ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

РАСЧЁТНО-ГРАФИЧЕСКАЯ РАБОТА

по дисциплине “Микропроцессорные системы”

Выполнил студент Ермаков Арсений Владимирович
Ф.И.О.

Группы ИВ-121

Работу принял _____ Гонцова А. В.
подпись

Защищена _____ Оценка _____

Новосибирск – 2024

Содержание

Задание

Необходимо реализовать сигнальный таймер/счетчик с выводом на семисегментный индикатор. Данный таймер должен отсчитать введенное с клавиатуры время и по окончании отсчета издать звуковой сигнал. На каждый отсчет на семисегментном индикаторе должно изменяться показание (уменьшаться или увеличиваться).

Описание компонентов

Для начала необходимо выбрать нужные составляющие устройства, среди которых должны быть:
1) Микроконтроллерная плата, которая выполняет функцию центрального управляющего устройства. В нашем случае – **Arduino Uno R3**, которая имеет следующие характеристики:

Характеристика	**Значение**
Микроконтроллер	ATmega328p
Ядро	AVR
Тактовая частота	16 МГц
Flash-память	32 КБ
RAM-память	2 КБ
EEPROM-память	1 КБ
Пины ввода-вывода	20
Пины с прерыванием	2
Пины с АЦП	6
Разрядность АЦП	10 бит
Пины с ШИМ	6
Разрядность ШИМ	8 бит
Аппаратные интерфейсы	1× UART, 1× I²C, 1× SPI
Напряжение логических уровней	5 В
Входное напряжение питания:	
через USB	5 В
через DC-разъём или пин Vin	7,5–12 В
Максимальный выходной ток пина 3V3	150 мА
Максимальный выходной ток пина 5V	1 А
Размеры	69×53 мм

Табл. 1.1. Технические характеристики Arduino Uno R3

Технические характеристики и составляющие **ATmega328p**:

1) Количество разрядов: 8
2) 32 КБ флэш- памяти ISP с возможностями чтения во время записи
3) 1 КБ EEPROM
4) 2 КБ SRAM
5) 23 линии ввода-вывода общего назначения
6) 32 рабочих регистра общего назначения
7) Три гибких таймера / счетчики с режимами сравнения, внутренними и внешними прерываниями
8) Последовательный программируемый USART

9) Байтовый 2-проводный последовательный интерфейс
10) Последовательный порт SPI
11) 6-канальный 10-битный аналого - цифровый преобразователь (8 каналов в пакетах TQFP и QFN / MLF)
12) Программируемый сторожевой таймер с внутренним генератором пять программно выбираемых режимов энергосбережения.
13) Устройство работает в диапазоне 1,8-5,5 вольт.
14) Пропускная способность устройства приближается к 1 MIPS на МГц.

Сам микроконтроллер состоит из следующих вычислительных блоков:

Рис. 1.1 Вычислительные блоки микроконтроллера

2) Клавиатура 4x4, с которой будет задаваться время отсчета.
3) Семисегментный индикатор, он же 7-сегментный экран, отрисовывает цифры.

4) Пьезоэлемент, по-простому – пищалка. Он будет нужен для воспроизведения сигнала в конце отсчета.
5) Малая макетная плата, на которой будут располагаться 7-сегментный индикатор и пьезоэлемент.
6) Два резистора, которые будут подключены к общим выходам 7-сегментного индикатора.
7) Соединительные провода.

Схема подключения

Первым делом на плате закрепляем 7-сегментный индикатор с необходимыми резисторами:

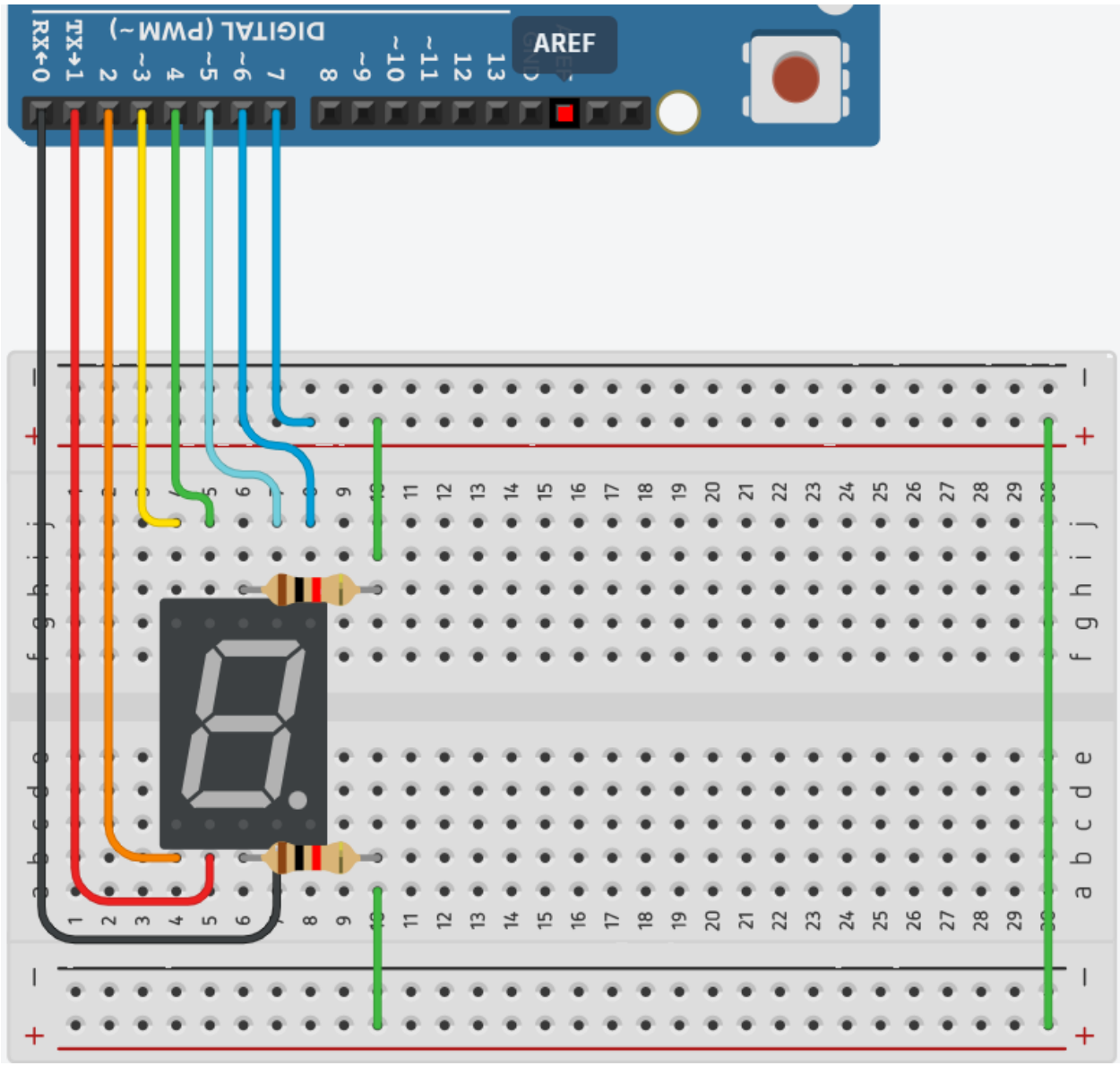

Рис. 2.1 Подключение 7-сегментного индикатора.

Для подключения будем использовать цифровые порты с 0 по 6 (типа D) для ножек C, D, E, G, F, A, B соответственно. Два выхода «общие» подключим к седьмому цифровому порту (типа D) через резисторы.

Далее в нашу схему добавляется пьезоэлемент, который подключаем с одной стороны к GND, с другой – к 8 цифровому порту, или же 0 порту типа B:

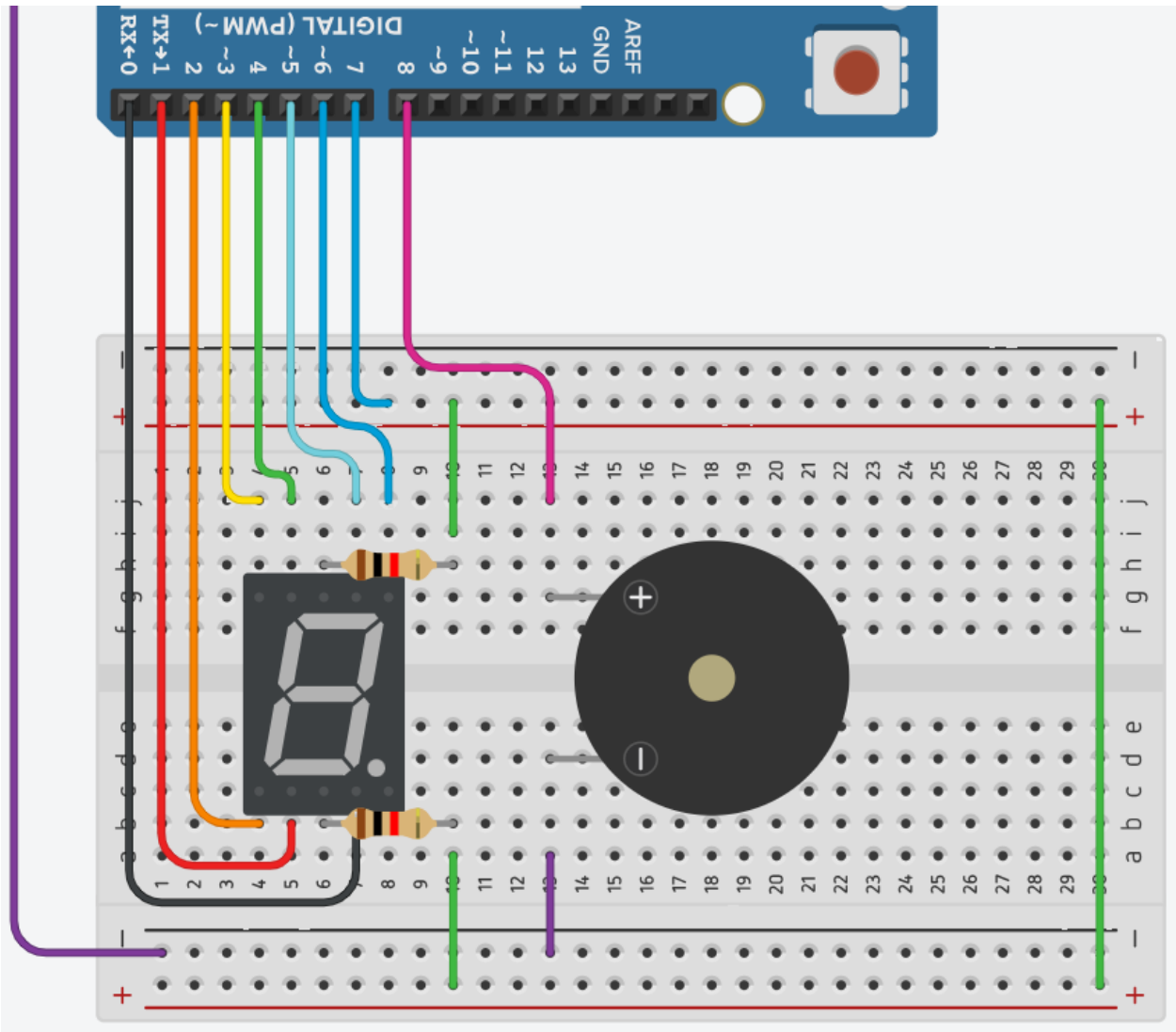

Рис. 2.2 Подключение пьезоэлемента

Затем подключаем клавиатуру 4x4: строку 1 подключаем к 13 цифровому порту (типа B), строки 2 – 4 присоединяем соответственно к 10 – 12 цифровым портам (типа B). Столбцы с 1 по 4 подключаем к аналоговым портам A2 – A5 соответственно:

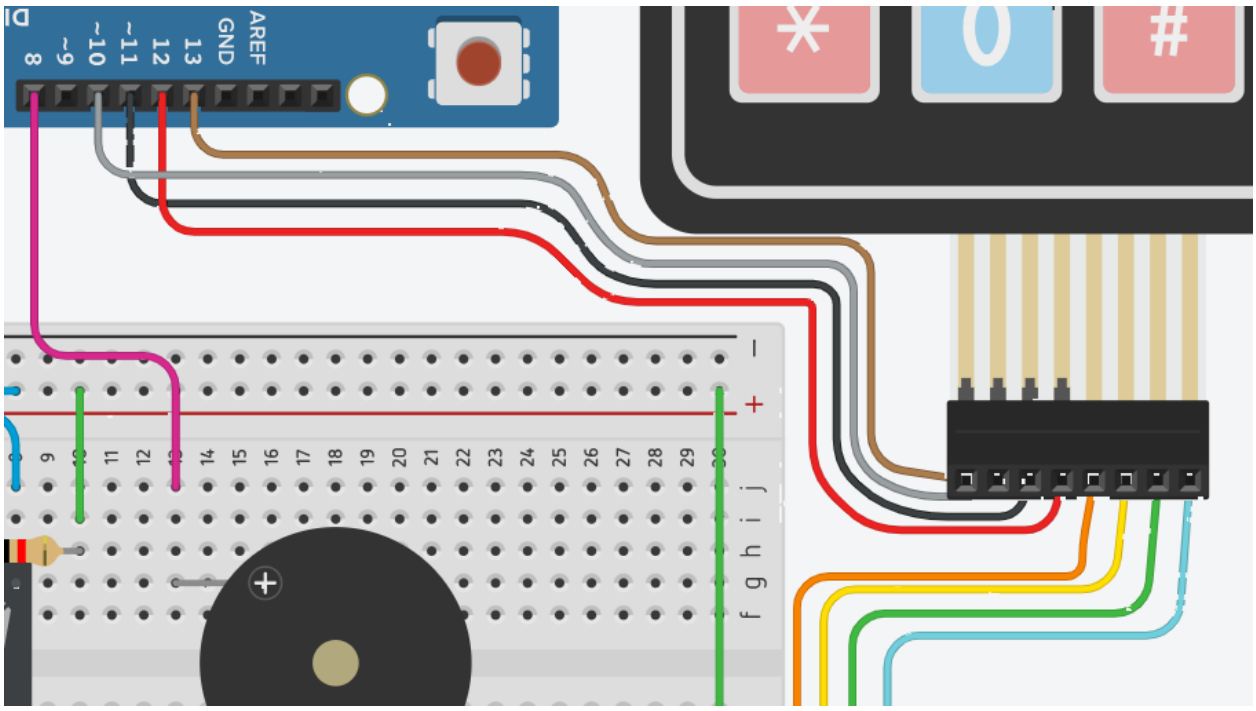

Рис. 2.3 Подключение строк клавиатуры 4x4

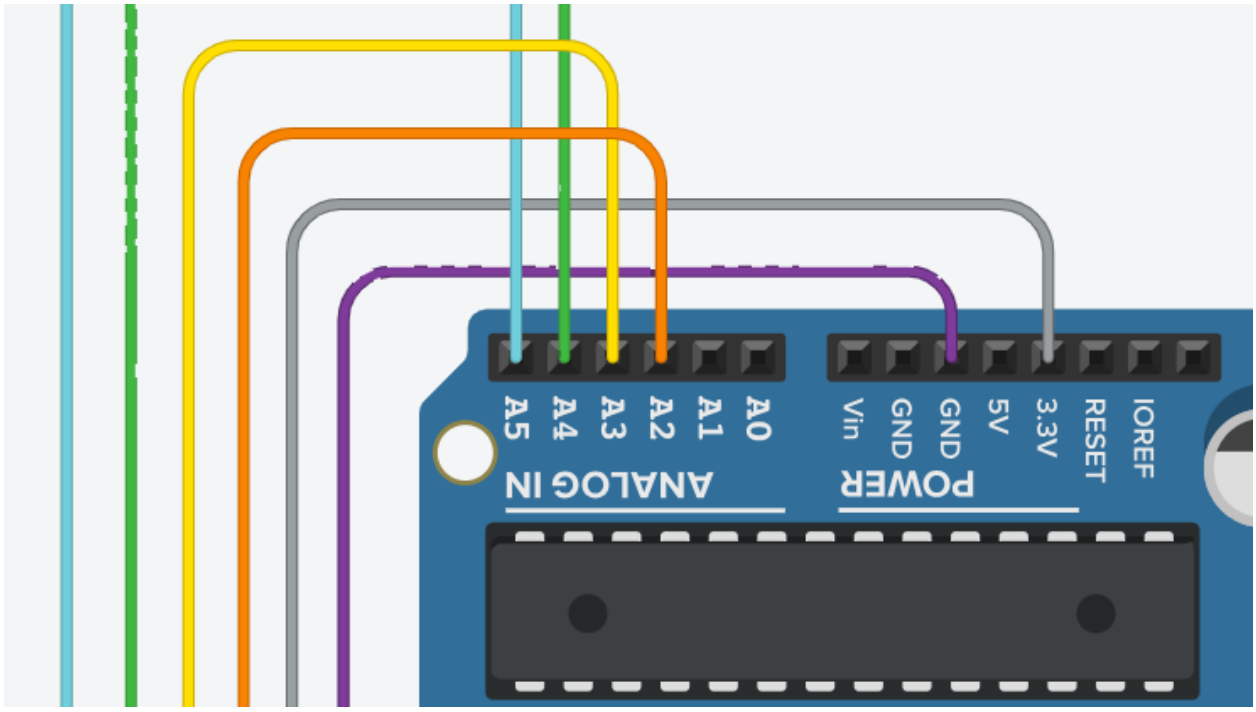

Рис. 2.4 Подключение столбцов клавиатуры 4x4

Разработка программы

Для начала в функции **main** устанавливаем направление данных (ввод/вывод) для портов и их начальные состояния:

```
1    DDRD |= 0b11111111;
2    DDRB |= 0b00000001;
3    DDRB |= 0b00111100;
4    PORTC |= 0b00111100;
```

Рис. 3.1 Установка состояний и направлений данных портов

Затем в бесконечном цикле запускаем два цикла for: внешний для обхода столбцов, внутренний – строк. Далее биты порта B, отвечающие за строки, сбрасываются, а затем устанавливаются уже нужные биты из массива **portMask**, чтобы активировать определённую строку:

```
1  for (int i = 0; i < 4; i++) {
2      PORTB &= 0x00;
3      PORTB |= portMask[i];
4      for (int j = 0; j < 4; j++) {
```

Рис. 3.2 Циклы по строкам и столбцам с установлением строк

```
1  const uint8_t portMask[] = {0b11011111, 0b11101111, 0b11110111,
2  0b11111011};
   const uint8_t pinMask[] = {0b00000100, 0b00001000, 0b00010000,
   0b00100000};
```

Рис. 3.3 Массивы portMack и pinMask

Далее в условии проверяем была ли нажата кнопка в текущих столбце и строке, если она была нажата, то вызывается функция **reduce** (запускает таймер на 7-сегментном индикаторе), индикатор «очищается» с помощью функции **clear_indikator** и проигрывается сигнал функцией **play_note**:

```
1  if (!(PINC & pinMask[j])) {
2              reduce(numbers[j][i]);
3              clear_indikator();
4  }
```

Рис. 3.4 Условие проверяющее нажатие кнопки

Массив **numbers**, в зависимости от нажатой клавиши клавиатуры 4x4, подает на вход функции **reduce** нужное число. Отрицательные числа в массиве – это знаки * и #, а также буквы A, B, C, D, которые не выводятся:

```
1  const int numbers[4][4] = {
```

```
2       {1, -1, 7, 4},
3       {2, 0, 8, 5},
4       {3, -2, 9, 6},
5       {-3, -4, -5, -6}
6   };
```

Рис. 3.5 Массив numbers

Функция **reduce** принимает на вход число numb – то количество секунд (за единицу измерения отвечает глобальная переменная **time_unit**), которое, если numb не отрицательное, должен отсчитать таймер и проиграть с помощью функции **play_note()**, что достигается с помощью цикла, в котором отрисовывается число на 7-сегментном индикаторе, устанавливается задержка в 1 секунду и проигрывается сигнал:

```
1   void reduce(int numb)
2   {
3     if (numb >= 0){
4       for (int z = numb; z >= 0; z--) {
5         draw(z);
6         _delay_ms(time_unit);
7       }
8       play_note();
9     }
10  }
```

Рис. 3.6 Функция reduce

Функция **draw** отрисовывает переданное ей на вход число. Для этого она «очищает» семисегментный регистр от предыдущего числа при помощи функции **clear_indikator** и в цикле задает значение (вкл/выкл) каждого сегмента 7-сегментного регистра при помощи массива **num** (рис. 2.6):

```
1   void draw(int numb) {
2       clear_indikator();
3       for (int x = 0; x < 7; x++) {
4           PORTD |= (num[numb][x] << x);
5       }
6   }
```

Рис. 3.7 Функция draw

```
1   const int num[11][7] = {
2       {1,1,1,0,1,1,1}, // 0
3       {1,0,0,0,0,0,1}, // 1
4       {0,1,1,1,0,1,1}, // 2
5       {1,1,0,1,0,1,1}, // 3
6       {1,0,0,1,1,0,1}, // 4
7       {1,1,0,1,1,1,0}, // 5
8       {1,1,1,1,1,1,0}, // 6
9       {1,0,0,0,0,1,1}, // 7
10      {1,1,1,1,1,1,1}, // 8
```

```
11      {1,1,0,1,1,1,1}  // 9
12  };
```
Рис. 3.8 Массив num

Функция **clear_indikator** «очищает» 7-сегментный регистр, устанавливая все его выводы в нулевое состояние:

```
1  void clear_indikator()
2  {
3          PORTD &= 0;
4  }
```
Рис. 3.9 Функция clear_indikator

Функция **play_note** воспроизводит звуковой сигнал. Она включает и выключает пин PB0, к которому подключен пьезоизлучатель, чередующимся образом в течение **count_play_note** итераций цикла. В результате получается звуковой сигнал с частотой, определяемой **frequency**:

```
1  const int frequency = 523;
2  const int count_play_note = 100;
```
Рис. 3.10 Необходимые переменные для функции play_note

```
1   void play_note() {
2
3     uint16_t period = (F_CPU / 8) / frequency;
4     uint16_t half_period = period / 4;
5     for (int i = 0; i < count_play_note; i++) {
6       PORTB |= (1 << PB0);
7       _delay_us(half_period);
8       PORTB &= ~(1 << PB0);
9       _delay_us(half_period);
10    }
11  }
```
Рис. 3.11 Функция play_note

Тестирование

В ходе тестирования необходимо проверить корректность работы разработанного устройства на сайте **TinkerCad** и убедиться в следующем:
1) Семисегментный индикатор корректно выводит числа (используются все семь сегментов, которые имеют одинаковую яркость).
2) Нажатие на любую клавишу клавиатуры, отвечающую за цифру, вызывает обратный отсчет от соответствующей цифры.
3) Пьезоэлемент воспроизводит сигнал в конце отсчета (после появления нуля на семисегментном индикаторе).
4) Все вышеперечисленные составляющие сигнального таймера должны работать без помех, бесконтрольных или незапланированных действий, строго и четко по заданию, из раза в раз абсолютно одинаково.

В ходе тестирования разработанного устройства какие-либо неполадки обнаружены не были, устройство работало в следующем режиме:
1) Режим ожидания (ожидается нажатие клавиши, индикатор не горит, пьезоэлемент не воспроизводит звуки):

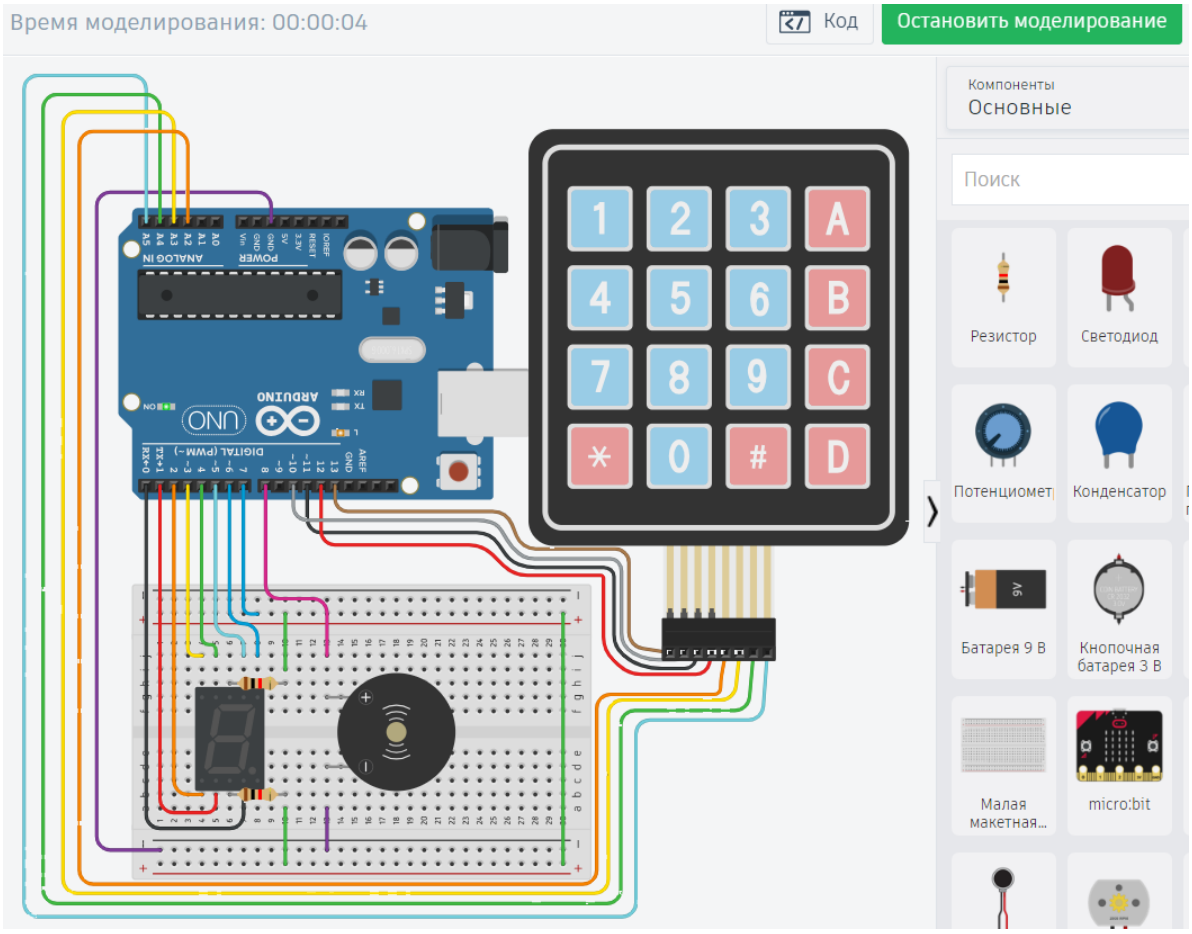

Рис. 4.1 Сигнальный таймер в режиме ожидания

2) Режим отсчета (после нажатия клавиши с цифрой, запускается таймер от выбранного числа до 0, пьезоэлемент не издает сигнал):

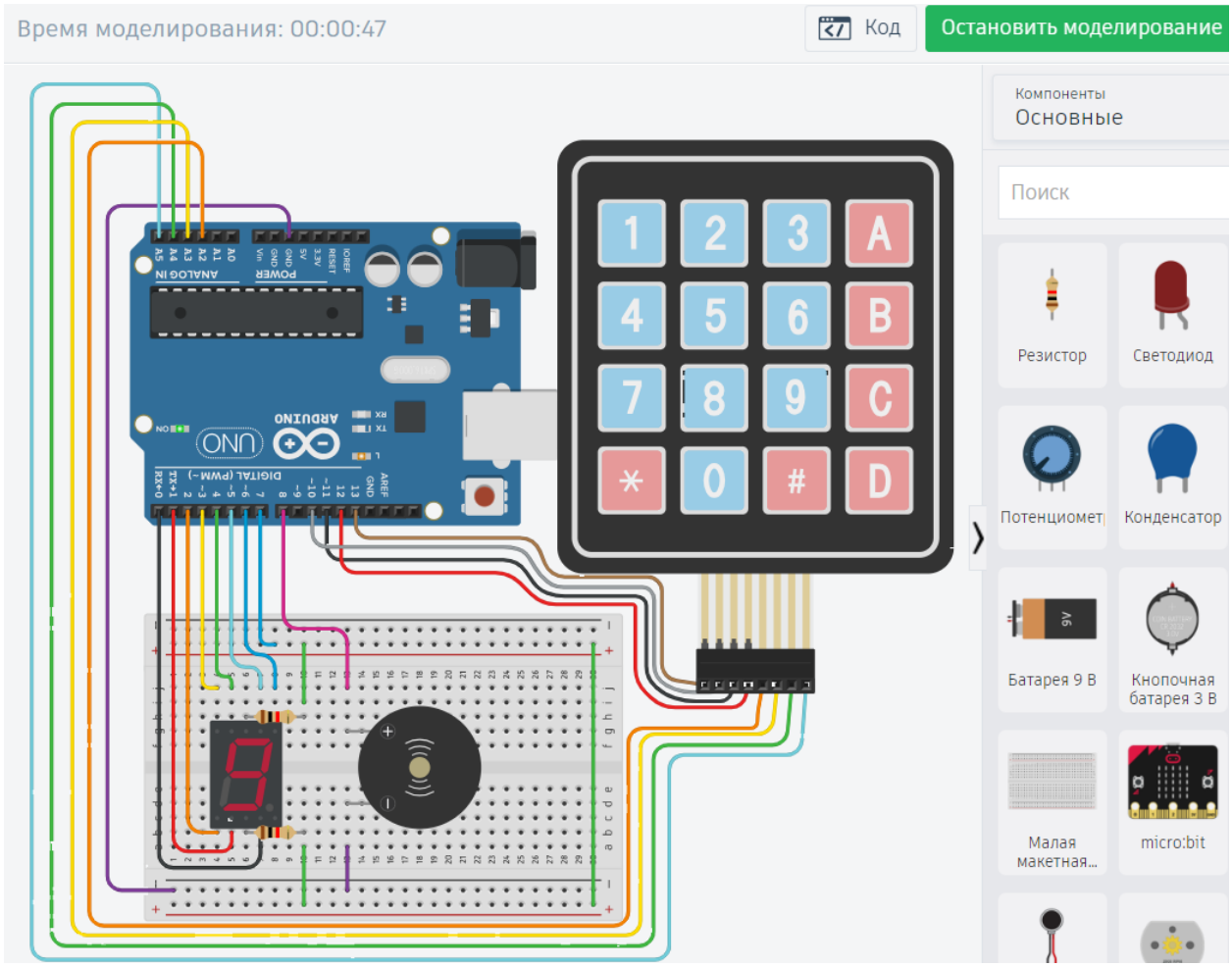

Рис. 4.2 Сигнальный таймер в режиме отсчета

Рис. 4.3а – з Семисегментный индикатор в работе

3) Режим воспроизведения сигнала (при всех тестах должен звучать одинаково, воспроизводится после того, как на 7-сегментном индикаторе появляется число 0 и проходит одна секунда времени). К сожалению, я не могу привести доказательства звучания сигнала с пьезоэлемента, но заверяю, что он работает по заданию.

Помимо вышеописанного тестирования в конструкторе **TinkerCad**, данное устройство было собрано «вживую» по схеме, отработанной на сайте **TinkerCad,** с тем же кодом и протестировано. Корректность работы сигнального таймера была подтверждена.

Заключение

В ходе выполнения работы мы сконструировали сигнальный таймер из микроконтроллерной платы Arduino Uno R3, клавиатуры 4x4, пьезоэлемента, семисегментного индикатора, семи резисторов, малой макетной платы и проводов.
Написали программу, которая считывает введенное с клавиатуры число, производит отсчет от него до нуля, после чего воспроизводит звуковой сигнал.
Полученное устройство было протестировано, необходимая функциональность была достигнута (как оговаривалось в задании).

Листинг кода

```
#include <avr/io.h>
#include <util/delay.h>

const int frequency = 523;
const int count_play_note = 100;
const int time_unit = 1000;

const int num[11][7] = {
    {1,1,1,0,1,1,1}, // 0
    {1,0,0,0,0,0,1}, // 1
    {0,1,1,1,0,1,1}, // 2
    {1,1,0,1,0,1,1}, // 3
    {1,0,0,1,1,0,1}, // 4
    {1,1,0,1,1,1,0}, // 5
    {1,1,1,1,1,1,0}, // 6
    {1,0,0,0,0,1,1}, // 7
    {1,1,1,1,1,1,1}, // 8
    {1,1,0,1,1,1,1}  // 9
};

void clear_indikator()
{
         PORTD &= 0;
}

void draw(int numb) {
    clear_indikator();
    for (int x = 0; x < 7; x++) {
        PORTD |= (num[numb][x] << x);
    }
}

void reduce(int numb)
{
  if (numb >= 0){
    for (int z = numb; z >= 0; z--) {
      draw(z);
      _delay_ms(time_unit);
    }
    play_note();
  }
}

void play_note() {

  uint16_t period = (F_CPU / 8) / frequency;
  uint16_t half_period = period / 4;
  for (int i = 0; i < count_play_note; i++) {
    PORTB |= (1 << PB0);
    _delay_us(half_period);
    PORTB &= ~(1 << PB0);
    _delay_us(half_period);
  }
}

```

```
56 const uint8_t portMask[] = {0b11011111, 0b11101111, 0b11110111,
57 0b11111011};
58 const uint8_t pinMask[] = {0b00000100, 0b00001000, 0b00010000,
59 0b00100000};
60
61 const int numbers[4][4] = {
62     {1, -1, 7, 4},
63     {2, 0, 8, 5},
64     {3, -2, 9, 6},
65     {-3, -4, -5, -6}
66 };
67
68 int main() {
69   DDRD &= 0;
70   DDRB &= 0;
71   PORTC &= 0;
72
73   DDRD |= 0b11111111;
74   DDRB |= 0b00000001;
75   DDRB |= 0b00111100;
76   PORTC |= 0b00111100;
77
78   _delay_ms(10);
79   while(1) {
80     _delay_ms(10);
81     for (int i = 0; i < 4; i++) {
82     PORTB &= 0x00;
83     PORTB |= portMask[i];
84     for (int j = 0; j < 4; j++) {
85         if (!(PINC & pinMask[j])) {
86             reduce(numbers[j][i]);
87             clear_indikator();
88         }
89     }
90   }
91   }
92   return 0;
93 }
```