Отчет по проекту
«Piano bot»

Выполнил студент гр. 3331506/80401

Редров В.С.

Проверил Хазанский Р.Р.

Санкт-Петербург

2021г.

Содержание

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.

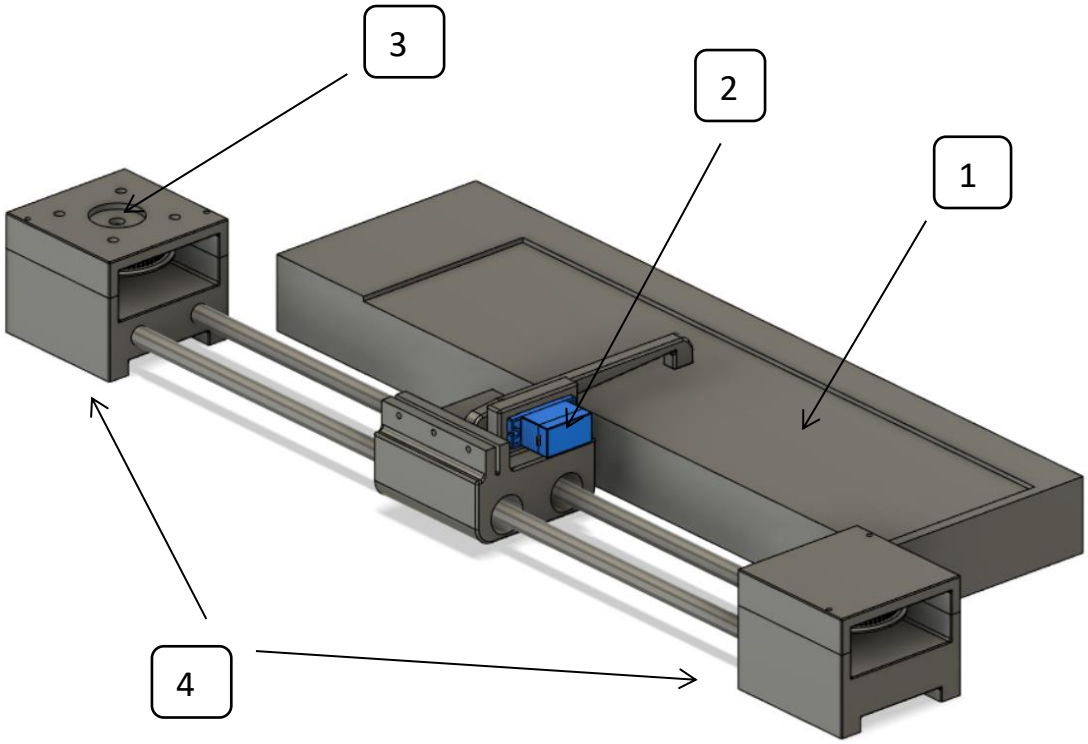

Рисунок 1 – Общий вид механической части
(1 - midi-клавиатура, 2 – сервопривод,
3 – место установки шагового двигателя)

Схема подключения

Схема подключения приведена на рисунке 2.

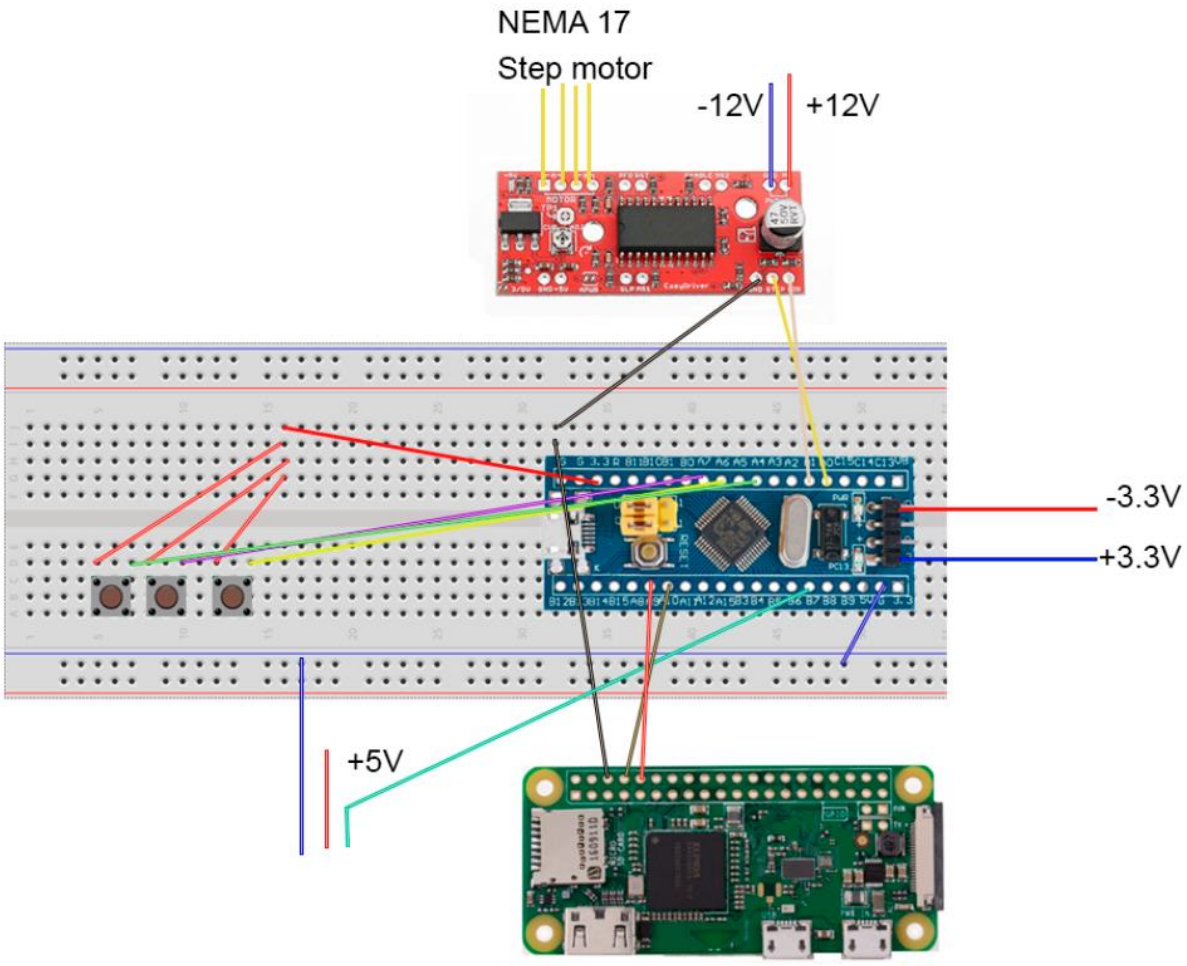

Рисунок 2 – Схема подключения

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 A10 и A9 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.

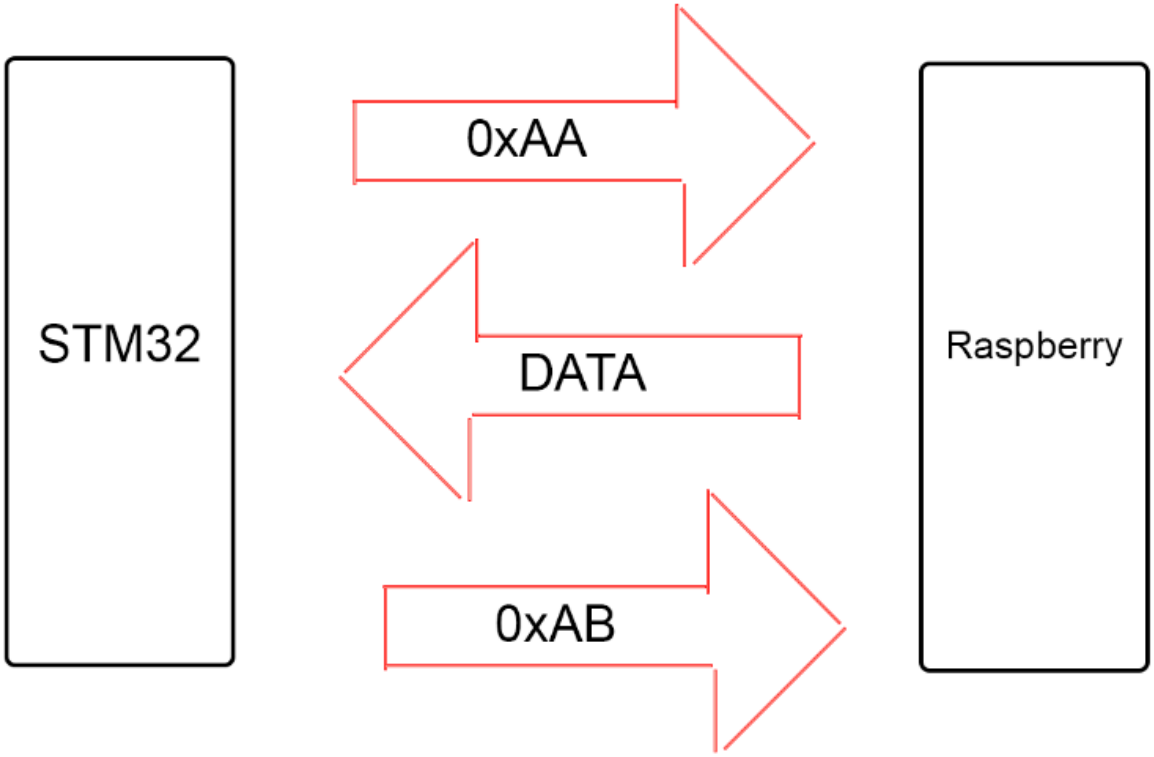

Рисунок 3 – Схема передачи данных

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 numb
ers of keys

    key_matching = {"C1":1, "C1#":2, "D1":3, "D1#":4, "E1":5, "F1":6, "F1#":7, "G
1":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 C 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