Projekt z mikrokontrolerów - sprawozdanie

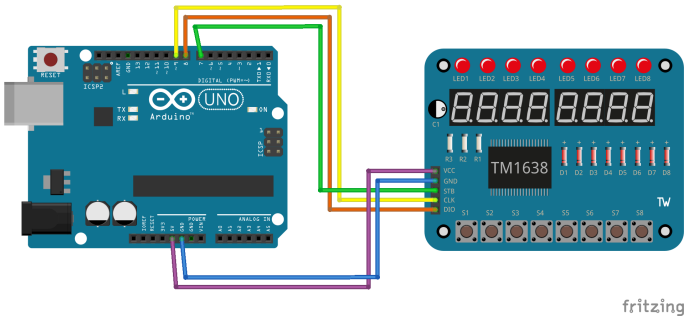
1 czerwca 2017

1 Informacje ogólne

1. Temat: Łamanie kodu
2. Numer grupy: 1b
3. Sklad osobowy: Sebastian Domarecki, Yurii Piets
4. Kierunek: informatyka
5. Rok studiów: II
6. Rok akademicki: 2016-2017

2 Opis działania

2.1 Schemat połączenia



TM1638	Arduino
VCC	5V
GND	GND
STB	Digital 7
CLK	Digital 9
DIO	Digital 8

2.2 Opis algorytmu

Schemat działania algorytmu opiera się w głównej mierze na zredukowanej zewnętrznej bibliotece TM1638.h udostępnianej na Githubie.

Program po inicjalizacji ekranu stringiem ośmiu spacji, aż do zakończenia łamania kodu znajduje się w pętli iterowanej po numerze dekodowanej cyfry.
Wewnątrz niej kolejna pętla for TIMES razy zmienia wyświetlane wartości niezdekodowanych cyfr na wygenerowane przypadowe dozwolone.
Za każdą iteracją drugiej pętli sprawdzane jest w sposób ciągły przez DISP czasu czy został wciśnięty przycisk, bądź wprowadzona komenda przez port szeregowy.

Po jej opuszczeniu włączany jest kolejny led oraz kopiowana wartość zdekodowanej cyfry z CODE do tablicy display do wyświetlenia.

HandleClick w zależności od stanu programu pozwala na jego zmianę po wykryciu sygnału z odpowiedniego przycisku. Następnie program czeka na zwolnienie wszystkich przycisków.

ReadInput najpierw czeka na zapełnienie buforu, następnie wykrywa rodzaj komendy i przekierowuje ruch do odpowiedniej funkcji. Dla błędnego polecenia wypisywany jest stosowny komunikat.

2.3 Elementy programu

2.3.1 Zmienne

CODE,TIMES,DISP - zgodnie z wytycznymi.

module - obiekt klasy TM1638 udostępniający prosty interfejs do obsługi płytki

state - zmienna typu wyliczeniowego wyświetlająca stan programu - dostępne IN_PROGRESS, WAITING, FINISHED, RESET

2.3.2 Stałe

strobe,clock,data - piny Arduino łączące się z płytką TM1638

DISPLAY_SIZE - wielkość wyświetlacza płytki

allowed_chars - tablica zawierająca dozwolone do wyświetlenia znaki, wszystkie inne są odrzucone

2.3.3 Funkcje

void handleClick(states *) - obsługa przycisków

void readInput(states *) - obsługa poleceń z łącza szeregowego

boolean initCode() , initTimes(), initDisp() - wywoływane przy readInput dla pierwszego przesłanego znaku odpowiednio C, N lub D, sczytują z łącza szeregowego nową wartość parametru pracy programu

boolean isAllowed(char) - pomocnicza funkcja sprawdzająca czy argument jest dozwolonym znakiem

inline void waitTillRelease() - pomocnicza funkcja czekająca na zwolnienie wszystkich przycisków

2.4 Używane biblioteki

Autorsko zredukowana wersja TM1638.h udostępniana na licencji GNU GPL v3 przez Ricardo Batista.
Link: https://github.com/rjbatista/tm1638-library

3 Listing kodu

```
1 #include <TM1638.h>

  // Parametry programu
```

```
char CODE[] = "3AE61Cb1";
byte TIMES = 10;
int DISP = 100;

// Ustawienia pinów
const byte strobe = 7;
const byte clock = 9;
const byte data = 8;

const byte DISPLAY_SIZE = 8;
const char allowedChars[] = {'0','1','2','3','4','5','6','7','8','9','A','b',
    'C','d','E','F'};

TM1638 module(data, clock, strobe);

//Cztery możliwe stany programu
typedef enum {IN_PROGRESS, WAITING, FINISHED, RESET} states;

//Definicje funkcji
void handleClick(states *);
void readInput(states *);
boolean initCode();
void initTimes();
void initDisp();
boolean isAllowed(char);
inline void waitTillRelease();

void setup() {
  Serial.begin(9600);
  module.clearDisplay();
}

void loop(){
  char *display = (char *) malloc(DISPLAY_SIZE + 1);
  byte leds = 0;
  module.setLEDs(leds);
  states state = IN_PROGRESS;

  for(byte i = 0; i < DISPLAY_SIZE; ++i){

    for(byte j = 0; j < TIMES; ++j) {

      for(byte k = i; k < DISPLAY_SIZE; ++k){
        // Niezdekodowana pozycja - przypadkowa wartość
        display[k] = allowedChars[random(0, 16)];
      }
      module.setDisplayToString(display);

      // Zmiany parametrów
      readInput(&state);
      long time = millis() + DISP; // Pobieraj info o przyciskach do tego
          czasu
      do
      {
        handleClick(&state);
```

```
      } while (time > millis());

      if(state == RESET){
        free(display);
        return;
      }
    }
    display[i] = CODE[i];

    leds = (leds << 1) +1; // leds*2 +1
    module.setLEDs(leds);
  }
  free(display);
  module.setDisplayToString(CODE); // Wynik łamania kodu
  // Zmiany parametrów po zakończeniu
  state = FINISHED;
  handleClick(&state);
}

void handleClick(states *state){
  byte key;

  switch(*state){
    case IN_PROGRESS:
      key = module.getButtons();
      if (key == 1) {
        *state = WAITING;
      }
      else break;
      waitTillRelease();
    case WAITING:
      for(key = 0; key != 1; key = module.getButtons()){
        delay(100);
      }
      *state = IN_PROGRESS;
      waitTillRelease();
      break;
    case FINISHED:
      for(key = 0; key != 2; key = module.getButtons()){
        readInput(state);
        delay(100);
        if(*state == RESET){
          break;
        }
      }
      break;
  }
}

void readInput(states *state){
  while(Serial.available() > 0){
    delay(100);
    char rc = Serial.read();
    switch(rc)
    {
```

```
    case 'C': // zmiana kodu
      Serial.println("C command");
      if(Serial.available() > 0) {
        if(initCode() != false){
          *state = RESET;
        }
      } else {
        Serial.println("Wrong format of command");
      }
      break;
    case 'N': // zmiana ilości iteracji
      Serial.println("N command");
      if(Serial.available() > 0) {
        initTimes();
      } else {
        Serial.println("Wrong format of command");
      }
      break;
    case 'D': // zmiana czasu wyświetlania
      Serial.println("D command");
      if(Serial.available() > 0) {
        initDisp();
      } else {
        Serial.println("Wrong format of command");
      }
      break;
    default:
      Serial.print("Wrong command: "); Serial.println(rc);
      break;
    }
  }
}

boolean initCode(){
  char newCode[DISPLAY_SIZE+1] = "\0";
  char rc;
  for(byte i = 0; i < 8 && Serial.available() > 0; ++i){
    rc = Serial.read();
    if(isAllowed(rc)){
      newCode[i] = rc;
    } else {
      Serial.print("Occured wrong value: "); Serial.println(rc);
      return false;
    }
  }
  memcpy(CODE, newCode, 8);
  Serial.println(CODE);
  return true;
}

void initTimes(){
  char newTimes[4] = "\0";
  char rc;
  for(byte i = 0; i < 3 && Serial.available() > 0; ++i){
    rc = Serial.read();
```

```
    if(rc >= '0' && rc <= '9'){
      newTimes[i] = rc;
    } else {
      Serial.print("Occured wrong value: "); Serial.println(rc);
      return false;
    }
  }
  sscanf(newTimes, "%d", &TIMES);
  Serial.println(TIMES);
  return true;
}

void initDisp(){
  char newTimes[5] = "\0";
  char rc;
  for(byte i = 0; i < 5 && Serial.available() > 0; ++i){
    rc = Serial.read();
    if(rc >= '0' && rc <= '9'){
      newTimes[i] = rc;
    } else {
      Serial.print("Occured wrong value: "); Serial.println(rc);
      return false;
    }
  }
  sscanf(newTimes, "%d", &DISP);
  Serial.println(DISP);
  return true;
}

boolean isAllowed(char value){
  for(byte i = 0 ; i < 16; ++i){
    if(allowedChars[i] == value){
      return true;
    }
  }
  return false;
}

inline void waitTillRelease() {
  while(module.getButtons() != 0) delay(20);
}
```