UI [<https://developer.android.com/guide/navigation/navigation-ui>]. Below is an image showing the navigation graph.



The navigation graph shows the destinations of the application, which are screens that are accessible. Different destinations can be accessed by doing the corresponding actions, which are represented with arrows in the graph.

In this app, the action that takes the user from the Connection screen to the Home screen is initiating a Bluetooth connection with a pressure switch and pressing the tick button. Once at the Home screen, the user can navigate to and from the Settings screen via the navigation drawer, which becomes visible by swiping from the left of the screen, pictured below.



This deviates slightly from the original design, which had the navigation drawer become available by swiping from the right, however that didn’t seem as natural nor as intuitive, so this approach was taken instead.

# Pressure Switch Device Testing

## Hardware Testing

### Development and Testing

The circuit was designed and tested according to the section of the circuit based on the priority of the function of the circuit section. As the pressure signals from the pressure sensors are the input to the pressure switch, it will dictate the design of the rest of the circuit. Therefore, the interface circuit for the pressure sensor was the first circuit to be designed and tested along with the regulators circuit that provides power to whole circuit. This then followed by the designing and testing of the digital and analogue output circuit. The sequence ensures the behaviour of the pressure sensors is understood before designing the output circuit which depend on the pressure reading from the pressure sensors. This also ensure the integration of the switch software can be test with the switch hardware after circuit for every section is finished. Figure 5.1.1 shown the development process of the pressure switch circuit.

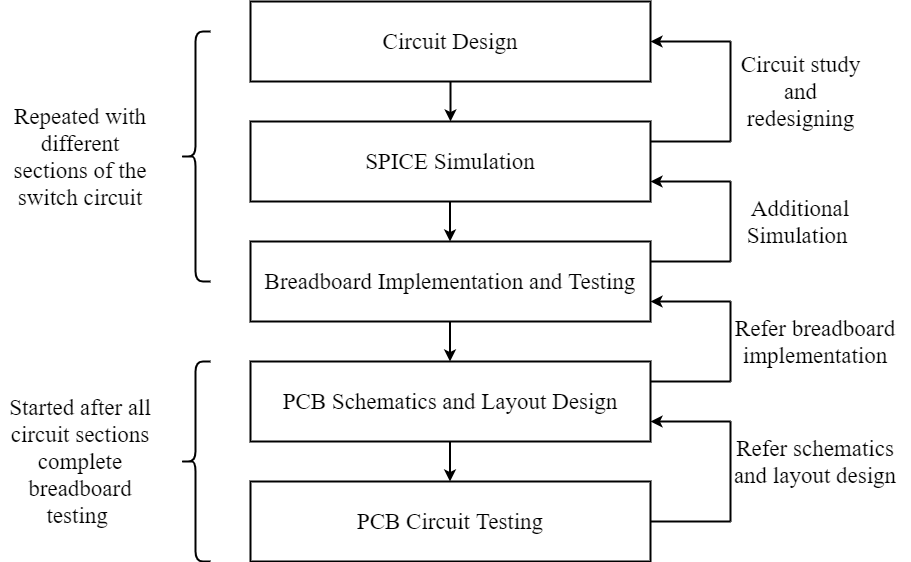


Figure 5.1.1 - Switch Circuit Development Flowchart

The designing of the circuit started with study of the circuit functionality required by the specifications. Then, the circuit design based on the circuit functionality is sketch. The process continued with the selection and comparison of different components from various manufacturer and component version. After the circuit designed has been sketch and the components of the circuit of the circuit has been determined, the circuit is first simulated in software before the components is procured for the breadboard testing and implementation. This step then repeated for another section of the circuit. In a case where there a problem found in the breadboard circuit during the testing, the simulation result will be revisited again to determine the cause of the problem before redesigning is done.

After the design of the whole circuit implemented in the breadboard, the different section of the circuit then integrated into one printed circuit board (PCB). The PCB schematics and layout design begin after the all the breadboard testing completed for all the circuit sections. This then follow by the PCB testing which may require further redesigning when error found.

### Circuit Simulation

The circuit simulation was performed in SPICE simulation software. The simulation allowed behaviour of the designed circuit to be determine and different brand of the same component to be tested before the physical implementation took place. LTSpice is the main simulation tool used for the circuit simulation but other tools such as TINA TI and other manufacturer online based simulation was also used for the simulation in cased the simulation file only available in one format.

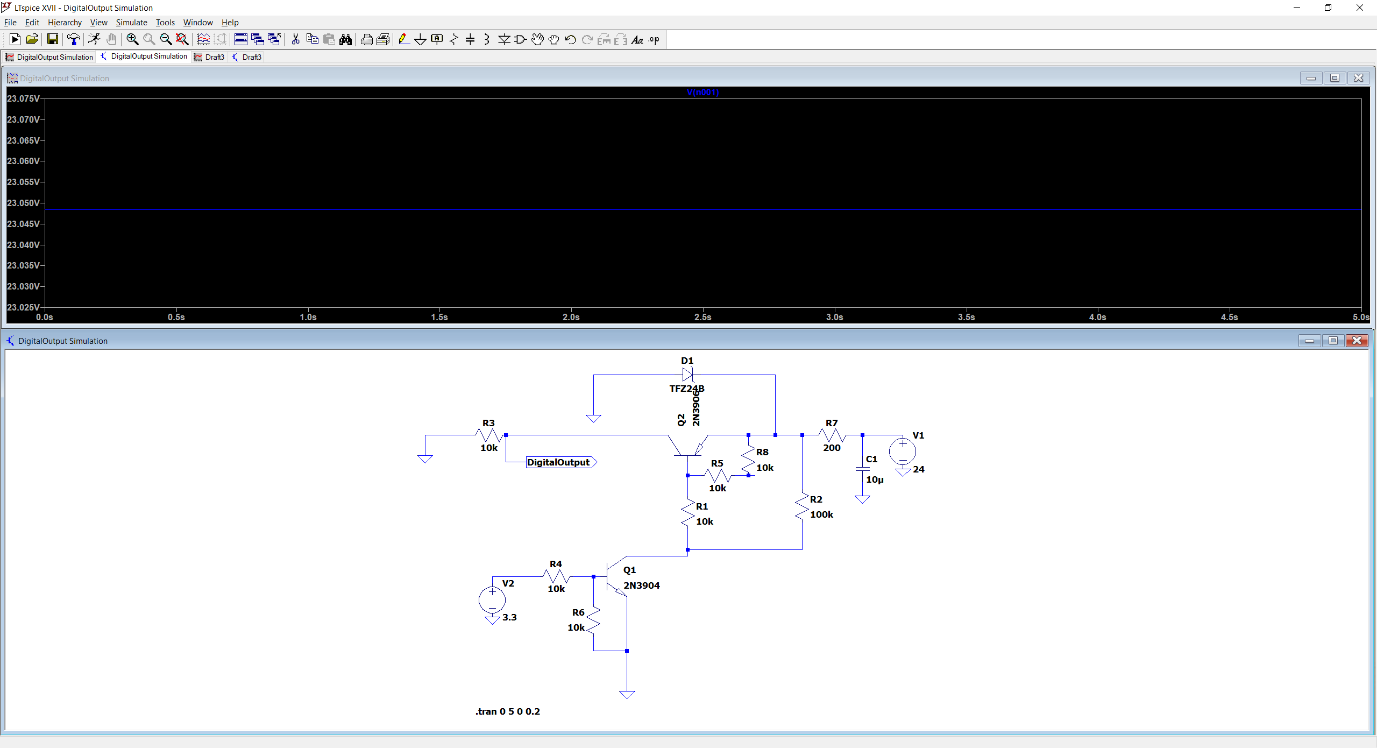


Figure 5.1.2 - LTSpice Circuit Simulation for Digital Output Circuit

### Breadboard Implementation and Testing

The physical test used the test setup shown in Figure 5.1.3, the pressure source to the pressure sensor is generated by the bicycle pump that is connected to the right most of the pneumatics hose using the Schrader valve pump adapter. This hose is connected to the pneumatics manifold tube-to-tube fitting with 3 outlets which is connected to the 3 pressure sensors.

To determine the pressure value generated by the bicycle pump, Panasonic Digital Display Pressure Sensor DP-00 (on the left of the pressure sensors) is used to compare pressures calculated in the microcontroller with the pressure display by this pressure switch. The Panasonic pressure switch also used as a benchmark against the pressure switch in term of the functionality. The pressure builds up inside the setup hose after several pumps can be release using the pressure valve at the end of the hose, this pressure also can be release quickly by turning the valve head of the bicycle pump on the right.

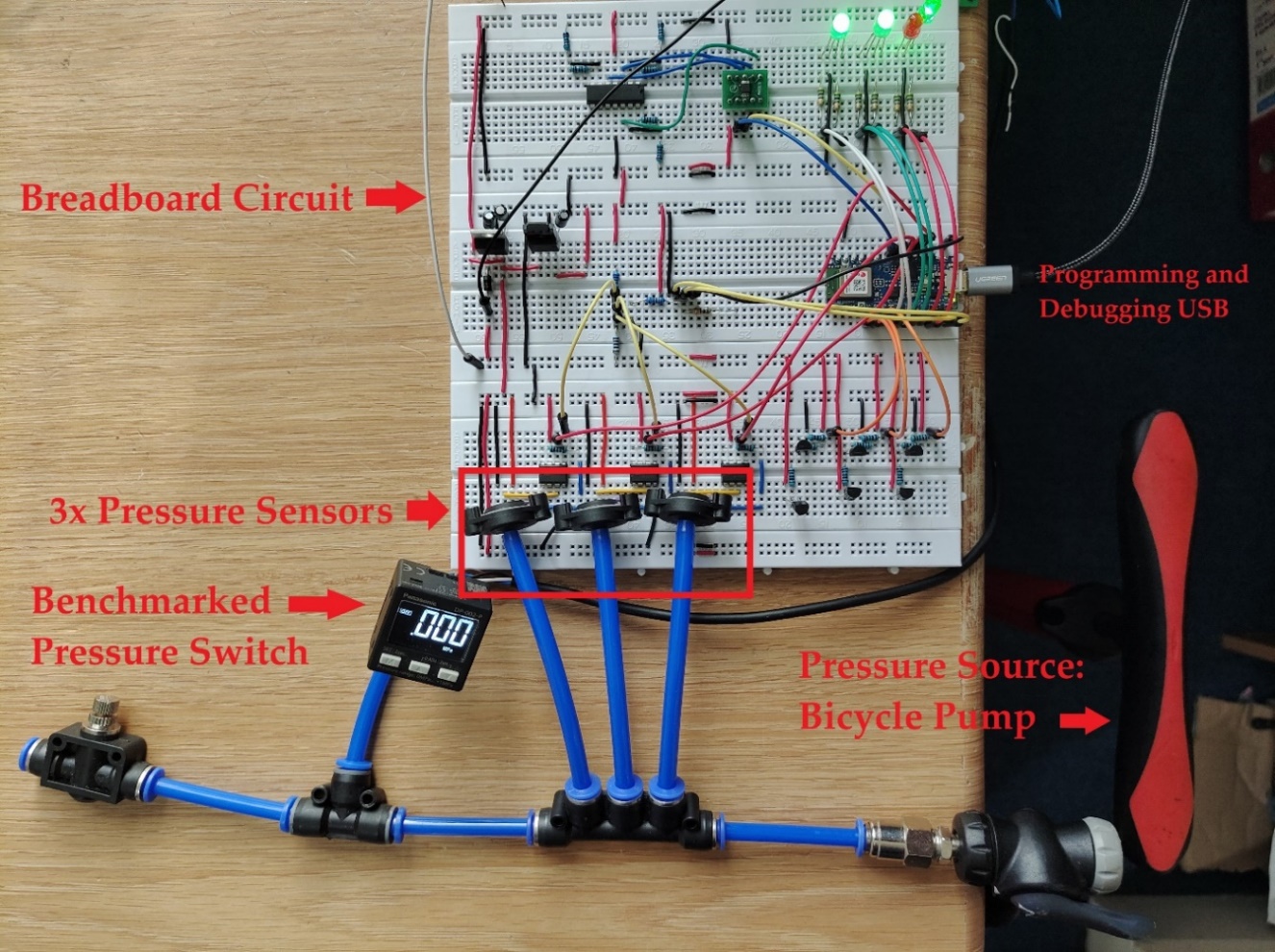


Figure 5.1.3: Switch test setup and breadboard circuit implementation

Even though, the bicycle pump has maximum pressure gauge of 11 bar, the maximum amount of pressure that can generated safely by hands is around 6 bar instead of the targeted pressure range of 10 bar. Therefore, an assumption that the pressure switch has same response at higher pressure was made.

The microcontroller board is connected to a computer for the programming and debugging using Micro USB cable during the testing. The pressure and other switch parameters are monitored using the serial debugger of the microcontroller IDE. A 12V DC power adapter is used to provide 12V supply voltage to the circuit from the mains.

### PCB Circuit Testing

The same test setup as used in breadboard implementation is used for the PCB circuit testing. The difference of the setup is the replacement of the breadboard switch circuit with a PCB circuit that has smaller footprint. A breadboard is used to mount the 3 digital output LED and to terminate the cables for the digital and analogue output signals for testing. The outputs of the circuit are tested by using the test lead of the multi-meter that clipped to a red wire. The PCB size is 6 times smaller than the previous breadboard switch circuit but still performing the same functions.

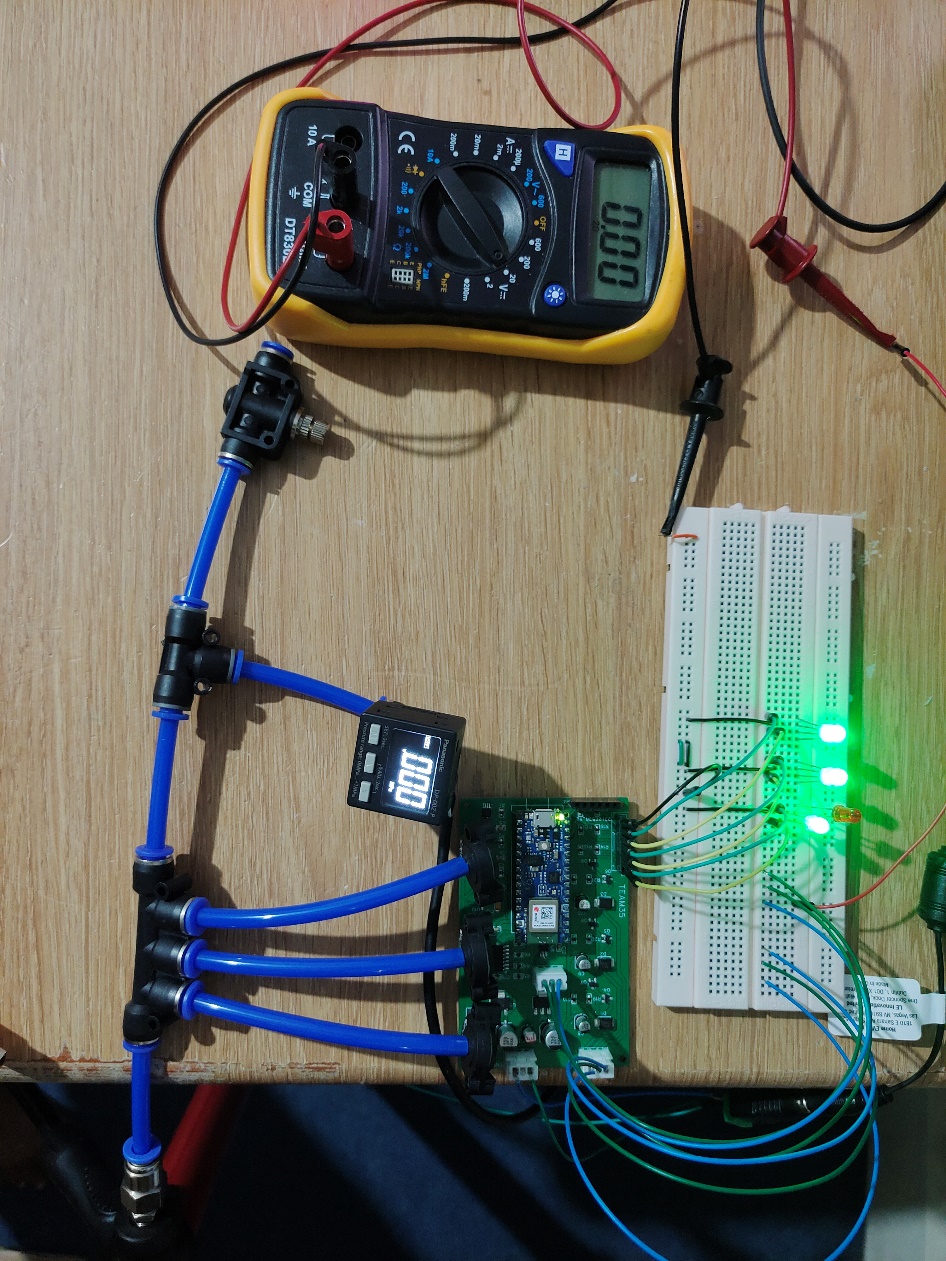
****

Figure 5.1.4 - Test setup for the switch PCB circuit

After the assembly of the PCB, a few electrical tests were done using the multi-meter to ensure the all the components work properly. The circuit is first tested with continuity test which test the presence of the bridge between the soldered components, this crucial for SMD components with pins that are close to each other such SOIC packages as this will result in short circuit of the pins and may damage the components. The second test is the resistance test, which test the resistance of each resistor in the PCB board to ensure the right resistors are soldered onto the PCB. Visual inspection also done several times at the solder joints to identify dry joint between the pins and the solder pads.

After the assembly test, the circuit functionalities test was performed, this test each section of the circuit and overall circuit functionalities. The test results are shown in the next page.

#### Circuit Testing Result

***Voltage Regulator and Current Consumption***

From LM2937 datasheet, both the 5V and 10V voltage regulators should be able to receive maximum input voltage of 26V and maximum transient voltage of 60V for less than 100ms. As most of the integration were done at home with limited equipment, all the development, testing and integration of the PSD were conduct using a 12V DC power adapter which allowed the debugging of the PSD hardware and software to be conduct at any convenience time which also relief the constraint in the hardware development side as a result of the shorter lab access duration.

With 12V DC supply that has maximum fluctuation of around 0.5V and observe duration of 5 minutes with the indicator LED turned on, the max voltage, min voltage and max deviation from the rated regulated for both regulators are shown in table Table 5.1.1. Both the regulator has small deviation when supply with 12V.

Table 5.1.1 - Typical Output Voltage from LM2937 Regulators

|  |  |  |  |
| --- | --- | --- | --- |
| Regulator | Max Voltage (V) | Min Voltage (V) | Max Deviation (%) |
| LM2937 5V | 5.02 | 5.01 | 0.4 |
| LM2937 10V | 9.98 | 9.96 | 0.4 |

The measured maximum current consumption for the whole switch is between 80mA and 90mA with both Bluetooth classic and BLE transmit at the same time with pressure applied to around 3 bar. The values are show in Table 5.1.2.

Table 5.1.2 - Total Pressure Switch Circuit Current Consumption

|  |  |  |  |
| --- | --- | --- | --- |
| Condition | Minimum | Typical | Maximum |
| Current Consumption (mA) | 47 | 65 | 93 |

***Pressure Sensor Input Circuit***

The signals from the amplifiers are offset with reference voltage of 1.35V from voltage reference IC, therefore this is the value that output from the instrumentation amplifier at atmospheric pressure which translate to 1675 ADC with 12-bit ADC resolution. The minimum and maximum range of ADC and the corresponding voltage are shown in Table 5.1.3. The maximum ADC and Voltage value at 10 bars are assumption value based on the calculated pressure sensor voltage span after amplified by factor of 10 which is 0.8V. This is because the maximum tested pressure based on test setup has limited pressure generating capability of around 6 bar or 0.6 MPa.

Table 5.1.3 - Pressure Sensor Voltage and ADC Value Range

|  |  |  |  |
| --- | --- | --- | --- |
| Min ADC | Min Voltage | Max ADC | Max Voltage |
| 1675 | 1.35V | 2668 | 2.15V |

The output signal from the in-amp are stable. However, the value receive by the microcontroller ADC will fluctuate to maximum 30 ADC values due to the noisy switching regulator on the Arduino Nano 33 BLE which affect the 3.3V ADC voltage reference, this is well discussed in [6]. The fluctuation may be small but may result in the pressure reading to go above the zero-pressure reading after the zero clear function is run. The issue could be solved in future work by replacing the board with only uBlox module and proper switching regulator layout or linear regulator.

***Digital Output Circuit***

With 12V supply voltage to the switch, the voltage level of logic low and logic high of the digital output for the 3 channels measured at the digital output header of the PCB are shown in Table 5.1.4. As expected, there voltage drop of about 0.5V due to current limiting resistor that affect the output voltage and losses in the transistors.

Table 5.1.4 - Digital Output Voltage at 12V Supply Voltage

|  |  |  |  |
| --- | --- | --- | --- |
| Channel | Channel 1 | Channel 2 | Channel 3 |
| Supply Voltage (V) | 12.23 | | |
| Logic LOW (V) | 0 | 0 | 0 |
| Logic HIGH (V) | 11.74 | 11.73 | 11.74 |

Table 5.1.5 show the maximum output current of the digital output when digital output is set at logic high. The currents were measured by grounding one of the multi-meter lead to ground and another lead clip to the output header.

Table 5.1.5 - Digital Output Maximum Output Current

|  |  |  |  |
| --- | --- | --- | --- |
| Channel | Channel 1 | Channel 2 | Channel 3 |
| Current (mA) | 41.7 | 42.2 | 42.0 |

***Analogue Output Circuit***

Both 1-5V and 0-10V analogue output will depend on the pressure measured by the microcontroller based on the ADC values. Ideally, for 1-5V output, the voltage output from the pressure switch at zero pressure will result in 1V signal output but due to the noise of the ADC values mentioned earlier, this cause the output to reflect the noise which result in slightly higher voltage at zero pressure. This also apply to 0-10V output. Table 5.1.6 show the typical value of analogue output voltage for both type of signals when zero pressure is applied to the switch.

Table 5.1.6 - Typical Output Analogue Output Voltage at Zero Pressure

|  |  |  |  |
| --- | --- | --- | --- |
| Channel | Channel 1 | Channel 2 | Channel 3 |
| 1-5V Output (V) | 1.05 | 1.05 | 1.41 |
| 0-10V Output (V) | 0.25 | 0.15 | 0.97V |

Overall, the circuit can perform the pressure sensor input reading and output digital and analogue signals with small issue with the ADC fluctuating reading which could be fix in the future work by directly using the Nina B306 microcontroller module instead of integrating whole Arduino Nano 33 BLE board on the PCB.

## Software Testing

The method of iterative functional testing was used throughout the development of the PSD embedded software. Testing the functions expected output against its actual output and identifying any inconsistencies/incorrect operation. All tests were carried out using a 2-input potentiometer circuit to simulate pressure sensor inputs within the Arduino IDE. The tests provide evidence for the desired, correct functionality of the PSD outlined in the specification document (Appendix A).

### Read ADC/ Rolling Average

Firstly, testing the ADC reads values and successfully converts them into corresponding pressure values is paramount within the PSD. Using the potentiometers as simulated pressure sensors, ADC values were gathered and converted.



Figure 5.2.1 - ADC Value Pressure Conversion

Figure 5.2.1 shows the output of the test from the Arduino serial monitor. The ADC values were read correctly and converted into their correct pressure form. Tests were carried out to ensure the Rolling Average works correctly. There are two states the Window of the Rolling Average is in: not full or full. The first test checks the rolling average functionality is correct when first being filled with ADC values.

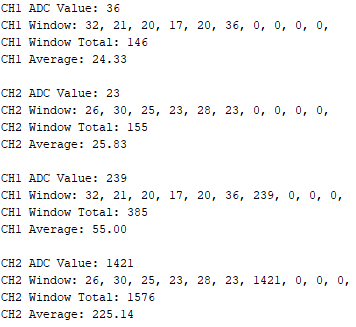


Figure 5.2.2 - Rolling Average Window Initialisation Test

Figure 5.2.2 shows the Window being first initialised and being filled with ADC values. The ADC value is read and then filled into the next Window element. In this state, the average is calculated using the number of values within the Window. The test proved the Rolling Average works correctly in this state.

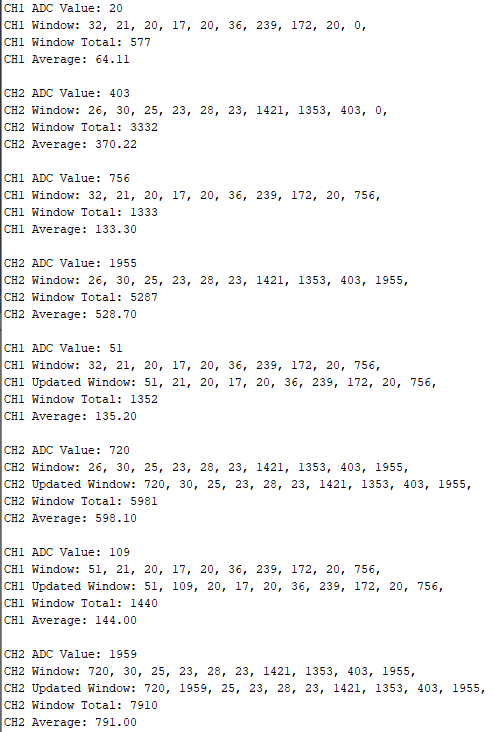


Figure 5.2.3 - Rolling Average Functionality Test

Figure 5.2.3 shows the Window being filled with ADC values just after initialisation. The ADC value is read and then filled into the next Window element. The Window is a FIFO buffer, therefore, when the last element is reached, the next ADC value overwrites the oldest value (seen in Figure 5.2.3). In this state, the average is calculated using the fixed Window size. The test proved the Rolling Average works correctly in this state.

### Peak/Bottom Hold and Reset

To meet REQ\_110 in the Specification (Appendix A), a function to hold the Peak and Bottom pressures is needed as well as, to reset these values upon request of the PSA.

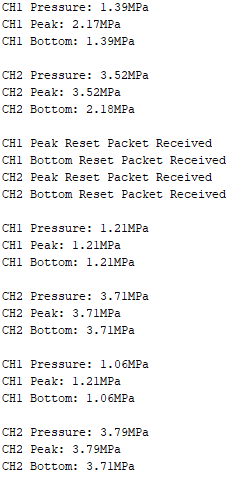


Figure 5.2.4 - Peak/Bottom Hold and Reset Test

A test was carried out with simulated pressure values from potentiometers. Figure 5.2.4 - Peak/Bottom Hold and Reset Testshows the Channel 1 and Channel 2 values being stored correctly into the relevant Peak/Bottom container.

Figure 5.2.4 also shows a Peak/Bottom reset test. The Peak/Bottom reset packets are received, then the next pressure value measured is used to overwrite the Peak and Bottom values for each respective Channel. This provides evidence for the correct operation of the Peak/Bottom Hold and Reset function, meeting REQ\_110.

### Zero Clear

To meet REQ\_120 in the Specification (Appendix A), a Zero Clear function for sensor calibration is needed, upon request of the PSA.

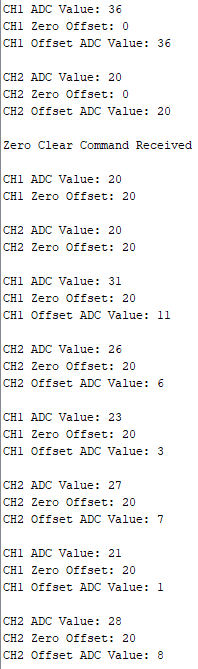


Figure 5.2.5 - Zero Clear Function Test

Figure 5.2.5 shows the Zero Clear function being requested by the PSA. This then causes each Channels Zero Offset to be the next measured pressure. Before the packet was received the ‘Offset ADC value’ for each Channel was large, however, after it is much smaller; providing evidence for increased accuracy due to calibration. The analogue output is directly proportional to the pressure measured, therefore, is also set to 0. This provides evidence for the correct operation of the Zero Clear function, meeting REQ\_120.

The addition of the Fault Detection was also tested.

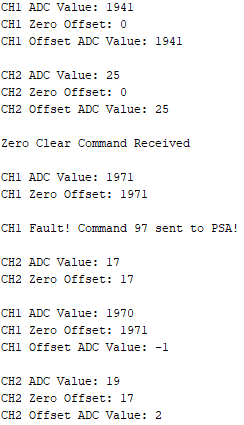


Figure 5.2.6 - Fault Detection/Predictive Maintenance Test

Figure 5.2.6 shows when the Zero Offset is above the 15% threshold, a fault packet is sent to the PSA with the appropriate command ID; providing evidence for the additional functionality of predictive maintenance.

### Hysteresis and Comparator Mode

To meet REQ\_180 and Stretch Objective\_02 in the Specification (Appendix A), Hysteresis and Comparator modes must be implemented. The test for both modes used exaggerated test values (see Table 5.2.1 - Test Parameter Values), to allow plots to be easier to analyse.

Table 5.2.1 - Test Parameter Values

|  |  |
| --- | --- |
| **Parameter:** | **Test Value:** |
| Hi | 10MPa |
| Hysteresis | 1MPa |
| Lo (Comparator) | 3MPa |
| Switch Mode | Normally Open |
| LED State | Green = On,  Red = Off |



Figure 5.2.7 - Hysteresis Mode Test (Arduino Serial Plotter)

Figure 5.2.7 shows the correct operation of the Hysteresis mode. The digital output state is correct at the relevant pressure values, alongside the LED’s lighting accordingly. This provides evidence for meeting REQ\_160, REQ\_170 and REQ\_180.

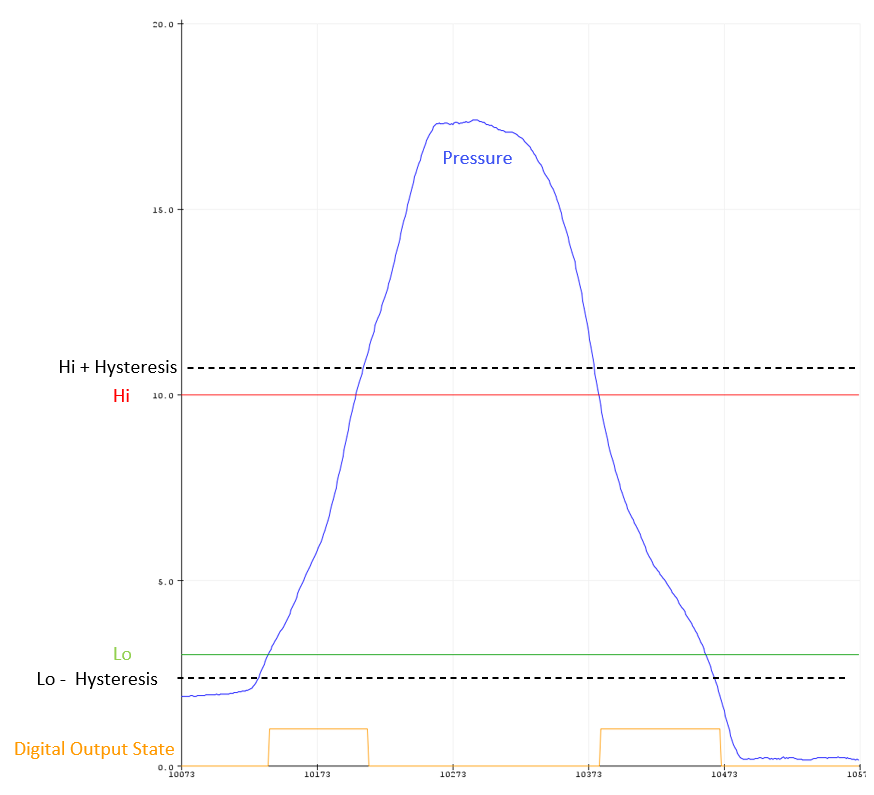


Figure 5.2.8 - Comparator Mode Test (Arduino Serial Plotter)

Figure 5.2.8 shows the correct operation of the Comparator mode. The digital output state is correct at the relevant pressure values, alongside the LED’s lighting accordingly. This provides evidence for meeting Stretch Objective\_02, REQ\_160, and REQ\_170.

### Bluetooth TX/RX

To meet REQ\_010 in the specification (Appendix A), Bi-Directional communication must be established between the PSA and PSD. Bluetooth Classic tests were carried out using the Serial Bluetooth Terminal App [5] on an Android phone, with BLE tests using the nRF Connect App [6] on an IOS phone.

Firstly, Bluetooth Classic was tested, carrying out tests for the Bluetooth TX/RX functions and the App Parameter Setup function.

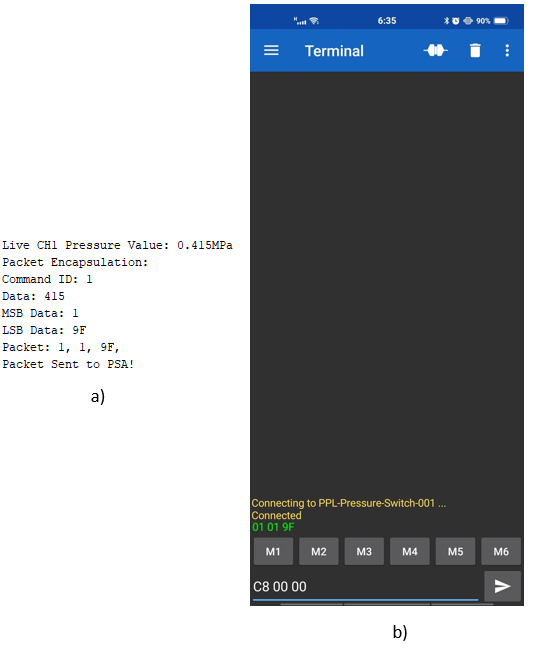


Figure 5.2.9 - Bluetooth Classic Packet Encapsulation and Transmit Test: a) PSD Serial Monitor, b) Android Serial Bluetooth Terminal [5]

A pressure value is sent from the PSD, with Figure 5.2.9 showing correct packet encapsulation. This is then received correctly on the Android phone; providing evidence unidirectional communication has now been established.

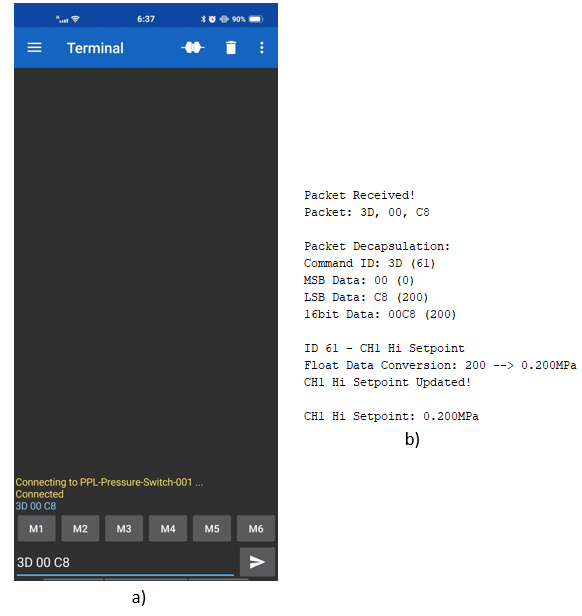


Figure 5.2.10 - Bluetooth Classic Packet Decapsulation and Receive Test: a) Android Serial Bluetooth Terminal [5], b) PSD Serial Monitor

Figure 5.2.10 provides evidence for the successful operation of the Bluetooth Classic RX function. A test value is sent from the Android Phone, in this test updating the CH1 Hi Setpoint to 0.2MPa (3D 00 C8). The PSD successfully decodes the packet (converting MSB and LSB to float pressure) and updates the correct parameter. Bi-directional communication has now been established, meeting REQ\_010.

The App Parameter Setup functionality was also tested. Figure 5.2.11 shows the successful operation of the App Parameter Setup function. The PSD receives the setup request (FA 00 00), and the PSD sends the packets to the PSA which can be seen in the green text.

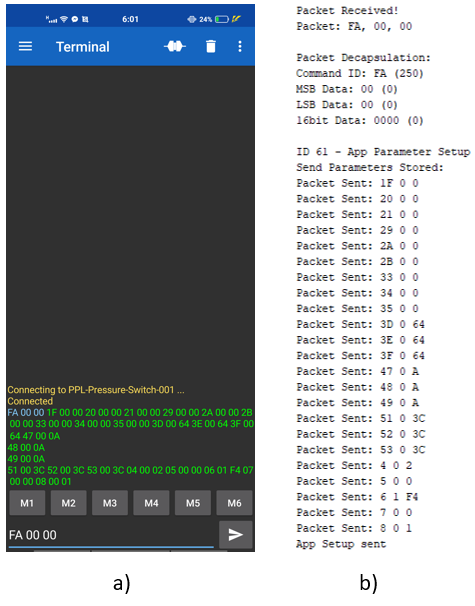


Figure 5.2.11 - Bluetooth Classic App Parameter Setup: a) Android Serial Bluetooth Terminal [5], b) PSD Serial Monitor

The BLE Service-Characteristic structure is tested for Bi-Directional communication, reading and writing to/from characteristics from the PSD.

Replicating the PSA using nRF connect [6], it connects to the PSD Service. Figure 5.2.12 shows when connected, the PSD Live Pressure characteristics being advertised as part of the Switch Service. The values for each Channel can be seen being advertised (BLE unidirectional communication).

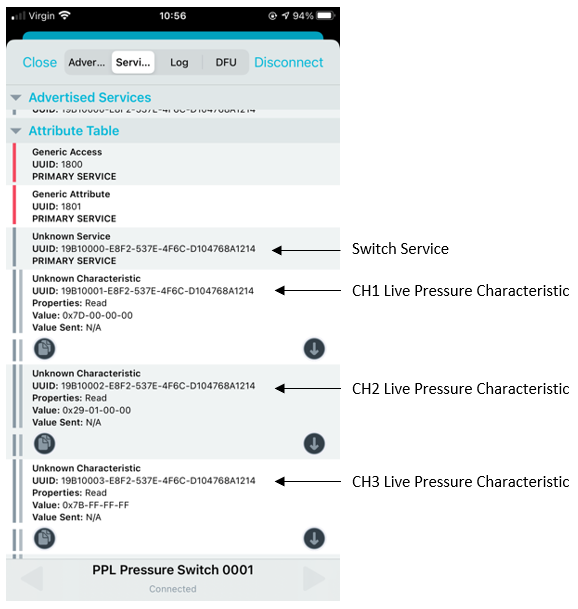


Figure 5.2.12 - BLE Live Pressure Advertisement Test using nRF Connect IOS [6]

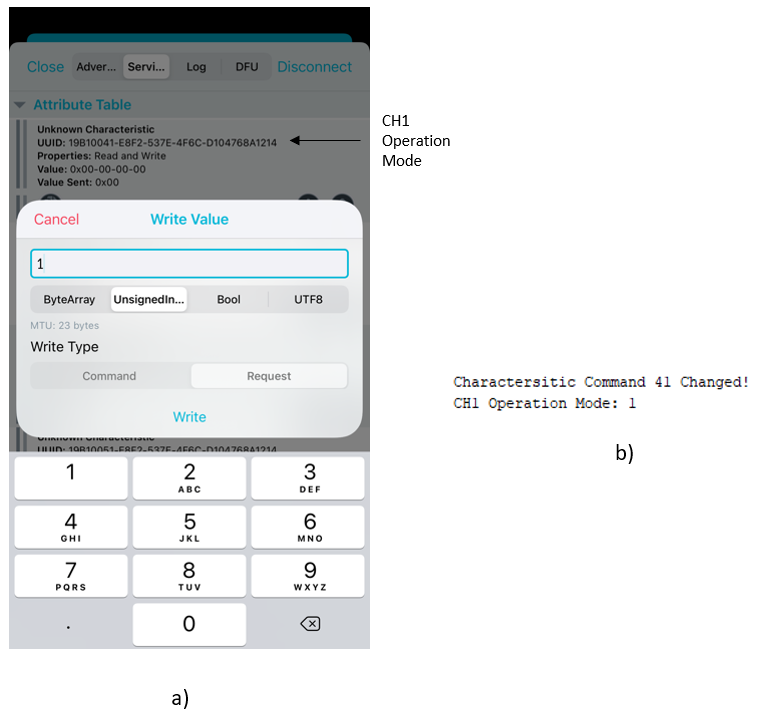


Figure 5.2.13 - BLE Update Parameter (BLE RX) Test: a) nRF Connect IOS [6], b) PSD Serial Monitor

To test the BLE RX function, the PSA (nRF Connect [6]) writes to a parameter – in this case the CH1 operation mode. Figure 5.2.13 shows the CH1 operation mode characteristic being written to with the value 1, and the PSD identifying a change has been made and updating the parameter value. Providing evidence for the successful implementation of bi-directional BLE communication; meeting REQ\_010.

### Analogue Output

To meet REQ\_150 and REQ\_190 in the Specification (Appendix A), an analogue output for each Channel with the ranges 1-5V and 0-10V ranges is required.

Using the potentiometer test circuit, pressures were simulated, and an algorithm was developed to mimic the output circuit. Figure 5.2.14 shows the correct operation of the analogue output, converting the pressure to the correct DAC duty cycle, which then can be seen to produce the expected pressure-dependent voltage.

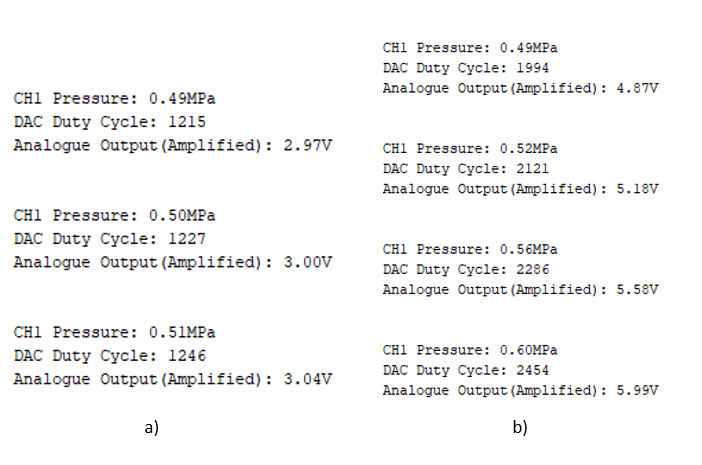


Figure 5.2.14 - Analogue Output Test: a) 1-5V range, b) 0-10V range

Figure 5.2.14 provides evidence for meeting REQ\_150 and the function is uniform across each Channel and works seamlessly, meeting REQ\_190.

# Pressure Switch App Testing

## Android App Testing

### Unit Testing

With the app design, we tried minimizing the possibility of error through design feature choice, such as using drop down menus where possible, and the use of NFC tags to initiate the Bluetooth communication. However, there is still room for erroneous user input in the Home screen of the application. The user may attempt to enter pressure values outside of the working range, for example.

# Integration

This chapter outlines the Integration process of the PSD, Hardware and Software. As well as the Integration of the Android and IOS PSA.

## PSD Hardware/Software Integration

In the Project’s Gantt Chart (see Appendix B), the PSD hardware and software development follows an Input-Process-Output-Communication (IPO-C) development model. Following this model, the hardware and software development is carried out in parallel, allowing for integration at the end of each completed development phase. This development practice is beneficial as for the PSD each service is dependent on one another, for example, the microcontroller processing (Hysteresis/Comparator mode) is reliant on the accurate pressure sensor measurement of the input circuit. Upon each stage of integration, thorough testing was carried out, repeating the PSD software tests (section 5.2) with the hardware integrated to validate correct functionality.

Once the input circuitry and software (Read ADC and Rolling Average functions) were developed, testing of the two when integrated was carried out. As the input circuit amplifies the pressure signals from the sensors, the input pressure signal will result in a larger voltage span as discussed in section 4.1.4; this then affects the pressure conversion calculation in the software. Within the software, the conversion factor (1000) is added within the Pressure Conversion to take into account the confined input voltage span. Alongside this, the Set Zero function is added within the setup phase to allow the ADC offset to be negated. Resulting in smooth, successful integration of the input process.

The external hardware and microcontroller software have an interface between the embedded code and output circuit. The processing functions (Hysteresis/Comparator modes) and pressure input are used to produce analogue/digital signals via the output circuitry. When integrating, the digital output and LED’s worked correctly with both operating modes due to the software being tested using a simulation circuit, very similar to the digital part of the output circuit. For the analogue output, the software manipulated the duty cycle for the DAC, to allow for both analogue output ranges to be satisfied. Resulting in successful integration of the process/output area.

In Evaluation, the integration between the software and hardware of the PSD worked seamlessly and cohesively. This is due to the parallel development and integration model adopted in the schedule of work. Ultimately, allowing for a fully operational PSD to be integrated ahead of schedule.

## Android PSA & PSD Integration

### Integration Testing

## IOS PSA & PSD Integration

# Management

# Conclusion

# Further Work

# Bibliography

1. "MCP4728 - Arduino Reference", Arduino.cc, 2020. [Online]. Available: https://www.arduino.cc/reference/en/libraries/mcp4728/. (Accessed: 21- Nov- 2020).
2. SMC Corporation. Operation Manual – ZSE30A(F), ISE30A. Accessed: 24/12/2020. [Online]. Available: https://www.smcpneumatics.com/pdfs\_smc\_instructional/PS-OML0003-C.pdf
3. "Arduino - ArduinoBLE", Arduino.cc, 2020. [Online]. Available: https://www.arduino.cc/en/Reference/ArduinoBLE. [Accessed: 04- Dec- 2020].
4. Play.google.com, 2020. [Online]. Available: https://play.google.com/store/apps/details?id=de.kai\_morich.serial\_bluetooth\_terminal&hl=en\_GB&gl=US. [Accessed: 22- Nov- 2020].
5. "‎nRF Connect: Bluetooth App", App Store, 2020. [Online]. Available: https://apps.apple.com/us/app/nrf-connect-bluetooth-app/id1054362403. [Accessed: 06- Dec- 2020].
6. Project Specification Document

**School of Electronics and Computer Science** **ELEC6200 MEng Group Design Project**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Project Specification and Plan**

GDP 35 Title: PPL Pressure Switch Setting App and Communications

Academic Supervisor(s): Yoshi Tsuchiya

Team Members: Lukman Kazeem, Muhammad Ariif Rozali,

Fedor Selensky, William Hazelden

Project Description:

To develop two devices (2 x pressure switches) and a Smartphone App that demonstrate how the functions for setting various parameters can be moved from the front panel of an electronic pressure switch to a new pressure switch that supports Smartphone and App connectivity.

Functional Requirements:

* ***REQ\_010 :*** Bi-directional communication between the remote ‘*Pressure Switch Device’*(PSD) and the smartphone ‘*Pressure Switch Application’*(PSA) is achieved over a Bluetooth network.
* ***REQ\_*020 :** To establish inter-device Bluetooth link [**REQ\_010**] users must perform a successful NFC read by physically touching the passive NFC tag, upon which the device entities auto-pair.
* ***REQ\_*021 :** The physical Switch Device must feature a passive NFC tag housing data the mobile uses to trigger inter-device Bluetooth paring and connection.
* ***REQ\_030* :** The Smartphone app fully supports an Android platform environment. (Additional OS such as IOS are nice to haves).
* ***REQ\_040* :** Reducing human errors when programming the device parameter settings (I.e. misidentifying Switch device) by implementing the NFC feature **[REQ\_020**] for verification purposes. (useful in cases where multiple PSDs are operating simultaneously within Bluetooth proximity)
* ***REQ\_050* :** The first UI screen in the application will provide instructions on how the user can establish interdevice Bluetooth connection. (Supporting **[REQ\_020**] and [**REQ\_040**])
* ***REQ\_060* :** Following a validated BT connection, the second screen within the mobile application depicts the interface through which live readings are reported, and switch parameters are set.
* ***REQ\_070* :** The mobile application allows the user to select the unit of pressure for all three pressure sensing channels.
* ***REQ\_080 :*** Each selectable unit of pressure requires values be reported with the following precision. Fig.1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pressure Unit** | ***MPa*** | ***KPa*** | ***kgf/cm2*** | ***Bar*** | ***PSI*** | ***mm Hg*** | ***cm Hg*** | ***inch Hg*** |
| **Precision Value** | 0.001 | 1 | 0.01 | 0.01 | 0.1 | 10 | 1 | 0.5 |

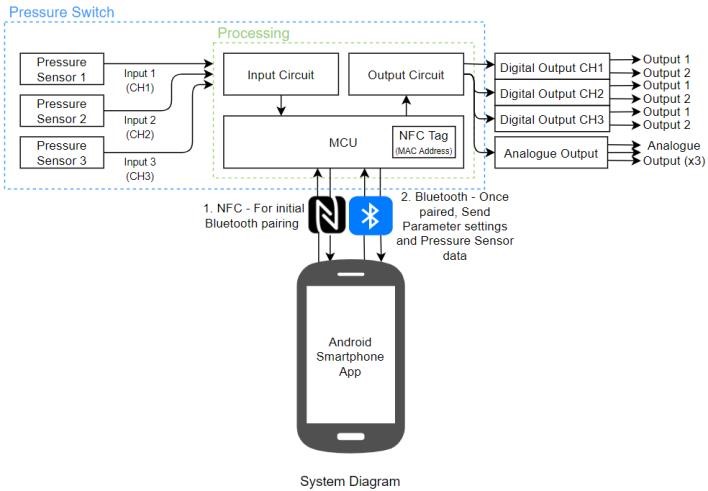
Fig.1 . Table showing the required precision of the values for each selectable pressure unit.

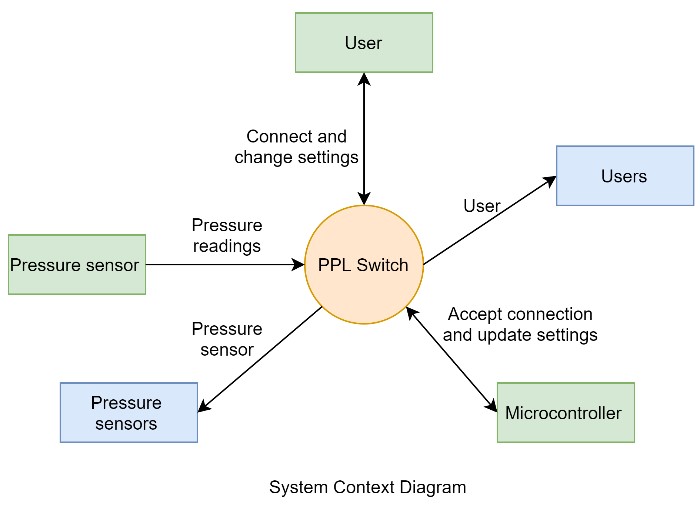
* ***REQ\_090*  :** Each of the three channels should report current/live pressure readings from each of the three channels in at a frequency of 60Hz.
* ***REQ\_100*  :** Each of the three channels operate on the same unit to report pressure values within the range 0 – 10 Bar according to the unit precision specified in [***REQ\_080]***
* ***REQ\_110*  :** Live Peak and Bottom Hold values is to be stored in PSD and reported by the Mobile Application and PEAK\_RESET, BOTTOM\_RESET functions are used to manually reset this value.
* ***REQ\_120*  :** The app offers ZERO\_CLEAR function, where the live pressure values and analogue outputs value are set to zero to compensate zero drift in pressure sensor by exposing pressure sensor to atmosphere
* ***REQ\_130*  :** The analogue and digital switch output response time is adjustable in steps (such as 20ms, 100ms, 500ms etc.) between 2ms – 500ms
* ***REQ\_140*  :** The second screen also allows for users to select the mode the pressure switch operates in (I.e. *HYSTERESIS\_MODE)*
* ***REQ\_150 :*** Each channel of PSD must support analogue voltage output signals of 1-5V and 0-10V.
* ***REQ\_160 :*** The PSD should have two digital switch outputs per channel (therefore six digital output), with selectable NC/NO per channel based on PNP transistor switch.
* ***REQ\_170 :***  The PSD should have at least two LED lights (Green and Red) fitted on per channel, to functions as the indicator for the digital switch output state in **[REQ\_160].**
* ***REQ\_180*  :** With respect to **[REQ\_140]** the behaviour of the systems pressure switch is dependent on the mode it is operating within.
* ***REQ\_190* :** PSD support 3 analogue outputs mimicking the control that would otherwise be generated by a real-life pressure switch.

Stretch Objectives:

* **Objective\_01:** Selectable PNP and NPN transistor switch digital output
* **Objective\_02:** Pressure Switch has switch Comparator Mode functionality
* **Objective\_03:** Analogue current output of 4-20mA
* **Objective\_04:** Oscilloscope-like graphical display of Pressure Sensor readings
* **Objective\_05:** Developed app for IOS

System Diagram (Hardware + Software Diagram Flow)

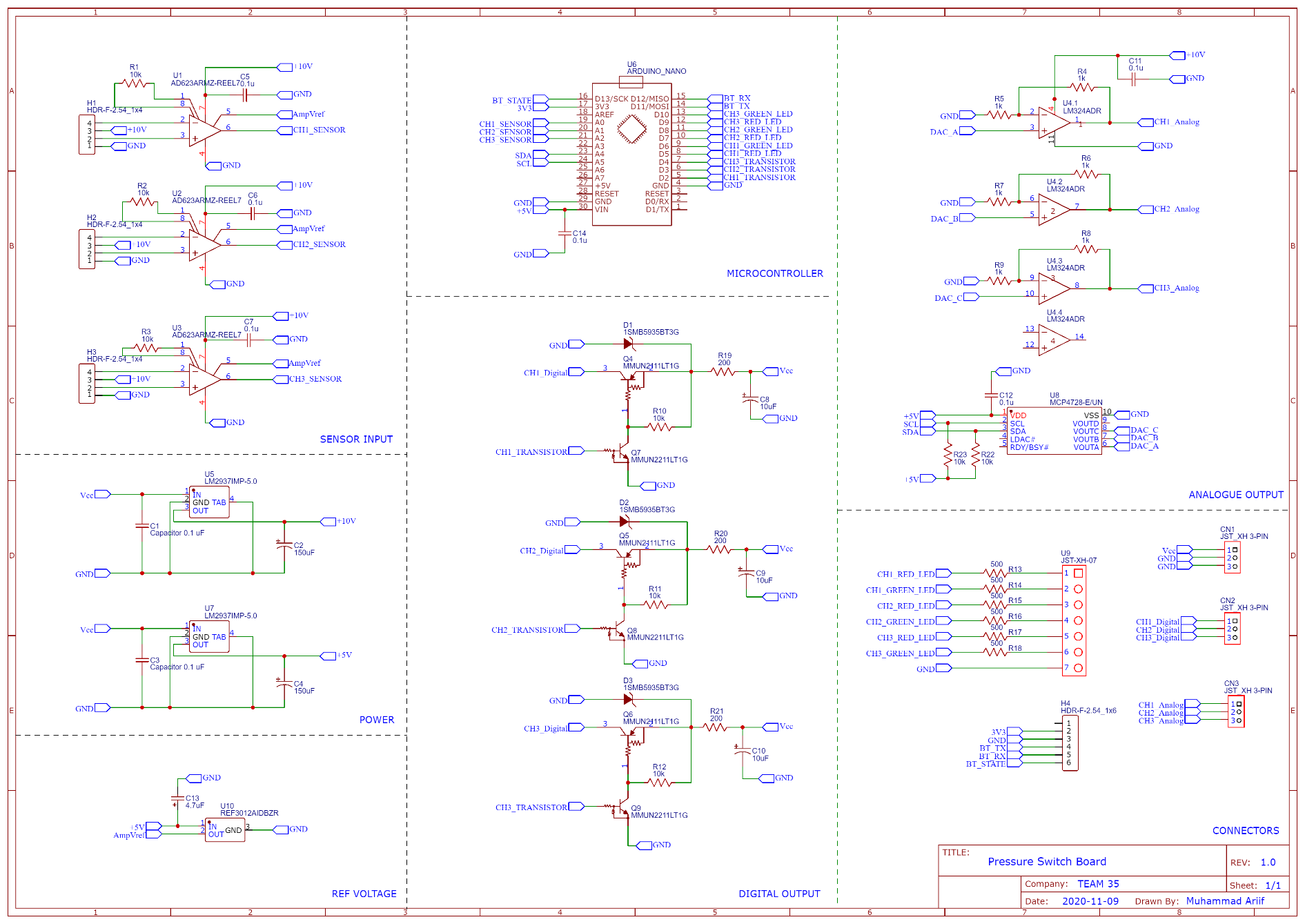




1. Gantt Chart



1. Full PCB Schematics



1. PCB Bill of Materials

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Manufacturer | Description | Part Number | Designator | Qty | Case | Price Each |
| JLPCB | PCB Board | PressureSwitchv1 | PSD | 1 | PCB | 0.3 |
| Arduino | Microcontroller | ABX00030 | U6 | 1 | Board | 21 |
| AD | Instrument Amplifier | AD623ARMZ | U1 | 3 | MSOP-8 | 4.728 |
| TI | 5V Voltage Regulator | LM2937IMP-5.0/NOPB | U5 | 1 | SOT-223 | 1.44 |
| TI | 10V Regulator | LM2937IMP-10/NOPB | U7 | 1 | SOT-223 | 1.872 |
| TI | 1.25V Voltage Ref | REF3012AIDBZR | U10 | 1 | SOT-23 | 1.1076 |
| Microchip | DAC | MCP4728-E/UN | U8 | 1 | MSOP-10 | 1.776 |
| TI | Opamp | LM324ADR | U4 | 1 | SOIC-14 | 0.2772 |
| ON Semicon | NPN Transistor | MMUN2211LT1G | Q7, Q8, Q9 | 3 | SOT-23 | 0.11 |
| ON Semicon | PNP Transitor | MMUN2111LT1G | Q4, Q5, Q6 | 3 | SOT-23 | 0.11 |
| Vishay | 10kΩ Resistor | CRCW060310K0FKEAHP | R1,R2,R3,R22,R23 | 6 | 0603 | 0.074 |
| Panasonic | 1kΩ Resistor | ERJPA3F1001V | R4,R5,R6,R7,R8,  R9,R10,R11,R12 | 9 | 0603 | 0.035 |
| Panasonic | 210Ω Resistor | ERJ-3EKF2100V | R19,R20,R21 | 3 | 0603 | 0.02 |
| Vishay | 510Ω Resistor | CRCW0603510RFKEA | R13,R14,R15,  R16,R17,R18 | 6 | 0603 | 0.041 |
| AVX | 0.1uF Cap | 06035C104KAZ2A | C5,C6,C7,C11,C12,C14 | 6 | 0603 | 0.0805 |
| Wurth Elek | 0.1uF Cap | 865230640001 | C1, C3 | 2 | Radial Can | 0.216 |
| Panasonic | 150uF Cap | EEEFTV151XAP | C4,C2 | 2 | Radial Can | 0.5868 |
| Panasonic | 10uF Cap | EEEFPV100UAR | C8,C9,C10 | 3 | Radial Can | 0.3468 |
| AVX | 4.7uF Cap | TAJA475K020RNJ | C13 | 1 | 1206 | 0.2688 |
| On Semicon | Zener Diode | 1SMB5935BT3G | D1, D2, D3 | 3 | 403A | 0.312 |
|  | | | Total | £48.0156 | | |

1. Bluetooth Command Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Command** | **TX** | **RX** | **Data** | **BLE Read/Write** |
| 250 | APP\_PARAMETER\_SETUP | PSA | PSD | 0 | N/A |
| 1 | SET\_LIVE\_PRESSURE\_READINGS\_CH1 | PSD | PSA | 0 – 1000 (float conversion) | R |
| 2 | SET\_LIVE\_PRESSURE\_READINGS\_CH2 | PSD | PSA | 0 - 1000 | R |
| 3 | SET\_LIVE\_PRESSURE\_READINGS\_CH3 | PSD | PSA | 0 - 1000 | R |
| 11 | SET\_PEAK\_PRESSURE\_CH1 | PSD | PSA | 0 - 1000 | R |
| 12 | SET\_PEAK\_PRESSURE\_CH2 | PSD | PSA | 0 - 1000 | R |
| 13 | SET\_PEAK\_PRESSURE\_CH3 | PSD | PSA | 0 - 1000 | R |
| 21 | SET\_BOT\_PRESSURE\_CH1 | PSD | PSA | 0 - 1000 | R |
| 22 | SET\_BOT\_PRESSURE\_CH2 | PSD | PSA | 0 - 1000 | R |
| 23 | SET\_BOT\_PRESSURE\_CH3 | PSD | PSA | 0 - 1000 | R |
| 91 | EXECUTE\_PEAK\_RESET\_CH1 | PSA | PSD | 0 | W |
| 92 | EXECUTE\_PEAK\_RESET\_CH2 | PSA | PSD | 0 | W |
| 93 | EXECUTE\_PEAK\_RESET\_CH3 | PSA | PSD | 0 | W |
| 101 | EXECUTE\_BOT\_RESET\_CH1 | PSA | PSD | 0 | W |
| 102 | EXECUTE\_BOT\_RESET\_CH2 | PSA | PSD | 0 | W |
| 103 | EXECUTE\_BOT\_RESET\_CH3 | PSA | PSD | 0 | W |
| 9 | EXECUTE\_ZERO\_CLEAR | PSA | PSD | 0 | W |
| 10 | EXECUTE\_SET\_DEFAULT | PSA | PSD | 0 | W |
| 31 | SET\_SWITCH\_MODE\_CH1 | BOTH | BOTH | 0: NO  1: NC | R/W |
| 32 | SET\_SWITCH\_MODE\_CH2 | BOTH | BOTH | 0: NO  1: NC | R/W |
| 33 | SET\_SWITCH\_MODE\_CH3 | BOTH | BOTH | 0: NO  1: NC | R/W |
| 41 | SET\_OP\_MODE\_CH1 | BOTH | BOTH | 0: Hysteresis  1: Comparator | R/W |
| 42 | SET\_OP\_MODE\_CH2 | BOTH | BOTH | 0: Hysteresis  1: Comparator | R/W |
| 43 | SET\_OP\_MODE\_CH3 | BOTH | BOTH | 0: Hysteresis  1: Comparator | R/W |
| 51 | SET\_LED\_STATE \_CH1 | BOTH | BOTH | 0: GREENON\_REDOFF  1: REDON\_GREENOFF  2: Normally RED  3: Normally GREEN | R/W |
| 52 | SET\_LED\_STATE \_CH2 | BOTH | BOTH | 0: GREENON\_REDOFF  1: REDON\_GREENOFF  2: Normally RED  3: Normally GREEN | R/W |
| 53 | SET\_LED\_STATE \_CH3 | BOTH | BOTH | 0: GREENON\_REDOFF  1: REDON\_GREENOFF  2: Normally RED  3: Normally GREEN | R/W |
| 61 | SET\_HI\_SETPOINT\_CH1 | BOTH | BOTH | 0 - 1000 | R/W |
| 62 | SET\_HI\_SETPOINT\_CH2 | BOTH | BOTH | 0 - 1000 | R/W |
| 63 | SET\_HI\_SETPOINT\_CH3 | BOTH | BOTH | 0 - 1000 | R/W |
| 71 | SET\_HYSTERESIS\_CH1 | BOTH | BOTH | 0 - 1000 | R/W |
| 72 | SET\_HYSTERESIS\_CH2 | BOTH | BOTH | 0 - 1000 | R/W |
| 73 | SET\_HYSTERESIS\_CH3 | BOTH | BOTH | 0 - 1000 | R/W |
| 81 | SET\_LO\_SETPOINT\_CH1 | BOTH | BOTH | 0 - 1000 | R/W |
| 82 | SET\_LO\_SETPOINT\_CH2 | BOTH | BOTH | 0 - 1000 | R/W |
| 83 | SET\_LO\_SETPOINT\_CH3 | BOTH | BOTH | 0 - 1000 | R/W |
| 4 | SET\_RESPONSE\_TIME | BOTH | BOTH | 0: 2ms  1: 20ms  2: 50ms  3: 100ms  4: 200ms  5: 500ms | R/W |
| 5 | SET\_USER\_PRESSURE\_UNIT | BOTH | BOTH | 0: MPa  1: KPa  2: Kgf\_cm3  3: Bar  4: PSI  5: mm\_Hg  6: cm\_Hg  7: Inch\_Hg | R/W |
| 6 | SET\_USER\_REFRESH\_RATE | BOTH | BOTH | 0: 200ms  1: 500ms  2: 1000ms | R/W |
| 7 | SET\_USER\_DISPLAY\_RESOLUTION | BOTH | BOTH | 0: 0.00 (2d.p.)  1: 0.000 (3d.p.) | R/W |
| 8 | SET\_ANALOGUE\_OUTPUT\_VOLT | BOTH | BOTH | 0: 1-5V  1: 0-10V | R/W |
| 97 | CH1\_FAULT | PSD | PSA | 0 | R |
| 98 | CH2\_FAULT | PSD | PSA | 0 | R |
| 99 | CH3\_FAULT | PSD | PSA | 0 | R |
| 111 | CH1\_DIGITAL\_OUTPUT\_STATE | PSD | PSA | 0: OFF  1: ON | R |
| 112 | CH2\_DIGITAL\_OUTPUT\_STATE | PSD | PSA | 0: OFF  1: ON | R |
| 113 | CH3\_DIGITAL\_OUTPUT\_STATE | PSD | PSA | 0: OFF  1: ON | R |