**CANCMDDC – A multi-channel CBUS-connected DC Controller**

My intention for my layout at home is to equip all the locos with DCC Decoders, and run them from a CANCMD station, maybe with a booster unit. As I really want to install sound decoders, this project will take quite a while to accomplish, so in the meantime they will have to run from an analogue DC controller.

I saw the video by Mark Riddoch M1118 where he describes his innovative idea for an Arduino-based DC controller (<https://www.youtube.com/watch?v=XsZtespnIKw&feature=youtu.be>), and the follow-up articles in the Forum by Ian Morgan M2775 in the forum (<https://www.merg.org.uk/forum/viewtopic.php?t=6376>) and was suitably intrigued.

The cost of components for such a device, when sourced from China via that well-known auction site, is around £20 – so perfectly do-able as a disposable interim solution.

I obtained a ‘kit’ of parts, and laid them out on the workbench to assess the size of case which would be required to house the assembly. I know that Ian used some very nice cases, but they look expensive, so an alternative was found. This is a clear plastic box, in which the treats the Ambassador spoils us with were supplied. Placing the components in this enclosure would force me into making a neat job of the wiring!

Drilling the mounting holes, etc., in the case was a bit of an adventure, as the plastic easily melts and binds to the drill – a slow speed, sharp drill and patience are all required.

Wiring the units together offered no real issues, and a colour-coding scheme for the various wires ensured that no mistakes were made, and future changes (if any) should be straightforward. A spiral book-binder was used as a distribution bus within the case for the longer wires, to keep them under control.

So far, hardware-wise, everything is almost exactly as Ian suggested and used. He did advise me that he wasn’t overly impressed with the LCD display he used, so I opted for an alternative. This is a 4-line x 20-character display, with built-in font generation. Any form of graphics display is therefore out of the question, apart from the fact that there are 8 user-definable characters stored in RAM available.

I used an Excel spreadsheet to design a couple of screens, using just the 8 characters plus a few of the built-in ones, to create a MERG logo, a STOP logo (emergency stop all), and some bar-graphs for the speed thermometer bars.

The other hardware change I employed was the addition of a 4 x 3 keypad. I added this primarily for testing when the unit isn’t connected to the CAN bus, but it can also be used for ‘local’ control when connected to the bus.

Having assembled all the components, it was time to get to grips with the software. Fortunately, both Mark and Ian had sprinkled copious comments throughout the code, making it easy to follow, despite never having written Arduino nor C/C++ code before (C#, Java and assembler, yes).

The new hardware is more-or-less supported by some freely-available libraries, again making inclusion of these items quite straightforward.

Most of the changes to the software were around the need to utilise the keypad to control locos. It is possible to attempt to change the speed of a loco which is already under control from an attached CAB – thus there is a need to support the Steal/Share philosophy to handle this situation.

The various ‘modes’ that the keypad can be in (select loco, change speed, etc.) are easily managed by the employment of a Finite State Machine (FSM), a construct with a small number of well-defined states, and, more importantly, well-defined transitions between those states. There is also a timeout facility, such that if a key is pressed, then no further action is taken, the key press is ignored and the display reverts to the normal speed display.

The power overload detector board was modified to bring the shutdown indicator out, and this is connected to the Arduino, so that it can alert the user - and connected CABs – that the power is currently switched off.

Various new CAN messages are now created and responded to, including RTR, TON, TOFF & DSPD.

Further improvements to the CAN support include detecting transmission fail when no other CAN modules are attached to the bus (this is the same as the CANCMD does), and instigating a self-enum when another module with the same CanId is seen to be transmitting on the bus.

There were a few spare digital input pins otherwise unused, so three of these were allocated as active-low address select inputs. This allows for there to be eight CANCMDDC units on one CAN segment. As each unit can control up to eight locos (tracks, really), a total of 64 individual track sections can be independently controlled – surely enough for anyone?

Whilst working on the code, some tidying-up was performed (but not much was needed!), principally reducing the data-types used in some places to reduce the memory consumption, especially of the RAM of which about 55% of the 8K available is currently used.

The program size itself is a meagre 10% of the available space. This leaves a great deal of space for any future developments. Such developments could include:

* The use of Node Variables (NVs) to control operation at run-time
* Responding to, or generating, CBUS events
* Provision of automation, such as shuttles, sequences, etc.
* Inclusion of a Wi-Fi access-point to support wireless throttles.
* Many more things I haven’t even thought of….

The current code is on the software wiki page.

Acknowledgements:

I am indebted to Mark Riddoch and Ian Morgan for their original design work, and the version number screen states this.

My thanks to Peter Brownlow for his continued assistance whilst working on the software, in order to understand speed-steps, can ids and self-enumeration, amongst other things.