DCC Accessory Decoder Using Arduino Nano

by Computski

Low cost DCC accessory decoder to drive 10 servos using an Arduino Nano and a Nano expansion board plus a few discrete components.

This unit is powered from the DCC track supply, it draws about 120mA. If you power it this way, there is no need for an opto isolator to couple in the dcc data signal.

It is capable of driving 10 independent servos, e.g. low cost SG90 types. If you are driving a cross over where the two sets of points must always either be thrown or closed, then you can common the two servos off the one output. This way you can drive even more servos. e.g. on my own shunting layout I have 12 servos driven off the 10 outputs.

The unit is programmed through the serial interface. You can set a DCC address for each output. Several outputs can also share the same address, and the output direction can be inverted if required. This way you can drive a cross over from two outputs each with a servo and not pay attention to the mechanical orientation of the servos, as you can invert one if required, and then assign them the same DCC address. This is the fool proof way of wiring a cross over compared to commoning the servo control on a single output which also requires you to have the mechanical orientation of your servos consistent.

When setting up, you can set the desired servo swing range and also park the servo mid-travel, which is useful to ensure a symetric swing about this. You can command individual pins to test a single servo, or emulate a DCC command to test a response to a DCC address.

You may also set the servo to power-down after moving, or be continuously powered.

Supplies:

Arduino nano board

Arduino nano expansion board, type with on-board regulator and servo-type output headers

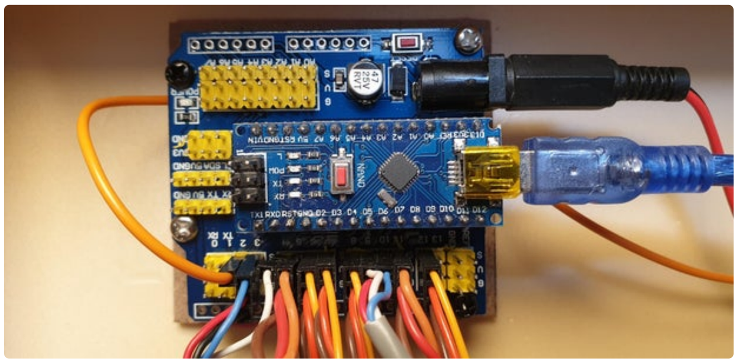
1 x small bridge rectifier eg W04

1 x 1000uF 16v electrolytic

1 x DC jack plug 2.1mm

1 x 10k resistor

1 x 1n914 signal diode or equiv



Step 1: Assemble the DCC Rectifier and Data Feed

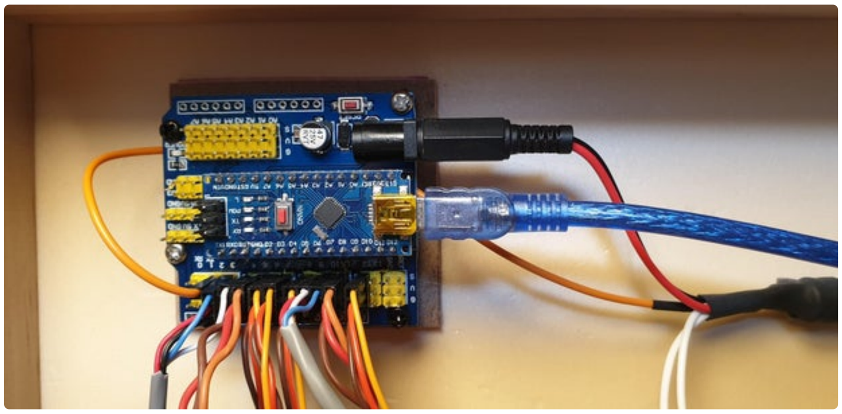
To power the unit we will rectify the DCC track power to generate a 12v feed. This also prrovides a DCC data signal to the unit via a diode and resistor. See schematic.

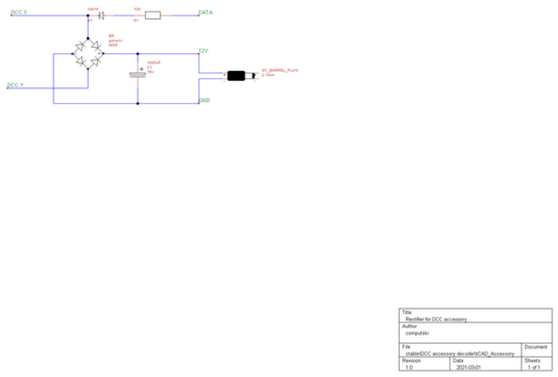
To protect the arduino input (pin2) we arrange the data feed to be active pull-down via a diode and resistor. The resistor limits the current to a safe level if something goes wrong and also guards against current induced lock-ups on the input pin. The nano pin2 is set as an input with on-board pull up. This way the nano pulls itself up, and is driven active low by the dcc data signal.

You can use any small bridge-rectifier such as the W04. I used a surface mount device and was able to heat-shrink the assembly into something little bigger than the smoothing capacitor.

The rectified DC is feed via a jack plug into the nano expansion board. This has an on-board 5v regulator and will power the nano.

NOTE: because we are powering the unit from DCC track power, there is no need to opto isolate the data feed into the Nano. This saves on cost and complexity.





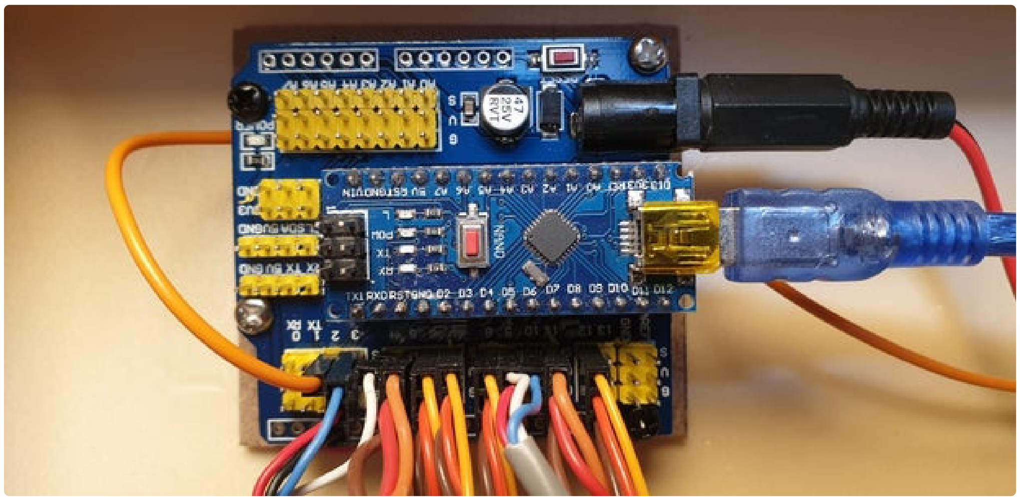
Step 2: Assemble the Nano + Expansion Board + Connect Rectifier

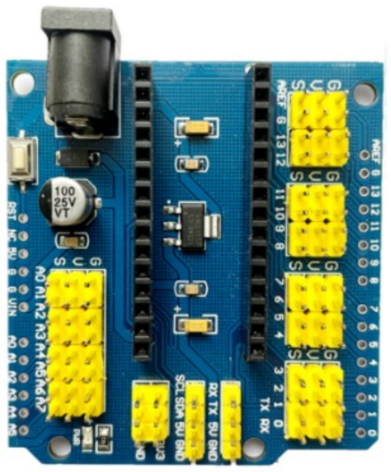
The nano just plugs straight into the expansion board. You can also use a UNO on this type of shield, however the extra spec of that device is not required.

The DCC data signal goes to pin 2 of the nano.

Pins 3 through 12 are the servo drives. On this expansion board, each output is conveniently arranged with ground, 5v, servo signal and is directly pin-compatible with servos such as the SG90. Easy.

Pin 13 is wired to the nano on-board LED and acts as a heartbeat indicator.





Step 3: Load the Software and Program the Unit

The software is in the INO file. This makes use of the NRMA dcc decoder library and also the servo controller library.

Once loaded, you should see the on-board led flash once per second. This is a heartbeat indicator that lets you know the unit is alive.

The servos now need to be set up through commands issued to the serial interface. You can set the DCC address, the swing range in degrees and whether the servo is 'inverted' i.e. that the closed position is extreme CW or extreme CCW rotation.

After you have assigned the servos their addresses and swing ranges, you can send commands to test them, and you can also command them to park half-way through rotation which is useful for mechanical set up on turnouts.

Note: all servo progamming is done via the serial port. It is not possible to program the unit over DCC, nor is this required. Generally you set everything up once and leave it as is. The servos are commanded to move with DCC turnout commands. You can also command them to move over serial for set-up / test.

CAUTION: You can program the unit and test servos purely on laptop USB power. If you wish to leave your laptop connected when the DCC track is powerered up, which you may want to do to observe the incoming DCC commands, then I recommend you run the laptop on battery power only. If you have it plugged into the wall, you might possibly create a ground loop through the laptop and DCC supply. I take no responsibiility for damaged laptops.

TIP: beware the latest version of the Arduino expects to talk to the nano via the new bootloader. Some of the generic nanos from ebay have the old bootloader installed. You need to change to 'old bootloader' in the Arduino IDE.

Download

https://www.instructables.com/ORIG/F9E/XWBG/KLQLCL5U/F9EXWBGKLQLCL5U.ino

Step 4: Program the Servo Outputs

This system is designed so that each servo output operates independently, and can be assigned a unique, or the same DCC address as any other output.

e.g. if you have a cross over with one servo wired to pin 3, and the second servo wired to pin 6, and you want both the servos to respond to DCC address 37, you program it like so;

```
s 3 37 40 0 0 \n

s 6 37 40 0 0 \n
```

where the command is s pin-assignment dcc-address swing-degrees invert(0=no) continuous(0=no)

you can then test this with a dcc emulation command

d 37 T\n where dcc address 37 is Toggled.

If you wish to set pin 6's servo mid point so you can ensure a symetric swing by moving the servo horn to the mdi position before screwing it down, you can do this with

p 6 n\n which commands that single output to the neutral position.

It is possible to progam swing ranges individually and also the movement 'sense' e.g. CW or CCW for closed. This is the 'invert' sense.

I recommend continuous=0. This de-powers the servo at end of movement and stops it from chattering.

Say you now want the servo on pin 6 to be inverted on the same DCC address and swing slightly more;

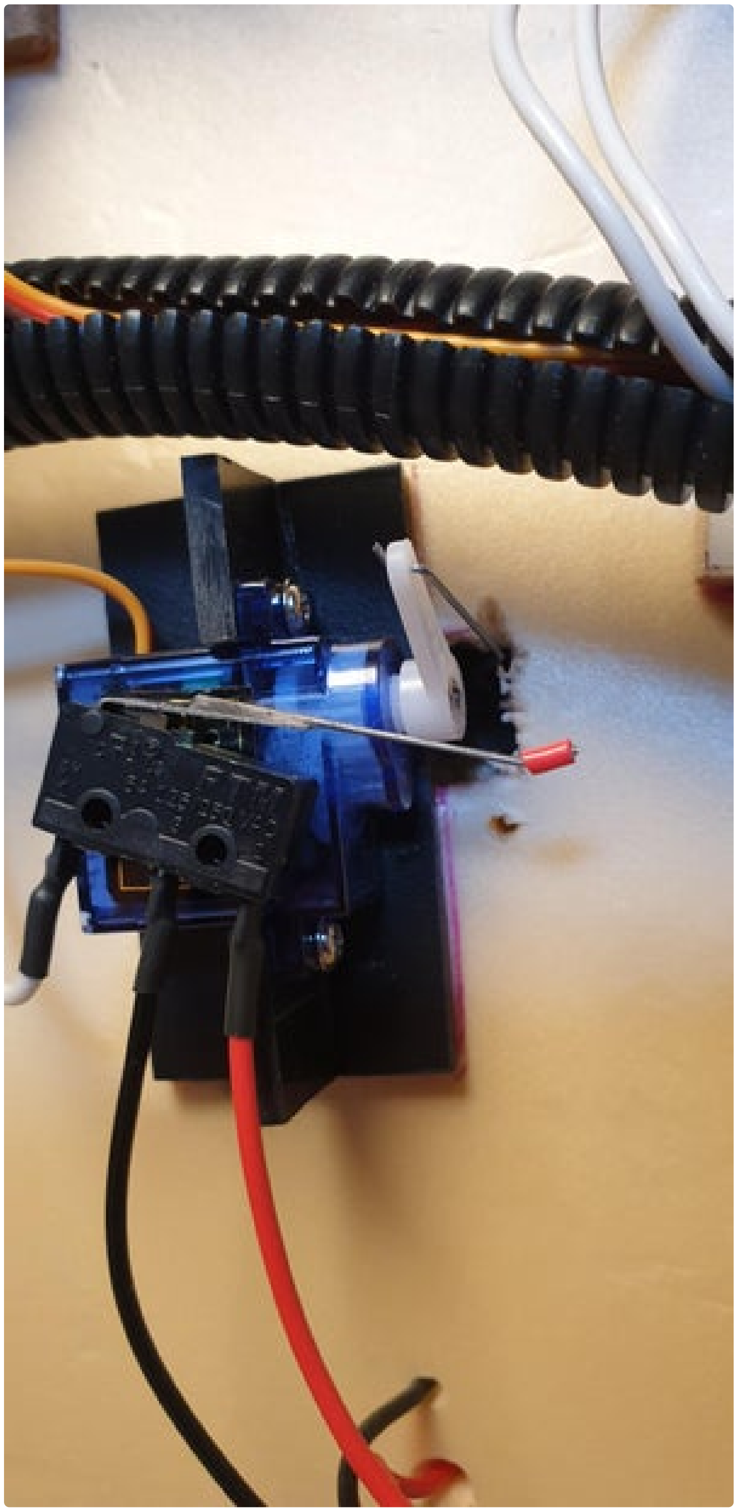
```
s 6 37 50 1 0\n
```

A few degrees extra swing can be useful to ensure the microswitch for powered frogs switches properly for example.

And to dump the currrent settings of all servos use x\n and you will see a dump like this;

```
pin 3 address 2 swing 30 invert 0 continuous 0
pin 4 address 1 swing 40 invert 1 continuous 1
etc for the remaining pins
```

Note: This system uses DCC addresses 1 to 2047 as discrete commandable addresses. This is common practice (aka 11 bit addresses) on commercial command stations, but it is a departure from the DCC specification itself which defines clusters 1 through 512 as an address, each with 4 individual turnouts on them (aka 9 bit addresses).



Step 5: Power Up DCC and Enjoy

Once you have programmed everything from the laptop over the serial interface, and run some dcc emulation tests using the d command, you are ready to power up the DCC.

CAUTION: You can program the unit and test servos purely on laptop USB power. If you wish to leave your laptop connected when the DCC track is powerered up, which you may want to do to observe the incoming DCC commands, then I recommend you run the laptop on battery power only. If you have it plugged into the wall, you might possibly create a ground loop through the laptop and DCC supply. I take no responsibiility for damaged laptops.

You can disconnect the laptop and then power up DCC. The unit will boot and after a few seconds you will see the heartbeat LED blink once per second. You should also hear all the servos move to their closed positions one by one over a period of about 5 seconds.

Now you can send some DCC commands to your turnouts and you should see them respond. Note that the nano decodes the DCC command and dumps it to the serial port, so you should see the TX LED blink breifly every time you send a DCC command to switch a turnout. If you don't, I recommend you check the DCC data feed is wired correctly and feeding into pin 2 on the Nano expansion board.

NOTE: some railoaders will advise that accessory controllers have their own power supply. One benefit of this, is that a loco driven over a turnout set against it will trip the DCC track power, but won't kill the accessory power because this is independent. BUT many locos have keep ailves fitted, so even if the track power cuts, the loco still climbs over the point and derails itself requiring hand-of-god intervention. This is why I prefer to keep it simple and power the accessory controller off DCC track power. It also obviates the need for an opto isolator.

I will post an instructable on servo mechanics soon. This will show how to build a servo mount from H channel PVC from your local hardware store, and use servos to support both turnouts and magnetic decouplers for the Kadee coupling system.