Designing a Smartwatch with Fitness Tracking Capabilities

Jake Drew
College of Engineering
Swansea University
Swansea UK
990723@swansea.ac.uk

***Abstract* – This research paper will outline the design and production considerations required at every stage of the manufacturing to produce a smartwatch. This smartwatch is designed with open-source electronic design automation (EDA) to produce a printed circuit board (PCB). The PCB will be programmed using Rust and is then embedded with a TensorFlow lite model. This model is generated using a dataset collected while performing both Running and Rowing. As a result, the smartwatch can detect which activity is being performed by the user.**

Keywords - Smartwatch; Machine Learning; Multi-level Activity Classification; TensorFlow; RP2040; KICAD; Embedded Rust;

I. Introduction

The aim of this project is to produce a smartwatch which can detect which activity the user is engaged in. Currently there is a vast array of choices for fitness trackers and smartwatches and while many different companies offer a variety of options and customisations, these options all come with close-source code or partially open-sourced with core components close-sourced, none of them come with open-sourced hardware design.

As such there are limited open-source solutions, therefore this project will be but fully expandable and programmable, this means if you want to add a Bluetooth functionality, it will be possible to use custom software to achieve that.

As part of the smartwatch software, will be the addition of a fitness activity classification machine learning model, which will be able to distinguish between which activity the smartwatch user is engaged in. This model will be tested to determine whether it can achieve 80%+ accuracy.

II. Requirements

In deciding the requirements, for the smartwatch's software, it should be able to distinguish between two different activities. The choice for these two is Running and Rowing as there is a substantial difference in the way the wrist moves relative to the body and as such it should be easier to distinguish between the two. These two activities are chosen due to the significant difference in ranges of motion between these activities allowing the TensorFlow model to distinguish between these activities more easily.

To determine the activity an accelerometer is necessary to monitor the movement of the smartwatch on the user. The smartwatch also requires the exposure of certain functionalities to enable future expansion.

Any wearable device which cannot last a full day becomes impractical to use due to charging throughout the day which can be tedious. As such a 2-day battery life average is an appropriate target, this will allow the smartwatch to have 2 days battery life in typical use but also to guarantee a minimum of 1 day battery life if the smartwatch is used extensively. This 2-day battery life is important for convenience, to compliment this convenience the ability to charge the device with standard USB (either micro or mini) and to program the microcontroller with USB.

III. Component Choices

When deciding on a microcontroller, there are a variety of considerations to think about. Such as what methods are available to program the device, what languages does the device support, can the microcontroller fulfil the requirements and is it overkill.

A common choice for hobbyists, is the ATMega328 which is more commonly known as the Arduino Uno [1]. This microcontroller has a large community supporting various libraries to simplify the development process. The ATMega328 is best programmed using the Arduino IDE which uses C++. This microcontroller costs £2.24 [2].

Another choice is the STM32 collection of microcontrollers is an ARM Cortex-M0+, these are typically used for more advanced hobbyists and industrial applications, this is programmed using C and is powerful enough to run TensorFlow Lite models on. This microcontroller costs £4.39 [3].

The final choice is the RP2040 which was only announced at the start of 2021, this microcontroller has a smaller library support than the rest, but it can by default be programmed in Python as well as C++ or Rust. The RP2040 is like the STM32 in that they are both ARM Cortex-M0+ and can run TensorFlow Lite models on. This microcontroller costs £0.52 [4].

Based on all these factors it, the RP2040 is the best option as it offers better or on-par capabilities at a lower cost. The Raspberry Pi Foundations who developed the RP2040, also have provided detailed document titles "Hardware Design with RP2040" [5] this document outlines the 'Minimal Design Example' which outlines the core components required to get a RP2040 working and the certain circuit design optimisations. This gives the project a boilerplate to append desired sub-processes onto.

In the 'Minimal Design Example' the basic components are, the RP2040 microcontroller, RP2040 decoupling capacitors, USB series termination resistors and 12MHz external oscillator. The "Hardware Design with RP2040" states the requirement for one external chip the RP2040 requires is a flash memory chip – specifically the W25Q128JVS, which is a 128Mbit (16Mbyte) quad SPI flash memory. This is where the program code is stored and where the RP2040 boots from. The W25Q128JVS is chosen due to the fact it is recommended by the "Hardware Design with RP2040" and these two components have great synergies.

For the accelerometer, a chip which uses Inter-Integrated Circuit (I^2C) for communication is required as this is a very common addressable serial communication bus, and the RP2040 has multiple I^2C connections. The most used accelerometer is the MPU6000 series, however recently these have become out-of-stock with Farnell not being restocked till November 2022. This delay is substantially out of the timeframe for the project [6]. As such a replacement chip the LSM6DS series, does everything the MPU6000 series does with the one differential being the smaller library support. But this is an issue which can be resolved with software.

The OLED module will be the 1.3 inch, SSH1106 OLED module as shown below in Figure 1. This uses I^2C and has a large library support in many programming languages.

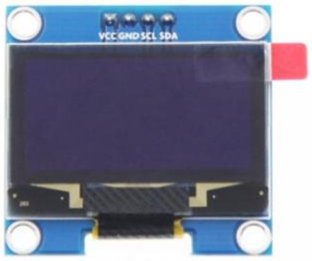

Figure 1: 1.3 inch I^2C OLED Module [7]

The lithium-polymer battery will be chosen as the largest available battery that is the same width and length as the OLED. The OLED has a width and length of 33.5mm by 35.4mm [7]. Figure 2 shows a 700mAh lithium-polymer battery [8] which has a 30mm by 35mm.

Figure 2: 700mAh Lithium-Polymer Battery [8]

Other smaller components such as resistors, capacitors and crystals will be selected depending on what each of the previously mention datasheets recommends.

IV. Power Supply

A. USB

When the device is connected via USB, to charge or to program. The 5V USB power supply is stepped down with the 3.3V regulator and is distributed throughout the circuit. The 5V is also used to charge the battery.

B. Battery

When no USB 5V is provided, the smartwatch will switch the battery. This 700mAh battery will slowly deplete over time as the smartwatch is used. To estimate this time,

Table 1: Components Power Consumption

Component	State	Consumption (mAH)
OLED	SSH1106 Controller	1.6
OLED	All LED's ON	24.3
OLED	25% ON	6.07
RP2040	Popcorn	24.8
RP2040	Sleep	4.5
DS1307	Active	1.5
DS1307	Standby	0.2
LSM6DS3	High	0.9
LSM6DS3	Low	0.29

To predict the expected consumption, we take the OLED's controllers power consumption and 25% of the LED being on. And then take the active states consumption for all the chips for only 1 hr a day and then the sleep state consumption for the rest of the day. The always on consumption takes the highest possible consumption for each component, RP2040 Popcorn mode is listed in the datasheet [9] as the high end of possible power consumption as the Popcorn demo "uses VGA video, I2S audio and 4-bit SD Card access, with a system clock frequency of 48MHz". The sleep state takes the lowest possible consumption of each chip. The results of these calculations are shown below in Table 2.

Table 2: Different States Consumption

State	Consumption (mAH)
Always On	34.875
Expected	10.833
Sleep	6.590

Taking the 700mAh battery and dividing it by the consumption is one way of estimating the lifetime of the device per charge cycle. However, there is a better way to do this, there is a Python library which enables the modelling of batteries – "Python Battery Mathematical Modelling (PyBaMM)" [10]. Using the Doyle-Fuller-Newman (DFN) model and simulating each consumption rate over a 5-day period, we can produce Figure 3.

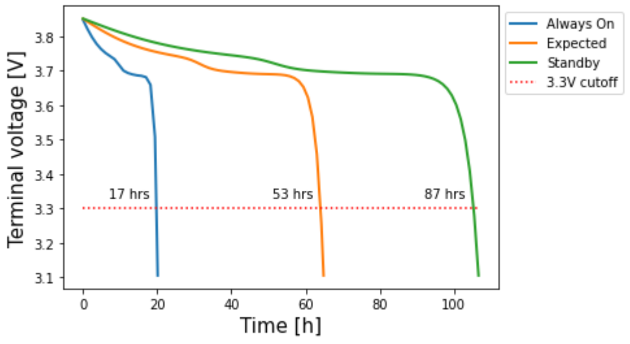

Figure 3: DFN Battery Voltage Simulation

The Figure 3 has a 3.3V cut-off, this is where the voltage regulator can no longer consistently output 3.3 volts to the circuit as due to the fact most components operating voltages require 3.3 volts. The device is thus considered 'out of charge'. The results of this simulation are that even at the maximum power consumption the device lasts 17hrs, which assuming 8hrs sleep is enough to last a day. However, the expected battery life is 57hrs which is more than enough to meet the requirement of a full day battery life.

V. Circuit Design

A. Power Management

For the power management sub-system of this design, a simple problem emerges, we require the ability to charge the battery as well as power the device simultaneously. This is because as stated in "Designing A Li-Ion Battery Charger and Load Sharing System with Microchip's Stand-Alone Li-Ion Battery Charge Management Controller" (AN1149) [11] - "It is not encouraged to attach the system load directly to Li-Ion batteries when using a stand-alone Li-Ion battery charge management controller". Therefore, we need some system to switch between USB 5V (VCC) and Battery (VDD).

When selecting the MOSFET "AN1149" [11] makes clear that for the MOSFET - "It is important to select a proper gate threshold voltage range so the MOSFET will be turned on" – the

LBSS84LT1G meets this range. When selecting the diode “AN1149” states that “*The Average Forward Current has to be rated greater than the maximum system load current for the application*”. B5819W is a great candidate as in the datasheet reference [3] it states the Repetitive Peak Forward Current is 1.5A and the Average Rectified Output Current is 1A, these two currents are more than greater than the RP2040. The Raspberry Pi Pico which uses the RP2040 and contains many common components has a max current draw of 92.8mA [12] - this current draw was recorded while running the popcorn demo the power consumption while running the smartwatch will be substantially less.

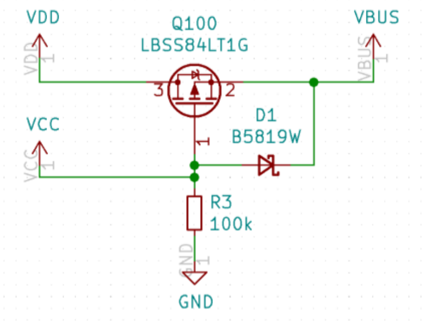

Figure 4: P-Channel MOSFET Load Sharing Bypass Switch

Figure 4 shows the finalised circuit for the load sharing, where VDD is the positive terminal of the lithium-polymer battery and VCC is the 5V USB pin.

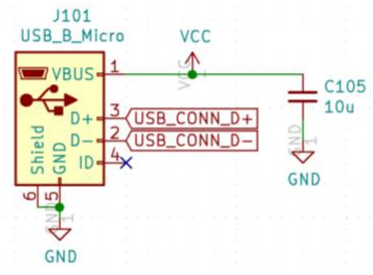

Figure 5: USB Micro Connector

Figure 5 shows the VCC being connected to the VBUS of the USB which is the 5V power line. The USB data differential pairs is then connected to the RP2040 as shown in Figure 6.

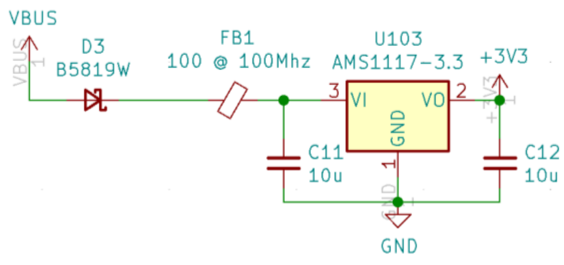

Figure 6: Linear Regulator

The power from the P-Channel MOSFET load sharing bypass switch must now be regulated to 3.3V for the microcontroller and other components. Using a linear regulator is the most reliable and simple method whereas using voltage dividers cannot work, due to difference in 5V USB and the lithium-polymer battery which will range from 4.2V to 2.7V depending on charge. Step down transformers are impractical just like voltage dividers as they both are not variable. When converting a consistent voltage down they would work but when the voltage on the supply side is voltatile it wouldn’t, therefore an active component to regulate, will result in a more stable supply voltage.

B. *Microcontroller*

As mentioned earlier, the “Hardware Design for RP2040” outlines common design choices for the RP2040 – as such reproducing those schematics is preferrable. Figure 7 shows the recommended RP2040 schematic and the required decoupling capacitors, this design is like for like as the datasheet states. The I^{2}C protocol requires two 4.7KΩ pull-up resistors on each line.

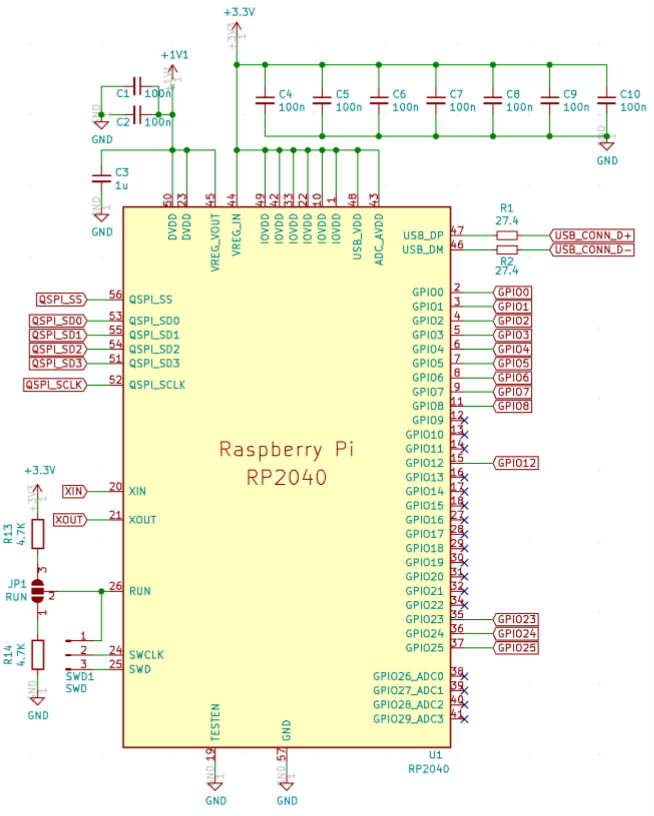

Figure 7: RP2040 Schematic

C. *Other Components*

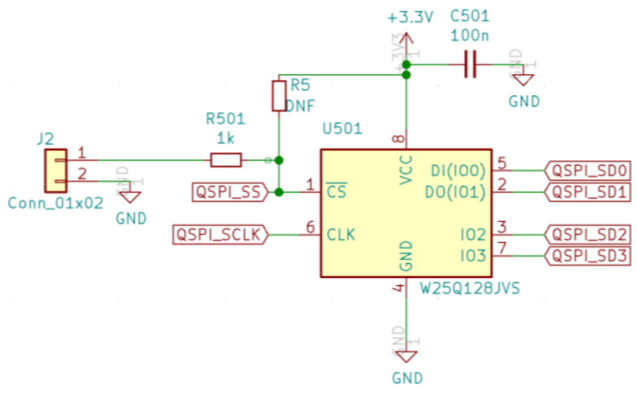

Figure 8: Flash Storage

As mentioned in the 'Component Choices' section the W25Q128JVS flash storage is the recommended chip and as such it was chosen, the "Hardware design with RP2040" shows the recommended circuit diagram as such for this smartwatch Figure 8 uses the recommended diagram.

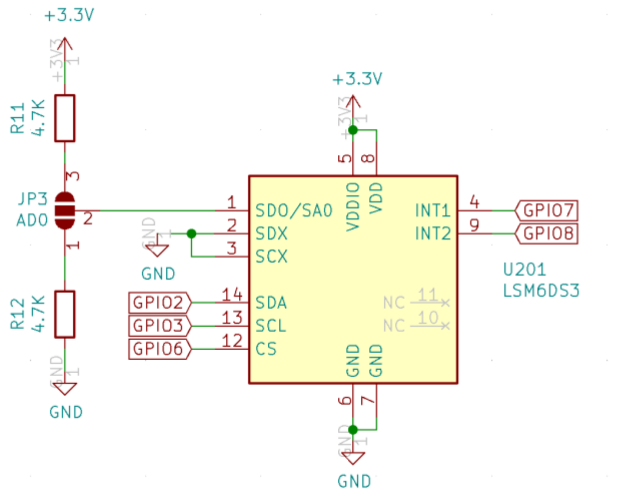

Figure 9: LSM6DS Accelerometer

The LSM6DS3 datasheet [13] in the section 'Application Hints' sets out the recommended layout for the chip and this is shown in Figure 9. However, the LSM6DS3 has two potential I^{2}C addresses 0x6A or 0x6B, which address the device uses depends on if pin 1 or SD0 is pulled low for 0x6A or pulled high for 0x6B. The jumper (JP3) can be soldered to pull it high or low.

For the real time clock which is a background chip with the sole purpose of keeping track of the time. The DS1307 is the most common RTC in the hobbyist community and has a large library support. In the DS1307 datasheet [14] the 'typical operating circuit' is the recommended circuit diagram and Figure 10 shows how this recommended circuit has been adjusted to fit this circuit.

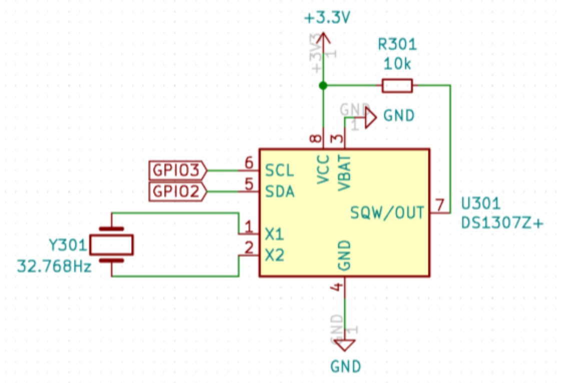

Figure 10: DS1307 Real Time Clock

VI. PCB DESIGN

When designing the PCB, the first consideration is size, because the OLED module and the lithium-polymer battery are the same width and length, the PCB should also be that size. Each chip will constitute a subsystem and all associated components should be placed as close to the subsystems chip as possible, this is to reduce noise as well as improve signal integrity.

Another way to reduce noise and improve signal integrity is the PCB stack up [15]. The PCB stack up is the purpose of each layer within the board. For this smartwatch the bottom and top layers will be for signal and the middle two layers will be ground and power planes. The Table 3 shows how this will work.

Table 3: PCB Stack Up

Layer	Use
Top	Signal
Inner 1	Ground Plane
Inner 2	3.3V Plane
Bottom	Signal

The 5V from USB and the RP2040's 1.1V won't be on the power plane as the wiring for these can be kept small and there is no need for these signals to travel across the board.

The top layer of the board will also be where the all assembled components will be placed, JLCPCB only allows for one side of the board to be assembled – as such the bottom layer will contain the 32.768kHz crystal which has to be hand soldered and any of the jumpers or test points.

Once all the components have been placed accordingly on the KICAD design, the design can be exported as a Gerber file according to JLCPCB specifications [16]. The Gerber files are the instructions for the PCB production process. Figures 11, 12, 13 and 14 show how each layer of the stack up looks.

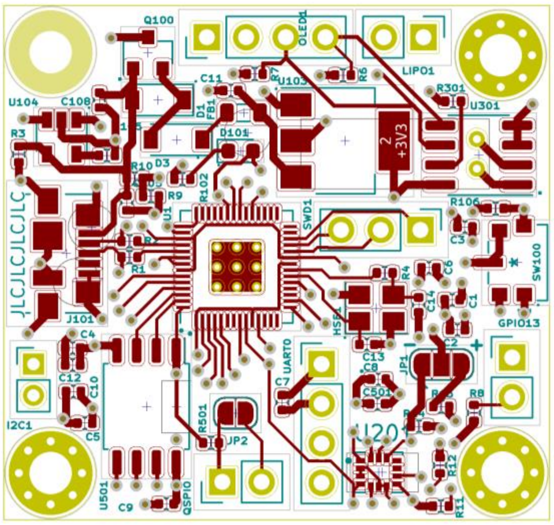
Figure 11: Top Signal Layer of the PCB

Figure 11 is the top layer of the PCB, and you can see the 4 mounting holes in each corner which are aligned with the mounting holes in the OLED as shown in Figure 1. Figure 11 also has many through holes to allow for the OLED and the battery to be connected, these are shown at the top of the board. The rest of the through holes are for various programming functions.

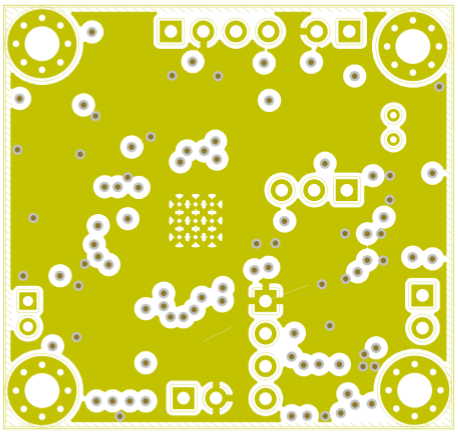
Figure 12: GND Power Plane Layer of the PCB

Figure 12 shows the ground plane and how each ground pin connected to one solid layer of copper which in turn reduces noise. Figure 13 shows the similar 3.3V power plane.

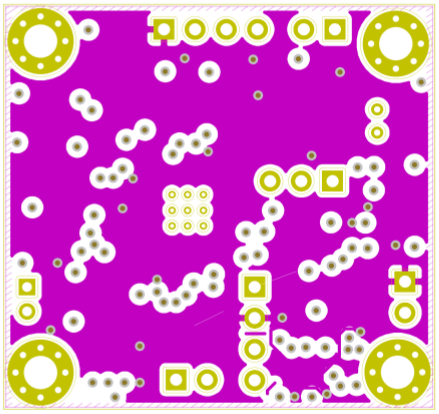
Figure 13: 3.3V Power Plane Layer of the PCB

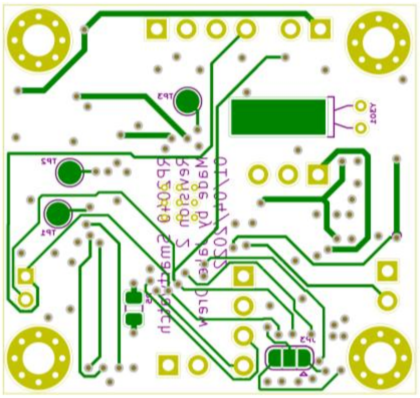
Figure 14: Bottom Signal Layer of the PCB

Finally Figure 14 shows the bottom signal layer and this layer contains all the components which are soldered on by hand as well as all test points and jumpers.

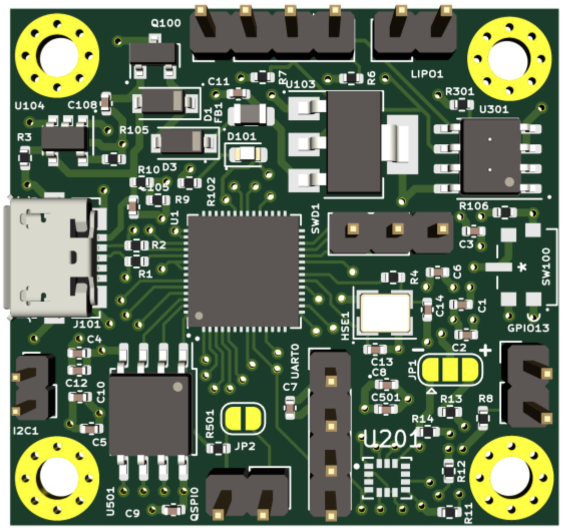
Figure 15: KICAD 3D Render of the PCB

Figure 15 is the 3D render of the PCB in KICAD, this is what the final board should look like.

After the Gerber files, the production process also requires the bill of materials (BOM) and footprint placements. These files are then uploaded to JLCPCB who offer PCB manufacture as well as part placement. All components except any through-hole components are placed by them. This substantially decreases the size of the board, if parts were needed to be soldered manually then the LSM6DS3 and the RP2040 of which are in a Land Grid Array (LGA-14L) [13] and Quad Flat No-lead (QFN) [9] package respectively. Both packages don't have any pins exposed and as such are incredibly difficult to hand solder.

The same goes for the resistors and capacitors which are all 0402 package which means the length is 0.4mm and width is 0.2mm. While still possible to hand solder the accuracy would be substantially reduced.

VII. PCB Result

Once the PCB have been sent off to JLCPCB it should take around 3 to 4 weeks for them to be printed, assembled, and shipped back to UK. This is up from around 1 to 2 weeks prior to Covid-19 and the subsequent chip shortage.

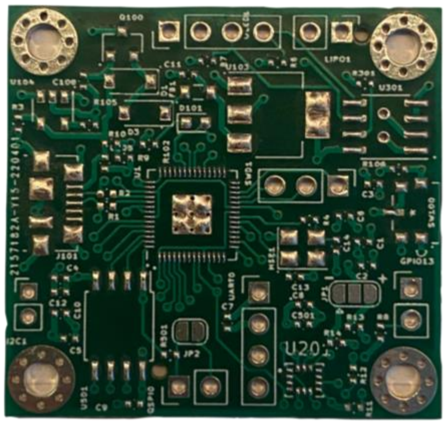

Figure 16: Finished PCB

Figure 16 shows how the finished PCB design without any components.

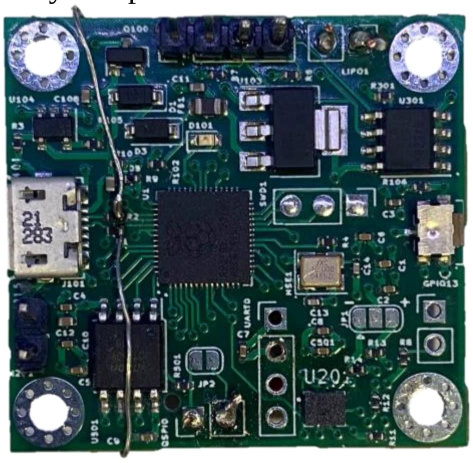

Figure 17: Assembled PCB

Figure 17 shows how the finished assembled PCB. This can now be connected to a computer via micro-USB and programmed.

VIII. ML Model Design

The machine learning aspect of this project will use TensorFlow, which is an open-source platform for machine learning (ML). As a platform TensorFlow models can be trained on one environment and deployed on another. For example, in this instance the model will be trained in Python and deployed on the smartwatch in Rust.

To deploy a TensorFlow model to a microcontroller it must be converted into a TensorFlow Lite model. TensorFlow Lite is designed to be deployed on mobile devices or ARM microcontrollers [17]. The process of running a TensorFlow Lite model on the smartwatch starts with the design of a ML model. This model then must be trained on a dataset, this will update the model and once trained it can be converted. The converted model can be embedded on the smartwatch and the device can run the model. The TensorFlow Lite core runtime only requires 16KB of space [18] on the microcontroller. As this smartwatch has a flash storage of 16MB, the TensorFlow Lite runtime only requires 0.1% of the available storage space.

A. *The Design*

For the activity recognition ML model, in the "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition" [19], the results show that using their own "DeepConvLSTM" model you can achieve a F-Score of 0.958. F-Score is a statistical way of determining accuracy of an algorithm. It divides the results into three categories: true positive, false positive and false negative. The true positive is instances where the algorithm was correct, and the false values are where the algorithm failed. A F-Score of 1.0 is the highest and 0.0 is the lowest. As a result, the "DeepConvLSTM" model "is suitable for multimodal wearable sensors" which is perfect for this smartwatch scenario.

Table 4: TensorFlow “DeepConvLSTM” Model Structure [19]

Input	Output	Name	Type
32,3	32,3	CONV1D_1_INPUT	INPUTLAYER
32,3	28,32	CONV1D_1	CONV1D
28,32	24,32	CONV1D_2	CONV1D
24,32	20,32	CONV1D_3	CONV1D
20,32	16,32	CONV1D_4	CONV1D
16,32	16,128	LSTM_1	LSTM
16,128	16,128	LSTM_2	LSTM
16,128	2048,1	FLATTEN	FLATTEN
2048,1	2048,1	DROPOUT	DROPOUT
2048,1	2,1	OUTPUT	DENSE
2,1	2,1	ACTIVATION_3	ACTIVATION

The architecture of the “DeepConvLSTM” model is shown in Table 4. The “Conv1D” are a layer of a 1-dimensional convolutional kernel, these “layers process the input only along the axis representing time” [19]. “LSTM” layers stand for long-short term memory these are recurrent layers that have feedbacks to remember previous input states. This is useful for activity recognition as the model needs to understand the movement of the watch, not just its present position.

The “Flatten” layer flattens the input from a 16 by 128 to a 2048 by 1 [20] and the “Dropout” layer adds noise [21] to train the model to recognises trends and prevent overfitting – which as described in “TinyML” by Pete Warden and Daniel Situnayake – “When a model is overfit, it has learned it’s training data too well” [22]. This means the model has effectively learnt the mark scheme and is unable to generalise the situations. To solve this, adding some more dropout to the model or decreasing the training iterations (epochs) would help resolve this.

Finally, the output goes through an activation layer [23] which applies an activation function to scale the output between 0 and 1.

B. *Training the Model*

To train this model, I used a dataset from the University of California, Irvine – “UCI Machine Learning Repository: Activity Recognition from Single Chest-Mounted Accelerometer Data Set” [24]. This dataset contains 7 different categories of activities, these activities can be condensed into two types, walking, and standing. Testing the model on this dataset while not perfect for the result, due to sheer size of the dataset, it represents a better test situation – 1.8 million rows

Code 1: Creating a Dataset from CSV in Python

```
column_names = ['X', 'Y', 'Z', 'Activity', 'Person']
df = pd.read_csv('data/2-acts.indexed.1-10.csv',
         header=1, names=column_names)

df_train = df[df['Person'] <= 8]

X_train, y_train = create_dataset(
   df_train[['X', 'Y', 'Z']],
   df_train.Activity,
   64,
   16
)
```

Code 1 shows how the UCI dataset can be converted to a TensorFlow compatible dataset. Firstly, the CSV is read and the “df[df['Person'] <= 8]” defines the training data as the first 8 People out of the 10. This is to divide the UCI dataset into 80% training and 20% test. The dataset is created by defining the input which is “df_train[['X', 'Y', 'Z']]” and then defining the category as “df_train.Activity” – this tells TensorFlow that the activity is the value you are trying to predict. Once this dataset is created the Table 4 “DeepConvLSTM”

Code 2: TensorFlow Python Code to Train Model

```
history = model.fit(
   x_train, y_train,
   epochs=10,
   batch_size=52 * 10,
   validation_data=(x_test, y_test),
   shuffle=True,

)
…
test_pred = np.argmax(model.predict(x_test), axis=1)
test_true = np.argmax(y_test, axis=1)
f = metrics.f1_score(test_true, test_pred, average='weighted')
```

Code 2 uses the “model.fit” function and defining the training data and setting a few parameters such as the validation data which is used to calculate loss [25] and the batch size which is the “number of samples” per epoch [26]. Once the model is trained the “metrics.f1_score” can be calculated. The results are show in Figure 18 and 19.

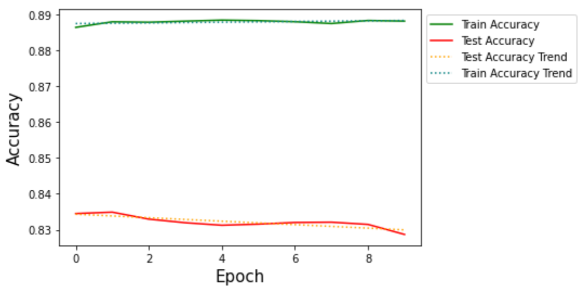

Figure 18: Training and Test Accuracy and Trends

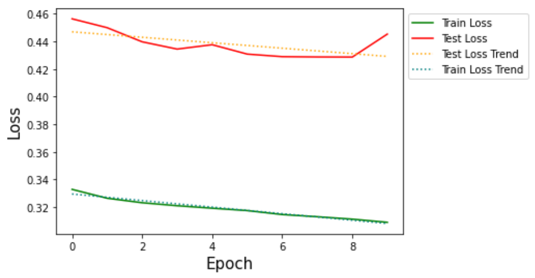

Figure 19: Training and Test Loss and Trends

The results of this model show that as the epoch increases the loss decreases, which shows the improvement of the model, even though the overall test accuracy gets marginally worse, the loss gets better. This means the model is producing less false positives.

Table 5: DeepConvLSTM UCI Dataset Model Results

Result	Value
Train Accuracy	0.888
Test Accuracy	0.832
Train Loss	0.319
Test Loss	0.438

A test accuracy of 0.832 or 83.2% is a good result and it meets the requirements. The loss in this example is the squared loss (L_2 Loss), this is the difference between the observation and the prediction squared.

C. Deploying the Model

Code 3: TensorFlow Python Code to Train Model [27]

```
model.save('models/activity_detection')
converter = tf.lite.TFLiteConverter.from_saved_model(
        'models/activity_detection')
tflite_model = converter.convert()
with open('activity_detection.tflite', 'wb') as f:
  f.write(tflite_model)
```

The model needs to be converted into a TensorFlow Lite model so that it can be embedded on the microcontroller. Code 3 shows how it can be done in Python, TensorFlow recommends [27] to save the model as a standard TensorFlow model first and then convert to TensorFlow Lite from that saved model.

Code 4: Running the TensorFlow model in Rust [28]

```
let model_array = include_bytes!(
        "../models/smartwatch.tflite");
let model = Model::from_buffer(&model_array[..]).unwrap();

const TENSOR_ARENA_SIZE: usize = 4 * 1024;
let mut arena: [u8; TENSOR_ARENA_SIZE] =
         [0; TENSOR_ARENA_SIZE];

let op_resolver = AllOpResolver::new();
let mut interpreter = MicroInterpreter::new(&model,
        op_resolver, &mut arena[..]).unwrap();
interpreter.input(0, &[0.0]).unwrap();
interpreter.invoke().unwrap();
let output = interpreter.output(0).as_data::<f32>();
```

In Code 4 the TensorFlow Lite model is first loaded as bytes. Next the Tensor arena is allocated in memory [29], which reserves this memory for the model when it runs. Then the "MicroInterpreter" is instantiated, and the interpreter is given the input, in the example Code 4 the input is set to 0.0. The interpreter is then invoked which runs the model on the input and the output of the model is returned via "interpreter.output(0)".

IX. Programming

The smartwatch design utilises the RP2040's ability to "boot as a USB mass storage device" [9] this means the device will appear on the computer just as any typical storage device would. To achieve this the smartwatch, needs to be in BOOTSEL mode which is achieved by connecting jumper 2 at the bottom of the board. Once connected as a "USB mass storage device".

To program in Rust, the use of a hardware abstraction layer , for the embedded RP2040 Rust ecosystem the GitHub library rp-hal [30] provides the bare bones to build the software on top of with features such as a file called "build.rs" which is takes the compiled Rust and writes it onto the "USB mass storage device".

X. Results

The result after assembling the smartwatch PCB with all components is show in Figure 20.

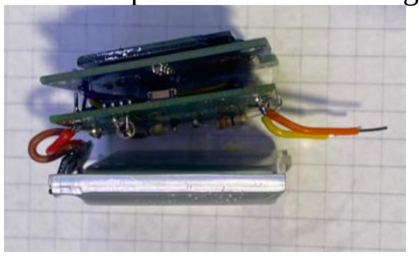
Figure 20: Side view of Assembled Board

Figure 20 shows the OLED and lithium-polymer battery on the top and bottom respectively, as well as the BOOTSEL enable wires (orange and yellow) which when connected boot the device as "USB mass storage device".

The overall size of the device is 35mm width, 33mm length and 19mm depth when the battery is pushed as close to PCB as possible. Figure 20 shows the battery with a sizeable gap between it and the PCB if no pressure applied. With the addition of some software, the board can run Rust, Python or C++. The final board can look like Figure 21 shown below, with ML models predictions of which activity the user is engaged in.

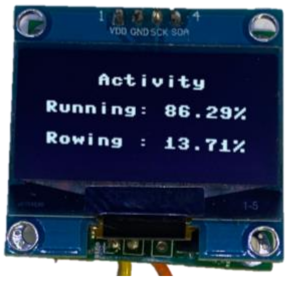

Figure 21: Front view of Assembled Board with Software

XI. Reflections

A few mishaps occurred during this process, the main one being the error in ordering the assembled JLCPCB where the 27.4Ω shown in Figure 7, where instead ordered with 27.4KΩ. This error was fixable due to the fact, the intention of the board being expandable meant USB test pads were placed on the bottom of the board therefore enabling a resistor to be placed across to fix this. Figure 20 shows those two resistors on the underneath of the board to amend this.

Another possible modification is the addition of a single pole double throw (SPDT) switch which is a switch where there are two possible paths. This could be used with the enabling of BOOTSEL mode which requires either the soldering of a jumper or connecting the two wires shown in Figure 20.

XII. References

[1] Arduino - ArduinoBoardUno [Internet]. Arduino.cc. 2022 [cited 6 February 2022]. Available from: https://www.arduino.cc/en/Main/arduinoBoardUno

[2] [Internet]. ATMega328. 2022 [cited 13 April 2022]. Available from: https://uk.farnell.com/microchip/atmega328-au/mcu-8bit-atmega-20mhz-tqfp-32/dp/1972086

[3] [Internet]. STM32L051K8T6. 2022 [cited 13 April 2022]. Available from: https://uk.farnell.com/stmicroelectronics/stm32l051k8t6/mcu-32bit-cortex-m0-32mhz-lqfp/dp/2444626

[4] [Internet]. RP2040TR13. 2022 [cited 13 April 2022]. Available from: https://uk.farnell.com/raspberry-pi/rp2040tr13/mcu-32bit-133mhz-qfn-56/dp/3766080?st=rp2040

[5] Hardware Design with RP2040 [Internet]. Raspberry Pi (Trading) Ltd; 2022 [cited 8 February 2022]. Available from: https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf

[6] [Internet]. MPU-6050. 2022 [cited 13 April 2022]. Available from: https://uk.farnell.com/invensense/mpu-6050/gyro-accel-6-axis-i2c-qfn-24/dp/1864742?st=MPU-6050

[7] 1.3inch IIC OLED Module SKU:MC130VX - LCD wiki [Internet]. Lcdwiki.com. 2022 [cited 15 April 2022]. Available from: http://www.lcdwiki.com/1.3inch_IIC_OLED_Module_SKU:MC130VX

[8] 603035 700mAh 3.7v Li-Polymer battery [Internet]. Hubats. 2022 [cited 11 April 2022]. Available from: https://hubats.com/product/603035-700mah-3-7v-li-polymer-battery-for-dvr-gps-mp3-mp4-vertical-computer-mouse-mobius-camera-video-recorder-bluetooth-column/

[9] RP2040 Datasheet [Internet]. Raspberry Pi (Trading) Ltd; 2020 [cited 6 February 2022]. Available from: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

[10] Sulzer V, Marquis S, Timms R, Robinson M, Chapman S. Python Battery Mathematical Modelling (PyBaMM). Journal of Open Research Software. 2021;9(1):14.

[11] Chu B. Designing A Li-Ion Battery Charger and Load Sharing System With Microchip's Stand-Alone Li-Ion Battery Charge Management Controller [Internet]. Microchip; 2008 [cited 19 March 2022]. Available from: https://ww1.microchip.com/downloads/en/AppNotes/01149c.pdf

[12] Raspberry Pi Pico Datasheet [Internet]. Raspberry Pi (Trading) Ltd; 2020 [cited 7 March 2022]. Available from: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

[13] LSM6DS3TR-C [Internet]. 3rd ed. 2017 [cited 12 March 2022]. Available from: https://www.st.com/resource/en/datasheet/lsm6ds3tr-c.pdf

[14] 14. DS1307 [Internet]. 3rd ed. Maxim; 2015 [cited 2 May 2022]. Available from: https://datasheets.maximintegrated.com/en/ds/DS1307.pdf

[15] How to Reduce Noise in a PCB Board [Internet]. Pcbway 2022 [cited 10 February 2022]. Available from: https://www.pcbway.com/blog/Engineering_Technical/How_to_Reduce_Noise_in_a_PCB_Board.html

[16] How to generate the Gerber files? [Internet] JLCPCB. [cited 29 March 2022]. Available from: https://support.jlcpcb.com/article/22-how-to-generate-the-gerber-files

[17] Build TensorFlow Lite for ARM boards [Internet]. TensorFlow. [cited 4 February 2022]. Available from: https://www.tensorflow.org/lite/guide/build_arm

[18] TensorFlow Lite for Microcontrollers [Internet]. TensorFlow. 2021 [cited 9 February 2022]. Available from: https://www.tensorflow.org/lite/microcontrollers

[19] Ordóñez F, Roggen D. Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. Sensors. 2016;16(1):115.

[20] Team K. Keras documentation: Flatten layer [Internet]. Keras.io. [cited 29 April 2022]. Available from: https://keras.io/api/layers/reshaping_layers/flatten/

[21] Team K. Keras documentation: Dropout layer [Internet]. Keras.io. [cited 29 April 2022]. Available from: https://keras.io/api/layers/regularization_layers/dropout/

[22] Situnayake D, Warden P. TinyML. [Place of publication not identified]: O'Reilly Media, Inc.; 2019.

[23] Team K. Keras documentation: Activation layer [Internet]. Keras.io. [cited 29 April 2022]. Available from: https://keras.io/api/layers/core_layers/activation/

[24] UCI Machine Learning Repository: Activity Recognition from Single Chest-Mounted Accelerometer Data Set [Internet]. Archive.ics.uci.edu. 2014 [cited 4 February 2022]. Available from: https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer

[25] Train a Keras model — fit [Internet]. Keras.rstudio.com. [cited 7 April 2022]. Available from: https://keras.rstudio.com/reference/fit.html

[26] Customize what happens in Model.fit | TensorFlow Core [Internet]. TensorFlow. 2022 [cited 9 April 2022]. Available from: https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit

[27] TensorFlow Lite converter [Internet]. TensorFlow. [cited 21 April 2022]. Available from: https://www.tensorflow.org/lite/convert/

[28] Kevin Hill G, Meadows R, Rosenthal J. GitHub - Recognition2/tfmicro [Internet]. GitHub. 2020 [cited 23 April 2022]. Available from: https://github.com/Recognition2/tfmicro

[29] Get started with microcontrollers | TensorFlow Lite [Internet]. TensorFlow. 2022 [cited 21 April 2022]. Available from: https://www.tensorflow.org/lite/microcontrollers/get_started_low_level

[30] GitHub - rp-rs/rp-hal: A Rust Embedded-HAL for the rp series microcontrollers [Internet]. GitHub. 2022 [cited 19 April 2022]. Available from: https://github.com/rp-rs/rp-hal