VSCODE_PRINT_SCRIPT_TAGS

Folder Arduino

3 printable files

Arduino\DAQ\DAQ.ino
Arduino\RTCInit\RTCInit.ino
Arduino\RTCInit\arduino_secrets.h

Arduino\DAQ\DAQ.ino

```
/*
 * Arduino DAQ Code
 *
 * Created:
 * 7/26/2023 by Siem Yonas
 * Last Modified:
 * 8/7/2023 by Siem Yonas
 */

#include <ArduinoBLE.h>
#include <SPI.h>
#include <SD.h>
#include "Arduino.h"
#include "uRTCLib.h"
#include <UnixTime.h>

#define USE_ARDUINO_INTERRUPTS false
#include <PulseSensorPlayground.h>

// Global vars and constants

// BLE
BLEService hrvService("180F"); // BLE HRV service

BLECharacteristic hrvChar("2A19", BLERead | BLENotify, 8); // Bluetooth Low Energy Characteristic to send HRV records
BLEByteCharacteristic errorChar("2A1A", BLERead | BLENotify); // Bluetooth Low Energy Characteristic to send error codes to device application
BLECharacteristic requestChar("2A1B", BLERead | BLEWriteWithoutResponse, 4); // Bluetooth Low Energy Characteristic to receive data requests from device application

bool transferInProgress; // Whether a data transfer is currently in progress
unsigned long lastTransferTime; // Millisecond of last transfer
const unsigned long TRANSFER_TIMEOUT = 250; // Number of milliseconds between each data transfer

File transferFile; // Current file used in data transfer
uint16_t transferDate[3]; // Current transfer date
char transferFilename[13]; // Current transfer filename
uint32_t transferFilePosition; // Current read position in the current transfer file

uint8_t* last_val = 0;

BLEDevice connectedDevice; // Current connected device central device, which should be the device application

// SD
const int CHIP_SELECT = 10; // Digital pin for the SD card's chip select

// RTC
uRTCLib rtc(0x68); // uRTCLib library object
UnixTime stamp(0); // Unix timestamp converter

// Pulse Sensor
const int PULSE_INPUT = A0; // Analog pin for pulse sensor
const int THRESHOLD = 550; // Threshold for pulse sensor signal for the PulseSensorPlayground library
PulseSensorPlayground pulseSensor; // PulseSensorPlayground library object

// HRV Calculation
float rmssd; // The calculated RMSSD in ms (the "HRV Metric")
float rrDiffSquaredTotal; // Intermediate value for calculating RMSSD (numerator under the radical)
int numRRDetected; // Counter for the number of RR intervals found in the measurement period

const int MINUTES_IN_WINDOW = 1; // The number of minutes to be used in a measurement window
const int BEATS_TIL_MEASURE = 5; // Number of heartbeats to detect before measurement
int beatsRemaining; // Current number of heartbeats remaining before starting a measurement

unsigned long hrvStartTime; // Millisecond where the HRV measurement began
unsigned long lastPeakTime; // Millisecond where the last peak was found
int lastRRInterval; // Duration of the last RR interval in milliseconds.

// Helper Functions

// Resets all HRV variables to their initial values.
void resetHrv() {
  // Reset beats remaining to its starting value.
  beatsRemaining = BEATS_TIL_MEASURE;

  // Zero out all integer values
  lastRRInterval = 0;
  lastPeakTime = 0;
  numRRDetected = 0;
  hrvStartTime = 0;

  // Set floats to -1
  rmssd = -1.0;
  rrDiffSquaredTotal = -1.0;
}

// Gets Unix Timestamp from the RTC module
uint32_t getUnixEpochTime() {
  // Set the timestamp converter's current date in UTC
  rtc.refresh();
  stamp.setDateTime(2000 + rtc.year(), rtc.month(), rtc.day(), rtc.hour(), rtc.minute(), rtc.second());
  return stamp.getUnix(); // Return the corresponsing unix timestamp
}

// Sets the filename character array to the current date data file.
void setFilename(char* filename) {
  rtc.refresh(); // Update the RTC
  setFilename(filename, 2000 + rtc.year(), rtc.month(), rtc.day()); // Pass RTC values into the general setFilename function
}

// Sets the filename character array to the selected date's data file.
void setFilename(char* filename, uint16_t year, uint8_t month, uint8_t day) {
  snprintf(filename, 13, "%04d%02d%02d.txt", year, month, day); // Format year, month, and day into the HRV record format (see requirement 3.3.2.2.3)
}

// Sends HRV records to the device application. Returns 1 on failure to open, and 0 for no errors.
int sendRecords() {

  if (!connectedDevice.connected()){
    transferInProgress = false;
    transferFile.close();
    Serial.println("Disconnect!");
    return 1;
  }

  uint32_t now = getUnixEpochTime();

  // If the transferFile is not open, open the next available day
  while (!transferFile){
    // Get the next date time stamp

    setFilename(transferFilename, transferDate[0], transferDate[1], transferDate[2]);

    stamp.setDateTime(transferDate[0], transferDate[1], transferDate[2], 0, 0, 0);
    stamp.getDateTime(stamp.getUnix() + 86400);

    transferDate[0] = stamp.year;
    transferDate[1] = stamp.month;
    transferDate[2] = stamp.day;

    uint32_t next_day = stamp.getUnix();

    if (SD.exists(transferFilename)){
      transferFilePosition = 0;
      transferFile = SD.open(transferFilename);
    }

    else {
      // If no more days exist, end file transfer and return.
      if (next_day > now) {
        transferInProgress = false;
        Serial.println("Data transfer Done!");
        return 1;
      }
    }
  }

  // Read the next record from the transferFile
  transferFile.seek(transferFilePosition);
  char record[19];

  transferFile.read(record, 18);

  // Parse record
  uint32_t unix;
  float transferRmssd;
  sscanf(record, "%d %f", &unix, &transferRmssd);
  transferRmssd = atof(record+11);

  // Pack record into HRV format and write to hrvChar
  uint8_t hrvValue[8];
  *((uint32_t*) hrvValue) = unix;
  *((float*) (hrvValue+4)) = transferRmssd;

  hrvChar.writeValue(hrvValue, 8);
  lastTransferTime = millis();

  // Update transferFilePosition
  transferFilePosition = transferFile.position();

  // Check if EOF, close file if so
  if (transferFilePosition+10 >= transferFile.size()) {
    transferFile.close();
  }

  return 0;
}

// Stores current HRV record to the SD card.
void storeRecord() {
  // Close transferFile, since only one file can be open at a time from SD
  if (transferInProgress)
    transferFile.close();

  // Get the current Unix timestamp
  uint32_t unix = getUnixEpochTime();

  // Get the current day data filename
  char storageFilename[13];
  setFilename(storageFilename);

  // Open current day data file for writing
  File storageFile = SD.open(storageFilename, FILE_WRITE);

  // Format a record
  char record[19];
  snprintf(record, 19, "%10d %6.2f\n", unix, rmssd);

  // Write record to file and close the file
  storageFile.write(record, 18);

  storageFile.close();

  // Pack record into HRV format and write to hrvChar
  uint8_t hrvValue[8];
  *((uint32_t*) hrvValue) = unix;
  *((float*) (hrvValue+4)) = rmssd;

  hrvChar.writeValue(hrvValue, 8);

  lastTransferTime = millis();

  // Reopen transferFile
  if (transferInProgress)
    transferFile = SD.open(transferFilename);
}

// Handles updating the HRV variables on each heartbeat.
void updateHrv() {

  // Get the currentMillisecond as the current peak time.
  unsigned long currentPeakTime = millis();

  // If there was a peak before this
  if (lastPeakTime != 0){
    int currentRRInterval = pulseSensor.getInterBeatIntervalMs(); // Get the interbeat interval between the two peaks
    numRRDetected++; // Increment the number of RR Intervals seen

    // If there was an RR interval before this
    if (lastRRInterval != 0) {
      // Get the squared difference of the RR Intervals and add this to rrDiffSquaredTotal
      float rrDiff = currentRRInterval - lastRRInterval;
      rrDiffSquaredTotal += rrDiff * rrDiff;
    }
    lastRRInterval = currentRRInterval; // Update last RR Interval.
  }

  lastPeakTime = currentPeakTime; // Update last peak time.

}

// Parses requestChar value and sets transferFile
void filenameFromRequestChar() {
  // Read the request characteristic
  const uint8_t* rawRequest = requestChar.value();

  // Variables to extract from the characteristic
  uint16_t year = *(uint16_t *) rawRequest;
  uint8_t month = *(uint8_t *) (rawRequest+2);
  uint8_t day = *(uint8_t *) (rawRequest+3);

  transferDate[0] = year;
  transferDate[1] = month;
  transferDate[2] = day;

  // setFilename to the requested date
  setFilename(transferFilename, year, month, day);
}

// Sets up BLE, RTC, SD, Pulse Sensor, and HRV values.
void setup() {

  // For Debugging, use the Serial
  Serial.begin(115200);
  //while (!Serial);

  // BLE

  // Start BLE library
  if (!BLE.begin()) {
    Serial.println("starting BLE failed!");

    // Hang Execution if BLE fails to start
    while(1);
  }

  Serial.println("BLE began!");

  // Set the BLE name to Tranquil+
  BLE.setLocalName("Traquil+");

  // Add all BLE services and characteristics
  BLE.setAdvertisedService(hrvService);
  hrvService.addCharacteristic(hrvChar);
  hrvService.addCharacteristic(errorChar);
  hrvService.addCharacteristic(requestChar);
  BLE.addService(hrvService);

  // Write null values to characteristics
  hrvChar.writeValue("");
  errorChar.writeValue(0);
  requestChar.writeValue("");

  // Advertise the BLE Device
  BLE.advertise();

  // SD

  // Start SD card library
  Serial.print("Initializing SD card...");
  if (!SD.begin(CHIP_SELECT)) {
    Serial.println("Card failed, or not present");

    // Hang Execution if SD card fails to initialize
    while (1);
  }
  Serial.println("card initialized.");

  // RTC

  // Start RTC module
  #ifdef ARDUINO_ARCH_ESP8266
    URTCLIB_WIRE.begin(0, 2); // D3 and D4 on ESP8266
  #else
    URTCLIB_WIRE.begin();
  #endif

  //rtc.set(0, 42, 16, 6, 2, 5, 15);

  Serial.println("RTC began!");

  // Pulse Sensor

  // Setup pulseSensor variables
  pulseSensor.analogInput(PULSE_INPUT);
  pulseSensor.setThreshold(THRESHOLD);

  // Start pulseSensor
  if (!pulseSensor.begin()) {
    Serial.println("Pulse Sensor failed to begin");

    // Hang execution if pulseSensor fails to begin
    while (1);
  }

  Serial.println("Pulse Sensor began!");

  // HRV
  resetHrv(); // Set all initial values of the HRV variables

  // Initialize all transferFile variables
  setFilename(transferFilename);

  transferFile = SD.open(transferFilename, FILE_WRITE);
  transferFile.close();

  transferFilePosition = 0;
  lastTransferTime = 0;
  transferInProgress = false;

  transferDate[0] = 2000 + rtc.year();
  transferDate[1] = rtc.month();
  transferDate[2] = rtc.day();

}

// Polls pulse sensor for new beats and handles nonblocking data transfers
void loop() {
  connectedDevice = BLE.central();

  // If a new transfer request comes in
  if (requestChar.written() && !transferInProgress) {
    // Set transferFilename and transferFilePosition according to the requestChar value

    Serial.println("New request");

    filenameFromRequestChar();

    Serial.println(transferDate[0]);
    Serial.println(transferDate[1]);
    Serial.println(transferDate[2]);

    transferInProgress = true; // Set transferInProgress to true

  }

  // If there is a current transfer in progress, send records.
  if (transferInProgress && millis() - lastTransferTime > TRANSFER_TIMEOUT) {
    Serial.println("Sending records");
    sendRecords();
  }

  // If an hrv measurement has begun and MINUTES_IN_WINDOW of minutes has passed
  if (beatsRemaining <= 0 && millis() - hrvStartTime > 60000 * MINUTES_IN_WINDOW) {
    float bpm = (numRRDetected+1)/MINUTES_IN_WINDOW; // Calculate BPM
    // If bpm is in range, store and send the rmssd measurement, else, send an error code.
    if (bpm > 40 && bpm < 240) {
      rmssd = sqrt(rrDiffSquaredTotal/(numRRDetected-1));
      Serial.print("RMSSD: ");
      Serial.println(rmssd);
      storeRecord();
    }
    else {
      errorChar.writeValue(1);
    }

    // Reset HRV variables for next window
    resetHrv();
  }

  // If a new heart beat is detected
  if (pulseSensor.sawNewSample() && pulseSensor.sawStartOfBeat()) {
    // If the measurement window hasn't begun
    if (beatsRemaining > 0) {
      // Decrement beatsRemaining. If this value reaches 0, begin measurements
      if (--beatsRemaining <= 0) {
        Serial.println("Started Window");
        hrvStartTime = millis();
        updateHrv();
      }
    }
    else {
      // Else, handle HRV variables
      updateHrv();
    }
  }
}
```

Arduino\RTCInit\RTCInit.ino

```
/*
 * Arduino RTC Initialization Code
 *
 * Created:
 * 8/2/2023 by Siem Yonas
 * Last Modified:
 * 8/5/2023 by Siem Yonas
 */

#include "Arduino.h"
#include "uRTCLib.h"
#include <UnixTime.h>
#include <SPI.h>
#include <WiFiNINA.h>

#include "arduino_secrets.h"
//////please enter your sensitive data in the Secret tab/arduino_secrets.h

// WiFi
char ssid[] = SECRET_SSID; // The WiFi network’s SSID (name)
char pass[] = SECRET_PASS; // The WiFi network’s password
int status = WL_IDLE_STATUS; // the WiFi radio's status

// RTC
uRTCLib rtc(0x68); // uRTCLib library object
UnixTime stamp(0); // Unix timestamp converter

// Handles WiFi communication RTC initialization.
void setup() {
  Serial.begin(9600);
  while (!Serial);

  // RTC

  // Start RTC module
  #ifdef ARDUINO_ARCH_ESP8266
    URTCLIB_WIRE.begin(0, 2); // D3 and D4 on ESP8266
  #else
    URTCLIB_WIRE.begin();
  #endif

  Serial.println("RTC began!");

  // WiFi
  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    while (true);
  }

  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
    Serial.println("Please upgrade the firmware");
  }

  // While the device is not connected to WiFi, attempt to connect to the WiFi
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }

  Serial.println("Connected to the network");

  // Retrieve Epoch from WiFi network
  unsigned long epoch; // The unix time from WiFi network;
  int numberOfTries = 0, maxTries = 6; // Variables for the number of tries to retrieve the Wifi Epocj

  do {
    epoch = WiFi.getTime();
    numberOfTries++;
  } while ((epoch == 0) && (numberOfTries < maxTries));

  if (numberOfTries == maxTries) {
    Serial.println("NTP unreachable!!");
    while (1);
  }

  Serial.print("Epoch received: ");
  Serial.println(epoch);

  // Initialize the RTC with the epoch
  stamp.getDateTime(epoch);

  rtc.set(stamp.second,
          stamp.minute,
          stamp.hour,
          stamp.dayOfWeek,
          stamp.day,
          stamp.month,
          (uint8_t)(stamp.year-2000)
         );


  Serial.println("RTC intialized!");
}

// Prints out current time each second
void loop() {
  // Update the RTC
  rtc.refresh();

  // Print the current DateTime
  Serial.print("RTC DateTime: ");
  Serial.print(rtc.year());
  Serial.print('/');
  Serial.print(rtc.month());
  Serial.print('/');
  Serial.print(rtc.day());

  Serial.print(' ');

  Serial.print(rtc.hour());
  Serial.print(':');
  Serial.print(rtc.minute());
  Serial.print(':');
  Serial.print(rtc.second());

  Serial.print(" DOW: ");
  Serial.print(rtc.dayOfWeek());

  Serial.println();

  // Wait a second
  delay(1000);
}

```

Arduino\RTCInit\arduino_secrets.h

```
#define SECRET_SSID "syhsop7p"
#define SECRET_PASS "lifeisgood"

```