Floppy Drive Orchestra

University of Colorado Boulder

Independent Study

Jeffery Lim
Jeffery.Lim@colorado.edu
Under supervision of Dr. Shalom Ruben

Contents

Chapter 1

Introduction

The goal of the floppy drive orchestra was to not only develop a working orchestra, but understand the data flow to allows music to be played from something that is used to read and write.

Chapter 2

Floppy Drive Characteristics

The first tests conducted were the analysis of a single floppy drive unit. Floppy drives come in three sizes: 8 in, 5.25 in, and 3.5 in. The large floppy drives come with different characteristics, however, the more modern 3.5 in floppy drive will be used.

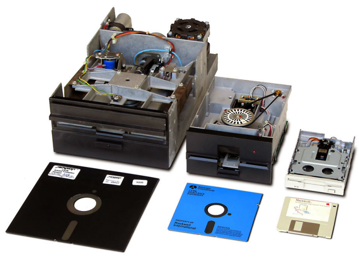

Figure 2.1: Floppy Drive of All Sizes

2.1 Power

The floppy drive takes a mini Molex cable as seen in Figure 2.2. A mini Molex cable has a 5V line as well as 12V. Modern floppy drives used to use the 12V line for the drive motor, however, more modern floppy drives only use the 5V to drive the entire drive.

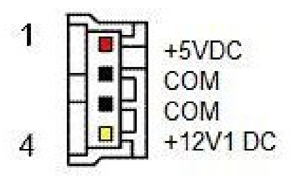

Figure 2.2: Floppy Drive Power Pinout

The power consumption of the drive when idling is an average of 50 mA and when the motor is active, it pulls 400 mA.

2.2 Floppy Pinout

The floppy pinout is shown in Figure 2.3. The bottom row, or the odd valued pins, are all grounded, where the top pins, or the even valued pins, are live. Each pin has a diferent functionality when it is grounded with the pins below. It is not necessary to connect the live pins to their respected ground pins, because the bottom row are all grounded. The actual names of the active pins are shown in Figure 2.4

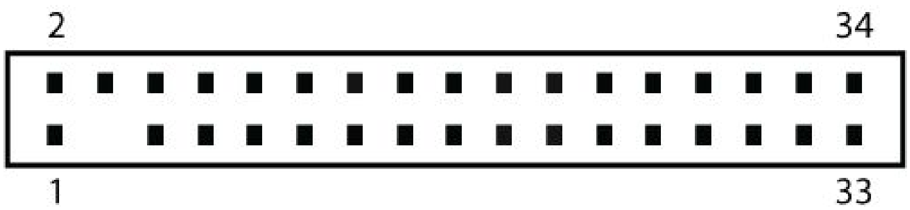

Figure 2.3: Floppy Drive Pinout

Pin	Name	Dir	Description
2	/REDWC	→	Density Select
4	n/c		Reserved
6	n/c		Reserved
8	/INDEX	←	Index
10	/MOTEA	→	Motor Enable A
12	/DRVSB	→	Drive Sel B
14	/DRVSA	→	Drive Sel A
16	/MOTEB	→	Motor Enable B
18	/DIR	→	Direction
20	/STEP	→	Step
22	/WDATE	→	Write Data
24	/WGATE	→	Floppy Write Enable
26	/TRK00	←	Track 0
28	/WPT	←	Write Protect
30	/RDATA	←	Read Data
32	/SIDE1	→	Head Select
34	/DSKCHG	←	Disk Change/Ready

Figure 2.4: Floppy Drive Pin Names

The only necessary pins to drive the floppy drive motor head are pin number 12 or 14, 18, and 20. Pins 12 and 14 are the drive select B and A. These are the drive enable pins. Some drives are B drives and others are A, so in order to test them, either one of them will need to be sleected. These pins are equivalent to a drive enable.

Pin 18 is a direction pin. The direction is what determines the direction of the motor drive. When the pin is grounded, the drive heads moves away from the pins, and when the pin is high, the drive returns towards the pins.

Pin 20 is a step pin. This pin drives the stepper motor, so every time the pin goes high, the stepper motor will go forward one tick.

2.3 Stepper Motor Movement

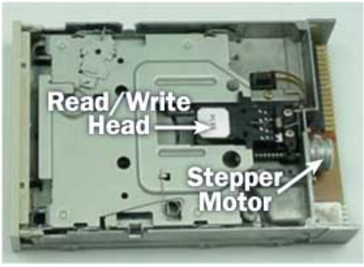

Figure 2.5: Floppy Drive With Top Off

The floppy drive motor head has a physical limit to how far it can go. To determine this value, a 1 Hz square wave is sent through to the floppy drive's direction pin (pin 18). The number of ticks is manually recorded. This can be run multiple times by simply disconnected or connecting the direction pin.

The total number of ticks a 3.5 in drive can go is 80, meaning the step pin (pin 20) needs to be toggled high 80 times before the drive reaches the limit. This means there needs to be a transition of high to low 80 times. Depending on the floppy drive, the motor will either continue to try and go forward, and others will not move.

2.4 Stepper Motor Bandwidth

Since the music will be played through the motor head, the bandwidth of the stepper motor will be a large limiting factor. For the test, a square wave from a waveform generator is connected to the step pin (pin 20), and the direction pin is connected to ground and disconnected in order to allow the motor head to switch direction.

The waveform generator is swepted from 1 Hz up until the motor head is no longer moving. The drive was able to handle up to 400 Hz, but afterwords, it was no longer consistent in terms of the speed of the drive.

This limit considerably restricts what the drive can play, and higher notes will need to be address.

Chapter 3

Wiring Diagram

Because the floppy drive only requires 5V and 400 mA each, there are two approaches in order to supply the power. A 5V power source with enough current to support all the drives is sufficient to power the system.

3.1 Power

As discussed in the previous chapter on floppy drive power characteristics, since each floppy drive, when running, draws 400 mA. For each floppy drive added to the system, the total power draw increases by 400 mA, so for 4 floppy drives, the total power consumption from the floppy drive system would be around 1600 mA, or 1.6 A.

3.2 Arduino Wiring

Chapter 4

MIDI Files

MIDI, or Musical Instrument Digital Interface, is a standard that has its own protocols, interface, and connectors. It allows a single file to contain multiple tracks for several instruments in its own channel. This allows a MIDI file to play several instruments at once, up to a maximum of 16 instruments. This advantage of the MIDI file allows multiple floppy drives to be played at once, like how a MIDI file would send data to several instruments.

4.1 MIDI Notes

The MIDI interface has notes that are mapped to specific piano keys at their respected frequencies.

Note	Octave										
	-1	0	1	2	3	4	5	6	7	8	9
C	0	12	24	36	48	**60**	72	84	96	108	120
C#	1	13	25	37	49	**61**	73	85	97	109	121
D	2	14	26	38	50	**62**	74	86	98	110	122
D#	3	15	27	39	51	**63**	75	87	99	111	123
E	4	16	28	40	52	**64**	76	88	100	112	124
F	5	17	29	41	53	**65**	77	89	101	113	125
F#	6	18	30	42	54	**66**	78	90	102	114	126
G	7	19	31	43	55	**67**	79	91	103	115	127
G#	8	20	32	44	56	**68**	80	92	104	116	
A	9	21	33	45	57	**69**	81	93	105	117	
A#	10	22	34	46	58	**70**	82	94	106	118	
B	11	23	35	47	59	**71**	83	95	107	119	

Figure 4.1: MIDI Note Number to Key

4.2 MIDI Messages

MIDI Message					
Status			Data	Data
Command (4 bits)		Channel(4 bits)	Note (8 bits)	Velocity (8 bits)
Note On	1001	nnnn	0xxxxxxx	0vvvvvvv
Note Off	1000	nnnn	0xxxxxxx	0vvvvvvv

Chapter 5

Software

5.1 MIDI Player

5.2 MIDI to Serial Driver

5.3 Arduino

5.4 SD Card(?))

Chapter 6

Arduino

6.1 Timer1 Library

6.2 Setup

6.3 Serial Reader

6.4 ISR

6.5 Floppy Drive Driver

6.6 SD Card(?)

Appendix A

A.1 Full C Code

```
// floppy - Independent Study Spring 2017 with Professor Shalom Ruben
// Needs to be used with a MiDi to Serial driver http://www.varal.org/ttymidi/
//ttymidi -b 115200 -s /dev/ttyACM0 -v
//aconnect -i and aconnect -o to find output and input drivers

#include <TimerOne.h>
#include <MIDI.h>

#define NUMDRIVES 4
#define MAXSTEPS 150
#define LED 13
#define RESOLUTION 25

//Pin 13 is reserved for LED

//drive Count starts at 12 i.e Drive 1: stepPins[1] = 12, dirPins[1] = 11
volatile int stepPins[NUMDRIVES]; //EVEN PINS
volatile int dirPins[NUMDRIVES]; //ODD PINS

//drive head position
volatile int headPos[NUMDRIVES];

//period counter to match note period
volatile int periodCounter[NUMDRIVES];

//the note period
volatile int notePeriod[NUMDRIVES];

//State of each drive
volatile boolean driveState[NUMDRIVES]; //1 or 0

//Direction of each drive
volatile boolean driveDir[NUMDRIVES]; // 1 -> forward, 0 -> backwards

//MIDI bytes
byte midiStatus, midiChannel, midiCommand, midiNote;


//Freq = 1/(RESOLUTION * Tick Count)
static int noteLUT[127];

void setup()  {

  //Init parameters
  int i,j;
  for(i = 0; i < NUMDRIVES; i++){
    driveDir[i] = 1; //Set initially at 1 to reset all drives
```

```
    driveState[i] = 0;

    periodCounter[i] = 0;
    notePeriod[i] = 0;

    headPos[i] = 0;

    stepPins[i] = (12-2*i);
    dirPins[i] = (12-2*i-1);
  }

  //Midi note setup found http://subsynth.sourceforge.net/midinote2freq.html
  double midi[127];
  int a = 440; // a is 440 hz...
  for (i = 0; i < 127; ++i)
  {
    midi[i] = a*pow(2, ((double)(i-69)/12));
  }

  //1/(resolution * noteLUT) = midi -> midi*resolution = 1/noteLUT
  for(i = 0; i < 127; i++){
    noteLUT[i] = 1/(2*midi[i]*RESOLUTION*.000001);
  }

  //Pin Setup
  pinMode(LED, OUTPUT);

  for(i = 0; i < NUMDRIVES; i++){
    pinMode(stepPins[i], OUTPUT);
    pinMode(dirPins[i], OUTPUT);
  }

  //Drive Reset
  for(i = 0; i < 80; i++){
    for(j = 0; j < NUMDRIVES; j++){
      digitalWrite(dirPins[j], driveDir[j]);
      digitalWrite(stepPins[j], 0);
      digitalWrite(stepPins[j], 1);
    }

    delay(50);
  }

  //Set Drive Pins to forward direction
  for(i = 0; i < NUMDRIVES; i++){
    driveDir[i] = 0;
    digitalWrite(dirPins[i], driveDir[i]);
  }

  delay(1000);

  Timer1.initialize(RESOLUTION); //1200 microseconds * 1 = 800 Hz, but 400 Hz output.
  Timer1.attachInterrupt(count);

  Serial.begin(115200); //Serial for MIDI to Serial Drivers
}

void loop()  {

  //Only looking for 3 byte command
  if(Serial.available() == 3){
    midiStatus = Serial.read(); //MIDI Status
```

```
    midiNote = Serial.read(); //MIDI Note
    Serial.read(); //Ignoring MIDI Velocity

    midiChannel = midiStatus & B00001111;
    midiCommand = midiStatus & B11110000;

    if(midiChannel < NUMDRIVES){
      if(midiCommand == 0x90){
        notePeriod[midiChannel] = noteLUT[midiNote];
      }
      else if(midiCommand == 0x80){
        notePeriod[midiChannel] = 0;
      }
    }
  }
}

void count(){
  int i;
  //For each drive
  for(i = 0; i < NUMDRIVES; i++){
    //If the desired drive is suppose to be ticking
    if(notePeriod[i] > 0){
      //tick the drive
      periodCounter[i]++;

      //If the drive has reached the desired period
      if(periodCounter[i] ≥ notePeriod[i]){

        //Flip the drive
        driveState[i] ^= 1;
        digitalWrite(stepPins[i], driveState[i]);

        //reset the counter
        periodCounter[i] = 0; //IT WAS == AND I COULDN'T FIGURE OUT WHY IT WASN'T WORKING

        headPos[i]++;
        //If the drive is at the maximum step, reset its direction
        if(headPos[i] ≥ MAXSTEPS){
          headPos[i] = 0;
          driveDir[i] ^= 1;
          digitalWrite(dirPins[i], driveDir[i]);
        }


      }
    }
    else{
      //Set Counter to 0
      periodCounter[i] = 0;

    }
  }

}
```