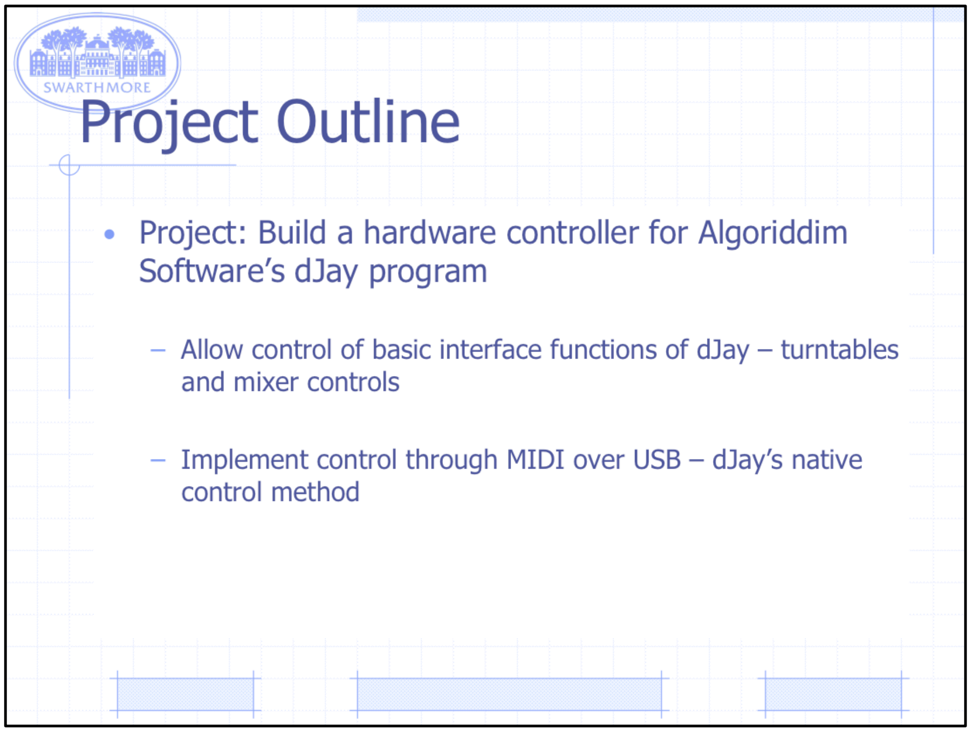
E91B Final Project: Arduino-Based Controller for DJ Software

Julian Leland
E91B, 2010

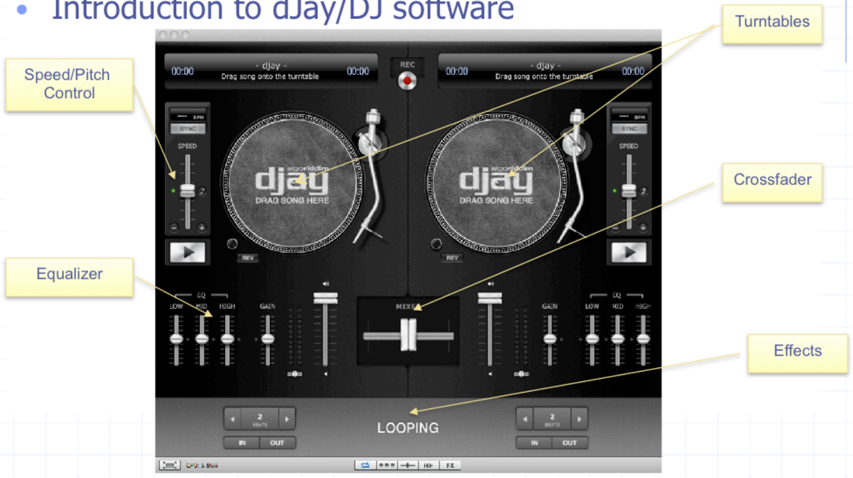


Project Outline

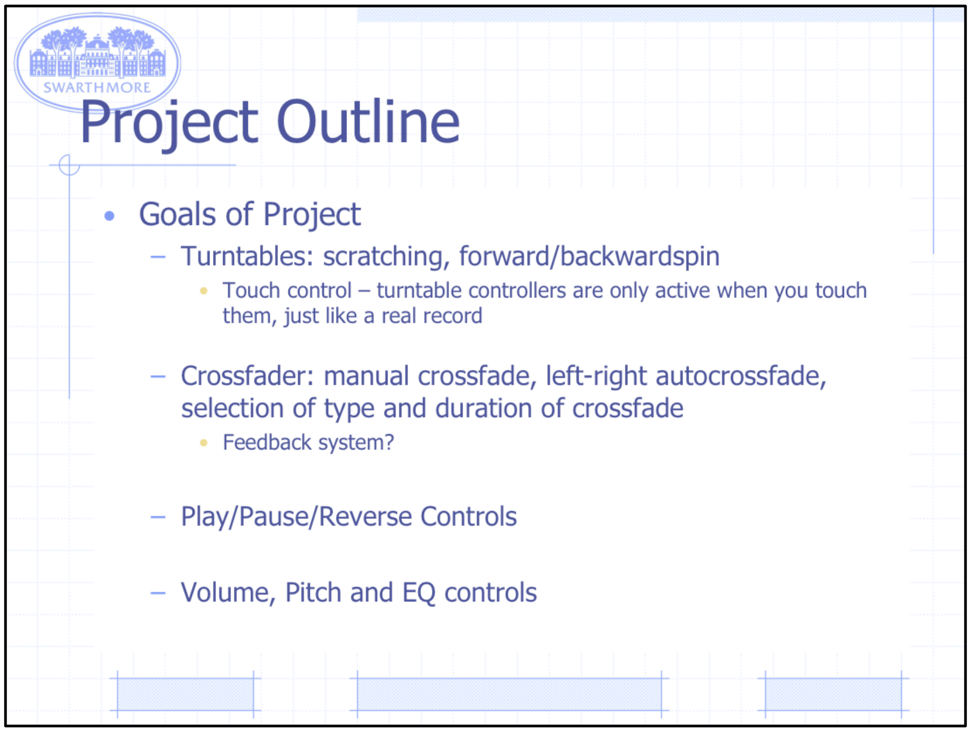
- Project: Build a hardware controller for Algoriddim Software's dJay program
  - Allow control of basic interface functions of dJay – turntables and mixer controls
  - Implement control through MIDI over USB – dJay's native control method

Project Outline

- Introduction to dJay/DJ software

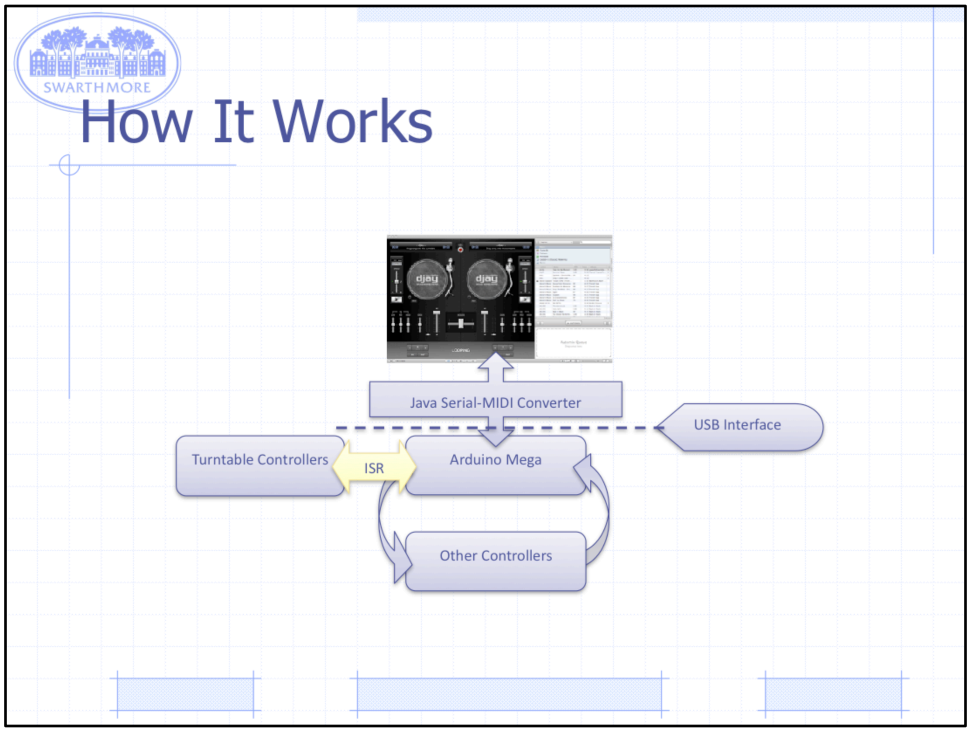


Not shown here:
•Window displaying iTunes music library
•Sampler – allows you to play short segments of audio via buttons



Project Outline

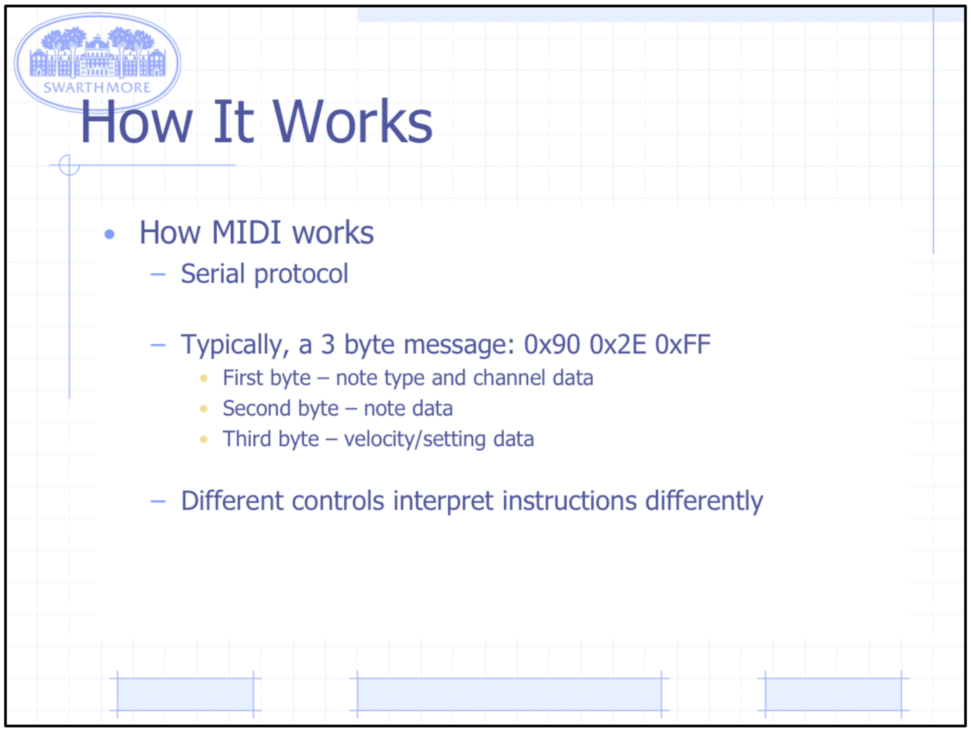
- Goals of Project
  - Turntables: scratching, forward/backwardspin
    - Touch control – turntable controllers are only active when you touch them, just like a real record
  - Crossfader: manual crossfade, left-right autocrossfade, selection of type and duration of crossfade
    - Feedback system?
  - Play/Pause/Reverse Controls
  - Volume, Pitch and EQ controls



Serial to MIDI converter: takes data coming in to serial port through USB, and tells computer that it's MIDI data.

Registers as a MIDI instrument

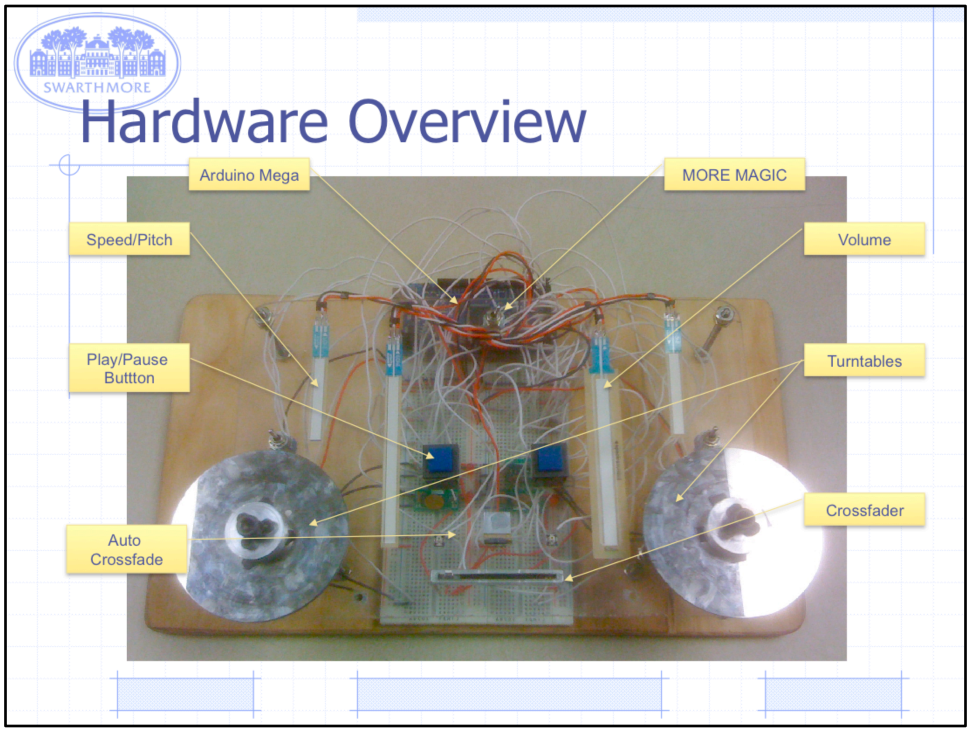
Not my creation – Mark Demers at SpikenzieLabs.com



How It Works

- How MIDI works
  - Serial protocol
  - Typically, a 3 byte message: 0x90 0x2E 0xFF
    - First byte – note type and channel data
    - Second byte – note data
    - Third byte – velocity/setting data
  - Different controls interpret instructions differently

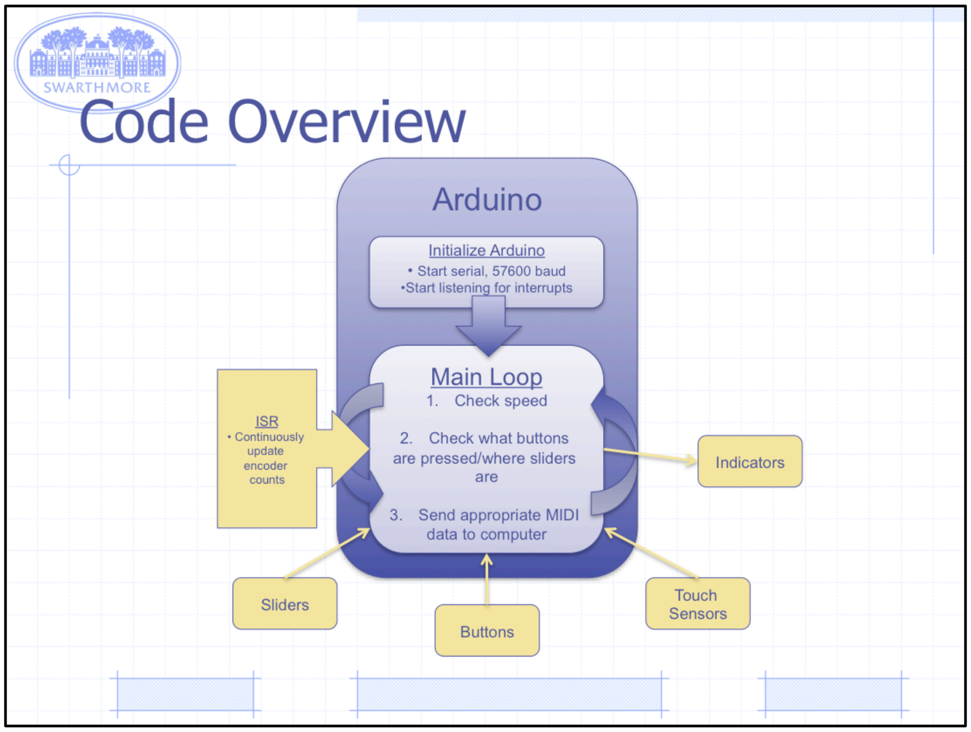
8 = Note Off
9 = Note On
A = AfterTouch (ie, key pressure)
B = Control Change
C = Program (patch) change
D = Channel Pressure
E = Pitch Wheel



Hardware Overview

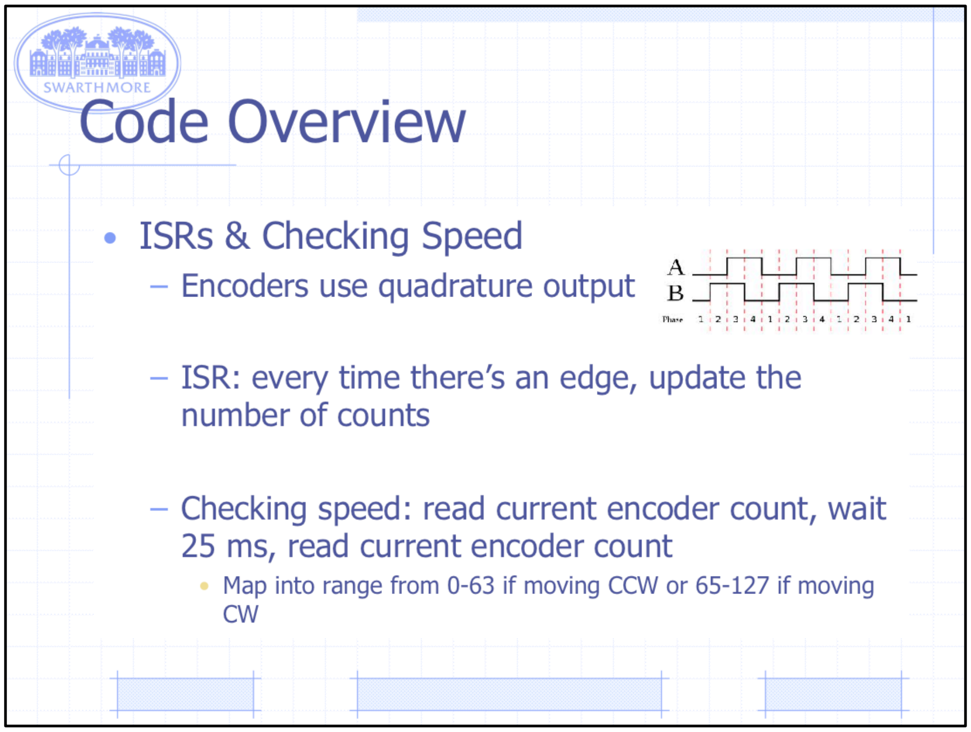
Turntables use capacitive touch sensors to detect touches on platters and on central hubs – platter scratches, hub seeks through song

MORE MAGIC cuts power to capacitive sensor chips – forces recalibration.



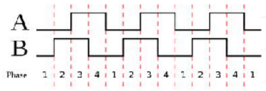
Code Overview

SWARTHMORE
Arduino
Initialize Arduino
• Start serial, 57600 baud
•Start listening for interrupts
Main Loop
1. Check speed
2. Check what buttons are pressed/where sliders are
3. Send appropriate MIDI data to computer
ISR
• Continuously update encoder counts
Indicators
Sliders
Buttons
Touch Sensors

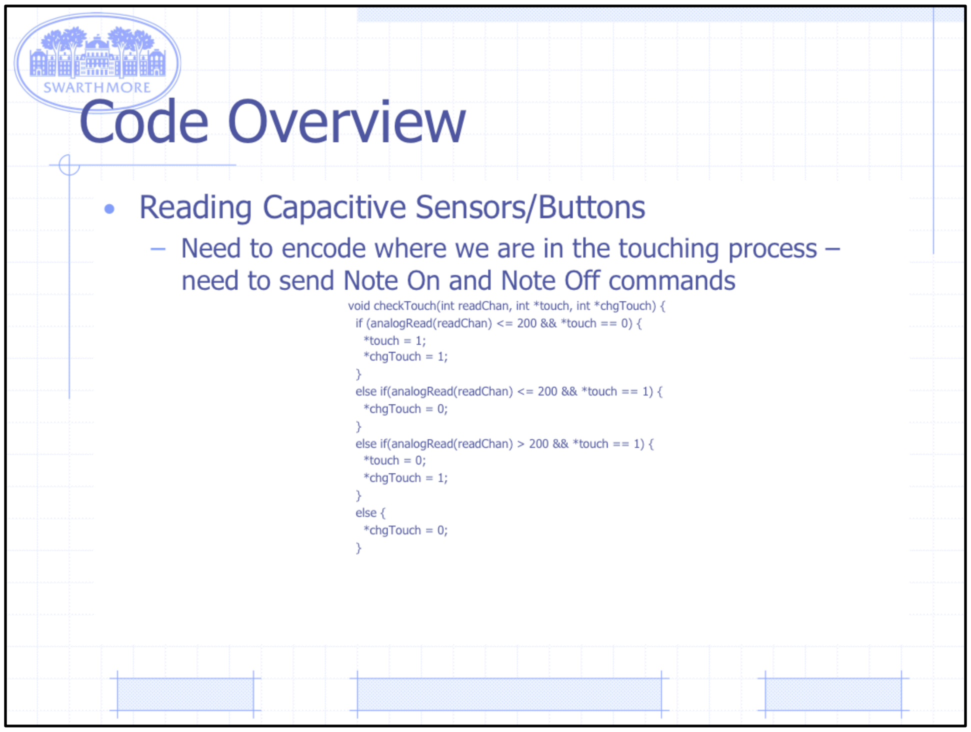


Code Overview

- ISRs & Checking Speed
  - Encoders use quadrature output
  - ISR: every time there's an edge, update the number of counts
  - Checking speed: read current encoder count, wait 25 ms, read current encoder count
    - Map into range from 0-63 if moving CCW or 65-127 if moving CW



ISR method is higher resolution than simply looking at rising or falling edges.

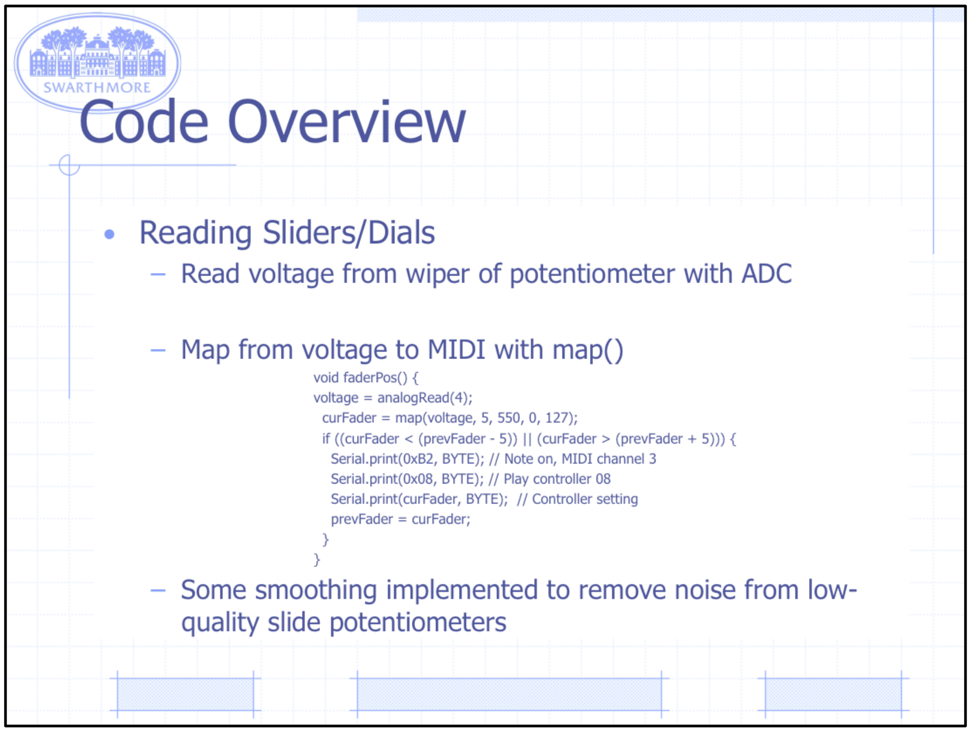


Code Overview

- Reading Capacitive Sensors/Buttons
  - Need to encode where we are in the touching process – need to send Note On and Note Off commands

```
void checkTouch(int readChan, int *touch, int *chgTouch) {
 if (analogRead(readChan) <= 200 && *touch == 0) {
  *touch = 1;
  *chgTouch = 1;
 }
 else if(analogRead(readChan) <= 200 && *touch == 1) {
  *chgTouch = 0;
 }
 else if(analogRead(readChan) > 200 && *touch == 1) {
  *touch = 0;
  *chgTouch = 1;
 }
 else {
  *chgTouch = 0;
 }
```

1) If we weren’t touched previously, and now we are, then record that we ARE being touched, and that this is a new and exciting experience for us.
2) If we were touched previously, and we’re still being touched, then record that we’re still being touched, but it’s lost its initial glitz.
3) If we were touched previously, but all of a sudden we’re not anymore, then complain! We didn’t know we had it so good until it was gone!
4) If we weren’t touched previously, and that’s not a new event, then nothing happens.

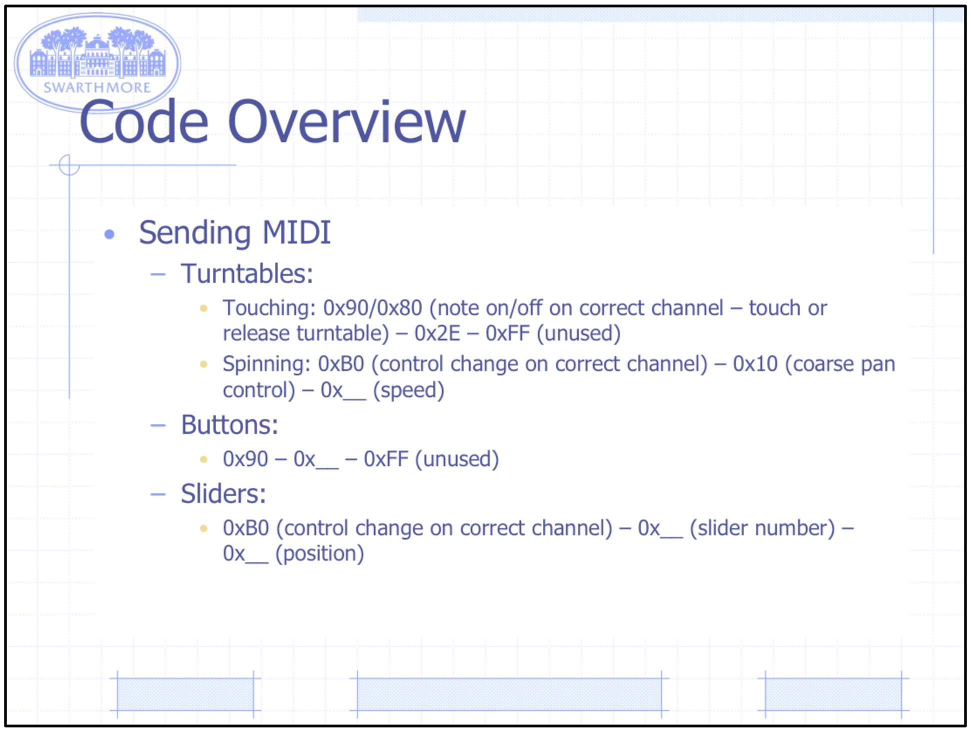


Code Overview

- Reading Sliders/Dials
  - Read voltage from wiper of potentiometer with ADC
  - Map from voltage to MIDI with map()

```
void faderPos() {
voltage = analogRead(4);
  curFader = map(voltage, 5, 550, 0, 127);
  if ((curFader < (prevFader - 5)) || (curFader > (prevFader + 5))) {
    Serial.print(0xB2, BYTE); // Note on, MIDI channel 3
    Serial.print(0x08, BYTE); // Play controller 08
    Serial.print(curFader, BYTE);  // Controller setting
    prevFader = curFader;
  }
}
```

  - Some smoothing implemented to remove noise from low-quality slide potentiometers



SWARTHMORE

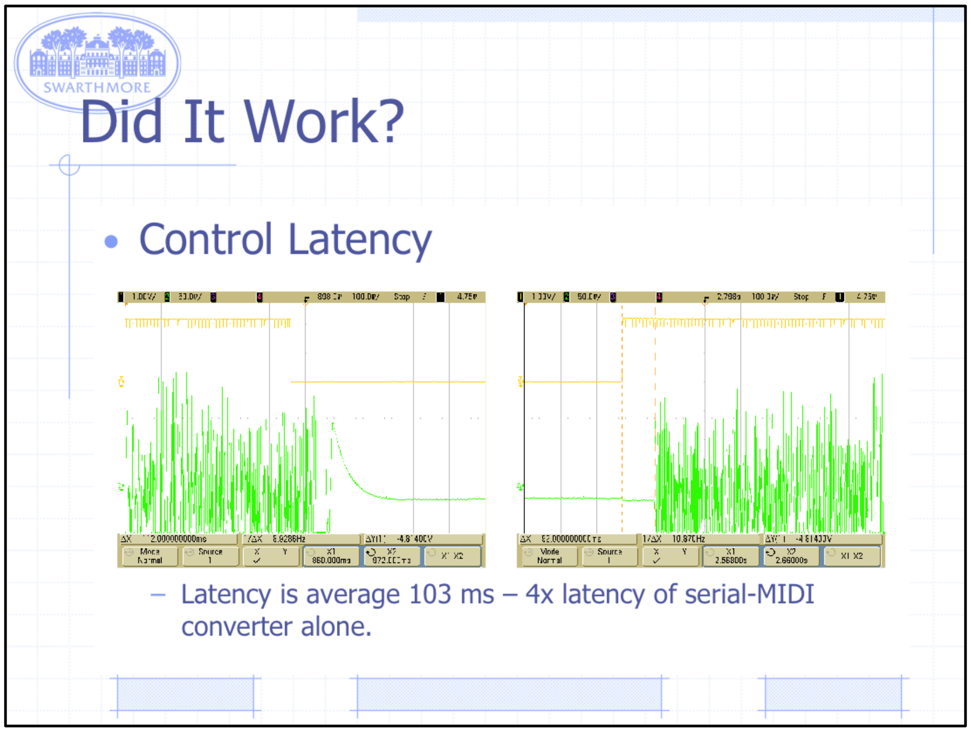
Code Overview

- Sending MIDI
  - Turntables:
    - Touching: 0x90/0x80 (note on/off on correct channel – touch or release turntable) – 0x2E – 0xFF (unused)
    - Spinning: 0xB0 (control change on correct channel) – 0x10 (coarse pan control) – 0x__ (speed)
  - Buttons:
    - 0x90 – 0x__ – 0xFF (unused)
  - Sliders:
    - 0xB0 (control change on correct channel) – 0x__ (slider number) – 0x__ (position)

All data is sent via serial using Serial.print() command

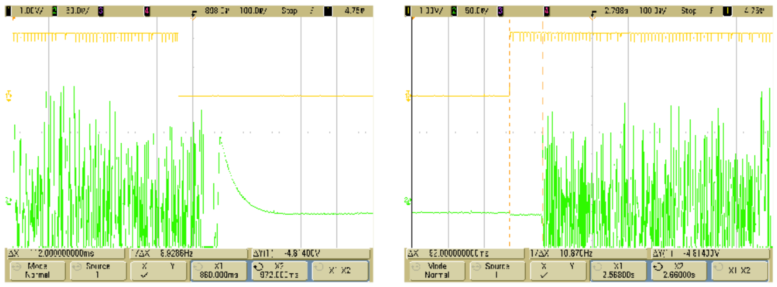
SWARTHMORE

Did It Work?



Did It Work?

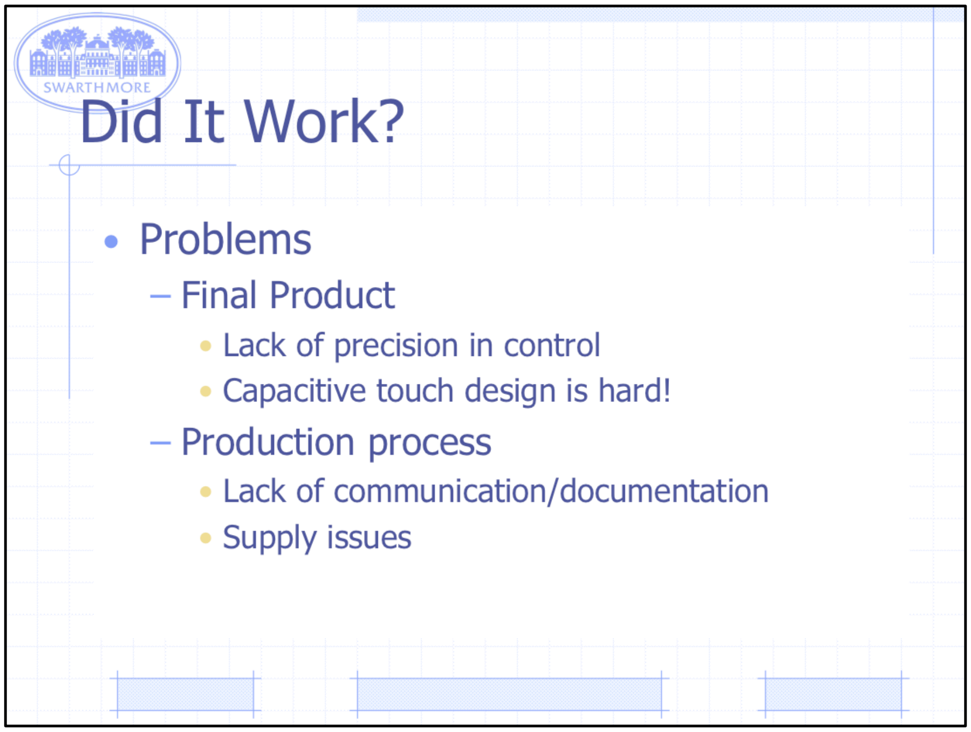
- Control Latency



  - Latency is average 103 ms – 4x latency of serial-MIDI converter alone.

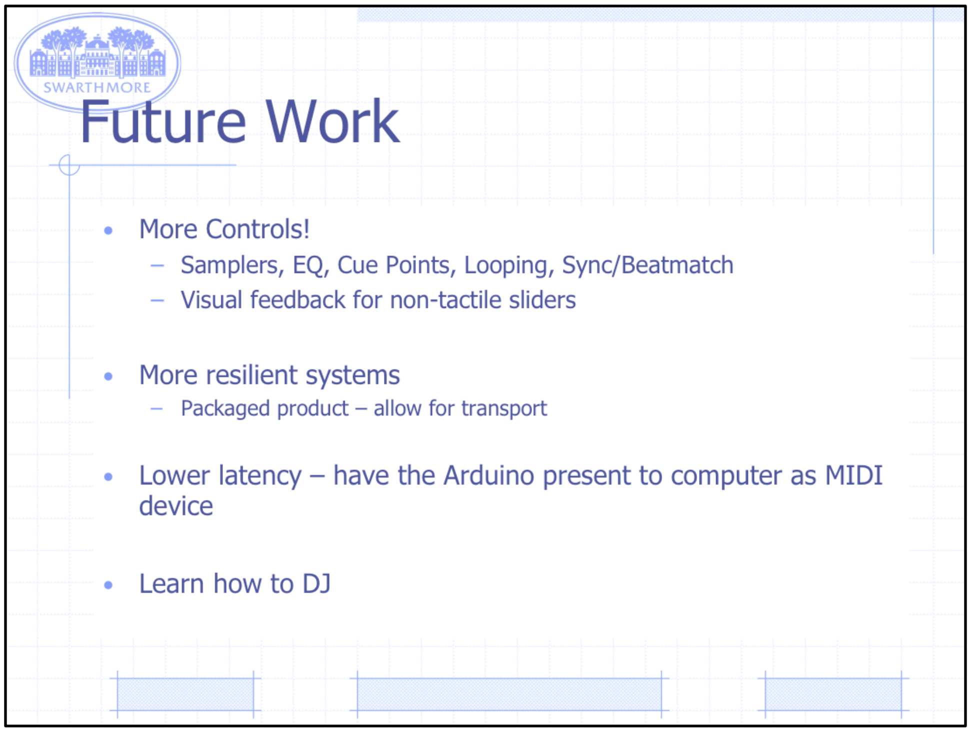
Industry standard for latency is < 10ms – should be less than 50 ms

Probably due to clunky Arduino code



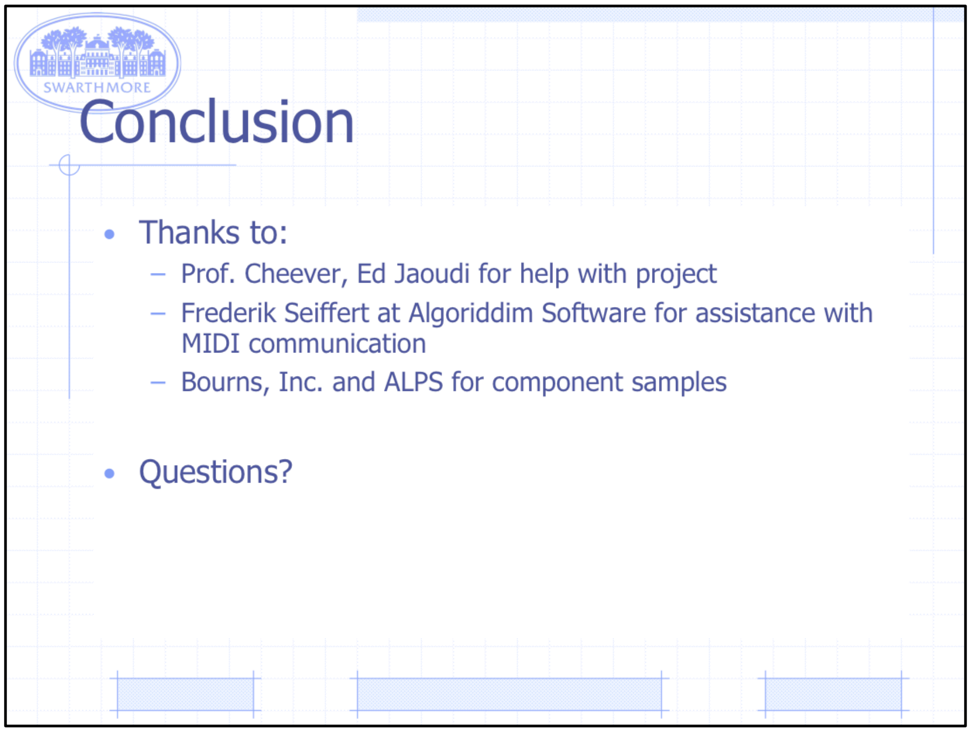
Did It Work?

- Problems
  - Final Product
    - Lack of precision in control
    - Capacitive touch design is hard!
  - Production process
    - Lack of communication/documentation
    - Supply issues



Future Work

- More Controls!
  - Samplers, EQ, Cue Points, Looping, Sync/Beatmatch
  - Visual feedback for non-tactile sliders
- More resilient systems
  - Packaged product – allow for transport
- Lower latency – have the Arduino present to computer as MIDI device
- Learn how to DJ



Conclusion

- Thanks to:
  - Prof. Cheever, Ed Jaoudi for help with project
  - Frederik Seiffert at Algoriddim Software for assistance with MIDI communication
  - Bourns, Inc. and ALPS for component samples
- Questions?