Code :-

/\* Edge Impulse ingestion SDK

 \* Copyright (c) 2022 EdgeImpulse Inc.

 \*

 \* Licensed under the Apache License, Version 2.0 (the "License");

 \* you may not use this file except in compliance with the License.

 \* You may obtain a copy of the License at

 \* http://www.apache.org/licenses/LICENSE-2.0

 \*

 \* Unless required by applicable law or agreed to in writing, software

 \* distributed under the License is distributed on an "AS IS" BASIS,

 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

 \* See the License for the specific language governing permissions and

 \* limitations under the License.

 \*

 \*/

// If your target is limited in memory remove this macro to save 10K RAM

#define EIDSP\_QUANTIZE\_FILTERBANK   0

/\*\*

 \* Define the number of slices per model window. E.g. a model window of 1000 ms

 \* with slices per model window set to 4. Results in a slice size of 250 ms.

 \* For more info: https://docs.edgeimpulse.com/docs/continuous-audio-sampling

 \*/

#define EI\_CLASSIFIER\_SLICES\_PER\_MODEL\_WINDOW 1

// this is the h0w many times you can speak in one second.

/\*

 \*\* NOTE: If you run into TFLite arena allocation issue.

 \*\*

 \*\* This may be due to may dynamic memory fragmentation.

 \*\* Try defining "-DEI\_CLASSIFIER\_ALLOCATION\_STATIC" in boards.local.txt (create

 \*\* if it doesn't exist) and copy this file to

 \*\* `<ARDUINO\_CORE\_INSTALL\_PATH>/arduino/hardware/<mbed\_core>/<core\_version>/`.

 \*\*

 \*\* See

 \*\* (https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-the-installed-cores-located-)

 \*\* to find where Arduino installs cores on your machine.

 \*\*

 \*\* If the problem persists then there's not enough memory for this model and application.

 \*/

/\* Includes ---------------------------------------------------------------- \*/

#include <PDM.h>

#include <speech\_recognition\_2\_inferencing.h>

#include <Arduino\_LPS22HB.h>  // Library for Barometer

#include <Arduino\_HS300x.h>   // Library for Temperarture and Humidity

#include <LiquidCrystal.h>    // Library for Screen

/\*\* Audio buffers, pointers and selectors \*/

typedef struct {

    signed short \*buffers[2];

    unsigned char buf\_select;

    unsigned char buf\_ready;

    unsigned int buf\_count;

    unsigned int n\_samples;

} inference\_t;

static inference\_t inference;

static bool record\_ready = false;

static signed short \*sampleBuffer;

static bool debug\_nn = false; // Set this to true to see e.g. features generated from the raw signal

static int print\_results = -(EI\_CLASSIFIER\_SLICES\_PER\_MODEL\_WINDOW);

//...............................................................

int led\_red = LEDR;      // LED Red

int led\_blue = LEDB;     // LED Blue

int led\_green = LEDG;    // LED Green

int relay1 = 3;

int relay2 = 2;

// initialize the library by associating any needed LCD interface pin

// with the arduino pin number it is connected to

const int rs = 8, en = 9, d4 = 4, d5 = 5, d6 = 6, d7 = 7;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

//.................................................................

/\*\*

 \* @brief      Arduino setup function

 \*/

void setup()

{

  // LED output

  //pinMode(led\_red,OUTPUT);

  //pinMode(led\_blue,OUTPUT);

  //pinMode(led\_green,OUTPUT);

    //digitalWrite(led\_red,HIGH);

    //digitalWrite(led\_blue,HIGH);

    //digitalWrite(led\_green,HIGH);

    // set up the LCD's number of columns and rows:..............................

    lcd.begin(16, 2); // LCD column and row

    // put your setup code here, to run once:

    Serial.begin(115200);

    // comment out the below line to cancel the wait for USB connection (needed for native USB)

    while (!Serial);

    Serial.println("Edge Impulse Inferencing Demo");

    // Intialising the Libraries................................................................................

    HS300x.begin();     // Temperature sensor begin

    BARO.begin();       // Barometer sensor begin

    //...............................................................................

    pinMode(relay1,OUTPUT);

    pinMode(relay2,OUTPUT);

    digitalWrite(relay1, HIGH);

    digitalWrite(relay2, HIGH);

    //.........................................................................................

    // summary of inferencing settings (from model\_metadata.h)

    ei\_printf("Inferencing settings:\n");

    ei\_printf("\tInterval: %.2f ms.\n", (float)EI\_CLASSIFIER\_INTERVAL\_MS);

    ei\_printf("\tFrame size: %d\n", EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE);

    ei\_printf("\tSample length: %d ms.\n", EI\_CLASSIFIER\_RAW\_SAMPLE\_COUNT / 16);

    ei\_printf("\tNo. of classes: %d\n", sizeof(ei\_classifier\_inferencing\_categories) /

                                            sizeof(ei\_classifier\_inferencing\_categories[0]));

    run\_classifier\_init();

    if (microphone\_inference\_start(EI\_CLASSIFIER\_SLICE\_SIZE) == false) {

        ei\_printf("ERR: Could not allocate audio buffer (size %d), this could be due to the window length of your model\r\n", EI\_CLASSIFIER\_RAW\_SAMPLE\_COUNT);

        return;

    }

}

/\*\*

 \* @brief      Arduino main function. Runs the inferencing loop.

 \*/

 static float thresh = 0.9;

void loop()

{

    bool m = microphone\_inference\_record();

    if (!m) {

        ei\_printf("ERR: Failed to record audio...\n");

        return;

    }

    signal\_t signal;

    signal.total\_length = EI\_CLASSIFIER\_SLICE\_SIZE;

    signal.get\_data = &microphone\_audio\_signal\_get\_data;

    ei\_impulse\_result\_t result = {0};

    EI\_IMPULSE\_ERROR r = run\_classifier\_continuous(&signal, &result, debug\_nn);

    if (r != EI\_IMPULSE\_OK) {

        ei\_printf("ERR: Failed to run classifier (%d)\n", r);

        return;

    }

    //.....................................................................................

    if(result.classification[0].value > thresh){ // .............// "BUJHAO"

      //digitalWrite(led\_red,LOW);   //  BLE Sense is ON when the pin  is LOW

      digitalWrite(relay1,HIGH);

      lcd.print("BUJHAO  ");

      delay(500);

      lcd.clear();

    }else if(result.classification[1].value > thresh){ // .......// "JALAO"

      //digitalWrite(led\_red,LOW);   //  BLE Sense is ON when the pin  is LOW

      //digitalWrite(led\_blue,LOW);

      digitalWrite(relay1,LOW);

      lcd.print("JALAO  ");

      delay(500);

      lcd.clear();

    }else if(result.classification[2].value > thresh){ // .......// "LIGHT OFF"

      //digitalWrite(led\_red,LOW);   //  BLE Sense is ON when the pin  is LOW

      //digitalWrite(led\_green,LOW);

      digitalWrite(relay2,HIGH);

      lcd.print("LIGHT OFF  ");

      delay(500);

      lcd.clear();

    }else if(result.classification[3].value > thresh){ // .......// "LIGHT ON"

      //digitalWrite(led\_red,LOW);   //  BLE Sense is ON when the pin  is LOW

      //digitalWrite(led\_blue,LOW);

      //digitalWrite(led\_green,LOW);

      digitalWrite(relay2,LOW);

      lcd.print("LIGHT ON  ");

      delay(500);

      lcd.clear();

    }else if(result.classification[4].value > thresh){ // .......// "PRESSURE"

      //digitalWrite(led\_blue,LOW);

      float pressure = BARO.readPressure();

      lcd.setCursor(0,0);

      lcd.print("Pressure = ");

      lcd.print(pressure);

      lcd.print("kPa");

      delay(700);

      lcd.clear();

    }else if(result.classification[5].value > thresh){ // .......// "TEMPERATURE"

      //digitalWrite(led\_green,LOW);

      float humidity = HS300x.readHumidity();

      float temperature = HS300x.readTemperature();

      lcd.setCursor(0,0);

      lcd.print("Humidity =");

      lcd.print(humidity);

      lcd.print(" %");

      lcd.setCursor(0,1);

      lcd.print("Temp =");

      lcd.print(temperature);

      lcd.print(" C");

      delay(700);

      lcd.clear();

    }

    // // "\_NOISE"

    // if(result.classification[6].value > 0.9){

    //   digitalWrite(led\_red,LOW);   //  BLE Sense is ON when the pin  is LOW

    // }else{

    //   digitalWrite(led\_red,HIGH);  //  BLE Sense is OFF when the pin is HIGH

    // }

    // // "\_UNKNOWN"

    // if(result.classification[7].value > 0.9){

    //   digitalWrite(led\_red,HIGH);

    // }else{

    //   digitalWrite(led\_red,HIGH);

    // }

    //..........................................................................................

    if (++print\_results >= (EI\_CLASSIFIER\_SLICES\_PER\_MODEL\_WINDOW)) {

        // print the predictions

        ei\_printf("Predictions ");

        ei\_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",

            result.timing.dsp, result.timing.classification, result.timing.anomaly);

        ei\_printf(": \n");

        for (size\_t ix = 0; ix < EI\_CLASSIFIER\_LABEL\_COUNT; ix++) {

            ei\_printf("    %s: %.5f\n", result.classification[ix].label,

                      result.classification[ix].value);

        }

#if EI\_CLASSIFIER\_HAS\_ANOMALY == 1

        ei\_printf("    anomaly score: %.3f\n", result.anomaly);

#endif

        print\_results = 0;

    }

}

/\*\*

 \* @brief      PDM buffer full callback

 \*             Get data and call audio thread callback

 \*/

static void pdm\_data\_ready\_inference\_callback(void)

{

    int bytesAvailable = PDM.available();

    // read into the sample buffer

    int bytesRead = PDM.read((char \*)&sampleBuffer[0], bytesAvailable);

    if (record\_ready == true) {

        for (int i = 0; i<bytesRead>> 1; i++) {

            inference.buffers[inference.buf\_select][inference.buf\_count++] = sampleBuffer[i];

            if (inference.buf\_count >= inference.n\_samples) {

                inference.buf\_select ^= 1;

                inference.buf\_count = 0;

                inference.buf\_ready = 1;

            }

        }

    }

}

/\*\*

 \* @brief      Init inferencing struct and setup/start PDM

 \*

 \* @param[in]  n\_samples  The n samples

 \*

 \* @return     { description\_of\_the\_return\_value }

 \*/

static bool microphone\_inference\_start(uint32\_t n\_samples)

{

    inference.buffers[0] = (signed short \*)malloc(n\_samples \* sizeof(signed short));

    if (inference.buffers[0] == NULL) {

        return false;

    }

    inference.buffers[1] = (signed short \*)malloc(n\_samples \* sizeof(signed short));

    if (inference.buffers[1] == NULL) {

        free(inference.buffers[0]);

        return false;

    }

    sampleBuffer = (signed short \*)malloc((n\_samples >> 1) \* sizeof(signed short));

    if (sampleBuffer == NULL) {

        free(inference.buffers[0]);

        free(inference.buffers[1]);

        return false;

    }

    inference.buf\_select = 0;

    inference.buf\_count = 0;

    inference.n\_samples = n\_samples;

    inference.buf\_ready = 0;

    // configure the data receive callback

    PDM.onReceive(&pdm\_data\_ready\_inference\_callback);

    PDM.setBufferSize((n\_samples >> 1) \* sizeof(int16\_t));

    // initialize PDM with:

    // - one channel (mono mode)

    // - a 16 kHz sample rate

    if (!PDM.begin(1, EI\_CLASSIFIER\_FREQUENCY)) {

        ei\_printf("Failed to start PDM!");

    }

    // set the gain, defaults to 20

    PDM.setGain(127);

    record\_ready = true;

    return true;

}

/\*\*

 \* @brief      Wait on new data

 \*

 \* @return     True when finished

 \*/

static bool microphone\_inference\_record(void)

{

    bool ret = true;

    if (inference.buf\_ready == 1) {

        ei\_printf(

            "Error sample buffer overrun. Decrease the number of slices per model window "

            "(EI\_CLASSIFIER\_SLICES\_PER\_MODEL\_WINDOW)\n");

        ret = false;

    }

    while (inference.buf\_ready == 0) {

        delay(1);

    }

    inference.buf\_ready = 0;

    return ret;

}

/\*\*

 \* Get raw audio signal data

 \*/

static int microphone\_audio\_signal\_get\_data(size\_t offset, size\_t length, float \*out\_ptr)

{

    numpy::int16\_to\_float(&inference.buffers[inference.buf\_select ^ 1][offset], out\_ptr, length);

    return 0;

}

/\*\*

 \* @brief      Stop PDM and release buffers

 \*/

static void microphone\_inference\_end(void)

{

    PDM.end();

    free(inference.buffers[0]);

    free(inference.buffers[1]);

    free(sampleBuffer);

}

#if !defined(EI\_CLASSIFIER\_SENSOR) || EI\_CLASSIFIER\_SENSOR != EI\_CLASSIFIER\_SENSOR\_MICROPHONE

#error "Invalid model for current sensor."

#endif