Hourglass

Autor : Radu-Alexandru Stancu

Introducere

In cadrul acestui proiect voi realiza o clepsidra creata prin conectarea a doua matrici de leduri, puse cap in cap pe diagonala. Pentru a verifica conditia de inclinare a clepsidrei voi folosi un accelerometru adafruit ADXL345.

Nisipul va fi simulat prin ledurile de pe matricile 8×8.

Lista de piese

1x Arduino Breadboard UNO, 1x Accelerometru Adafruit ADXL345, 2x Matrici LED 8×8, 1x switch, 1x baterie de 9 V.

Hardware Design

Aceasta este o afisare rudimentara a componentelor si cum vor fi legate intre ele. Inafara de accelerometrul de pe breadboard celelalte componente nu sunt permanente si probabil nu vor aparea in cadrul proiectului final.

Aceasta este schema imbunatatita a proiectului. Schema a fost realizata in programul Fritzing.

Aceasta este schema electrica a proiectului, realizata tot in programul Fritzing.

Software Design

In cadrul realizarii software am folosit functionalitatile de baza ale librariilor exstente in Arduino IDE. Pe langa acestea am inclus si o librarie externa LedControl.h care ma ajuta cu manipularea ledurilor in matricile de 8×8 si una pentru accelerometrul ADX345 datorita functiilor implementate in cadrul acesteia pentru controlul si calibrarea axelor pe care acesta le foloseste cand determinam cum ar trebuii sa “curga” nisipul pe matricile led. Mai jos este atasat codul sursa al proiectului.

```
#include "Arduino.h"
#include "LedControl.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>

Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified();


class NonBlockDelay {
    unsigned long iTimeout;
  public:

    void Delay (unsigned long t)
    {
      iTimeout = millis() + t;
      return;
    }

    bool Timeout (void)
    {
      return (iTimeout < millis());
    }

    unsigned long Time(void)
    {
       return iTimeout;
    }
};


#define MATRIX_A  1
#define MATRIX_B  0

// Values are 260/330/400
#define ACC_THRESHOLD_LOW 300
#define ACC_THRESHOLD_HIGH 360

// Matrix
#define PIN_DATAIN 12
```

```
#define PIN_CLK 11
#define PIN_LOAD 10

// Accelerometer
#define PIN_X A4
#define PIN_Y A5

// Rotary Encoder
#define PIN_ENC_1 3
#define PIN_ENC_2 2
#define PIN_ENC_BUTTON 7

#define PIN_BUZZER 14

// This takes into account how the matrixes are mounted
#define ROTATION_OFFSET 90

// in milliseconds
#define DEBOUNCE_THRESHOLD 500

#define DELAY_FRAME 100

#define DEBUG_OUTPUT 1

#define MODE_HOURGLASS 0
#define MODE_SETMINUTES 1
#define MODE_SETHOURS 2

byte delayHours = 0;
byte delayMinutes = 1;
int mode = MODE_HOURGLASS;
int gravity;
LedControl lc = LedControl(PIN_DATAIN, PIN_CLK, PIN_LOAD, 2);
NonBlockDelay d;
int resetCounter = 0;
bool alarmWentOff = false;


/**
 * Get delay between particle drops (in seconds)
 */
long getDelayDrop() {
  // since we have exactly 60 particles we don't have to multiply by 60 and
then divide by the number of particles again :)
  return delayMinutes + delayHours * 60;
}


#if DEBUG_OUTPUT
void printmatrix() {
  Serial.println(" 0123-4567 ");
```

```
  for (int y = 0; y<8; y++) {
    if (y == 4) {
      Serial.println("|----|----|");
    }
    Serial.print(y);
    for (int x = 0; x<8; x++) {
      if (x == 4) {
        Serial.print("|");
      }
      Serial.print(lc.getXY(0,x,y) ? "X" :" ");
    }
    Serial.println("|");
  }
  Serial.println("-----------");
}
#endif



coord getDown(int x, int y) {
  coord xy;
  xy.x = x-1;
  xy.y = y+1;
  return xy;
}
coord getLeft(int x, int y) {
  coord xy;
  xy.x = x-1;
  xy.y = y;
  return xy;
}
coord getRight(int x, int y) {
  coord xy;
  xy.x = x;
  xy.y = y+1;
  return xy;
}



bool canGoLeft(int addr, int x, int y) {
  if (x == 0) return false; // not available
  return !lc.getXY(addr, getLeft(x, y)); // you can go there if this is
empty
}
bool canGoRight(int addr, int x, int y) {
  if (y == 7) return false; // not available
  return !lc.getXY(addr, getRight(x, y)); // you can go there if this is
empty
}
bool canGoDown(int addr, int x, int y) {
```

```
  if (y == 7) return false; // not available
  if (x == 0) return false; // not available
  if (!canGoLeft(addr, x, y)) return false;
  if (!canGoRight(addr, x, y)) return false;
  return !lc.getXY(addr, getDown(x, y)); // you can go there if this is
empty
}



void goDown(int addr, int x, int y) {
  lc.setXY(addr, x, y, false);
  lc.setXY(addr, getDown(x,y), true);
}
void goLeft(int addr, int x, int y) {
  lc.setXY(addr, x, y, false);
  lc.setXY(addr, getLeft(x,y), true);
}
void goRight(int addr, int x, int y) {
  lc.setXY(addr, x, y, false);
  lc.setXY(addr, getRight(x,y), true);
}


int countParticles(int addr) {
  int c = 0;
  for (byte y=0; y<8; y++) {
    for (byte x=0; x<8; x++) {
      if (lc.getXY(addr, x, y)) {
        c++;
      }
    }
  }
  return c;
}


bool moveParticle(int addr, int x, int y) {
  if (!lc.getXY(addr,x,y)) {
    return false;
  }

  bool can_GoLeft = canGoLeft(addr, x, y);
  bool can_GoRight = canGoRight(addr, x, y);

  if (!can_GoLeft && !can_GoRight) {
    return false; // we're stuck
  }

  bool can_GoDown = canGoDown(addr, x, y);
```

```
  if (can_GoDown) {
    goDown(addr, x, y);
  } else if (can_GoLeft&& !can_GoRight) {
    goLeft(addr, x, y);
  } else if (can_GoRight && !can_GoLeft) {
    goRight(addr, x, y);
  } else if (random(2) == 1) { // we can go left and right, but not down
    goLeft(addr, x, y);
  } else {
    goRight(addr, x, y);
  }
  return true;
}



void fill(int addr, int maxcount) {
  int n = 8;
  byte x,y;
  int count = 0;
  for (byte slice = 0; slice < 2*n-1; ++slice) {
    byte z = slice<n ? 0 : slice-n + 1;
    for (byte j = z; j <= slice-z; ++j) {
      y = 7-j;
      x = (slice-j);
      lc.setXY(addr, x, y, (++count <= maxcount));
    }
  }
}



/**
 * Detect orientation using the accelerometer
 *
 *     | up | right | left | down |
 * ---------------------------------
 * 400 |    |       | y    | x    |
 * 330 | y  | x     | x    | y    |
 * 260 | x  | y     |      |      |
 */
int getGravity() {
  int x = analogRead(PIN_X);
  int y = analogRead(PIN_Y);
  sensors_event_t event;
  accel.getEvent(&event);
  x = event.acceleration.x;
  y = event.acceleration.y;

  if ( (x > 7) && (abs(y) < 6) ) {return 180;}
  if ( (abs(x) < 6) && (y < -7) ) {return 270;}
```

```
  if ( (abs(x) < 6) && (y > 7) ) {return 90;}
  if ( (x < -7) && (abs(y) < 6) ) {return 0;}

}



int getTopMatrix() {
  return (getGravity() == 90) ? MATRIX_A : MATRIX_B;
}
int getBottomMatrix() {
  return (getGravity() != 90) ? MATRIX_A : MATRIX_B;
}




void resetTime() {
  for (byte i=0; i<2; i++) {
    lc.clearDisplay(i);
  }
  fill(getTopMatrix(), 60);
  d.Delay(getDelayDrop() * 1000);
}




/**
 * Traverse matrix and check if particles need to be moved
 */
bool updateMatrix() {
  int n = 8;
  bool somethingMoved = false;
  byte x,y;
  bool direction;
  for (byte slice = 0; slice < 2*n-1; ++slice) {
    direction = (random(2) == 1); // randomize if we scan from left to right
or from right to left, so the grain doesn't always fall the same direction
    byte z = slice<n ? 0 : slice-n + 1;
    for (byte j = z; j <= slice-z; ++j) {
      y = direction ? (7-j) : (7-(slice-j));
      x = direction ? (slice-j) : j;
      // for (byte d=0; d<2; d++) { lc.invertXY(0, x, y); delay(50); }
      if (moveParticle(MATRIX_B, x, y)) {
        somethingMoved = true;
      };
      if (moveParticle(MATRIX_A, x, y)) {
        somethingMoved = true;
      }
    }
  }
  return somethingMoved;
}
```

```
/**
 * Let a particle go from one matrix to the other
 */
boolean dropParticle() {
  if (d.Timeout()) {
    d.Delay(getDelayDrop() * 1000);
    if (gravity == 0 || gravity == 180) {
      if ((lc.getRawXY(MATRIX_A, 0, 0) && !lc.getRawXY(MATRIX_B, 7, 7)) ||
          (!lc.getRawXY(MATRIX_A, 0, 0) && lc.getRawXY(MATRIX_B, 7, 7))
      ) {
        // for (byte d=0; d<8; d++) { lc.invertXY(0, 0, 7); delay(50); }
        lc.invertRawXY(MATRIX_A, 0, 0);
        lc.invertRawXY(MATRIX_B, 7, 7);
        tone(PIN_BUZZER, 440, 10);
        return true;
      }
    }
  }
  return false;
}



void alarm() {
  for (int i=0; i<5; i++) {
    tone(PIN_BUZZER, 440, 200);
    delay(1000);
  }
}



void resetCheck() {
  int z = analogRead(A3);
  if (z > ACC_THRESHOLD_HIGH || z < ACC_THRESHOLD_LOW) {
    resetCounter++;
    Serial.println(resetCounter);
  } else {
    resetCounter = 0;
  }
  if (resetCounter > 20) {
    Serial.println("RESET!");
    resetTime();
    resetCounter = 0;
  }
}
```

```
void displayLetter(char letter, int matrix) {
  // Serial.print("Letter: ");
  // Serial.println(letter);
  lc.clearDisplay(matrix);
  lc.setXY(matrix, 1,4, true);
  lc.setXY(matrix, 2,3, true);
  lc.setXY(matrix, 3,2, true);
  lc.setXY(matrix, 4,1, true);

  lc.setXY(matrix, 3,6, true);
  lc.setXY(matrix, 4,5, true);
  lc.setXY(matrix, 5,4, true);
  lc.setXY(matrix, 6,3, true);

  if (letter == 'M') {
    lc.setXY(matrix, 4,2, true);
    lc.setXY(matrix, 4,3, true);
    lc.setXY(matrix, 5,3, true);
  }
  if (letter == 'H') {
    lc.setXY(matrix, 3,3, true);
    lc.setXY(matrix, 4,4, true);
  }
}



void renderSetMinutes() {
  fill(getTopMatrix(), delayMinutes);
  displayLetter('M', getBottomMatrix());
}
void renderSetHours() {
  fill(getTopMatrix(), delayHours);
  displayLetter('H', getBottomMatrix());
}




void knobClockwise() {
  Serial.println("Clockwise");
  if (mode == MODE_SETHOURS) {
    delayHours = constrain(delayHours+1, 0, 64);
    renderSetHours();
  } else if(mode == MODE_SETMINUTES) {
    delayMinutes = constrain(delayMinutes+1, 0, 64);
    renderSetMinutes();
  }
  Serial.print("Delay: ");
  Serial.println(getDelayDrop());
```

```
}
void knobCounterClockwise() {
  Serial.println("Counterclockwise");
  if (mode == MODE_SETHOURS) {
    delayHours = constrain(delayHours-1, 0, 64);
    renderSetHours();
  } else if (mode == MODE_SETMINUTES) {
    delayMinutes = constrain(delayMinutes-1, 0, 64);
    renderSetMinutes();
  }
  Serial.print("Delay: ");
  Serial.println(getDelayDrop());
}



volatile int lastEncoded = 0;
volatile long encoderValue = 0;
long lastencoderValue = 0;
long lastValue = 0;
void updateEncoder() {
  int MSB = digitalRead(PIN_ENC_1); //MSB = most significant bit
  int LSB = digitalRead(PIN_ENC_2); //LSB = least significant bit

  int encoded = (MSB << 1) |LSB; //converting the 2 pin value to single
number
  int sum  = (lastEncoded << 2) | encoded; //adding it to the previous
encoded value

  if(sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011)
encoderValue--;
  if(sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000)
encoderValue++;

  // Serial.print("Value: ");
  // Serial.println(encoderValue);
  if ((encoderValue % 4) == 0) {
    int value = encoderValue / 4;
    if (value > lastValue) knobClockwise();
    if (value < lastValue) knobCounterClockwise();
    lastValue = value;
  }
  lastEncoded = encoded; //store this value for next time
}



/**
 * Button callback (incl. software debouncer)
 * This switches between the modes (normal, set minutes, set hours)
 */
```

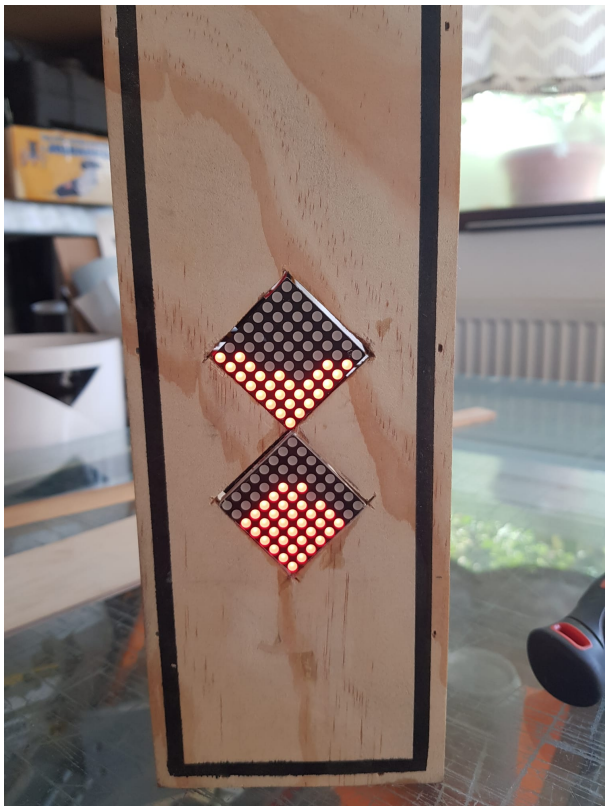
```
volatile unsigned long lastButtonPushMillis;
void buttonPush() {
  if((long)(millis() - lastButtonPushMillis) >= DEBOUNCE_THRESHOLD) {
    mode = (mode+1) % 3;
    Serial.print("Switched mode to: ");
    Serial.println(mode);
    lastButtonPushMillis = millis();

    if (mode == MODE_SETMINUTES) {
      lc.backup(); // we only need to back when switching from
MODE_HOURGLASS->MODE_SETMINUTES
      renderSetMinutes();
    }
    if (mode == MODE_SETHOURS) {
      renderSetHours();
    }
    if (mode == MODE_HOURGLASS) {
      lc.clearDisplay(0);
      lc.clearDisplay(1);
      lc.restore();
      resetTime();
    }
  }
}



/**
 * Setup
 */
void setup() {
  Serial.begin(9600);

  // while (!Serial) {
  //   ; // wait for serial port to connect. Needed for native USB
  // }

  if(!accel.begin())
   {
      Serial.println("No ADXL345 sensor detected.");
      while(1);
   }

  // setup rotary encoder
  pinMode(PIN_ENC_1, INPUT);
  pinMode(PIN_ENC_2, INPUT);
  pinMode(PIN_ENC_BUTTON, INPUT);
  digitalWrite(PIN_ENC_1, HIGH); //turn pullup resistor on
  digitalWrite(PIN_ENC_2, HIGH); //turn pullup resistor on
  digitalWrite(PIN_ENC_BUTTON, HIGH); //turn pullup resistor on
  attachInterrupt(digitalPinToInterrupt(PIN_ENC_1), updateEncoder, CHANGE);
```

```
  attachInterrupt(digitalPinToInterrupt(PIN_ENC_2), updateEncoder, CHANGE);
  attachInterrupt(digitalPinToInterrupt(PIN_ENC_BUTTON), buttonPush, RISING);

  // Serial.println(digitalPinToInterrupt(PIN_ENC_1));
  // Serial.println(digitalPinToInterrupt(PIN_ENC_2));
  // Serial.println(digitalPinToInterrupt(PIN_ENC_BUTTON));

  randomSeed(analogRead(A0));

  // init displays
  for (byte i=0; i<2; i++) {
    lc.shutdown(i,false);
    lc.setIntensity(i,0);
  }

  resetTime();
}



/**
 * Main loop
 */
void loop() {
  delay(DELAY_FRAME);

  sensors_event_t event;
  accel.getEvent(&event);

  // update the driver's rotation setting. For the rest of the code we
pretend "down" is still 0,0 and "up" is 7,7
  gravity = getGravity();
  lc.setRotation((ROTATION_OFFSET + gravity) % 360);

  // handle special modes
  if (mode == MODE_SETMINUTES) {
    renderSetMinutes(); return;
  } else if (mode == MODE_SETHOURS) {
    renderSetHours(); return;
  }

  // resetCheck(); // reset now happens when pushing a button
  bool moved = updateMatrix();
  bool dropped = dropParticle();

  // alarm when everything is in the bottom part
  if (!moved && !dropped && !alarmWentOff && (countParticles(getTopMatrix())
== 0)) {
    alarmWentOff = true;
    alarm();
  }
```

```
  // reset alarm flag next time a particle was dropped
  if (dropped) {
    alarmWentOff = false;
  }
}
```

Rezultate Obţinute

Mai jos voi atasa poza versiunii finale a proiectului si un videoclip pentru a-i demonstra functionalitatea.



https://www.youtube.com/shorts/NSxISB9ePCA

Concluzii

In concluzie, acest proiect a fost o oportunitate de a-mi testa abilitatile de “tinkering” cum imi place mie sa cred. Partea hardware a proiectului a durat in jur de 12 ore din simplul motiv ca a fost prima data cand am cositorit pini pe placute, lucru surprinzator de satisfacator cand vezi rezultatul final in fata. Timpul alocat research-ului si a dezvoltarii software a codului a durat undeva la 4 zile, in jur de 5-6 ore pe zi. Multe exemple de accelerometre in proiecte foloseau varianta veche a piesei pe care o aveam eu, astfel multe conexiuni difera, asa ca a trebuit sa caut data sheet-uri de componente si cum sa le folosesc. All in all, am descoperit o noua pasiune si sper ca in timp sa pot realiza proiecte din ce

in ce mai bune.

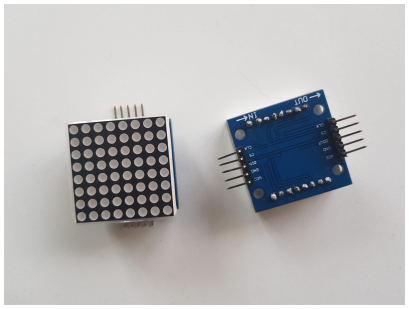
Download

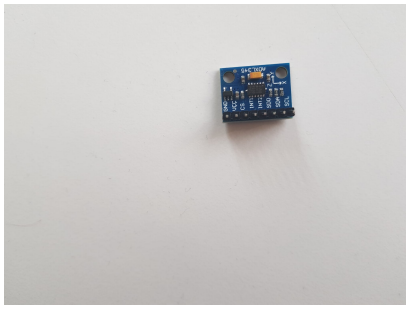
O arhivă (sau mai multe dacă este cazul) cu fişierele obţinute în urma realizării proiectului: surse, scheme, etc. Un fişier README, un ChangeLog, un script de compilare şi copiere automată pe uC crează întotdeauna o impresie bună .

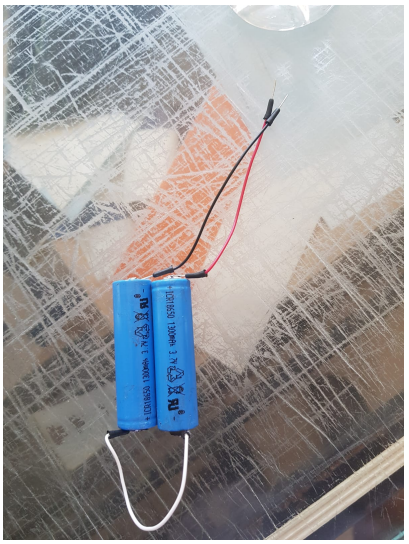
Fişierele se încarcă pe wiki folosind facilitatea **Add Images or other files**. Namespace-ul în care se încarcă fişierele este de tipul **:pm:prj20??:c?** sau **:pm:prj20??:c?:nume_student** (dacă este cazul). **Exemplu:** Dumitru Alin, 331CC → **:pm:prj2009:cc:dumitru_alin**.

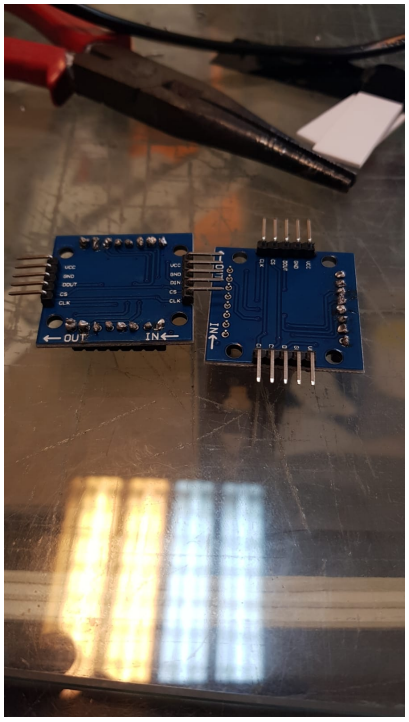
Jurnal

Aici voi arata progresul in cadrul lipirii cu fludor a pinilor pe piese si a bateriei externe pe care o folosesc.

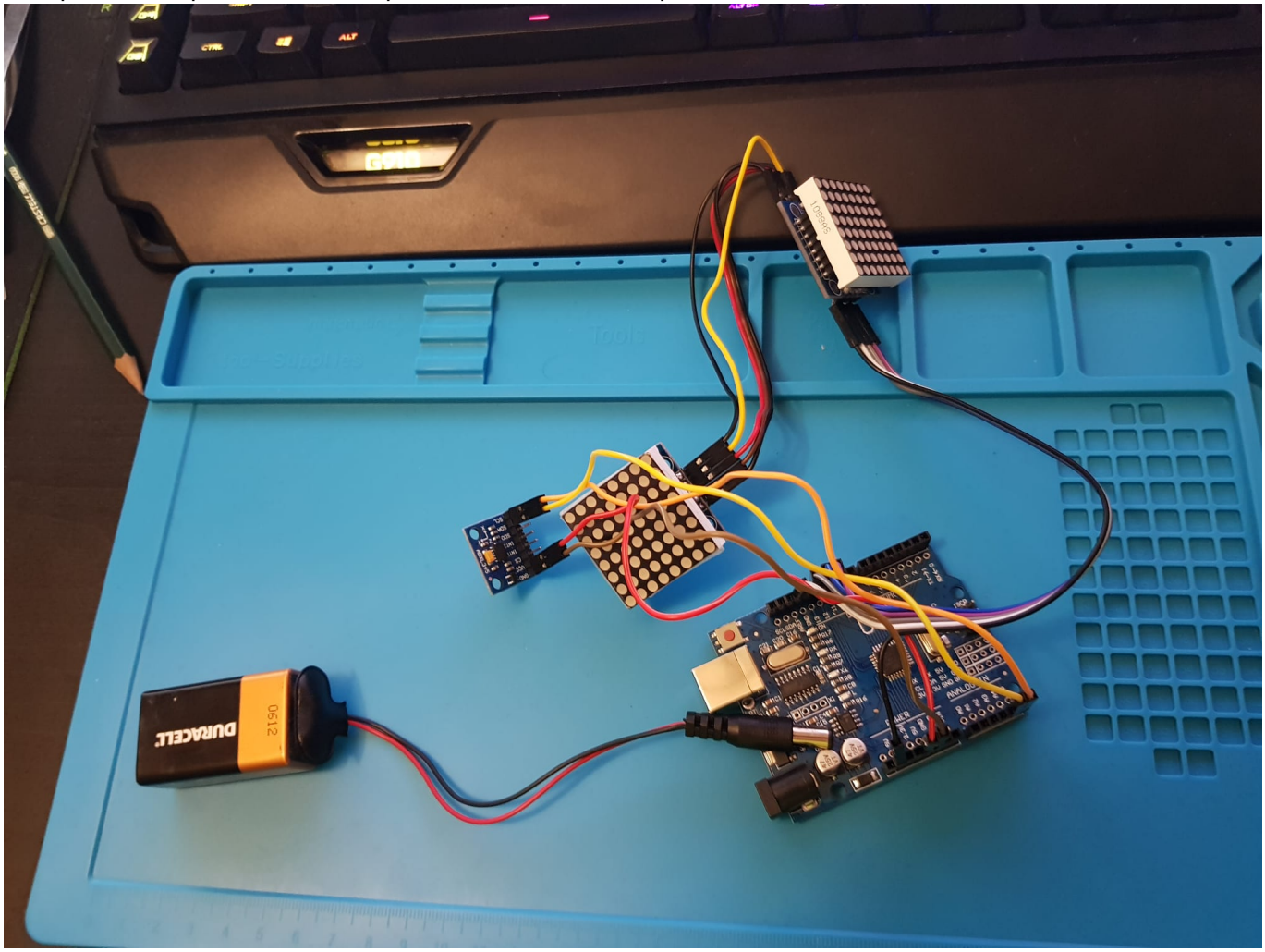








Matricile de leduri, accelerometrul si bateria de 7.4V inainte de izolare cu banda. Ultima poza din dreapta este o poza din cadrul primei incercari de a lipi cu fludor.



Acesta este cablajul complet al proiectului.

Bibliografie/Resurse

- 8x8 LED Matrix
- 8x8 LED Matrix generator
- adxl345 interface
- adxl345 track orientation

Export to PDF