#define arraysize 200

/\*\*\*\*\*\*Variables to read the current values from 20 milli second samples and get the I peaks for each 20 milliseconds.\*\*\*\*\*\*\*/

int count = 0;

int currentValue[200];

int arraySize = 200;

int currentPeakIndex;

float currentPeak;

int peaks[arraysize];

int Min\_JumpMagnitude;

int IndexofC1;

int Jump\_Difference;

int NumberOfDevices = 4 ; // variable to store number of Devices

/\*\*\*\*\*\*\* C1's variable needed for identification \*\*\*\*\*\*\*/

int True\_flag = 0;

int prevValue = 0;

int current\_Value = 0;

/\*\*\*\*\*\*\* Structure For Learning\*\*\*\*\*\*\*\*/

struct TemplateLibrary

{

int JumpMagnitude ;

int ShapeMagnitude;

int ShapeIndex;

int ToleranceC2;

float AvgSteadyState;

float SettlingTime;

int flag ;

} ;

/\*\*\*\*\*\*\* Structure ForRuntime\*\*\*\*\*\*\*\*\*/

struct Runtime

{

int previousSteadyState;

int JumpMagnitude;

int ShapeMagnitude;

int ShapeIndex;

float AvgSteadyState;

int SettlingTime;

int indexofC1;

} ;

/\*\*\*\*\*\* Object Declaration for the above Structures\*\*\*\*\*\*/

TemplateLibrary device[10];

Runtime D;

/\*\*\*\*\*\*\* Function to find the current peak \*\*\*\*\*\*\*/

int findCurrentPeak ( int currentArray[200] )

{

int currentPeakIndex = 0;

for ( int i = 0; i < arraySize; i++ )

{

if ( currentArray[i] > currentArray[currentPeakIndex] )

{

currentPeakIndex = i;

}

}

return currentPeakIndex;

}

/\*\*\*\*Function which initially sets the values for the jump magnitude based on the trails of readings\*\*\*\*\*/

void setValues()

{

device[0].JumpMagnitude = 522 - 514;

device[1].JumpMagnitude = 521 - 514;

device[2].JumpMagnitude = 873 - 514;

device[3].JumpMagnitude = 617 - 514;

for(int i = 0; i<NumberOfDevices ; i++)

{

device[i].flag = 0;

}

}

/\*\*\*\*\* Function to find the minimum of the jump magnitude among the available devices\*\*\*\*\*/

void Find\_Min\_JumpMagnitude()

{

int MinimumValue = device[0].JumpMagnitude;

for(int i = 0; i<3 ; i++)

{

if(device[i].JumpMagnitude<MinimumValue)

{

MinimumValue = device[i].JumpMagnitude;

}

}

Min\_JumpMagnitude = MinimumValue;

/\*Serial.println("min:");

Serial.println(Min\_JumpMagnitude);

\*/

}

/\*\*\*\*\*\* Function to get the Ipeak among the 20 Milliseconds reading\*\*\*\*\*\*/

int Get\_Ipeak()

{

for ( int i = 0; i < 200; i++ )

{

currentValue[i] = analogRead(A1);

}

currentPeakIndex = findCurrentPeak ( currentValue );

return(currentValue[currentPeakIndex]);

}

/\*\*\*\*\*\*\* Function of C1 to find the jump value \*\*\*\*\*\*/

void C1\_Jump()

{

D.previousSteadyState = prevValue;

prevValue = current\_Value;

current\_Value = Get\_Ipeak(); //If true finding the first true value

D.JumpMagnitude=current\_Value - D.previousSteadyState; //Finding the Magnitude of the first array value

if(D.JumpMagnitude > Min\_JumpMagnitude)

{

peaks[0] = D.previousSteadyState; //Initialising the zeroth , abrupt and first array value

peaks[1] = prevValue; //Taking random value into the main array

peaks[2] = current\_Value; //Taking my n+1 th term as my threshold

for(int j=3 ; j<arraysize ; j++)

{

peaks[j]=Get\_Ipeak(); //Getting all the array values till n=200

}

C1\_Identification();

displayVal();

D.indexofC1 = 2;

}

}

/\*\*\*\*\* Function to identify which device is on through the C1 jump value and reference table comparison \*\*\*\*\*\*/

void C1\_Identification()

{

for(int i=0; i<NumberOfDevices ;i++)

{

if((D.JumpMagnitude <= ( device[i].JumpMagnitude + 4) ) && ((D.JumpMagnitude >= (device[i].JumpMagnitude - 4))))

{

device[i].flag = 1;

}

else

{

device[i].flag = 0;

}

}

}

/\*\*\*\*\* Function to display which device is on based on the flag values set\*\*\*\*\*\*/

void displayVal()

{

Serial.println(D.previousSteadyState);

Serial.println(prevValue);

Serial.println(current\_Value);

Serial.println(D.JumpMagnitude);

for(int i=0; i<NumberOfDevices; i++)

{

if(device[i].flag == 1)

{

Serial.println("Device: ");

Serial.print(i);

Serial.print(" is ON");

Serial.println();

}

else if(device[i].flag == 0)

{

Serial.println("Device: ");

Serial.print(i);

Serial.print(" is OFF");

Serial.println();

}

}

for(int i = 0; i<NumberOfDevices ; i++) // again setting the flags to zero after identifing the device and printing

{

device[i].flag = 0;

}

}

void setup()

{

Serial.begin(9600);

analogReference('EXTERNAL');

pinMode(A1,INPUT);

setValues(); // this function is called only once from the setup to initialize the values

Find\_Min\_JumpMagnitude();

}

void loop()

{

// during the initial condition both my prevvalue and current value are read as my analog vales to avoid false triggering

while(count == 0)

{ prevValue = Get\_Ipeak();

count =1;

}

current\_Value = Get\_Ipeak();

D.JumpMagnitude = current\_Value - prevValue; // calculating the difference i.e. the jump values

if(D.JumpMagnitude > 0 )

{

C1\_Jump();

}

prevValue = current\_Value;

}