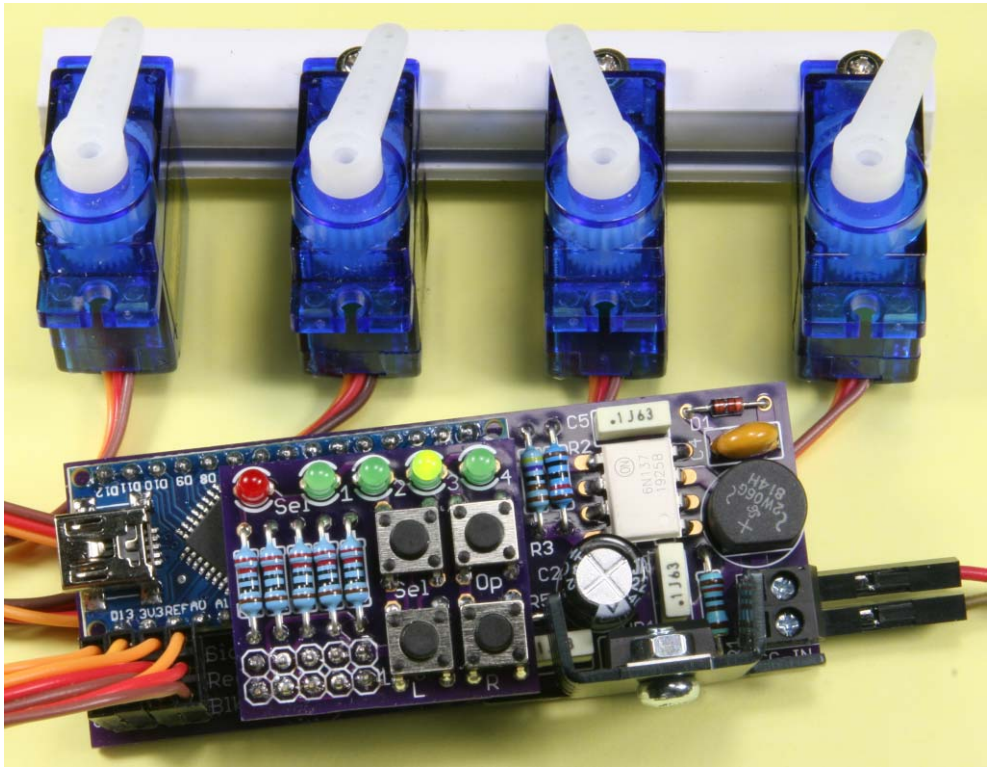
Build Your Own Quad Servo DCC Decoder – Part 2

Introduction

As presented in Part 1 of this article, the Quad Servo DCC Decoder (QSDD), based around a standard Arduino Nano module, is an accessory decoder that you can build yourself for less than $20, and which is capable of accurately controlling the position of up to four servos. If you followed the constructional details given in Part 1, and have now built your own QSDD and loaded its software ("sketch"), then this concluding part covers full details of how to set the throws and transit rate of each servo with precision, and to assign addresses to the servos so that they can be controlled through whichever DCC system you have. Finally, there are some of the deeper technical details of the QSDD for those readers who are interested.

For the remainder of the article, it is assumed that the two QSDD modules, the decoder and the keypad, are connected, either directly (as shown below) or via a ribbon cable, and that four servos are plugged into the appropriate headers.



All photographs by the author

The QSDD should be powered from your DCC system, by a connection to the track or directly to the command station from which it can also receive standard DCC accessory commands. Commands can be sent using either handheld controllers or suitable software, such as JMRI Panel Pro (http://jmri.org/help/en/html/apps/PanelPro/index.shtml) or my own A-Track application (https://www.a-train-systems.co.uk/atrack.htm), running on your computer.

The keypad can also be used to operate the servos by hand, using the fitted pushbuttons, either when plugged directly into the decoder or, more conveniently, via a ribbon cable up to 40 inches long. The latter allows you to see and set up your turnouts when the decoder and servos are out

of sight under the layout. Note that, once set up, if you choose to operate the servos only via DCC commands, then there is no requirement to have the keypad connected to the decoder.

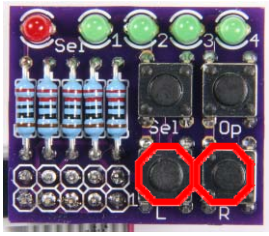
Servo Setup – Step by Step

Although a USB connection has no effect on the setup process itself, which can be carried out whether one is present or not, if you have the USB cable from the computer to the Arduino Nano connected, with the Arduino IDE running (whether it has the QSDD sketch loaded or not), then messages to confirm of all changes made during setup can be displayed on your computer screen as they occur. This can be quite helpful, especially when performing setup for the first time. Displaying messages just requires that the Serial Monitor is enabled in the Arduino IDE by clicking the relevant icon ( ) at the right-hand end of the toolbar, with the rate set at 115200 baud, and that DEBUG messages are left enabled in the sketch (which is how the code is supplied) .

Note : After the QSDD sketch has been loaded, any time the USB cable is plugged into the Arduino Nano you will see the red **Sel** LED and one or more of the green LEDs flash on and off a few times as the USB connection is fully established, depending on the version of the Arduino IDE installed on your computer. Once the USB link is stable, the decoder board will be reset automatically, lighting, then extinguishing, all of the LEDs once again, and moving the servos to their last defined position.

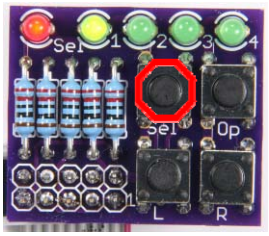
Step 1

The first step is to find the centre point of each servo's movement. Standard micro servos such as the SG90 have a nominal range of 180 degrees, from full right at 0° to full left at 180° although, because they are built to fairly wide tolerances, the range can vary by up to 15° either way.



Assuming you have the decoder and keypad boards plugged together, powered, and with the software loaded, press both **L** and **R** pushbuttons on the keypad together, then release them. All servos should then move to their centre positions (90°). You can then adjust the fitment of the servo actuating arms, if necessary, so that they align with whatever position you want as the centre of each servo's range.

Step 2



Press and release the Select (**Sel**) pushbutton. The red **Sel** LED will light, together with green LED **1**, and servo 1 will move to its current left limit position. You can now adjust the left limit position leftwards, one degree at a time, by pressing the Left (**L**) pushbutton, or rightwards by pressing the Right (**R**) pushbutton. Hold down either **L** or **R** pushbuttons until you reach the left limit position required by the linkage to the turnout that you intend to use.

Step 3

Press and release the **Sel** pushbutton again. The red **Sel** and green **1** LEDs remain lit, but servo 1 will now move to its current right limit position which can be adjusted, as in Step 2 above, by pressing either **L** or **R** pushbuttons.

Step 4

Press and release the **Sel** pushbutton. The red **Sel** and green LED **1** remain lit, and servo 1 will now move at its set rate to its left limit position, then return at the same rate to its right limit position. If the transit rate is either too fast or too slow for your requirements, then press the **L** pushbutton to make the rate slower (Leisurely), or the **R** pushbutton to speed the rate up (Rapid). After each pushbutton press, the servo will move from right to left and back again at the new rate. Note that the decoder will not respond to pushbutton presses while a servo is moving, but that you can keep either pushbutton depressed continuously, and watch the transit

rate for servo 1 either speed up or slow down until it reaches the rate you want. Note that, when either the slowest or fastest rate has been reached, movement of the servo will stop, and the decoder will not respond to further presses of whichever button caused it to reach its lower or upper rate limit.

Step 5

Press and release the **Sel** pushbutton. This time the red **Sel** LED will blink at a steady rate, with green LED **1** lit. You now have the option of saving all the position and rate adjustments you have made as configuration variable (CV) values, replacing the original default settings. These saved CV values will be held permanently in the Arduino Nano memory even when power is switched off.

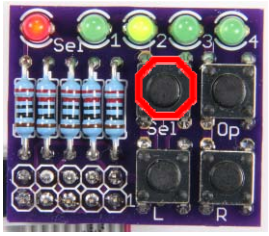
To save the new CV values for servo 1, press the **R** pushbutton (Retain the changes, Replacing the current CV values). Alternatively, if you have not made any changes to the servo 1 parameters, or you want to discard the adjustments you made, press the **L** pushbutton instead (Losing changes and Leaving the current CV values as they are).

After the changes are either saved or discarded, the green LED **1** will go out, although the red **Sel** LED will continue to blink.

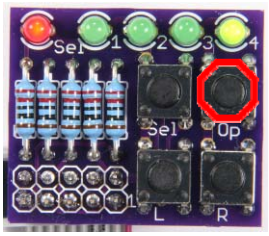
Note : You also have the option at Step 5 of pressing and releasing the **Sel** pushbutton instead of either the **L** or **R** pushbuttons. This will immediately take you to Step 6, as described below and discard any changes you may have made to the servo 1 parameters (like pressing the **L** pushbutton).

Step 6

Press and release the **Sel** pushbutton once again to complete the setup process for servo 1 (unless you skipped here directly from Step 5). The red **Sel** LED will stop blinking, while green LED **1** will go out and green LED **2** will light in its place. You can now repeat the actions of steps 2 through 6 to carry out the setup of Servos 2, 3, and 4. After you have completed the setup of servo 4, pressing and releasing the **Sel** pushbutton will terminate the setup process, and all of the LEDs will be extinguished.



If you wish to carry out the setup of individual servos, rather than of all four then, at either Step 2 or Step 6, press and release the **Op** pushbutton. This will skip the setup of the currently-selected servo, extinguishing the corresponding green LED, and lighting the green LED for the next servo. Pressing the **Op** pushbutton when servo 4 has been selected will terminate the complete setup process (which can then, of course, be restarted at servo 1 by pressing and releasing the **Sel** pushbutton).

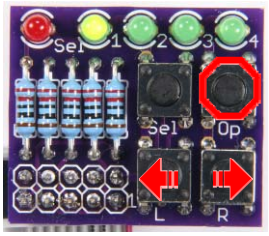


Once you have the movement range of each servo set to what you believe is required, you can fit the servos into whatever mounts and linkages you are using to connect to the layout turnouts. After everything is secured, repeat the servo setup process to fine-tune the position limits and transit rates.

Note : After a servo is driven to the required limit position, it is "*detached*", which means that it is no longer actively driven by the decoder and that only the friction in the servo geartrain is holding the linkage in place. You may, therefore, choose to move the limit positions out by a degree or two to ensure that the driven turnout points stay in position. The problem does not arise with Peco turnouts, for example, which have an over-centre spring to maintain the switched position.

A **short video** of the decoder setup sequence is available in the **Model Railroad Hobbyist** pages on YouTube – click this link to view.

Servo Operation – Manual



Correct operation of the connected servos can be checked by pressing the **Op** pushbutton on the keypad. This will light the green **1** LED to show that servo 1 is selected.

Press the **L** pushbutton to move servo 1 to its left limit position, at the set rate, or the **R** pushbutton to move servo 1 to its right limit position. No movement will result, of course, if the servo is already positioned at the selected limit.

Further presses of the **Op** pushbutton will light green LEDs **2**, **3**, and **4**, in turn, selecting the corresponding servos 2, 3, and 4, and detaching the previously-selected servo. When the relevant green LED is illuminated, the currently-selected servo can then be operated by using the **L** and **R** pushbuttons, as described.

A final press of the **Op** pushbutton will switch all LEDs off, with all servos detached.

Connecting the keypad to the decoder board using a 10-way ribbon cable with the appropriate connectors would allow the keypad to be mounted on a fascia and used for direct control of your turnouts, if that is how you wish to operate. A cable up to 40 inches (1 metre) long can be used.

Servo Operation – DCC Commands

When the decoder is connected to your layout track, you can operate any of the servos (and the turnouts to which they are linked) by issuing the appropriate command from your DCC system to whichever address has been assigned to that servo. Default addresses 1, 2, 3, and 4 are assigned as part of the pre-loaded software, but you are subsequently free to assign whatever addresses you wish to the four servos, as explained in the next section on setting decoder output addresses.

The key sequences to operate the servos, and their attached turnouts, are outlined below for some of the common DCC system handheld controllers –

NCE
Press ‘SELECT ACCY’
Type in one of the set addresses followed by ‘ENTER’
Press ‘1’ or ‘2’ to throw the turnout (depending on its current position)
If the turnout does not move, press ‘SELECT ACCY’ twice to throw it in the opposite direction.

Digitrax
Press ‘SWCH’
Type in one of the set addresses
Press ‘OPTN / t’ or ‘CLOC / c’ to throw the turnout (depending on its current position)

Lenz
Press ‘F’ then ‘5’ (LH100) or ‘Points/Signals’ (LH101
Type in one of the set addresses followed by ‘ENTER’ or ‘Points/Signals’
Press ‘+’ or ‘-’ (LH100) or ‘M’ (LH101) to throw the turnout (depending on its current position)

MRC
Press ‘ACCY’
Type in one of the set addresses followed by ‘ENTER’
Press ‘1’ or ‘2’ to throw the turnout (depending on its current position)

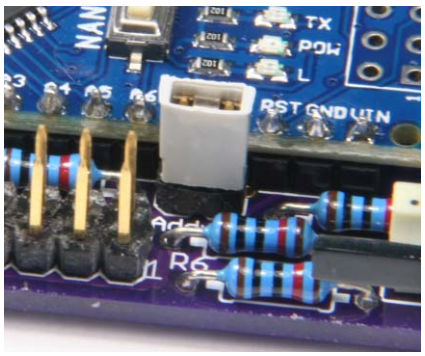
When DCC commands are used to operate the servos, it is not necessary to have the keypad fitted to the decoder board, nor a USB connection to the Arduino Nano module. The decoder board on its own provides all of the functions of a standard accessory decoder.

After each operation is completed, whether initiated manually or via DCC command, the current position of the servo involved is stored in the relevant configuration variable (CV59, 64, 69, or

74). The positions are retained when the decoder is powered off so that, when switched on again, the servos will be repositioned to where they were last commanded.

Setting Servo (Turnout) Addresses

While the four default addresses (1, 2, 3 and 4) may be adequate if your layout only has four turnouts, it will usually be necessary to assign different individual addresses to the decoder for each servo.



To start assigning addresses, fit a shorting link across the two pins of the programming header on the decoder board (labelled "Addr"). For the next stage, attachment of the keypad is optional, so we will assume that it has been removed in order to fit the shorting link and has been left disconnected.

From your DCC system prepare to send an accessory command to the decoder, as described in the preceding section, by selecting the Accessory function for your type of DCC system. Type in the address you wish to assign to servo 1, followed by the ENTER key if required, then throw the turnout. You can select either direction of throw since the command direction is ignored, and only the address used is recorded.

To be accepted, addresses must lie in the range of 1 to 2043, ie. two- or four-digit addresses are equally acceptable. Address values outside of this range will simply be ignored and will not replace whatever value is currently stored for the relevant servo (turnout) address.

Now select the Accessory function again on your handheld controller, enter the address to be assigned to servo 2, and proceed to throw the turnout. Repeat this sequence twice more to program the addresses required for servos 3 and 4.

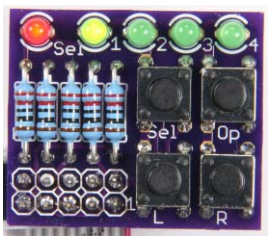
All four entered addresses will now be stored in configuration variables 41 through 48, with each address held in two consecutive CV locations, eg. servo 1 address in CV41 and CV42, servo 2 address in CV43 and CV44, etc.

Remove the shorting link from the two-pin programming header to terminate the setting of servo addresses. This action will reset the decoder and then allow it to operate normally. The shorting link can be stored on one of the programming header pins until the next time it is required.

Note : If you have a USB connection to the Arduino Nano from your PC, and the Arduino IDE Serial Monitor is enabled (as described earlier in the Servo Setup section) confirmation (DEBUG) messages for each servo address programming step will be displayed in the Serial Monitor window.

Monitoring and Manual Control of Setting Servo Addresses

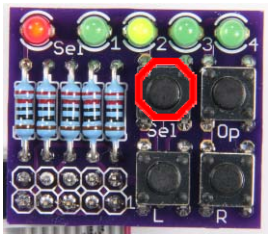
Connecting the keypad to the decoder board, most conveniently via a cable, when the shorting link is placed on the Addr header, offers you more flexibility in assigning addresses and also provides visual feedback through the keypad LEDs during the programming process.



With the shorting link in place, the red **Sel** LED and green LED **1** will be lit. Setting the address for servo 1, using your handheld controller as described previously, will cause the red **Sel** LED to blink as an acknowledgement that the address has been received, green LED **1** will go out, and green LED **2** will light. Subsequently, entering addresses for servos 2, 3, and 4 will repeat the sequence with green LEDs 2, 3, and 4.

After all four addresses have been received, they will be stored in CVs 41

through 48, all of the green LEDs will be extinguished, and the red **Sel** LED will blink continuously. In this state the decoder will not respond to any pushbutton presses nor to any DCC commands. Remove the shorting link to terminate programming, turning off the red **Sel** LED. This will reset the decoder and return it to normal operation.



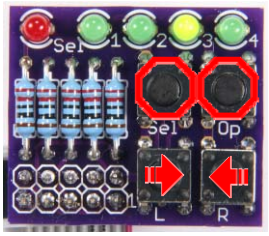
However, with the keypad connected and the Addr shorting link in place, you have the option of choosing which servo addresses to program. Starting with the red **Sel** and green **1** LEDs lit, pressing the Select (**Sel**) pushbutton will turn green LED **1** off and green LED **2** on, and skip entry of a new address for servo 1. You can now enter a new address for servo 2 or skip this also by pressing the **Sel** pushbutton again – which will turn green LED **3** on in place of green LED **2** and offer you the opportunity of programming a new address for servo 3.

When you reach the option of programming an address for servo 4, entering a new address (or pressing **Sel** to skip programming) will turn off all green LEDs, store any entered addresses in the relevant CVs, and leave the red **Sel** LED blinking continuously. As stated before, in this state the decoder will not respond to any pushbutton presses nor to any DCC commands. Remove the shorting link to terminate programming, turning off the red **Sel** LED, resetting the decoder and returning it to normal operation. Store the shorting link on one of the programming header pins until the next time it is required.

Note : As mentioned before, if you have a USB connection to the Arduino Nano from your PC, confirmation (DEBUG) messages for each servo address programming step will be displayed in the Serial Monitor window.

Reversing Servo Direction

After mounting the servos to the turnouts on your layout, you may find that you need some of them to operate in the opposite direction, so that generating a Route command from your DCC system switches the turnout to the Route (diverging) direction rather than setting it in the Normal (straight) direction.



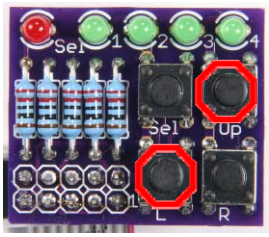
To change a servo operating direction, connect the keypad to the decoder and press the **Op** pushbutton one or more times to light the green LED corresponding to the servo to be reversed.

Now press and release the **Sel** pushbutton **once**. The red **Sel** LED will light briefly to acknowledge that the required change has been saved to the relevant configuration variable (CV56, 61, 66, or 71), after which you will find that pressing the **L** pushbutton will move the selected servo to the right, and the **R** pushbutton moves it left. The servo will react similarly to received DCC commands from your system.

Complete the process by pressing the **Op** pushbutton as many more times as required to extinguish all of the green LEDs.

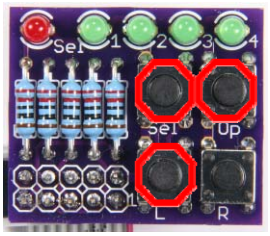
Exactly the same sequence of actions will return the servo to normal rather than reverse operation.

Resetting the Decoder



As mentioned earlier, towards the end of the “Software for the QSDD” section in Part 1 of this article, to reset the decoder and restart the uploaded software, you would normally press the reset button on the Arduino Nano. However, with the keypad plugged directly on to the decoder board, the reset button is not easily accessible, so, as an alternative, just press and hold down the **L** pushbutton on the keypad. Now press the **Op**

pushbutton, then release both buttons to reset the Arduino Nano. All of the LEDs should light and then go out, and all of the connected servos should move to (or remain at) their last commanded position.



You can also, if you wish, restore all of the decoder parameters to their default values (see the list of configuration variables (CVs) given in Part 2 of this article). With the keypad plugged into, or connected to, the decoder board, press and hold down the **L** pushbutton on the keypad. Now press and hold down the **Op** pushbutton then, with both buttons held down (and a little manual dexterity), press and release the **Sel** pushbutton. Finally, release both **L** and **Op** pushbuttons.

As with the normal Reset, all of the LEDs should light and then go out. However, this time, the red **Sel** LED will light briefly again as the CV default values are loaded, followed by all of the connected servos moving clockwise to their default rightmost position (20 degrees right of centre).

Take care, if the servos are linked to a layout turnout, that this movement can be accommodated and will not cause damage to the linkage or to the servo, bearing in mind that the plastic geartrain fitted in most servos is easily damaged if the servo is driven against an inflexible endstop.

Configuration Variables and DCC Operations

Like all NMRA-compliant decoders, the QSDD holds its working data in a set of configuration variables (CVs) consisting of –

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 - Slow Rate (1 to 16)
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 - Slow Rate (1 to 16)
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 - Slow Rate (1 to 16)
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 - Slow Rate (1 to 16)
71	1	Direction - 1 = Normal Operation, 0 = Reverse Operation
72	70	Right Limit
73	110	Left Limit
74	70	Current Position

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 AD = "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).

If connected to a programming track, the QSDD, unlike the majority of commercial accessory decoders, allows you to read the current values of all CVs and then write (program) new values if required. When you do this, you need to ensure that the keypad is attached to the decoder since the necessary acknowledgement back to the DCC Command Station (a pulse of current) is generated by briefly switching on all five of the keypad LEDs together.

Reading and writing of CVs values via a programming track (Service Mode) can be done using a handheld controller connected to all major types of DCC system but (unless someone can tell me differently) not from JMRI Decoder Pro which simply decides that the decoder is not a locomotive and refuses to proceed further.

However, 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.

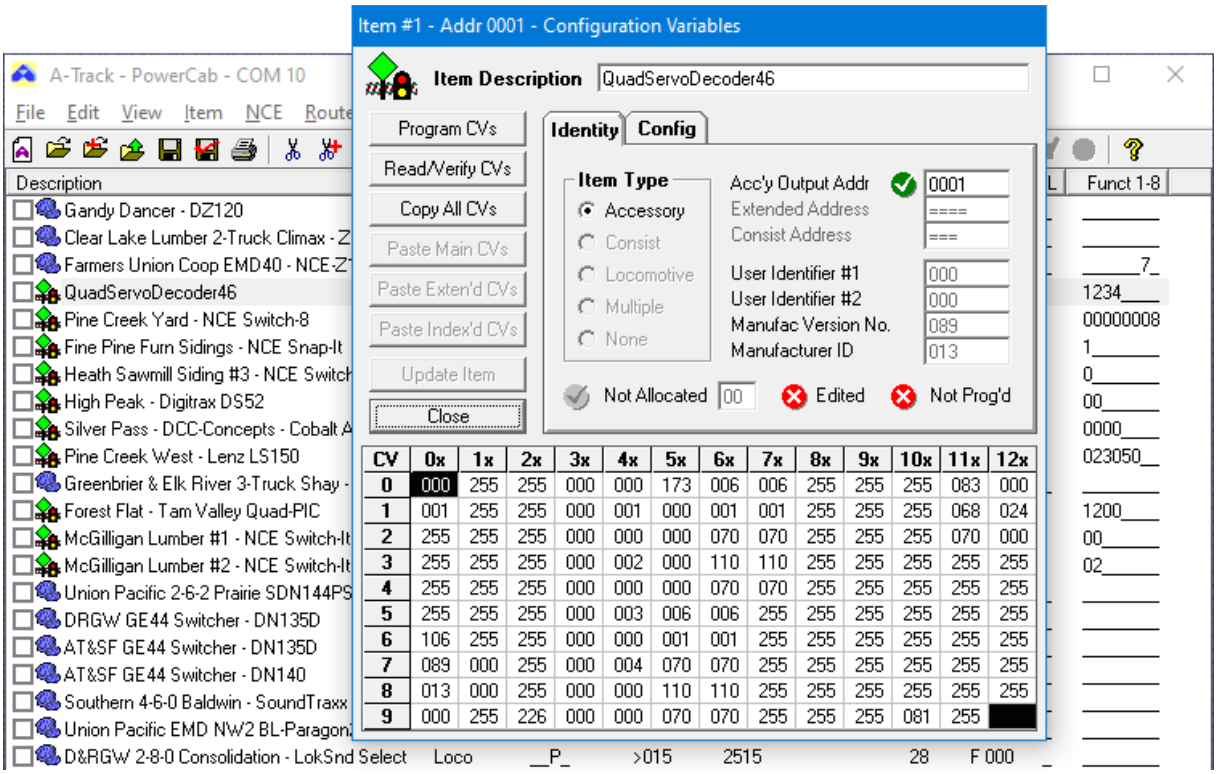
Note : While an NCE Power Cab system will happily program the QSDD in any mode, the older NCE Power Pro system has much less tolerant programming hardware. If you wish to use a Power Pro to read or write QSDD CVs, then you either need to connect to the QSDD via a programming booster such as the SoundTraxx PTB100, or have the Arduino Nano module powered from a computer USB port while programming is taking place. If using Direct programming mode from the Power Pro results in too many errors (misread CV values) then Paged mode is recommended.

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.



CV	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	10x	11x	12x
0	000	255	255	000	000	173	006	006	255	255	255	083	000
1	001	255	255	000	001	000	001	001	255	255	255	068	024
2	255	255	255	000	000	000	070	070	255	255	255	070	000
3	255	255	255	000	002	000	110	110	255	255	255	255	255
4	255	255	255	000	000	000	070	070	255	255	255	255	255
5	255	255	255	000	003	006	006	255	255	255	255	255	255
6	106	255	255	000	000	001	001	255	255	255	255	255	255
7	089	000	255	000	004	070	070	255	255	255	255	255	255
8	013	000	255	000	000	110	110	255	255	255	255	255	255
9	000	255	226	000	000	070	070	255	255	255	081	255	

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 function to transfer them to the connected decoder.

Technical Details

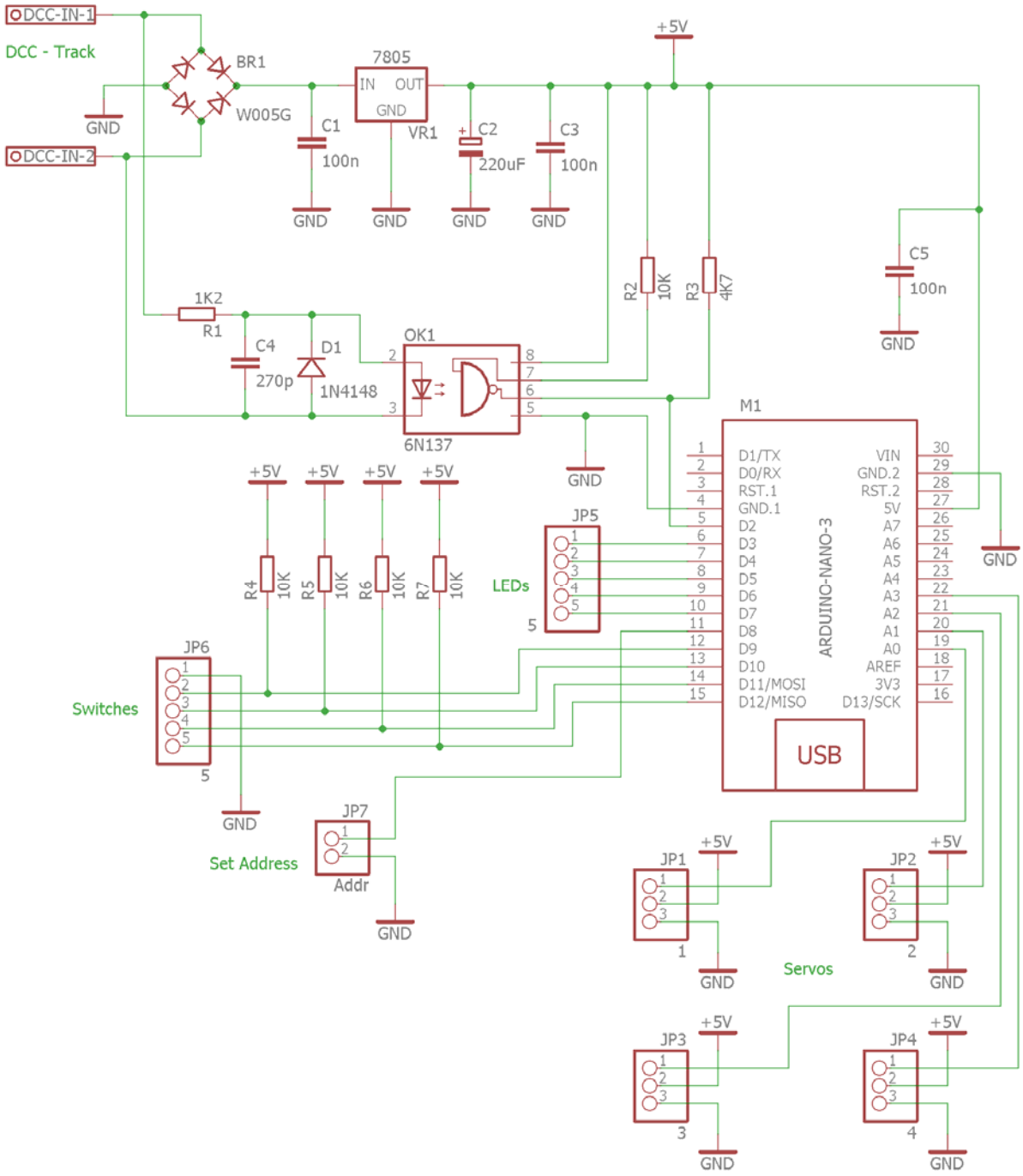
For anyone who is interested, the circuit schematics of the decoder and keypad boards are shown on the following pages.

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 DCC signal from the layout track (normally 14 - 16 volts AC) is rectified by diode bridge BR1 to supply around 14 volts DC to 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 all four servos are commanded to drive simultaneously. Hence the need for the heatsink attached to the regulator.

The input DCC signal is also 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

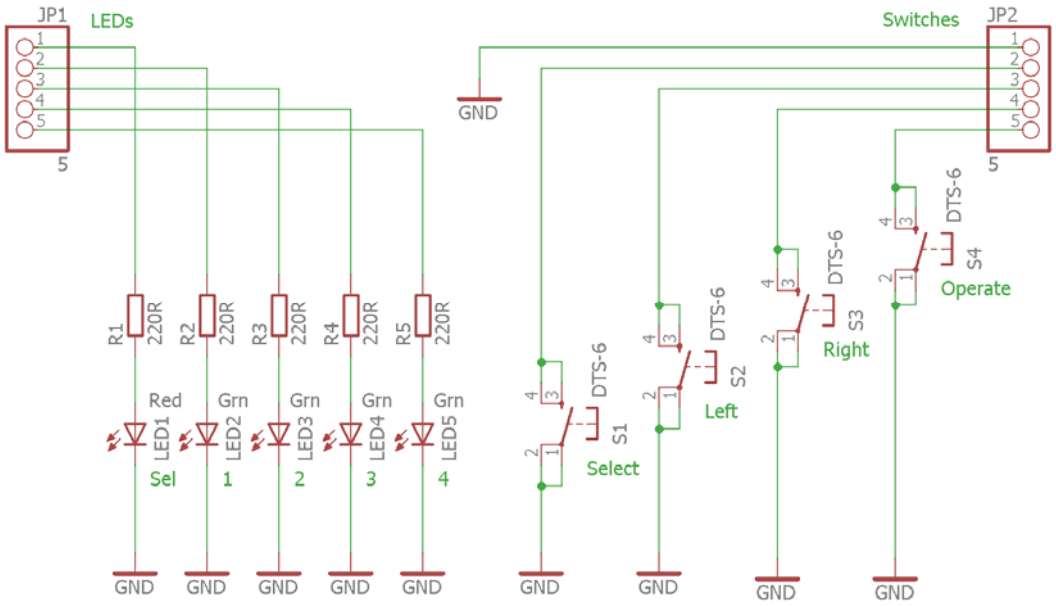
The remainder of the circuitry covers the various inputs to, and outputs from, the Arduino Nano.

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 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.



Quad Servo DCC Decoder – Keypad

Dr Terry Chamberlain

Terry Chamberlain got into model railroading almost by accident in the 1990s when he responded to a request from some modellers in California to build a DCC system based around an Atari personal computer – and he had to build a simple layout to prove that it all worked. Eventually the project evolved into A-Track, a Windows application to provide full computer support for the complete range of NCE DCC systems, with facilities similar to JMRI's Decoder Pro and Panel Pro.

Terry is a professional electronics engineer and spent most of his career in the UK defence industry designing, and managing the development of, large real-time computer systems for the Royal Navy. Now that he has retired he is still hoping to build the logging and mining layout he has been planning for years (after several visits to Colorado) – but keeps getting distracted by new computer and electronics projects for model railroading.