Schluff - The Sleep Monitor
Created by Michael Sklar

Last updated on 2020-11-02 05:11:52 PM EST
Designing a Sleep Monitor

We spend a third of our lives sleeping, and the quality of our sleep can have a big impact on how we feel during the day. Of course, being that we’re asleep during that time, it makes it hard to accurately gauge how comfortable we are at the moment. Sleep studies are one way to get accurate data - but they’re a big deal to run! Could we DIY?

Monitoring a sleep environment combined with heart rate data can tell us a lot about our needs. Here are some things that Schluff can do to help you maximize your nighty rest.

**Temperature** - The ideal sleeping temperature is claimed to be on the cooler side near 65F. Sleeping at temperatures too far from that should result in reduced duration and quality of sleep.

**Humidity** - Ideally 45%, but anywhere between 30-50% humidity can offer a good nights rest.

**Light / Lux** - You want these both to read zero during sleep. Recording light levels through the night can help pinpoint regular light disturbances (car headlights, full moon, porch lights, etc).

**Noise** - Did something go bump in the night? Is it quieter tonight than other nights? Did you wake yourself up again from snoring too loud? Maybe your partner is the noise maker?

**Heart Rate** - How low can you go? Ideal heart rates will vary from person to person, but generally a low resting heart rate means you are recovered and ready to go. A higher than normal heart rate indicates that you need additional rest. HR can also be used to indicate different states of sleep with REM having an irregular HR and the light and deep having lower more stable heart rates.
Components
This sleep monitor is loaded up with sensors, display and wireless abilities. All of the components used in this project are available in the Adafruit store.

Adafruit Feather 32u4 Bluefruit LE
$29.95
IN STOCK
Add To Cart

Adafruit FeatherWing OLED - 128x32 OLED Add-on For Feather
$14.95
IN STOCK
Add To Cart

DHT22 temperature-humidity sensor + extras
$9.95
IN STOCK
Add To Cart
Adafruit TSL2561 Digital Luminosity/Lux/Light Sensor Breakout

OUT OF STOCK
Out Of Stock

Electret Microphone Amplifier - MAX9814 with Auto Gain Control

$7.95
IN STOCK
Add To Cart

Heart Rate Educational Starter Pack with Polar Wireless Sensors

$65.00
IN STOCK
Add To Cart

Half-size breadboard

$5.00
IN STOCK
Add To Cart
USB cable - USB A to Micro-B

$2.95
IN STOCK
Add To Cart

Stacking Headers for Feather - 12-pin and 16-pin female headers

$1.25
IN STOCK
Add To Cart
This fritzing breadboard diagram is straight forward. Here is the line by line summary:

**MCU Power:**
- Feather [3V] --> Breadboard VCC [Red Power Rail]
- Feather [GND] --> Breadboard GND [Blue Ground Rail]

**Mic:**
- Max9814 [GND] --> Breadboard [GND]
- Max9814 [Vdd] --> Breadboard [VCC]
- Max9814 [Gain] --> Breadboard [VCC]
- Max9814 [OUT] --> Feather [A0] (analog)

**Heart Rate:**
- Polar F5 [GND] --> Breadboard [GND]
- Polar F5 [VCC] --> Breadboard [VCC]
- Polar F5 [OUT] --> Feather #11 (digital in)

**Light:**
- TSL2561 [Vin] --> Breadboard [VCC]
- TSL2561 [GND] --> Breadboard [GND]
- TSL2561 [SDA] --> Feather #2 [SDA]
- TSL2561 [SCL] --> Feather #3 [SCL]

**Temp/Humidity:**

- DHT22 [VCC] --> Breadboard [VCC]
- DHT22 [OUT] --> Feather #5
  Also connect through a 10k Resistor --> Breadboard [VCC]
- DHT22 [GND] --> Breadboard [GND]

The wiring and pin connects can be seen a little easier with the OLED display removed.

The Feather 32u4 BLE datasheet summary can be helpful so I've included it below.
Download Libraries

This project requires a bunch of Arduino libraries. You will need to install all of these libraries to follow this guide.

Open up the Arduino library manager:

Search for the Adafruit GFX library and install it

If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install Adafruit_BusIO (newer versions will install this dependency automatically).

Search for the Adafruit SSD1306 library and install it

Search for the DHT Sensor library and install it

Search for the Adafruit TSL2561 library and install it

Search for the Adafruit Sensor library and install it
Search for the Adafruit nRF51 library and install it

You'll need to install the Timer library manually. Download it from the following link and import it into the IDE with Sketch -> Include Library -> Add .ZIP Library

https://adafruit.it/sb9

We also have a great tutorial on Arduino library installation at:
Arduino Feather Setup

Set up your Feather 32u4 and Arduino IDE by visiting the Learn guide for the 32u4 here. Once you've installed all the drivers and software, and tested the Feather come back.

https://adafruit.it/IHa

https://adafruit.it/IHa
Arduino Code

This project makes use of four different sensors, a display and BLE for communication so there is a fair amount of code here. To make things easier I've modularized the setup() and loop() code into sections based on each of the different components being used. This way you should be able to cut and paste full functions to work with your own mix of sensors.

Before you can cut and paste the mountain of code this project uses you will need to load up a piece of example code so you can get the second BluefruitConfig.h header file loaded as a second tab in the Arduino App. I'll load up the nRF51 heartratemonitor as an example.
Once the heartratemonitor example is loaded simply select all the code in the first 'heartratemonitor' tab, delete it and
go to 'File' -> 'Save As' with a new name for your project. Then go ahead and paste the code below into the first tab
with your new project name.

```c
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#include <TSL2561.h>
#include <Timer.h>

#ifndef DEBUG
#define DEBUG_PRINTF(x) Serial.print(x)
#define DEBUG_PRINTFDEC(x) Serial.print(x, DEC)
#define DEBUG_PRINTFNL(x) Serial.println(x)
#else
#define DEBUG_PRINTF(x)
#define DEBUG_PRINTFDEC(x)
#define DEBUG_PRINTFNL(x)
#endif

// temperature humidity sensor
#define DHTPIN 5       // what digital pin we're connected to
#define DHTTYPE DHT22  // DHT 22
DHT dht(DHTPIN, DHTTYPE);
```
// light sensor
TSL2561 tsl(TSL2561_ADDR_FLOAT);

// OLED display buttons
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);
#if defined(ESP8266)
#define BUTTON_A 0
#define BUTTON_B 16
#define BUTTON_C 2
#define LED 0
#elif defined(ARDUINO_STM32F2_FEATHER)
#define BUTTON_A PA15
#define BUTTON_B PC7
#define BUTTON_C PC5
#define LED PB5
#elif defined(TEENSYDUINO)
#define BUTTON_A 4
#define BUTTON_B 3
#define BUTTON_C 8
#define LED 13
#else
#define BUTTON_A 9
#define BUTTON_B 6
#define BUTTON_C 5
#define LED 13
#endif
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"
#include "BluefruitConfig.h"

Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);

// A small helper
void error(const __FlashStringHelper*err) {
  Serial.println(err);
  for(;;);
}

// The service information
int32_t hrmServiceId;
int32_t hrmMeasureCharId;
int32_t hrmLocationCharId;

// globals

// max9814 (mic board used for noise measurements)
const int sampleWindow = 50; // Sample window width in mS (50 mS = 20Hz)
unsigned int sample;

// humidity, temperature, fahrenheit
float h,t,f;

// audio mic noise
double noise;

// light levels
uint32_t lum;
uint16_t ir, full, lux;
// Polar Heart Rate Monitor
Timer minute;
const int HR_RX = 10;
byte hr_oldSample, hr_sample;
int beatcount = 0;
int hrm_bpm = 0;

// average sensor data
#define AVG_COUNT 2         // number of passes
int average_counter = 0;    // current location
float temp_avg,hum_avg;
uint16_t full_avg, lux_avg;
double noise_avg;
int heart_rate_avg;

double sensor_max_loops = 250000;
double sensor_loop = 0;

void setup() {
    Serial.begin(9600);           // serial baud rate
    DEBUG_PRINTLN("setup begin");
    dht.begin();                  // temperature and humidity
    oled_setup();                 // OLED display module for feather
    tsl2561_light_sensor_setup(); // light level sensor
    heart_rate_monitor_setup();   // HRM Polar setup code
    ble_communication_setup();    // Bluetooth UART datalogging
    DEBUG_PRINTLN("finished setup");
}

void loop() {
    DEBUG_PRINTLN("loop begin");
    heart_rate_monitor_loop();   // heart rate monitor (any commercial HRM)
    sensor_loop++;

    if(sensor_loop == sensor_max_loops) {
        sensor_loop = 0;

        dht11_temp_humidity_loop(); // temperature and humidity DHT-11 sensor read and serial output
        button_scan_loop();         // checking A, B and C buttons for a press on the OLED featherwing
        max9814_noise_loop();       // audio mic noise levels
        tsl2561_light_loop();       // visible light and lux
        accumulate_sensors_loop(); // add up the sensor samples

        if(average_counter == AVG_COUNT) {
            average_sensors_loop();   // divide accumulated sensor data by AVG_COUNT
            display_oled_loop();      // show sensor data on OLED featherwing
            ble_datalogging_loop();   // send to BlueFruit iOS app
            reset_average_samples_loop(); // reset values
        }
    }
}
DEBUG_PRINTLN("loop finished");
}

void ble_communication_setup() {
  /* Initialise the module */
  Serial.print(F("Initialising the Bluefruit LE module: "));
  if(!ble.begin(VERBOSE_MODE)) {
    error(F("Couldn't find Bluefruit, make sure it's in Command mode & check wiring?"));
  }
  Serial.println(F("OK!")));

  /* Perform a factory reset to make sure everything is in a known state */
  Serial.println(F("Performing a factory reset: "));
  if(! ble.factoryReset()) {
    error(F("Couldn't factory reset");
  }

  /* Disable command echo from Bluefruit */
  ble.echo(false);

  Serial.println("Requesting Bluefruit info: ");
  /* Print Bluefruit information */
  ble.info();
}

void oled_setup() {
  Serial.println("OLED FeatherWing test");
  // by default, we'll generate the high voltage from the 3.3v line internally! (neat!)
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C addr 0x3C (for the 128x32)
  // init done
  Serial.println("OLED begun");

  // Show image buffer on the display hardware.
  // Since the buffer is initialized with an Adafruit splashscreen
  // internally, this will display the splashscreen.
  display.display();
  delay(1000);

  // Clear the buffer.
  display.clearDisplay();
  display.display();

  Serial.println("IO test");

  pinMode(BUTTON_A, INPUT_PULLUP);
  pinMode(BUTTON_B, INPUT_PULLUP);
  pinMode(BUTTON_C, INPUT_PULLUP);

  // text display tests
  display.setTextSize(0);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
}

void tsl2561_light_sensor_setup() {
  // light sensor
  if(tsl.begin()) {
    Serial.println("Found light sensor");
  } else {

Serial.println("No light sensor?");
for(;;);
}

tsl.setGain(TSL2561_GAIN_16X); // set 16x gain (for dim situations)
tsln.setTiming(TSL2561_INTEGRATIONTIME_13MS); // shortest integration time (bright light)

void heart_rate_monitor_setup() {
// every 60 seconds
int tickEvent = minute.every(60000, average_beats);

pinMode(HR_RX, INPUT); //Signal pin to input
Serial.println("Waiting for heart beat...");

//Wait until a heart beat is detected
while(!digitalRead(HR_RX)) {
    Serial.println("Heart beat detected!");
}

void dht11_temp_humidity_loop() {
   // Reading temperature or humidity takes about 250 milliseconds!
   // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
   h = dht.readHumidity();
   // Read temperature as Celsius (the default)
   t = dht.readTemperature();
   // Read temperature as Fahrenheit (isFahrenheit = true)
   f = dht.readTemperature(true);

   // Check if any reads failed and exit early (to try again).
   if(isnan(h) || isnan(t) || isnan(f)) {
       Serial.println("Failed to read from DHT sensor!");
       return;
   }

   // Compute heat index in Fahrenheit (the default)
   float hif = dht.computeHeatIndex(f, h);
   // Compute heat index in Celsius (isFahrenheit = false)
   float hic = dht.computeHeatIndex(t, h, false);
}

void button_scan_loop() {
    if(!digitalRead(BUTTON_A)) display.print("A");
    if(!digitalRead(BUTTON_B)) display.print("B");
    if(!digitalRead(BUTTON_C)) display.print("C");
}

void max9814_noise_loop() {
    unsigned long startMillis= millis(); // Start of sample window
    unsigned int peakToPeak = 0; // peak-to-peak level

    unsigned int signalMax = 0;
    unsigned int signalMin = 1024;

    // collect data for 50 mS
    while(millis() - startMillis < sampleWindow) {
        sample = analogRead(0);
    }
if(sample < 1024) {      // toss out spurious readings
    if(sample > signalMax) {
        signalMax = sample;  // save just the max levels
    } else if(sample < signalMin) {
        signalMin = sample;  // save just the min levels
    }
}

peakToPeak = signalMax - signalMin; // max - min = peak-peak amplitude
noise = (peakToPeak * 3.3) / 1024;  // convert to volts

void tsl2561_light_loop() {
    // light
    // Simple data read example. Just read the infrared, fullspecrtrum diode
    // or 'visible' (difference between the two) channels.
    uint16_t x = tsl.getLuminosity(TSL2561_VISIBLE);
    lum = tsl.getFullLuminosity();
    ir = lum >> 16;
    full = lum & 0xFFFF;
    lux = tsl.calculateLux(full, ir);
}

void heart_rate_monitor_loop() {
    minute.update();

    hr_sample = digitalRead(HR_RX);  // Store signal output

    if(hr_sample && (hr_oldSample != hr_sample)) {
        beatcount++;
    }

    hr_oldSample = hr_sample;       // Store last signal received
}

void average_beats() {
    hrm_bpm = beatcount;
    Serial.print("beat count: ");
    Serial.println(hrm_bpm);
    beatcount = 0;
}

void accumulate_sensors_loop() {
    temp_avg += f;
    hum_avg += h;
    full_avg += full;
    lux_avg += lux;
    noise_avg += noise;
    average_counter++;
}

void average_sensors_loop() {
    temp_avg = temp_avg / AVG_COUNT;
    hum_avg = hum_avg / AVG_COUNT;
    full_avg = full_avg / AVG_COUNT;
    lux_avg = lux_avg / AVG_COUNT;
    noise_avg = noise_avg / AVG_COUNT;
    noise_avg *= 100;
}
void reset_average_samples_loop() {
    temp_avg = 0;
    hum_avg = 0;
    full_avg = 0;
    lux_avg = 0;
    noise_avg = 0;
    average_counter = 0;
}

void display_oled_loop() {
    display.clearDisplay();
    display.setCursor(0,0);
    display.print("Temp ");
    display.print(temp_avg);
    display.print(" Hum ");
    display.print(hum_avg);
    display.println();
    display.println("Light ");
    display.print(full_avg);
    display.print(" Lux ");
    display.print(lux_avg);
    display.println();
    display.println("Noise ");
    display.print(noise_avg);
    display.println();
    display.print("Heart Rate: ");
    display.print(hrm_bpm);
    display.display();
}

void ble_datalogging_loop() {
    char str_temp_avg[3], str_hum_avg[3], str_noise_avg[3];
    char buffer[20];
    ble.setMode(BLUEFRUIT_MODE_DATA);
    dtostrf(temp_avg,2,0,str_temp_avg);
    dtostrf(hum_avg,2,0,str_hum_avg);
    dtostrf(noise_avg,2,0,str_noise_avg);
    snprintf(buffer, 20, "%s,%s,%d,%d,%s,%d", str_temp_avg, str_hum_avg, full_avg, lux_avg, str_noise_avg, hrm_bpm);
    ble.print(buffer);
}
Download the Adafruit Bluefruit LE Connect App from the Apple App Store.

Run the downloaded app and select the Adafruit Bluefruit LE device.
You should now see the data being sent from the Arduino BLE device over to your phone by pressing the 'Uart' button second from the lower left. The data is being sent over in comma separated format as:

temperature, humidity, light, lux, noise and heart rate
80, 56, 242, 46, 5, 0
80, 57, 241, 46, 5, 63
80, 57, 239, 45, 5, 63
80, 57, 237, 45, 5, 69
80, 58, 235, 44, 5, 67
79, 58, 236, 45, 5, 67
79, 58, 234, 44, 5, 66
79, 58, 233, 44, 5, 66
79, 58, 232, 44, 5, 65
79, 58, 230, 44, 8, 64
79, 58, 229, 43, 5, 64
79, 58, 231, 43, 5, 65
80, 58, 227, 42, 5, 65
80, 58, 225, 42, 5, 65
80, 58, 225, 42, 5, 63

Sent: 0 bytes  Received: 353 bytes
macOS BLE Setup

macOS offers good BLE support and only requires one library to be installed. The Adafruit Python BluefruitLE library (https://adafruit.it/fQF)

```
git clone https://github.com/adafruit/Adafruit_Python_BluefruitLE.git
cd Adafruit_Python_BluefruitLE
sudo python setup.py install
```

This what the default Schluff output looks like on a macOS system which is connected to the Feather 32u4 BLE using Bluetooth and relaying the data to the adafruit.io website. The output is in the order of temperature, humidity, light, lux, noise and heart rate.

```
Using adapter: Default Adapter
Disconnecting any connected UART devices...
Searching for UART device...
Connecting to device...
Discovering services...
T: 80 H: 36 Li: 134 Lux: 22 N: 4 HR: 69
T: 80 H: 36 Li: 133 Lux: 22 N: 6 HR: 69
T: 80 H: 36 Li: 133 Lux: 23 N: 4 HR: 71
T: 80 H: 37 Li: 134 Lux: 23 N: 7 HR: 71
```

The library used in this guide is deprecated in favor of the Adafruit_Blinka_bleio library. See this guide for more information: https://learn.adafruit.com/circuitpython-ble-libraries-on-any-computer
This script will run on a macOS system. It will automatically enable Bluetooth and connect to an Arduino BLE controller. You will need to have installed the necessary package in the macOS BLE setup page (https://adafruit.io/sBl), have created an Adafruit.io account and update line #10 of the code below with 'YOUR ADAFRUIT IO KEY'. Please keep the single quotes around your key.

```python
# BLE Catcher Modified by : Mikey Sklar
# Original Code Credit : Tony DiCola @ Adafruit
# BLE Catcher Modified by : Mikey Sklar @ Adafruit
import Adafruit_BluefruitLE
from Adafruit_BluefruitLE.services import UART

from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO KEY')

# Get the BLE provider for the current platform.
ble = Adafruit_BluefruitLE.get_provider()

# Initialize the BLE system. MUST be called before other BLE calls!
ble.initialize()

def main():
    # Clear any cached data because both bluez and CoreBluetooth have issues with
    # caching data and it going stale.
    ble.clear_cached_data()

    # Get the first available BLE network adapter and make sure it's powered on.
    adapter = ble.get_default_adapter()
    adapter.power_on()
    print('Using adapter: {0}'.format(adapter.name))

    # Disconnect any currently connected UART devices. Good for cleaning up and
    # starting from a fresh state.
    UART.disconnect_devices()

    # Scan for UART devices.
    print('Searching for UART device...')
    try:
        adapter.start_scan()
        # Search for the first UART device found (will time out after 60 seconds
        # but you can specify an optional timeout_sec parameter to change it).
        device = UART.find_device()
        if device is None:
            raise RuntimeError('Failed to find UART device!')
    finally:
        ...
```

© Adafruit Industries
finally:
    # Make sure scanning is stopped before exiting.
    adapter.stop_scan()

    print('Connecting to device...')
    device.connect()  # Will time out after 60 seconds, specify timeout_sec parameter
    # to change the timeout.

    print('Discovering services...')
    UART.discover(device)

    while True:
        uart = UART(device)
        receive = uart.read(timeout_sec=300)  # 5min timeout

        if receive:
            sd = receive.split(",");
            sd[4] + " HR: " + sd[5])
        else:
            continue

        if len(sd) == 6:
            aio.send('Temperature', sd[0])
            aio.send('Humidity', sd[1])
            aio.send('Light', sd[2])
            aio.send('Lux', sd[3])
            aio.send('Noise', sd[4])
            aio.send('Heart Rate', sd[5])
        else:
            continue

    # Start the mainloop to process BLE events, and run the provided function in
    # a background thread. When the provided main function stops running, returns
    # an integer status code, or throws an error the program will exit.
    ble.run_mainloop_with(main)

Run the python BLE catcher script on your macOS box in a constant loop to automatically reconnect should a
disconnect occur. Use ctrl\ to break out.

while true; do python ble_catcher_parser_pusher.py ; done
Logging and Graphing with Adafruit.io

Adafruit.io is a super easy way to save, graph and review your nightly sleep data. The code provided in the macOS Catcher Script (https://adafruit.it/sBm) will log to adafruit.io. The only thing necessary to do is to create an adafruit.io account and add your key to the macOS catcher script (line# 10). The feeds will automatically be setup.

Please go to 'io.adafruit.com (https://adafruit.it/fsU)' and create an accountant if you have not already.

This is where you can find the key once logged into your Adafruit IO account.

The feeds will be automatically generated once the AIO key has been added to the macOS catcher script. No setup is necessary on the Adafruit IO site itself. Just start running the catcher script and let the Arduino connect to it.
You can work with and monitor the data by creating groups and dashboards. This is an intuitive process once data is uploading. Here is an example of a group graph I created by combining all my feeds (sensors) and clicking on the 'graph it' option.
Pure Hackery

The code and schematics provided here include some solid examples for monitoring a room environment, a human heart rate, pushing data to different Apple devices and finally saving everything to Adafruit.io. What I should also divulge is how well it actually works.

Here are the hacks:

- The light level goes to zero pretty fast. The difference between a pitch black room and a pretty dark room is not being picked up well by the TSL2561 sensor. It does a great job of distinguishing brightness levels during the day.
- The heart rate strap provides excellent data, but you need to be within a few feet of the receiver. This means the sleep monitor hardware will need to be right beside your bed.
- The sleep monitor has LEDs and a display so this could be a distracting. Some well placed electrical tape could resolve this.
- The iOS Bluefruit App will easily catch the data and save it to CSV files. I've not been able to use the iOS Bluefruit App to forward the data directly adafruit.io.
- My macOS bluetooth catcher script likes to find reasons to exit so I always start it up with a:

```bash
while true ; do python ble_catcher_parser_pusher.py ; done
```