Robert DiNapoli – rjgee@hotmail.com
V1.12

Introduction

Our goal here is to control the insides of an R-81 jukebox using an Arduino. Full control on the turntable, magazine, coin mech, play counter, and some lights. With the added bonus of having an mp3 player.

Usage

The goal is to simulate normal jukebox usage. Or at least what I think is normal jukebox usage, seeing how I've never seen one in action. Assuming you are in free play mode, or have credits inserted, the 'make any selection' light will be lit. If you aren't in free play mode and don't have any credits inserted, you won't be able to play any songs. (Other commands will work though.)

Entering song numbers from 100 to 299 will store them in an internal queue. And then songs are picked off the queue one at a time to be sent to the mp3 player or the phono mechanism. When a song is playing, the 'record playing' light will be lit. And then the light will go out when the song is over.

Normally, you'll see the song number that is currently playing in the led display. The exception is when you are entering a command you will see the command number that you are entering. Once you finish

the 3rd digit of a command, the display may flash 2 or 3 times to signal it took the command, and then it will revert back to displaying the number of the song that is currently playing.

If you start to enter a song number or command and realize that you fat fingered it, pressing the reset button will clear the display allowing you to start over.

If you do the complete build, there are 2 switches:
Switch 1 - controls free play. If you don't have free play enabled, you will need to insert credits via grounding pin 18 on the Arduino (presumably wired to a coin mech). If no credits are inserted, the 'make any selection' light will be out out and you won't be able to play songs.
Switch 2 – controls whether we are playing music via the mp3 player or using the phono mechanism.
On startup, you'll see rst (reset) in the display for 1 second. Followed by mp3 or pho (phono) for 1 second. If you ever see rst pop up when you aren't expecting it, then you have some bad grounding issue going on that is causing your Arduino to reset.

There are timers that run to check and make sure that a magazine scan or record transfer isn't taking too long. If one of the following 4 errors occur, they are critical. We stop all system operation and clear the song list. Check your hardware.

Scan: A complete carousel rotation takes about 16 seconds. If we scan for more than 20 seconds, we will interrupt the scan, turn off the magazine motor and detent coil, and display “scn” on the display. You will need to press any key on the keypad to clear the message. Check to find out why your scan didn't complete. Magazine turning or stuck? Encoder wired in?

Scan – Stage 2. A record scan is split into 2 parts. Stage 1, we will scan asynchronously for record number – 1. In stage 2, we only need to move 1 more magazine position. There is still a small chance something could go wrong. The stage 2 timer is set to 1 second. If it takes more than 1 second to advance 1 magazine position, we turn off the detent coil, magazine motor and trigger error code “SC2”.

Record transfer: A transfer of the record from the magazine to the turntable takes between 5 to 6 seconds. If the transfer doesn't complete in 9 seconds, we turn off the toggle shift coil and transfer motor and display “trf” on the display. You will need to press any key on the keypad to clear the message. You should NOT scan for any new records until you are sure the current record has been returned – and the gripper arm is in the proper position.

Record return: Functions exactly the same as the record transfer timer except it displays “rtn” in the display.

Command cheat sheet

```
000 - Stop current song
100 to 299 - Queue the song to play
600 to 699 – Commands to control the RGB lights
700 to 730 - Set the volume - mp3 player only
750 - Normal play mode
751 - Sequential play mode
752 - Random play mode
753 – Random play mode, all folders
801 to 898 - if on mp3player, switch folder / directory ( 1 - 98)
```

```
995 – Magazine manual scan - phono only
996 - Encoder alignment - phono only
997 - Display preferences: version, folder, volume, play mode: (0 normal, 1
sequential, 2 random)
998 - Save preferences to the eeprom
999 – Reset System
```

000 – Stop current song from playing. If there are other songs in the queue, the next song will play. In phono mode, this will return the record from the turntable to the magazine (acting like the cancel button). If you happened to lose power and there is a record on the turntable at jukebox startup, you can use this command to return the record to the magazine.

100 – 299 – Queue a song to play. The queue isn't unlimited... but it's likely bigger than any amount of songs you are going to enter. Songs in the queue are not preserved at power off. The queue functions the same whether you are in mp3 or phono mode.

600 – 699 – Commands to control the lights (if you have added them). See the command cheat sheet under the lights section.

700 – 730 – Soft set the volume if in mp3 player mode. You can enter this command in phono mode, but it won't have any effect. 700 is basically muted. 730 is full volume.

750 – Normal play mode where songs are played from the queue (play mode 0)

751 – Sequential play. Will keep playing songs in sequence 1 – 200, as long as you have credits or free play mode is enabled. This setting can be saved to the eeprom so that on startup it will automatically start playing songs, but only in mp3 player mode. The reason is that on startup, we don't know if there is a record left on the turntable from a bad shutdown. So I don't want it scanning for a new song and putting a new record on the turntable. This command will function in phono mode – you will just have to enter it on every startup. (play mode 1)

A note about the queues: The way sequential and random mode work is that when a song finishes playing, it checks the queue for the next song. If there is no next song, it will call the sequential or random mode to add a new song to the queue. For example, you can have random mode turned on, and still enter a song number into the queue via the keypad. It will be the next song to play, and then it will go back to random mode. Yes, it's pretty slick.

752 – Random play. Randomly selects songs from 1 – 200. Keeps playing songs as long as credits are available, or you are in free play mode. You can save this to the eeprom so that on startup it will automatically start playing songs. This command only functions in mp3 player mode – don't want to stress the phono magazine scan with searches. (play mode 2)

753 – Random play mode, all folders. Randomly selects a song from 001 to as many you have on the sd card in all folders. Keeps playing songs as long as credits are available, or you are in free play mode. You can save this to the eeprom so that on startup it will automatically start playing songs. This command only functions in mp3 player mode. It will display a number on the digit display panel, but it won't mean much – you won't be able to correlate the absolute song number back to any kind of list you are keeping. You can read more notes in the code about this as to why.

801 – 898 – Change the folder number on the mp3 player. The mp3 player can access 98 folders. Each folder can hold 200 songs. (Technically 99 folders with 255 songs each, but I'm reserving 99 for future use. And keeping it limited to 200 songs in each folder to align with the jukebox). This setting is also saved via eeprom. So if you have folder 1 for Oldies, Folder 2 for Christmas, Folder 3 for Rock, it will remember which folder you are in on startup (as long as you enter the 998 command to save current settings.)

995 – Enables scan of the magazine by pressing the 1 button on the keypad. Press 0 to exit this mode. This functions similar to the scan switch. Phono mode only, and it doesn't work if it thinks a record is already playing.

996 – Encoder alignment mode. This shows the position of the encoder in the display with numbers from 0 to 99. Press 0 on the keypad to exit. It's a nice function to help align your encoder. Phono mode only. If a record is playing while you have this command entered, the record won't return to the magazine until you exit this routine by pressing zero. (This is pretty handy if you are testing records – just go into encoder alignment mode, and you can keep changing records without risk of the gripper arm trying to transfer a record back to the magazine or the phono arm returning to starting position.)

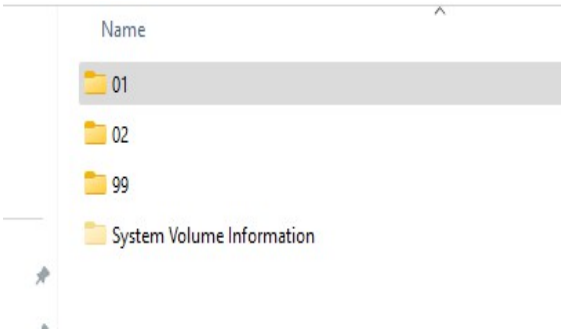
997 – Display preferences stored in the eeprom. Software version. Current folder (mp3 mode only). Last song played (that has been saved to the eeprom via the 998 command. It is NOT the current last song played.) Volume setting (for mp3 mode). Play mode (normal (nor), sequential (seq), random (ran), random all folders (ral)).

998 – Save preferences to the eeprom (current folder, current last song played, volume, play mode). On machine startup, these are read back into memory and used. This will also save custom color data for lights, and the current light pattern you are using (assuming you have them added).

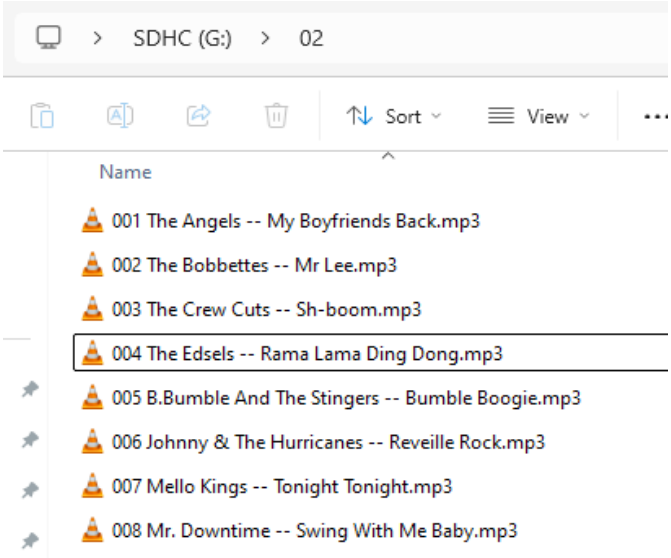
999 – Reset the System. You will get a prompt “Reset – 1 for yes 0 No”. Press 1 to reboot. Press any other number to exit without rebooting. I added this mainly to be able to switch between mp3 player and phono since the physical switches are only read on startup.

Setting up your sdcard for the mp3 player

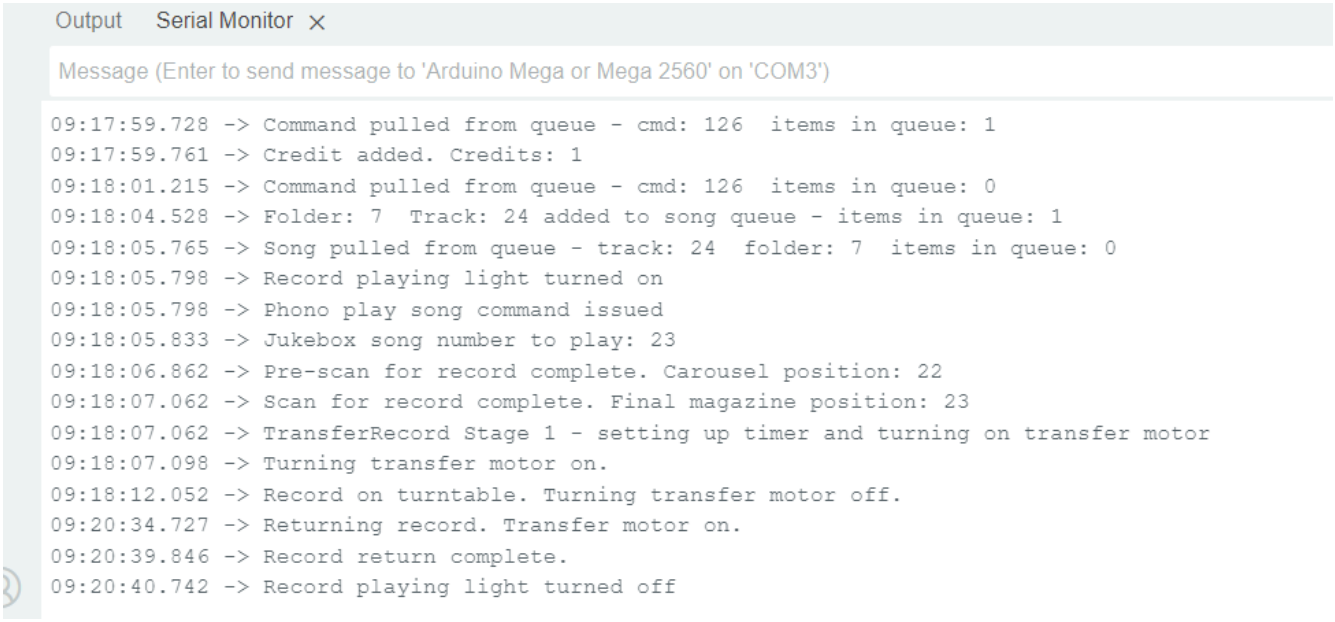
Create as many folders as you want. They must be numbered 01 – 99. 2 digits ONLY. You cannot have 2 digits followed by a folder name. I think the dfmini player supports up a 32gb sdcard. I'm using a 32gb card and it works fine. I haven't tested larger cards.



In each folder, number the songs from 001 – 200. Songs MUST begin with the 3 digit number, although you can have any text you want after it, such as the artist or song name. One of my earlier tries was to have the songs numbered from 100 – 299 to match up with what the jukebox has on the display tags. But the mp3 player will only accept song numbers up to 255. Trying to play song 256 results in nothing being played (at least with the Umlife mp3player). See appendix 1 for a notes on a quick program I wrote that will auto number any songs in a directory so that you don't have to do it by hand.



For debugging, connect the Arduino to your computer running the Arduino IDE. It outputs lots of debugging information.



Notes:

Internally in the queue, songs are stored numbered 1 – 200, regardless if we are in phono or mp3 mode. Song numbers 1 – 200 match up nicely with the mp3 player format. When the phono is active, we'll adjust the song number on the fly to match up with the phono player (100 – 299). When we detect the song number being greater than 199, we'll activate the toggle shift coil to play the flip side of the record.

Building the assembly

Parts list

Required:

- Arduino Mega 2560 R3 – yes, you need this one and not a cheaper one because we're going to be using a lot of input/output pins
- 4 channel relay module. (You'll need one more relay if you want to support the play counter. I'm using an 8 channel module, but only because I had it on hand)
- 5v 1 amp power supply or 7.5 – 24vdc 1 amp power supply if building the voltage regulator below
- TM1637 4 digit led display
- Lot of wire
- We are assuming that your jukebox has the original power supply in it. We will need that to supply 28vac and 28vdc for the magazine motor, transfer motor and detent coil.

Optional

- 2 small switches (to control free play or phono / mp3 mode. You can control it from code if you want to skip the switches)
- Molex connectors and header pins for interfacing with the encoder/keypad/mech unit. You could also just jam wires in the connector if you are looking for the cheapest option.

Optional, to control the Jukebox make any selection and record playing lights

- 2 5k resistors, ¼ watt
- 2 220 ohm resistors ¼ watt
- 2 2n2222 transistors

Optional, power supply

- LM7805 voltage regulator, TO220 case
- TO220 heatsink
- 2 10uf capacitors, 25v
- 1 .1uf capacitor

Optional, if building the mp3 player

- dfmini mp3 player
- RCA female phono plugs

General build notes

If building any of the circuits, test on a breadboard,if you have access to one first.

I made the mistake of using a stiff wire, which made my assembly into the case a bit tough. Use a flexible wire if you have one available.

Overall, this is a pretty flexible design. You can wire in just the mp3 player. Or just the phono. Or both. You can use an off the shelf 5v power supply. Or build the one below. You can even just wire the encoder and display portion and have an easy to use off the shelf device to check the encoder position.

If you are using an already built 5v power supply, wire up the keypad and led module first. This will allow you to test functionality as you wire up the rest of the components.

Use molex or similar connectors when possible so that you can remove individual parts from the machine.

I didn't give thought ahead of time to putting the mp3 player in a box. Now, I have to open the case to remove the sdcard every time I want to add songs.

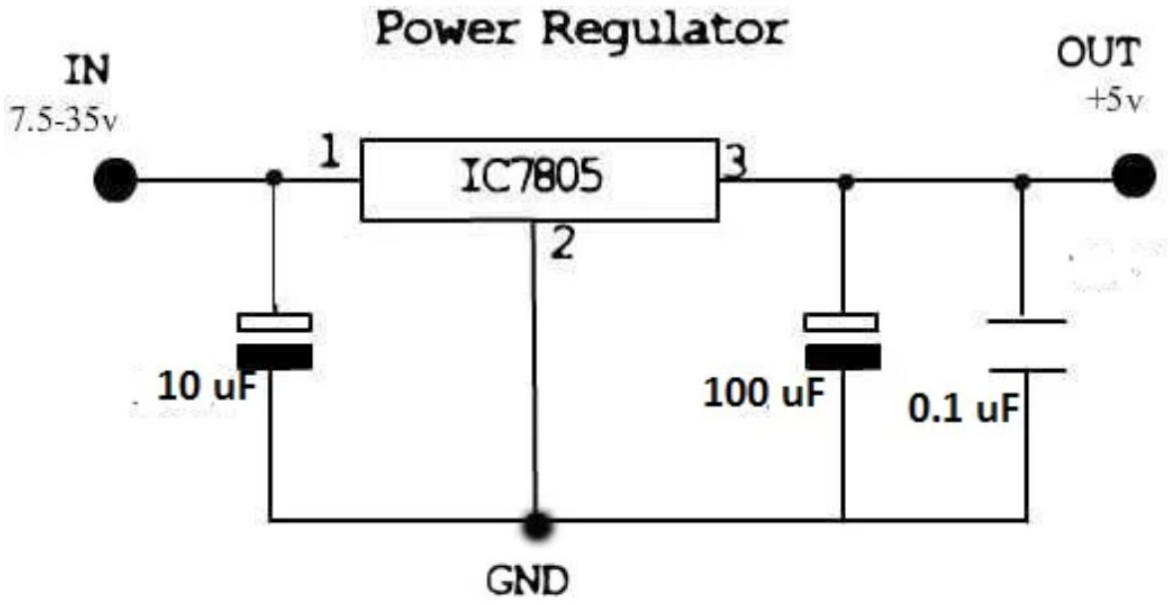
I didn't want to spend money on an Arduino shield board that had screws for wire connection at the time of build, so my wires are stabilized in the Arduino using hot glue. I'll probably redo this at a later time.

Ground wire: Each component that needs a ground should get it's own ground connection from the voltage regulator. (For example, do not wire a ground from the power supply to the encoder, and then from the encoder to the mech unit wiring.) At least that's my understanding for best practice on grounding.

Build – power supply

If you have a 5v 1 amp power supply, you can skip this. You will be powering several things from the power supply, so it will need multiple output leads. (USB powering the Arduino seemed to be enough power to run everything, but I wasn't completely comfortable with it and opted for powering everything with a separate power supply. I didn't have a 5v power supply handy, so this simple circuit will take care of allowing you to use almost any other power supply you have on hand. The LM7805 will handle an input voltage DC of up to 35v. But the higher the voltage you feed it, the harder it has to work to convert the voltage down to 5vdc. I'm running on a 12vdc 1amp power adapter that I had on hand.

Step 1: The Datasheet

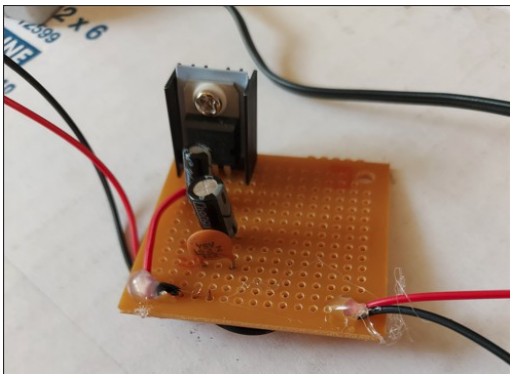


We're going to be building this circuit shown above (and in the link shown below). The diagram shown above has a 100uf capacitor hanging off leg 3 of the LM7805, while my build used a 10uf capacitor. You can use either. The 10uf /100uf capacitors DO have a polarity. Be sure that you are connecting the negative lead of the capacitor to ground! The .1uf capacitor has no polarity and you can connect it in either direction.

https://www.instructables.com/7805-Regulator-5V-Module-Easy-Tutorial/

Once you've built it, use a multimeter and confirm you are getting 5v out of it.

Mine looks like this:

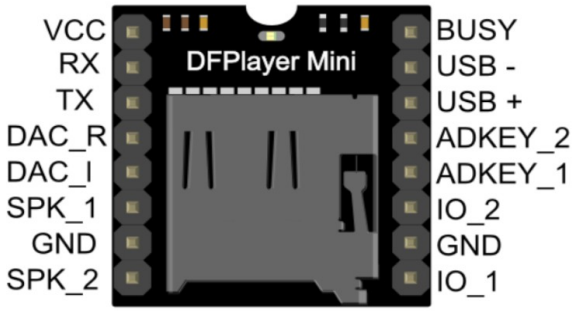


Build – switches

This is fairly simple. One end of each switch goes to ground. (as in ground or the negative lead from the power supply you just built). The other ends of the switches go to pins 15 and 14 of the Arduino. The first switch (pin15) is for free play. The second switch (pin14) controls whether we are using an mp3 player or controlling the physical phono mechanism. You can skip these switches and control everything from software if you want by adding a few lines of code. You can see a picture of my switches in a plastic case a few pages down.

Build – mp3 player

Here is a picture of the dfmini mp3 player



Rotate your dfmini so that the sdcard is facing the right. The busy pin should now be in the top left corner. We will call that pin 1, and pins 1-8 will run across the top row. Pin 9 will be VCC, and continue to pin 16 as SPK_2. We don't have true stereo here. (The dfmini supports does support it by using the DAC_R and DAC_L pins, but it was stressing the amp that I was using too much.) We're going to wire as follows:

Pin 1 (busy) to pin 48 on the Arduino

Pin 7 (gnd) to ground on your voltage regulator

Pin 9 (vcc) to +5v

Pin 10 (rx) to pin 12 on the Arduino

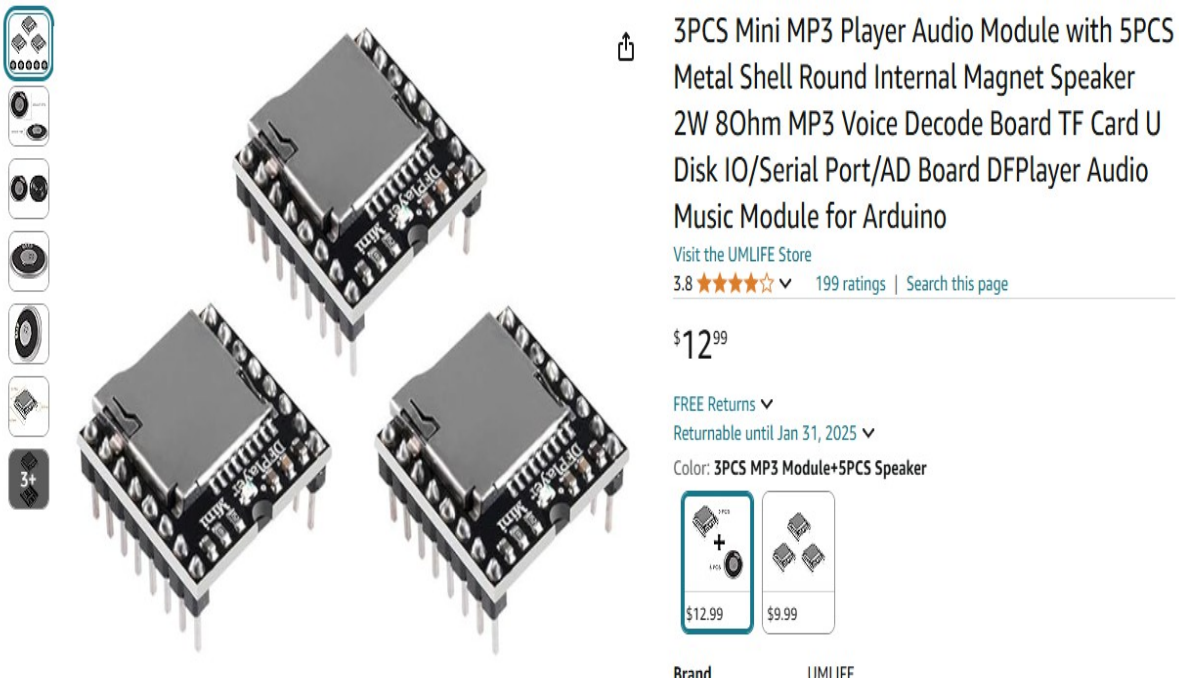
Pin 11 (tx) to pin 13 on the Arduino

Pin 14 (speaker 1) to one leg of each RCA phono jack

Pin 15 (gnd) to ground on your voltage regulator

Pin 16 (speaker 2) to the second leg of each RCA phono jack

There are differences in some of the dfmini players. The one I'm using is from “Umlife”. Here's a current picture of it on Amazon.



Build – light control

We're going to be building 2 very simple circuits. Each one to control one of the jukebox lights. Right now, the software only controls the 'make any selection' light, and the 'record playing' light. You can wire for more lights if you want to extend my code by using any unused digital pins on the Arduino.

Make any selection light

Leg 1 of the 2n2222 transistor goes to ground.

Leg 2 of the 2n2222 transistor goes to a 5k resistor. The other end of the 5k resistor goes to pin 46 on the Arduino (via a long wire).

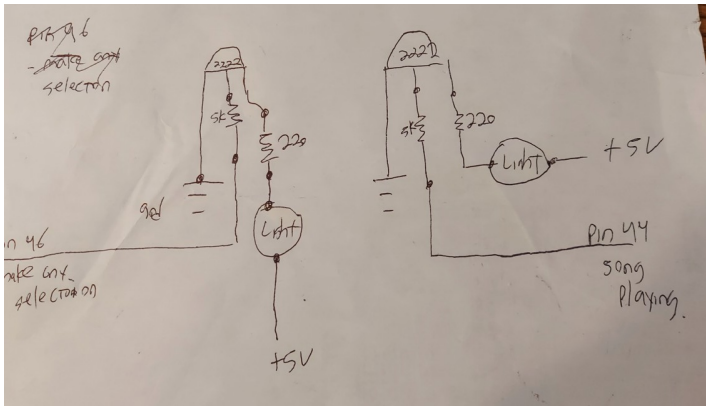
Leg 3 of the 2n2222 transistor goes to a 220 ohm resistor. The other end of this resistor gets wired to 1 leg of the light bulb (via a long wire). The other side of the light bulb goes to +5v on our voltage regulator (via a long wire).

Record playing light

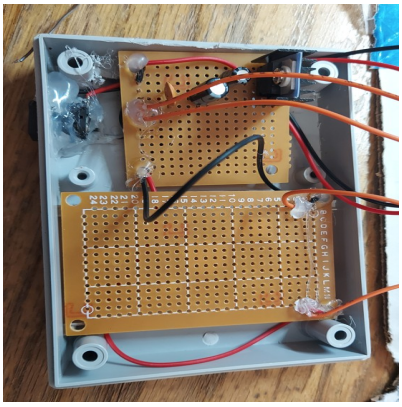
Leg 1 of the 2n2222 transistor goes to ground.

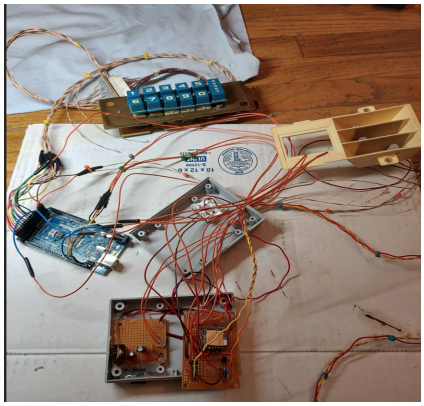
Leg 2 of the 2n2222 transistor goes to a 5k resistor. The other end of the 5k resistor goes to pin 44 on the Arduino (via a long wire).

Leg 3 of the 2n2222 transistor goes to a 220 ohm resistor. The other end of this resistor gets wired to 1 leg of the light bulb (via a long wire). The other side of the light bulb goes to +5v on our voltage regulator (via a long wire).



At this point, we're done with all the circuit build work. We just have a lot of wiring left to do. My voltage regulator, mp3 player, and light control circuits fit in a small box as shown below in the 2nd picture. I'll put a label on the case when finished to hide the imperfections in drilling.





Wiring – LED display

Via some long wires:

Wire Pin 50 from the Arduino to the DATA pin on the TM1637 module

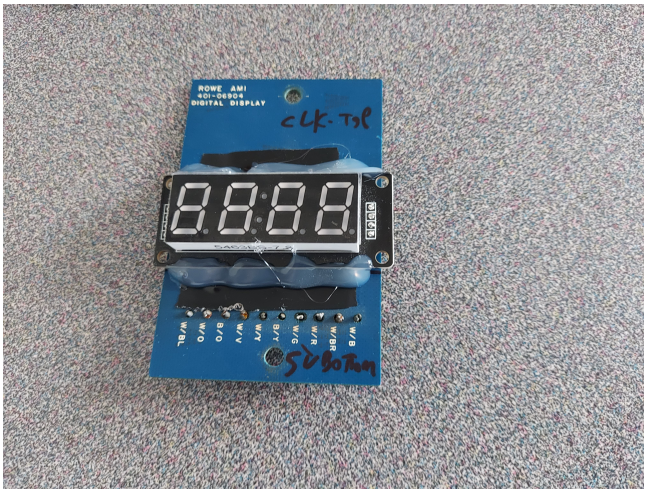
Wire Pin 52 from the Arduino to the CLK pin on the TM1637 module

Wire +5v from our voltage regulator to +5v on the TM1637 module

Wire GND from our voltage regulator to GND on the TM1637 module

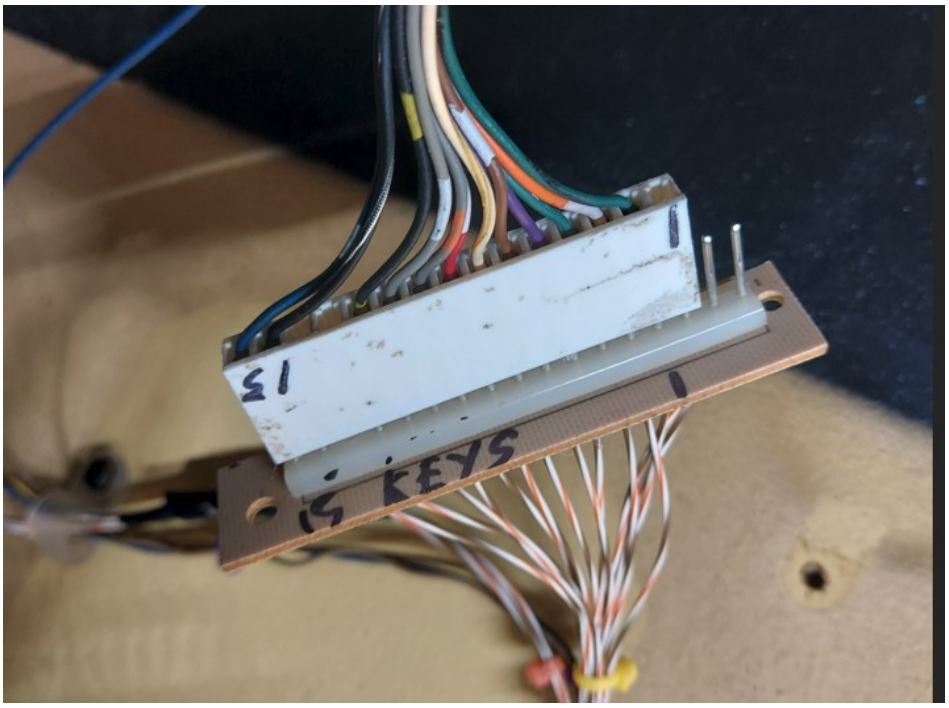
I was really hoping to get this working using the original display unit. It required an external led control chip and lots of wiring. I was close to have it working satisfactory and managed to blow one of the led segments. Sigh. So TM1637 to the rescue. Even though this has 4 digits, you can only see 3 digits in the jukebox display. Plus, its a bit bigger and brighter than the original display. You can just hot glue the the TM1637 display to a piece of cardboard, plastic, or whatever you have lying around. I

used the original display board, since mine was toast.



Wiring – Keypad

Using this picture as a guide, notice I've labeled pin 15 on the left (reset key). You should see the black wire second from the left (pin 14 - ground). This may not follow the manual... not sure. (Ignore the 2 extra pins shown on the right. I'm just using some header pins I had on hand.)



Via some long wires:

Wire Pin 2, green wire from the keypad to pin 42 on the Arduino (0 key)

Wire Pin 3, white/orange wire from the keypad to pin 40 on the Arduino (1 key)

Wire Pin 4, green/white wire from the keypad to pin 38 on the Arduino (2 key)

Wire Pin 5, purple/white wire from the keypad to pin 36 on the Arduino (3 key)

Wire Pin 6, brown/white wire from the keypad to pin 34 on the Arduino (4 key)

Wire Pin 7, yellow/white wire from the keypad to pin 32 on the Arduino (5 key)

Wire Pin 8, red/white wire from the keypad to pin 30 on the Arduino (6 key)

Wire Pin 9, grey/orange wire from the keypad to pin 28 on the Arduino (7 key)

Wire Pin 10, grey/white wire from the keypad to pin 26 on the Arduino (8 key)

Wire Pin 11, black/yellow wire from the keypad to pin 24 on the Arduino (9 key)

12 – not used

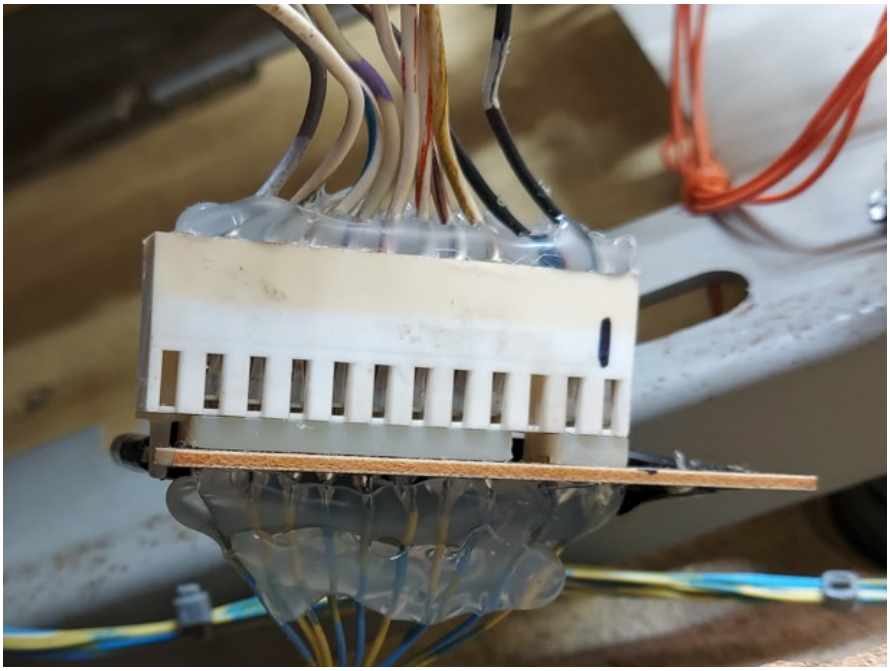
13 – not used

Wire Pin 14, black wire from the keypad to GND on the voltage regulator

Wire Pin 15, blue/white wire from the keypad to pin 22 on the Arduino (reset key)

Wiring – Encoder

Using this picture as a guide, where the black/white wire on the right is pin 1



Wire pin 1 from the encoder to GND

Pin 2 – strobe – not used – in retrospect I should wire this up as insurance that the encoder is turning when I'm telling it to turn. Maybe coming in a 2nd version.

Pin 3 – not used / key

Wire pin 4 from the encoder to pin 37 on the Arduino (128 position)

Wire pin 5 from the encoder to pin 35 on the Arduino (64 position)

Wire pin 6 from the encoder to pin 33 on the Arduino (32 position)

Wire pin 7 from the encoder to pin 31 on the Arduino (16 position)

Wire pin 8 from the encoder to pin 29 on the Arduino (8 position)

Wire pin 9 from the encoder to pin 27 on the Arduino (4 position)

Wire pin 10 from the encoder to pin 25 on the Arduino (2 position)

Wire pin 11 from the encoder to pin 23 on the Arduino (1 position)

Pin 12 – not used

Wiring – Relay module to Arduino

Wire GND from the voltage regulator to GND on the relay module

Wire +5v from the voltage regulator to +5v on the relay module

Wire pin 53 from the Arduino to pin IN1 on the relay module (transfer motor)

Wire pin 51 from the Arduino to pin IN2 on the relay module (detent coil)

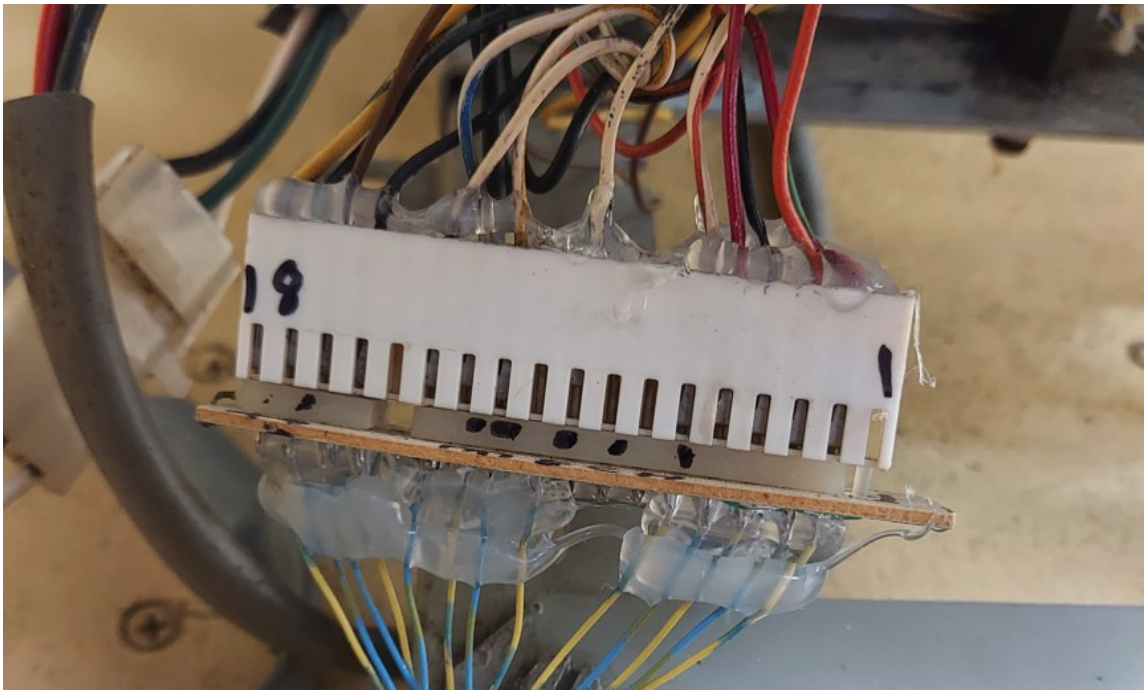
Wire pin 49 from the Arduino to pin IN3 on the relay module (magazine motor)

Wire pin 47 from the Arduino to pin IN4 on the relay module (toggle shift coil)

Wire pin A14 from the Arduino to pin IN5 on the relay module (play counter)

Wiring – Mech control to relay module and Arduino

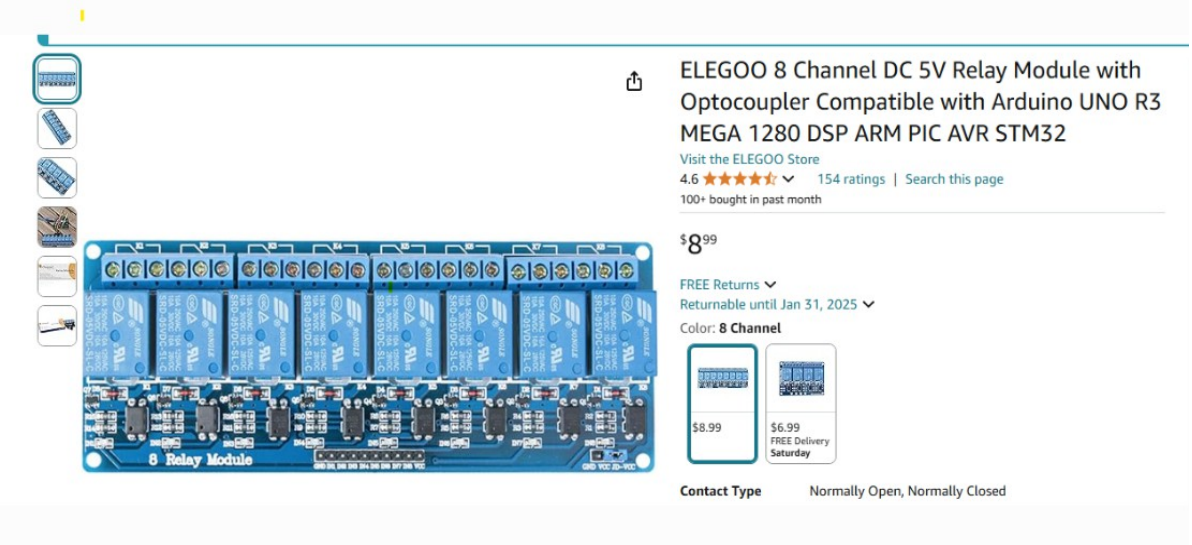
Using this as a guide, where the yellow/black wire is labeled pin 18 (may differ from manual)



Wire pin 15 (ground) from the mech control to GND on the Arduino or voltage regulator

Wire pin 13 (cam switch 2 status) from the mech control to pin 21 on the Arduino

Each relay module will have 3 terminals. I'm going to label them from the left starting with letter A. So relay 1 has letters ABC, relay 2 has DEF and so on. We're assuming here, that the middle connection (B, for example) is common. And the 3rd connection (C, for example) is normally open.



Wire pin 15 (gnd) from the mech control to pin B on relay 1

Wire pin 16 (transfer motor, 28vac) from the mech control to pin C on relay 1

Wire pin 5 (28vdc) from the mech control to pin E on relay 2

Wire pin 12 (detent coil) from the mech control to pin F on relay 2

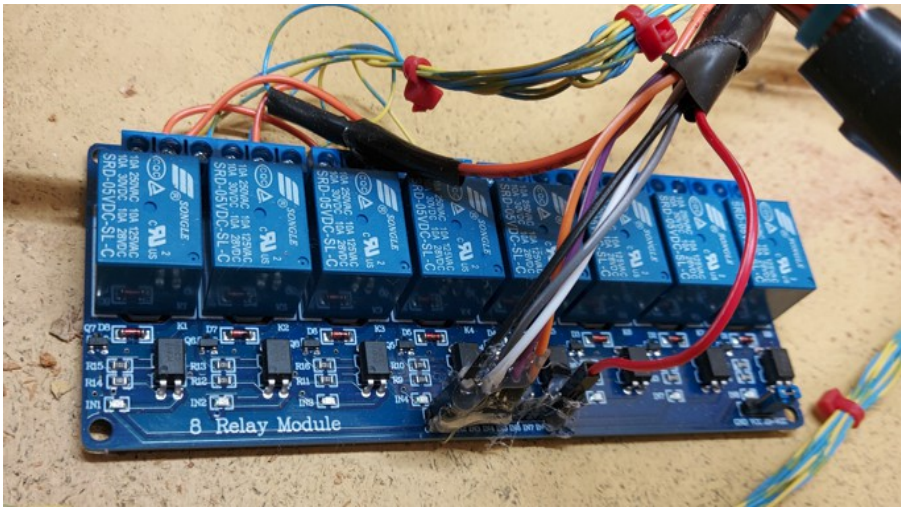
Wire pin 15 (gnd) from the mech control to pin H on relay 3

Wire pin 16 (magazine motor, 28vac) from the mech control to Pin I on relay 3

Wire pin 5 (28vdc) from the mech control to pin K on relay 4

Wire pin 6 (toggle shift coil) from the mech control to pin L on relay 4.

(As a shortcut in your wiring, I wired pin 5 from the mech control to E on the relays. and then I just jump pin E to pin K. and a similar shortcut for ground on the relay modules.)



Wiring – Play counter

Wire pin 5 (28vdc) from the mech control to N on relay 5.

Wire one of the connection wires from the play counter to GND.

Wire the 2nd wire from the play counter to O on relay 5

The play counter only increments when playing records. It does not increment for mp3 song plays.

Wiring – Power the Arduino

Finally! Wire +5v from the voltage regulator to +5 in the Arduino and GND from the voltage regulator to GND on the Arduino.

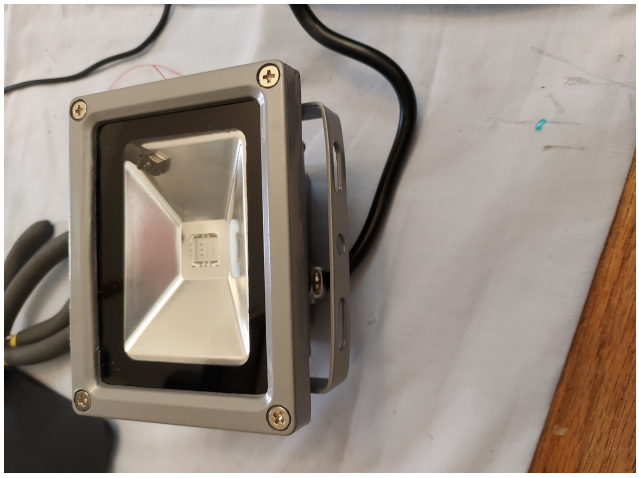
Wiring – Coin mech

I don't have a coin mech, so I could not fully test this yet. But if Pin 18 from the Arduino is grounded, it will insert a credit if free play mode is off. It's run via an interrupt, so it shouldn't lose any coins. And it has debounce built in... but not really tested yet.

Bonus! – Project within a project – Front panel lights

I wasn't overly happy with the front panel florescent lights. Plus, one of them was out, and you already know it's a bit of an oddball size so I can't just walk into a home depot to buy one. I had a few 10w floodlights lying around. I was planning to use them for some landscaping, but never got around to it.

They look like this.

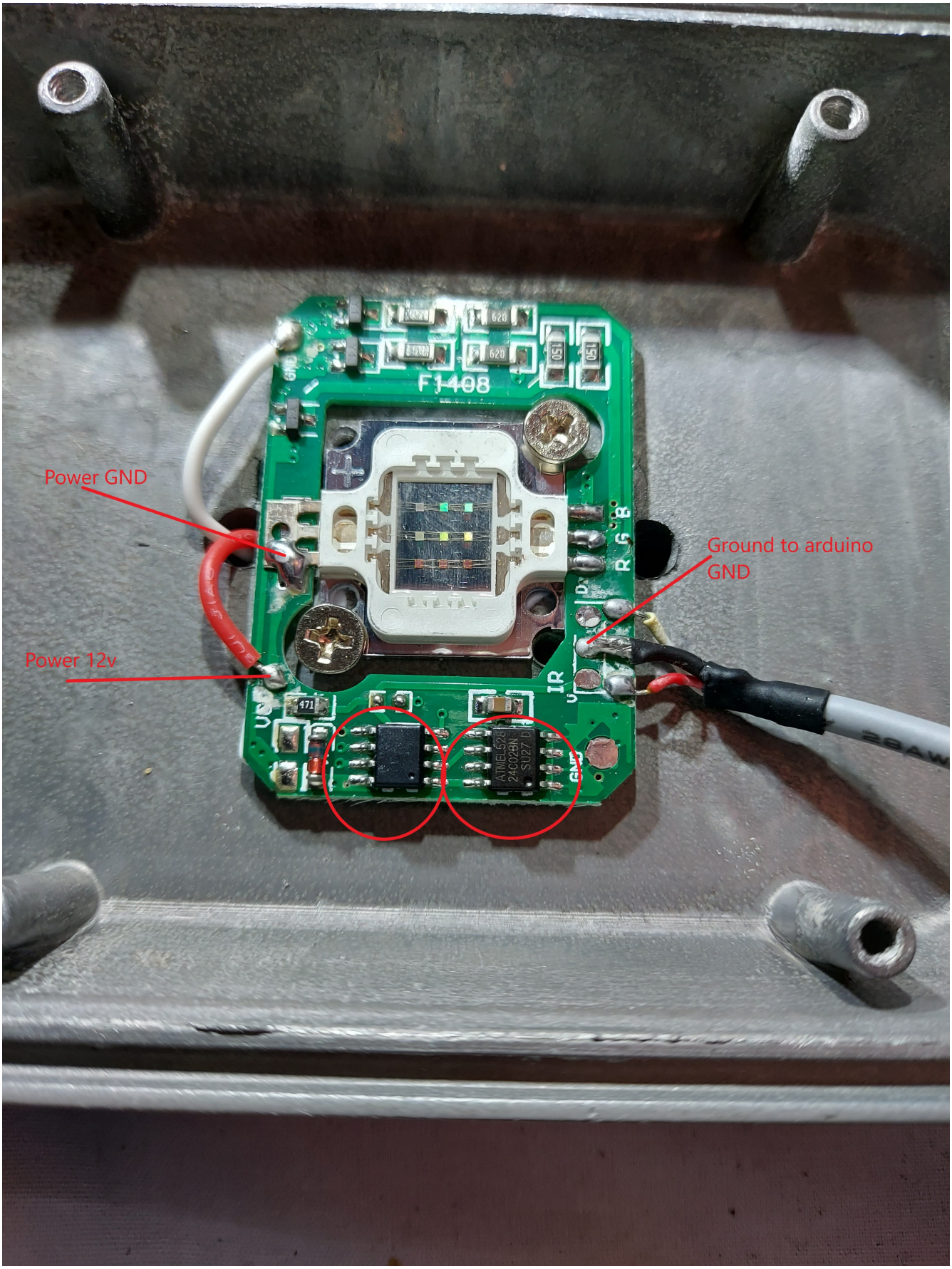


I don't think this particular brand is available any more, but there are are lot of copies – right down to the same case. They mount pretty easily in the front panel frame of the jukebox, and give me a lot of possibilities for light control. I would have been more happy with 4 or 8, but I only had 3 on hand. And we would have needed to extend the Arduino with more external circuitry if we wanted more than 4 since we only have 12ish PWM pins available on the Arduino mega and each light needs 3 pins.

When you open it, you'll see hopefully see a control board like this with 2 small 8 pin chips. One of them will likely be atmel xxx chip. The other may be unlabeled. They are circled in the photo below.

You will need to repeat these steps for each of the 3 lights.

Step 1: You'll need to remove those 2 chips that are shown circled in the photo below.

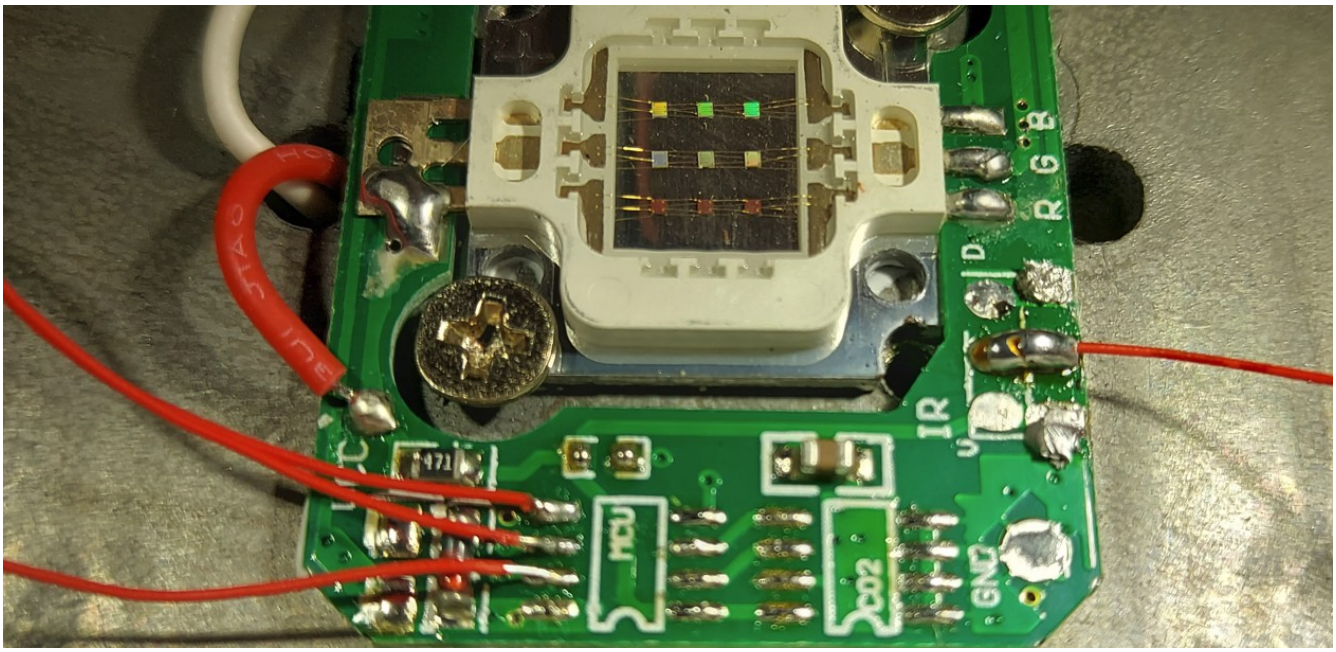

Power GND
Power 12v
Ground to arduino GND
F1408
R G B
IR
GND
ATMEL 528
24C02BN
SO27

Step 2: You'll need to decide if you want to keep the built in power supply on the lights. The back section of the lights have a mini power supply that takes 120vac and converts it down to 12vdc. You can keep the mini power supply in place if you want, but I opted to remove the built in power supply and power my lights directly from 12vdc. All 3 lights at full brightness will pull down almost 2amps at 12vdc, so you'll need a decent 12v power supply. I'm using a 3amp 12v one that I had lying around. Note that “power GND” labeled on the photo above does not need to get tied to ground anywhere else in the circuit.

Step 3: (optional) The 3 wires on the right go to an infra red sensor so that the light can receive signals from the ir controller. I choose to remove them. The ir controls aren't going to work anymore since we removed the control chips.

Step 4: The black wire on the right. You'll need a wire from that pin that goes to Arduino ground. (All 3 lights will have a ground wire that need to go to GND on the Arduino).

Step 5: Using the picture below as a guide, you'll need to solder on 3 wires as shown below. If your circuit board looks different, you can trace the wires back to the pins on the light marked R G B. Each one will go through a transistor (The picture is cropped and you can't see the transistors in the screenshot below). For this iteration of the circuit board, the top wire is red, the middle wire green, and the bottom wire blue. The connections are pretty small (I used wire wrap wire), so once you get the soldering in place, I recommend hot gluing the wires in place so they don't move.



Each of the 3 wires you just soldered will need to go to the Arduino. And each wire should have a 5k resistor inline. So Arduino → 5k resistor → red. Arduino → 5k resistor → blue. Arduino → 5k resistor → Green.

Wiring recap. Each light will have 4 wires that go to the Arduino: GND, Red, Green, and Blue. (And

possibly 2 power wires coming from each light that go a 12v power supply.)

Light 1
Red wire to Pin 2 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Green wire to Pin 3 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Blue wire to Pin 4 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Gnd to ground on the Arduino.

Light 2
Red wire to Pin 5 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Green wire to Pin 6 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Blue wire to Pin 7 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Gnd to ground on the Arduino.

Light 3
Red wire to Pin 8 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Green wire to Pin 9 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Blue wire to Pin 10 on the Arduino (with a 5k resistor in-between the Arduino and the light)
Gnd to ground on the Arduino.

This is what my light panel looks like (I haven't put the ties in place to hold the wiring yet. And you'll probably want to use some kind of molex connectors when wiring so that you can completely remove the light panel from the jukebox.

In the code, you'll need to uncomment this line if you want to enable the RGB lighting. Remove the 2 slashes at the beginning of the line, and you're good to go.

//#define RGBLIGHTS //comment out this line if you aren't going to use RGB lights in your front panel

Lights commands cheat sheet

//600 - lights off
//601 - orange solid on – (you can guess I have the orange paneled jukebox)
//602 - red solid on
//603 - green solid on
//604 - blue solid on
//605 - custom color solid on
//606 - red alternative blink - 2 second timer
//607 - blue alternative blink - 2 second timer

//608 - green alternative blink - 2 second timer
//609 - custom alternative blink with custom timer
//610 - sparkle red - 2 second timer
//611 - sparkle green - 2 second timer
//612 - sparkle blue - 2 second timer
//613 - sparkle custom with custom timer - will sparkle with multiple colors
//614 - sparkle2 - red green blue - 100ms timer
//615 - sparkle2 - light green, teal, skyblue - 100ms timer
//616 - sparkle2 - blue, white, aqua - 100ms timer
//617 - sparkle2 - red, white, blue - 100ms timer
//618 - sparkle2 - custom colors with custom timer
//619 - sparkle2 - random colors - 100ms timer
//620 - plasma - red green blue - 5 second timer
//621 - plasma - light green, teal, skyblue - 5 second timer
//622 - plasma - blue, white, aqua - 5 second timer
//623 - plasma - red, white, blue - 5 second timer
//624 - plasma - custom colors with custom timer
//625 - plasma - random colors - 5 second timer
//626 - cycle - red green blue - 4 second timer
//627 - cycle - light green, teal, skyblue - 4 second timer
//628 - cycle - blue, white, aqua - 4 second timer
//629 - cycle - red, white, blue - 4 second timer
//630 - cycle - custom colors with custom timer
//631 - cycle - random colors - 4 second timer
//632 - gradient - red green blue - 4 second timer
//633 - gradient - light green, teal, skyblue - 4 second timer
//634 - gradient - blue, white, aqua - 4 second timer
//635 - gradient - red, white, blue - 4 second timer
//636 - gradient - custom colors with custom timer
//637 - gradient - random colors - 4 second timer
//638 - fade colors - red green blue - 4 second timer
//639 - fade colors - light green, teal, skyblue - 4 second timer
//640 - fade colors - blue, white, aqua - 4 second timer
//641 - fade colors - red, white, blue - 4 second timer
//642 - fade colors - custom colors with custom timer
//643 - fade colors - random colors - 4 second timer
//644 - moving bars - red green blue - 4 second timer
//645 - moving bars - light green, teal, skyblue - 4 second timer
//646 - moving bars - blue, white, aqua - 4 second timer
//647 - moving bars - red, white, blue - 4 second timer
//648 - moving bars - custom colors with custom timer
//649 - moving bars - random colors - 4 second timer
//650 - glow - red - 7 second timer
//651 - glow - green - 7 second timer
//652 - glow - blue - 7 second timer
//653 - glow - red, custom color with custom timer
//654 - glow - random color - 7 second timer

//690 - color test - enter 000 000 000 to exit

//697 - input custom brightness values
//698 - input custom timer value
//699 - create custom pattern - 3 colors, timer and brightness

You will notice that there are a bunch of colors and patterns built in that you can access just by entering a 3 digit number. You may also notice that some of the patterns call for a customized color. You can enter a custom color and timer from the front panel keypad. (You can only have 1 customized pattern stored.) Colors are entered in R G B format. First, you enter a number from 0 – 255 for the red value. Next, 0-255 for the green value. And finally 0 – 255 for the blue value. Enter all number as 3 digits. If you want to enter the number 57, you would enter 057 on the keypad. (Some of the patterns only use 1 color, even if you have 3 colors entered via command 699.)

For the patterns that take a random color - The random colors are chosen when you enter the command number. Sometimes you get nice colors. Sometimes they are bad. Just enter the command again and you'll get new random colors.

The built in colors above really aren't really optimized. Feel free to change them in the code once you find values that you like.

690 – Color test – This is an easy way to test RGB values. You can't enter this mode if a record is playing, but it will work if an mp3 is playing, or you are in phono mode with no record playing. After entering 690 on the keypad, you will see c1r (color 1 red) displayed in the panel for 1 second. The display then goes blank and you can enter your first 3 digit number for the red value. Next you will see c1g (color 1 green) – enter the green value. And finally c1b (color 1 blue) – enter the blue value. Similar to entering a song number, if you screw up, like entering a 6 for the first digit, just press reset, and it will clear out the digit. If you enter a number that is larger than 255, it won't accept it, and you will need to enter that color again.

After entering all 3 numbers, the front panel will now change to a solid color representing the RGB value you entered. The orange or blue outer shell will modify the colors a bit, so you'll need a bit of exploration to find suitable colors you like.

You will stay in mode 690 and you can keep entering R G B values for testing until you enter 000 000 000. Once you enter 0 for all values, you'll return to normal jukebox mode.

697 – Normally, we are at full brightness on the RGB leds, but we can customize a bit and cut down the brightness to create mellow colors. When you enter this mode, you will see br (brightness red) on the display for 1 second. And then the display will go blank. Enter the maximum red brightness value (0 – 255). Once again, use 3 digits. 042 if you want the maximum red brightness to be 42. After entering red, follow the same procedure for green (bg on the display) and blue (bb on the display). Brightness is a global setting, and applied to ALL patterns. Brightness can be saved with command 998 and will automatically be applied upon startup.

698 – Most of the routines use a built in timer to control how fast or slow a light fades in and out or flashes. For some patterns, you can set the timer from the front panel keypad. After entering 698, you will see tmr (timer) on the led display for 1 second. Enter a 5 digit value that represents the number of milliseconds on the timer. If you want a 100ms timer, enter 00100. If you want a 5 second timer, enter 05000. The custom timer value can be saved with command 998 and will automatically be applied

upon startup if you are using a pattern that takes custom timing.

699 – You can't enter this mode if a record is playing.

This command allows you to enter a full pattern: 3 colors, a timer, and brightness. The “custom” routines listed above will use these values (as well as the values from command 697 and 698). After you enter this routine, you will see c1r on the led display for 1 second. Enter the red value for color 1. Be sure to use 3 digit format for each number. Next, c1g. Enter the green value for color 1. Next, c1b. Enter the blue value for color 1.

Follow the same pattern for c2r, c2g, c2b. And then c3r, c3g, c3b. 3 colors total.

Next, you'll be presented with br, bg, bb. Enter the maximum brightness for red, green, and blue. Once again, 3 digit format for each number.

Next, you'll be presented with tmr. Enter a 5 digit value. (see command 698 for more info)

Once you have this information entered, you will go back to normal jukebox mode. Be sure to enter 998 if you want to save this data to the eeprom so that it can be used on startup. If you have a custom pattern playing, it is automatically updated to use the new colors and timing.

Code

You can pull down the code from github:https://github.com/rjdinap/ArduinoJukeboxR81

Take the Jukebox_R81.ino, LightsEffects and RecordPlayer.ino files and put them in 1 directory for the Arduino IDE.

Take the ALA-FIXED directory, and copy it to the Arduino libraries folder. Then, start (or restart if you already had it loaded) the Arduino IDE. Go to Sketch → Include library, and choose the ALA-FIXED library that we just copied.

This is my first Arduino project. The code may not be the prettiest, and it does use mostly global variables.

By default, songs are played in first in, first out fashion, both on the mp3 player and phono. For phono mode only, you can enable R81 style queuing by setting the following variable to 1. (This is a compile time option.). With the number set to 1, when playing records, if you have multiple songs queued up to play, the next song to play will be the **closest** song to the current magazine position.

```
int r81Queuing = 0; // set to 0 to play songs in a first in, first out fashion. set to 1 to emulate r81 queuing mode – it scans the current song queue, and the next song to play will by the song that is closest to the current physical magazine position - only active in phono mode
```

Right now, there is only a main branch in Github – no versioning.

History

So, I was looking for a new item to tinker with. I found a guy who had 13 different jukeboxes for sale, all 'fixer uppers'. “I want a jukebox”, I thought. I further mused, “It would look cool in my basement that I'm never going to have time to finish because I keep buying projects.” I looked inside - it looked mostly complete, so I picked up an Ami Rowe R-81 from 1977 for $100 and hauled it 4 hours back to my house. In retrospect, it was $100 too much. First time I've ever seen a jukebox, so I'm starting at beginner level on this. I'm pretty sure this guy knew exactly what he was selling me - pretty much everything in the box was bad. No needle. Cartridge wires disconnected. Keys gummed up and unresponsive. Turntable not turning. Encoder mechanism off and the encoder fingers bent. The magazine motor wouldn't turn freely. Speakers needed to be reformed. No coin mech. Pre amp bad, and the amp was hacked where it had no control knobs. And at least one of the control boards was bad.

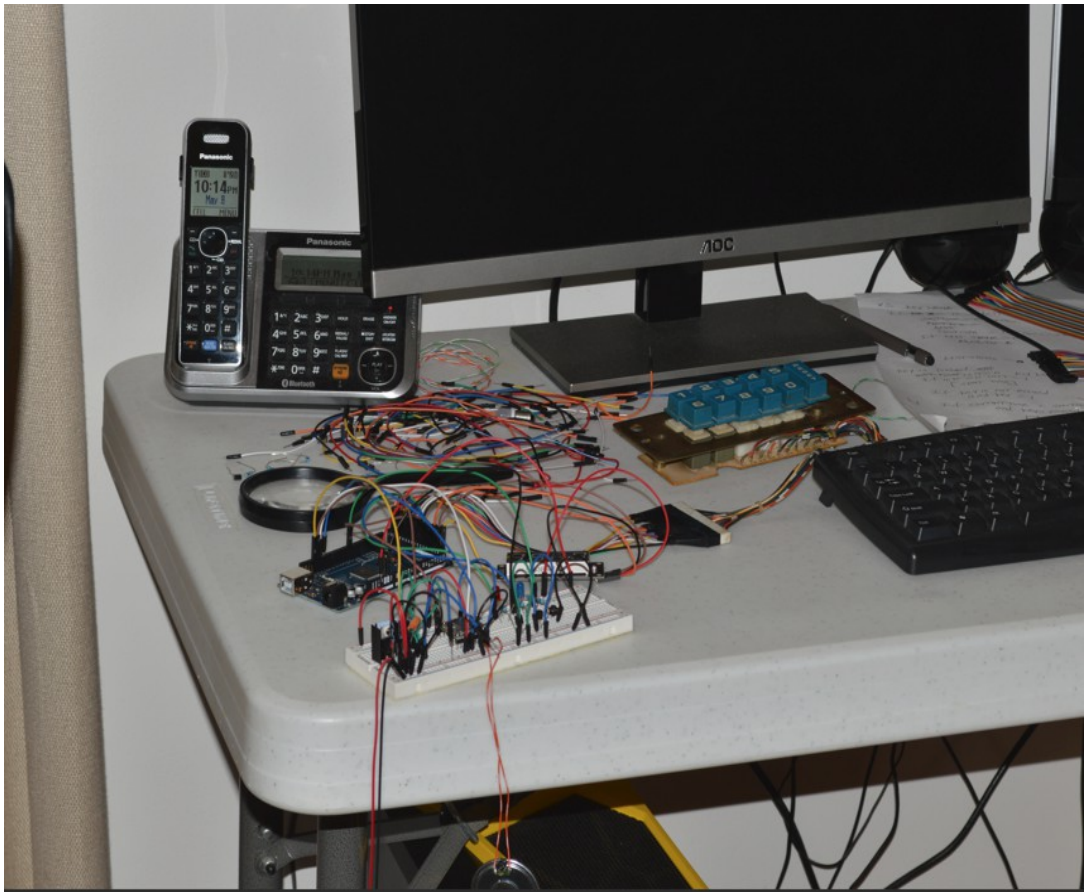
With time, I got most of these things working. Except for the amp. I don't have the knowledge to repair it (yet). And at least one of the control boards was bad. I think I narrowed it down to the mech unit board being bad. I wasn't ready to pay for a new amp or amplifier repair. Or pay for a new mech board and hope that's the only board that was bad. So, this project morphed into “Maybe I'll just stick an mp3 player in there”. And then it morphed into “Maybe I'll try to control the phono mechanism with an Arduino”. And so here we are.

I get it - If you're a purist, this isn't for you. I own a few arcade games, and the thought of sticking an lcd monitor in there doesn't sit well. But for the amount of things wrong with this box, it was going to wind up as a donor for parts, or in a landfill. For this project, my goal was to get this working as cheaply as possible using parts I (mostly) had on hand. Right now, this is a prototype. At some point, I'll redo the wiring and make it look prettier and put the circuit boards in a case.

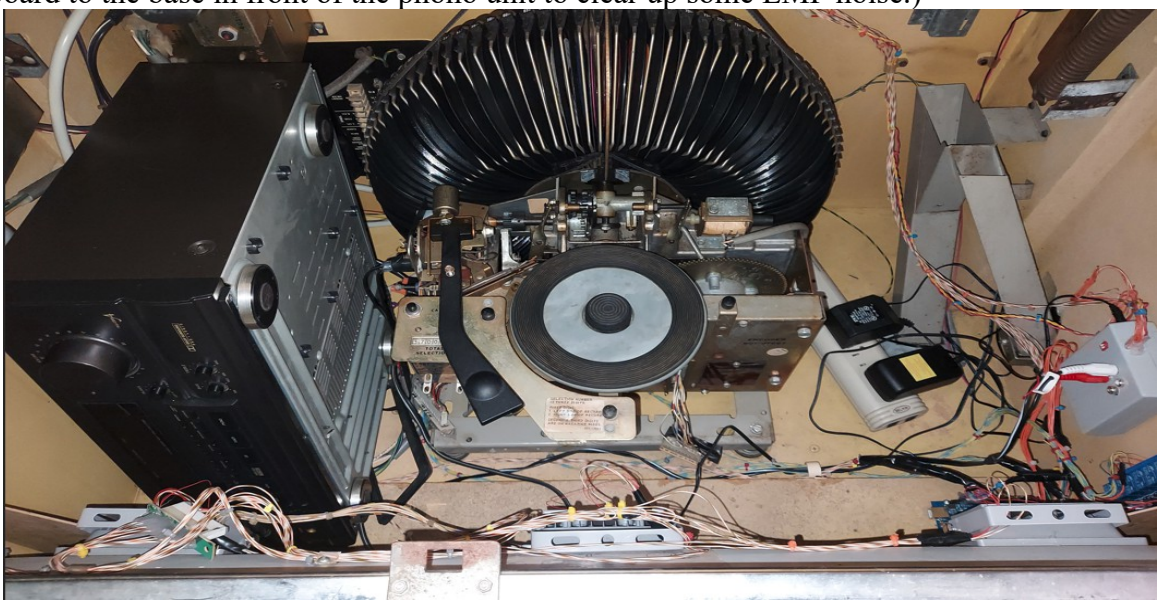
This is my first Arduino project. Actually, my first microcontroller project (except for a pic12c509 autostop chip for pachislo machines about 20 years ago.) It's my first time wiring a breadboard. And the first time I've seen the inside of a jukebox. Lots of firsts here for me!

You'll notice my pictures use a lot of hot glue to hold things in place. I've been down the road more than once where I spend money as I go to make something look great and get stopped by something I can't fix. Now that it's working, at some point I could see myself redoing it with proper cabling. But... I have other projects I want to get to. And this turntable has 370k plays on it. I'm waiting to find a cheap or free donor jukebox within travel distance so I can get a new turntable and encoder. For this jukebox at this time, the redo will have to wait.

Here's a picture of my initial testing of the voltage regulator, mp3 player and credit light control wired up to the keypad and led display. Enjoy, and happy hacking!



Almost finished! I need to move the amp (and replace it at some point with something smaller), and cleanup the wiring a bit. But fully functional at this point. (This is an old picture. I had to move the relay board to the base in front of the phono unit to clear up some EMF noise.)



Appendix 1– mp3 file renamer

https://github.com/rjdinap/dfmini-mp3-file-renamer

The repository above contains a quick and dirty vb.net (vs 2013) program that will automatically sort and renumber all .mp3 files in a given directory. It beats renumbering hundreds of files by hand!
The repository contains a pre compiled .exe program in the /bin/debug folder if you don't have Visual Studio.

Appendix 2 – EMF

Unbelievable. (I know you see what I did there). I thought I was done with all the "build" work and just needed (and wanted) to focus on the software. But I noticed that every once in a while, my Arduino would reset. If you have the computer connected and you are debugging, you may see text like this:

```
09:48:48.211 -> Force record return activated.
09:48:48.211 -> Returning record. Transfer motor on.
09:48:53.361 -> Record return complete.
09:48:54.838 -> Record playing light turned off
09:48:54.838 -> Song pulled from queue - track: 44  folder: 7  items in queue: 0
09:48:54.871 -> Record playing light turned on
09:48:54.871 -> Phono play song command issued
09:48:54.871 -> Jukebox song ▯umber to play: 43
09:48:55.268 -> Pre-scan for record complete. Carousel position: 42
09:48:55.300 -> No need for scan. Carousel position: 43
09:48:55.300 -> TransferRecord Stage 1 - setting up timer and turni▯▯▯on transfer motor
09:48:55.338 -> Turning transfer motor on.
09:49:00.325 -> Record on turntable. Turning transfer motor off.
```

That's EMF noise, and there is a lot of it going on while the motors are active.

I've originally tried:

- Opto isolation on the relay module – with it's own power supply – no change
- Moving the relay module and wiring away from the Arduino – This seemed to solve some of the noise problem. (Also stating here that I've used the cheapest wire that I had on had – in most cases bell telephone wire. So it doesn't have any form of emf protection)

Solution: I was researching snubbers, flyback diodes and emf wire wrapping solutions when I

accidentally seemed to solve the issue: Originally, I moved the relay module as far away from the Arduino as possible. But it was still sitting on the same wall as the Arduino and mp3 player. Moving the relay board to the bottom of the cabinet (in front of the phono mechanism) has been rock solid. (Why? I don't know. It's a lot closer to the Arduino now than it was before.) I'm about 1000 plays in, and ZERO emf noise picked up in debugging. And zero resets!

Changelist

- 1.01– fix encoder pin numbers in PDF, show rst(reset) and mp3/pho on startup. Add some documentation to PDF.
- 1.02 - add 999 command to reset. Add documentation on dfmini mp3 file renamer. Fix a bug I introduced in the ALA library for maximum brightness. Add 753 command for multi folder random mp3 player. Brightness is now applied to all rgb lights patterns.
- 1.03 – Added documentation and code to support the play counter hardware
- 1.04 – Record transfer timer added. Code cleanup
- 1.05 – Typo fix
- 1.06 – Switch to digitalWriteFast. Code cleanup
- 1.07 – Add digitalWriteFast to the encoder read routine
- 1.08 – Fix first time startup – no data in eeprom. If in phono mode, display encoder position on startup in the led display.
- 1.09 – change queuing libraries and add R81 queue emulation mode as an option
- 1.10 – Add AlmostRandom library. I wasn't happy with the randomness I was getting using noise from the analog pins. Fix another first time startup issue reading from the eeprom.
- 1.11 – fix in the record scanning routine. Clear the song queue on a critical error. Add error code SC2
- 1.12 Fix randomization issue. Increase time between playing mp3 songs to 3.5 seconds.

TODO

- Add a schematic drawing of the whole system
- More robust real-time checking of whether a scan or transfer is in place instead of relying upon a flag

Like it? Hate it? Built it? Drop me a line with comments or suggestions: rjgee@hotmail.com