Vitezometru si Odometru pentru bicicleta, cu recorder pentru 99 de lap-uri.

Introducere

Prezentarea pe scurt a proiectului vostru:

- Vitezometru si Odometru, bazat pe o placa de baza Arduino Uno
- monitorizarea vitezei si a performantei pe durata a 99 de ture
- nevoia unei cunostinte de a avea un vitezometru
- elimina nevoia de a purta un telefon in buzunar pentru a monitoriza performantele ce pot fi eventual eronate

Descriere generală

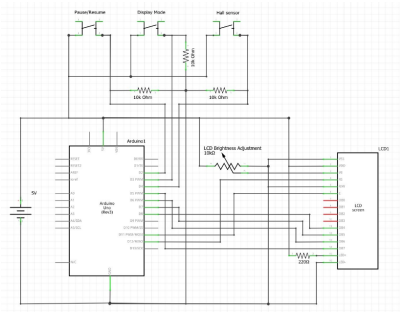
O schemă bloc cu toate modulele proiectului vostru, atât software cât şi hardware însoţită de o descriere a acestora precum şi a modului în care interacţionează.

Exemplu de schemă bloc: http://www.robs-projects.com/mp3proj/newplayer.html

Hardware Design

Aici puneţi tot ce ţine de hardware design:

- Arduino Uno - 1
- Hall Effect Sensor - 1
- Push Buttons - 2
- 16×2 LCD Display - 1
- 10kΩ Potentiometer - 1
- 10kΩ Resistor - 2



Software Design

- Arduino IDE
- Additional Libraries: Wire/EEPROM/RTClib/LiquidCrystal

```
#include <LiquidCrystal.h>
```

```
LiquidCrystal lcd(12, 11, 8, 7, 6, 5);

float bicycleWheelCircumference = 2.1206;

const int pauseButton = 2;
boolean lastPauseButton = LOW;
boolean currentPauseButton = LOW;

const int displayModeButton = 3;
boolean lastDisplayModeButton = LOW;
boolean currentDisplayModeButton = LOW;

const int revolutionButton = 4;
boolean lastRevolutionButton = LOW;
boolean currentRevolutionButton = LOW;

boolean startShown = HIGH;

boolean paused = LOW;
boolean pausedShown = LOW;
unsigned long pausedStartTime = 0;

boolean wheelTurningShown = LOW;
unsigned long wheelTurningStartTime = 0;

boolean cycleSafelyShown = LOW;
unsigned long cycleSafelyStartTime = 0;

unsigned long lastRevolutionStartTime = 0;
unsigned long revolutionTime = 0;

int currentDisplayMode = 0;
int showLap = 0;
int lapCurrentlyShown = 100;
int currentLap = 0;

float currentDistance;
unsigned long currentDuration;
int currentMaximumKPH;
int currentAverageKPH;
int currentKPH;

float arrayDistance[100];
unsigned long arrayDuration[100];
int arrayMaximumKPH[100];
int arrayAverageKPH[100];

unsigned long revolutionCount = 0;
unsigned long currentTime = 0;
unsigned long lapStartTime = 0;
```

```
float km = 0.00;
float kph = 0.00;
int intHours;
int intMinutes;
int intSeconds;

unsigned long milliSecondsInSecond = 1000;
unsigned long milliSecondsInMinute = 60000;
unsigned long milliSecondsInHour = 3600000;

void setup()
{

  pinMode (revolutionButton, INPUT);
  pinMode (pauseButton, INPUT);
  pinMode (displayModeButton, INPUT);

  arrayMaximumKPH[0] = 0;

  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(2, 0);
  lcd.print("PRESS BUTTON");
  lcd.setCursor(4, 1);
  lcd.print("TO START");

}

void loop() {

  currentTime = millis();

  currentRevolutionButton = debounce(lastRevolutionButton,
revolutionButton);
  if (lastRevolutionButton == HIGH && currentRevolutionButton == LOW) {

    if (!startShown && !paused) {

      revolutionCount++;

      lcd.setCursor(0, 0);
      lcd.print("+");
      wheelTurningShown = HIGH;
      wheelTurningStartTime = currentTime;

      if (lastRevolutionStartTime > 0) {

        revolutionTime = currentTime - lastRevolutionStartTime;

        kph = (3600000 / revolutionTime) * bicycleWheelCircumference / 1000;
```

```
      currentKPH = kph;

      if (currentMaximumKPH < currentKPH) {
        currentMaximumKPH = currentKPH;
      }
    }
    lastRevolutionStartTime = currentTime;
  }
}
lastRevolutionButton = currentRevolutionButton;

currentPauseButton = debounce(lastPauseButton, pauseButton);
if (lastPauseButton == LOW && currentPauseButton == HIGH) {

  if (startShown) {

    startShown = LOW;

    showCycleSafely();
    cycleSafelyShown = HIGH;
    cycleSafelyStartTime = currentTime;

    currentLap = 1;
    resetLapVariables();
    currentDisplayMode = 1;

  }
  else {

    if (paused) {

      paused = LOW;

      showCycleSafely();
      cycleSafelyShown = HIGH;
      cycleSafelyStartTime = currentTime;

      currentLap++;

      if (currentLap > 99) {
        currentLap = 99;
        lapCurrentlyShown = 100;
      }

      resetLapVariables();
      currentDisplayMode = 1;
    }

    else {

      paused = HIGH;
```

```
        currentDuration = currentTime - lapStartTime;

        if (currentDuration < 2000) {
          currentLap--;
        }
        else {

          if (revolutionCount > 0) {
            currentDistance = revolutionCount * bicycleWheelCircumference /
1000;
            currentAverageKPH = currentDistance * 3600000 / currentDuration;
          }

          arrayDistance[currentLap] = currentDistance;
          arrayDuration[currentLap] = currentDuration;
          arrayAverageKPH[currentLap] = currentAverageKPH;
          arrayMaximumKPH[currentLap] = currentMaximumKPH;

          arrayDistance[0] = arrayDistance[0] + currentDistance;
          arrayDuration[0] = arrayDuration[0] + currentDuration;
          arrayAverageKPH[0] = arrayDistance[0] * 3600000 /
arrayDuration[0];
          if (currentMaximumKPH > arrayMaximumKPH[0]) {
            arrayMaximumKPH[0] = currentMaximumKPH;
          }
        }

        cycleSafelyShown = LOW;

        showPaused();
        pausedShown = HIGH;
        pausedStartTime = currentTime;

        showLap = currentLap;
        currentDisplayMode = 3;

        lapCurrentlyShown = 100;
      }
    }
  }
  lastPauseButton = currentPauseButton;

  currentDisplayModeButton = debounce(lastDisplayModeButton,
displayModeButton);
  if (lastDisplayModeButton == LOW && currentDisplayModeButton == HIGH) {

    if (startShown) {

      startShown = LOW;
```

```
      showCycleSafely();
      cycleSafelyShown = HIGH;
      cycleSafelyStartTime = currentTime;

      currentLap = 1;
      resetLapVariables();
      currentDisplayMode = 1;

    }
    else {

      if (!cycleSafelyShown && !pausedShown) {

        if (!paused) {

          if (currentDisplayMode == 1) {
            currentDisplayMode = 2;
          }
          else {
            currentDisplayMode = 1;
          }

          showLabels(currentDisplayMode);
        }

        else {
          currentDisplayMode = 3;
          showLap++;
          if (showLap > currentLap) {
            showLap = 0; // Show totals
          }
        }
      }
    }
  }
  lastDisplayModeButton = currentDisplayModeButton;

  if (wheelTurningShown && !startShown && !paused && (currentTime >=
(wheelTurningStartTime + 250))) {
    wheelTurningShown = LOW;
    lcd.setCursor(0, 0);
    lcd.print(" ");
  }

  if (!startShown && !paused && (currentTime >= (lastRevolutionStartTime +
10000)) && currentKPH > 0) {
    currentKPH = 0;
  }

  if (cycleSafelyShown && (currentTime >= (cycleSafelyStartTime + 2000))) {
    cycleSafelyShown = LOW;
```

```
    showLabels(currentDisplayMode);
  }

  if (pausedShown && (currentTime >= (pausedStartTime + 2000))) {
    pausedShown = LOW;
    showLabels(currentDisplayMode);
  }

  if (!startShown && !paused) {

    currentDuration = currentTime - lapStartTime;

    if (revolutionCount > 0) {
      currentDistance = revolutionCount * bicycleWheelCircumference / 1000;

      currentAverageKPH = currentDistance * 3600000 / currentDuration;
    }
  }

  if (!startShown && !cycleSafelyShown && !pausedShown) {

    if (currentDisplayMode < 3) {

      lcd.setCursor(1, 0);
      lcd.print(currentDistance);
      lcd.print(" km");

      lcd.setCursor(14, 0);
      if (currentKPH < 10) {
        lcd.print(" ");
      }
      lcd.print(currentKPH);

      computeHMS(currentDuration);
      lcd.setCursor(1, 1);
      if (intHours < 10) {
        lcd.print("0");
      }
      lcd.print(intHours);

      lcd.print(":");
      if (intMinutes < 10) {
        lcd.print("0");
      }
      lcd.print(intMinutes);

      lcd.print(":");
      if (intSeconds < 10) {
        lcd.print("0");
      }
      lcd.print(intSeconds);
```

```
    lcd.setCursor(12, 1);
    lcd.print("A");

    if (currentDisplayMode == 1) {
      lcd.setCursor(12, 1);
      lcd.print("A");
      lcd.setCursor(14, 1);
      if (currentAverageKPH < 10) {
        lcd.print(" ");
      }
      lcd.print(currentAverageKPH);
    }
    else {
      lcd.setCursor(12, 1);
      lcd.print("M");
      lcd.setCursor(14, 1);
      if (currentMaximumKPH < 10) {
        lcd.print(" ");
      }
      lcd.print(currentMaximumKPH);
    }
  }

  else {

    if (lapCurrentlyShown != showLap) {

      lapCurrentlyShown = showLap;

      lcd.clear();

      lcd.setCursor(0, 0);
      if (showLap == 0) {
        lcd.print("T ");
      }
      else {
        lcd.print(showLap);
      }

      lcd.setCursor(3, 0);
      lcd.print("Avg");
      lcd.setCursor(7, 0);
      lcd.print(arrayAverageKPH[showLap]);
      if (arrayAverageKPH[showLap] < 10) {
        lcd.print(" ");
      }

      lcd.setCursor(10, 0);
      lcd.print("Max");
      lcd.setCursor(14, 0);
```

```
        lcd.print(arrayMaximumKPH[showLap]);
        if (arrayMaximumKPH[showLap] < 10) {
          lcd.print(" ");
        }

        lcd.setCursor(0, 1);
        lcd.print("        ");
        lcd.setCursor(0, 1);
        lcd.print(arrayDistance[showLap]);

        computeHMS(arrayDuration[showLap]);
        lcd.setCursor(8, 1);
        if (intHours < 10) {
          lcd.print("0");
        }
        lcd.print(intHours);

        lcd.print(":");

        if (intMinutes < 10) {
          lcd.print("0");
        }
        lcd.print(intMinutes);

        lcd.print(":");

        if (intSeconds < 10) {
          lcd.print("0");
        }
        lcd.print(intSeconds);
      }
    }
  }
}

void computeHMS(unsigned long duration) {

  float floatHours;
  float floatMinutes;
  float floatSeconds;

  intHours = 0;
  intMinutes = 0;
  intSeconds = 0;

  if (duration >= 1000) {
      floatSeconds = duration / milliSecondsInSecond % 60;
      intSeconds = floatSeconds;

      floatMinutes = duration / milliSecondsInMinute % 60;
      intMinutes = floatMinutes;
```

```
      floatHours = duration / milliSecondsInHour % 24;
      intHours = floatHours;
  }
}

void resetLapVariables() {
  revolutionCount = 0;

  lapStartTime = currentTime;

  currentDistance = 0;
  currentDuration = 0;
  currentMaximumKPH = 0;
  currentAverageKPH = 0;
}

void showCycleSafely() {
  lcd.clear();
  lcd.setCursor(5, 0);
  lcd.print("CYCLE");
  lcd.setCursor(4, 1);
  lcd.print("SAFELY!");
}

void showPaused() {
  lcd.clear();
  lcd.setCursor(4, 0);
  lcd.print("PAUSED!");
}

void showLabels(int currentDisplayMode) {

  lcd.clear();
  switch (currentDisplayMode)     {
  case 1:
    lcd.setCursor(12, 0);
    lcd.print("S");
    lcd.setCursor(12, 1);
    lcd.print("A");
    break;
  case 2:
    lcd.setCursor(12, 0);
    lcd.print("S");
    lcd.setCursor(12, 1);
    lcd.print("M");
    break;
  }
}

boolean debounce(boolean last, int pin)
```

```
{
  boolean current = digitalRead(pin);
  if (last != current) {
    delay(5);
    current = digitalRead(pin);
  }
  return current;
}
```

Rezultate Obţinute

Care au fost rezultatele obţinute în urma realizării proiectului vostru.
Concluzii
Download

O arhivă (sau mai multe dacă este cazul) cu fişierele obţinute în urma realizării proiectului: surse, scheme, etc. Un fişier README, un ChangeLog, un script de compilare şi copiere automată pe uC crează întotdeauna o impresie bună .

Fişierele se încarcă pe wiki folosind facilitatea **Add Images or other files**. Namespace-ul în care se încarcă fişierele este de tipul **:pm:prj20??:c?** sau **:pm:prj20??:c?:nume_student** (dacă este cazul). **Exemplu:** Dumitru Alin, 331CC → **:pm:prj2009:cc:dumitru_alin**.

Jurnal

Puteți avea și o secțiune de jurnal în care să poată urmări asistentul de proiect progresul proiectului.
Bibliografie/Resurse

- WIRE Library
- EEPROM Library
- RTC Library
- Liquid Crystal

Export to PDF