Računalni sustavi stvarnog vremena

TUMAČENJE MORSEOVOG KODA

PROJEKTNI ZADATAK

Grupa:

Bruno Junaković

Ivan Gudelj

Osijek, 2022.

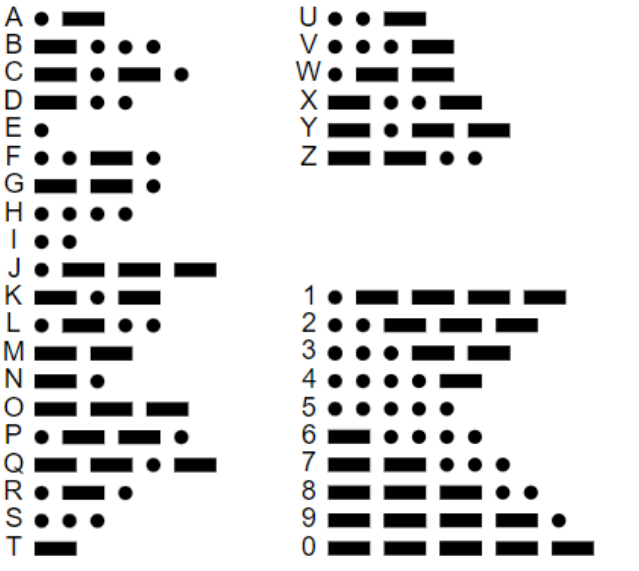
SADRŽAJ

1. Opis projektnog zadatka

Tema projekta je tumačenje Morseovog koda koristeći se emiterom (konkretno LED dioda) i prijemnikom (konkretno fotootpornik). U dokumentaciji je opisano projektiranje, izrada te programiranje sustava na mikroupravljaču Croduino 2 Basic.

1.1. Cilj projektnog zadatka

Cilj projekta je omogućiti korisniku unošenje poruke putem Serial Monitor-a u obliku signalizacije putem LED diode na jednom Croduino mikroupravljaču, dok se drugi Croduino mikroupravljač koristi za primanje signala uz pomoć fotootpornika. Nakon primanja i dekodiranja poruke, istu je bilo potrebno ispisati na LCD ekranu. Morseova abeceda se nalazi na Slici 1.1



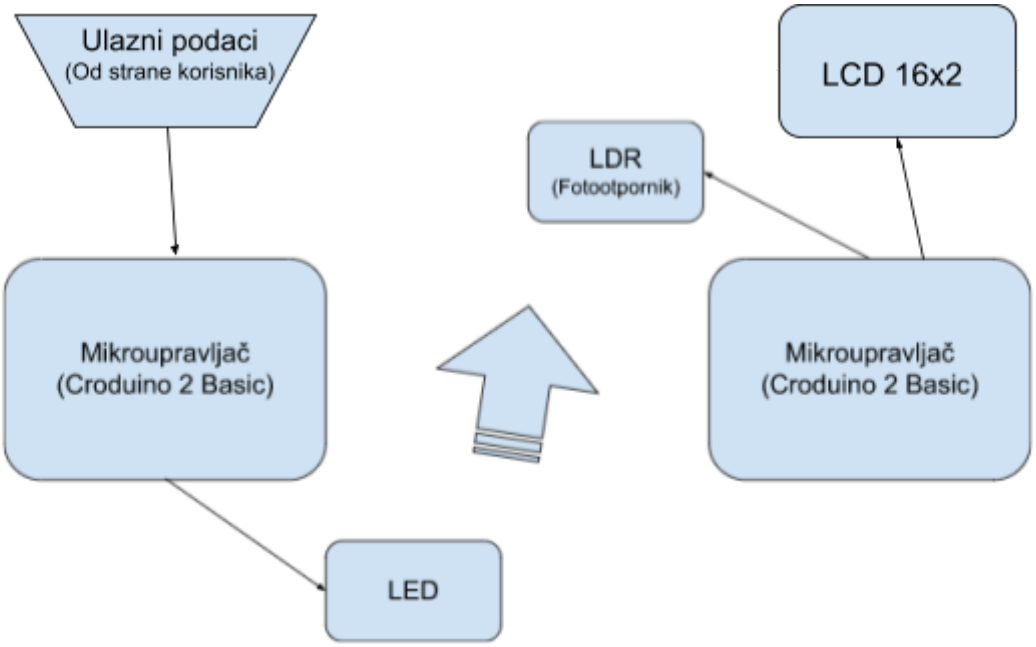
Slika 1.1 Međunarodno priznata Morseova abeceda

2. Prijedlog rješenja

2.1. Opis sustava

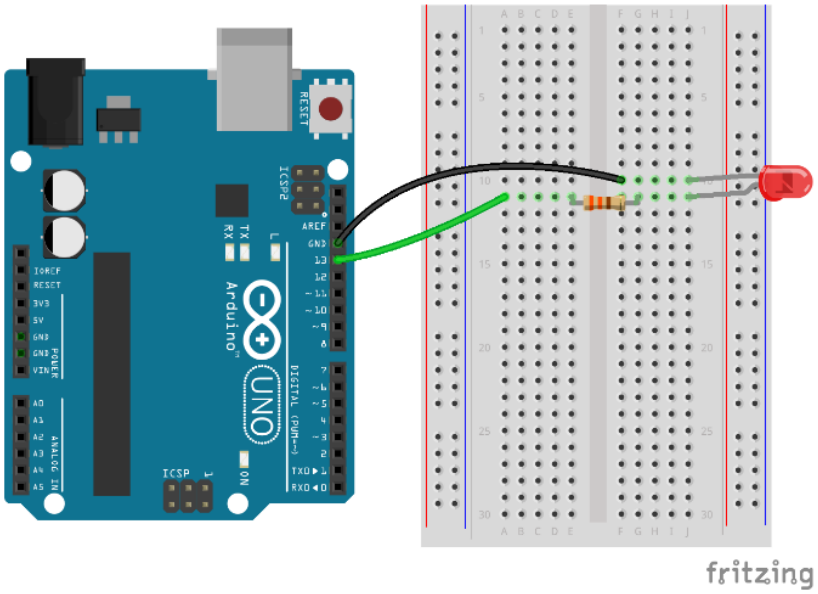
Program se treba odvijati tako što korisnik unosi poruku putem serijske komunikacije (*Serial Monitor*), analizirajući slovo po slovo prvi mikroupravljač generira niz signala u obliku kombinacije točaka i crta (ovisno o pročitanom znaku) koji se potom preko LED diode emitira. Drugi mikroupravljač koristi fotootpornik kako bi primio svaki znak te ga dekodirao. Nakon dekodiranja svakog znaka, znakovi se ispisuju na LCD zaslonu (IIC adapter + LCD 16x2).Na slici 2.1 se nalazi blok shema projektiranog sustava koji je sastavljen od sljedećih sklopova:

- 2x Croduino Basic2
- 1x 330Ω Otpornik (Povezan na LED)
- 1x 10kΩ Otpornik (Povezan na fotootpornik)
- 1x LED
- 1x Fotootpornik
- 1x *Breadboard*

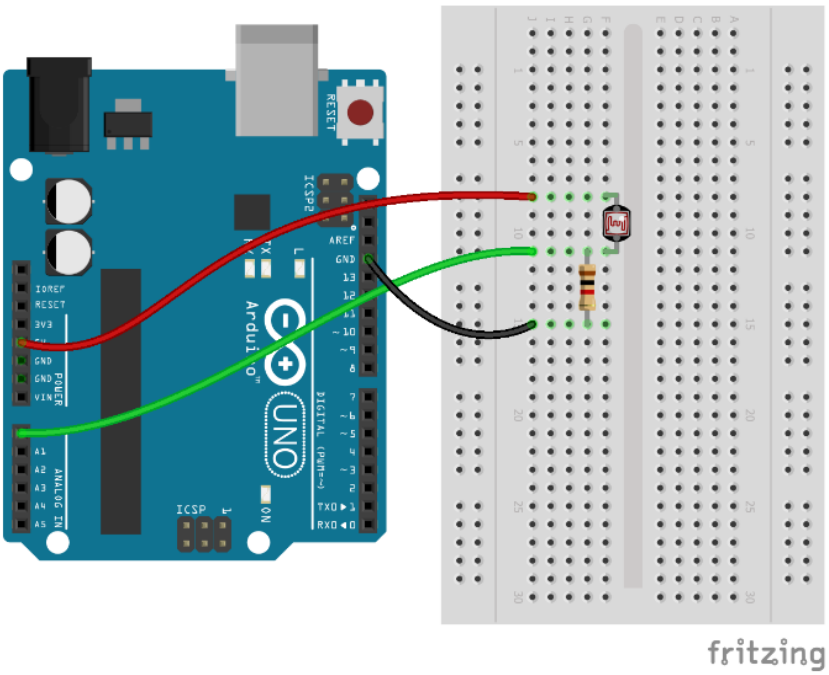


Slika 2.1. Blok shema sustava za generiranje i tumačenje Morseove abecede

Korištene već postojeće biblioteke u kodu su *Wire* (Biblioteka koja omogućava komunikaciju s I2C/TWI uređajima.) i *LiquidCrystal_I2C* (Biblioteka koja omogućava kontrolu I2C *display*-a). Na slici 2.2 i slici 2.3 se nalazi shema spajanja cijelog sustava.



Slika 2.2. Shema spajanja sustava za emitiranje Morseove abecede

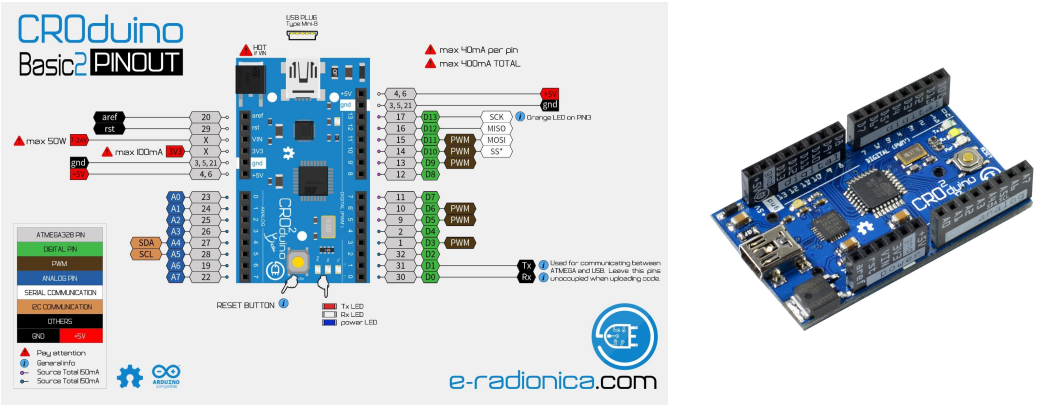


Slika 2.3. Shema spajanja sustava za tumačenje Morseove abecede

3. Opis rješenja

3.1. Glavni sustav (Croduino Basic2)

Croduino Basic2 je standardna hrvatska Arduino kompatibilna pločica. Kao i svi Croduini do sada, Basic2 je također 100% kompatibilan s Arduinom (Nano). Primjena Croduino Basic2 pločice u projektu je omogućila izvedbu jednostavnog sustava za generiranje i prijevod Morseove abecede. Croduino je univerzalni mikroupravljač zasnovan na Atmel tehnologiji te je idealan za razvoj upravljačke elektronike i robotike. Sustav je otvorenog koda temeljen na jednostavnoj razvojnoj pločici s ulazno/ izlaznim konektorima te besplatnom programskom podrškom s jednostavnim korisničkim sučeljem. Croduino ima ograničen broj digitalnih ulaza-izlaza i analognih ulaza. Karakterizira ga 14 digitalnih ulazno/ izlaznih pinova (od čega se 6 može koristiti kao PWM (eng. *Pulse Width Modulation*)), 8 analognih ulaza te 16 MHz oscilator, tipka za reset, ICSP pristupni kontakti i mini- B USB konektor. Programiranje uređaja se izvodi iz integriranog razvojnog okruženja koje postoji za Windows, Mac i Linux operativni sustav u programskom jeziku sličnom C- u.



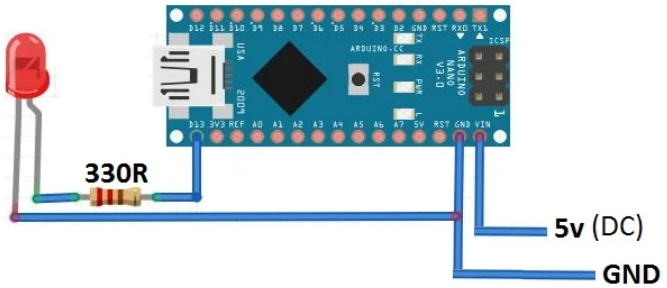
Slika 3.1. Croduino Basic2 [Izvor:https://github.com/e-radionicacom/Croduino-Basic2-Eagle-Files]

3.1.1. Analogni ulazi

Arduino sadrži 6 kanalni 10- bitni AD pretvornik. Korištenjem funkcije *analogRead*(), ulazni napon podijeli se između 0 i 5 V u integer vrijednosti između 0 i 1023. Funkcija *analogRead*() prima broj pina s kojeg se želi čitati te vraća integer vrijednosti između 0 i 1023. Dobiva se korak kvantizacije od: 5 V/ 1024 = 0.0049 V (4.9 mV). Za čitanje s analognog ulaza potrebno je 100 µs, stoga je najviša brzina čitanja 10000 puta u sekundi.

3.1.2. Digitalni ulazi/Izlazi

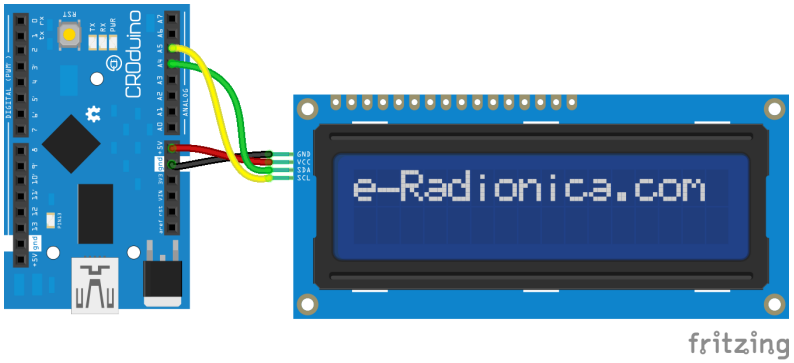
Funkcija *digitalRead*() čita vrijednosti sa određenog pina i vraća HIGH ili LOW, dok funkcija *digitalWrite*() upisuje vrijednost HIGH ili LOW na odabrani pin. U projektu smo koristili D13 (digitalni pin 13) za emitiranje Morseove abecede preko LED. Shema spajanja se nalazi na Slici 3.2.



Slika 3.2 Shema spajanja LED na digitalni pin 13

3.2. IIC LCD adapter + 16x2 adapter

LCD displej sadrži 16 izvoda. Pristup i kontrola displeja izvršava se putem 12 izvoda. Croduino Basic2 je ograničen s brojem digitalnih izvoda, stoga je odabrano rješenje sa tzv. I2C modulom (zbog uštede prostora). I2C je priključen na display te omogućuje značajnu uštedu broja pinova mikrokontrolera. Za potrebe programiranja korištena je *LiquidCrystal_I2C* biblioteka. Također potrebno je uključiti i *Wire.h*. I2C uređaji komuniciraju sa 2 signala SDA i SCL. Shema spajanja I2C modula i Croduino Basic2 prikazana je na slici 3.3.



Slika 3.3. Shema spajanja LCD 16x2 modula s I2C adapterom

Neke od osnovnih funkcija koje su korištene pri izradi projekta su:

lcd.print(); - ispisuje nešto na lcd ekranu. Ako se u zagradi nalazi varijabla, nju se ispisuje na lcd. Ako se u zagradi nalazi tekst pod navodnicima(npr. “Hello World!”), taj tekst se ispisuje na LCDu.

lcd.setCursor(x,y); - postavlja kursor(kao npr. onaj u Wordu) na određenu poziciju. Prva koordinata(x) označava poziciju gledajući s lijevo na desno(npr. 0 će biti prvi znak), dok druga koordinata označava poziciju gledajući od gore prema dolje(npr. 0 će biti prvi red, a 1 će biti drugi red). Primjer korištenja: lcd.setCursor(0,1); - postavlja kursor na početak drugoga reda. Sada možemo koristiti lcd.print(); kako bismo nešto ispisali u drugi red.

lcd.clear(); - briše sve što je ispisano na LCDu

lcd.home(); - odlazi s kursorom na početak(0,0), ali ne čisti ekran!(ne radi isto kao i *lcd.clear();*)

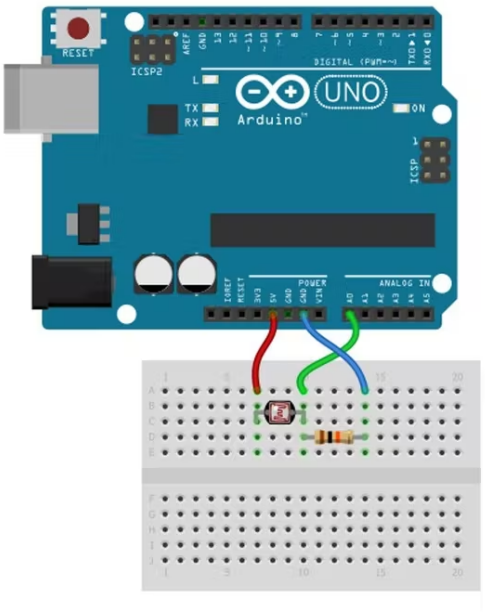
lcd.backlight(); - pali pozadinsko osvjetljenje

lcd.noBacklight(); - gasi pozadinsko osvjetljenje

3.3. Fotootpornik

Fotootpornik se izrađuje od poluvodiča s velikim električnim otporom. Ako svjetlo padne na fotootpornik, s dovoljno velikom frekvencijom (granična frekvencija), poluvodič ce upiti fotone svjetlosti i izbaciti elektrone, koji stvaraju električnu struju, u zatvorenom strujnom krugu.. Otpor mu se mijenja pod utjecajem svjetlosti koja pada na njega te se upravo zbog toga primjenjuje kao senzor za prijem Morseovog koda. U model je ugrađen fotootpornik LDR07 koji se koristi kao senzor te ovisno o vrijednosti okoline omogućuje razliku između upaljene i ugašene LED.

Fotootpornik je serijski spojen sa otpornikom od 10 kΩ te preko toga na analogni pin A0. Takav spoj čini naponsko dijelilo. Drugi kraj fotootpornika spaja se na 5 V.



Slika 3.4 Shema spajanja fotootpornika

3.4. Programska podrška za emitirajući mikroupravljač

```
#define DEBUG


const int UNIT_LENGTH = 100;  //Trajanje 1 unita
const int DIT_LENGTH = UNIT_LENGTH * 1;   //Točka
const int DAH_LENGTH = UNIT_LENGTH * 3;   //Crtica
const int ELEMENT_GAP = UNIT_LENGTH * 1;   //Vrijeme do idućeg elementa
const int SHORT_GAP = UNIT_LENGTH * 2;   //Vrijeme pauze ukoliko je slovo
const int MEDIUM_GAP = UNIT_LENGTH * 4;  //Vrijeme pauze ukoliko je
razmak
const int LEDPIN = 13;   //Pin za diodu koja emitira signal

/*Baza koja sadrži Morseovu abecedu pojedinog elementa (ukoliko element nije u
tablici ili ima vrijednost NULL ne može se prenijeti Morseovom abecedom)*/
const char* codeset[] = {
  /* ! */ "-.-.--",
  /* " */ ".-..-.",
  /* # */ NULL,
  /* $ */ "...-..-",
  /* % */ NULL,
```

```
/* & */ ".-...",
/* ' */ ".----.",
/* ( */ "-.--.",
/* ) */ "-.--.-",
/* * */ NULL,
/* + */ ".-.-.",
/* , */ "--..--",
/* - */ "-....-",
/* . */ ".-.-.-",
/* / */ "-..-.",
/* 0 */ "-----",
/* 1 */ ".----",
/* 2 */ "..---",
/* 3 */ "...--",
/* 4 */ "....-",
/* 5 */ ".....",
/* 6 */ "-....",
/* 7 */ "--...",
/* 8 */ "---..",
/* 9 */ "----.",
/* : */ "---...",
/* ; */ "-.-.-.",
/* < */ NULL,
/* = */ "-...-",
/* > */ NULL,
/* ? */ "..--..",
/* @ */ ".--.-.",
/* A */ ".-",
/* B */ "-...",
/* C */ "-.-.",
/* D */ "-..",
/* E */ ".",
/* F */ "..-.",
/* G */ "--.",
/* H */ "....",
/* I */ "..",
/* J */ ".---",
/* K */ "-.-",
/* L */ ".-..",
/* M */ "--",
/* N */ "-.",
/* O */ "---",
/* P */ ".--.",
/* Q */ "--.-",
/* R */ ".-.",
/* S */ "...",
/* T */ "-",
/* U */ "..-",
/* V */ "...-",
```

```
  /* W */ ".--",
  /* X */ "-..-",
  /* Y */ "-.--",
  /* Z */ "--..",
  /* [ */ NULL,
  /* \ */ NULL,
  /* ] */ NULL,
  /* ^ */ NULL,
  /* _ */ "..--.-"
 };


const char* getCode(char c) {  //Pronađi kod za predani znak u bazi i vrati ga
  // Prebaci u velika slova ako su mala poslana
  if ('a' <= c && c <= 'z') {
    c = c - ('a' - 'A');
  }
  if ('!' <= c && c <= '_') {
    return codeset[c - '!'];
  }
  return NULL;
}

void setup() {
  pinMode(LEDPIN, OUTPUT);   //Definiranje digitalnog pina za LED
  Serial.begin(19200);   //Pokretanje serijske komunikacije za emitiranje poruke
  Serial.println("Spremno!");
}

//Element znaka za emitiranje je točka
void dit()
{
#ifdef DEBUG
  Serial.print(".");
#endif
  digitalWrite(LEDPIN, HIGH);//Upali LED diodu na trajanje točke te ju opet ugasi
  delay(DIT_LENGTH);
  digitalWrite(LEDPIN, LOW);
  delay(ELEMENT_GAP);
}
//Element znaka za emitiranje je crtica
void dah()
{
#ifdef DEBUG
  Serial.print("-");
#endif
  digitalWrite(LEDPIN, HIGH);//Upali LED diodu na trajanje crtice te ju opet ugasi
  delay(DAH_LENGTH);
  digitalWrite(LEDPIN, LOW);
```

```
  delay(ELEMENT_GAP);
}

//Delay za idući znak ukoliko je trenutni znak bio slovo
void letter()
{
#ifdef DEBUG
  Serial.print(" ");
#endif
  delay(SHORT_GAP);
}
//Delay za idući znak ukoliko je trenutni znak bio razmak
void space()
{
#ifdef DEBUG
  Serial.println();
#endif
  delay(MEDIUM_GAP);
}


void play(const char * input) {
  int inputLength = strlen(input);

  for (int i = 0; i < inputLength; i++) {
    char c = input[i];

    if (c == ' ') {
      space();
    }
    else {
      const char* code = getCode(c);  //Dohvati kod iz baze u kojoj je abeceda

      if (code == NULL) {
        continue;
      }

      int codeLength = strlen(code);
  //Emitiranje pojedinog znaka svakog slova
      for (int j = 0; j < codeLength; j++) {
        if (code[j] == '.') {
          dit();
        }
        else if (code[j] == '-') {
          dah();
        }
      }

      letter();
```

```
    }
  }
}

// Petlja koja se vrti iznova
void loop() {
  while (Serial.available()){
    text = Serial.readString();   //Spremanje poruke od strane korisnika
    Serial.print(text);
    play(text);
    Serial.print("Finished");
  }
  delay(1000);
}
```

3.5. Programska podrška za prijemni mikroupravljač

```
// Configuration
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

//Minimalna izmjerena duljina je 10ms i radi s LDR07

const int UNIT_LENGTH = 100;
const int BUFFER_SIZE = 5;


enum class Signal: byte {
  NOISE = 0,
  DIT = 1,
  DAH = 2,
  ELEMENTGAP = 3,
  GAP = 4,
  LONGGAP = 5
};

struct MorseCodeElement {
  Signal m_signal;
  unsigned long m_duration;
};

class MorseCodeBuffer {
  int m_size;
  int m_head;
  int m_tail;
  MorseCodeElement* m_buffer;

public:
  MorseCodeBuffer(int size) {
    //Koristi viška element za razlikovanje punog i praznog
    size++;

    m_size = size;
    m_head = 0;
    m_tail = 0;
    m_buffer = new MorseCodeElement[size];
  }

  bool Enqueue(MorseCodeElement element) {
    int new_tail = (m_tail + 1) % m_size;

    // Puno je?
    if (new_tail == m_head) {
      return false;
```

```
    }

    m_tail = new_tail;
    m_buffer[m_tail] = element;

    return true;
  }

  bool TryDequeue(MorseCodeElement* element) {
    // Prazno je?
    if (m_head == m_tail) {
      return false;
    }

    *element = m_buffer[m_head];
    m_head = (m_head + 1) % m_size;
    return true;
  }

  int GetCount() {
      if (m_head == m_tail) {
        return 0;
      }

      return (m_tail - m_head + m_size) % m_size;
  }
};

class AdaptiveLogicLevelProcessor {
  int m_sensorMinValue = 1023;
  int m_sensorMaxValue = 0;
  int m_sensorMedianValue = 511;
  unsigned long m_sensorCalibrationTime = 0;
  bool m_calibrated;

public:
  AdaptiveLogicLevelProcessor() {
    m_sensorMinValue = 1023;
    m_sensorMaxValue = 0;
    m_sensorMedianValue = 511;
    m_sensorCalibrationTime = 0;
  }

  bool process(int sensorValue, int* digitalInputValue) {
    unsigned long currentTime = millis();

    //Rekalibracija senzorskih vrijednosti
    if (currentTime - m_sensorCalibrationTime > 5000) {
      if (m_sensorMinValue < m_sensorMaxValue) {
```

```
      if (m_sensorMaxValue - m_sensorMinValue > 20) {
        m_sensorMedianValue = m_sensorMinValue + (m_sensorMaxValue -
m_sensorMinValue) / 2;
        m_calibrated = true;
       } else {
        Serial.println();
        Serial.print("Unreliable LOW/HIGH: ");
        Serial.print(m_sensorMinValue);
        Serial.print(' ');
        Serial.print(m_sensorMaxValue);
        Serial.println();
        m_calibrated = false;
       }
     }

      m_sensorMaxValue = 0;
      m_sensorMinValue = 1023;
      m_sensorCalibrationTime = currentTime;
   }

   if (m_sensorMinValue > sensorValue) {
     m_sensorMinValue = sensorValue;
   }

   if (m_sensorMaxValue < sensorValue) {
     m_sensorMaxValue = sensorValue;
   }

   if (!m_calibrated) {
     return false;
   }

   *digitalInputValue = sensorValue > m_sensorMedianValue ? HIGH : LOW;
   return true;
  }
};

class MorseCodeElementProcessor {
  unsigned long m_previousTime = 0;
  int m_previousSignal = LOW;

  int m_oneUnitMinValue;
  int m_oneUnitMaxValue;
  int m_threeUnitMinValue;
  int m_threeUnitMaxValue;
  int m_sevenUnitMinValue;
  int m_sevenUnitMaxValue;
```

```
public:
  MorseCodeElementProcessor(int unitLengthInMilliseconds) {
    m_oneUnitMinValue = (int)(unitLengthInMilliseconds * 0.5);
    m_oneUnitMaxValue = (int)(unitLengthInMilliseconds * 1.5);

    m_threeUnitMinValue = (int)(unitLengthInMilliseconds * 2.0);
    m_threeUnitMaxValue = (int)(unitLengthInMilliseconds * 4.0);

    m_sevenUnitMinValue = (int)(unitLengthInMilliseconds * 5.0);
    m_sevenUnitMaxValue = (int)(unitLengthInMilliseconds * 8.0);
  }

  bool process(int newSignal, MorseCodeElement* element) {
    unsigned long currentTime = millis();
    unsigned long elapsed;
    bool shouldBuffer = false;

    element->m_signal = Signal::NOISE;

    //Ako je prije bilo ISKLJUČENO, sad je UKLJUČENO
    if (m_previousSignal == LOW && newSignal == HIGH) {
      elapsed = currentTime - m_previousTime;
      element->m_duration = elapsed;

      if (m_sevenUnitMinValue <= elapsed) {
        element->m_signal = Signal::LONGGAP;
        shouldBuffer = true;
      } else if (m_threeUnitMinValue <= elapsed && elapsed <= m_threeUnitMaxValue)
{
        element->m_signal = Signal::GAP;
        shouldBuffer = true;
      } else if (m_oneUnitMinValue <= elapsed && elapsed <= m_oneUnitMaxValue) {
        element->m_signal = Signal::ELEMENTGAP;
        shouldBuffer = true;
      } else {
        element->m_signal = Signal::NOISE;
        shouldBuffer = true;
      }

      m_previousSignal = HIGH;
      m_previousTime = currentTime;
    } else if (m_previousSignal == HIGH && newSignal == LOW) {
      elapsed = currentTime - m_previousTime;
      element->m_duration = elapsed;

      if (m_threeUnitMinValue <= elapsed && elapsed <= m_threeUnitMaxValue) {
        element->m_signal = Signal::DAH;
        shouldBuffer = true;
      } else if (m_oneUnitMinValue <= elapsed && elapsed <= m_oneUnitMaxValue) {
```

```
      element->m_signal = Signal::DIT;
      shouldBuffer = true;
    } else {
      element->m_signal = Signal::NOISE;
      shouldBuffer = true;
    }

    m_previousSignal = LOW;
    m_previousTime = currentTime;
  }

  return shouldBuffer;
 }
};

class MorseCodeProcessor {
 private:
  static const int TREE_SIZE = 255;
  //Binarno stablo za pretraživanje sadržaja pojedinog elementa
  static constexpr char tree[TREE_SIZE] = {
    '\0', '\0', '\0', '5', '\0', '\0', '\0', 'H', '\0', '\0', '\0', '4', '\0', '\0', '\0', 'S',
    '\0', '\0', '$', '\0', '\0', '\0', '\0', 'V', '\0', '\0', '\0', '3', '\0', '\0', '\0', 'I',
    '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'F', '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'U',
    '\0', '?', '\0', '\0', '\0', '_', '\0', '\0', '\0', '\0', '\0', '2', '\0', '\0', '\0', 'E',
    '\0', '\0', '\0', '&', '\0', '\0', '\0', 'L', '\0', '"', '\0', '\0', '\0', '\0', '\0', 'R',
    '\0', '\0', '\0', '+', '\0', '.', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'A',
    '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'P', '\0', '@', '\0', '\0', '\0', '\0', '\0', 'W',
    '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'J', '\0', '\'', '\0', '1', '\0', '\0', '\0', '\0',
    '\0', '\0', '\0', '6', '\0', '-', '\0', 'B', '\0', '\0', '\0', '=', '\0', '\0', '\0', 'D',
    '\0', '\0', '\0', '/', '\0', '\0', '\0', 'X', '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'N',
    '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'C', '\0', ';', '\0', '\0', '\0', '!', '\0', 'K',
    '\0', '\0', '\0', '(', '\0', ')', '\0', 'Y', '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'T',
    '\0', '\0', '\0', '7', '\0', '\0', '\0', 'Z', '\0', '\0', '\0', '\0', '\0', ',', '\0', 'G',
    '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'Q', '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'M',
    '\0', ':', '\0', '8', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', 'O',
    '\0', '\0', '\0', '9', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '0', '\0', '\0', '\0'
  };

  bool m_error;
  int m_start;
  int m_end;
  int m_index;
  Signal m_previousInput;

  void reset() {
    m_error = false;
    m_start = 0;
    m_end = TREE_SIZE;
    m_index = (m_end - m_start) / 2;
```

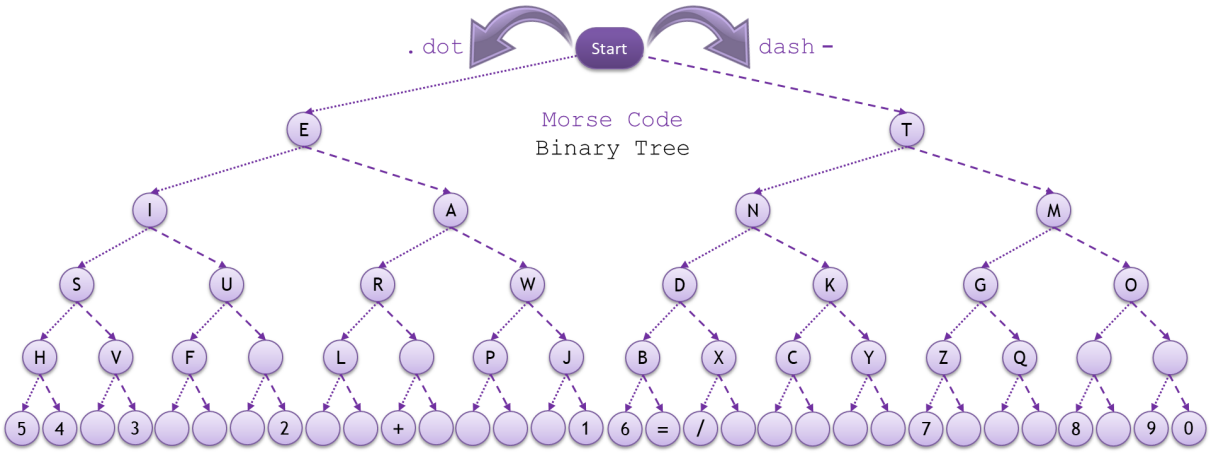
```
  }

 public:
  MorseCodeProcessor() {
   reset();
   m_previousInput = Signal::NOISE;
  }

  bool process(Signal input, char* output) {
   bool completed = false;

   if (!m_error && input == Signal::DIT) {
    if (m_start == m_index) {
     m_error = true;
    } else {
     m_end = m_index;
     m_index = m_start + (m_end - m_start) / 2;
    }
   } else if (!m_error && input == Signal::DAH) {
    if (m_end == m_index) {
     m_error = true;
    } else {
     m_start = m_index + 1;
     m_index = m_start + (m_end - m_start) / 2;
    }
   } else if (input == Signal::GAP || input == Signal::LONGGAP) {
    completed = !m_error && tree[m_index] != 0;

    if (completed) {
     output[0] = tree[m_index];
     output[1] = '\0';

     if (input == Signal::LONGGAP) {
      output[1] = ' ';
      output[2] = '\0';
     }
    }

    reset();
   }

   m_previousInput = input;

   return completed;
  }
};

constexpr char MorseCodeProcessor::tree[];
```

```
MorseCodeBuffer buffer(BUFFER_SIZE);
MorseCodeProcessor morseCodeProcessor;
AdaptiveLogicLevelProcessor logicLevelProcessor;
MorseCodeElementProcessor morseCodeElementProcessor(UNIT_LENGTH);


/*************************************************************
 * Timer interrupt funkcija za obradu analognog signala
 *************************************************************/
SIGNAL(TIMER0_COMPA_vect) {
  cli();

  int digitalInputValue;

  if (logicLevelProcessor.process(analogRead(A0), &digitalInputValue)) {
    MorseCodeElement element;

    if (morseCodeElementProcessor.process(digitalInputValue, &element)) {
      buffer.Enqueue(element);
    }
  }

  sei();
}

//Inicijalizacija LCD display-a.
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

void setup() {
  Serial.begin(9600);
  lcd.begin(16,2);              // potrebno da bismo mogli koristiti LCD.
                          // 16,2 oznacava dimenziju LCDa
  lcd.backlight();            // upali pozadinsko osvjetljenje
  delay(1000);
  lcd.noBacklight();            // ugasi pozadinsko osvjetljenje
  delay(1000);
  lcd.backlight();            // opet upali pozadinsko osvjetljenje

  // Postavi timer koji će se pozvati svake milisekunde
  cli();
  OCR0A = 0xAF;
  TIMSK0 |= _BV(OCIE0A);
  sei();
}

/*************************************************************
 * Pomoćna funkcija za sigurno uklanjanje stavke iz međuspremnika
 *************************************************************/
bool TryDequeueSafe(MorseCodeElement* element) {
```

```
  // Isključivanje stavke iz međuspremnika dok se onemogućuje
  //prekid kako ne bi oštetili status međuspremnika
  cli();
  bool result = buffer.TryDequeue(element);
  sei();

  return result;
}

char* output = new char[3];

void loop() {
  char result[50] = "";
  MorseCodeElement element;

  // Isprazni buffer
  while (TryDequeueSafe(&element)) {
    if (element.m_signal == Signal::DIT) {
      Serial.print(".");
    } else if (element.m_signal == Signal::DAH) {
      Serial.print("-");
    }

    if (morseCodeProcessor.process(element.m_signal, output)) {

      Serial.print('(');

      //lcd.setCursor(0,0);
      lcd.print(output);
      Serial.print(output);
      Serial.print(')');
    }

    if (element.m_signal == Signal::LONGGAP) {
      Serial.println();
    }
  }
}
```

Glavni problem bio je odrediti optimalno vrijeme za trajanje točke i trajanje crtice. Mjerenjima smo zaključili da je minimalna vrijendost koja se može očitati s fotootpornika 10ms te smo se odlučili za pristup s prekidima koristeći Timer Interupt. Svakih 10ms se provjerava da li LED svijetli te koliko dugo već svijetli. Nakon toga se utvrđuje da li je predani znak zapravo slovo ili razmak ili nešto treće. Nakon potvrde o kategoriji dolazi do

pretraživanja binarnog stabla (Slika 3.5) kako bi pronašli koji je znak u pitanju te prijevodu. Nakon prijevoda ispisujemo slovo po slovo na LCD Display.



Slika 3.5. Binarno stablo Morseove abecede (- specijalni znakovi)

4. Programska realizacija sustava

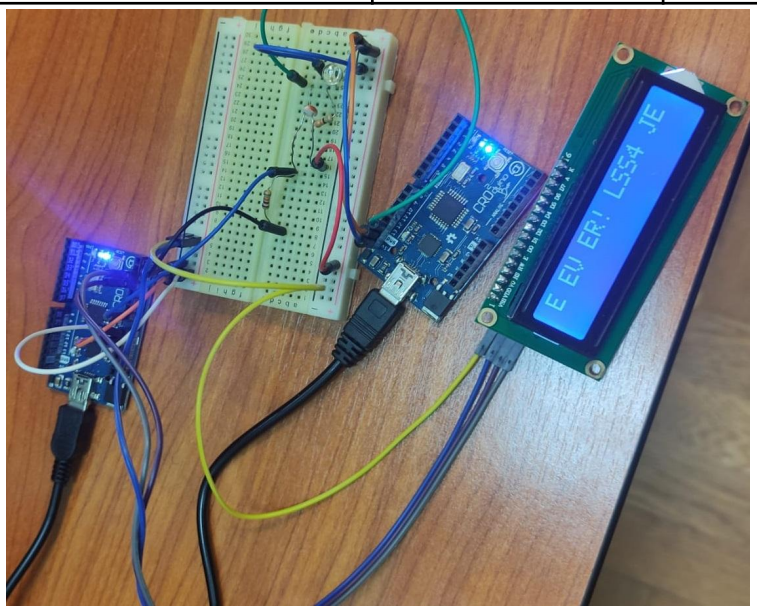
Arduino je kreirao svoj programski paket koji korisnicima omugućuje unos programa, njihovo jednostavno kompajliranje te prebacivanje na mikrokontrolersku pločicu. U razvojnom okruženju korisnik dobiva i povratnu informaciju od kompajlera (u slučaju da 15 program sadrži greške), a osim toga može prikazivati i slati podatke preko serijske veze (npr. USB- a) što može olakšati testiranje uređaja jednom kada je isprogramiran. Programski paket se može besplatno preuzeti sa Arduinovih web stranica. Odabirom na TOOLS potrebno je podesiti BOARD koji se koristi. U ovom slučaju potrebno je odabrati Arduino Nano te odabrati SERIAL PORT na koji je priključen.

Za povezivanje Arduino mikrokontrolerske pločice i računala potreban je USB kabel. Osim u postupku programiranja, također se može koristiti i za uspostavljanje serijske veze sa drugim uređajima te prijenos podataka, a ujedno i kao napajanje. Nakon odabira potrebnog BORD- a i porta na koji je mikrokontroler priključen te napisanog programa potrebno je odabirom na UPLOAD učitati program u mikrokontroler.

5. Proizvodna dokumentacija

Tablica. 4.1. Potrebne komponente za realizaciju projekta.

R.Br	Naziv	Tip i vrijednost	Količina
1.	Croduino Basic2	mikroupravljač	2
2.	LCD 16x 2 + I2C adapter	display	1
3.	LDR07	fotootpornik	1
4.	LE dioda, crvena 5mm	LED	1
5.	Otpornik 10kΩ	otpornik	1
6.	Otpornik 330Ω	otpornik	1



Slika 4.1 Realiziran sustav tumačenja Morseove abecede tijekom testiranja nasumične poruke

6. Zaključak

Iako je Morseova abeceda danas rjeđe korištena, ovaj projekt je bio odličan za upoznavanje s radom nekih od osnovnih funkcija Croduino Basic2 mikroupravljača te ostalih komponenti korištenih tijekom izrade projekta. Svaki od sustava ima svojih prednosti, a isto tako u svakom ostaje još mjesta za nadogradnju.

Projekt je moguće još proširiti npr. koristeći se nekim korisničkim sučeljem (aplikacijom) kojim bi prenijeli poruku koja će biti poslana koristeći Morseovu abecedu. Nedostatak projekta jest to što fotootpornik ima relativno veliku osjetljivost na svjetlinu ukoliko nije “mrak”, primljeni signal možda ne bude točan. To je moguće ispraviti npr. izgradnjom kućišta.