计 算 机 组 成 原 理

实

验

报

告

学　　院 软件学院

年　　级 2017 级

班　　级 1 班

学　　号 3017218063

姓　　名 刘兴宇

2019 年 5 月 22 日

天津大学

计算机组成原理上机实验报告

题目：MIDI 通信协议实现 Arduino 与 PC 间的数据传输程序

学院名称 软件学院

专　　业 软件工程

学生姓名 刘兴宇

学　　号 3017218063

年　　级 2017 级

班　　级 1 班

时　　间 2019 年 5 月 22 日

目　录

实验 3：MIDI 通信协议实现 Arduino 与 PC 间的数据传输程序

1. 实验目的

1）进一步理解数据格式在实际工程中的应用。理解 MIDI 通信协议在数据传输过程中，各种类型的数据的编码方式。

2）使用 MIDI 协议完成 Arduino 与 PC 间进行数据交换的程序，进一步理解 MIDI 的编码/解码方式。

3）使用 Arduino 构建一个有创意的 MIDI 控制器程序，可以实现 Arduino MIDI 信息播放。

2. 实验内容

1）阅读参考材料 MIDI 教程等资料，理解其数据传输的字节编码及顺序，进一步理解计算机组成原理中数据的 Little Endian/Big Endian 的含义。

2）实现 Arduino 的 MIDI 基本通信协议, 能够通过串口的 MIDI 协议实现 Arduino 与 PC 间的数据交换。

3） Arduino 的 MIDI 数据格式设置：如果下述（表 1）不能满足控制要求时，同学可以自行添加。但应该符合 MIDI 协议的要求。Arduino 在接收 PC 发送的数据过程中，如果超过 1 秒钟没有收到数据，应返回到监听状态而不应出现死锁的状态。PC 的命令超出 Arduino 功能以外的操作，Arduino 可以忽略此命令。注意 MIDI 的 channel 编号 0-0xF（十进制 0-15）在音乐术语里被定义成 channel 1 到 16，但实际 MIDI 协议发送数据时是 0-0xF。

表 1 使用部分 MIDI 协议实现 Arduino 与 PC 间的数据传输

MIDI Cmd (4bitHex)	MIDI Channel (4bitHex)	FirstByte Data Hex	Second Byte Data Hex	单位	Bit 数	说明：FirstByte/SecondByte 的 bit7 固定为 0. 剩下 bit 位用于存放数据，见（表 2）。例：要传送 8bit 数据 X，Little endian 格式，需要在 FirstByte 的 bit0-6 放 X 的 bit0-6，在 SecondByte 的 Bit0 放 X 的 bit7。以此类推。

MIDI Cmd (4bitHex)	MIDI Channel (4bitHex)	FirstByte Data Hex	Second Byte Data Hex	单位	Bit 数	说明：FirstByte/SecondByte 的bit7固定为0. 剩下bit位用于存放数据，见（ 表 2)。例：要传送8bit数据X，Little endian 格式，需要在FirstByte 的 bit0-6 放 X 的bit0-6，在SecondByte的Bit0放X的bit7。以此类推。
0xE	AD 通道号 (0..7)	1byte (0-6bit)	1byte (7-13bit)	-	14	数据使用 10bit 即 0-9bit, 10-13bit为通道号。此命令用于PC读取Arduino的指定AD通道号的 AD 转换结果时。FirstByte/SecondByte 为固定的0x11。Arduino返回实际的AD转换结果
0x9	Arduino DO编号 (0..0xF)	0/1	Reserved	-	1	用于 PC 设定 Arduino 相应的输出端状态。0 为低电平，1 为高电平。Arduino返回设置后的I/O口状态。
0xC	Arduino DI/DO编号 (0..0xF)	PC 端的此数据字节固定为0x66。Arduino 返回指定 I/O口的状态 0/1	Reserved	-	1	用于 PC 读取 Arduino 相应数字I/O的状态。0为低电平，1为高电平。
0xD	Arduino的I/O端口编号(PWM端控制) (0-0xF)	1byte (0-6bit)	1byte (bit7)，数据的bit13=1表示只读出Arduino的PWM设定值	- -	8	2 字节数据中使用 0-7bit 构成PWM数据。PC发送此命令的数据bit13=0 时，表示设定 Arduino的端口的PWM数值。 PC 发送此命令的数据 bit13=1时，表示读取Arduino的PWM口的PWM数值
0xF	0xF	0-6bit	7-13bit	ms	14	14bit数据表示MCU上电运行毫秒数值的 16 进制值。PC 端读Arduino 时间信息时发送 0xFF 0x55 0x55, Arduino返回14bit的时间信息
0xF	0x9	0-6bit	7-13bit	-	14	学号后8位的16进制值。 PC 端读 Arduino 学号信息时发送0xF9 0x55 0x55, Arduino返回14bit的学号信息
0xF	0xA	0x7F	0x7F	-	14bit	Arduino 接收到无效指令返回的信息

MIDI Cmd (4bitHex)	MIDI Channel (4bitHex)	FirstByte Data Hex	Second Byte Data Hex	单位	Bit 数	说明：FirstByte/SecondByte 的bit7固定为0. 剩下bit位用于存放数据，见（表 2）。例：要传送8bit数据X，Little endian 格式，需要在FirstByte 的 bit0-6 放 X 的 bit0-6，在SecondByte的Bit0放X的bit7。以此类推。
0xF	0xA	0x55	0x55	-	14bit	Arduino 接收到有效指令返回的信息

表 2 数据 bit 位对照表

bitNo	7	6	5	4	3	2	1	0	说明
FirstByte	X	6	5	4	3	2	1	0	格子中数值表示要发送数据的bit位
SecondByte	X	D	C	B	A	9	8	7	格子中数值表示要发送数据的bit位

4）能够将一些传感器 AD 转换结果、数字 I/O 状态等映射到相应的 MIDI 指令中，在 PC 端使用已有 MIDI 合成器软件播放 Arduino 发送的 MIDI 信息。该项内容为开放性创意实验，学生们可以通过互联网查找信息，制作出具有独特有创意的 MIDI 乐器。

3. 实验步骤（要细化如何实现的思路或流程图）

1） 硬件电路搭建，实现温度、光强数据采集及 5 个 LED 的 PWM 控制。

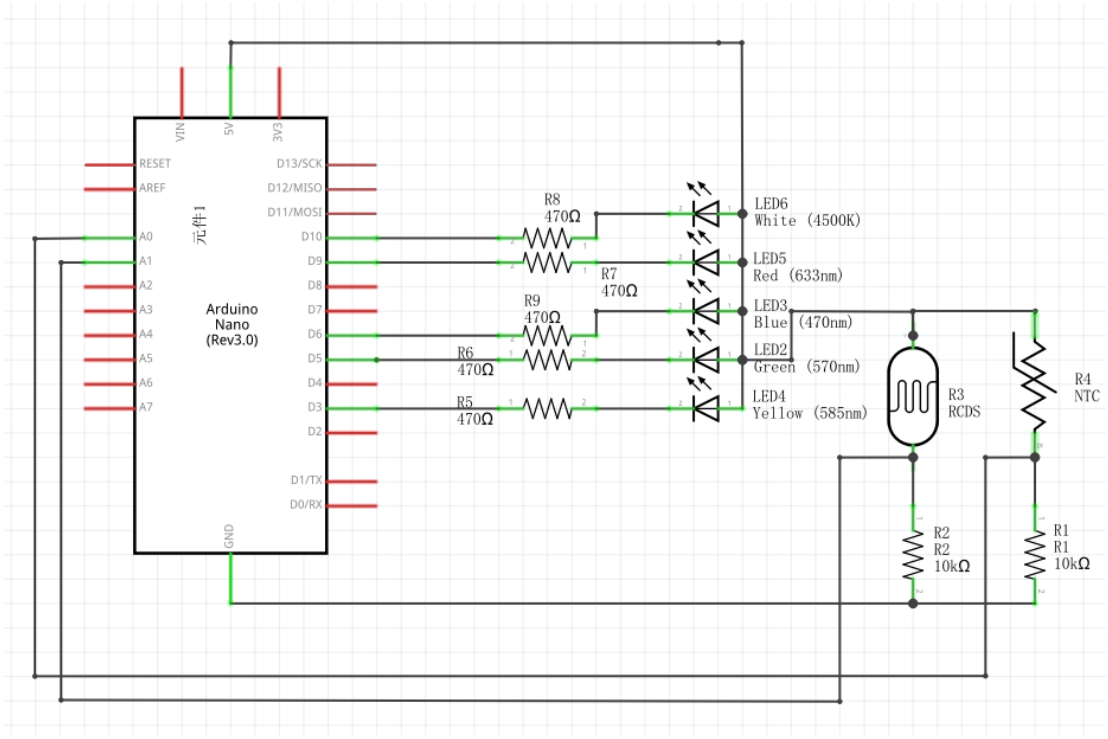


图 1 MIDI 实现温度、光强数据采集及 LED 的 PWM 控制电路

2） 实现 Arduino 的温度、光强的 ADC 数据采集功能。

3） 实现 Arduino 的 5 个 LED 的 PWM 控制功能。

4） 采用 MIDI 协议实现上述数据与 PC 间的数据交互功能。PC 可以接收到温度和光强信息，LED 现在的 PWM 状态值，和 Arduino 现在上电运行的时间。

5） 采用 MIDI 协议实现 PC 对 Arduino 上的 5 只 LED 的亮度进行控制。

6） 在 PC 端的使用 Arduino 的串口监视器、或其他串口工具测试上述 MIDI 通信协议传输数据的正确性

7） PC 端串口调试软件可以保存上述数据到 log 文件中。用于数据分析和作为实验报告中验证结果。

8） Arduino MIDI 创新实验部分，由于本实验的 Arduino 开发板只有串口（UART）能够传输 MIDI 信息，所以需要在 PC 端使用串口到 MIDI 的虚拟重定向软件 HairlessMIDI（见图 2）将 MIDI 信息定向到 MIDI 合成器（例如 Microsoft GS Wavetable Synth），实现 MIDI 信息的演奏。并且需要把 HairlessMIDI 的菜单 File/Preference 里 BaudRate 设定成与 Arduino 一致的 Baud Rate（见图 3），Arduino 开发板上实现创新 MIDI 控制器的具体步骤请同学们自行完成。

注意：在调试过程中，每次下载 Arduino 程序时，需要断开 SerialPort 和 Midi-Serial Bridge 连接，即图 2 中的 Midi-Serial Bridge On 的 CheckBox 需要处于没有对勾的状态，这样才能下载 Arduino 程序，否则 PC 端串口被独占，无法更新 Arduino 的程序。

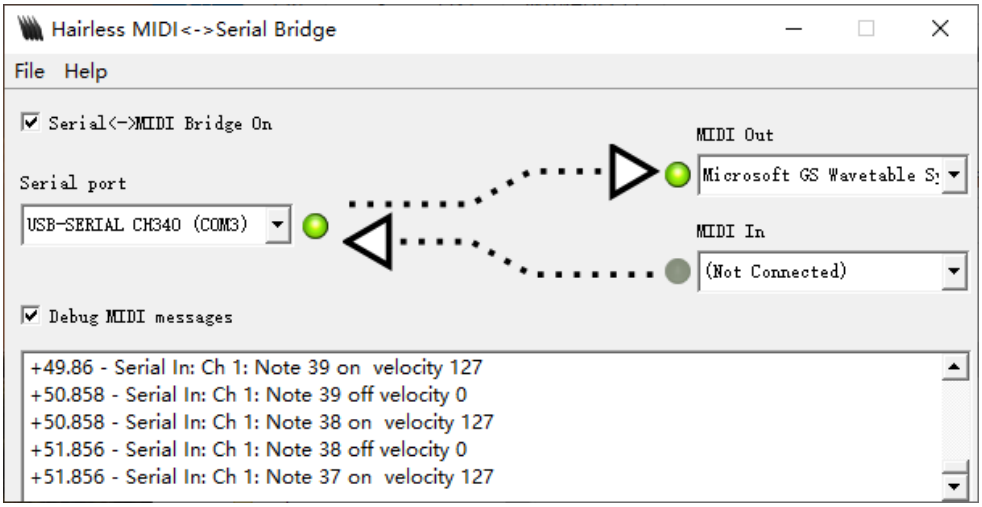


图 2 MIDI-Serial 虚拟重定向软件 HairlessMIDI 运行画面

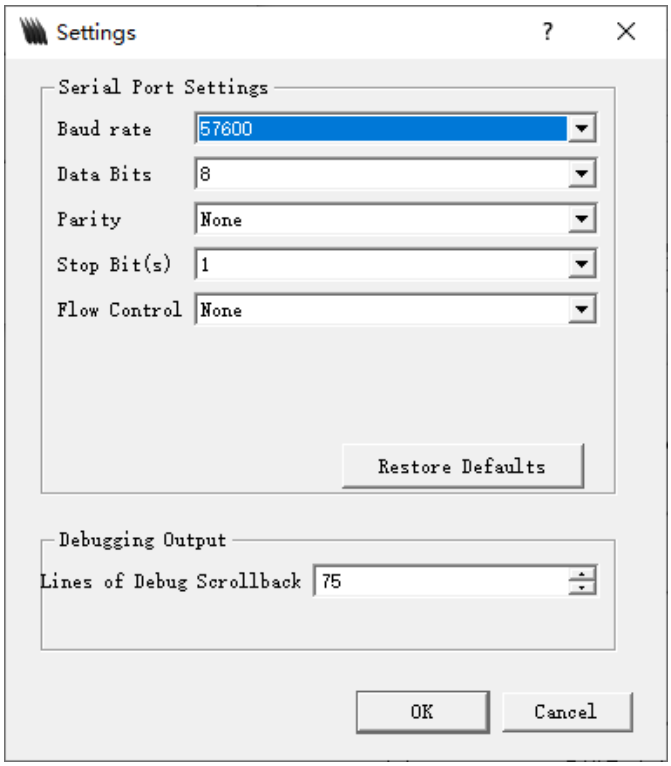


图 3 MIDI-Serial 虚拟重定向软件 HairlessMIDI 的 Preference 设置波特率与 Arduino

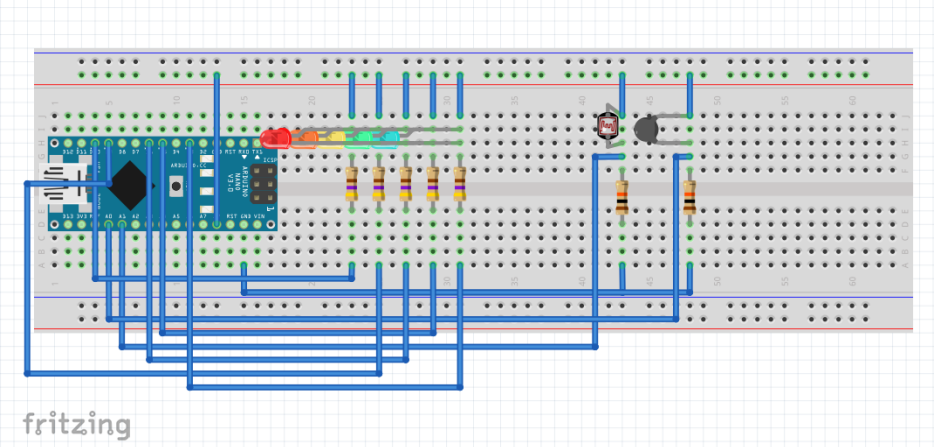
4. 实验分析

（一）实验结果

1.1 项目名称：硬件电路搭建，实现温度、光强数据采集及 5 个 LED 的 PWM 控制。

1.2 操作步骤：按照电路图搭建验证电路

1.3 实际结果描述、结论：搭建的实物图如下：



2.1 项目名称： 实现 Arduino 的温度、光强的 ADC 数据采集功能。

2.2 操作步骤：编写 test1，验证步骤 2

使用 analogRead()函数从串口直接读取对应端口得到的 AD 值，具体代码如下：

```
double getTemperature(){
  return analogRead(0);
}

double getLight(){
  return analogRead(1);
}

void loop() {
    printf("Test 1: \n");
    Serial.print("Temperature ADC: ");
    Serial.println(getTemperature());
    Serial.print("Light ADC: ");
    Serial.println(getLight());

    delay(10000);
}
```

2.3 实际结果描述、结论：

编译并运行程序，打开串口监视器，即可看到如下结果：

```
Test 1:
Temperature ADC: 535.00
Light ADC: 764.00
```

3.1 项目名称：实现 Arduino 的 5 个 LED 的 PWM 控制功能

3.2 操作步骤：编写 test2，验证步骤 3

使用 PWM 值控制灯的亮度，通过 analogWrite()函数向目标 I/O 端口写入 PWM 值，来控制灯的明暗。首先先设置连接了 LED 灯的端口为输出模式。

PWM 值范围为 0-255，其中 0 为最亮，255 为最暗。具体代码如下：

```
void setup() {
      Serial.begin(115200);
      printf_begin();
      pinMode(3,OUTPUT);
      pinMode(5,OUTPUT);
      pinMode(6,OUTPUT);
      pinMode(9,OUTPUT);
      pinMode(10,OUTPUT);
}

void loop() {
      for(int i = 255; i > 0; i--){
          analogWrite(3, i);
          analogWrite(5, i);
          analogWrite(6, i);
          analogWrite(9, i);
          analogWrite(10, i);
          delay(10);
        }

      delay(1000);
}
```

3.3 实验结果描述及结论：

编译并运行程序，观察到所有 LED 灯由暗变亮（10 毫秒延迟使得变化过程可以被分辨），每次循环间隔 1 秒

4.1 项目名称：采用 MIDI 协议实现上述数据与 PC 间的数据交互功能。

4.2 操作步骤：编写 test3，验证步骤 4、5、6

MIDI 通信协议，使用 24bit 作为一个指令单元。其指令结构为：

- 4bit Command：指示了 MIDI 指令的类型
- 4bit Channel：指示了 MIDI 指令作用的通道编号
- 16bit Data：指令所携带的数据。其中实际数据为 14bit，以小端法存储。0-6bit 存储在 Data 的第一个字节 firstByte 的 0-7 位中，7-13bit 存储在 Data 的第二个字节 secondByte 的 0-7 位中。firstByte 和 secondByte 的最高位均为 0.

设计思路：模拟 MIDI 的数据传输格式。使用串口调试助手发送十六进制的指令，然后判断 Command 命令类型来进行各项操作，最后返回响应的数据和一个 14bit（使用 16bit 的 int 模拟）的状态码：指令符合规范，有效返回 ok_code: 0xFA5555，无效返回 err_code: 0XfA7F7F（指令长度不正确、不存在对应命令、超时指令）

具体实现：

- 指令的读取：

使用 Serial.read()可以立刻从缓冲区中读取一个字节的数据。但是若缓冲区中没有数据该函数将返回空值。Serial.available()函数可以返回当前缓冲区的字节数。因此使用一个 while 循环，当缓冲区有字节可以读取时，再使用 Serial.read()读取。

```
while (true) {
  if (Serial.available() > 0) {
    temp = Serial.read();
    break;
  }
}
```

- 提取 Command 和 Channel。MIDI 数据的第一个字节包含了 Command 和 Channle，各 4bit，因此使用以下代码，提取 Command、Channel：

```
command = (temp >> 4) & 0xF;
channel = temp & 0xF;
```

- 获取 FirstByte 和 SecondByte：各读取 1 字节的数据。在读取前使用 millis()函数获取当前上电时间，记录为 start_time，在循环等待输入过程中，每次循环使用 millis()-start_time 判断已等待时间，超过 5 秒未接收到数据，跳出循环，并返回超时信息和 err_code。具体代码如下：（以获取 firstByte 为例）

```
int start_time = millis();
bool flag = true;
while (true) {
  if (Serial.available() > 0) {
    first_byte = Serial.read();
    break;
  }
  if (millis() - start_time > 5000) {
    Serial.print("Time out. Return code: ");
    Serial.println(error_code);
    flag = false;
    break;
  }
}
if (!flag) {
  continue;
}
```

- 读取 AD 值

指令格式：命令 0xE，通道范围 0-7，Data 为固定的 0x11 的指令为读取 AD 值。判断 Channel 和 Data 是否满足条件，不满足返回错误信息和 err_code，后同。

获取 AD 值的函数为：使用 analogRead()函数获取 AD 值，并且拟合为 MIDI 格式的两字节数据返回（使用 int），并返回 ok_code，具体代码如下：

```
if (channel == 0) {
  Serial.print("The AD number of temperature is: ");
  Serial.println(getTemperature());
  Serial.print("Return code: ");
  Serial.println(ok_code);
} else {
  Serial.print("The AD number of Light is: ");
  Serial.println(getLight());
  Serial.print("Return code: ");
  Serial.println(ok_code);
```

```
int getTemperature() {
  int result = 0;
  int temp = 520;
  temp = ((temp << 2) >> 2) & 0x3FFF;
  byte temp_1 = (byte)(temp & 0x7F); //low
  byte temp_2 = (byte)((temp >> 7) & 0x7F);  //high
  result = (temp_1 << 8) | temp_2;
  return result;
}

int getLight() {
  int result = 0;
  int temp = (int)analogRead(1);
  temp = ((temp << 2) >> 2) & 0x3FFF;
  byte temp_1 = (byte)(temp & 0x7F); //low
  byte temp_2 = (byte)(temp >> 7);  //high
  result = (temp_1 << 8) | temp_2;
  return result;
}
```

- 设定输出端 I/O 状态

指令格式：命令 0x9，通道范围 0-F，Data 的 firstByte 为 0 或 1，指示设置对应端口的 I/O 状态为 0（低电平）、1（高电平）。SecondByte 为任意保留值。

先将端口设为输出模式，再使用 digitalWrite（channel， LOW/HIGH）函数设置高电平或低电平，在设置后使用 analogRead()函数获取此时的 I/O 状态。最后返回 ok_code

具体代码如下：

```
if (first_byte == 0) {
  pinMode(channel, OUTPUT);
  digitalWrite(channel, LOW);
  Serial.print("New status: ");
  Serial.println(getPWM(channel));
  Serial.print("Return code: ");
  Serial.println(ok_code);
} else if (first_byte == 1) {
  pinMode(channel, OUTPUT);
  digitalWrite(channel, HIGH);
  Serial.print("New status: ");
  Serial.println(getPWM(channel));
  Serial.print("Return code: ");
  Serial.println(ok_code);
```

```
int getPWM(byte number) {
  pinMode(number, INPUT);
  int result = 0;
  int temp = (int)analogRead(number);
  temp = ((temp << 2) >> 2) & 0x3FFF;
  byte temp_1 = (byte)(temp & 0x7F); //low
  byte temp_2 = (byte)(temp >> 7);  //high
  result = (temp_1 << 8) | temp_2;
  return result;
}
```

- 读取输出端 I/O 状态

指令格式：命令 0xC，通道 0-F，Data 的 firstByte 为固定值 0x66，SecondByte 为任意保留值。

先将端口设为 INPUT，再使用 analogRead()函数读取电平值，将数据转换为 MIDI 的数据存储格式（2 字节存储 14bit）

```
if (first_byte == 0x66) {
  Serial.print("The number of channel ");
  Serial.print(channel);
  Serial.print(" is: ");
  Serial.println(getPWM(channel));
  Serial.print("Return code: ");
  Serial.println(ok_code);

int getPWM(byte number) {
  pinMode(number, INPUT);
  int result = 0;
  int temp = (int)analogRead(number);
  temp = ((temp << 2) >> 2) & 0x3FFF;
  byte temp_1 = (byte)(temp & 0x7F); //low
  byte temp_2 = (byte)(temp >> 7);  //high
  result = (temp_1 << 8) | temp_2;
  return result;
}
```

- 设置或读取端口 PWM 值

指令格式：命令 0xD，端口 0-F，Data 的第 13bit 即 SecondByte 的第 7bit 为 1，代表设置 PWM 值，反之为读取 PWM 值。

截取 Data 的第 13bit 判断设置或读取

```
byte f = second_byte >> 6;
```

使用 analogRead 读取 PWM 值:

```
if (f == 1) { //读取
  Serial.print("The PWM number of channel ");
  Serial.print(channel);
  Serial.print(" is: ");
  Serial.println(getPWM(channel));
  Serial.print("Return code: ");
  Serial.println(ok_code);
```

使用 analogWrite 设置 PWM 值

```
} else { //写
  int PWM = first_byte | (second_byte << 7);
  pinMode(channel, PWM);
  analogWrite(channel, PWM);
  Serial.print("Return code: ");
  Serial.println(ok_code);
}
```

- 获取上电时间

指令格式：命令 0xF，通道 0xF，Data 固定为 0x5555

使用 millis()函数获得 Arduino 板子当前上电时间（毫秒），并转换为 MIDI 格式

注：millis()数据返回为 long 型，应该转换为 int 型再进行处理。

```
if (first_byte == 0x55 && second_byte == 0x55) {
  Serial.print("Time: ");
  Serial.print(getTime());
  Serial.println("ms");
  Serial.print("Return code: ");
  Serial.println(ok_code);
int getTime() {
  int result = 0;
  int temp = (int)millis();
  temp = ((temp << 2) >> 2) & 0x3FFF;
  byte temp_1 = (byte)(temp & 0x7F); //low
  byte temp_2 = (byte)(temp >> 7);  //high
  result = (temp_1 << 8) | temp_2;
  return result;
}
```

- 获取学号末 8 位

指令格式：命令 0xF，通道 0x9，Data 固定为 0x5555

直接将 017218063（我的学号）进行 MIDI 格式处理后输出

```
if (first_byte == 0x55 && second_byte == 0x55) {
  Serial.print("Student Number: 17218063");
  Serial.println(getNumber());
  Serial.print("Return code: ");
  Serial.println(ok_code);

int getNumber(){
  int result = 0;
  int temp = 17218063;
  temp = ((temp << 2) >> 2) & 0x3FFF;
  byte temp_1 = (byte)(temp & 0x7F); //low
  byte temp_2 = (byte)(temp >> 7);  //high
  result = (temp_1 << 8) | temp_2;
  return result;
}
```

4.3 实验结果描述及结论：

由于串口调试助手无法正确地以十六进制形式输出返回数据，而是输出的对应十进制的 ASCII 码值，因此实验结果的数据采用十进制数据进行演示。

- 获取 AD 值:16405845 = 0XfA5555 的 MIDI 格式对应十进制值，2052 同理，后同

➢ 设置 IO 电平

➢ 读取 IO 电平

- 设置、读取 PWM 值

- 获取上电时间

- 获取学号：改为直接输出字符串

- 无效输入、超时输入

66 位无效命令，程序判断到无效命令就结束了读取，不再读取数据，返回监听状态。D9 位有效命令，程序收到后等待数据，超过 5 秒没有数据，程序中断读取，返回监听状态。

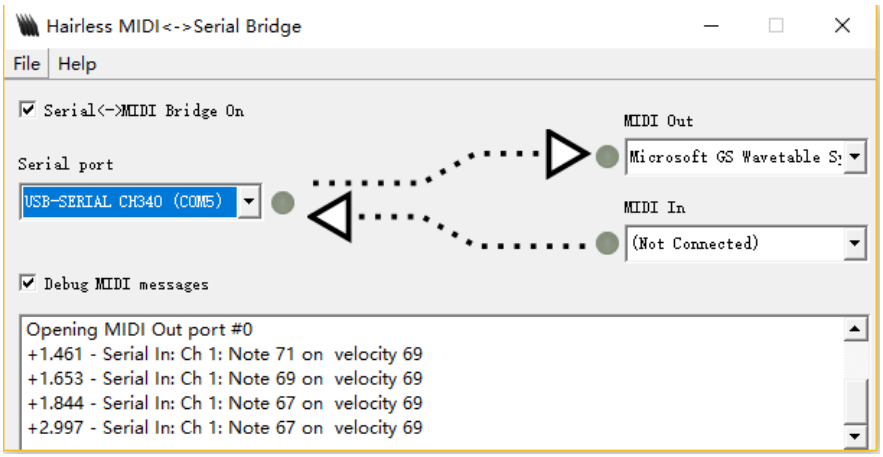
5.1 项目名称：PC 端串口调试软件可以保存上述数据到 log 文件中。

5.2 操作步骤：在串口调试助手中将数据保存到文本文档中

6.1 项目名称：Arduino MIDI 创新实验部分，演奏乐曲

6.2 操作步骤：

- 首先，Arduino Nano 板子没有支持 MIDI 协议的演奏配件。使用 HairlessMIDI 软件进行重定向，在电脑上播放 MIDI 音乐



- 使用 noteOn 函数，发送 MIDI 信息，演奏音乐。三个参数分别为 cmd：演奏的乐器种类，pitch：音符，velocity：音量

```
void noteOn(int cmd, int pitch, int velocity) {
  Serial.write(cmd);
  Serial.write(pitch);
  Serial.write(velocity);
}
```

- 音符：C 大调中音 Do 音对应的音符为 48，根据全音音阶增加 2，伴音音节增加 1 的原则，可以推出 C 大调的其它所有音阶对应的音符，以及升降调音阶。乐曲中采用的三个八度：低音 D，中音 M，高音 H，如下图定义

```
#define D1 36
#define D2 38
#define D3 40
#define D4 41
#define D5 43
#define D6 45
#define D7 47

#define M1 48
#define M2 50
#define M3 52
#define M4 53
#define M5 55
#define M6 57
#define M7 59

#define H1 60
#define H2 62
#define H3 64
#define H4 65
#define H5 67
#define H6 69
#define H7 71
```

- 速率与时长：定义了一个 bpm 值，代表每分钟的四分音符数目，即乐曲的播放速度。同时定义了 note4、note8、note16、note6 等函数代表不同音符、休止符的时长，如下图所示

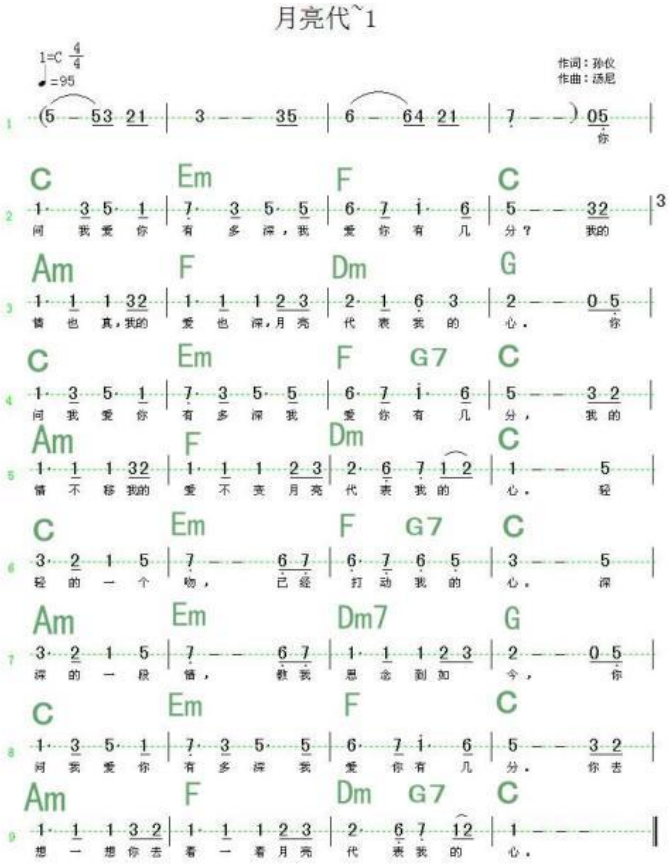
```
void note4(){
  double gap = 60.0/bpm * 1000;
  delay((int)gap);
}

void note6(){
  double gap = 60.0/bpm/1.5 * 1000;
  delay((int)gap);
}

void note8(){
  double gap = 60.0/bpm/2 * 1000;
  delay((int)gap);
}

void note16(){
  double gap = 60.0/bpm/4 * 1000;
  delay((int)gap);
}
```

- 和弦：在音符之间不添加任何时长，连续播放音符，模拟和弦。因为指令执行速度较快，很难分辨和弦音符的先后。控制音符的音量可以达到不同的和弦效果。一个和弦的例子如下图所示

```
noteOn(0x90, M2, 0x45);
noteOn(0x90, H5, 0x10);
note8();
```

6.3 实验结果描述及结论：

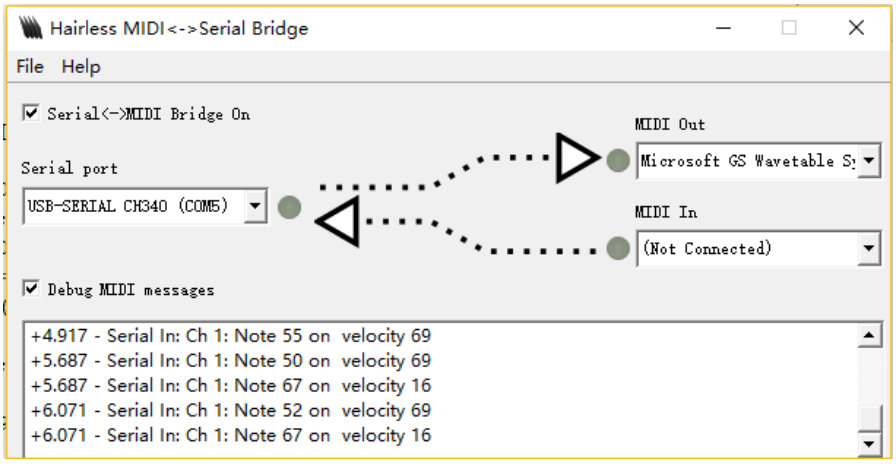
演奏 C 大调经典曲目《月亮代表我的心》（齐秦版，带前奏），乐曲简谱为：



部分代码为：

```
//                你问      我爱      你有      多深    我爱      你有
int ml[6][2] = {{D5,Ml},{M3,M5},{Ml,D7},{M3,M5},{M5,M6},{M7,Hl}};
for(int i = 0 ; i < 6 ; i++){
  noteOn(0x90, ml[i][0], 0x45);
  note8();
  noteOn(0x90, ml[i][1], 0x45);
  if (i != 5){
    note8();
  }
  note4();
}
noteOn(0x90, M6, 0x45);//几
note4();
noteOn(0x90, M5, 0x45);//分
note4();
note4();
note4();
```

播放效果：



（二）实验问题报告

项目名称：　采用 MIDI 协议实现上述数据与 PC 间的数据交互功能。

开发环境：　WINDOWS 10 X64

问题：在使用 analogRead()读取电平值前，要使用 pinMode()函数将端口设置为 INPUT 模式，才可以读取到正确的数据。同理，在使用 analogWrite()设置电平值前，要设置为 OUTPUT 模式才可以正确设置数据。

项目名称：　采用 MIDI 协议实现上述数据与 PC 间的数据交互功能。

开发环境： WINDOWS 10 X64

问题：雪梨蓝串口调试助手，以 16 进制输出数据将数据看做字符串，输出了对应的 ASCII 码，不是真实的十六进制值。

项目名称： Arduino MIDI 创新实验部分，演奏乐曲

开发环境： WINDOWS 10 X64

问题：HairlessMIDI，在每次重新下载数据到 Arduino 板子后，需要断开连接，才能打开 hairlessMIDI，在链接 Arduino，播放音乐。

5. 实验结论及心得体会

这次实验让我深入理解了 Arduino 的数据交互模式，尤其是通过串口进行数据交互的方式。以及 MIDI 通信协议的数据格式。此外还对 MIDI 演奏乐曲有了一定了解，使用 MIDI 演奏乐曲十分有趣。