**Отчет по проекту   
«Piano bot»**

Выполнил студент гр. 3331506/80401

Редров В.С.

Проверил Хазанский Р.Р.

Санкт-Петербург

2021г.

# Содержание

[Содержание 2](#_Toc72427238)

[Механическая часть 3](#_Toc72427239)

[Схема подключения 4](#_Toc72427240)

[Протокол коммуникации 5](#_Toc72427241)

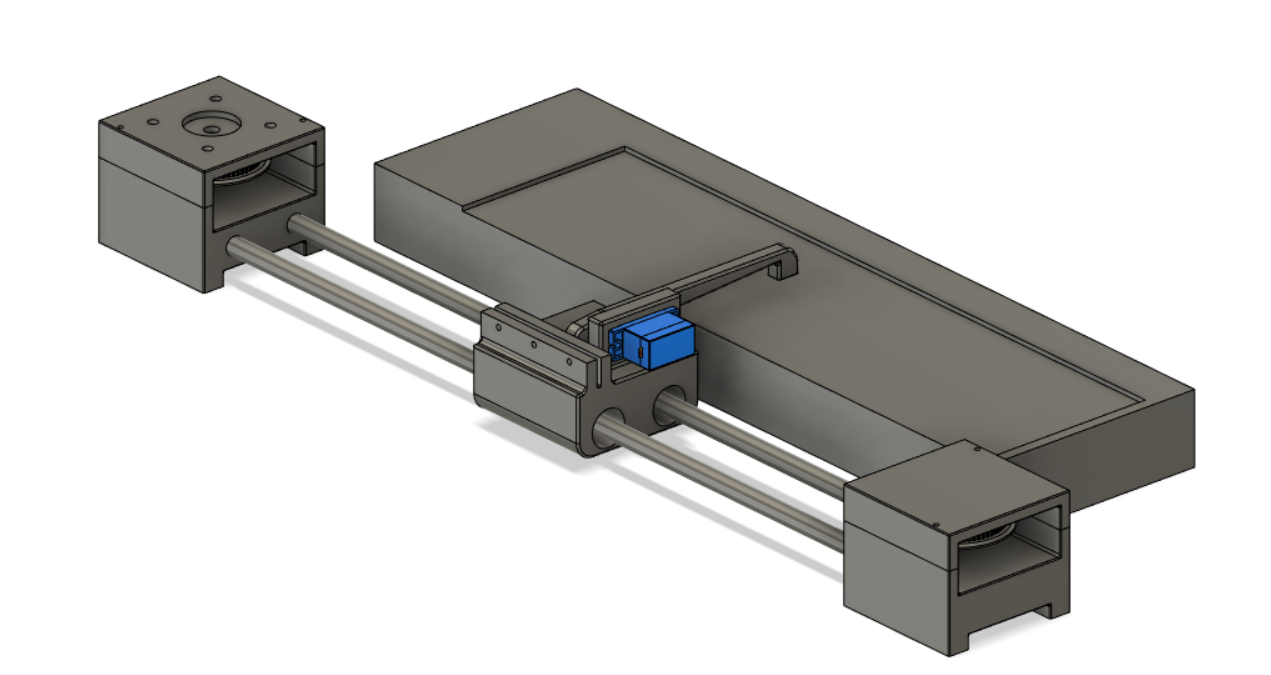
[Обзор кода 6](#_Toc72427242)

[Вместо заключения 12](#_Toc72427243)

# Механическая часть

Механическая часть проекта представляет собой базу, перемещающуюся по двум направляющим посредством шагового двигателя через ременную передачу, и палец, который приводится в движение сервоприводом, а также два опорных элемента.

4



3

1

2

Рисунок 1 – Общий вид механической части  
(1 - midi-клавиатура, 2 – сервопривод,   
3 – место установки шагового двигателя)

# Схема подключения

Схема подключения приведена на рисунке 2.

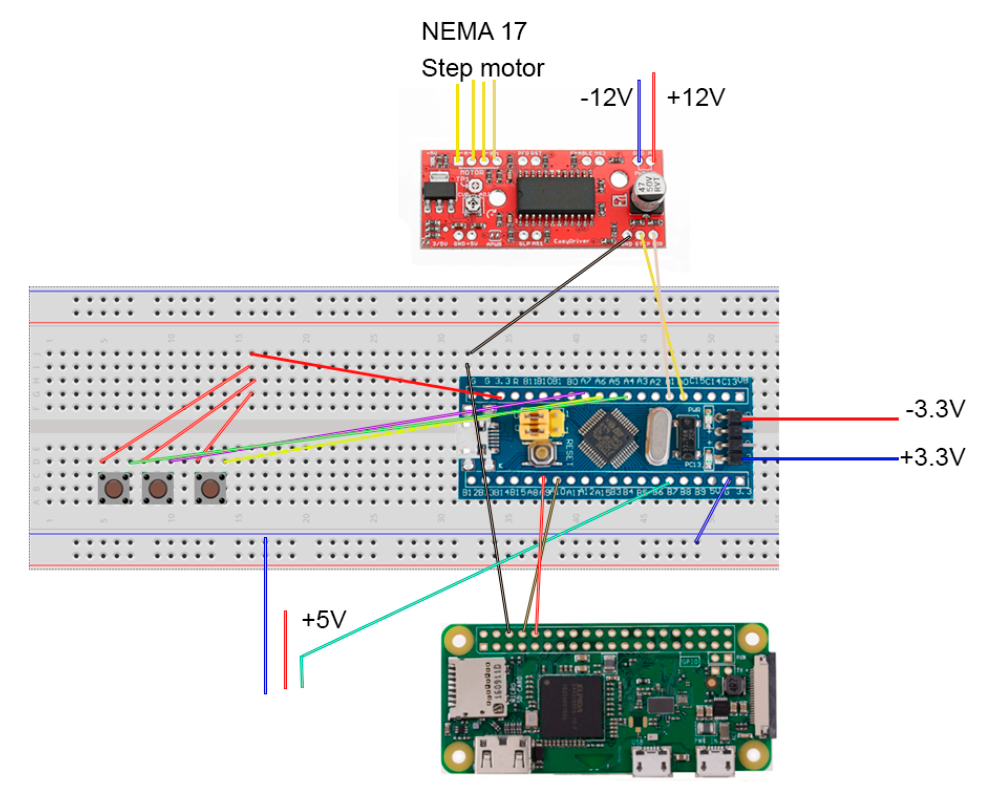


Рисунок 2 – Схема подключения

На схеме присутствует три источника питания: 12В, 5В, 3.3В, для шагового двигателя, сервопривода и STM соответственно. Выводы Raspberry GPIO14 (TX), и GPIO15 (RX) подключены соответственно к выводам А10 и А9 STM. Вывод B7 STM соединен с управляющим выводом сервопривода, земли сервопривода и STM также соединены. К выводам A0 и A1 подсоединены выводы step и dir драйвера, земли соединены. Кнопки подсоединены к выводам A6, A7, A4. Эти ноги находятся в режиме Pull Up.

# Протокол коммуникации

Для коммуникации между STM32 и Raspberry используется протокол UART. Схема передачи одного байта представлена на рисунке 3.

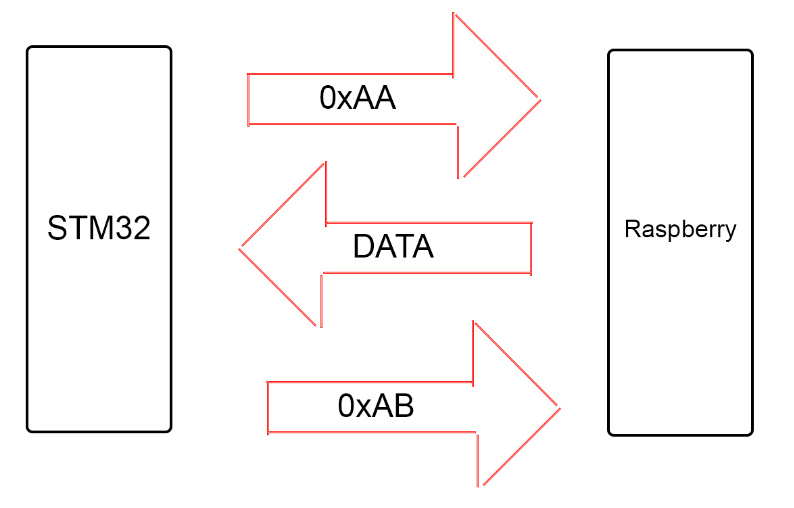


Рисунок 3 – Схема передачи данных

Как видно на рисунке для начала общения STM посылает «байт готовности к получению», который является числом 0xAA. После чего Raspberry отправляет байт данных. При успешном получении данных STM посылает подтверждающий байт 0xAB.

В случае если «байт готовности к получению» не пришел, или пришел битый. Raspberry обрывает связь. Если Raspberry получает неверный подтверждающий бит, будет инициирована повторная попытка передачи данных. В случае провала повторной попытки связь будет прекращена.

# Обзор кода

По большей части в коде присутствуют поясняющие комменты, поэтому здесь я разберу только какие-то основные либо напротив – узкие моменты.

Начнем обзор с кода функции main.

int main(void)

{

    // initialization

    step\_motor\_init();

    servo\_init();

    usart\_init();

    create\_keys\_array();

#ifdef Calibrate

    // blink to show that program is ready for calibration

    GPIO\_WriteBit(GPIOC, GPIO\_Pin\_13, Bit\_RESET);

    delay(20000000);

    GPIO\_WriteBit(GPIOC, GPIO\_Pin\_13, Bit\_SET);

    calibrate();

#endif

    uint16\_t length = 0;

    get\_byte(&length);

    uint16\_t notes[length];

    uint16\_t delays[length];

    receive\_music(notes, delays, length);

    // wait for finger button pushed to start playing music

    while (!GPIO\_ReadInputDataBit(GPIOA, GPIO\_Pin\_4)){

        GPIO\_WriteBit(GPIOC, GPIO\_Pin\_13, Bit\_RESET);

        delay(1000000);

        GPIO\_WriteBit(GPIOC, GPIO\_Pin\_13, Bit\_SET);

        delay(1000000);

    }

    play\_music(notes, delays, length);

    // light LED up to show that track is over

    GPIO\_WriteBit(GPIOC, GPIO\_Pin\_13, Bit\_RESET);

    while (1){}

    return 0;

}

В начале проходит этап инициализации. Отдельно конфигурируется периферия для шагового двигателя, сервопривода и uart. Создается массив соответствия количества шагов шагового мотора и номеру клавиши midi клавиатуры.

После – вызывается функция калибровки шагового мотора. Рассмотрим ее.

void calibrate(void){

    while(1){

        if (GPIO\_ReadInputDataBit(GPIOA, GPIO\_Pin\_6)){

            move\_motor(MOTOR\_RIGHT, 8000);

        }

        else if (GPIO\_ReadInputDataBit(GPIOA, GPIO\_Pin\_7)){

            move\_motor(MOTOR\_LEFT, 8000);

        }

        if (GPIO\_ReadInputDataBit(GPIOA, GPIO\_Pin\_4)){

            move\_finger(FINGER\_DOWN);

            steps = C2\_STEPS;

            delay(10000000);

            move\_finger(FINGER\_UP);

        #ifdef Try\_Key

            try\_keys();

        #endif

            return;

        }

    }

}

Все, что тут происходит – скан двух кнопок, отвечающих за двигатель и отдача команды на его поворот. И скан кнопки опускания пальца, она же кнопка окончания калибровки. В момент ее нажатия в текущий глобальный параметр steps записывается заранее посчитанное число, соответствующее количеству шагов мотора, чтобы достичь клавиши C2(до второй октавы). После чего вызывается функция try\_keys, которая поочередно нажимает на все клавиши для того, чтобы мы могли убедиться, что все работает хорошо.

Функции move\_motor, move\_finger и try\_keys достаточно просты, оставлю их здесь без комментариев.

void move\_motor(uint16\_t direction, uint16\_t dly){

    if (direction == 1){

        GPIO\_WriteBit(GPIOA, GPIO\_Pin\_1, Bit\_SET);

        steps --;

    }

    else{

        GPIO\_WriteBit(GPIOA, GPIO\_Pin\_1, Bit\_RESET);

        steps ++;

    }

    // make one step

    GPIO\_WriteBit(GPIOA, GPIO\_Pin\_0, Bit\_SET);

    delay(dly);

    GPIO\_WriteBit(GPIOA, GPIO\_Pin\_0, Bit\_RESET);

    delay(dly);

}

void move\_finger(uint16\_t angle){

    TIM4->CCR2 = angle;

}

/\* going to every key and push it \*/

void try\_keys(void){

    for (uint8\_t i = 0; i < 25; i++)

    {

        go\_to\_key(i, 6000);

        move\_finger(FINGER\_DOWN);

        delay(2000000);

        move\_finger(FINGER\_UP);

        delay(30000);

    }

}

Функция go\_to\_key, используемая в try\_keys представлена ниже, посмотрим на нее подробнее.

void go\_to\_key(uint8\_t key, uint16\_t dly){

    if (keys\_array[key] > steps){

        while(steps != keys\_array[key])

            move\_motor(MOTOR\_LEFT, dly);

    }

    else if (keys\_array[key] < steps)

        while(steps != keys\_array[key])

            move\_motor(MOTOR\_RIGHT, dly);

}

Все, что она делает, это смотрит, где сейчас находится палец относительно необходимой клавиши. Затем начинает двигаться в нужном направлении до тех пор, пока параметр steps не станет равен заранее посчитанному параметру, соответствующему нужной клавише.

После окончания калибровки в main вызывается функция получения байта по uart. Первым нам придет байт длины массивов. Рассмотрим эту функцию.

/\* geting one byte via USART

   if received byte is unexpected, requesting second attempt

   if second attempt is failed, blocks program and blinking C13 LED\*/

void get\_byte(uint16\_t\* byte){

    while (!USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE)){}

    USART\_SendData(USART1, 0xAA);

    while(!USART\_GetFlagStatus(USART1, USART\_FLAG\_RXNE)){}

    \*byte = (uint16\_t) USART\_ReceiveData(USART1);

    while (!USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE)){}

    USART\_SendData(USART1, 0xAB);

}

В целом эта функция реализует ранее описанный способ коммуникации между raspberry и stm.

После получения длины массивов вызывается функция получения всего трека, который заключается в двух массивах – номеров клавиш и задержек (на сколько долго держать клавишу нажатой).

/\*fills notes and delays array with bytes received via USART\*/

void receive\_music(uint16\_t \* notes, uint16\_t \* delays, uint16\_t length){

    for (uint8\_t i = 0; i < length; i++)

    {

        get\_byte(&notes[i]);

    }

    uint16\_t end\_byte = 0;

    get\_byte(&end\_byte);

    if (end\_byte != END\_OF\_NOTES){

        while (1){

            GPIO\_WriteBit(GPIOC, GPIO\_Pin\_13, Bit\_RESET);

            delay(20000000);

            GPIO\_WriteBit(GPIOC, GPIO\_Pin\_13, Bit\_SET);

            delay(10000000);

        }

    }

    for (uint8\_t i = 0; i < length; i++)

    {

        get\_byte( &delays[i]);

    }

}

Сначала эта функция получает массив клавиш, после чего, ожидает получить END\_OF\_NOTES байт, который сигнализирует о том, что массив с клавишами окончен, если на вход был получен другой код, то STM попадает в бесконечный цикл, в котором мигает светодиодом, сигнализируя об ошибке.

Если массив клавиш получен успешно, то продолжается получение массива задержек. После чего функция заканчивается.

В конце вызывается функция play\_music

void play\_music(uint16\_t \* notes, uint16\_t \* delays, uint8\_t length){

    for (uint8\_t i = 0; i < length; i++)

    {

        go\_to\_key(notes[i] - 1, 2000);

        move\_finger(FINGER\_DOWN);

        delay(delays[i] \* 5000000);

        move\_finger(FINGER\_UP);

        delay(500000);

    }

}

По виду схожая с try\_keys, но теперь она получает на вход два массива и перемещает мотор и палец согласно данным в массиве.

Код, использованный на raspberry для отправки данных:

import serial

from binascii import hexlify

from time import sleep

# arrays to be transmitted

NOTES = ["D2#", "B1", "D2#", "D2#", "A1#", "F2#", "E2", "D2#",

        "D2#", "B1", "D2#", "D2#", "A1#", "F2#", "E2", "D2#",

        "D2#", "B1", "G2#", "E2", "A1#", "F2#", "E2", "C2#"]

DURATIONS = [2, 2, 2, 2, 2, 2, 1, 1,

            2, 2, 2, 2, 2, 2, 1, 1,

            2, 2, 2, 2, 2, 2, 1, 1]

ser =  serial.Serial("/dev/ttyAMA0")

def parse\_notes(notes): # function for translating notes into array of order numbers of keys

    key\_matching = {"C1":1, "C1#":2, "D1":3, "D1#":4, "E1":5, "F1":6, "F1#":7, "G1":8, "G1#":9, "A1":10, "A1#":11, "B1":12,

                "C2":13, "C2#":14, "D2":15, "D2#":16, "E2":17, "F2":18, "F2#":19, "G2":20, "G2#":21, "A2":22, "A2#":23, "B2":24,

                "C2":25}

    keys = []

    for note in notes:

        keys.append(key\_matching[note])

    return keys

def send\_data(data): #funciton for sending one byte of data with error check

    read = hexlify(ser.read())

    print("Received: ", read)

    # if READY\_FOR\_RECEIVING byte received, start the transmition

    # otherwise break the connection

    if read == hexlify(chr(0xAA)):

        ser.write(chr(data))

        print("Sent: ", data)

        read = hexlify(ser.read())

        print("Received: ", read)

        if read != hexlify(chr(0xAB)):

            print("Error code has been received. Retrying...")

            ser.write(chr(data))

            print("Sent: ", data)

            read = hexlify(ser.read())

            print("Received: ", read)

            if read != hexlify(chr(0xAB)):

                print("The retry failed. Exiting.")

                exit()

    else:

        print("Wrong ready code. Exiting.")

        ser.close()

        exit()

if \_\_name\_\_ == '\_\_main\_\_':

    keys = parse\_notes(NOTES)

    print(keys)

    length = len(keys)

    send\_data(length)

    for key in keys:

        send\_data(key)

    send\_data(30)               # sending END\_OF\_KEYS\_ARRAY byte

    for duration in DURATIONS:

        send\_data(duration)

    ser.close()

# Вместо заключения

Работать над проектом мне понравилось, я в очередной раз окунулся в мир странностей языка С и с радостью бежал на Python, чтобы писать код для raspberry.

Проект считаю полностью законченным, или по крайней мере соответствующим ТЗ, которое я сам и писал в марте.

Работа над этим проектом дала возможность еще раз вспомнить все пройденные темы и попробовать испытать их на хоть сколько-нибудь реальном проекте. Было сложно, но мне сильно помогли заготовки кода, которые я делал на практических занятиях, иначе я бы потратил на порядок больше времени.

Как и любой проект, его можно было сделать лучше. Например:

* Еще немного оптимизировать код и полностью исключить повторяющиеся участки
* Добавить третий массив с паузами между проигрываемыми нотами, чтобы можно было играть еще красивее :)
* Учитывать при переходе к следующей ноте время, которое необходимо потратить, чтобы до нее добраться, чтобы не сбиваться с ритма.
* Учитывать время подъема и опускания пальца
* И много чего еще…

но это уже совсем другая история

