CG4002 Embedded System Design Project
August 2021 semester

“Dance Dance”
Design Report

Group **16**	Name	Student #	Specific Contribution
Member #1	Xie Yaoyue	A0205294H	Hardware Sensors
Member #2	Marcus Ng Yong	A0201743R	Hardware FPGA
Member #3	Koh Rui Wen Tammy	A0202616R	Internal Communications
Member #4	Manuntag Manuel II Papa	A0202130J	External Communications
Member #5	Wang Zihao	A0204706M	Software Machine Learning

For our group, since we only have 5 members, no one is assigned to do the dashboard design. For the design report, Section 1 System Functionalities, Section 2 Overall System Architecture, Section 6 Project Management Plan and Section 7 Societal and Ethical Impact are written by all members together. Section 3.1 Hardware Sensors is written by Yaoyue. Section 3.2 Hardware FPGA is written by Marcus. Section 4.1 Internal Communications is written by Tammy. Section 4.2 External Communications is written by Manuel. Section 5.1 Software Machine Learning and Appendix are written by Zihao.

Section 1 System Functionalities [All members]

The ultimate design of the system is to classify 9 different dance moves and identify the relative positions of 3 different dancers. The details of the system functionalities are illustrated below in the format of feature lists and user stories.

Feature Lists

- The system will be able to differentiate the different dance moves.
- The system will be able to evaluate the level of synchronization.
- The system will be able to identify the relative positions of the dancers.
- The system is user-friendly and easily operable.

User Stories

As a:	**I want:**	**So that:**
Dance Coordinator	To know how synchronized the dancers are	I can instruct the dancers on how to improve their timing
Dance Coordinator	To receive data on the synchronization of dancers as quickly as possibly	I can give prompt feedback to the dancers
Dancer Coordinator	To know the position of the dancers	I can give feedback to the correct dancers
Dancer Coordinator	To know the dance movement that is performed by the dancer	I can give feedback to the dancers regarding the specific dance movement
Dancer	The positions of sensors to be not in the way	I can dance without being restricted
Dancer	The sensors to be attached are	I don’t need to worry about

	secure	sensors falling off
Dancer	To know if I'm doing the move right	I can be corrected if I'm wrong

Section 2 Overall System Architecture [All members]

2.1 High-level system architecture

The overall system architecture and high-level system architecture are shown in Figures 1 and 2 respectively.

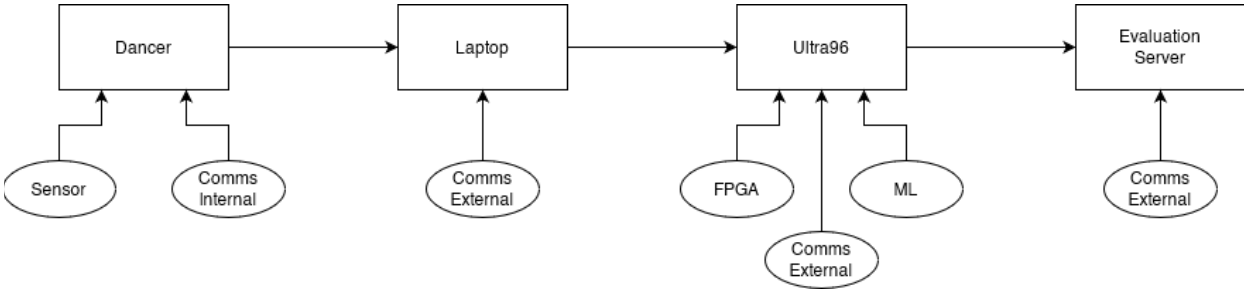


Figure 1. Overall system architecture

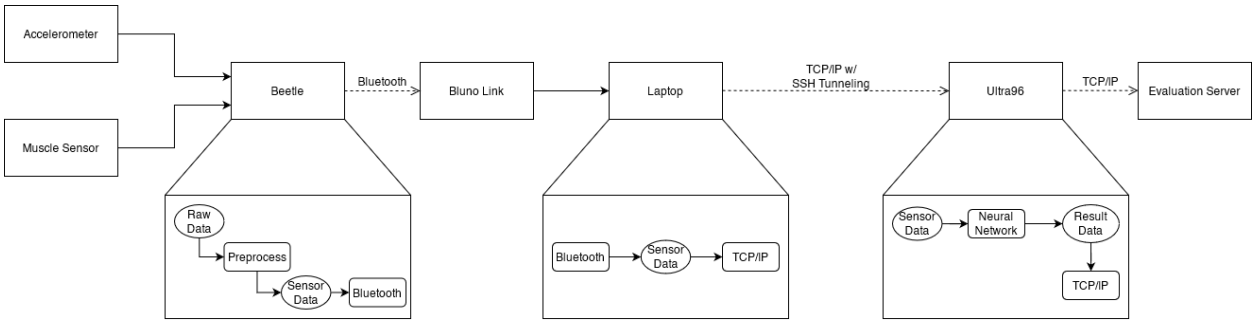


Figure 2. High-level system architecture

The beetle will take in motion data from the accelerometer and gyroscope and preprocess the motion data. The beetle will establish a Bluetooth connection with the laptop and send the motion data in packets following the BLE Protocol Stack to the laptops (Bluno link) along with timestamps (for synchronization).

The laptop will establish a TCP/IP connection with the Ultra96 with SSH Tunneling. The laptop will add delay information to the timestamps (for NTP), encrypt and encode the motion data and send them all to the Ultra96.

The Ultra96 will establish a TCP/IP connection with the evaluation server. The Ultra96 will run the Neural Network to evaluate the motion data and predict the dance move. The Ultra96 will calculate the synchronization delay from the information provided by the laptop and the beetle. The dance move will be encrypted and encoded along with the synchronization delay and then sent to the evaluation server where it will be evaluated.

2.2 Final form of the system

Since all dance movements are symmetrical on both the left and right sides of the body, we can detect the dance movements by capturing the data from the right side of the body. Hence, for each dancer, a hardware set containing 1 3.7V battery + 1 voltage regulator + 1 DFR0339 + 1 MPU6050 is to be attached to the upper right arm of the dancer through an armband (Figure 3).

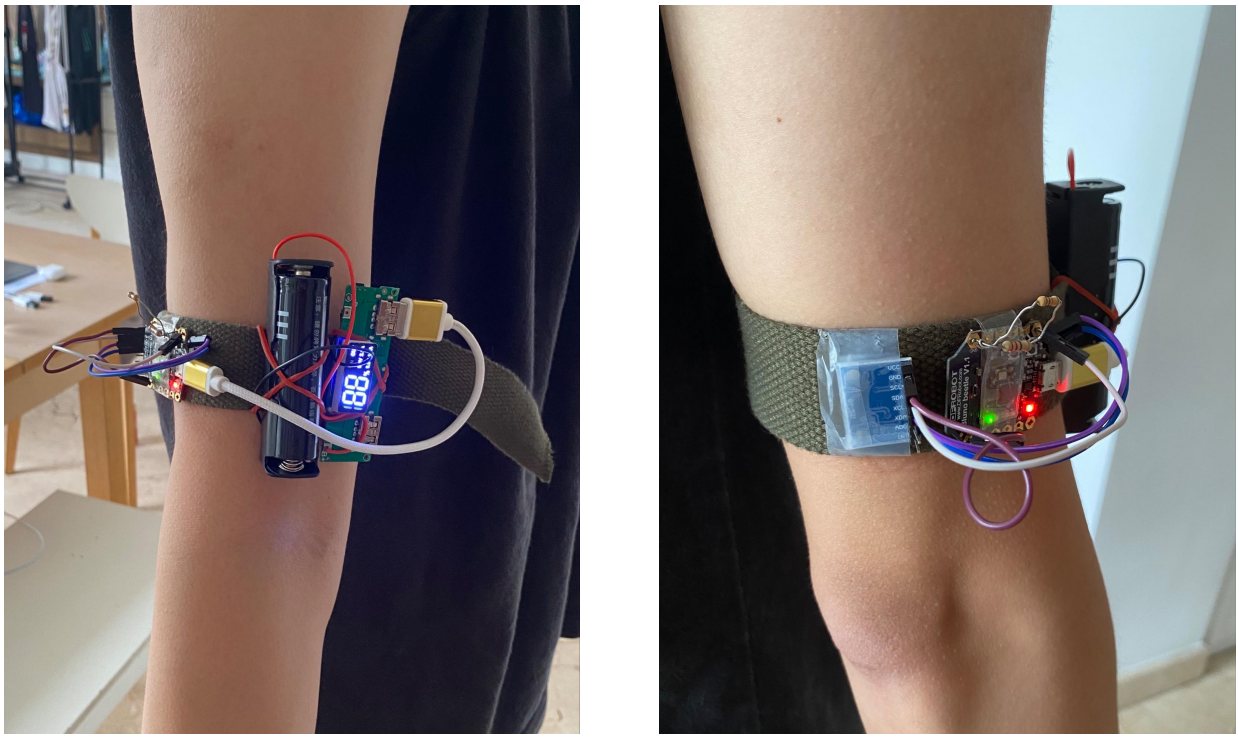


Figure 3. Final hardware design for dance move classification

As we need to measure the fitness level of one of the dancers, an additional hardware set containing 1 3.7V battery + 1 voltage regulator + 1 DFR0339 + 1 AT-04-001 is to be attached to the upper left arm of that dancer (Figure 4). This is because all the dance moves require arm movements, hence the fatigue level of the dancer at the upper arm would be more noticeable. With the increased fatigue level of the dancer at the arms, the EMG signal would see an increase in its amplitude and decrease in its frequency.

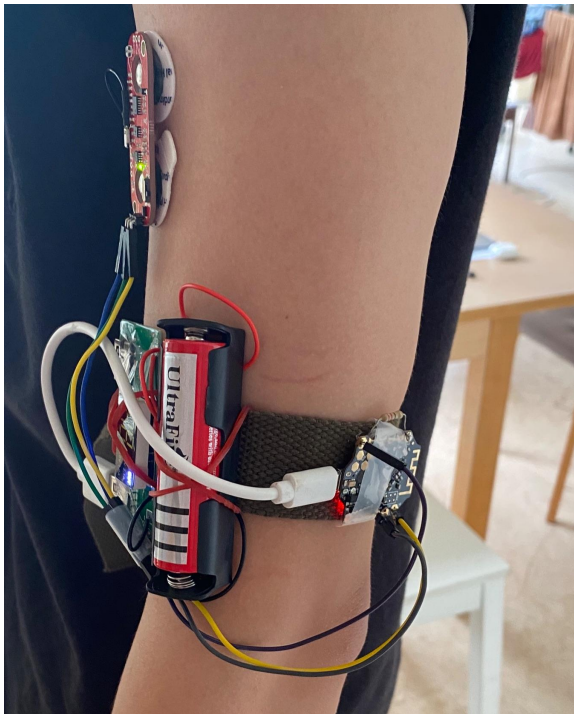


Figure 4. Final hardware design for EMG signals

2.3 Main algorithm for activity detection

Below is a summary of the major steps algorithm. The details of each step will be illustrated in the sections below. The collaboration diagram is shown above in Figure 2.

1. Motion and muscle data from the gyroscope and accelerometer is transmitted to the Beetle.
2. The Beetle takes in the data through reading the analog input and Arduino's built-in "Wire.h" library.
3. The Beetle preprocesses the data and transmits it through Bluetooth to the laptop.
4. The laptop will encrypt then encode the motion data before transferring to Ultra96 through TCP/IP and SSH tunneling.
5. Ultra96 evaluates the motion data and uses its previously trained machine learning model to predict the dance move based on the data. A detailed explanation of how the training data is gathered and used is illustrated in section 5.1.4 model training, validation and testing.
6. The muscle data and the model's prediction of the dance move will be encrypted, encoded, and sent to the evaluation server through TCP/IP.

Section 3 Hardware Details

Section 3.1: Hardware sensors [Yaoyue]

3.1.1 Components / Devices

The following components/devices will be used for each hardware set:

- 18650 3.7V rechargeable battery
- 18650 Charging Module[1]
- DFR0339 Bluno Beetle[2]
- MPU6050 GY-521 Triple Axis Accelerometer[3]
- AT-04-001 MyoWare™ Muscle Sensor[4]

The 18650 charging module will take in voltage input from the 18650 3.7V rechargeable battery with a voltage of 3.7V and then supply a steady 5V to the Bluno Beetle. This particular charging module is chosen as it not only serves as a voltage regulator, but it also offers a circuit board protection feature and LED digital display. This is helpful for

---

[1] *2.21SG$ 20% OFF|LED Dual USB 5V 2.4A Micro/Type C USB Mobile Power Bank 18650 Charging Module Lithium Battery Charger Board Circuit Protection|Integrated Circuits|—AliExpress*. (n.d.). Aliexpress.Com. Retrieved September 4, 2021, from //www.aliexpress.com/item/1005002133023221.html?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp=

[2] *DFR0339 DFRobot Datasheet.* DFRobot. https://www.application-datasheet.com/pdf/dfrobot/dfr0339.pdf

[3] *MPU6050.* http://www.haoyuelectronics.com/Attachment/GY-521/mpu6050.pdf

[4] *3-lead Muscle / Electromyography Sensor for Microcontroller Applications.* Myoware. https://cdn.sparkfun.com/datasheets/Sensors/Biometric/MyowareUserManualAT-04-001.pdf

our project in two ways: Firstly, this module can prevent the Bluno Beetle from being overpowered which may damage the board; Secondly, we can easily see the battery level and recharge the battery if necessary. Each Bluno Beetle will then be connected to one motion processing unit. An additional Bluno Beetle will be connected to the Myoware Muscle Sensor to collect the muscle data. All Bluno Beetles will obtain sensor data and transmit the data through the Bluno Link to the laptop.

3.1.2 Schematic

The following figure depicts the hardware set for collecting motion data (Figure 5a).

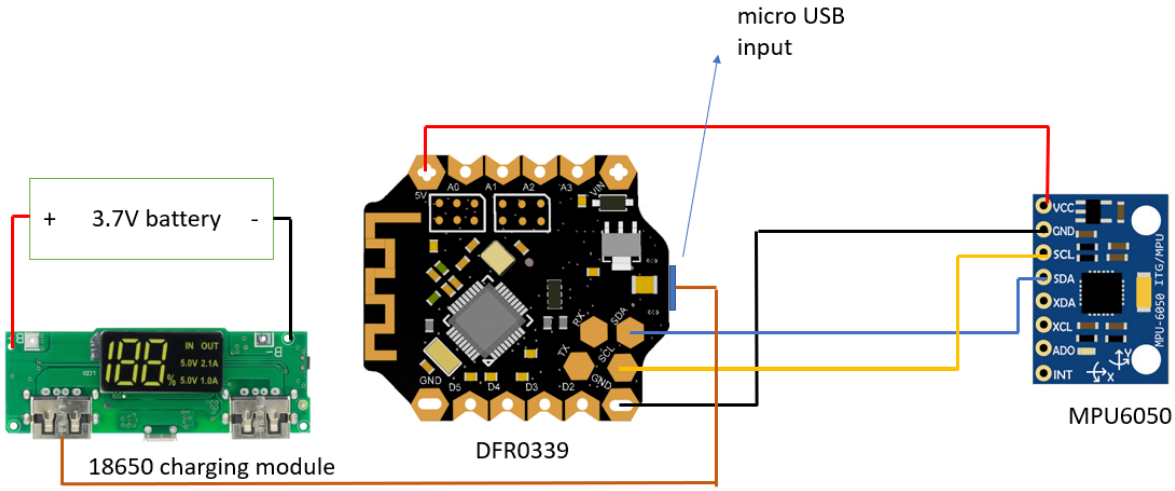


Figure 5a. Schematic for hardware set which collects motion data

The following figure depicts the hardware set for collecting muscle data (Figure 5b).

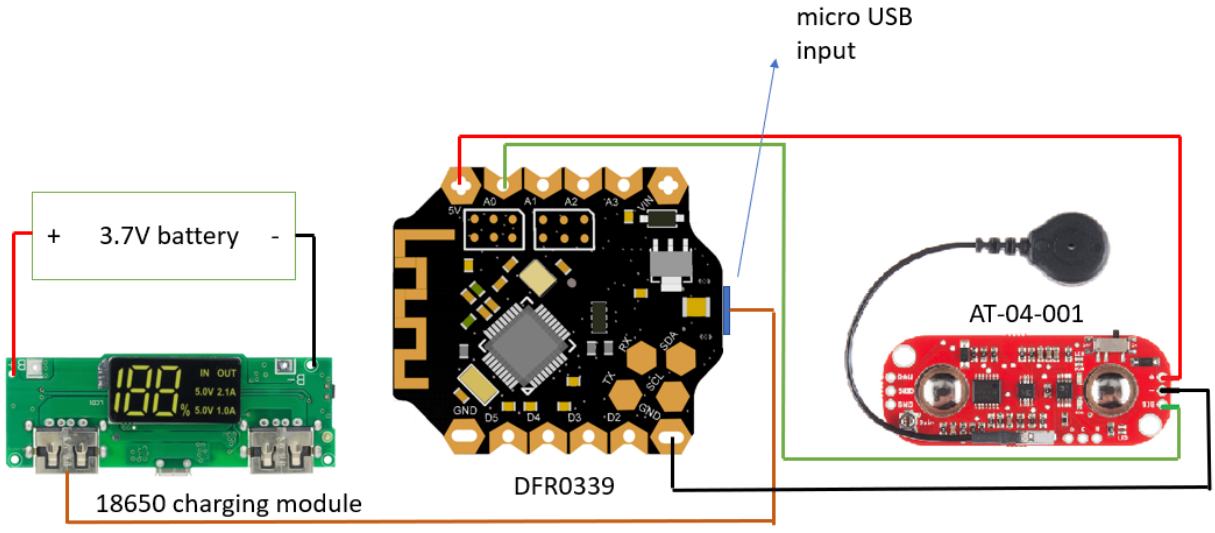


Figure 5b. Schematic for hardware set which collects muscle data

3.1.3 Pin table

The following pin table describes the connection between the Bluno Beetle and the Motion Processing Unit.

DFR0339 (Bluno Beetle)	MPU6050 (Motion Processing Unit)
5V	VCC
SDA	SDA
SCL	SCL
GND	GND

The following pin table describes the connection between the Bluno Beetle and the Myoware Muscle Sensor.

DFR0339 (Bluno Beetle)	AT-04-001 (Myoware Muscle Sensor)
5V	+
GND	-
A0	SIG

3.1.4 Operating voltage

The input voltage of the 18650 charging module is between 3.0V to 4.2V. The current drawn by the 18650 charging module is about 20mA.

The operating voltage of the Bluno Beetle is 5V. Due to insufficient equipment to measure the current drawn from the Bluno Beetle (it is powered by micro USB cable), I take the maximum current output of a laptop USB port as a reference, and assume that the current drawn by the Bluno Beetle shall not exceed 1A. In reality, the current drawn from the Bluno Beetle should be much less than 1A. I shall show in the following section on battery design analysis that even by taking 1A as the current drawn by the Bluno Beetle, the battery design implemented will suffice.

The operating voltage of the Motion Processing Unit is between 2.375V to 3.46V. However, as the breakout board has a voltage regulator, we can connect the board directly to the 5V source from the Bluno Beetle. The current drawn by the Motion Processing Unit is about 6mA.

The operating voltage of the Myoware Muscle Sensor is between 2.9V and 5.7V, thus we can connect the board directly to the 5V source from the Bluno Beetle. The current drawn by the Myoware Muscle Sensor is about 20mA.

3.1.5 Battery design analysis

In the following battery design analysis, we assume that the hardware system needs to last for 1 hour:

No. of batteries = 3.7V / 3.7V = 1
Power_in = Power_in (18650 Charging Module) + Power_in (DFR0339) + Power_in (MPU6050) + Power_in (AT-04-001) = 20mA * 3.7V + 1A x 5V + 6mA * 5V + 20mA * 5V = 5.204W
Total voltage of batteries = 3.7V
Load energy = 5.204W * 1h = 5.204Wh
Load capacity = 5.204Wh / 3.7V = 1.4065Ah = 1406.5mAh
Load capacity per battery = 6000mAh
No. of parallel branches = 1406.5 / 6000 = 1 (rounded up)

3.1.6 Data reading

We made use of the Arduino platform's in-built "Wire" library to establish an I2C connection between the Bluno Beetle and the MPU. Using this library, we could obtain the gyroscope and accelerometer sensor values[5]. After the sensor values are read, the lower register of each sensor value has to be swapped with that of the higher register as the MPU and the AVR chip on the Bluno Beetle orders the bytes differently: the AVR chip has a low byte at the lower address, while the MPU has a high byte at the lower address.

[5] Tutorial: How to use the GY-521 module (MPU-6050 breakout board) with the Arduino Uno. (2017, October 5). *Michael Schoeffler*. https://mschoeffler.com/2017/10/05/tutorial-how-to-use-the-gy-521-module-mpu-6050-breakout-board-with-the-arduino-uno/

```
// Read raw values, 14 bytes at once, containing acceleration, temperature and gyro
int read_gyro_accel_vals(uint8_t* accel_t_gyro_ptr) {

  accel_t_gyro_union* accel_t_gyro = (accel_t_gyro_union *) accel_t_gyro_ptr;

  int error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *) accel_t_gyro, sizeof(*accel_t_gyro));

  uint8_t swap;
  #define SWAP(x,y) swap = x; x = y; y = swap

  SWAP ((*accel_t_gyro).reg.x_accel_h, (*accel_t_gyro).reg.x_accel_l);
  SWAP ((*accel_t_gyro).reg.y_accel_h, (*accel_t_gyro).reg.y_accel_l);
  SWAP ((*accel_t_gyro).reg.z_accel_h, (*accel_t_gyro).reg.z_accel_l);
  SWAP ((*accel_t_gyro).reg.t_h, (*accel_t_gyro).reg.t_l);
  SWAP ((*accel_t_gyro).reg.x_gyro_h, (*accel_t_gyro).reg.x_gyro_l);
  SWAP ((*accel_t_gyro).reg.y_gyro_h, (*accel_t_gyro).reg.y_gyro_l);
  SWAP ((*accel_t_gyro).reg.z_gyro_h, (*accel_t_gyro).reg.z_gyro_l);

  return error;
}

int MPU6050_read(int start, uint8_t *buffer, int size)
{
  int i, n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);
  if (n != 1) // no byte is written
    return (-10);

  n = Wire.endTransmission(false);    // hold the I2C-bus
  if (n != 0)  // error in ending transmission
    return (n);

  Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true); // request data and release I2C-bus after data is read
  i = 0;
  while(Wire.available() && i<size)
  {
    buffer[i++]=Wire.read();
  }
  if ( i != size)
    return (-11);

  return (0);  // return : no error
}
```

No library is required to read the muscle sensor values as they could be read directly from the analog input of the beetle.

3.1.7 Data Processing

The signal from the muscle sensor is already filtered, rectified, and amplified, thus no additional data-processing needs to be done on our end. On the other hand, raw data is received from the MPU, and hence data pre-processing needs to be done. Data pre-processing of MPU involves calibration, conversion of units, and noise removal.

3.1.7.1 Calibration

Errors of MPU were pre-computed before the dance move. This can be done by taking the average of the first 10 readings. After which, the errors were stored within global variables. During the actual dance, the errors calculated were subtracted from the raw readings from the accelerometer and gyroscope.

3.1.7.2 Conversion of units

The raw readings from the accelerometer and gyroscope were further converted into their appropriate units.

The accelerometer has a full-scale range of +/- 2g with Sensitivity Scale Factor of 16,384$LSB(Count)/g$ by default. Thus, we can convert the units of the raw accelerometer values to m/s^2 by dividing them by 16384.

On the other hand, the gyroscope has a full-scale range of +/- 250 °/s with Sensitivity Scale Factor of 131 $LSB(Count)/°/s$ by default. Thus, we can convert the units of the raw gyroscope values to °/s by dividing them by 131.

3.1.7.3 Noise Removal

Noise removal was achieved through the use of a complementary filter. The problem with the accelerometer and gyroscope sensor values is that accelerometer data is noisy on short time scales, and gyroscope data drifts on longer timescales. The complementary filter combines both for greater accuracy on accelerometer and gyroscope sensor data. The code snippets below show how the complementary filter is applied.

```
// Get angle values from accelerometer
float RADIANS_TO_DEGREES = 180/3.14159;
float accel_angle_y = atan(-1*accel_x/sqrt(pow(accel_y,2) + pow(accel_z,2)))*RADIANS_TO_DEGREES;
float accel_angle_x = atan(accel_y/sqrt(pow(accel_x,2) + pow(accel_z,2)))*RADIANS_TO_DEGREES;

float accel_angle_z = 0;

// Compute the gyro angles
float dt =(t_now - get_last_time())/1000.0;
float gyro_angle_x = gyro_x*dt + get_last_x_angle();
float gyro_angle_y = gyro_y*dt + get_last_y_angle();
float gyro_angle_z = gyro_z*dt + get_last_z_angle();
```

```
// Apply the complementary filter
float alpha = 0.96;
float angle_x = alpha*gyro_angle_x + (1.0 - alpha)*accel_angle_x;
float angle_y = alpha*gyro_angle_y + (1.0 - alpha)*accel_angle_y;
float angle_z = gyro_angle_z;  //Accelerometer doesn't give z-angle
```

3.1.8 Motion detection

The start of a move could be detected if there is a drastic change in the accelerometer and gyroscope values after a break between the dance moves. I set the period of this break to be 2s. This is because there are some small breaks within each dance move that should not be considered. We consider the dancer to be moving if the sum of the absolute values of all the accelerometer readings exceeds 1.9 or the sum of the absolute values of all the gyroscope readings exceeds 0.5. However, as the sensor values are captured at high frequency, it may not be accurate to just deduce whether the dancer is strictly dancing from the sensor values captured at a single iteration. Hence, only if movement is captured consecutively for 4 iterations then we consider the dancer to be dancing. Vice versa, the dancer is considered to have stopped dancing if the sensor values do not change a lot for at least 4 iterations. The code snippet below shows the start-of-move detection algorithm.

```
if((abs(accel_x2) + abs(accel_y2) + abs(accel_z2) > 1.9)
|| (abs(angle_x2) + abs(angle_y2) + abs(angle_z2) > 0.5)
|| (abs(accel_z2) > 0.9 && abs(accel_x2) < 0.2 && abs(accel_y2) < 0.2)){ // when doing scarecrow dance move
  countHigh++;
  if(countHigh == 4) {
    countHigh = 0;
    if(millis() - endMoveTime >= 2000 && hasStarted == false) {
      if(Serial.available())Serial.println("Start movement");
      hasStarted = true;
      hasEnded = false;
    }
  }
  countLow = 0;
}
else {
  countLow++;
  if(countLow == 4 && hasEnded == false) {
    countLow = 0;
    endMoveTime = millis();
    hasEnded = true;
    hasStarted = false;
  }
  countHigh = 0;
}
```

3.1.9 Muscle fatigue detection

We used the mean spectral frequency (MNF) of the EMG signal to detect muscle fatigue. MNF can be observed to decrease with time as muscle fatigue level increases. The definition of MNF is given by:

$$MNF = \sum_j f_j P_j / \sum_j P_j$$

where f_j is the frequency value of EMG power spectrum at the frequency bin j, and P_j is the EMG power spectrum at the frequency bin j.

The arduinoFFT library was used to implement Fast Fourier Transform (FFT) on the EMG signal. The FFT algorithm that this library uses works with a finite number of samples and this number needs to be a power of 2. Hence, the EMG signal was sampled at a frequency of 1000Hz, and for each 128 samples, a FFT was performed and the corresponding MNF was calculated. To reduce the unnecessary noise in the MNF graph, the average of every 4 MNF values would be printed in the serial monitor.

The following graph shows the MNF of the EMG signal that was captured from a dancer during the final project evaluation on 11th Nov 2021.

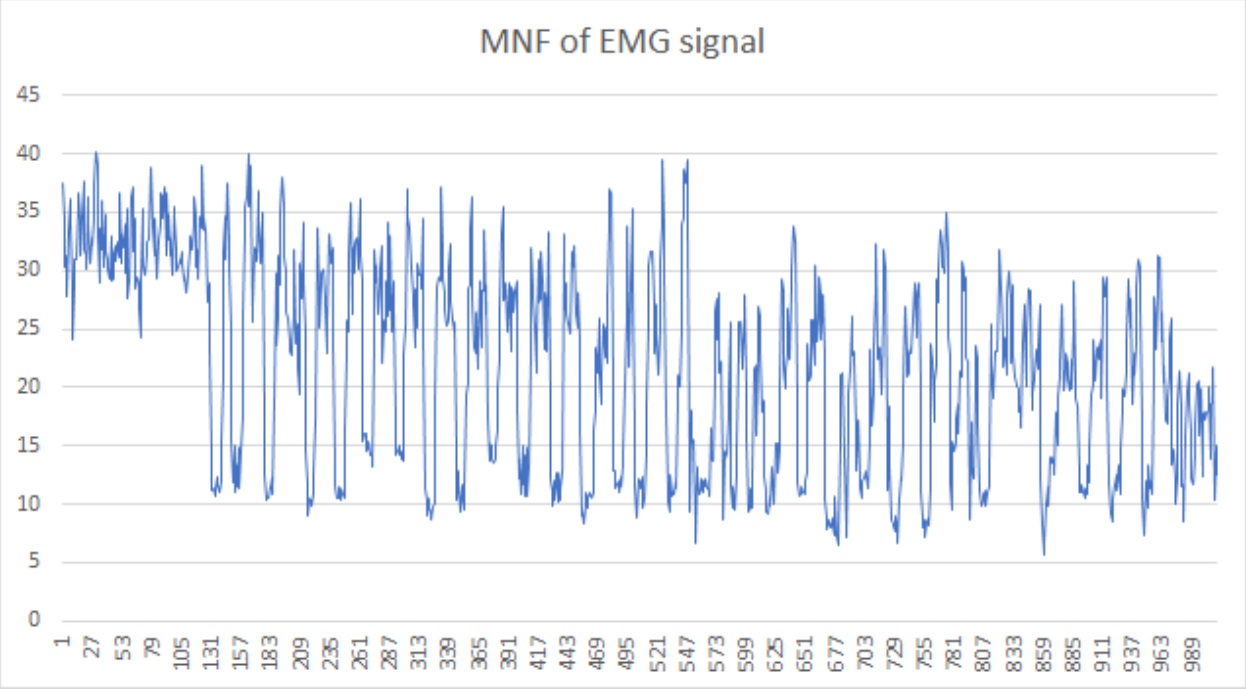


Figure 6: Graph of MNF of EMG signal plotted against time

The overall trend of the graph is decreasing, indicating greater muscle fatigue levels across time. In addition, the MNF decreased drastically during the dance moves, which means that there was greater muscle fatigue during the dance.

Section 3.2: Hardware FPGA [Marcus]

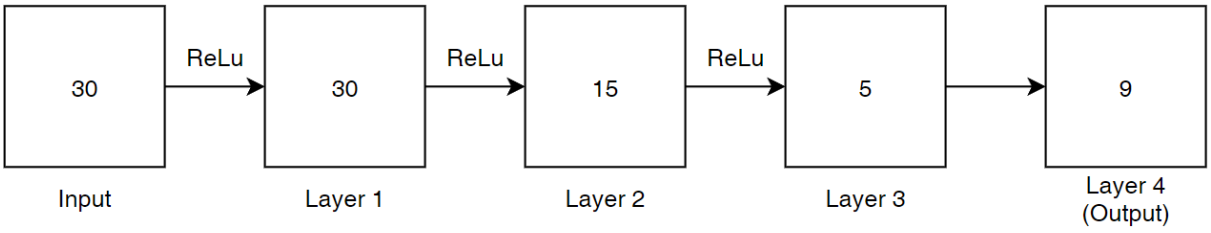
3.2.1 Ultra96 synthesis and simulation

The synthesis will be set up for the target device, xczu3eg-sbva484-1-i, which is the part on the board Ultra96 v2. Synthesis will be done once in Vitis High Level Synthesis (HLS). Vitis HLS will synthesize the C code which contains the neural network algorithm into a Register Transfer Level (RTL) implementation and package it into an Intellectual Property (IP) for use in Vivado Design Suite. In Vivado synthesis, implementation and bitstream are generated. Synthesis transforms the RTL design into a gate-level representation. Implementation does the necessary steps to place and route the netlist onto the FPGA resources, while meeting the design's logical, physical and timing constraints. Finally, the bitstream is generated to program the FPGA.

Simulation in Vitis HLS with C/RTL Cosimulation to verify that the RTL is functionally identical to the C source code. This can be done by creating a test bench case, simulating an input and verifying that the output matches that of the neural network trained.

3.2.2 How the neural network model will be realized on the FPGA board

The neural network used is a multilayer perceptron (MLP). The simplified layout of the MLP is shown below, more details on the MLP is in Section 5.1 on machine learning.



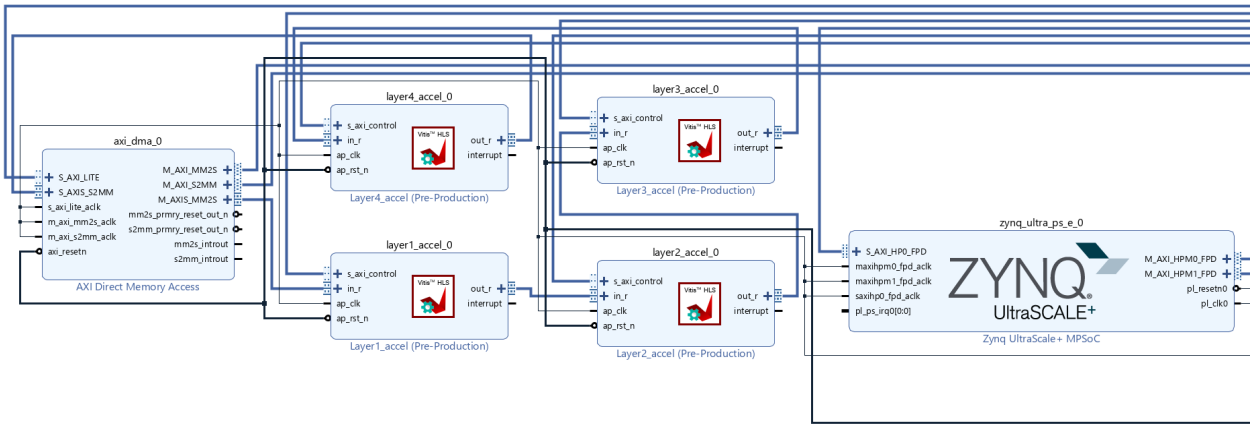
Each box is a layer with its label underneath. The number in the box represents the number of neurons or nodes in that layer. The rectified linear activation function (ReLu) is used at each layer except the output.

The code snippet below is the main function in Vitis HLS that does the computation for the value of each neuron in Layer 1. This function can be extended to subsequent layers by changing the values of N1 and N2 which are defined in a header file.

```
template <typename T> void kernel_layer1(T a[N1], T out[N2]) {
loop1:
    for (int i = 0; i < N2; i++) {
    loop2:
        DataType sum = 0;
        //neuron
        for (int j = 0; j < N1; j++) {
            sum += a[j] * weight[j][i];
        }
        sum += bias[i];
        //activation function
        if (sum > 0) {
            sum = sum;
        } else {
            sum = 0;
        }
        out[i] = sum;
    }
  return;
}
```

loop1 iterates through every neuron in the out to compute its value, while loop2 iterates through every neuron in a, in this case a is the input layer and out is Layer 1. The weight and bias are constants which are defined in a header file, these values are derived after training of the neural network and will be stored in blocks of read-only memory (ROM) when implemented in the FPGA.

Vitis HLS then exports the layers as IPs in Vivado Design Suite, where connections between the required IPs are done in the block design in IP Integrator. The main connection is between the Zynq Processor, the Direct Memory Access (DMA) and the Layers, this is illustrated by the following diagram.



The connection shows that processed data is passed into the neural network on the FPGA via DMA and the computed output is then passed back to the processor through the DMA as well.

The bitstream to program the FPGA is then generated and uploaded to the Ultra96 for usage. Programming the FPGA board with the bitstream on the Ultra96 is done using the Overlay class in PYNQ (Python productivity for Zynq). This is shown in the following code snippet.

```python
from pynq import (allocate, Overlay)

np.set_printoptions(suppress=True)

ol = Overlay('mlp.bit')

dma = ol.axi_dma_0
l1 = ol.layer1_accel_0
l2 = ol.layer2_accel_0
l3 = ol.layer3_accel_0
l4 = ol.layer4_accel_0

DIM1 = 30 #input layer
DIM2 = 9  #output layer
input_buffer = allocate(shape=(DIM1,), dtype=np.float32)
output_buffer = allocate(shape=(DIM2,), dtype=np.float32)

CTRL_REG = 0x00
AP_START = (1<<0) # bit 0
AUTO_RESTART = (1<<7) # bit 7

def run_kernel():
    dma.sendchannel.transfer(input_buffer)
    dma.recvchannel.transfer(output_buffer)
    l1.write(CTRL_REG, (AP_START | AUTO_RESTART)) #initialise layers
    l2.write(CTRL_REG, (AP_START | AUTO_RESTART))
    l3.write(CTRL_REG, (AP_START | AUTO_RESTART))
    l4.write(CTRL_REG, (AP_START | AUTO_RESTART))
    dma.sendchannel.wait()
    dma.recvchannel.wait()
```

The DMA is allocated two buffers, which matches the sizes of the input and output layer for sending data to and receiving data from the FPGA. The run_kernel function is called to start data transfer to the FPGA and initialise the layers and wait to receive the output.

3.2.3 Design's timing, power and area

The design's performance can be seen in the project summary in Vivado Design Suite. Area utilization can be viewed in the amount of flip-flops (FF), lookup tables (LUT) and digital signal processing (DSP) used in the design. The following figures show the timing, power and area utilization of the design.

Design Timing Summary

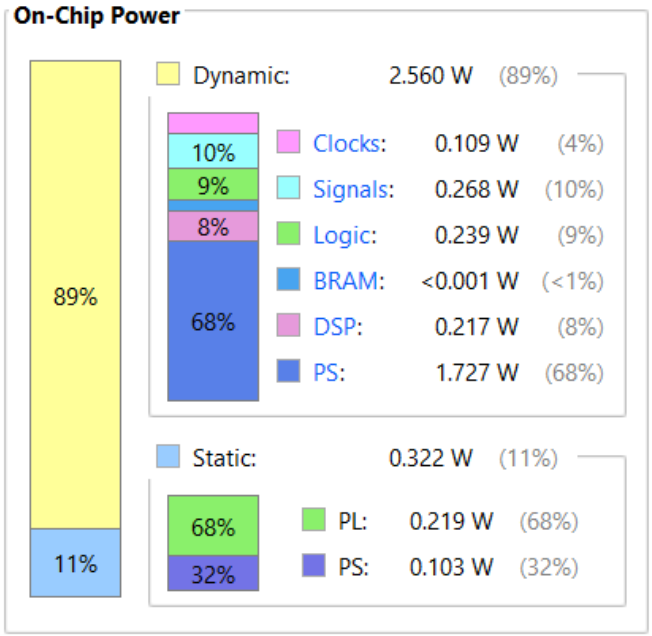
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.753 ns	Worst Hold Slack (WHS):	0.010 ns	Worst Pulse Width Slack (WPWS):	3.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	90641	Total Number of Endpoints:	90641	Total Number of Endpoints:	40932

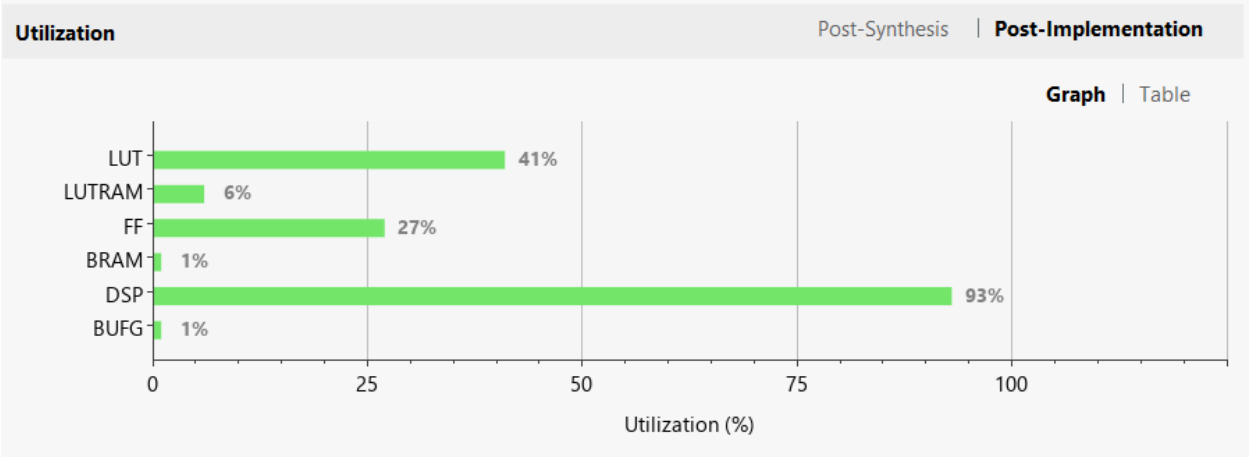
All user specified timing constraints are met.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

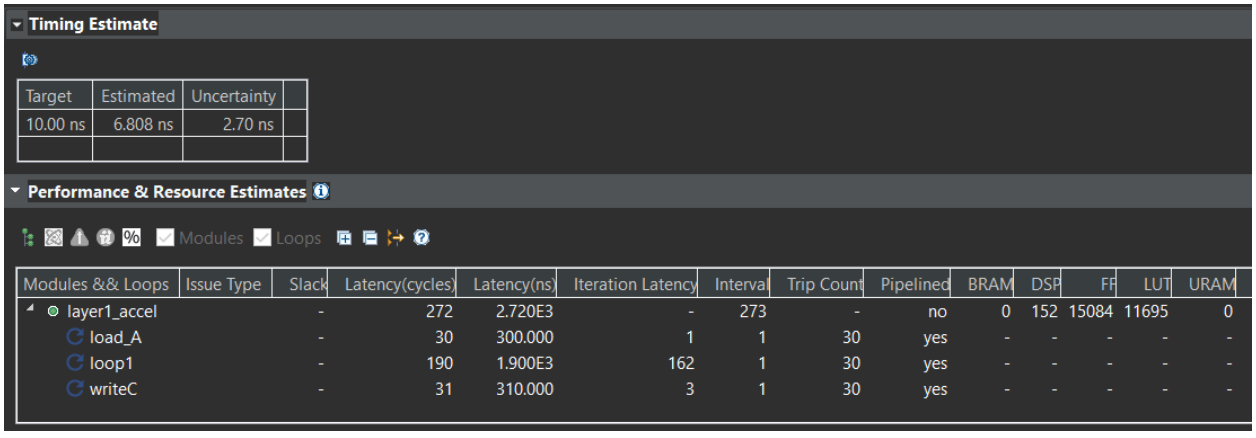
Total On-Chip Power:	**2.882 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**32.9°C**
Thermal Margin:	67.1°C (24.2 W)
Effective ϑJA:	2.7°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity





Alternatively, Vitis HLS shows the timing and resource estimates for each layer, below is the estimates for layer 1.



Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	6.808 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules && Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
layer1_accel	-	-	272	2.720E3	-	273	-	no	0	152	15084	11695	0
load_A	-	-	30	300.000	1	1	30	yes	-	-	-	-	-
loop1	-	-	190	1.900E3	162	1	30	yes	-	-	-	-	-
writeC	-	-	31	310.000	3	1	30	yes	-	-	-	-	-

3.2.4 Potential optimizations of the neural network accelerator

The neural network accelerator is already optimized through the use of loop pipelining, allowing computation, reading of input and writing of output to be done simultaneously. This can be seen in the resource estimate in Vitis HLS from Section 3.2.3. However, because of the high usage of DSP with the current implementation, another approach to optimization is also used.

The optimization implemented at layer 3 of the neural network accelerator is loop unrolling. The following code snippet shows the usage of #pragma HLS unroll.

```
template <typename T> void kernel_layer3(T a[N1], T out[N2]) {
loop1:
    for (int i = 0; i < N2; i++) {
#pragma HLS unroll
    loop2:
```

Because of the high usage of DSP, optimization is required to allow the implementation of the neural network without exceeding the available resources. The optimization here mainly uses FF and LUT rather than DSP. Thus, this allows the implementation of all the layers by distributing the resources.

There are multiple other optimizations that can be implemented. One such is using fixed-point representation to compute the neuron values. Vitis HLS has a library for this purpose as shown below.

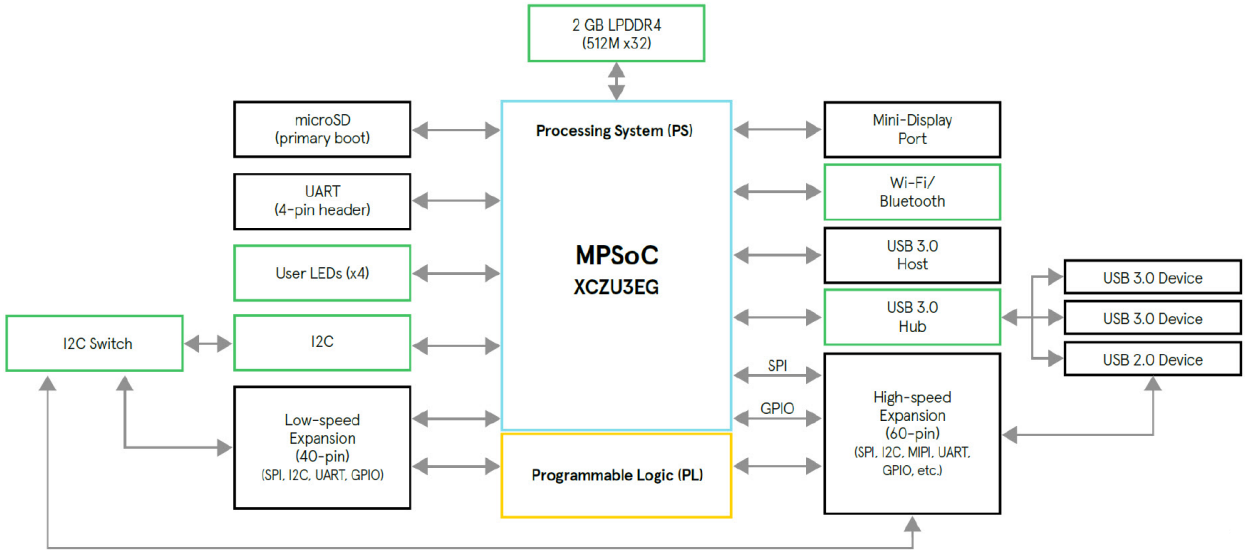
```
#include <ap_fixed.h>
...
ap_fixed<18,6,AP_RND > my_type;
```

This defines an 18 bit variable with 6 bits representing the integer and 12 bits representing the decimals, and it rounds to positive infinity. However, this optimization was not implemented largely due to the difficulty of precision. Because the number of bits for integer and decimal must be fixed, one must verify that the calculations do not exceed the fixed point representation by looking through the values. Thus, floating-point representation is the default for this design.

Another optimization that can improve the flexibility of the neural network accelerator is to read the weight and bias from a file on loading the overlay, rather than storing the data in ROMs. This allows the weight and bias to be changed efficiently without having to update the layer IPs and regenerating the bitstream each time.

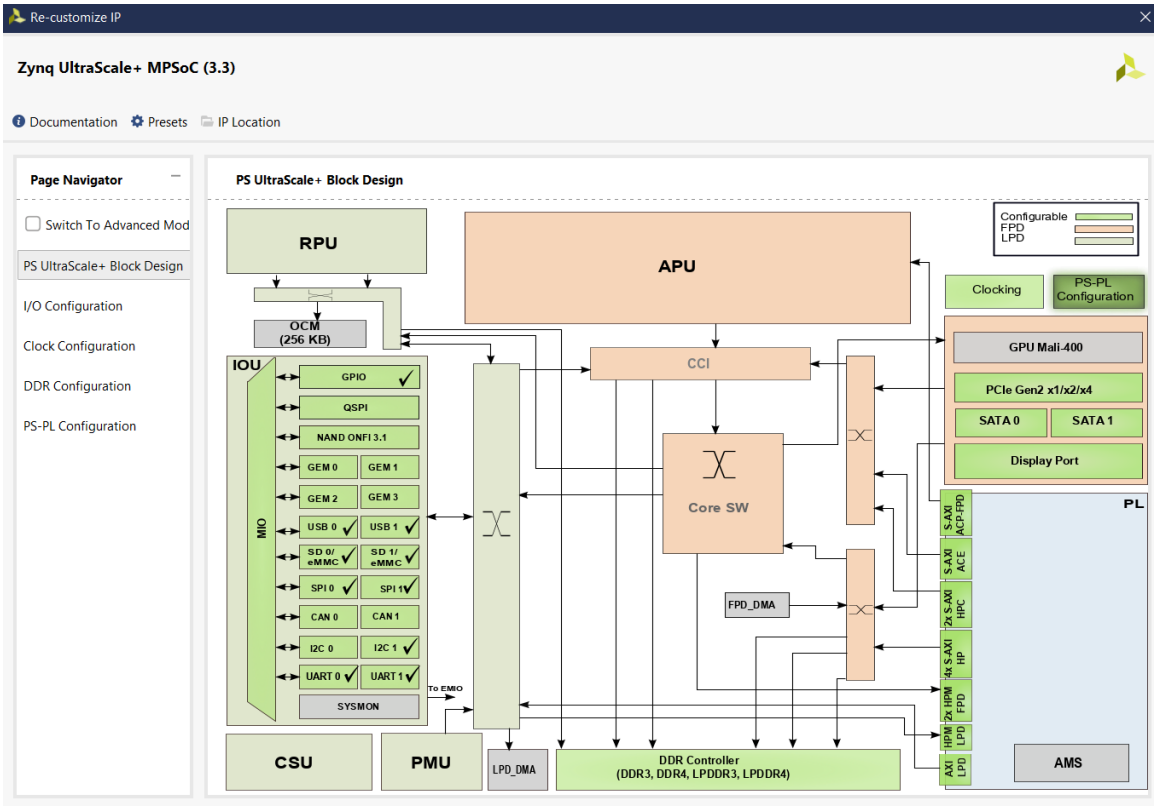
3.2.5 Power management of Ultra96 CPU + FPGA

The following block diagram shows the layout of the Ultra96.



The main components involved in the neural network accelerator are PS and PL.

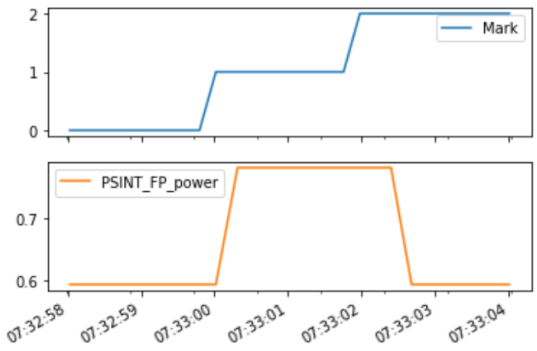
As such, peripherals like LEDs, Mini-Display Port, USB ports can be disabled to reduce power consumption. This can be achieved in Vivado Design Suite by directly modifying the MPSoC IP in the IP Integrator. The figure below shows the customized IP with display port disabled as seen by the absence of the tick.



Power consumption can then be monitored by observing the power lines using PYNQ. The figure below illustrates how the power can be graphed for reading.

```
%matplotlib inline
recorder.frame.plot(subplots=True)
```

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f66fb2780>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f66f36320>],
      dtype=object)
```



Power consumption can also be adjusted by varying the clock of the Ultra96 PS and PL. The PS clocks can be changed in Vivado IP integrator similar to disabling the peripherals. PL clocks on the other hand can be changed through PYNQ as shown below.

```
from pynq import Clocks

print(f'CPU:   {Clocks.cpu_mhz:.6f}MHz')
print(f'FCLK0: {Clocks.fclk0_mhz:.6f}MHz')
print(f'FCLK1: {Clocks.fclk1_mhz:.6f}MHz')
print(f'FCLK2: {Clocks.fclk2_mhz:.6f}MHz')
print(f'FCLK3: {Clocks.fclk3_mhz:.6f}MHz')
```

```
CPU:   1199.988000MHz
FCLK0: 499.995000MHz
FCLK1: 499.995000MHz
FCLK2: 499.995000MHz
FCLK3: 499.995000MHz
```

```
from pynq import Clocks

Clocks.fclk0_mhz = 50
Clocks.fclk1_mhz = 50
Clocks.fclk2_mhz = 50
Clocks.fclk3_mhz = 50
```

Section 4 Firmware & Communications Details

Section 4.1 Internal Communications [Tammy]

4.1.1 Task Management

Internal communications was done by making use of the BLE Link component. Since each dancer would only require 1 Beetle, each python code run would only need to focus on receiving data from that one Beetle and then sending it to the server.

Thus, the task on the Beetles was managed sequentially, where it first received data from the Beetle, processed it and sent it to the server.

```
while(True):
    if connected:
        data = str(retrieveData())
        clean = dataCleaning(data)

        if data != "":
            if clean:
                sendData(data)
```

The above table shows part of our main code. The first step of retrieving data from the Beetle is done through ‘data = str(retrieveData())’. This would call the retrieveData() function to retrieve data from the beetle and assign it to ‘data’. The ‘data’ is then sent to a dataCleaning() function to make sure the data received is of the right format. If dataCleaning() returns ‘True’, that means the data is of the right format. It will then send the data to the server through the sendData() function.

4.1.2 Setup And Configuration of BLE Interface

The Beetle and BLE Link would be configured by using a series of AT commands through arduino's serial monitor. These AT commands would enable the user to set up the Beetle for bluetooth communications and can be accessed by following these steps:

1) Plug in the Beetle/BLE Link
2) Open Arduino IDE
3) Select the serial port for the Beetle/BLE link
4) Open the Serial Monitor
5) Select "No line ending"
6) Type "+++" and hit the send button

If the steps are completed successfully, a message "Enter AT Mode" should be displayed. Upon entering AT Mode, change "No line ending" to "Both NL & CR". Now, AT commands can be used to configure the bluetooth devices.

The following table lists AT commands that were used in this project.

AT Command	Function	Remarks
AT+SETTING=	Set device to initial settings	'?' - queries current setting 'DEFAULT' - sets to initial setting 'DEFPERIPHERAL' - sets to initial slave setting 'DEFCENTRAL' - sets to initial host setting
AT+MAC=?	Queries the device's MAC address	
AT+BIND=	Set binding bluetooth device MAC address	'?' - queries current binding MAC address '[MAC address]' - bind device to [MAC address] - Eg. "AT+BIND=0xC4BE84267CEC" binds the current device to the device with MAC address "0xC4BE84267CEC"

When an AT command has been entered successfully, an "OK" message would be displayed.

The Bluno Beetle should be set as the peripheral device while the BLE Link should be set as the central device. For each pair of Bluno Beetle and BLE Link, the MAC address of each device should be bound to that of the other in that pair. This ensures that the Bluno Beetle is always connected to the same BLE Link which enables us to be able to accurately keep track of each device.

Upon successful configuration, the Beetle can now be connected to a different power source while the BLE Link is powered by the computer where the client side code is to be run.

4.1.3 Protocol Over BLE

When the Beetle is powered, it would begin to search for a bluetooth device to connect to. Similarly, the BLE Link would search for a bluetooth device when it is powered. Because each pair of Beetle and BLE are bound to one another via their MAC addresses. They would find and connect with one another.

After establishing a connection, the computer will start a handshaking protocol with the Beetle through the BLE Link.

The following code is an excerpt of the main function.

```
connected = False

while(True):
    if connected:

        …

    else:
        connected = deviceHandshaking()
```

The variable ‘connected’ is set to ‘False’ upon initialization. It will thus fail the if condition and run the else condition. There, it calls the function deviceHandshaking().

```
def deviceHandshaking():
    bluno.write(b'hello')
    receive = bluno.readline().decode('ascii')
    if "ack" in receive:
        timeout = 0
        print("Device connected")
        return True
    else:
        return False
```

The function deviceHandshaking() would first send a “hello” to the Beetle. If the Beetle successfully receives this message, it will reply with “ack”. The code checks to see if “ack” has been received and if it has, it will output a message “Device connected” and return True, otherwise, it would return False. The variable ‘connected’ in the main function would then be set to True or remain as False as determined by the handshaking function.

On the Beetle side, it will be ready to send data when the laptop polls for it once it replies with “ack”. The code below shows the handshaking on the Beetle side. It will continuously run deviceHandshaking() until a connection is established which sets ‘check’ to 1 and ends the while loop.

```
int deviceHandshaking()
{
  int check = 0;
```

```
  while(check == 0) {
    if (Serial.available()){
      String received = Serial.readString();
      if (received != "") {
        Serial.println("ack");
        check = 1;
      }
    }
  }
}
```

Transfer of data is initiated by the laptop. When the python code for the client side is run, it'll first start by setting up all the functions and variables, after which, it will start to call for data using the following code.

MAIN FUNCTION	retrieveData()

…
 	
while(True):
  data = str(retrieveData())
 	
...
	try:
	  bluno.write(b'1')
	  data = bluno.readline().decode('ascii')
	  return data
	...

The python code connects to the BLE Link by calling the port the BLE Link is plugged into. After discovering which port the BLE Link is, we use this initialisation.

```
bluno = serial.Serial("Port Name/Number", 9600, timeout = 1)
```

The baud rate is set to 9600 which gives us good reliability whilst still ensuring that we have enough data to make our predictions.

The main function repeatedly calls the retrieveData() function, and in that function, it'll first write '1' to the Beetle. The Beetle writes to the serial output only when Serial.available() is true. Serial.available() would be false until it receives an input from the BLE Link, and in this case, we just send '1'. After which "bluno.readline().decode('ascii')" is used to read the data that has been received and assign it to the variable 'data'.

There are 2 packet types, namely "Read" and "Write".

No.	Packet Type	Detail

1	Read	Reading data from destination
2	Write	Writing data from destination

Packet types to initiate and acknowledge connections have been incorporated into the handshaking portion of the code, thus it does not have its own explicit packet type. Packet types to acknowledge the receiving of data has also been dropped since the laptop would simply drop incomplete or misformatted data.

4.1.4 Managing Reliability

Reliability of the data will be handled through the following: handshaking, constant polling from Beetle and dropping of misformatted data.

Handshaking has been discussed in the earlier segment so it will not be detailed here. A 2-way handshake would begin at the start of a connection to ensure that both the Beetle and BLE Link are connected to one another. In the event that there is a loss of connection, a 2-way handshake would occur again to ensure that the devices are reconnected before continuing the sending of data.

The constant polling of data by the BLE link ensures 2 things. Firstly, it ensures the connection is not lost due to idleness. Secondly, it ensures that it is always ready to receive data and that there would be no or minimal loss of data. If it discovers that the Beetle has no data to send, it will continue to poll but at a slower rate. The time between polls can be adjusted by changing the value of time.sleep() in the main function.

The reliability of data would be done through the dataCleaning() function.

```
def dataCleaning(rawData):
    partitions = rawData.count(",")
    if partitions == 5:
        array = rawData.split(",")
        for num in array:
            x = num
            if x == "" or x.count(".") > 1 or x.count("-") > 1:
                return False
        return True
    elif rawData.strip() == "Start movement":
        return True
    else:
        return False
```

What it does is to take in the raw data and run various tests on it to see if it fits the correct data format. For example, an expected data would be “Start movement”, which indicates that the user has started to move. If it receives this, it will return True which then passes the data on to the server. However, if it receives jumbled data like “123,12,,,1”, it will find that this data is of the wrong format and just drop the data. The decision to drop the data was made because the amount of misformatted data is small and trying to get this data from the Beetle again would cost time and may lead to other data being lost.

Section 4.2 External Communications [Manuel]

4.2.1 Communication between Ultra96 and evaluation server (w13_eval_client.py)

Library APIs

Python modules used are: os, sys, random, time, socket, base64, and Crypto.Cipher:

```
import os
import sys
import socket
import time
import random
import base64
from Crypto.Cipher import AES
```

Secure Communications

Python’s Socket is a low-level programming module specially used for networking purposes like creating TCP/IP servers or clients. The server will be the provided evaluation server and the client will be the Ultra96. Since both the Evaluation server and the Ultra96 are inside the NUS Firewall, there is no need to do SSH tunneling.

Using input arguments, the IP address, Port number and the Group ID can be determined:

```
if len(sys.argv) != 4:
        print('Invalid number of arguments')
        print('python eval_server.py [IP address] [Port] [groupID]')
        sys.exit()

ip_addr = sys.argv[1]
```

```
port_num = int(sys.argv[2])
group_id = sys.argv[3]
```

The TCP/IP socket is created using:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = (ip_addr, port_num)
sock.connect(server_address)
```

Message Format

The message format will follow the evaluation server's chosen format of:

```
position, 1 action, sync
E.g.: "1 2 3|mermaid|3.0250000000000000"
```

The positions part will be one of the following strings according to the out of the Neural Network process:

```
POSITIONS = ['1 2 3', '3 2 1', '2 3 1', '3 1 2', '1 3 2', '2 1 3']
```

The action part will be one of the following strings according to the out of the Neural Network process:

```
ACTIONS = ['mermaid', 'jamesbond', 'dab', 'window360', 'cowboy', 'scarecrow', 'pushback', 'snake']
```

The position and action will be read from the output.txt of the predict.py

```
line_count = 0
        file = open('/home/xilinx/jupyter_notebooks/Overlays/Week13/output.txt', "r")
        Lines = file.readlines()
        file.close()

        for line in Lines:
        line_count += 1
        message = line.strip()
```

The sync timing is collected from the 3 write_offset.txt and then calculated in the sync_calculate() function

```
sync_calculate():
    maxTime = 0
        minTime = 0
```

```
my_array = [0,0,0]

f1 = open('write_offset01.txt', "r")
time01 = f1.readlines()

for x in time01:
if x != '':
x = float(x.rstrip())

my_array[0] = max(time01)

f2 = open('write_offset02.txt', "r")
time02 = f2.readlines()

for y in time02:
y = float(y.rstrip())

my_array[1] = max(time02)

f3 = open('write_offset03.txt', "r")
time03 = f3.readlines()

for z in time03:
z = float(z.rstrip())

my_array[2] = max(time03)

maxTime = max(my_array)
minTime = min(my_array)
sync = float(maxTime) - float(minTime)

open('write_offset01.txt', "w").close()
open('write_offset02.txt', "w").close()
open('write_offset03.txt', "w").close()

return int( sync*1000 )
```

Then the data is sent to eval_server.py using the socket.sendto() class

```
sock.sendto(encrypt_message(message + '|' + str(sync) + '.'),(server_address))
```

Encryption Protocol

Encryption Standard (AES) is provided by Python's Crypto.Cipher module. In this process a key needs to be provided (secret_key in eval_server.py) and an Initial Vector (iv in the code) needs to be generated in the Client and then sent to the Server with the cipher text. The evaluation server uses Cipher block chaining (CBC) and so will the client.

```
AES.new(key, AES.MODE_CBC, iv)
```

Once encrypted, the iv will be added in front of the cipher_text and then encoded using base64 encoding provided by Python's base64 module. Finally, this encoded message will be sent through the internet via TCP/IP. The server will then decode the encoded message, get and use the iv and the secret_key to decrypt the message and parse the information accordingly.

```
encrypt_message(plain_text):
        secret_key = '1fc394163ee2e70ad768b72ab23e83a3'

        iv = os.urandom(16)

        plain_text = plain_text.replace( '\n', '' )

        plain_text = '#' + plain_text;
        print('[+] ' + plain_text)

        aes = AES.new(secret_key, AES.MODE_CBC, iv)

        if len(plain_text)%16 != 0:
        plain_text = plain_text.ljust(32, '0')
        print('[+] ' + plain_text)
        else:
        pass

        encrypted_data = aes.encrypt(plain_text)

        encoded_data = base64.b64encode(iv + encrypted_data)

        return encoded_data
```

4.2.2 Communication between laptops and Ultra96 (w13_arduino.py)

Library APIs

Python modules used are

```
import sys
import ntplib
import time
from time import ctime
import os
```

Secure Communications

The Laptop-Ultra96 uses similar python socket methods of establishing a TCP/IP connection. The server will be the Ultra96 and the client will be the Laptop with the Bluno Link. Since the Ultra96 is inside the NUS Firewall, SSH tunneling is required to be performed by the server and the client.

It is done by using a terminal to run w13_ssh_server.py on the Ultra96

- ssh <user>@sunfire.comp.nus.edu.sg
  - Password
- ssh xilinx@137.132.86.239
  - Password
- su
  - Password
- cd jupyter_notebooks/CommsExternal
- **Ultra96:** python3 w13_ssh_serverWW.py XXXX

Another terminal to forward the port from Ultra96 to Sunfire

- ssh <user>@sunfire.comp.nus.edu.sg
  - Password
- **Sunfire:** ssh -N -f -L YYYY:localhost:XXXX xilinx@137.132.86.239
  - Password

Another terminal to forward the port from Sunfire to the Laptop where the user will use the ZZZZ port to run w13_arduino.py

- **Laptop:** ssh -N -f -L ZZZZ:localhost:YYYY <user>@sunfire.comp.nus.edu.sg
  - Password

The dancers will connect to their corresponding ssh_server number using their corresponding port numbers.

Dancer 1: w13_arduino.py 10001 ->w13_ssh_server01.py 10001
Dancer 2: w13_arduino.py 10002 ->w13_ssh_server02.py 10002
Dancer 3: w13_arduino.py 10003 ->w13_ssh_server01.py 10003

Message Format

The message format will start with the timestamped Start movement packet then the movement data packets will follow.

```
Start movement/tc0/tc3/
0.22,1.09,0.05,0.02,0.40,-0.01
0.37,1.15,0.00,0.21,-0.38,-0.04
0.37,1.14,-0.01,0.20,-0.37,-0.04
0.39,1.15,-0.02,0.22,-0.72,-0.11
...
```

4.2.3 Network Time Protocol for Dance Synchronization

Network Time Protocol (NTP) can be used to synchronize all participating computers to within a few milliseconds of Coordinated Universal Time (UTC). Using a hierarchical, semi-layered system, the laptops will be synchronized over a network to stratum 1 servers (NTP servers). The beetles will be synchronized to the stratum 2 servers (Laptops). Due to the slower Bluetooth connection between the beetles and the laptops, calibrations (clock offset) will be done to compensate by setting benchmarks beforehand. Due to the remote style of the project, the laptops' connection speeds to the Ultra96 will also be benchmarked and necessary calibrations (clock offset) to the clock synchronization algorithm will be made. Sensors will send a start packet to mark the beginning of the dance move. Local timestamps will accompany the motion data to the Ultra96 to calculate the relative time difference between packets from different dancers. With the relative time difference, clock offset and the connection delay, the actual time difference can be calculated for synchronization delay.

The response is requested from asia.pool.ntp.org

```
c = ntplib.NTPClient()
response = c.request('asia.pool.ntp.org', version=3)
```

The synced the hardware time to the NTP server is calculated in the getActualTime() function

```
getActualTime():
        return time.time() + response.offset
```

The Start movement data from bluetooth are then timestamped before sending it to w13_ssh_server.py

```
timestamp_message(message):
        global amount_received
        global tc0, tc3
        tc0 = getActualTime()
        timestampMessage = str(message) + '/' + str(tc0)
        timestampMessage = timestampMessage + '/' + str(tc3)

        return timestampMessage
```

At the w13_ssh_server.py, the time_offset between the laptop and the ultra96 is calculated in the time_offset() function

```
time_offset():
        global tc0, tc3, ts2, ts1, timeOffset, roundTripTime
        ts2 = getActualTime()
        tc3 = getActualTime()
        roundTripTime = ( ts1 - float(tc0) ) - ( ts2 - float(tc3) )
        print('[+] roundTripDelay : %s' % roundTripTime)
        clockOffset = ts1 - float(tc0) - ( roundTripTime / 2 )
        print('[+] clockOffset : %s' % clockOffset)

        return getActualTime() + clockOffset
```

Finally the results are written on their corresponding write_offset.txt for eval_client.py to read.

```
clockOffset = time_offset( )

With open ("/home/xilinx/jupyter_notebooks/CommsExternal/write_offset01.txt", "a")
as f:
    f.write( str(clockOffset) + '\n')
```

Section 5 Software Details

Section 5.1 Software Machine Learning [Zihao]

In the initial design report, we have done an analysis based on an online dataset to assist our design. The dataset used is Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living (Weiss, Yoneda, & Hayajneh, 2019). Weiss etc. conducted the experiment by allocating accelerometers and gyroscopes on the wrist

and waist separately (i.e., in total, 4 sensors are used). However, in our final prototype, we only use one set of accelerometer and gyroscope placed on the right upper arm. Moreover, we integrate the dance move classification with the relative position detection algorithm to form a pipeline for the final system. The high-level system architecture diagram is shown below in Figure 8. In summary, we take 50 rows of data and use a classifier to determine the participant is dancing or switching position. If the participant is dancing, we take 80 rows of data to perform feature engineering and dance move classification. If the participant is switching positions, we wait for all data to be received and carry out the position detection algorithm.

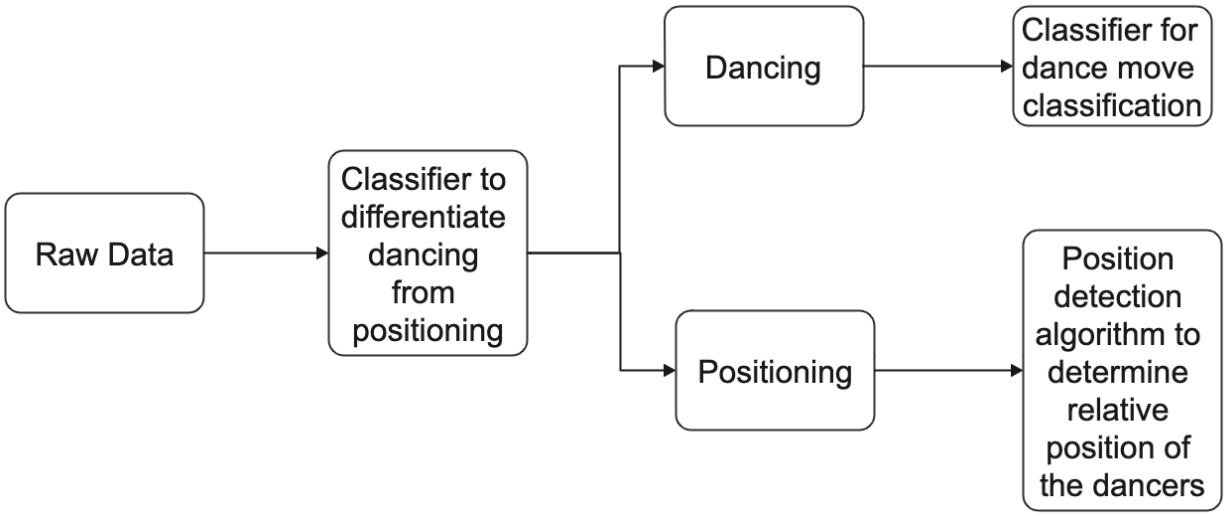


Figure 8. High-level system architecture diagram for relative position and dancing classification

5.1.1 Dance move classification

This section will introduce the algorithm for dance move classification. In summary, this section includes data segmentation, data preprocessing, feature engineering, model training, and model validation.

5.1.1.1 Data segmentation

Since the data will be sent to the software continuously row by row, it is crucial to choose the appropriate sliding window size to analyze the incoming data. If the window size is too small, insufficient information will be collected for data analysis, leading to inaccurate classification results. If the window size is too large, too much information will be passed in for classification and there may be multiple movements or events

occurring in the chosen window, leading to high runtime and incorrect classification results.

Initially, we decided to set the sensor sampling rate at 20Hz and use a sliding window size of 5 seconds without overlapping and make the prediction based on the 5-second data. However, in the actual test, we realized that the initial design is too slow for the classification to perform. In the final design, we change the sensor sampling rate to 200Hz and update the sliding window size to 0.3 seconds with an overlapping of 59 rows of data, which speeds up the dance move classification. In the actual dancing test, we exclude the first 10 rows of the data and use the following 80 rows of data to carry out the classification. This is because the very first 10 rows of data may contain inaccurate information as the dancer needs to lift his hand to the correct position before performing the dance move. The data visualization of 9 dance moves of one dancer in the time domain is shown in Appendix A.

5.1.1.2 Data pre-processing and feature engineering

In the initial design report, we include two methods to extract useful information from the raw data, namely data transformation and feature engineering. After actual testing, we decided to use the second method, which is feature engineering.

The first method is to perform data transformation on the raw data. Firstly, we pass the raw data into a fifth-order butterworth bandpass filter to extract useful information from a particular frequency band. Secondly, the results will be squared to increase the difference in the intensity of different activities for easier differentiation (i.e., smaller values get smaller and larger values get larger, leading to a more obvious gap among different activities). Thirdly, we will apply a Savitzky-Golay filter to smooth the data. Fourthly, the results will be standardized before passing into the machine learning model. The results will be scaled into the range of 0 to 1, which will ensure data in all dimensions have the same scale. This will ensure all features have the same influence on the final classification outcome. The overall flow is summarised as fifth-order butterworth bandpass filter → squaring the results → Savitzky-Golay filter → normalization. Even though this method has a high 10-fold cross-validation accuracy, it is not robust for unseen test data (i.e., test accuracy is low) and the runtime is relatively long. Thus, we do not use this method in the final design.

The second method is to perform feature engineering on the raw data. The features can be classified into three different domains, namely time-domain features, frequency domain features (i.e., after fast Fourier Transform) and time frequency domain features (i.e., after Discrete Wavelet Transform). In the final design, we decide to use the second

method as it takes a shorter time to provide the classification result. We extract three time-domain features (i.e., standard deviation, mean absolute value and variance) and two frequency domain features (i.e., maximum power spectral density and subband power) (Figure 9). Since the runtime is relatively fast, there is no need to carry out dimensionality reduction techniques like Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).

```
# calculate mean absolute of data in a given
# window size of 60 rows of data
def mean_absolute(data, length):
  import math
  import numpy as np
  sf = 200
  window = int(sf * 0.3)
  length = math.floor((length - window) / 1)
  average = [0] * length
  i = 0
  for x in range(length):
    average[x] = abs(np.mean(data[i:i+window]))
    i = i + 1
  return average
```

```
# calculate standard deviation of data in a
# given window size of 60 rows of data
def standard_deviation(data, length):
  import math
  import numpy as np
  sf = 200
  window = int(sf * 0.3)
  length = math.floor((length - window) / 1)
  std = [0] * length
  i = 0
  for x in range(length):
    std[x] = np.std(data[i:i+window])
    i = i + 1
  return std
```

```
# calculate variance of data in a given
# window size of 60 rows of data
def variance(data, length):
  import math
  import numpy as np
  sf = 200
  window = int(sf * 0.3)
  length = math.floor((length - window) / 1)
  var = [0] * length
  i = 0
  for x in range(length):
    var[x] = np.var(data[i:i+window])
    i = i + 1
  return var
```

```
# calculate bandpower of data in a given window size of
# 60 rows of data
def subbandpower(data, length):
  import math
  import numpy as np
  sf = 200
  window = int(sf * 0.3)
  length = math.floor((length - window) / 1)
  sbp = [0] * length
  i = 0
  for x in range(length):
    sbp[x] = bandpower(data[i:i+window], sf, [1, 20], 'welch')
    i = i + 1
  return sbp
```

```
# calculate max power spectral density of data in a given
# window size of 60 rows of data
def max_psd(data, length):
  from scipy import signal
  import numpy as np
  import math
  sf = 200
  win = 0.5 * sf
  window = int(sf * 0.3)
  length = math.floor((length - window) / 1)
  psd_max = [0] * length
  i = 0

  for x in range(length):
    freq, psd = signal.welch(data[i:i+window], sf, nperseg = win)

    # remove low frequency component (f < 1 Hz)
    for j in range(len(freq)):
      if freq[j] < 1:
        psd[j] = 0

    psd_max[x] = abs(np.max(psd))
    i = i + 1

  return psd_max
```

```
def bandpower(data, sf, band, method='welch', window_sec=None, relative=False):
    from scipy.signal import welch
    from scipy.integrate import simps

    band = np.asarray(band)
    low, high = band

    # Compute the modified periodogram (Welch)
    if method == 'welch':
        if window_sec is not None:
            nperseg = window_sec * sf
        else:
            nperseg = (2 / low) * sf

        freqs, psd = welch(data, sf, nperseg=nperseg)

    # Frequency resolution
    freq_res = freqs[1] - freqs[0]

    # Find index of band in frequency vector
    idx_band = np.logical_and(freqs >= low, freqs <= high)

    # Integral approximation of the spectrum using parabola (Simpson's rule)
    bp = simps(psd[idx_band], dx=freq_res)

    if relative:
        bp /= simps(psd, dx=freq_res)
    return bp
```

Figure 9. Feature engineering for dance move classification

5.1.1.3 Potential machine learning models

We have planned to implement three potential machine learning models to classify the dance movements, namely k-nearest neighbours (kNN), support vector machine with linear kernel and multilayer perceptrons (MLP). There are two criteria for us to decide the appropriate machine learning model, namely classification performance and runtime. The kNN will not be a suitable model as the runtime is relatively long. It has to loop through all examples in the dataset to locate the k nearest neighbours. Thus, it is not feasible for kNN to give real-time responses to the user regarding the classification results as it is not user-friendly.

5.1.1.4 Model training, validation and testing

Our group has 5 members and we are all involved in the data collection process. Each of us performs a 3-minute dance for all 9 dance moves and the data collected are included in model training, validation and testing.

At first, after feature engineering, data of 9 dance moves are used to train and validate different machine learning models. We split the data into a training set (i.e., 75%) and a validation set (i.e., 25%). By using a 10-fold cross validation, we fit data from the training set into classifiers and generate a confusion matrix, classification report (i.e., accuracy, precision, recall and f1-score) and runtime analysis respectively. After comparing the classification performance and runtime, we decided to use multilayer perceptrons. The multilayer perceptron contains 1 input layer of 30 nodes, 3 hidden layers of 30 nodes, 15 nodes and 5 nodes respectively, and 1 output layer of 9 nodes. The initial learning rate is set to 0.01, the activation function for input layer and hidden layers are ReLU and the activation function for output layer is softmax. After training the MLP classifiers, we output the weights and bias and rebuild the model on the FPGA. After rebuilding the model on the FPGA, we tested it with unseen data to verify the robustness of the classifier.

In conclusion, using this feature engineering method and MLP as the classifier, the 10-fold cross validation accuracy is 96% with a standard deviation 2% and the testing accuracy is about 98%.

5.1.2 Relative position

This section will introduce the algorithm for relative position detection. In summary, this section introduces a binary classification classifier to differentiate whether the dancer is doing a dance move or switching position and the detailed relative position detection algorithm.

5.1.2.1 Dancing or positioning

After receiving the data, in order to understand the current status of the dancer, we built another classifier to perform a binary classification to differentiate whether the dancer is doing a dance move or switching position. If the classification result is dancing, we carry out dance move classification and if the classification result is positioning, we carry out relative position detection.

We adapt the pipeline from dance move classification and alter the hyperparameters. We use the sliding window size of 0.2 second with an overlapping of 39 rows of data. For the feature engineering, we use three time domain features (i.e., standard deviation, mean absolute value and variance) and two frequency domain features (i.e., maximum power spectral density and subband power) (Figure 10). Since the runtime is relatively fast, there is no need to carry out dimensionality reduction techniques like Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).

```
# calculate standard deviation of data in a
# given window size of 40 rows of data
def standard_deviation_binary(data, length):
  import math
  import numpy as np
  sf = 200
  window = int(sf * 0.2)
  length = math.floor((length - window) / 1)
  std = [0] * length
  i = 0
  for x in range(length):
    std[x] = np.std(data[i:i+window])
    i = i + 1
  return std
```

```
# calculate mean absolute of data in a given
# window size of 40 rows of data
def mean_absolute_binary(data, length):
  import math
  import numpy as np
  sf = 200
  window = int(sf * 0.2)
  length = math.floor((length - window) / 1)
  average = [0] * length
  i = 0
  for x in range(length):
    average[x] = abs(np.mean(data[i:i+window]))
    i = i + 1
  return average
```

```python
# calculate variance of data in a given window
# size of 40 rows of data
def variance_binary(data, length):
  import math
  import numpy as np
  sf = 200
  window = int(sf * 0.2)
  length = math.floor((length - window) / 1)
  var = [0] * length
  i = 0
  for x in range(length):
    var[x] = np.var(data[i:i+window])
    i = i + 1
  return var
```

```python
# calculate bandpower of data in a given window size of
# 40 rows of data
def subbandpower_binary(data, length):
  import math
  import numpy as np
  sf = 200
  window = int(sf * 0.2)
  length = math.floor((length - window) / 1)
  sbp = [0] * length
  i = 0
  for x in range(length):
    sbp[x] = bandpower(data[i:i+window], sf, [1, 20], 'welch')
    i = i + 1
  return sbp
```

```python
def bandpower(data, sf, band, method='welch', window_sec=None, relative=False):
    from scipy.signal import welch
    from scipy.integrate import simps

    band = np.asarray(band)
    low, high = band

    # Compute the modified periodogram (Welch)
    if method == 'welch':
        if window_sec is not None:
            nperseg = window_sec * sf
        else:
            nperseg = (2 / low) * sf

        freqs, psd = welch(data, sf, nperseg=nperseg)

    # Frequency resolution
    freq_res = freqs[1] - freqs[0]

    # Find index of band in frequency vector
    idx_band = np.logical_and(freqs >= low, freqs <= high)

    # Integral approximation of the spectrum using parabola (Simpson's rule)
    bp = simps(psd[idx_band], dx=freq_res)

    if relative:
        bp /= simps(psd, dx=freq_res)
    return bp
```

```python
# calculate max power spectral density of data in a given
# window size of 40 rows of data
def max_psd_binary(data, length):
  from scipy import signal
  import numpy as np
  import math
  sf = 200
  win = 0.5 * sf
  window = int(sf * 0.2)
  length = math.floor((length - window) / 1)
  psd_max = [0] * length
  i = 0

  for x in range(length):
    freq, psd = signal.welch(data[i:i+window], sf, nperseg = win)

    # remove low frequency component (f < 1 Hz)
    for j in range(len(freq)):
      if freq[j] < 1:
        psd[j] = 0

    psd_max[x] = abs(np.max(psd))
    i = i + 1

  return psd_max
```

Figure 10. Feature engineering for dancing-or-positioning

After feature engineering, we split the data into a training set (i.e., 75%) and a validation set (i.e., 25%). By using 10-fold cross-validation, we fit data from the training set into classifiers and generate a confusion matrix, classification report (i.e., accuracy, precision, recall and f1-score) and runtime analysis respectively. Similar to dance move classification, the multilayer perceptron contains 1 input layer of 30 nodes, 3 hidden layers of 30 nodes, 15 nodes and 5 nodes respectively, and 1 output layer of 1 node. The initial learning rate is set to 0.01, the activation function for the input layer and hidden layers are ReLU and the activation function for the output layer is softmax. Lastly, we tested it with unseen data to verify the robustness of the classifier. In conclusion, using this feature engineering method and MLP as the classifier, the 10-fold cross-validation accuracy is 100% with a standard deviation of 0%, and the test accuracy is 100% as well.

5.1.2.2 Relative position detection

The relative position detection algorithm contains two main parts. The first part is to determine whether the dancer is turning left or right and the second part is to use permutation to update dancers' current position.

To determine whether the dancer is turning left or right, we initially tried to use a classifier to perform binary classification. However, even though the 10-fold cross-validation accuracy is 100%, the model is not robust for unseen test data. Thus, in the final design, we use thresholding instead of using a machine learning method. After all data are received, we use the last 5 y-axis gyroscope values to determine whether the dancer is turning left or right. If the last 5 y-axis gyroscope values are negative, the dancer is turning right and if the last 5 y-axis gyroscope values are positive, the dancer is turning left (Figure 11). This is because the turning direction is different, leading to a different sign on the y-axis gyroscope reading.

```
def directioning(data):
  # check the last 5 datapoints to differentiate left or right
  count1 = 0
  count2 = 1
  length = len(data)
  test = data[length-4: length, 4]
  for x in test:
    if x < 0:
      count1 = count1 + 1 # right
    elif x > 0:
      count2 = count2 + 1 # left

  if count1 > count2:
    return False # right
  elif count2 > count1:
    return True # left
```

Figure 11. Function to determine the turning direction

After we know the turning direction of the dancers, we can permute all possibilities and determine the current positions of the dancers based on their previous positions (Figure 12). For example, given that the previous position is 123, dancer 1 turns right, dancer 2 turns left and dancer 3 turns left, the position will be updated to 231 and this is the only answer. In summary, using this method yields a 100% testing accuracy.

```
if (len1 == 0 and len2 == 0 and len3 == 0): # all three do not move
  pass
elif(len1 == 0): # dancer1 does not move
  temp = position2
  position2 = position3
  position3 = temp
elif (len2 == 0): # dancer2 does not move
  temp = position1
  position1 = position3
  position3 = temp
elif (len3 == 0): # dancer3 does not move
  temp = position1
  position1 = position2
  position2 = temp
else: # all three moves
  direction1 = directioning(dancer1)
  direction2 = directioning(dancer2)
  direction3 = directioning(dancer3)
  if (position1 == left and position2 == middle and position3 == right): # 123
    if (direction2 == True): # dancer1 to right, dancer2 to left, dancer3 to left --> 231
      position1 = right
      position2 = left
      position3 = middle
    elif (direction2 == False): # dancer1 to right, dancer2 to right, dancer3 to left --> 312
      position3 = left
      position2 = right
      position1 = middle
```

```
elif (position1 == left and position2 == right and position3 == middle): # 132
  if (direction3 == True): # dancer1 to right, dancer2 to left, dancer3 to left --> 321
    position1 = right
    position2 = middle
    position3 = left
  elif (direction3 == False): # dancer1 to right, dancer2 to left, dancer3 to right --> 213
    position3 = right
    position2 = left
    position1 = middle
elif (position1 == middle and position2 == left and position3 == right): # 213
  if (direction1 == False): # dancer1 to right, dancer2 to right, dancer3 to left --> 321
    position1 = right
    position2 = middle
    position3 = left
  elif (direction1 == True): # dancer1 to left, dancer2 to right, dancer3 to left --> 132
    position3 = middle
    position2 = right
    position1 = left
elif (position1 == middle and position2 == right and position3 == left): # 312
  if (direction1 == False): # dancer1 to right, dancer2 to left, dancer3 to right --> 231
    position1 = right
    position2 = left
    position3 = middle
  elif (direction1 == True): # dancer1 to left, dancer2 to left, dancer3 to right --> 123
    position3 = right
    position2 = middle
    position1 = left
```

```
elif (position1 == right and position2 == left and position3 == middle): # 231
  if (direction3 == True): # dancer1 to left, dancer2 to right, dancer3 to left --> 312
    position1 = middle
    position2 = right
    position3 = left
  elif (direction3 == False): # dancer1 to left, dancer2 to right, dancer3 to right --> 123
    position3 = right
    position2 = middle
    position1 = left
elif (position1 == right and position2 == middle and position3 == left): # 321
  if (direction2 == False): # dancer1 to left, dancer2 to right, dancer3 to right --> 132
    position1 = left
    position2 = right
    position3 = middle
  elif (direction2 == True): # dancer1 to left, dancer2 to left, dancer3 to right --> 213
    position3 = right
    position2 = left
    position1 = middle
```

Figure 12. Function to update current positions

Section 6 Project Management Plan [All members]

Week	Task to do	Members involved
4	Design Report	All
	Solder Work	Yaoyue
	Try and error for several data processing techniques and machine learning models on an online dataset	Zihao
	Setup Vivado Design Suite and Vitis HLS for the Ultra96 v2	Marcus
	Establish Bluetooth connection between Beetle and laptop	Tammy
	Establish Client - Eval Server connection using TCP/IP	Manuel
5	Try and error for several data processing techniques and machine learning models on an online dataset	Zihao
	Send packets through Bluetooth	Tammy
	Set up a cryptography standard for all Clients and Servers	Manuel
	Connect the IP developed in HLS to FPGA in Vivado	Marcus

	Design Suite	
	Get all hardware sensors to connect with Beetles	Yaoyue
6	Individual progress checkpoint	All
	Implement an appropriate neural network model from scratch using python	Zihao
	Design wearable	Yaoyue
	Research and Implement SSH Tunneling for Laptop - Ultra96 connection	Manuel
	Improve reliability of connections	Tammy
	Program FPGA with PYNQ and attempt a roundtrip data transfer between FPGA and CPU	Marcus
Recess week	Implement another machine learning model from scratch using python	Zihao
	Research and Implement NTP synchronization	Manuel
	Research methods on data-preprocessing and implement it if necessary	Yaoyue
	Ensure concurrency and security	Tammy
	Implement the neural network model in C/C++ for the HLS	Marcus
7	Individual subcomponent test and video recording of individual subcomponent	All
	Data collection (5 people with 8 dance movements)	All
	Integration of individual subcomponents to the final product	All
8	Integration of individual subcomponents to the final	All

	product	
	Retrain the machine learning model with the actual dataset for the first three dance moves	Zihao
9 and 10	First system checkpoint (1 dancer, first 3 moves, no positions)	All
	Complete the positioning algorithm	All
11	Second system checkpoint (3 dancers, first 3 moves, positions)	All
	Peer review	All
	Retrain the machine learning model with the actual dataset for all 8 dance moves	Zihao
12	Ensure the whole system is working	All
13	Final evaluation (3 dancers, 9 moves, positions)	All
	Final design report, documentation and code submission	All

Section 7 Societal and Ethical Impact [All members]

Activity detectors can be used as healthcare-monitoring devices. In hospitals, healthcare-monitoring devices are commonly used to monitor the health status of ill patients. They can give immediate alerts to changes in the patients' status. Home monitoring applications for chronic and elderly patients can be used to collect periodic or continuous data and be uploaded to a physician and can allow long-term care and trend analysis.

Moreover, activity detectors can also be used in artificial intelligence (AI) robotic design. For example, in the realm of domestic service robots such as smart furniture, if the robots can detect human activities, they can perform actions to further assist individuals to accomplish tasks. In the case that one opens the pill container and the robot is able

to detect this activity, it can serve the water to the host immediately. This will increase the well-being of households.

Such activity detector systems have raised ethical concerns such as breaches in users' privacy. While cameras, microphones and navigation systems like GPS are commonly perceived as privacy-sensitive and require explicit user permission in current mobile operating systems, many inconspicuous sensors such as accelerometers, gyroscopes and barometers are less well-understood in terms of their privacy implications, and also less protected[6]. Highly personal information may be inferred from the data, breaching the user's privacy.

In order to address such concerns, we can use some safety management software such as safeguard apps (i.e., standalone / in-built in operating systems), which can give users control over what apps can siphon sensor data from their devices or what apps would be allowed to do with the sensor data. Moreover, we can create awareness in the general public.

References

Base64 — Base16, Base32, Base64, Base85 Data Encodings — Python 3.9.7 documentation. (n.d.). Python.Org. Retrieved September 4, 2021, from https://docs.python.org/3/library/base64.html

Crypto.Cipher package — PyCryptodome 3.9.9 documentation. (n.d.). Python.Org. Retrieved September 4, 2021, from https://pycryptodome.readthedocs.io/en/latest/src/cipher/cipher.html

del Toro, S. F., Santos-Cuadros, S., Olmeda, E., Álvarez-Caldas, C., Díaz, V., & San Román, J. L. (2019). Is the Use of a Low-Cost sEMG Sensor Valid to Measure Muscle Fatigue? *Sensors (Basel, Switzerland)*, *19*(14), 3204. https://doi.org/10.3390/s19143204

GeeksforGeeks. (2021, January 19). Network Time Protocol (NTP). https://www.geeksforgeeks.org/network-time-protocol-ntp/

ntplib. (2021, May 28). PyPI. https://pypi.org/project/ntplib/

[6]Xu, Z., Zhu, S.: SemaDroid: a privacy-aware sensor management framework for smartphones. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, pp. 61–72. ACM Press (2015). https://dl.acm.org/doi/abs/10.1145/2699026.2699114?casa_token=kGliBr6SPAgAAAAA:HkicW9FumzDSpjKtsSnEr2rSx7Eeyj5c0ssTa73rjsRE7_yihoFh6SmiKGUvRXjZku65PFvRwGWx

Socket — Low-level networking interface — Python v3.0.1 documentation. (n.d.). Python.Org. Retrieved September 4, 2021, from https://docs.python.org/3.0/library/socket.html

Sridhar, J. (2018, February 8). Using AES for Encryption and Decryption in Python Pycrypto [Blog]. Novixys Software Dev Blog. https://www.novixys.com/blog/using-aes-encryption-decryption-python-pycrypto/#3_Initialization_Vector

Weiss, G. M., Yoneda, K., & Hayajneh, T. (2019). Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living. *IEEE Access*, *7*, 133190–133202. https://doi.org/10.1109/ACCESS.2019.2940729

Wikipedia contributors. (2021, July 22). Block cipher mode of operation. Wikipedia. https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_block_chaining_(CBC)

Appendix A [Zihao]
Data visualisation of one dancer in the time domain