

Projekt z mikrokontrolerów - sprawozdanie

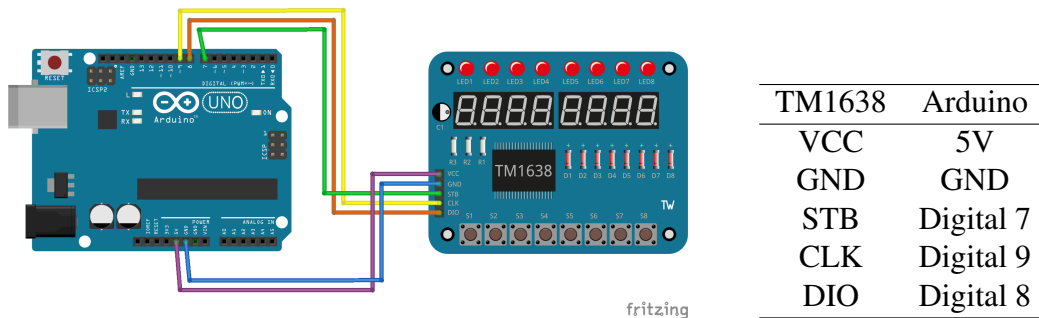
1 czerwca 2017

1 Informacje ogólne

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

2 Opis działania

2.1 Schemat połączenia



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 "00000000", następnie 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 przypadkowe dozwolone.

Za każdą iteracją drugiej pętli sprawdzane jest 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.

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, czytują 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 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";
5 | byte TIMES = 10;
   | int DISP = 100;
   |
   | // Ustawienia pinów
```

```

10 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'};
15 TM1638 module(data, clock, strobe);

//Cztery możliwe stany programu
typedef enum {IN_PROGRESS, WAITING, FINISHED, RESET} states;
20 //Definicje funkcji
void handleClick(states *);
void readInput(states *);
boolean initCode();
25 void initTimes();
void initDisp();
boolean isAllowed(char);
inline void waitTillRelease();

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

35 void loop() {
    char *display = (char *) malloc(DISPLAY_SIZE + 1);
    byte leds = 0;
    module.setLEDs(leds);
    states state = IN_PROGRESS;
40
    for(byte i = 0; i < DISPLAY_SIZE; ++i) {

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

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

50
            // Zmiany parametrów
            readInput(&state);
            long time = millis() + DISP; // Pobieraj info o przyciskach do tego
                czasu
            do
55            {
                handleClick(&state);
            } while (time > millis());

            if(state == RESET) {
60                free(display);
                return;

```

```

    }
    }
    display[i] = CODE[i];
65
    leds = (leds << 1) + 1; // leds*2 + 1
    module.setLEDs(leds);
}
free(display);
70
module.setDisplayToString(CODE); // Wynik łamania kodu
// Zmiany parametrów po zakończeniu
state = FINISHED;
handleClick(&state);
}
75
void handleClick(states *state) {
    byte key;

    switch(*state) {
80
        case IN_PROGRESS:
            key = module.getButtons();
            if (key == 1) {
                *state = WAITING;
            }
85
            else break;
            waitTillRelease();
        case WAITING:
            for(key = 0; key != 1; key = module.getButtons()) {
                delay(100);
90
            }
            *state = IN_PROGRESS;
            waitTillRelease();
            break;
        case FINISHED:
95
            for(key = 0; key != 2; key = module.getButtons()) {
                readInput(state);
                delay(100);
                if(*state == RESET) {
100
                    break;
                }
            }
            break;
    }
}
105
void readInput(states *state) {
    while(Serial.available() > 0) {
        delay(100);
        char rc = Serial.read();
110
        switch(rc)
        {
            case 'C': // zmiana kodu
                Serial.println("C command");
                if(Serial.available() > 0) {
115
                    if(initCode() != false) {
                        *state = RESET;
                    }
                }
            }
        }
    }
}

```

```

    }
    } else {
        Serial.println("Wrong format of command");
120    }
    break;
    case 'N': // zmiana ilości iteracji
        Serial.println("N command");
        if(Serial.available() > 0) {
125            initTimes();
        } else {
            Serial.println("Wrong format of command");
        }
        break;
130    case 'D': // zmiana czasu wyświetlania
        Serial.println("D command");
        if(Serial.available() > 0) {
            initDisp();
        } else {
135            Serial.println("Wrong format of command");
        }
        break;
    default:
        Serial.print("Wrong command: "); Serial.println(rc);
140    break;
    }
}
}

145 boolean initCode() {
    char newCode[DISPLAY_SIZE+1] = "\0";
    char rc;
    for(byte i = 0; i < 8 && Serial.available() > 0; ++i) {
        rc = Serial.read();
150        if(isAllowed(rc)) {
            newCode[i] = rc;
        } else {
            Serial.print("Occured wrong value: "); Serial.println(rc);
            return false;
155        }
    }
    memcpy(CODE, newCode, 8);
    Serial.println(CODE);
    return true;
160 }

void initTimes() {
    char newTimes[4] = "\0";
    char rc;
165    for(byte i = 0; i < 3 && Serial.available() > 0; ++i) {
        rc = Serial.read();
        if(rc >= '0' && rc <= '9') {
            newTimes[i] = rc;
        } else {
170            Serial.print("Occured wrong value: "); Serial.println(rc);
            return false;

```

```

    }
}
sscanf(newTimes, "%d", &TIMES);
175 Serial.println(TIMES);
    return true;
}

void initDisp() {
180 char newTimes[5] = "\0";
    char rc;
    for(byte i = 0; i < 5 && Serial.available() > 0; ++i) {
        rc = Serial.read();
        185 if(rc >= '0' && rc <= '9') {
            newTimes[i] = rc;
        } else {
            Serial.print("Occured wrong value: "); Serial.println(rc);
            return false;
        }
    }
190 }
    sscanf(newTimes, "%d", &DISP);
    Serial.println(DISP);
    return true;
}
195

boolean isAllowed(char value) {
    for(byte i = 0 ; i < 16; ++i) {
        if(allowedChars[i] == value) {
            return true;
        }
    }
    200 }
    return false;
}

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

```