
Introduction

Explanation of concept

Have you ever wanted to optimize the sleep you get at night? From service staff working night shifts to doctors staffing the emergency room to athletes on professional training cycles, many people in society only have a limited window of time in which they can get the sleep they need for the night.

What if there was a way to guarantee restful sleep during this window of time?

Sleep therapists recommend waking from sleep after completion of a REM cycle, which is the latter half of the general sleep cycle. A full sleep cycle typically lasts from 90-110 minutes, with an approximate breakdown of the length of each phase provided in the diagram below.

Sleep Stages	Type of Sleep	Other Names	Normal Length
Stage 1	NREM	N1	1-7 minutes
Stage 2	NREM	N2	10-25 minutes
Stage 3	NREM	N3, slow-wave sleep (SWS), delta sleep, deep sleep	20-40 minutes
Stage 4	REM	REM Sleep	10-60 minutes

(Originally from the Sleep Foundation, found at https://www.sleepfoundation.org/stages-of-sleep)

Standard alarms work for this purpose in theory; in practice an alarm may wake you up midway through a REM cycle, which results in grogginess upon awakening. Our final project, the REM Clock, proposes a solution to this issue by designing a novel product that wakes up users after the completion of a REM cycle as opposed to a fixed alarm time. Users still set an alarm time, but they also input a desired quantity of REM cycles. If they have completed their indicated REM cycles before their set alarm time, the alarm will sound early, at that precise time

when the cycle is completed. (For example, John sets an alarm for 8:00 AM with REMcount = 2. If he finishes 2 REM cycles at 7:45 AM, the alarm wakes him up early, and he may feel more energized than if he had woken up at 8:00 AM). The REM Clock is implemented primarily with a heart rate monitor and a programmable Arduino Uno. This will be elaborated further in the Materials section.

Materials

Guide of components needed to recreate project

- Buzzer:
    - Buzzer, 3-24V Electronic Buzzer Alarm Sounder Continuous Tone Buzzer 85 to 95dB Door Buzzer w/ 100mm Cable Length
- Heart Rate Sensor:
    - PulseSensor.com The Real & Original Pulse Sensor Plug-in for Your Project.
- LCD Display:
    - HiLetgo 2pcs HD44780 1602 LCD Display Module DC 5V 16x2 Character LCM Blue Blacklight New
- Arduino Uno
- Breadboard
- Male to Male connecting wires
- 2 9V batteries and holders

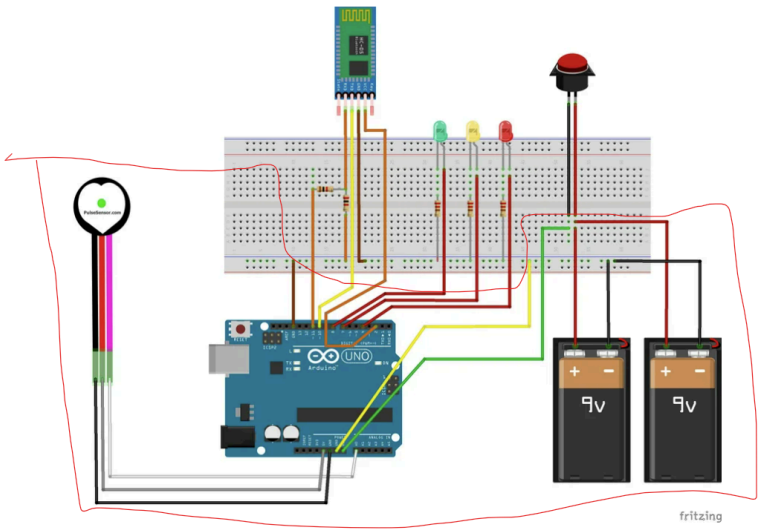
The basic materials needed are really just the heart rate sensor and the Arduino Uno. An improvement of this project may consider manufacturing the heart rate monitor into a ring, with chips linked in the future development section.

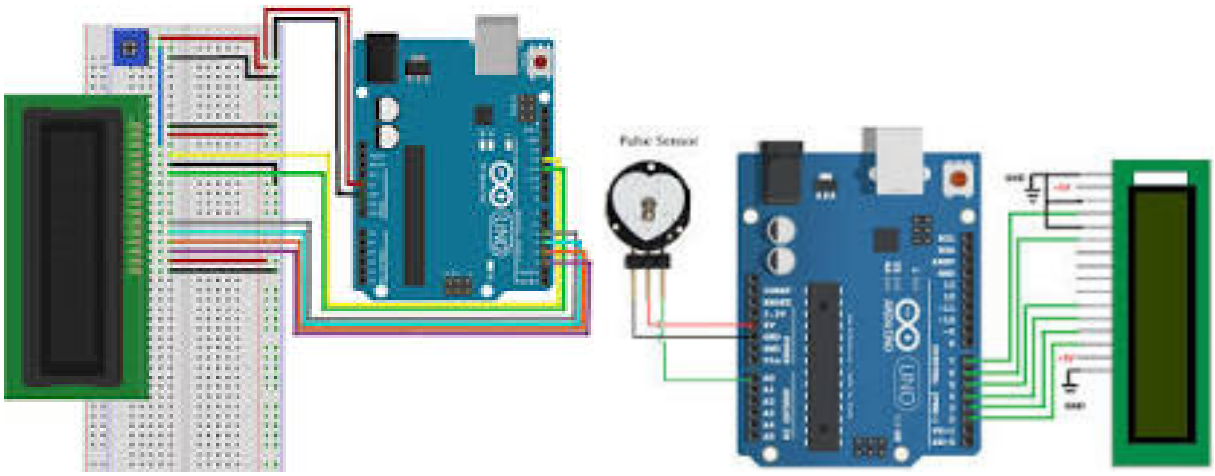
Design

Following walk through of how to set-up with Arduino Uno

We used the following reference materials when constructing our circuit

https://www.instructables.com/ZazHRM-a-Bluetooth-Heart-Rate-Monitoring-System-fo/





To design the algorithm, we implemented the following pseudo code:

a. Record heart rate data via `pulseSensor.getBeatsPerMinute()`; store as int
b. Update the alarm time accordingly:
    i. To approximate the time of REM cycle completion, the goal is to first approximate deep sleep completion (easily marked as the sleep phase in which lowest heart rate likely occurs). To find lowest heart rate in cycle:
        1. Store initial heart rate in a champion int variable. Compare current heart rate with next heart rate (after delay(20)). If next is lower, store nextHeartRate as the new champion. Store the time at which this champion heart rate occurs in a Chronos DateTime object.
        2. Wait till no instance of lower heart rate has occurred in a period of 20 minutes. This suggests that heart rate has stopped decreasing and deep sleep phase may be over
        3. Set Chronos DateTime object for REM Cycle completion for 50 minutes after the time set for the champion heart rate. This approximates the time from the end of deep sleep to the end of REM sleep.
    ii. Implement a functionality so that the REM Clock will never wake someone up after their set alarm time, only before this time if they have completed their REM Cycle:
        1. Instantiate a Chronos DateTime object in setup() with user input for the time they want to set alarm for
        2. Check if the target time set for REM cycle completion in Step b.i.3 above is set after the Chronos DateTime object inputted by the user
            a. If so, override the target REM time using boolean to switch conditions, and set the alarm for the user-inputted time
            b. If not, and the amount of user-inputted REM cycles was completed, maintain REM target time
            c. If not, and more REM cycles remain, program will keep running till the total count of REM cycles matches the desired amount inputted by user

Findings

Progress log of what works and what needs improvement

In the end, we were able to get the project working to our specifications. The Arduino program correctly detects the end of a period of deep sleep, updates the REM cycle accordingly, and updates the alarm time accordingly. The buzzer is functional (and quite loud) and will serve as a proper alarm. Images of the completed circuit and LCD display setup are shown in the gallery section.

The LCD pins {D4, D5, D6, D7, E, RS}are connected to ports {2, 3, 4, 5, 11, 12} of the Arduino, respectively. V0 of the LCD is connected to a voltage divider on the board with a 1kΩ and 4kΩ resistor, yielding a voltage of 1V for optimal contrast on the LCD. The buzzer is connected to port 6. The heart rate monitor is connected to A0. All remaining VSS and VDD for each component are connected to the ground and 5V ports of the Arduino as applicable.

Getting all the parts to work correctly was quite difficult and took up the majority of our lab time. Both of our initial heart rate chips were faulty, as well as one of our two LCD displays.

Code

.ino file also attached separately in zip file submission

```
// set-up low-level interrupts for most acurate BPM math.
#define USE_ARDUINO_INTERRUPTS true

// include the library code:
#include <LiquidCrystal.h>
#include <PulseSensorPlayground.h>
#include <TimeLib.h>
#include <Chronos.h>

// initialize the library by associating any needed LCD interface pin with the arduino
pin number it is connected to
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// creates an instance of the PulseSensorPlayground object called "pulseSensor"
PulseSensorPlayground pulseSensor;
const int PulseWire = 0;            // pulseSensor PURPLE WIRE connected to ANALOG PIN
0
const int LED = LED_BUILTIN;        // the on-board Arduino LED
int Threshold = 550;                // heartbeat signal threshold (borrowed from
example sketch)

int minHeartRate = 1000;            // log the lowest heart rate in sleep cycle to
approximate deep sleep phase (NREM3)
bool alarmSound = false;            // set true when alarm rings
bool alarmSet = false;              // true when end of REM cycle has been calculated/
int currentREM = 0;                 // count # of current REM cycles
bool forceStop = false;             // true when targetTime exceeds user input alarm
time
const int targetREM = 3;            // take desired # of user REM cycles as input (set
to 3 for now, will develop further after creating user interface/app)
int myBPM;                          // global bpm variable

Chronos::DateTime candidateTime;    // candidate time at which minimum heart rate
occurs in a 20 min period
Chronos::DateTime alarmTime;        // predicted time for end of REM Cycle, based on
conditions of candidateTime
```

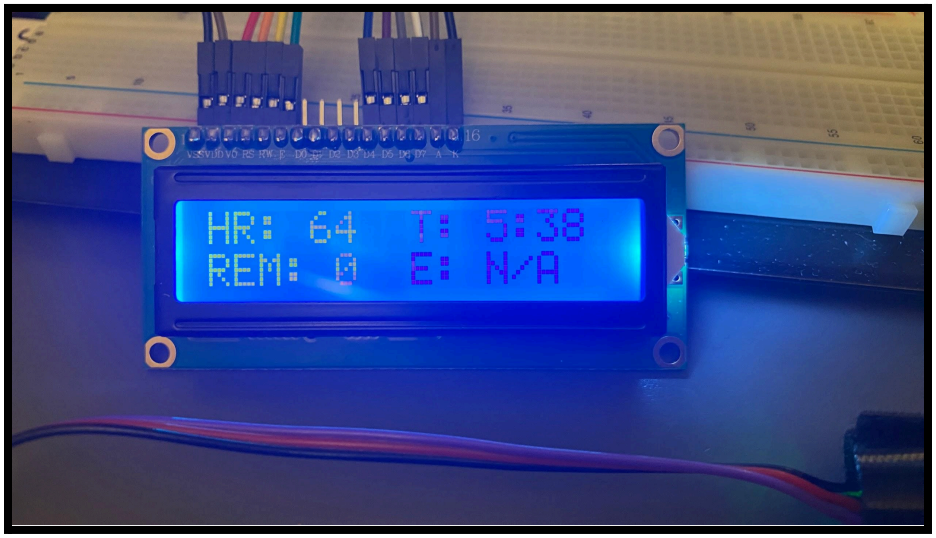
```
Chronos::DateTime forceStopTime;    // latest time at which the user wants to wake up (i.e. a normal alarm at 8:00 AM). for now, hard-coded. In the future, will allow user to enter input via app



void setup() {
 // Sets serial monitor
 Serial.begin(9600);

 // Sets LCD object
 lcd.begin(16, 2);

 // Sets time to now (year, month, date, hour, sec, ms)
 Chronos::DateTime::setTime(2023, 05, 9, 05, 30, 00);

 // Configure the PulseSensor object, by assigning our variables to it.
 pulseSensor.analogInput(PulseWire);
 pulseSensor.blinkOnPulse(LED);
 pulseSensor.setThreshold(Threshold);

 // Double-check the "pulseSensor" object was created and "began" seeing a signal.
 if (pulseSensor.begin()) {
   Serial.println("We created a pulseSensor Object !");  //This prints one time at Arduino power-up,  or on Arduino reset.
 }

 // set candidate time to now time
 candidateTime = Chronos::DateTime::now();

 // hard-coded for now, will eventually let user input alarm time via app
 // (year, month, day, hour, second, ms)
 forceStopTime = Chronos::DateTime(2023, 5, 9, 5, 40, 00);
}

void updateAlarmTime() {
 // if alarm time is already set, do nothing
 if (!alarmSet) {
   // if 20 minutes has elapsed since the previous min heartrate, a deep sleep period should be over
   if (Chronos::DateTime::now() > candidateTime + Chronos::Span::Minutes(20)) {
     currentREM++;
```

```
    // if projected end of REM is past forceStopTime, set alarm to forceStopTime
    if (Chronos::DateTime::now() + Chronos::Span::Minutes(50) > forceStopTime) {
      alarmTime = forceStopTime;
      alarmSet = true;
    // otherwise if we've reached the target REM cycles then set alarm to end of REM
    } else if (currentREM == targetREM) {
      alarmTime = Chronos::DateTime::now() + Chronos::Span::Minutes(50);
      alarmSet = true;
    }
    candidateTime += Chronos::Span::Minutes(70);
  } else if (myBPM < minHeartRate) {
      candidateTime = Chronos::DateTime::now();
  }
  //jank solution for restarting min champion after REM cycle
  if (candidateTime == Chronos::DateTime::now()) {
    minHeartRate = myBPM;
  }
 }
}

void loop() {
 if (pulseSensor.sawStartOfBeat()) {
   myBPM = pulseSensor.getBeatsPerMinute();
   updateAlarmTime();
 }

 //wellness stat display
 lcd.clear();
 lcd.setCursor(0,0);
 lcd.print("HR: ");
 lcd.print(myBPM);
 lcd.setCursor(0,1);
 lcd.print("REM: ");
 lcd.print(currentREM);

 //row 2, Clock display
 lcd.setCursor(8,0);
 lcd.print("T: ");
 lcd.print(Chronos::DateTime::now().hour());
 lcd.print(":");
 lcd.print(Chronos::DateTime::now().minute());
```

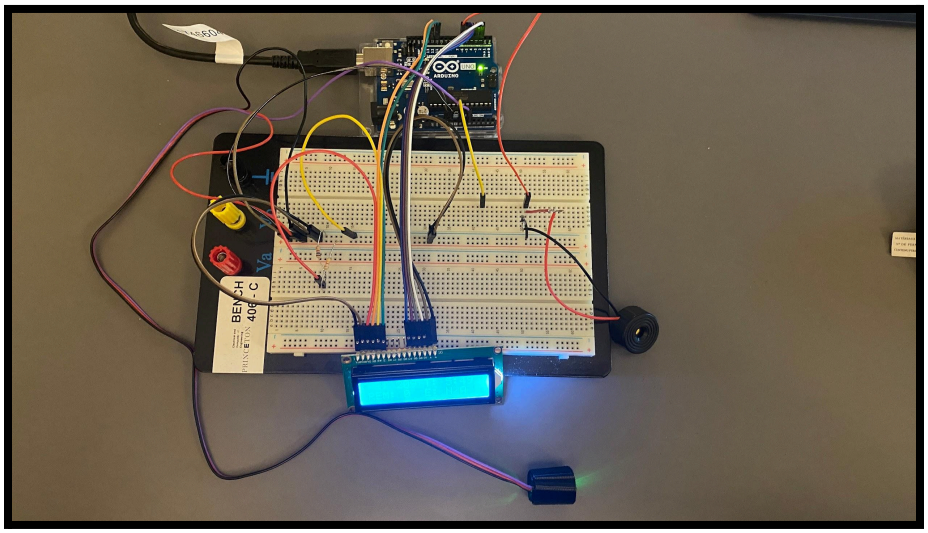
```
lcd.setCursor(8,1);
lcd.print("E: ");
if (!alarmSet) {
  lcd.print("N/A");
} else {
  lcd.print(alarmTime.hour());
  lcd.print(":");
  lcd.print(alarmTime.minute());
}

if (alarmSet) {
  analogWrite(6,127);
} else {
  analogWrite(6,0);
}

delay(20);
```

Gallery

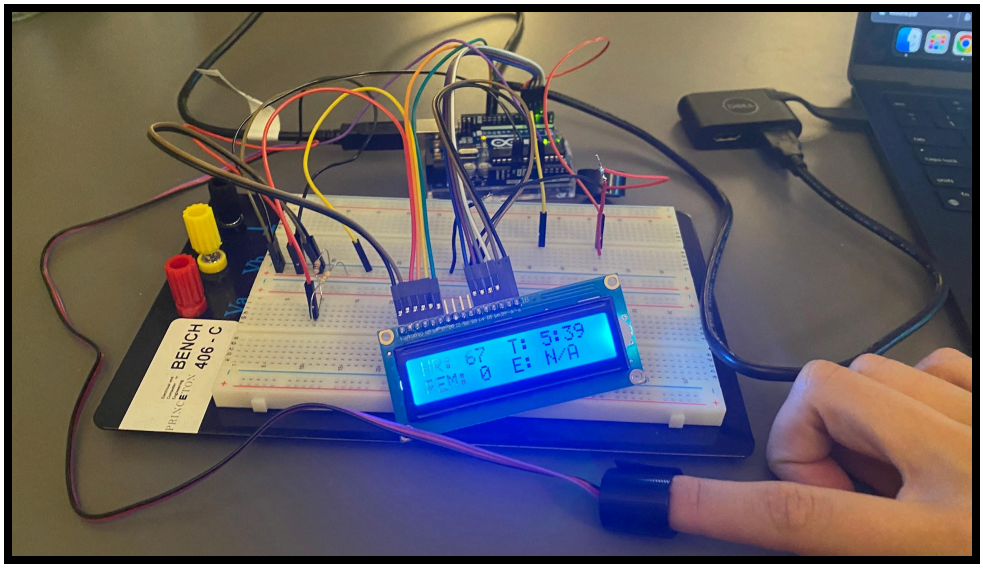
Images of the project at final stage and intermediary steps



HR: Displays the heart rate of the user
REM: Displays the current REM cycle the user is in
T: Displays current time
E: Displays the alarm time, or N/A if it has not been calculated yet



Top-down view of the breadboard



REMclock in use

Goals for Future Development

Improving the functionality of the REM Clock

Currently, it remains to be seen whether or not this algorithm/setup is effective in identifying the completion of REM sleep cycles. Identifying this would require thorough experimentation and refinement including the use of volunteers in a sleep lab who would wear the device over night.

Moreover, the end goal would be to make this into a minimally invasive product that can be worn easily by users, such as a ring:

(DALL-E generated artwork)

To accomplish this end, we would need to use entirely different components, including a smaller heart rate monitor and a smaller microprocessor chip. Along with integrating charging port functionality, all components would ideally be connected via a printed circuit board and placed within a 3D printed ring model from a CAD software.

On the software side, the ring would be supplemented by an app that is a conventional iOS/Android app with the alarm features specified in this document. Cross-compatibility with Apple watches and Fitbits would be a very promising feature.

Please reach out if you have any comments or feedback on the viability of this product and if you think it is worth developing/researching further. Thank you for reviewing this project!

Best, Shivam and Maxwell