Build Your Own Quad Servo DCC Decoder – Update

Introduction

Following a number of comments from modellers who built Quad Servo DCC Decoders (QSDD), as published in the **February** and **March** 2020 issues of MRH Magazine, I have now made several improvements to both the software and the hardware of the original version.

Handling Multiple Servo Commands

The published version of the software (4.6) only allowed one of the four attached servos to move at any time, which was a bit of a limitation when setting layout routes involving a number of turnouts. So I set about adding some code to allow all servos to be operated simultaneously – only to discover that Geoff Bunza, who wrote the original code on which the QSDD is based, had already incorporated exactly that facility. The problem lay in the incorrect way I had modified and then called Geoff's routines. I have now fixed this so that all servos can respond simultaneously to DCC commands.

Programmable via JMRI Decoder Pro

While investigating the multi-command problem, I was contacted by another QSDD builder, Drew Aldridge from California, who wanted to be able to access the QSDD Configuration Variables using JMRI Decoder Pro. In my original article I had suggested that the reason this failed was that JMRI didn't recognise that accessory decoders could have CVs, since most commercial accessory decoders don't use them. However, Drew pointed out that he could read and write CVs from other accessory decoders which did use them, so I investigated properly this time. The real problem lay with my QSDD initialisation – which took far too long to complete after power was applied to the programming track. Because Decoder Pro was not getting a timely response, it legitimately cycled power to the track and the QSDD – which just repeated the failure cycle *ad infinitum* . . .

The solution, now implemented, was to speed up QSDD initialisation – which now takes a maximum of 0.15 seconds. As soon as power is applied, instead of leisurely switching all of the QSDD Keypad LEDs on, one by one, and then switching them off again, you will now see only a brief flash of the red Select LED before the QSDD is ready for JMRI Decoder Pro on your programming track, allowing you to read and write all of the QSDD CVs (as listed in Part 2 of the original article, or the documentation on my website at https://www.a-train-systems.co.uk/download.htm#Projects). When adding the QSDD to your roster in Decoder Pro, by clicking New Loco, select NMRA as the manufacturer (at the top of the list) and 'NMRA accessory decoders' as the decoder type.

The updated software for the Arduino Nano (QuadServo_DCC-Decoder_5-2.ino) can also be downloaded from my website page, as given above.

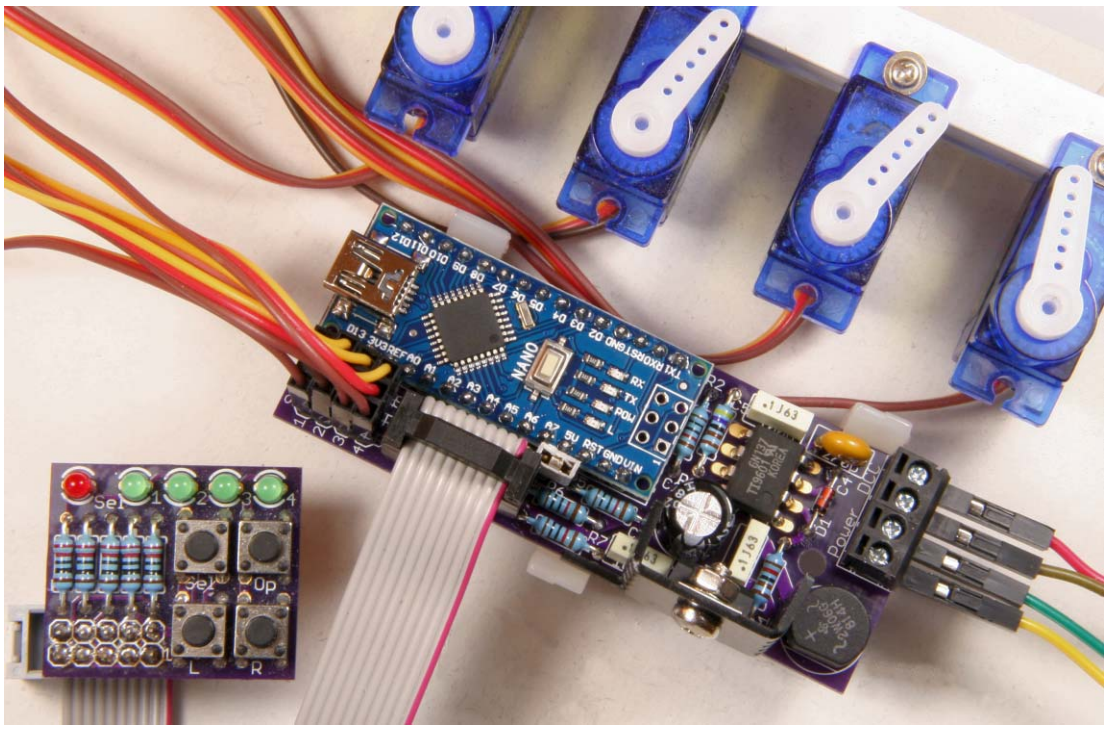
See the notes at the end of this document for details on accessing Configuration Variables.

Powered from External Supply

Several users were not keen on the QSDD drawing all of its power, including that for the attached servos, from the track DCC supply, since the stall current taken by a servo can be in excess of 600mA. A minor change has, therefore, been made to the QSDD hardware to provide a separate power input which will accept any supply from 9 to 15 Volts AC or DC. The DCC input from the track then has only to supply the 10mA or so taken by the optoisolator. Details of the change can be found in the updated schematic diagram at the end of this document.

Since there was no simple way to modify the existing printed-circuit board to separate the DCC input from the QSDD power supply, a new PCB was designed with an extra pair of terminals for

the external power supply, as shown on the picture below. If you are happy to power everything from the DCC track bus then simply connect both sets of terminals to the DCC supply.



All photographs by the author

As you can see in the photograph, the QSDD continues to use exactly the same small detachable keypad which plugs on to the decoder and is used to manually adjust the servo throws and set the speed of servo movement.

As before, the keypad can also be used to operate the servos by hand, using the fitted pushbuttons. For convenience, as well as being able to be plugged directly into the decoder, the keypad can alternatively be connected via a ribbon cable up to 40 inches long, allowing you to see and set up your turnouts when the decoder and servos are out of sight under the layout.

Once set up, there is no need to keep the keypad plugged in. It can, if you wish, be completely disconnected from the decoder board. The decoder will then drive the connected servos to operate your turnouts in response to standard DCC commands sent to the track from your command station, using either handheld controllers or suitable software (such as JMRI Panel Pro or my own A-Track application) running on your computer.

New PCB Available

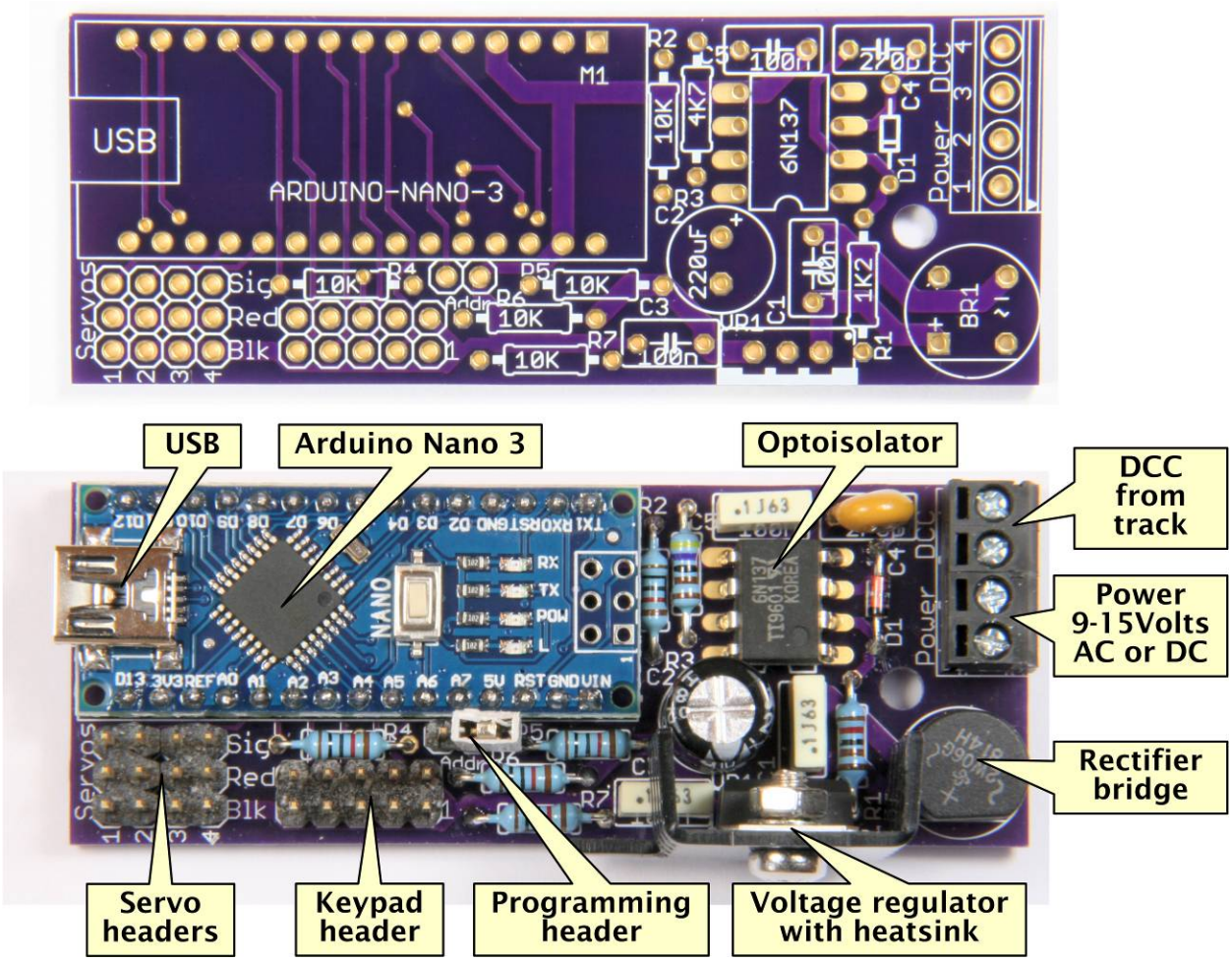
To support use of an external power supply, a revised printed-circuit board for the QSDD is available from OSH Park at https://oshpark.com/shared_projects/zMo5jegR where you can order a set of PCBs, or download the board file to send to any other PCB manufacturer of your choice. The QSDD-2 PCB is 0.1" longer than the original version so costs a few cents more – but, as an added bonus, it now includes a single 3.2mm dia. hole (next to the terminal block and rectifier bridge) to assist in mounting the assembled unit wherever required. The unchanged QuadServo-Keypad PCB can also be ordered from OSH Park if required (https://oshpark.com/shared_projects/7ATX5aqB).

The parts list and assembly details for the QSDD (as published in Part 1 of the original article) remain essentially as before, with the single addition of a second 2-way terminal block for the independent external power supply.

Both QSDD versions run the same (updated version 5.2) software, and the manual setup procedures and operation via the keypad remain exactly the same as described in the original article (and in the accompanying MRH video at https://www.youtube.com/watch?v=Ox-X1uAWssc&feature=youtu.be).

Assembly Details

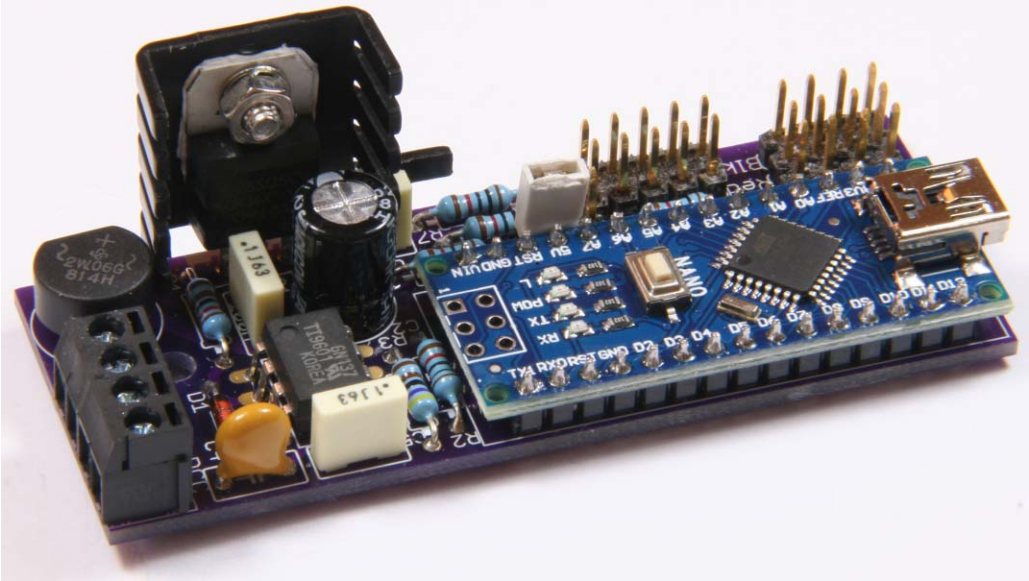
The only changes on the QSDD-2 PCB, after the addition of the Power terminal block and the mounting hole, are in the placement of the components around the optoisolator and the voltage regulator, as can be seen if the photographs below are compared to those in Part 1 of the original published article –



Also refer to Part 1 of the original article for a full parts list together with suggested sources of supply. Note that two 2-way terminal blocks are fitted rather than a single 4-way item since this is generally a cheaper option. You will find that 2-way and 3-way terminal blocks cost less, per terminal (especially if purchased in bulk), than larger blocks and, from most manufacturers, the small blocks are designed to clip neatly together to form larger blocks with any required number of terminals.

Assembly of the updated QSDD-2, plus a keypad, if required, follows exactly the same steps as the original with regard to the order of fitting components. A minor difference in the final stage of assembly is that, if the heatsink identified in the parts list is fitted with the fins facing inwards towards the centre of the PCB, you only need to carefully straighten one of the bottom pair of fins (beware of the soft aluminium cracking and breaking the fin away from the main heatsink body),

to clear capacitor C3. The other bottom fin now fits in the space between the rectifier bridge and resistor R1, as shown in the photograph below –



With the QSDD-2 assembled, you can now plug the keypad on to the 2 x 5 pin header on the decoder, either directly or using the ribbon cable described in Part 1, and connect the specified USB lead (A Plug to Mini B Plug) from your computer to the Arduino Nano module. If everything is assembled correctly, the Arduino Nano should light two of its on-board LEDs, one steady and one flashing (depending on the current internal state of the Nano), ready to receive its software program (QuadServo_DCC-Decoder_5-2.ino).

Full details of how to set up the Arduino IDE on your computer, and then use it to download the software to the QSDD can be found in Part 1 of the original article, together with the initial steps necessary to set up and test your newly-assembled decoder.

The additional steps required to program servo addresses of your own choice into the decoder, and to set the servo throws precisely to suit the turnout linkages on your layout were covered in Part 2 of the original article, and remain exactly the same for the updated version.

Accessing Configuration Variables and DCC Operations

You can access the QSDD Configuration Variables (CVs) either directly using your DCC system's Handheld controller, and following the manufacturer's instructions, or through your computer (and your system's Command Station) using either JMRI Decoder Pro or my own A-Track application.

However, in all cases you will need to connect your QSDD to the programming track. While this was straightforward using the original QSDD hardware (having only a single DCC connection), with the updated QSDD-2 hardware you need to disconnect the external power supply (and any USB cable) from the decoder, and connect both sets of terminals (DCC and external power) to the programming track. This ensures that the Command Station receives proper feedback from the QSDD, and its attached Keypad, when reading or writing CVs, in the form of a small current surge when all of the Keypad LEDs are briefly switched on.

The set of Configuration Variables used by the QSDD to hold its working data remains more or less as presented in Part 2 of the original article, with the addition of an Extended Decoder Version group (CV109 – CV112) to identify the decoder as a QSDD together with the software version, as listed in the table below –

CV No.	Default Value	Description
01	1	Board Address LSB – internal use – ignore any value loaded here
07	89	NmraDCC Version
08	13	Manufacturer (Do-It-Yourself)
09	0	Board Address MSB – internal use – ignore any value loaded here
29	226	Decoder Configuration – Extended Accessory + Output Addressing
41	1	Output 1 Address LSB
42	0	Output 1 Address MSB
43	2	Output 2 Address LSB
44	0	Output 2 Address MSB
45	3	Output 3 Address LSB
46	0	Output 3 Address MSB
47	4	Output 4 Address LSB
48	0	Output 4 Address MSB
50	0	Load Default CV Values if Not = 173 (0xAD), Auto set = 173 after load
55	6	Servo 1 - Rate (1 = Fast to 16 = Slow)
56	1	Direction - 1 = Normal Operation, 0 = Reverse Operation
57	70	Right Limit
58	110	Left Limit
59	70	Current Position
60	6	Servo 2 - Rate (1 = Fast to 16 = Slow)
61	1	Direction - 1 = Normal Operation, 0 = Reverse Operation
62	70	Right Limit
63	110	Left Limit
64	70	Current Position
65	6	Servo 3 - Rate (1 = Fast to 16 = Slow)
66	1	Direction - 1 = Normal Operation, 0 = Reverse Operation
67	70	Right Limit
68	110	Left Limit
69	70	Current Position
70	6	Servo 4 - Rate (1 = Fast to 16 = Slow)
71	1	Direction - 1 = Normal Operation, 0 = Reverse Operation
72	70	Right Limit
73	110	Left Limit
74	70	Current Position
109	81	Extended Decoder Version 1 – “Q”
110	83	Extended Decoder Version 2 – “S”
111	68	Extended Decoder Version 3 – “D”
112	82	Software Version – Hex = 0x52

In addition to this "working set", there are three other CVs which control "special" functions –

CV No.	Default Value	Description
120	0	Set = 120 to load Default CVs – must be cleared manually
121	24	Address to set CV Values in Operations Mode (Program on Main) – LSB
122	0	Address to set CV Values in Operations Mode (Program on Main) – MSB

Dealing first with the "special" functions, the address held in CVs 121 and 122 is used to write a new value to any CV within the decoder using Operations Mode (otherwise called Programming on the Main). Set your DCC system via your Handheld controller to Operations Mode and prepare to use a Locomotive Address (**not** an Accessory Address) equal to the address held in CVs 121 and 122, which is 24 by default.

Select the CV number to be programmed, enter the new value, and press ENTER (or whatever key is required by your system to complete the command). Since it is not possible to read back the values of CVs in Operations Mode, you will have to judge, by the subsequent behaviour of the decoder, whether the change of CV value was a success.

The other "special" function, initiated by programming a value of 120 into CV120 (via Operations Mode), is handled by the NmraDcc library and loads all CVs with their default values the next time the decoder is either reset or powered up. Reset is accomplished by pressing the Reset button on the Arduino Nano or, by holding down both the **L** and **Op** pushbuttons on the keypad and then releasing them.

However, it appears that CV120 is not cleared automatically by the NmraDcc library, so that CVs continue to be reset to their default values each time the decoder is restarted until you change the value in CV120 yourself.

To avoid this inconvenience, the QSDD has been set up to use CV50 as an alternative. Any value *except* 173 in this CV will reset all CVs to their default values when the decoder is next started or reset, following which the value of CV50 will be set to 173 (hex 0xAD = "All Default") automatically. This means that the default load only occurs once, and will happen automatically when the decoder sketch is loaded into the Arduino Nano for the first time (when all CVs will normally contain the value 255).

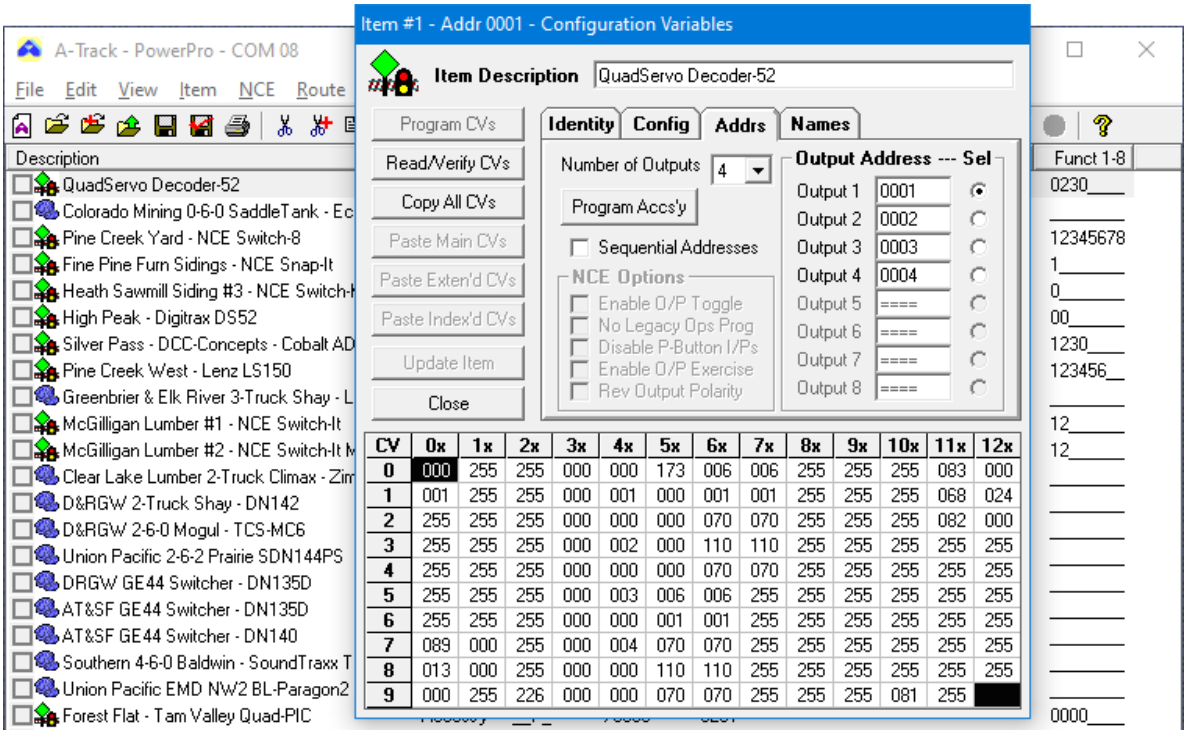
Assuming that you have the QSDD connected to your programming track, as described in the introduction to this section of the document, and have the keypad attached to the decoder, you can then read or write any of the CV values. Following the software update to software version 5.2 all of the restrictions on the type of DCC system you can use for programming, as outlined in Part 2 of the original article no longer apply, and any variety of Command Station will work.

If you have any type of NCE DCC system plus a Windows computer, my own application, A-Track (www.a-train-systems.co.uk/atrack), will happily read and program the QSDD CVs and allow you to save a record of them to file. You can download A-Track and its comprehensive documentation from my website (https://www.a-train-systems.co.uk/download.htm#ATWindows).

The principal advantage of being able to save a complete set of CVs to file shows itself when you have a layout with a lot of turnouts and multiple QSDDs to drive them. After setting up one decoder, and its four associated turnouts, to your satisfaction, you can then take a copy of the amended CVs and transfer them in a single operation to all of your other QSDDs.

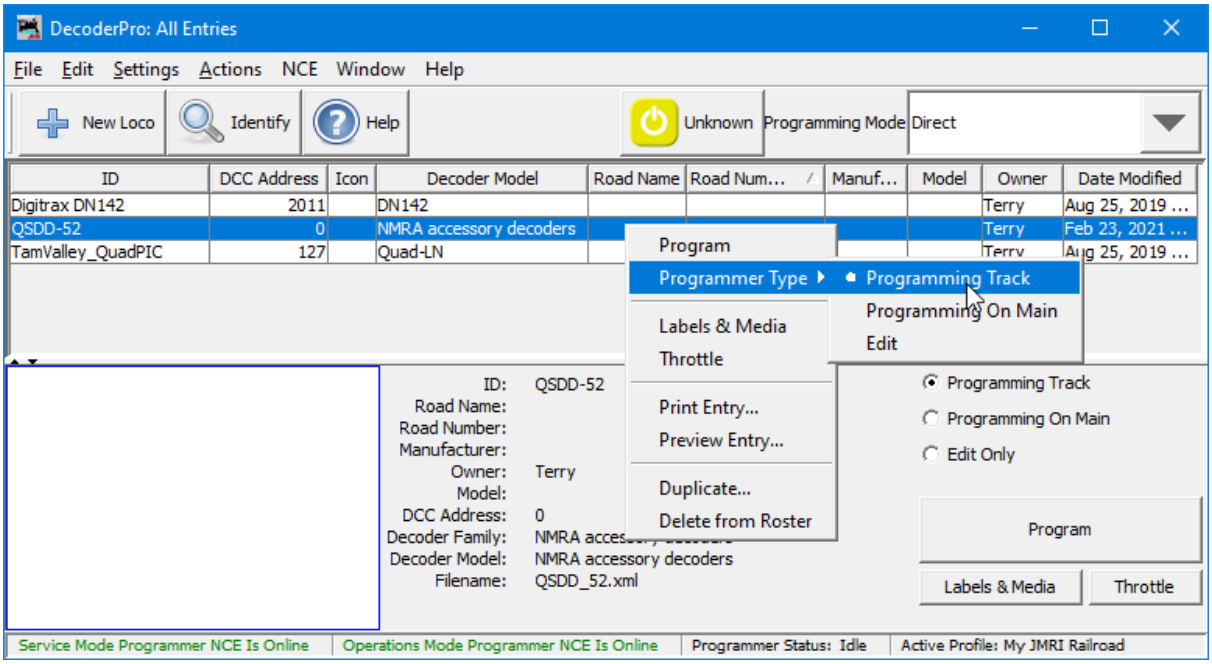
Setting up these other QSDDs, when they are transferred from the programming track to the layout, will consist only of small adjustments to the servo throws to compensate for differences between individual servos (and perhaps turnouts or linkages).

The screenshot below of the A-Track application shows the Configuration Variables window for the decoder with all of the relevant CVs set at their default values –

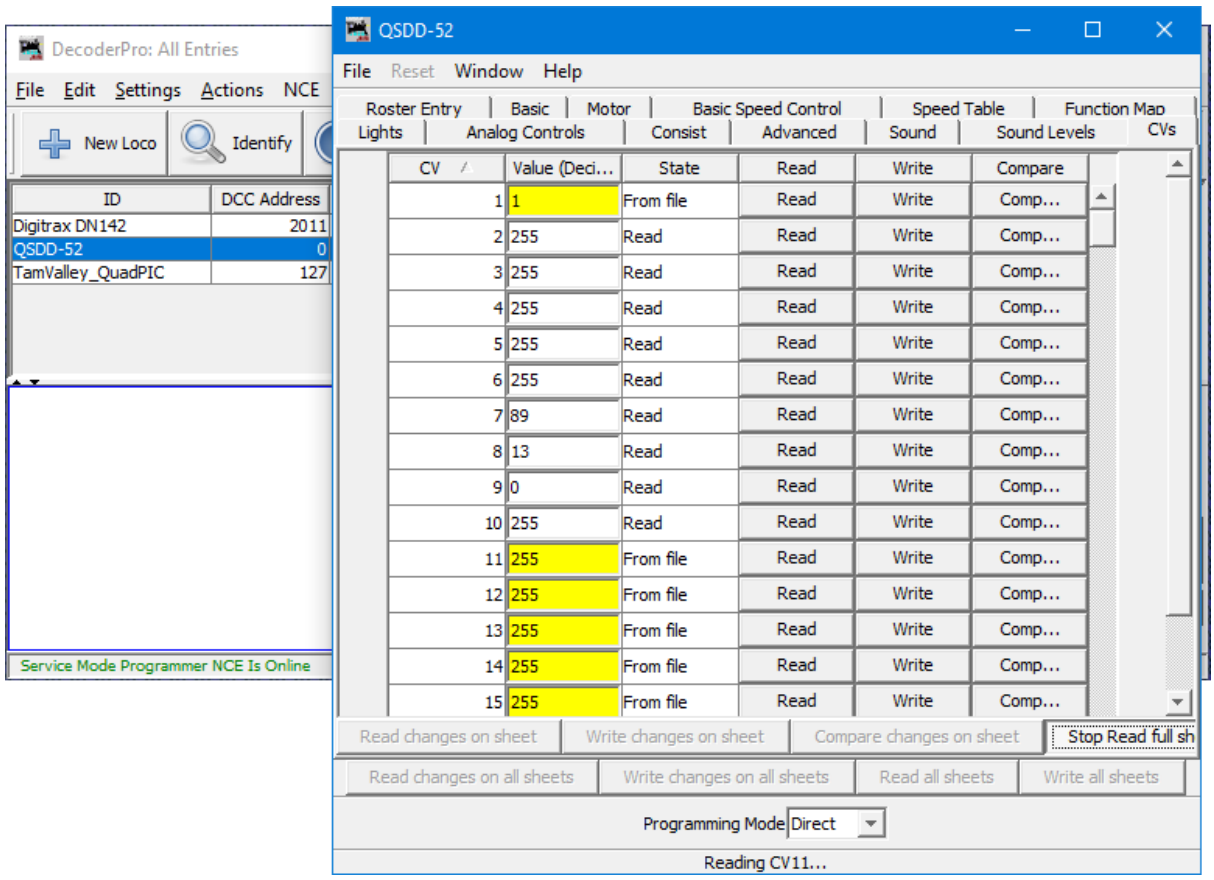


Adjustments to any of the CV values can be made simply by typing in a new value (or set of values, and then using the application's Program CVs function to transfer them to the connected decoder.

With Command Stations other than NCE, you can JMRI's Decoder Pro software (https://www.jmri.org/help/en/html/apps/DecoderPro/index.shtml) to similarly read and program QSDD CVs, and to save a copy to a file on your computer. Adding the QSDD to your roster in Decoder Pro is done by clicking New Loco, selecting NMRA as the manufacturer (at the top of the list) and then 'NMRA accessory decoders' as the decoder type. You can then proceed to access the QSDD on the programming track –



– and then click 'Program' to open the programming window, select the 'CVs' tab, and click 'Read full sheet' to start reading the QSDD Configuration Variables –



Technical Details

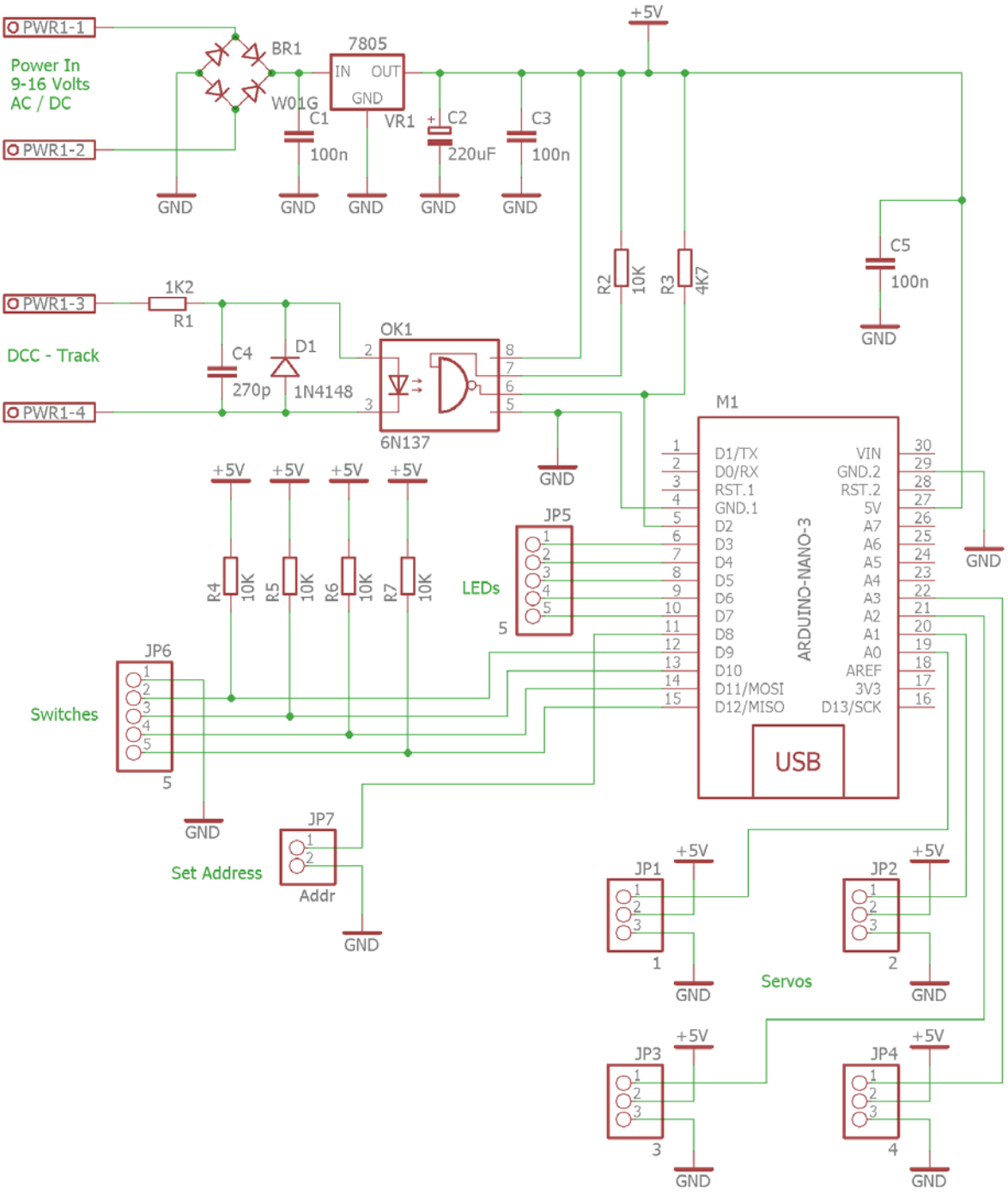
For anyone who is interested, the circuit schematic of the updated QSDD-2 decoder board is shown on the following page. The schematic of the keypad board is unchanged from that presented in Part 2 of the original article.

Power for the board from an external source (9 - 15 volts AC or DC) is rectified as necessary by diode bridge BR1 to supply the voltage regulator VR1 which, in turn, supplies +5 volts to the Arduino Nano, its associated circuitry, and the attached servos. Although the normal current through the regulator, with servos inactive, of about 45 mA keeps its power dissipation below 0.5 watts, this could rise briefly to as much as 10 watts if the maximum input voltage is being applied and all four servos are commanded to drive simultaneously. Hence the need for the heatsink attached to the regulator.

The DCC input circuitry is adapted from the design by Wolfgang Kuffer (https://mrrwa.org/dcc-decoder-interface/) and is used by Geoff Bunza as the basis for several of his projects.

The input DCC signal is connected to the input of optoisolator OK1 via resistor R1. Capacitor C4 filters out any high-voltage spikes from the track, and diode D1 prevents the optoisolator input diode from being fatally reverse-biased by the negative-going part of the DCC signal.

The output from optoisolator OK1 is a replica of the DCC waveform, but at a safe +5 volt level, so that DCC command packets can be input to digital input D2 of the Arduino Nano module. Here they are decoded by the NmraDcc library functions, and relevant accessory commands then passed to the QSDD sketch code.



Quad Servo DCC Decoder – Decoder

The remainder of the circuitry covers the various inputs to, and outputs from, the Arduino Nano.

The LEDs mounted on the keypad are driven from outputs D3 – D7 via connector JP5 on the decoder and connector JP1 on the keypad with resistors R1 – R5 on the keypad setting the current through each LED to approximately 13 mA.

Pins D9 – D12 are set as inputs which are connected to the four pushbuttons mounted on the keypad via connector JP6 on the decoder and connector JP2 on the keypad. The inputs are

normally pulled up to a HIGH level by resistors R4 – R7 on the decoder, but are taken to a LOW level (GND or 0 volts) whenever the appropriate pushbutton is pressed.

Pin D8 is similarly taken to a LOW level whenever the programming shorting link is fitted to header JP7. This pin uses an internal pull-up, rather than an additional external resistor, to maintain its level HIGH when the link is not fitted.

Finally, Pins A0 – A3 are each set as outputs to drive one of the four attached servos, connected via the three-pin headers, JP1 – JP4 on the decoder board.

