MaxDuino

Alan Lomax M8640

Alan Lomax was born in Widnes, Cheshire and moved to Canada at the age of 7 with his family. His early years involved exploring techniques of percussive disassembly (aka hitting things with hammers). His formal education followed a technical path with a bachelor's degree in Electrical Engineering and a master's degree in Management Science both from the University of Waterloo. His working life started off with sensors, control valves, and low voltage electrical power circuits. His middle career work centred around chemical plant automation and control systems and this took him and his family on assignments around the world. This culminated in Alan being the lead technical manager for a large control system replacement project. In retirement Alan has rediscovered his passion for model railways and how it integrates many of the things he dealt with while working. Within MERG Alan is a contributing member to the Arduino SIG and the RFID SIG. Alan and his wife Dawn are happily retired and live in Sarnia, Ontario, Canada.

Abstract

This article describes an Arduino based printed circuit board for use on a variety of layouts. The circuit design is flexible enough for use in many applications such as animations, signals, lighting, sound effects switching and many more. It is also very useful as a software development platform enabling many different features to be explored with minimal fuss.

Background

My railway club is embarking on a new club layout which is to be modular in nature. This represented a perfect opportunity to press 'reset' and remedy many of the identified shortcomings existing on that which came before. One of the identified shortcomings was how several of my Arduino implementations were cobbled together using a lot of point-to-point wiring and add on modules for things like power conditioning, sensor inputs, sound, RS485 communications, and NeoPixel implementation. With his in mind, it was resolved to design a custom 'carrier' PCB to implement all of these features in a compact design I have now called the MaxDuino. The heart of the design is a basic standard Arduino NANO and the circuit design allows the various subsystems to be included or not according to the desired features.

The Design Process

My design process started with a block diagram (Fig. 1) indicating the desired features and what I/O pins I planned to use for that feature. From this a schematic was created using KiCad design software[1] . I went through a couple of design iterations making incremental improvements along the way. Most of the improvements were optimizations of PCB real estate, such as selecting smaller footprints for capacitors which in turn allowed extra features to be added like dedicated 5-volt screw terminals and the NeoPixel were late additions. While the PCB's were being manufactured at JLCPCB[2] I developed the Arduino software test suite of sketches. These were then used to validate the operation of various features as the PCB was built up. The current version (v1.1) has had all features tested and it has functionally 'passed' the full test suite.

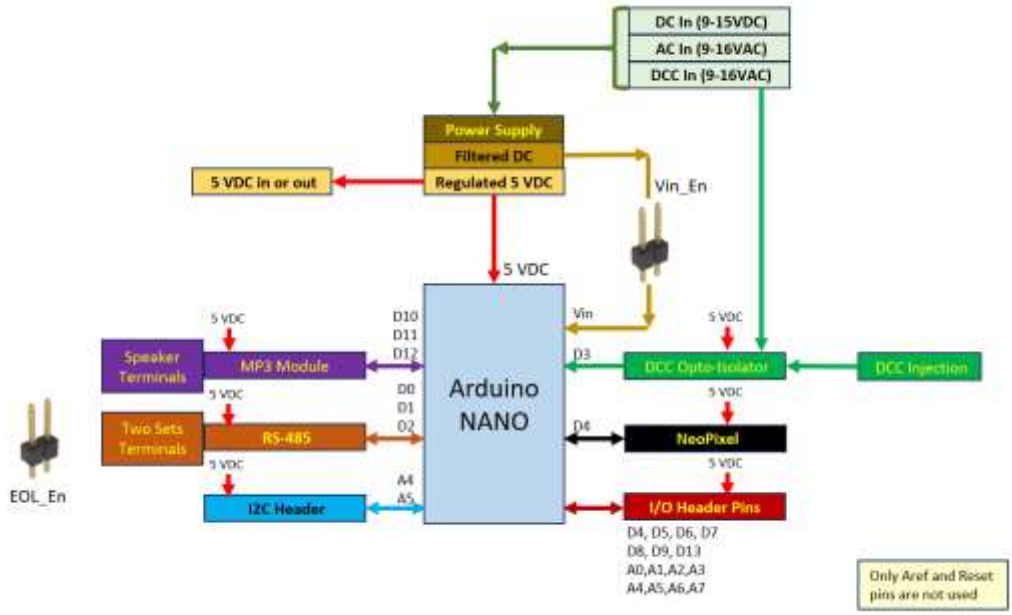

Fig. 1 – MaxDuino Block Diagram

The 3D view of the board is shown in Fig. 2 below. Onto the 3D view I have added callouts showing the major functional areas that generally match the block diagram shown previously. As is self-evident there is not a lot of free space available on this board but in the little that was left over I dropped in a small white square of silkscreen which with the use of a sharpie marker is handy for keeping track of specific boards which may have specific software loaded.

The test software and how it was iterative developed eventually led me to include a standard function in each of my test sketches to output a software 'fingerprint' to the serial monitor at bootup. As long as I remember I used 19200 as a standard speed it will on power up send out some pertinent information, chiefly what source code file was used, what version I have assigned to it, and when it was actually compiled. Additional serial outputs are of course possible and will vary by sketch and application.

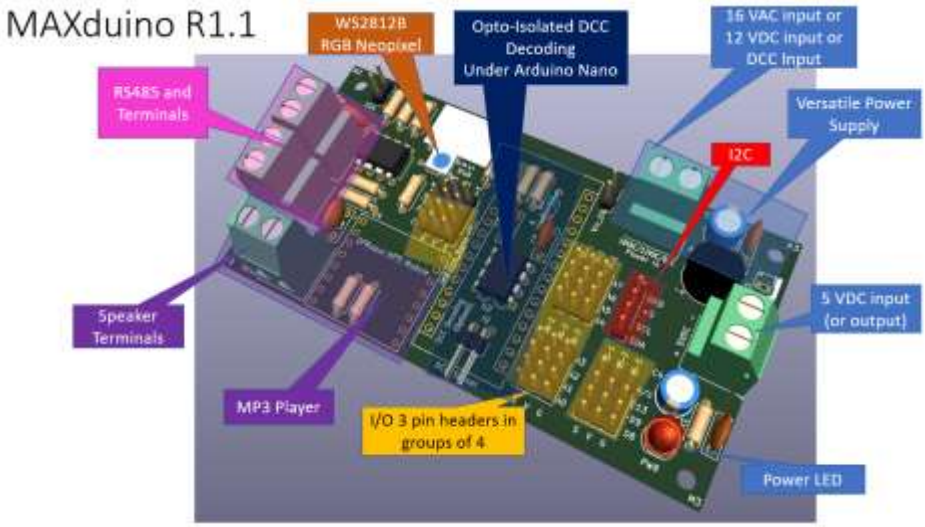

Fig. 2 – MaxDuino 3D View

Key Features List

The following sections describe the main functional areas of the board and what they contribute to the overall operation.

On any given build of the PCB not all sections need be present thus giving considerable flexibility depending on the intended application. If the full PCB is built up it represents a standard building block of functionality and can provide many Arduino features which can be implemented over time consistent with a phased modular club layout approach.

Power Input

From the outset it was my self-imposed choice to make it a design requirement to have a very flexible power supply system. The root of this was my own home layout has a 12 VDC auxiliary bus available but there are other club members that only have a 5VDC buss running on their layout. The club layout was tending toward a 12 auxiliary power system but this was not cast in stone at the time. Not wanting to paint myself into a corner the power supply needed to be flexible.

My flexible input power supply section schematic is shown in Fig. 3 below and was based on several reference circuit designs for the L7805 chip. One of the better references on the L7805 chip itself can be found here.[3]

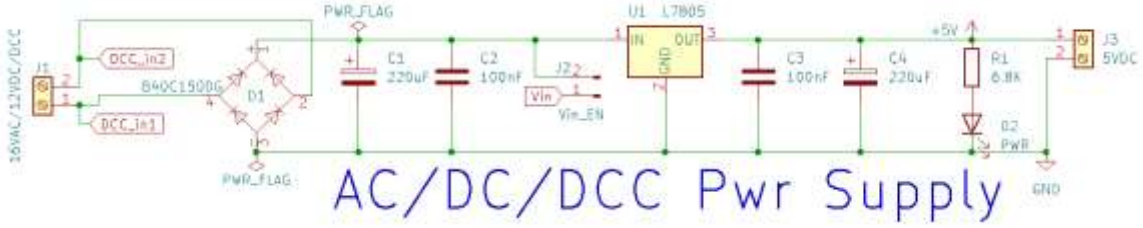

Fig. 3 – Power Supply Schematic

Power comes in from the left and can be any of 16VAC, 12VDC, or even a DCC feed. This goes through the full wave bridge rectifier D1 and smoothing capacitor C1 and filter capacitor C2. This filtered DC voltage is the input to the 5-volt regulator chip L7805 which is capable of providing 1.5 amps and is also internally protected against short circuits. A pair of pins is provided as J2 which if a jumper were installed there would route this power to the Vin pin on the Arduino. The use of a TO-220 heat sink is recommended, especially at higher input voltages like when using DCC for a power source. The 5-volt regulated output is filtered with C3 and C4 to improve transient response and this produces a rock steady 5-volt supply and is suitable for significant Arduino projects.

On the right-hand side I have provided a second set of screw terminals which can be used as a 5-volt output (to supply other circuits), or optionally this can be used as a 5 volt input (in which case the L7805 regulator and everything to the left of it is not required.

Finally, a simple LED with a relatively high 6.8K current limiting resistor is used to indicate the presence of 5 volts. This LED works even without an Arduino present and it does not care where the 5 volts is coming from.

Power Supply Notes:

This circuit utilizes what might be considered a robust amount of capacitance, however as the power leads could be relatively long on the club layout, and the operating environment is uncertain, I do anticipate them having significant electrical noise. Similarly, the 5-volt output could potentially also be supplying (or at the receiving end) of a long run. The capacitors can of course be left out or other values used however if no space were provided on the PCB at all then they would be difficult to retrofit after the fact.

Care must be observed when J2 is used to ensure Vin is 12VDC or less. The Arduino's on-board regulator is not rated beyond this and so could be damaged.

On the left-hand side the flags are indicating that the power inputs are also routed over to the DCC optoisolator circuit (more on DCC Decoding below).

Arduino

The Arduino portion of the schematic (Fig. 4) is not particularly remarkable other than showing that the majority of pins are used. Indeed, the only pins not used are the 3v3 output, the analog reference Aref, and the two Reset pins.

Physically during PCB assembly, the Nano is mounted using female header pins to standoff from the MaxDuino PCB as the space under the Arduino is used for the DCC decoding circuitry.

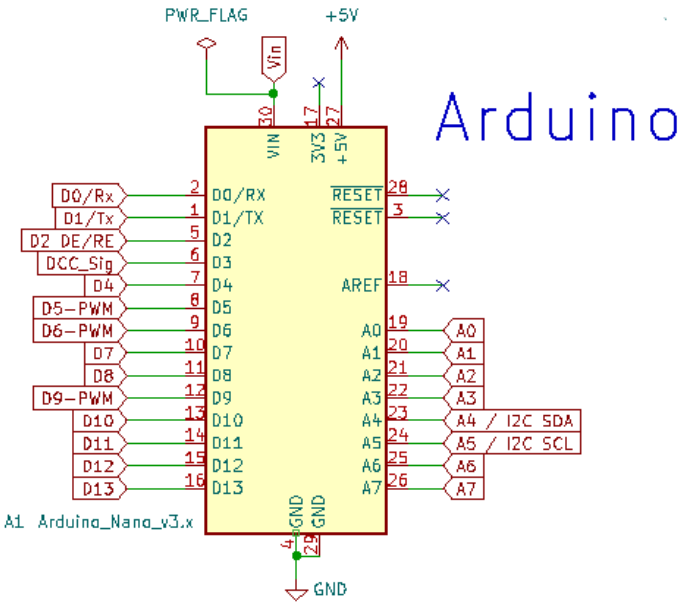

Fig. 4 – Arduino Pin Usage

DCC Decoding

For my own layout I wanted the option to implement an Arduino based stationary decoder. This turned out to be one of the more challenging aspects with the main factor being some poor component choices on the initial circuit I found on the internet. [4] I eventually found a good in-depth analysis [5] of that original circuit which suggested some significantly different component values in the same fundamental circuit design. I experimented with these using a solderless breadboard before settling on the values now shown in Fig.5 below. Referring to the schematic the DCC signal comes from the power circuit on the left as DCC_in1 and DCC_in2. These signals are routed through two marked isolation points (Implemented as thin traces on the underside of the PCB that can be scratched open if needed). The DCC signal then goes through the 2K2 current limiting resistor R2 and then through the optoisolator U2. D3 provides reverse polarity protection for the optoisolator diode. C5 (in conjunction with R2) provide transient noise rejection. The output of the optoisolator goes to Arduino pin D3 and is pulled high by a relatively low value R4 which is specifically selected to significantly improve the risetime of the outgoing square wave. (This is quite different than the normal use of pullup resistors). Pins 1,4 and 7 are not used. The original source showed a pull up resistor on pin 7 but this was not actually required as the optoisolator is enabled by default due to internal pullups.

During board assembly these components should be amongst the first to be installed if the DCC decoding functionality is desired. This is purely for access reasons as after the Arduino header pins are installed they will restrict PCB access in this area.

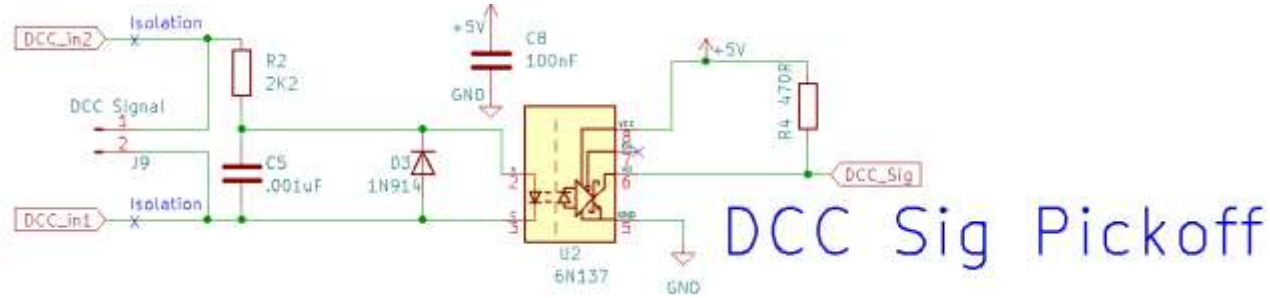

Fig. 5 – DCC Decoding

DCC Decoder NOTES:

As mentioned, a number of issues were encountered when implementing this portion of the schematic. What I first observed was that nothing was being decoded although the sketch was running fine. Reviewing the code I saw that decoding zeroes and ones depended upon some precise timing triggered by interrupts. It turned out that as the DCC signal is being produced by an H bridge type circuit their occurs at the zero crossover a point where two MOSFETs will stop conducting just as two others start. If these components are mismatched in any way a transient signal (glitch) can occurs at each and every zero crossing in the DCC waveform. Unless filtered out these transient glitches can be significant enough to trigger the opto-isolator which causes a software interrupt on the Arduino pin. Since bit decoding depends on the length of the pulse it follows that resetting the timing on every ZERO crossing results in nothing getting properly decoded. The nefarious aspect of this is that the magnitude of the glitch depends on the magnitude of the DCC voltage. For the original author it might not have been an issue but for me (and obviously others) using 14VDC for DCC did cause issues. Some motor shields are likely better than others with regards to the zero-crossing glitch phenomenon and this is also a consideration.

If DCC is not used as the MAXduino power supply the two power inputs DCC_In1 and DCC_In2 will of course have full input power on them whatever the source is. This will not normally be an issue because of the high value of R2 but of course no DCC decoding can happen unless some changes are made.

To decode DCC where DCC is not the input power follow these steps:

First the two DCC isolation traces must be made open circuit. These traces are clearly marked on the bottom of the MAXduino PCB (X marks the spot). Opening these two traces where indicated disconnects the 'power supply' from the 'DCC decoder'. Once these traces are opened then the PCB power source is completely independent of DCC signal decoding. A DCC signal can now be input via J9 which is implemented as a pair of 90 degree header pins located on the PCB under the Arduino.

MP3 Player

The DFRobot MP3 player is a powerful and inexpensive module that can read and play MP3 files from an SD card. It has a low-level audio signal outputs and also an 8-ohm speaker output driven by a low power audio amplifier.

Commands are sent to the module, and results fed back from this module using standard serial messaging (Transmit = Tx and Receive – Rx). It is only necessary to connect these to the Arduino and send out appropriate commands like "Play Track 1".

This module operates at 5 volts but is noted for clicking whenever commands are sent at this voltage. I have implemented a simple voltage divider to drop the Arduinos sent commands down to about 3 volts to solve this.

I read on the manufacturers page [6] that a simple 1K resistor in series does the same thing but I had trouble coming to grips with how this worked electrically so I stuck with what I knew.

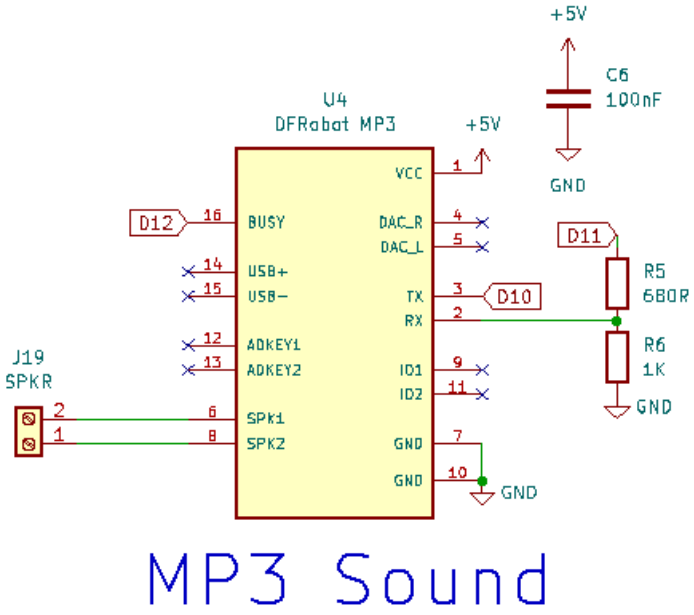

Fig. 6 – MP3 Player

I did have some issues initially with this but it turned out there are two different parts for very similar MP3 modules and I had them jumbled up in my parts box. The now recommended one is branded as a **DFRobot MP3 player** whereas the one that gave me issues was branded as **mp3-tf-16p**. The symptoms manifested as an inability to change the volume (it started loud and stayed that way). Other commands like changing tracks or play/pause all worked fine. I now believe the issue might be attributed to a library incompatibility but once I found a working combination, I purged the offending parts and stuck with what worked.

NeoPixel

This is a very simple circuit and uses a single bare NeoPixel chip generically called a WS2812B. In the MaxDuino implementation I use the physical space of one LED but now have the opportunity to display many colour coded feedback signals such as green for running, red for error or similar. There are thousands of colour combinations possible [7]

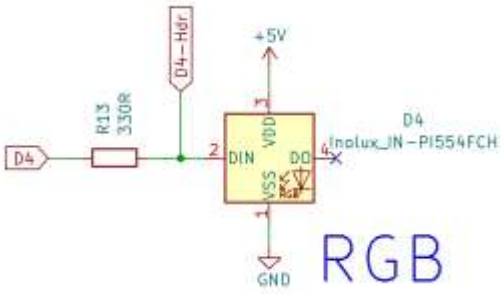

Fig. 7 – NeoPixel

For those not familiar with the chip it is about 5mm to a side and consists of 3 LEDs (Red Green and Blue) plus a "single bit processor" all in the one tiny surface-mount package as shown in Fig. 7b.

Addressed commands consisting of the desired R,G,B values on a scale of 0-255 are sent to the chip via the Din pin. If the address is 'zero' this chip acts on the command, otherwise it decrements the address and repeats the command out the Data Output pin presumably to the next chip in the chain.

You can see how when arranged as a strip these are commonly referred to as addressable LED strips.

Fig. 7b – The NeoPixel WS2812B chip

RS-485

It is frequently the case where it is desired for an Arduino to operate in conjunction with some external controller, or in coordination with other Arduinos. This implies some sort of communication mechanism which in this case is RS-485. This standard is more of an electrical standard than a full-blown protocol but it does have some compelling features namely it is robust and it is inexpensive. The downside of it is it is relatively slow and lacks error checking. I have used it previously in industrial settings covering many hundreds of meters and I can say it works as advertised in spite of the harsh environment and less than ideal wiring practices.

The network consists of two wires that daisy chain from device to device. At each end of the network a 120 ohm 'end of line' resistor is used to prevent voltage reflections causing issues.

In the MaxDuino implementation I used the same circuit found on the inexpensive external modules available from China, but ensured the EOL resistor was only enabled if a jumper was installed at the JR7 location.

Fig. 8b – RS-485 Module

For those who use JMRI there is within it the ability to define an interface called the Chubb Computer/Model Railroad Interface better known as CMRI. Originally CMRI nodes were hardware I/O boards for the JMRI software to talk to but now CMRI more commonly refers to software emulations of that original hardware. Most relevant for MaxDuino is that there is an Arduino library that allows the Arduino to emulate one or more of these CMRI nodes. The bottom line here is that JMRI can send on/off type commands to the CMRI emulator and the Arduino can act on them.

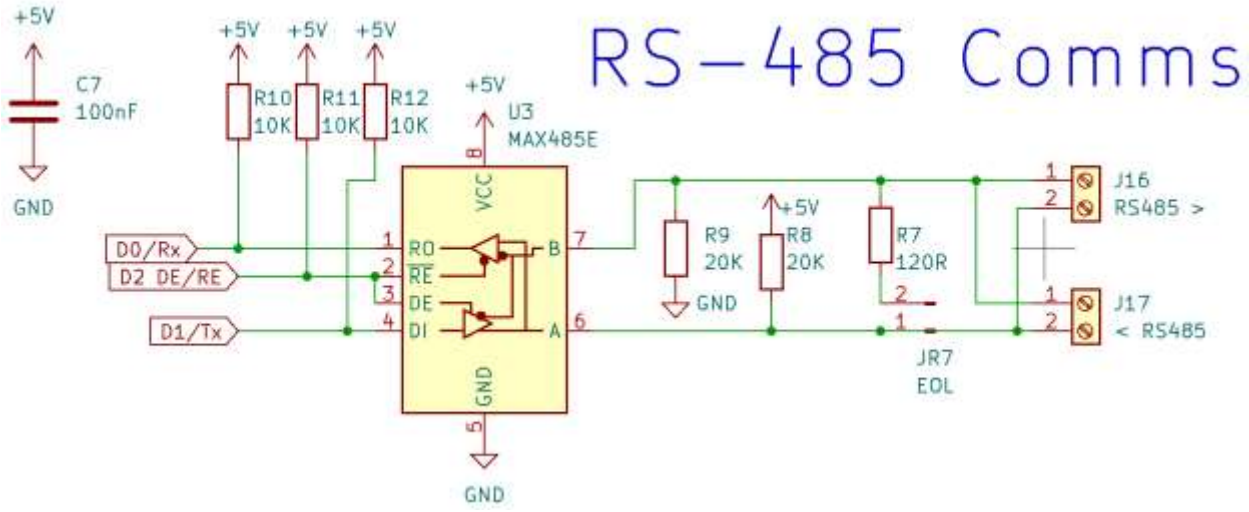

Fig. 8 – RS-485

I/O Headers including I2C

Nothing particularly exciting here other than a brief description of an improvement I made during the development process. In the Original Rev 0 MaxDuino I had all my 3 pin headers laid out as horizontal strips of 3. During a show and tell session a fellow club member asked if I could colour code the pins which turned out to be more work than anticipated. Fortunately, I was overhauling the PCB layout anyway and now the pin headers are organized to allow vertical strips of 4 and these which can be colour coded at build time, for example red headers for +5V, black for Ground, and Yellow for Signal. Definitely an improvement and I will carry this forward in future designs, but it did get messy for a while.

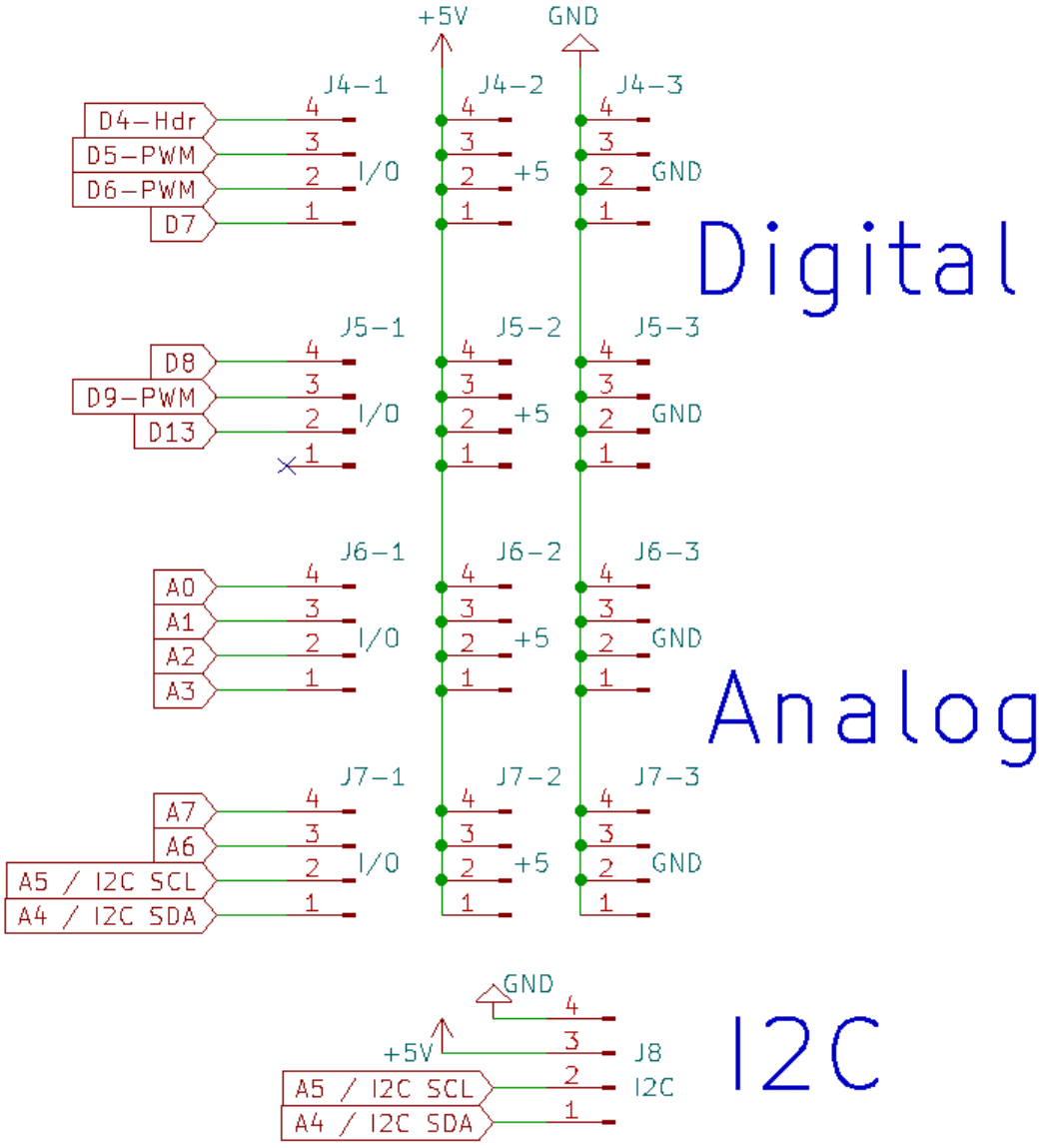

Fig. 9 – Headers

Full Schematic

Figure 10 shows the complete schematic as produced in KiCad. Each sub section has been discussed in detail previously.

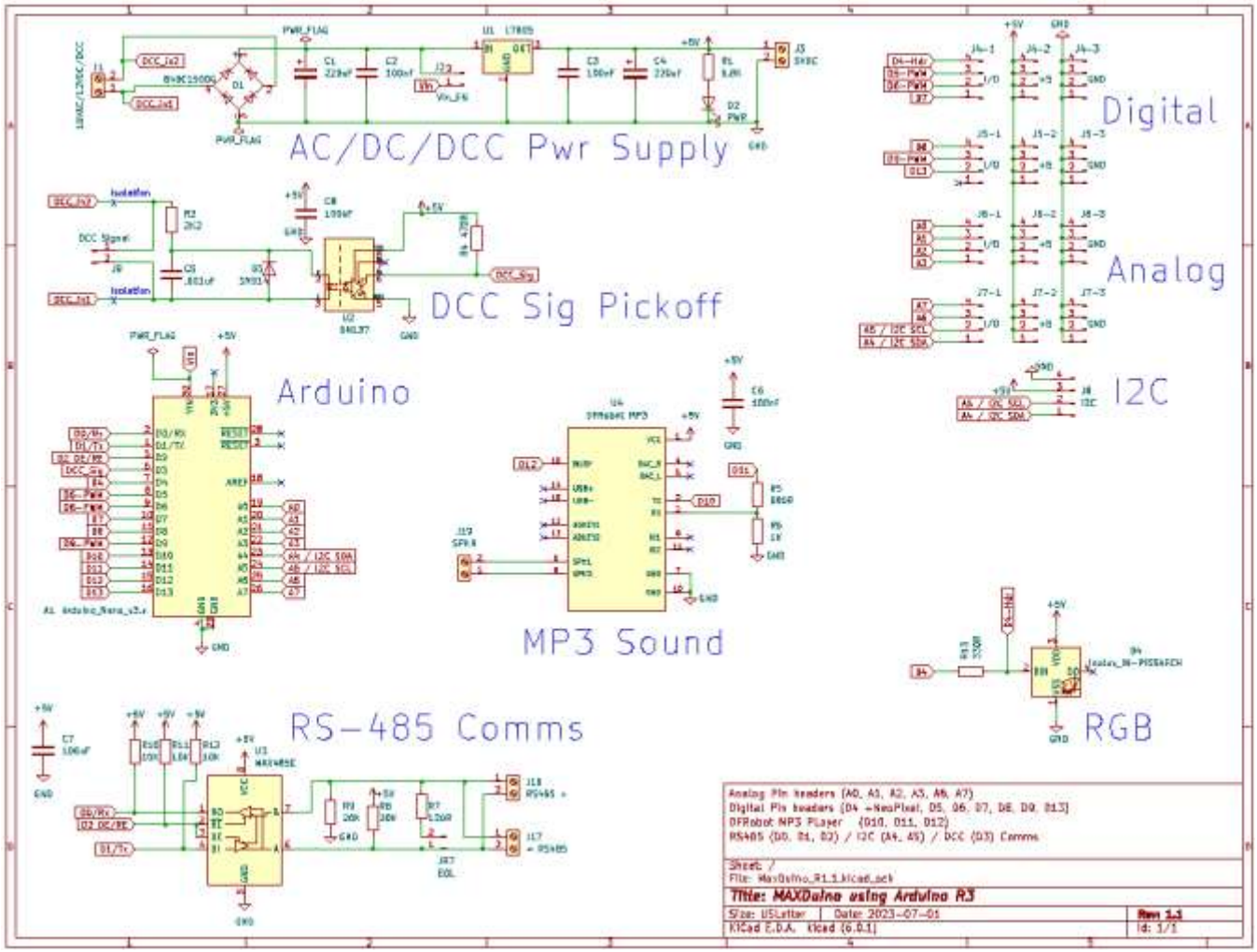

Fig. 10 – Full Schematic

Gerbers, Full Build Instructions and Test Sketches

(Make Available on GitHub)

1. Circuit Geeks article on NeoPixels, https://www.circuitgeeks.com/ws2812b-addressable-rgb-led-strip-with-arduino/

References

1. KiCad PCB design software, kicad.org/
2. My PCB Manufacturer JLCPCB, jlcpcb.com/
3. About the 7805 Regulator, https://www.electronicsforu.com/technology-trends/learn-electronics/7805-ic-voltage-regulator
4. Rudy's Model Railway DCC Decoder https://rudysmodelrailway.wordpress.com/software/
5. In depth analysis of DCC Decoding circuit, https://wakwak2popo.wordpress.com/2020/12/11/dcc-sniffer/
6. DFRobot MP3 player module: https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299
7. RGB Colour Picker: https://www.rapidtables.com/web/color/RGB_Color.html