A Complete Arduino DCC Controller

Digital Command Control (DCC) is a great way to control multiple trains on a model railway layout. Unfortunately, commercial DCC systems can be quite expensive. Here we present an Arduino-compatible Controller shield that can form the basis of a DCC system. It can also be used as a DCC booster or even as a high-current DC motor driver.

by Tim Blythman

You can put together this DCC controller, which incorporates a base station and optionally also a programmer, for a fraction of the price of a commercial unit.

Combine it with a PC, and you have a potent and flexible model railway control system.

It's based on the Arduino platform, and it's easy to build. You can also add boosters to the system easily, just by building a few more shield boards.

DCC is still the 'state-of-the-art' in terms of off-the-shelf model railway systems, so if you have a model railway layout but don't have a DCC system (or have a DCC system that's inadequate for your needs), now is the time to upgrade!

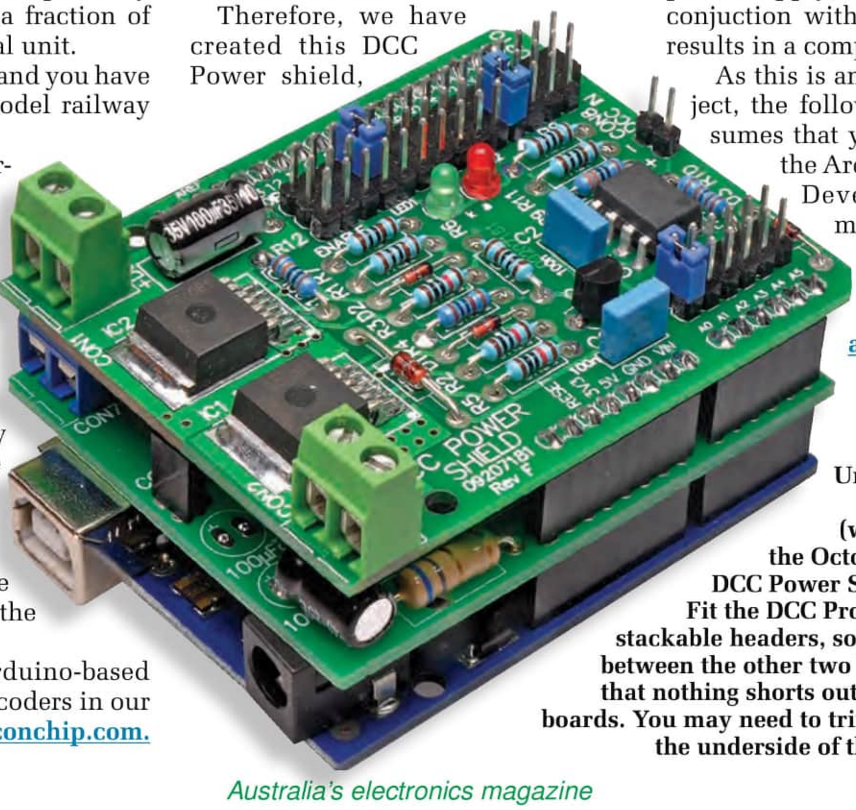
We published an Arduino-based DCC Programmer for Decoders in our October 2018 issue (siliconchip.com.au/Article/11261). Since then, we have had numerous requests for a DCC Base Station or Booster.

Therefore, we have created this DCC Power shield, which is the final piece of the puzzle.

Adding this (and an appropriate power supply) to the Programmer, in conjuction with DCC-capable locos, results in a complete DCC system.

As this is an Arduino-based project, the following description assumes that you are familiar with the Arduino IDE (Integrated Development Environment).

To download the free IDE software, go to siliconchip.com.au/link/aatq



A complete DCC control system can be made by adding a Uno board and the DCC Programmer Shield (which we described in the October 2018 issue) to the DCC Power Shield, as shown here. Fit the DCC Programmer Shield with stackable headers, so it can be sandwiched between the other two boards, and take care that nothing shorts out between the adjacent boards. You may need to trim some of the pins on the underside of the DCC Power Shield.

Two locos, one track – but both are under individual control of the DCC system. As you can *just* see, the loco in front even has its headlight on – also switched on or off at will via DCC. Want more than two trains? DCC has up to 10,000 addresses available!

We are using version 1.8.5 of the IDE for this project, and suggest that if you have an older version installed, that you upgrade it now.

What is DCC?

We went into a bit of detail on DCC in the DCC Programmer article, so we'll only cover the basics here. If you want to learn more, read the aforementioned article from October 2018, and possibly the article describing DCC in detail from February 2012 (siliconchip.com.au/Article/769).

DCC is designed to allow multiple model trains to be controlled on a single track, with the same set of tracks carrying power for the trains and also digital control commands.

Older command controls systems exist; we detailed the construction of one such system (in five parts!) in 1998. This was named the Protopower 16, and it was based on another system called CTC16. This worked similarly to the system used to control multiple servo motors on model aircraft.

But that system was limited to 16 locomotives, while Digital Command Control has around 10,000 addresses available; probably well beyond the scope of most model railroads (and many full-scale railroads too!).

The most basic method of model train control is for a single throttle to apply a variable DC voltage to the track, which drives the train's motor directly. Instead, a DCC base station delivers a high-frequency square wave to the track. The base station encodes binary control data into this signal by varying the width of each pulse (see Fig.1).

A digital decoder on each vehicle receives commands and also rectifies the AC track voltage to produce DC. The decoder then uses this to drive the motor and can also control lights, sound effects (like a horn or engine) or even a smoke generator.

There are also accessory decoders which can be used to control things such as points and signals using the same DCC signals.

The DCC standard is produced by the National Model Railroad Association (based in the USA; see siliconchip.com.au/link/aaww). These standards are available for download, which means that anyone can use them. As a result, many different manufacturers are making DCC-compatible equipment.

Our Base Station will work with many commercially-available decoders. There is a vast array of manufacturers of DCC equipment, so we can only test a small subset. All of those we have tested have worked well, as should be expected from a proper application of the standard.

Terminology

A Base Station in DCC terminology is, essentially, the brains of the system. Typically it receives commands from attached throttles controlled by people, or perhaps a computer. These commands then dictate what DCC data needs to be sent to the trains to control them.

The Base Station generates a continuous stream of DCC data packets to control and update all trains, signals and points as needed.

A Booster is a simple device which takes a low-level DCC signal and produces a DCC signal of sufficient power to drive a set of tracks. Many smaller DCC systems consist of a single unit which combines a Base Station with a Booster, while larger systems might have separate units, including multiple Boosters.

Our DCC Power Shield works as a Booster. An attached and properly programmed Arduino board can be used as the Base Station smarts, thus creating a basic DCC system in a single unit. Extra DCC Power Shields can be deployed as separate Boosters, with an Arduino attached to monitor each and check for faults.

When programmed with the DCC++ software, the Arduino board and DCC Power Shield can be combined with

Features & specifications

- *Based on the Arduino Uno*
- *Provides a DCC output of 12-22V peak at up to 10A, or more with some changes*
- *Can operate as a base station or booster*
- *Compatible with DCC++ and JMRI (DecoderPro/PanelPro) software*
- *Opto-isolated input for use as DCC slave*
- *Works with our DCC Programmer shield from the October 2018 issue*
- *Can also be used as a brushed motor driver*
- *All Arduino pin assignments configurable via jumpers*

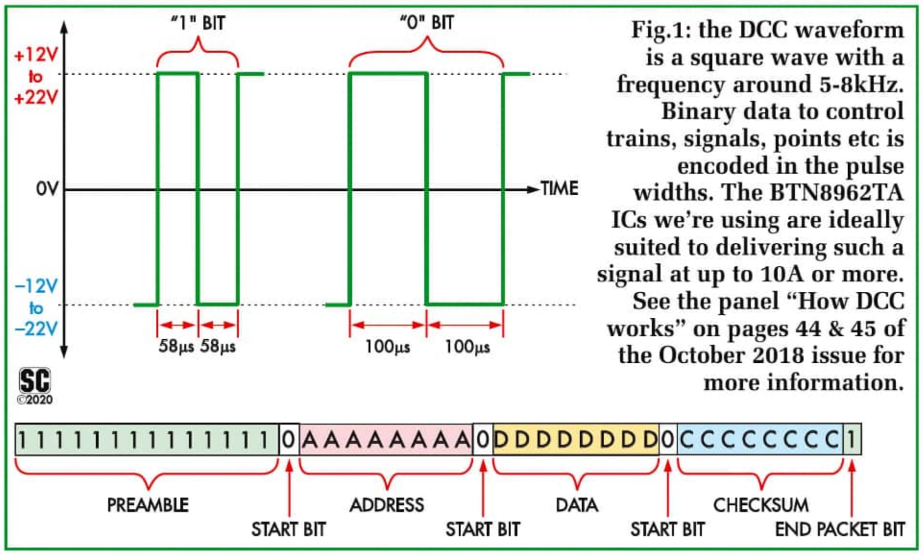


Fig.1: the DCC waveform is a square wave with a frequency around 5-8kHz. Binary data to control trains, signals, points etc is encoded in the pulse widths. The BTN8962TA ICs we're using are ideally suited to delivering such a signal at up to 10A or more. See the panel "How DCC works" on pages 44 & 45 of the October 2018 issue for more information.

our earlier DCC Programming Shield to create a compact, economical and fully-featured DCC system.

Power source

A DC power source is needed to run the DCC Power Shield. The DCC standards suggest that Boosters should produce 12V-22V peak, so your chosen power source needs a regulated DC output in this range.

For modest current requirements (up to around 5A), a laptop power supply is a good choice. Many of these have a nominal 19V DC output at several amps. Any fully DCC-compatible trains and decoders should handle this fine, but it's worth checking any that you aren't sure about.

Decoders are supposed to work down to around 7V. Given that the track, wiring and locomotives are bound to drop some voltage, a 12V 'power brick' type supply works well enough for driving trains. However, we found that this sometimes wasn't enough to allow decoder programming to occur.

If you need more current than a laptop power supply can provide, you will need to find a dedicated power supply in the 12-22V range. Many suitable high-power 'open frame' switchmode supplies are available from various suppliers.

One thing to note is that while some Arduino boards (including genuine boards) can tolerate up to 20V on their VIN inputs, some clones use lower-rated voltage regulators which can only handle 15V.

We have provided an option for a zener diode to help manage this variation; read on for more information on how the circuit works.

DCC Power Shield circuit

The circuit of the DCC Power Shield is shown in Fig.2. Its key function is to turn a steady DC voltage into a DCC-modulated square wave. For this, we need a full H-bridge driver. To keep it simple, we have used a pair of BTN8962 half-bridge driver ICs (IC1 and IC2).

The BTN8962 comes in a TO-263-7 package, which is a surface-mounting part, although quite a large one. It is not difficult to solder. There are two of these, one driving each side of the track. They are supplied with out-of-phase input signals to produce the required alternating output drive.

Their supply pins (pins 1 & 7) are connected directly across the incoming DC supply from CON1, labelled VIN.

A 100µF electrolytic capacitor bypasses this supply. While this may seem like a low value to use, the current drawn by IC1 and IC2 is quite steady as when one output goes high, at the same time, the other goes low. The outputs of IC1 and IC2 connect to screw terminal CON2, and then onto the tracks.

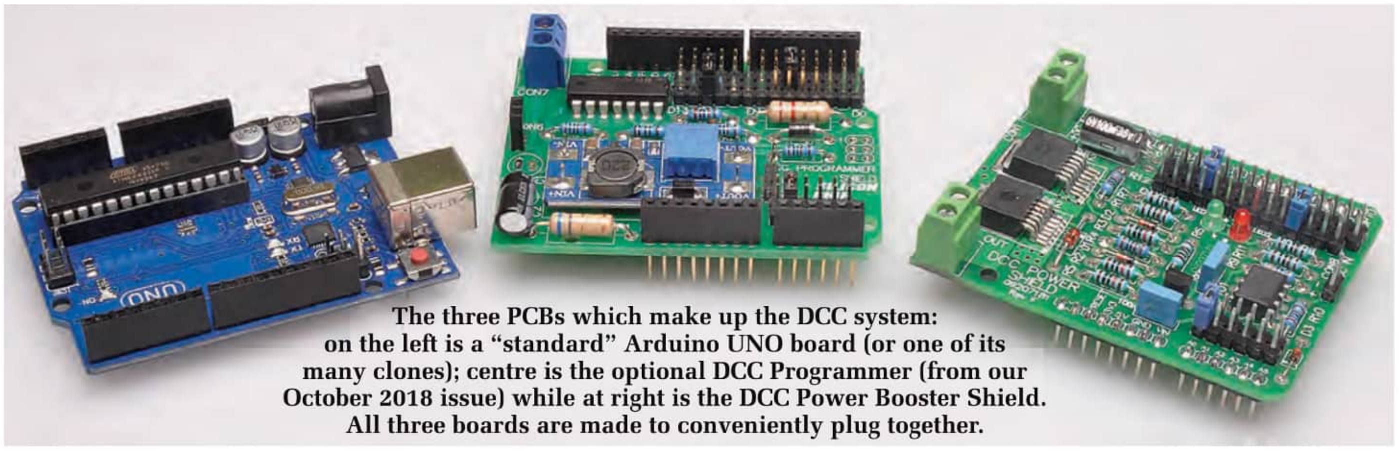
The state of the IN pins (pin 2) determines whether the output pins (4 & 8) are driven high or low. The SR input pin controls the output slew rate.

We've tied this to ground to give the fastest possible slew rate. The "INH" pins (pin 3) need to be brought high to enable the outputs. These are connected together and have a 100kΩ pull-down resistor so that the outputs default to off.

The enable signal connects back to an Arduino pin via a 10kΩ resistor and jumper JP1, allowing the Arduino to enable or disable the outputs as required. JP1 lets any Arduino digital pin connect to the enable signal, to suit the software used.

The IS pins (pin 6) on IC1 and IC2 are outputs that source a current proportional to the current being drawn from the output of each IC (plus a small offset current, which is compensated for in software). These currents are combined in a 'diode-OR' circuit formed by diodes D1 & D2 and then fed to a 1kΩ resistor to convert the combined current into a voltage.

This then passes to an RC low-pass filter (20kΩ/100nF) for smoothing. The 2ms time constant means that peaks in the current due to the rapidly changing



The three PCBs which make up the DCC system: on the left is a "standard" Arduino UNO board (or one of its many clones); centre is the optional DCC Programmer (from our October 2018 issue) while at right is the DCC Power Booster Shield. All three boards are made to conveniently plug together.

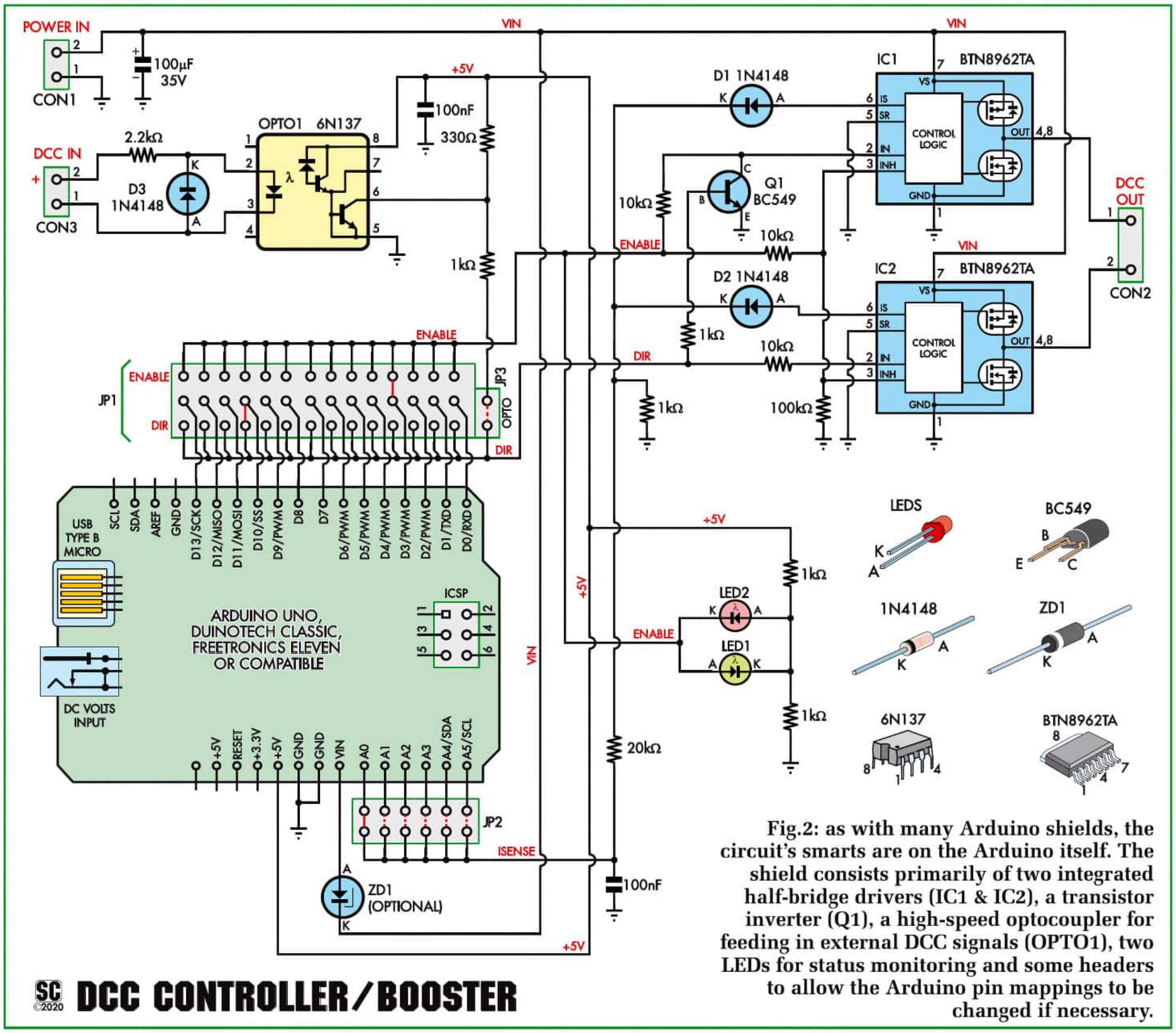


Fig.2: as with many Arduino shields, the circuit's smarts are on the Arduino itself. The shield consists primarily of two integrated half-bridge drivers (IC1 & IC2), a transistor inverter (Q1), a high-speed optocoupler for feeding in external DCC signals (OPTO1), two LEDs for status monitoring and some headers to allow the Arduino pin mappings to be changed if necessary.

DCC signal are ignored, but faults can still be detected quickly. The resulting smoothed voltage is fed to one of the Arduino analog input pins via jumper JP2, to allow the Arduino to monitor the track current.

JP2 allows any of the Arduino analog inputs to be used to monitor track current, again allowing us to choose whichever pin suits the Arduino software.

The IS pins will also source current if IC1 or IC2 detect an internal fault condition; as far as the software is concerned, this is equivalent to a very high current being drawn from the output and is treated the same way.

Bridge driving signals

The input signal to pin 2 of IC2 comes from another one of the Arduino digital outputs via a 10kΩ series resistor. Once again, any Arduino digital pin can be used, and this too is selected by a jumper shunt on JP1.

A simple inverter circuit produces the out-of-phase signal to drive the IN pin of IC1. The signal that goes to pin 2 of IC2 is also fed to the base of NPN transistor Q1 via a 1kΩ resistor. Q1's collector is pulled up by a 10kΩ resistor to the ENABLE line. So as long as ENABLE is high, meaning the outputs of IC1 and IC2 are active, input pin 2 of IC1 is inverted compared to input pin 2 of IC2.

Opto-isolated input

To allow a separate base station to be used, an optoisolated input is provided at CON3. This can accept a logic-level DCC signal, or even a 'track voltage' (12-22V) signal from another DCC system.

The signal at CON3 passes through a 2.2kΩ series resistor and into the LED of OPTO1. 1N4148 diode D3 is connected in reverse across this LED, to protect it from high reverse voltages.

If a logic-level DCC signal is applied to CON3, then the polarity markings need to be observed, as current will only flow through OPTO1 when the voltage at pin 2 is high. A bipolar DCC signal can be connected either way around.

OPTO1 is a 6N137 high-speed opto-isolator which has a nominal forward current of 10mA. Thus the 2.2kΩ resistor is suitable for voltages up to around 22V, ie, the maximum expected from a DCC system.

The output of OPTO1 is supplied

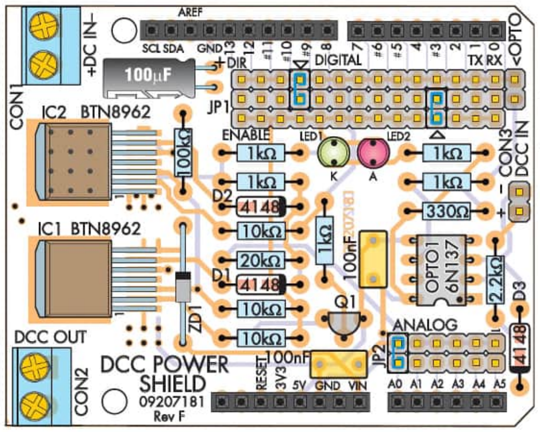
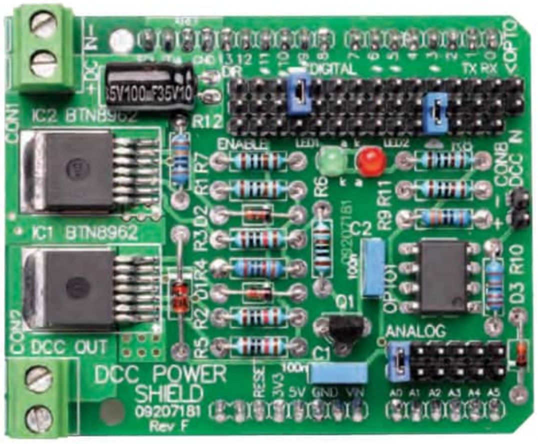


Fig.3: the seven-pin half-bridge driver ICs are mounted on the left, near the power input (CON1) and track (CON2) terminals. The jumper positions shown here are those required to use both the open-source DCC++ software and our example sketches. The jumpers are mostly handy if you want to use this shield as a DC motor driver, so that you can connect the required functions to PWM pins.



with 5V from the Arduino board, bypassed by a 100nF capacitor. A 330Ω pull-up resistor sets the logic high level.

The output from OPTO1's pin 6 is fed via a 1kΩ protection resistor to jumper JP3. This allows the DCC signal to be fed directly to the input of bridge drivers IC1 & IC2.

In this case, a jumper on JP1 can be used to feed the same signal to one of the Arduino's digital pins, which would then be configured as an input.

Due to the open-collector output of OPTO1, this signal is inverted compared to that applied to CON3.

But this can be solved simply by reversing the connections from CON2 to the tracks.

This reversibility of the DCC signal is a necessary feature, as a locomotive may be placed on the track either way and must be able to work with an inverted signal.

The only time this matters is when different boosters feed two adjoining tracks. In that case, you will need to make sure that the signals are in-phase.

Other features

Status LEDs LED1 & LED2 are connected to the ENABLE signal with 1kΩ current-limiting resistors to GND and 5V respectively.

So if ENABLE is high, green LED1 lights up, and if it's low, red LED2 lights up instead. If ENABLE is high-impedance, such as when the Arduino is in reset, neither LED lights. A single bi-colour LED could be fitted either for LED1 or LED2 to achieve the same effect.

If fitted, ZD1 feeds DC from CON1 to the VIN input of the Arduino board. Its value is chosen to limit the Arduino input voltage to a safe level at the maximum expected voltage from CON1.

For example, for 22V into CON1, ZD1 can be an 8.2V type, so 13.8V is fed to the Arduino VIN pin. A 1W, 8.2V zener diode can pass up to 120mA, which should be enough to power the Arduino and any connected shields.

We've left enough space to fit a 5W zener diode if you need more current than that, although if you're going to be applying less than 22V to CON1, you could also use a lower voltage zener, which could then pass more current before reaching its 1W limit.

For situations where the voltage on CON1 is suitable for direct connection to VIN (typically under 15V for clones or 20V for genuine Arduino boards), then a wire link can be fitted in place of ZD1.

However, it would still be a good idea to fit a low voltage zener (eg, 3.3V) as this will reduce the dissipation in the Arduino's regulator. Just make sure that the voltage fed to the Arduino's VIN pin will not drop below 7V.

If you aren't sure whether your Arduino can handle more than 15V, check the onboard regulator. It's usually in an SOT-223 three-pin SMD package with a hefty tab.

Genuine Arduino Uno boards usually have an NCP1117 regulator, rated to handle up to 20V. Clones often have an AMS1117 instead, which is only rated to 15V.

If ZD1 is left off, the supplies are separate (although their grounds will be connected). This allows the Arduino to be powered via its USB connector, eg, from a controlling computer.

DCC Programming

Many DCC Base Stations have a separate output for programming decoders.

In other words, programming is not done via the main high-current output driver, which is usually kept connected to the layout.

For this reason, you may wish to have the DCC Power Shield and October 2018 DCC Programmer shield plugged into the same Arduino. The DCC++ software is designed to handle this.

However, this does complicate the power supply arrangements a bit.

Firstly, the DCC Programmer shield has a maximum supply voltage of 15V, so regardless of the type of Arduino board you are using, you will need to ensure that the VIN pin is no higher than 15V.

Also, in this case, it would be best to build the DCC Programmer shield without the MT3608 boost module, and fit the jumper shunt on CON8 between pins 1 and 2, so that the VIN supply is used for programming power.

The DCC Programmer shield can draw up to 200mA from VIN, so the dissipation of ZD1 will increase substantially. You will need to choose its value carefully, or use a 5W zener.

Another option, if the system will always be connected to a computer, is to build the DCC Programmer Shield with the MT3608 boost module and fit it below the DCC Power Shield, then leave out ZD1 from the Power Shield.

The DCC Programmer Shield will then be powered from the computer's 5V USB supply, while the DCC Power Shield is still powered via CON1.

Construction

The DCC Power Shield is built on a double-sided PCB in a typical Arduino shield shape, coded 09207181 and measuring 68.5 x 55mm. Use the overlay diagram, Fig.3, as a guide during construction.

Start by fitting IC1 and IC2. As you

can see, although these are surface-mounting components, they are quite large. Because of this, and the fact that they sit on large copper pours, it will require quite a bit of heat to make good solder joints.

Flux paste and solder braid (wick) will come in handy, as will tweezers. Apply some flux paste to the pads first, to make soldering easier.

Working on one at a time, start by tacking one of the end pins in place to locate the device.

Once you are happy that each is centrally located within the footprint, load some solder on the tip of your iron and apply it to each of the smaller pads. Ensure that the resulting solder fillets are solid.

Use the solder braid to remove any solder bridges. The two end pins, numbers 1 and 7, are ground and power respectively. It's a good idea to add a bit of extra solder to these pins to help with current and heat handling.

Finally, solder the large tab of each device. Hold the iron tip at the point where the tab meets the pad on the PCB. Heat the pad until it melts solder applied to it. Feed in solder until a rounded, but not bulging fillet is formed and allow it to cool.

Next, fit the 12 resistors. The PCB silkscreen is marked with the values, and you should check these match with a multimeter as they are fitted, to ensure they are the correct value. Solder close to the PCB, then trim the leads close to the underside.

Then install the three small 1N4148 diodes (D1-D3) where shown in Fig.3, ensuring that they are correctly orientated

If fitting ZD1, do that now. Make sure that its cathode band faces towards the top of the PCB. Then mount the rectangular MKT capacitors, which are not polarised.

Now install NPN transistor Q1, with its body orientated as shown. You may need to crank the leads out to fit the PCB pads. Solder it in place, ensuring it is pushed down firmly against the PCB. If you plan to fit another shield above this one, then its top should not be more than 10mm above the PCB.

The electrolytic capacitor should be mounted on its side to allow another board to be stacked above this one. Its longer, positive lead must go in the pad towards the top of the board as shown.

Fit OPTO1 next. Check that its notch or pin 1 dot faces in the direction shown. Carefully bend the pins to allow it to fit into the PCB pads and solder it in place.

Parts list – Arduino DCC Controller

1 Arduino Uno or equivalent
1 12-22V DC high-current supply (see text)
1 double-sided PCB coded 09207181, 68.5mm x 55mm
1 set of Arduino headers, standard male or stackable (1 x 6-way, 2 x 8-way, 1 x 10-way)
2 2-way 5/5.08mm pitch PCB-mount screw terminals (CON1,CON2)
  [Jaycar HM3172, Altronics P2032B]
2 15-way pin headers (JP1,JP3)
1 14-way pin header (JP1)
2 6-way pin headers (JP2)
4 jumper shunts/shorting blocks

Semiconductors
2 BTN8962TA half-bridge drivers, TO263-7 (IC1,IC2) [Digi-key, Mouser]
1 6N137 high-speed optoisolator, DIP-8 (OPTO1)
1 BC549 100mA NPN transistor (Q1)
1 green 3mm LED (LED1)
1 red 3mm LED (LED2)
1 1W or 5W zener diode to suit your situation (ZD1; see text)
3 1N4148 signal diodes (D1-D3)

Capacitors
1 100µF 35V electrolytic
2 100nF MKT

Resistors (all 1/4W 1% metal film)

1 100kΩ	1 20kΩ	3 10kΩ	1 2.2kΩ	5 1kΩ	1 330Ω

Headers

The various headers should be fitted next. Note that if you already know which Arduino pins will be used for the DIR, ENABLE and ISENSE signals and they will not change, you could omit JP1-JP3 and fit wire links in their places.

To connect to the Arduino, you can use either regular headers or stackable headers. We recommend using the Arduino board as a jig to ensure that the pins are square and flush to the PCB.

Stackable headers can be more tricky to mount as they need to be soldered from below. If possible, use those with 11mm-long pins (some that have 8mm pins, which don't leave much room to solder).

Thread the headers through the shield and into the Arduino board. Flip the whole assembly over so that the shield is resting flat against the pins, then solder the end pins of each group in place to secure the headers.

You can then remove the shield from the Arduino board and solder the remaining pins in place, before retouching the end pins.

It's easiest to use single-row pin headers for JP1-JP3, snapped to length and soldered side-by-side for JP1 and JP2.

If you are snapping 40-way headers to do this, you will need at least two. Rather than fitting JP3 as a separate two-way header, you can make the top two rows of JP1 longer by one pin (ie, 15 pins rather than 14).

The last step in the construction is to fit the two screw terminals to CON1 and CON2, with their wire entry holes facing the outside edge of the board. Ensure that they are flat against the PCB; this is particularly important if you need to stack a shield above this one.

You may need to trim the underside of CON2, as this could foul the DC jack of an attached Uno board. Similarly, the underside of CON1 comes close to the metal shell of the USB connector of an attached Uno.

It's a good idea to add a layer of electrical tape on top of the USB connector on the Arduino board, to make sure they can't short if the boards flex.

Jumper settings

We suggest that you connect DIR to D10, ENABLE to D3 and ISENSE to A0, as shown in Figs.2 & 3. This suits our software. There are triangular silkscreen markings on the PCB to indicate the default jumper locations for JP1.

To use the board as a DCC Booster with our supplied software, add a fourth jumper across JP3 at upper-right.

Software

There are a few different ways this shield can be used, and each has its own software requirement. We'll describe a few of these possibilities. The following assumes that you have fitted the jumpers to the default locations described above.

DCC++

We mentioned the DCC++ software in our October 2018 article. It is designed to work with either an Uno or Mega board; we paired it with the Uno previously, and the discussion in this article assumes the same.

The Uno is adequate to work with the JMRI (Java Model Railroad Interface) software and will naturally cost less than a Mega.

The DCC++ project also includes a Processing-based GUI application for your PC that can interface with the Base Station, although this has been customised to work with a layout belonging to the DCC++ software designer.

Alternatively, you can use JMRI. We also covered this software in the previous article. JMRI can be downloaded from **www.jmri.org/download/index.shtml**

There are versions for macOS, Windows and Linux. It can even be run on Raspberry Pi single-board computers.

Follow the installation instructions, including installing Java if necessary.

As we mentioned, our hardware is compatible with DCC++ in base station mode.

There is more information, including the required Arduino sketch, available for download from: **https://github.com/DccPlusPlus/BaseStation**

This software is designed to work with several commonly-available Arduino motor driver shields. But these shields need some modifications to work, whereas our hardware only requires the correct jumpers to be set. The default setting in DCC++ for the MOTOR_SHIELD_TYPE of '0' will work with our hardware.

Open the Arduino IDE, select the Uno board and its serial port via the menus and open the DCC++ Base Station sketch that you've downloaded. Then upload the sketch to the Uno. If you open the serial monitor at 115,200 baud, you will see a banner message; this indicates that the Base Station software is working as expected.

You can also interact with the Base Station through serial commands. The protocol is detailed in the PDF file that is included in the DCC++ Base Station project ZIP file.

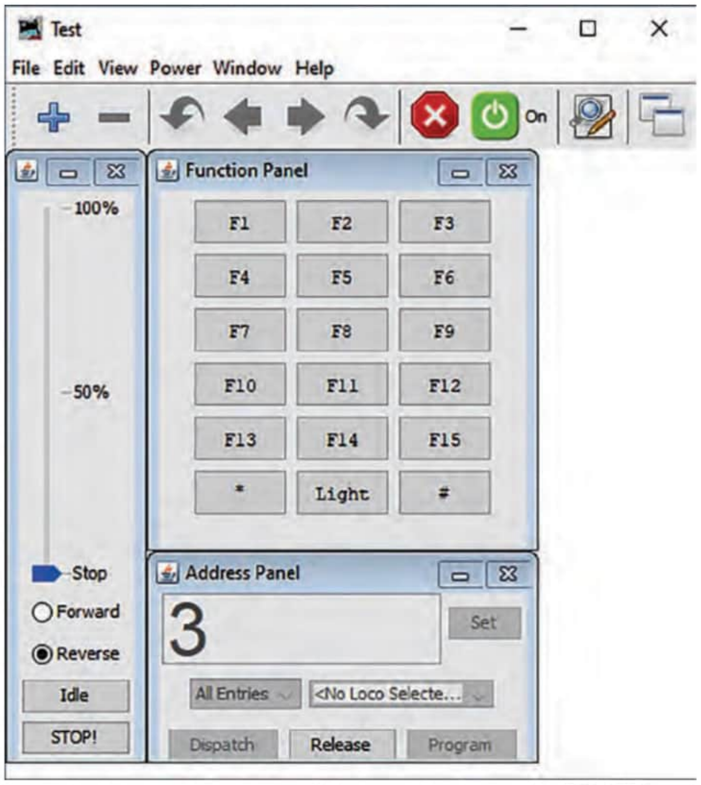
Once you have tested this, close the Serial monitor and open the DecoderPro program. Go to Edit -> Preferences, and under Connections, choose DCC++ as System Manufacturer, DCC++ Serial Port as System connection. Ensure that the serial port setting matches that of the Uno.

Save the settings and close DecoderPro, so that it can reload the new settings. Re-open DecoderPro and under Edit -> Preferences, choose Defaults, and ensure that the name of the new connection name is used for all connections (instead of "Internal").

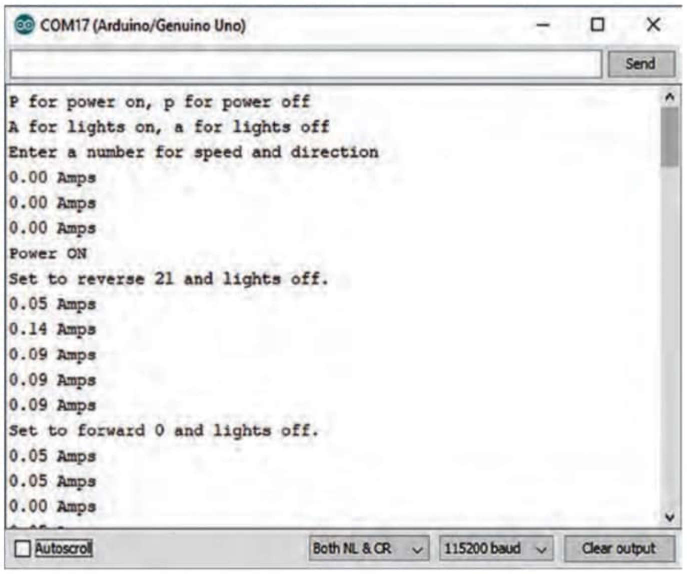
Unless you have other hardware you want to use, you should select DCC++ for all options.

Save, close and re-open DecoderPro again. Click the red power button in DecoderPro and ensure that it turns green. The LED on the DCC Power Shield should switch from red to green.

The simplest way to drive trains is to select Actions -> New Throttle, set the locomotive address and manipulate the controls (see Screen1).



Screen1: while JMRI's DecoderPro program has many features, it also has a set of basic tools for controlling trains. This throttle window allows speed, direction and light functions to be controlled. You can even switch track power directly; the green icon at upper right mimics the status LEDs on the shield.



```
P for power on, p for power off
A for lights on, a for lights off
Enter a number for speed and direction
0.00 Amps
0.00 Amps
0.00 Amps
Power ON
Set to reverse 21 and lights off.
0.05 Amps
0.14 Amps
0.09 Amps
0.09 Amps
0.09 Amps
Set to forward 0 and lights off.
0.05 Amps
0.05 Amps
0.00 Amps
```

Screen2: while very basic, our standalone sketch named "DCC_Single_Loco_Control.ino" allows power, speed, direction and lights to be controlled by commands in the serial monitor. The software can be modified to control multiple locos. Advanced Arduino users could use it as the basis of an automated layout control system.

JMRI can do a lot of different things, so we suggest you read its manual to find out about its capabilities. The JMRI project also includes PanelPro, which can be used to design track and signal diagrams for controlling a model layout.

Adding the DCC Programmer

If you have already built the DCC Programmer, then the Arduino board is already programmed to work with the DCC Power Shield, and the DCC Power Shield can be added to the stack, ideally at the top.

As noted earlier, the choice of zener diode and power supply will be more complicated if you want to construct an all-in-one setup. Since this is likely to be a smaller system, we suggest that a modest power supply will be suitable.

Using the DCC++ software with JMRI is the same as noted above.

Using it as a booster

When a signal is fed in via the optoisolated input (CON3), the DCC Power Shield is effectively working as a booster. The signal can be from another Base Station or system, with the DCC Power Shield turning that signal into a more powerful DCC signal that can be used to drive trains.

While it might not seem that an Arduino is needed in this case, it's a good idea to have one as we can program it to monitor the DCC signal and intervene if there is a problem. So we've written a sketch to allow an Arduino to take on this supervisory role.

There are two main conditions to check for. Firstly, we want the booster to be able to protect the shield if too much current is being drawn from it.

This could be due to an overload or even a short circuit, such as a metal object being dropped across the tracks.

Thus, our sketch continually monitors the voltage present on its A0 pin via its internal analog-to-digital converter (ADC). If it gets above a certain threshold, the power to the track is cut by pulling the ENABLE pin low.

A timer starts and the sketch attempts to re-apply power after it expires. If the short circuit is still present, then the over-current condition re-occurs, power is cut again and the timer re-starts.

The other condition we need to consider is if the incoming DCC signal is lost. This could be for any reason, such as if the connection to CON3 is broken or the upstream DCC Base Station has a fault. In any case, when there is no signal at CON3, the input to IC1 is held high and IC2's input is low. There is then an unchanging DC voltage across the tracks.

This may not sound like a problem, but some DCC locomotives can be programmed to undergo 'DC conversion'.

When a locomotive decoder detects that there is a steady DC voltage present, the locomotive behaves as if it was on a conventional 'single-throttle' layout and will typically set off in one direction at full speed (hopefully not towards the end of the track…).

This feature was initially added to allow DCC locomotives to be used on conventional layouts, perhaps as an aid to owners transitioning to DCC from DC systems.

Fortunately, the DC conversion feature can be turned off in the decoder by setting a configuration variable. You can use a DCC Programmer such as from our October 2018 article to do this.

In any case, the sketch detects that the DCC signal is no longer changing and pulls the ENABLE line low, disabling the track output and preventing such runaways.

To enable the use of the optoisolated input, add a jumper across JP3. Leave the jumper on 'DIR' for pin D10 in place; D10 is set as an input in the software and is used to monitor the incoming DCC signal.

The Booster sketch is called "DCC_Shield_passthrough_supervisor.ino". This uses a library to perform the precision timing needed to generate the DCC waveform, called "TimerOne".

This can be installed via the Library Manager by searching for "timerone" or from the ZIP file we have included with our software package.

Open the sketch, select the Uno and the serial port and upload it. Disconnect the USB cable and connect your power source to CON1. The red LED should light. Connect a valid DCC signal to CON3 and the green LED should light. You should then have a valid DCC signal at CON2.

Using the DCC Booster Shield as a motor driver

The DCC Booster Shield can be used as a high-current motor driver shield. In this case, the signal on the DIR pin determines the motor direction, and a pulse-width modulated signal is applied to ENABLE to control the speed.

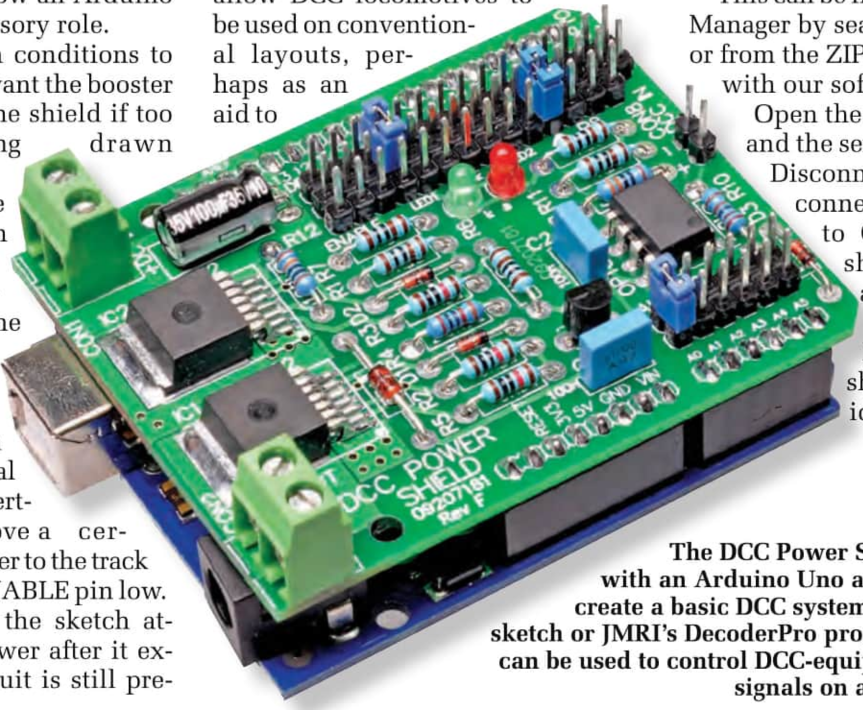
The BTN8962 has active freewheeling, so no external diodes are needed.

If used like this, LED1 and LED2 will both appear to be on at the same time, with green LED1 becoming brighter and red LED2 dimmer as the duty cycle increases.

As noted earlier, the 100µF electrolytic capacitor is adequate for a DCC application. A larger value may be needed for motor driving.

We suggest leaving ZD1 off, as larger motors will create hefty spikes at the end of each drive pulse.

Keeping the two supply rails separate will prevent this from damaging the Arduino board.



The DCC Power Shield can be combined with an Arduino Uno and DC power supply to create a basic DCC system. Using our standalone sketch or JMRI's DecoderPro program, this combination can be used to control DCC-equipped trains, points and signals on a model railway layout.

A standalone sketch

We've also created a simple standalone sketch that produces a DCC signal, suitable for controlling a single locomotive.

The decoder identification number has been set to 3 (which is the default for new, unprogrammed decoders), although it can be changed in the code.

We suggest you use this option if you want to try out DCC for the first time.

We can't offer advice on fitting decoders; there are so many options for both decoder choices and how they are connected.

The companies that make the decoders do offer advice (and many have custom decoders to suit specific locomotives).

After all, they want to make it easy for you to buy their products.

Our standalone sketch also requires the "Timer One" library mentioned above, so make sure that is installed

Set the jumpers on the shield to the default positions and connect the Uno to the computer. Open the "DCC_Single_Loco_Control.ino" sketch and select the Uno board and its serial port. Press the Upload button to compile and upload the sketch, then open the Serial Monitor at 115,200 baud (see Screen2).

You can now enter commands as numbers which correspond to the desired locomotive speed, in 128 steps. Thus, numbers from -127 to 127 are accepted. You should ensure that 28/128 step speed mode is set on your locomotive decoder.

Type "P" (upper case) to turn track power on and "p" (lower case) to turn it off. The power will automatically turn off if current over half an amp is detected. You can also use "A" and "a" to turn on and off the loco's headlights.

The program is elementary, but it has several unused functions to send all manner of DCC packets to the track. If you are comfortable with Arduino, you should have no trouble adapting it to do something more advanced.

Current limitations

Using the specified components and the DCC++ software, the shield can easily deliver up to 10A. This is mostly limited by the screw terminal connectors. The DCC++ software also has a hard-coded current limit which kicks in at around 10A.

Of course, the software limit is easy to change, but any hardware changes should be done with care.

The output driver ICs are capable of handling around 30A, with the PCB tracks topping out around 20A.

In any case, everything runs cool well below the 10A limit, so maintaining this limit is good for component longevity.

DCC has a wide range of operating voltages, so to increase output power, it may be easier to increase the supply voltage.

Most locomotives use PWM speed control on their motors, so a higher supply voltage simply means a lower PWM duty cycle (and thus current consumption) for the same speed.

We haven't done any tests above 10A, but if you are set on increasing the current capacity of the DCC Power Shield, then you should ditch the screw terminal connectors and solder thick copper wires directly to the board (ideally, to the power pins of IC1 & IC2).

If the wires can handle 20A, then your modified DCC Power Shield should have no trouble doing that.

To go higher than this will probably mean that IC1 and IC2 need some heatsinking, as well as even thicker wires. We suggest that you instead consider using more, smaller boosters. For example, you could modify the Booster sketch to monitor and drive multiple DCC Power Shields stacked above it.

A larger system

If you are planning a system with multiple Boosters, either because you need the power or it otherwise makes sense to do so, then there are a few minor caveats.

When running multiple boosters, avoid daisy-chaining the DCC signal from one Booster to the next. Instead, fan out the DCC signal from one Base Station to all the Boosters.

Many commercial base stations have a low-powered DCC signal output (Digitrax names this Railsync), which is ideally suited for this purpose.

The first problem with a daisy-chain configuration is that if one Booster goes down, then so do all those that are downstream, as the DCC signal will be shut off.

Secondly, each Booster also has a small but measurable delay in propagating the signal. In our case, this is around 4μs, due to the switching time of the BTN8962s.

This delay is not usually a problem, but it may become one at the boundary where the tracks from two Boosters meet (where there would typically be an insulator, to prevent one Booster feeding another Booster's section of track).

Where the tracks meet, a train may be briefly fed by both the Boosters. If there is a delay between the signals from the two Boosters, then it may appear to be a short circuit if the two Boosters are delivering opposite polarity voltages at that instant.

This is less likely to occur if the Boosters are well synchronised, which should be the case if all are being fed the same signal.

You should also ensure that the Boosters are fed with similar supply voltages, so that one Booster does not try to power another Booster's track when the train bridges their join.

You must also ensure that the Boosters are wired with the correct polarity where the tracks meet.

For situations where the polarity can change (such as in a reversing loop), check out our Reverse Loop Controller in the October 2012 issue (**siliconchip.com.au/Article/494**). SC