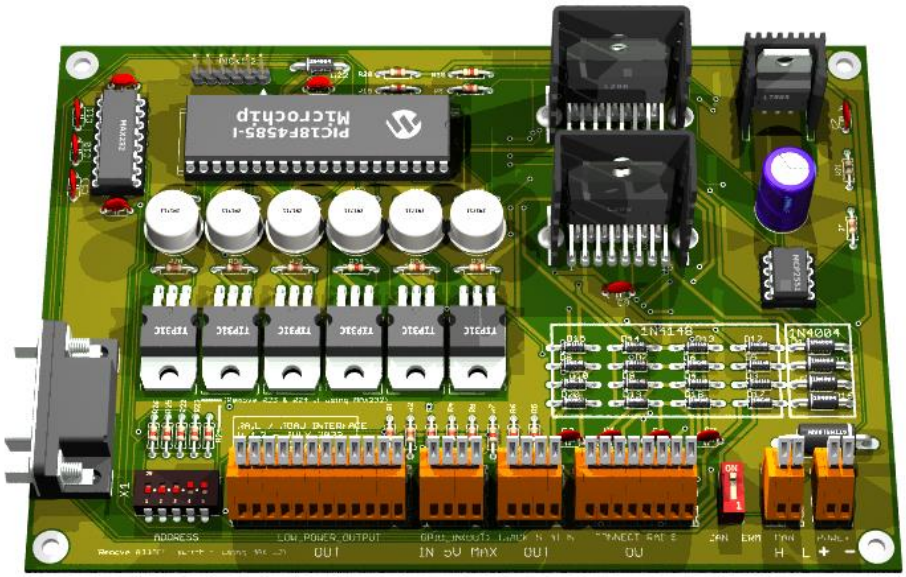
Reference Guide

RAIL / ROAD INTERFACE

V 1.2 (July 2023)



Written by: Philippe Cavenel
17 July 2023
Last update July, 17 2025
Documentation Version 1.8

Content

Illustrations

1 Preamble

1.1 History

History	Change	Date
Version 1.0	Creation	18/May/2021
Version 1.1	New PCB design	18/May/2023
Version 1.2	Replace H-bridge transistors with L298s	22/July/2023

1.2 Abbreviations

DCC: Digital Command Control (defined by a NMRA standard)
NMRA: National Model Railroad Association
AMS: Auto Motor Sport (Faller)
HO: Half-O (model railway scale corresponding to 1:87)
DIY: Do It Yourself
PC: Personal Computer
USB: Universal Serial Bus
FTDI: Future Technology Devices International
CAN (bus): Controller Area Network
TTL: Transistor-Transistor logic
GPI : Global Purpose Input-Output
LED: Light-Emitting Diode
PWM: Pulse Width Modulation
EMC: ElectroMagnetic Compatibility
STL: STereo-Lithography
STEP: STandard for the Exchange of Product model data
DRC: Design Rule Check
ERC: Electrical Rule Check

1.3 Typographic convention

The file names are in `Courier New` format, for example `RailDriverCamJob.cam` file

The online order examples or file contents are in bold **`Courier New`** on a grey background, for example

```
stl2pov.exe myModel.stl > myModel.inc.
```

Warning is in bold red inside a frame

Binary names are in bolt like for example **stl2pov.exe**

2 Disclaimer

The information contained in this document has been obtained from sources believed to be reliable. However, it may contain technical inaccuracies or typographical errors.

The author reserves the right to correct such errors as soon as they are brought to his attention. We strongly recommend that you check the accuracy and relevance of the information provided in this document.

The information contained in this document is subject to change at any time, and may have been updated. In particular, it may have been updated between the time it was downloaded and the time the user takes cognizance of it.

Use of the information contained in this document is at the user's sole risk, and the user assumes full responsibility for any consequences arising therefrom, without any liability or recourse against the author.

In no event shall the author be liable for any damages whatsoever resulting from the interpretation or use of the information contained herein.

3 Introduction

3.1 Document purpose

Why this document?

Faller AMS car networks do not have a DCC-type control mode (NMRA standard). This makes it difficult to have several vehicles on the same network. Using diodes, it is possible to operate a maximum of 2 vehicles per track, which is not very many. By creating track sections and vehicle detections per track section, you can have many more vehicles on the track. On the other hand, the realism of these networks is not respected with operating speeds that are too high, and by using PWM-type pulsed current mechanism, a much more realistic effect can be achieved.

As soon as this type of board is developed, it seems interesting to have a universal programmable interface that can also be used to drive a railway network, and thus support DCC mode (NMRA).

Document content

This document describes the development of a DIY Faller AMS network board for HO rail and road models and the various associated software (Firmware for PIC18F4585 and PC software) developed since 2021.

The board is designed to communicate with a PC via an RS232 serial link (or via a USB connection using an FTDI cable connected to the RS232 connector, available at Amazon: DSD TECH SH-RS232G USB to DB9 Female serial cable Integrated FTDI FT232RL chip) and with each other via a CAN bus.

Each board can drive up to 4 sections in analog or digital DCC mode (NMRA standard) with electric vehicle presence detection, and 6 low-power controls for lighting or small mechanisms (such as turnout controls). Finally, 8 TTL inputs/outputs (GPIOs) 4 are dedicated to vehicle presence detection, and 4 enable the addition of detectors, control LEDs, or the linking of boards to generate automatic control mechanisms (blocking).

Boards cannot individually mix DCC and analog modes, but it is possible to run one board in DCC mode and another in analog mode.

Each board has a 5-bit address (so 31 can be used, i.e. 128 slots for the whole network). The PC interface board called the “master” is set to address 31 by default. Each other boards should have at least the bit 0 or 1 or 2 set to 0.

Warning

The current 2023 version at the time of writing is V1.2, and this document describes this version.
This board was developed for personal usage and on an old but readily available and quickly mastered environment that can simply be replaced by a tool like KiCad EDA or Altium on more recent computers.

All technical indications (such as layer numbers in the Eagle tool) therefore refer to these tools (see Technical References below). To simplify use of this documentation, a zip file containing all developments is available for download.

Finally, the development of a professional version would require the use of SMD components, a minimum EMC and countries compliance study, integration into a great casing, cost analysis, user manual, and targeting of customers for a possible market launch. This is not the purpose of this documentation. This development should be considered as a simple prototype for personal usage.

3.2 Technical References

In order to reduce development costs and simplify board assembly, the entire board is mounted using radial components, and the design was conceived on an available professional version 6.6.0 of Eagle, MPLAB 8.92 and POV-Ray 3.7 running Mac OS High Sierra on a MacBook Pro in early 2011, 2.7 GHz Intel Core I7 with 16 GB DDR3 1333 MHz memory. This computer runs Mac OS and Windows on Parallel desktop V16.5.1. Development environment needs tools on both operating system Mac OS and Windows. Pending the development of a bootloader for the master and slave boards, the firmware is updated using a PICkit™ 2 programmer/debugger (PG164120).

Eagle 6.6.0

EAGLE contains a schematic editor, for designing circuit diagrams. Schematics are stored in files with `.SCH` extension, parts are defined in device libraries with `.LBR` extension. Parts can be placed on many sheets and connected together through ports.

The PCB layout editor stores board files with the extension `.BRD`. It allows back-annotation to the schematic and auto-routing to automatically connect traces based on the connections defined in the schematic.

EAGLE saves Gerber and PostScript layout files as well as Excellon and Sieb & Meyer drill files. These are standard file formats accepted by PCB fabrication companies, but given EAGLE's typical user base of small design firms and hobbyists, many PCB fabricators and assembly shops also accept EAGLE board files (with extension `.BRD`) directly to export optimized production files and pick-and-place data themselves.

EAGLE provides a multi-window graphical user interface and menu system for editing, project management and to customize the interface and design parameters. The system can be controlled via mouse, keyboard hotkeys or by entering specific commands at an embedded command line. Keyboard hotkeys can be user defined. Multiple repeating commands can be combined into script files (with file extension `.SCR`). It is also possible to explore design files utilizing an EAGLE-specific object-oriented programming language (with extension `.ULP`).
(source Wikipedia)

POV-Ray 3.7

POV-Ray (Persistence of Vision Raytracer), or POV, is a free raytracing software available on a wide variety of platforms (Windows, Mac OS, GNU/Linux, etc.). It was originally based on DKBTrace sources, and to a lesser extent on Polyray.

POV-Ray does not have an integrated graphical interface (3D modeler) like most current synthesis software, but uses scene description scripts, in which all objects, lights, etc. must be described.

Modelers dedicated solely to POV-Ray exist (KPovModeler, Moray, Yet another POV-Ray modeller...), while many others export to the POV-Ray file format. Its file format is ASCII, and the default file extension is "`.pov`".

This provides basic shapes (spheres, boxes, toroids, etc.) on which Boolean operations can be performed using CSG. It also makes it possible to create volumes or surfaces based on mathematical functions, such as isosurfaces.

```
Example: function {x*x - F/y*y + z*z} draws a kind of gravity well, with F representing
its force.
```

It is also possible to import objects from other software (such as 3D Studio Max, Poser, etc.), which will be rendered in POV-Ray as an assembly of triangles, as many software programs are compatible, but it is very difficult to export POV-Ray objects to other formats.

POV-Ray can also be used to create animations.
(source Wikipedia)

MPLAB 8.92

MPLAB is a proprietary freeware integrated development environment for the development of embedded applications on PIC microcontrollers, and is developed by Microchip Technology.

MPLAB X is the latest edition of MPLAB, and is developed on the NetBeans platform. MPLAB and MPLAB X support project management, code editing, debugging and programming of Microchip 8-bit PIC and AVR (including ATMEGA) microcontrollers, 16-bit PIC24 microcontrollers, as well as 32-bit SAM (ARM) and PIC32 (MIPS) microcontrollers.

MPLAB is designed to work with MPLAB-certified devices such as the MPLAB ICD 3 and MPLAB REAL ICE, for programming and debugging PIC microcontrollers using a personal computer. PICKit programmers are also supported by MPLAB.

MPLAB X supports automatic code generation with the MPLAB Code Configurator and the MPLAB Harmony Configurator plugins.
(source Wikipedia)

PICKIT 2

The PICkit™ 2 programmer/debugger (PG164120) is a low-cost development tool with a comprehensive interface for programming and debugging microchips or microcontrollers.

Its comprehensive Windows® programming interface supports basic families (PIC10F, PIC12F5xx, PIC16F5xx), mid-range families (PIC12F6xx, PIC16F), controller families **PIC18F**, PIC24, dsPIC30, dsPIC33, and PIC32, 8-bit, 16-bit and 32-bit, and a large number of EEPROM-type serial microchips.

With the powerful Integrated Development Environment (IDE), PICkit™ 2 lets you debug on-circuit directly, on most PIC® microcontrollers. On-circuit debugging runs, pauses, and executes the program step by step, while the microcontroller is integrated into the application. When paused on a breakpoint, file registers can be examined and modified.
(source https://pickit2.software.informer.com)

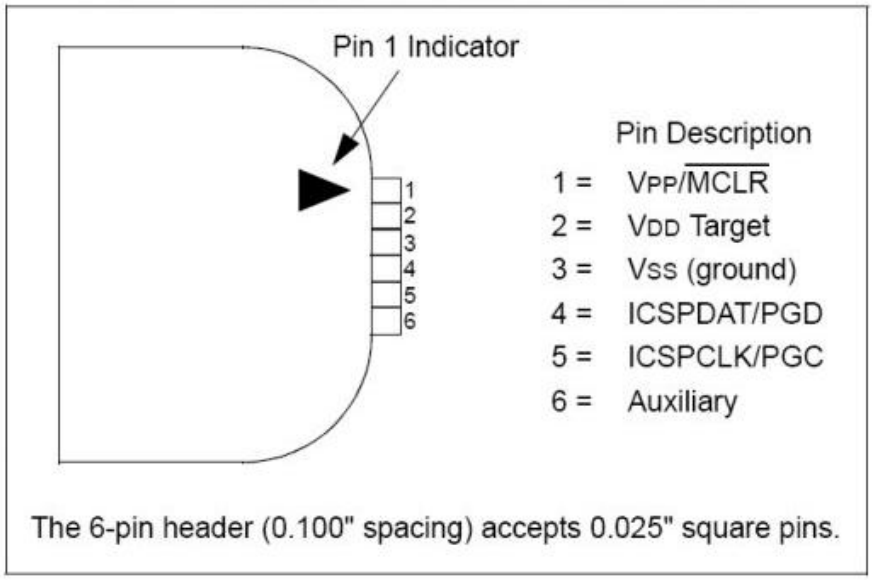


Figure 1 PICkit 2 Programmer Connector pinout

4 Electronic design

4.1 Global architecture

The complete system consists of a PC running a rail or road network control application. This may be a fully developed application, or an existing one with a programmable interface driver. The interface between this PC and the control boards is via a 115200 Baud serial bus connected to a board with an RS232 serial interface. Only one of these boards, known as the “Master” board, has this interface.

The master board communicates with the other boards, known as slaves, via a 500K Baud CAN bus. The master board contains an RS232 transmitter (MAX232). By default, this board has address 31 (the address selector on this board must be removed, as well as resistors R23 and R24, giving it the value 31 by hardware setting). Address 31 is therefore forbidden on other boards, as it indicates to the PIC18F the presence of an RS232 transmitter.

If the user inadvertently sets address 31 on a board that does not contain an RS232 transmitter module, the PIC18F will begin by testing the presence on the TX line (RC6/TX/CK, pin 25) of either level 0 (switch bit 4 to 0) or level 1 (switch bit 4 to 1). If either of these levels is detected, address 31 has been selected by mistake, and the PIC18F goes into standby mode until the address is changed.

If two cards have the same address, the overall behavior of the system may be inconsistent, without compromising network security.

Each board is supplied with a power supply between 12 and 24V DC. The optimum supply voltage depends on the type of equipment operating on the network. The maximum theoretical voltage supported is 32V, but such a voltage could destroy the equipment connected to the board. In practice, we advise you not to exceed a supply voltage of 20V especially if you use DCC environment.

Each board can drive 4 PWM or DCC blocks and 6 low-power modules. The boards also feature 8 TTL-level inputs/outputs (not to be used for direct control of a power module).

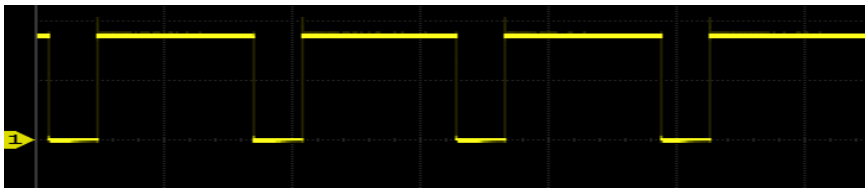


Figure 2 PWM Mode

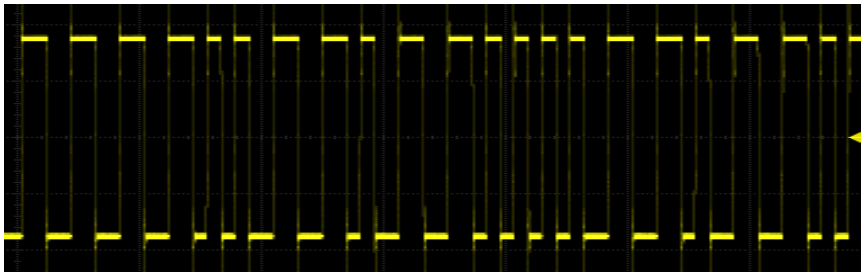


Figure 3 DCC NMRA mode

The system is controlled using a command language sent to the master board via the RS232 serial link. This language has two modes: a programming mode and a control mode.

- In programming mode, it is possible to specify the direction of each board's GPIOS, and to trigger automatic actions on events, e.g. "on detection of a vehicle on block 3 of board 8, feed block 6 of board 5 in the forward direction at speed 10" (speeds are between 0 and 15).

- In control mode, commands can be given, e.g. "feed block 3 of board 2 in the reverse direction at speed 2".

Each event (detection or loss of presence of a vehicle or change of status on a GPIO or TIMER triggered) generates an event which is sent to the other boards on CAN bus.

The commands and language are described in the following chapters.

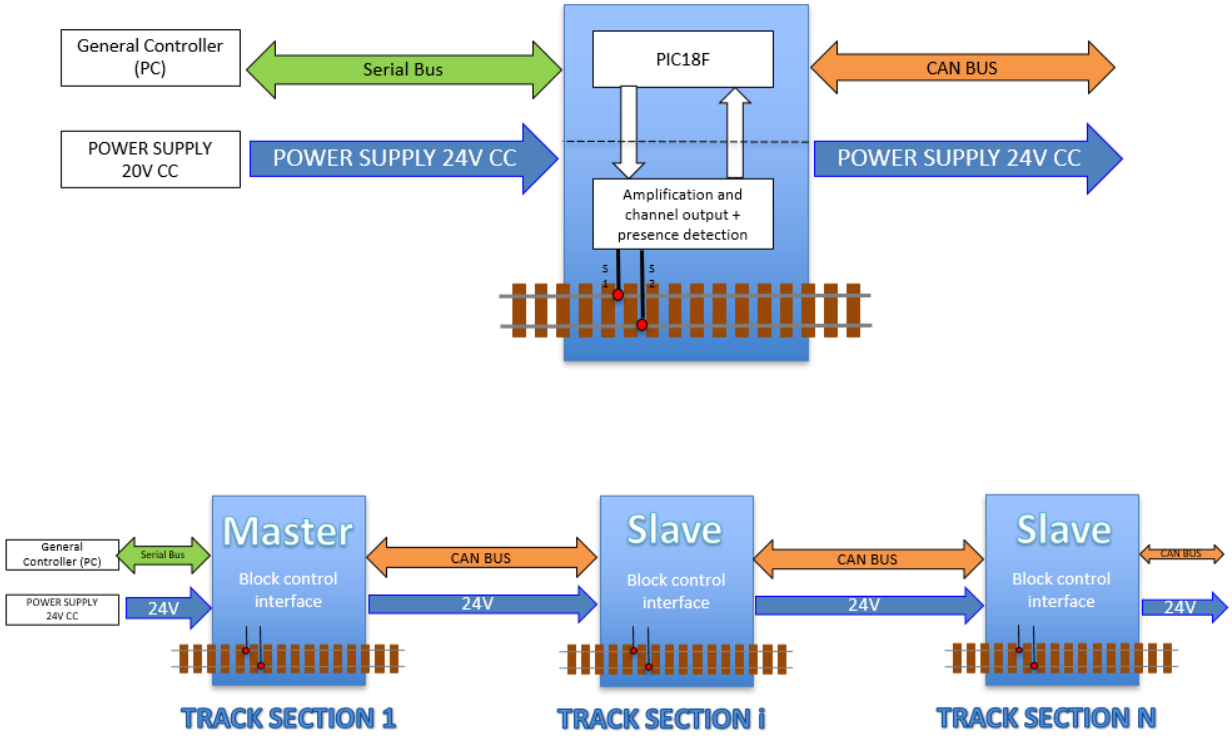


Figure 4 Master/slave board operating diagram

4.2 Data transfer between boards

Data transfer between boards is via a 500 KBaud CAN bus. The boards are connected to the CAN bus.

The first and last boards in this network have a switch for connecting a terminating resistor.

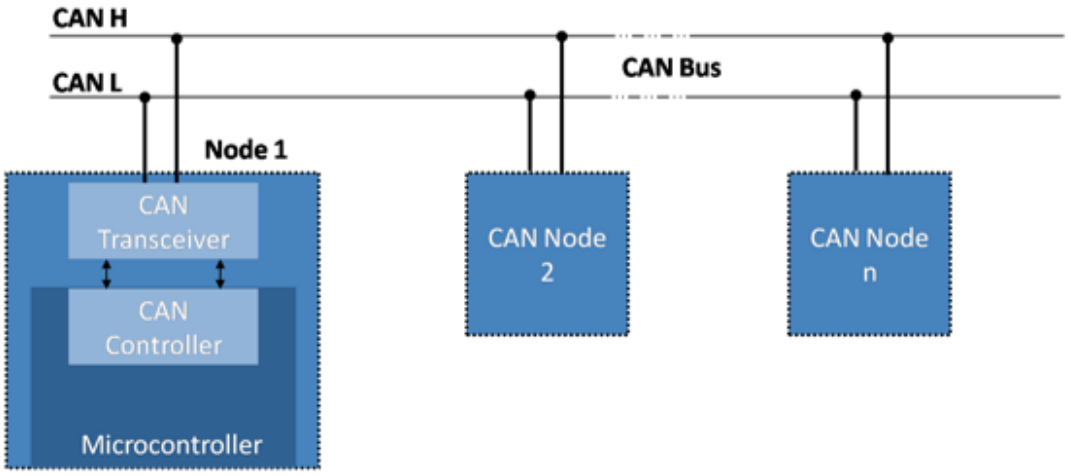


Figure 5 CAN bus wiring diagram

A transceiver MCP2551 is used to link the PIC18F to the CAN bus.

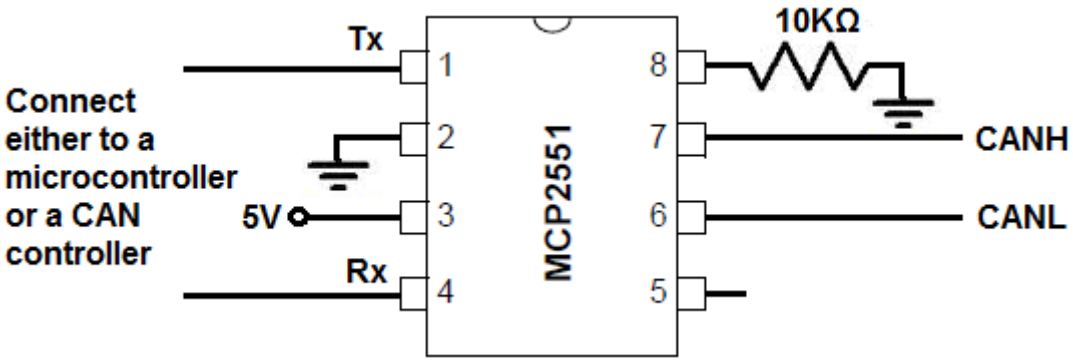


Figure 6 Transceiver MCP2551 wiring diagram

4.3 Data transfer between master board and PC

Data transfer between the master board and the PC is via an RS232 serial bus at 115200 baud. The link to the PC and the PIC18F is done via a MAX232 transmitter, which can be connected to an FTDI RS232/USB cable if the user so wishes.

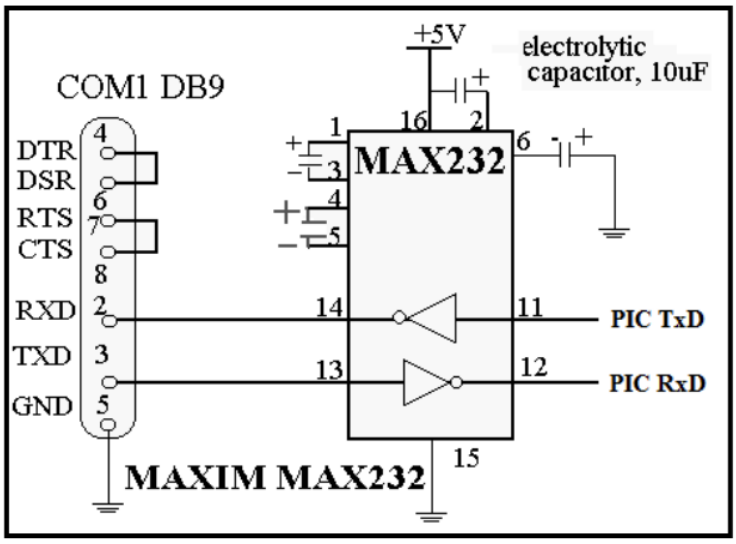


Figure 7 (Proposed by Kyle Thomson Revised 8/18/2009)

4.4 Schematic

The schematic consists of two parts:

- The first contains the 5V power supply (Note the STTH5L06RL inversion protection diode on this power supply). The L7805 could be replaced by a DollaTek 5V 1A regulated board, more robust.(https://www.amazon.fr/DollaTek-r%C3%A9gulateur-bornes-lentr%C3%A9e-LM7805/dp/B081JMJZG6)

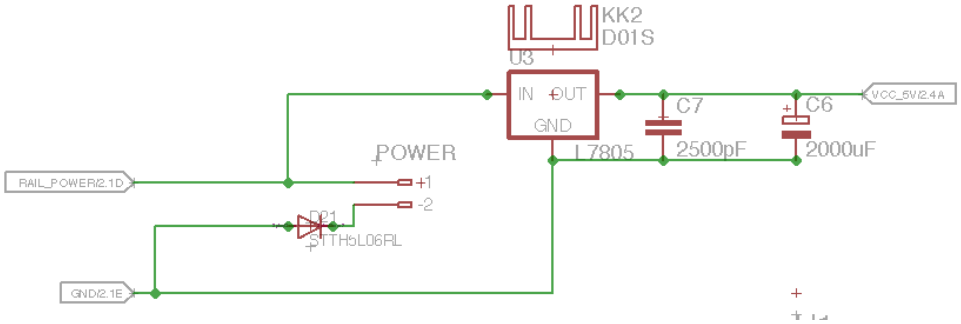


Figure 8 5V power supply with standard 7805

- The RS232 and CAN bus transmitters and transceivers, the programming connector for the PIC18F and the board address selector.

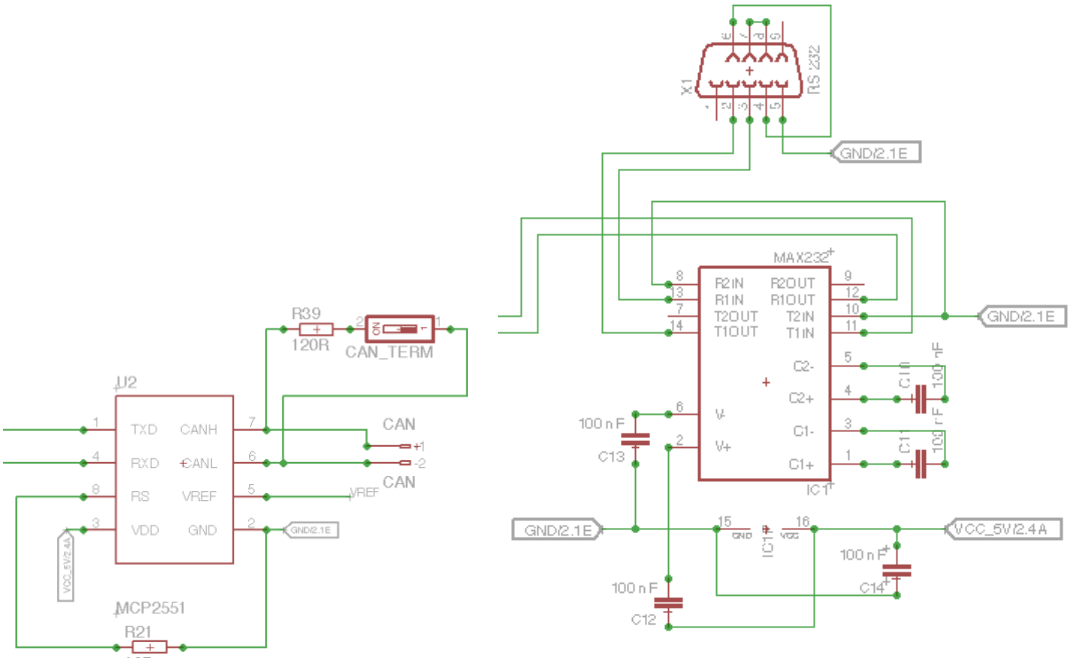


Figure 9 CAN and RS232 interface

- This assembly enables the master and slave boards to be wired on the same design. For the master board, solder the RS232 DB9 connector, the MAX232 transmitter and remove resistors R23 and R24 as well as the address selector. For the slave board, do not solder the RS232 DB9 connector, nor the MAX232 transmitter but keep C14.

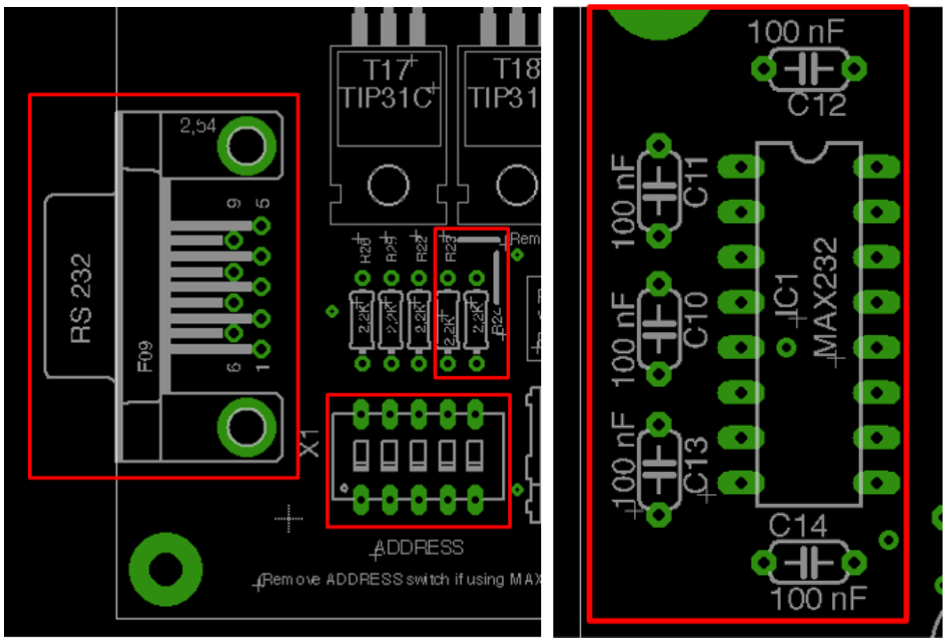


Figure 10 Master or Slave board

- The second contains the L298 power drivers and the 6 TIP31C transistors in Darlington circuit controlled by the PIC18F, as well as the power outputs to the rail or road network.

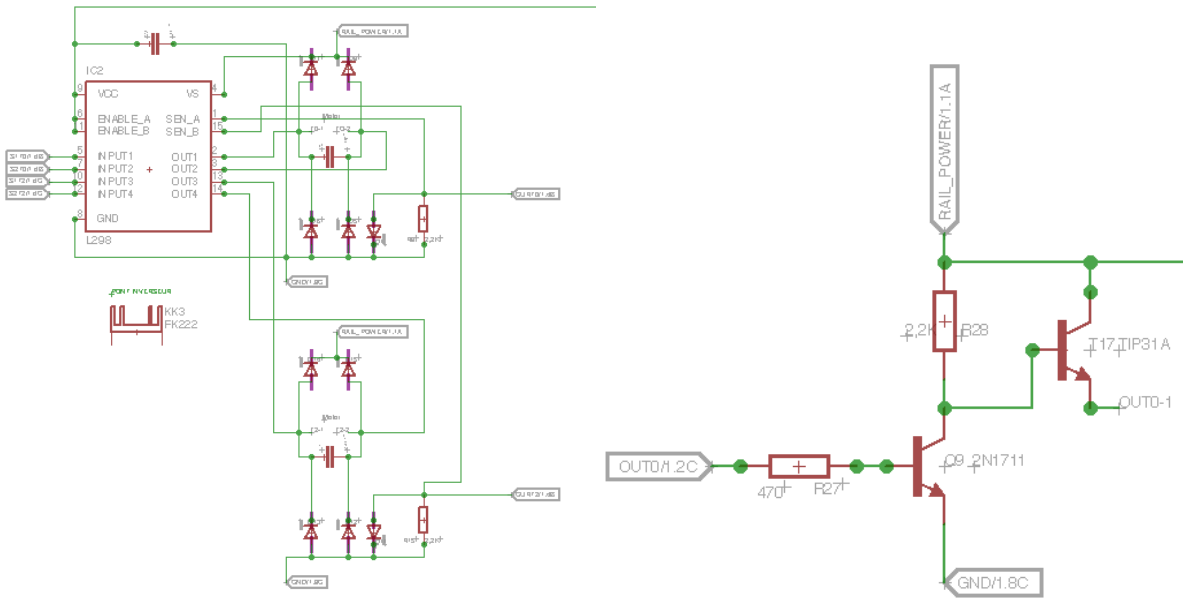


Figure 11Power boost

- Note that it is essential to place decoupling capacitors as close as possible to the components on the 5V supply, otherwise there is a risk of catastrophic loss of control due to PIC18F shutdown. These decoupling capacitors are 100nF on the 5V supply and 2500pF on the power outputs of the tracks to be supplied.

- Vehicle presence is detected by measuring the voltage across a diode at the output of the L298. When no load is present, a voltage close to 0V is measured at the terminal of a 2K2 resistor connected to ground, whereas when a load is present, the connection to ground is made via the 1N4004 diode, whose 0.7V bias voltage is then detected by the PIC18F.

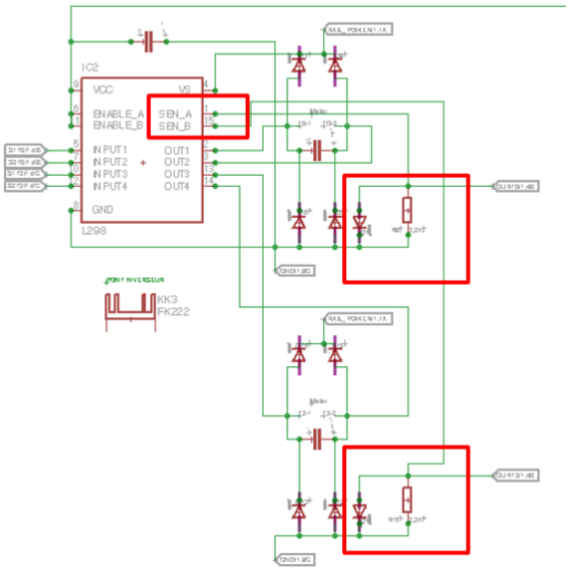
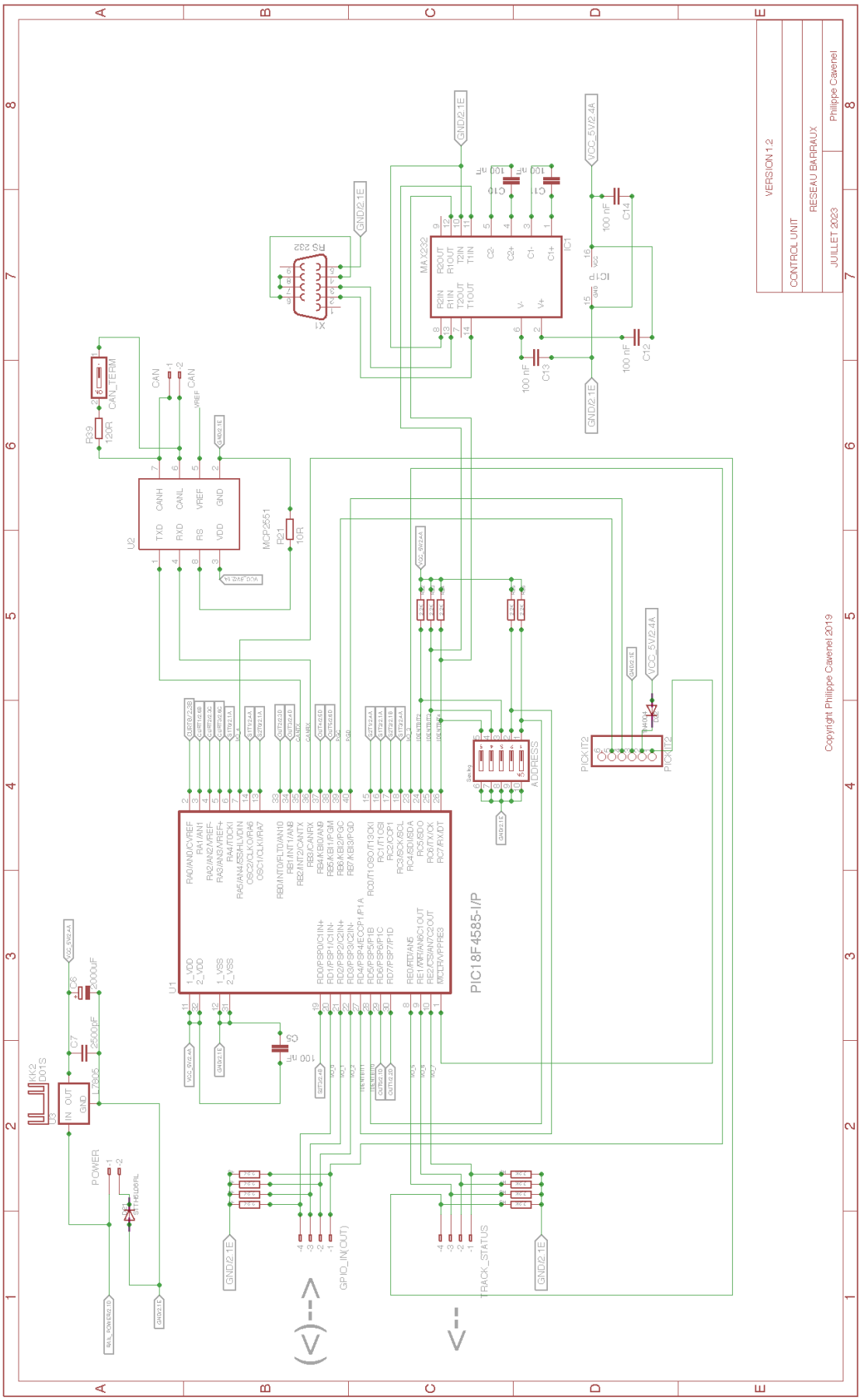
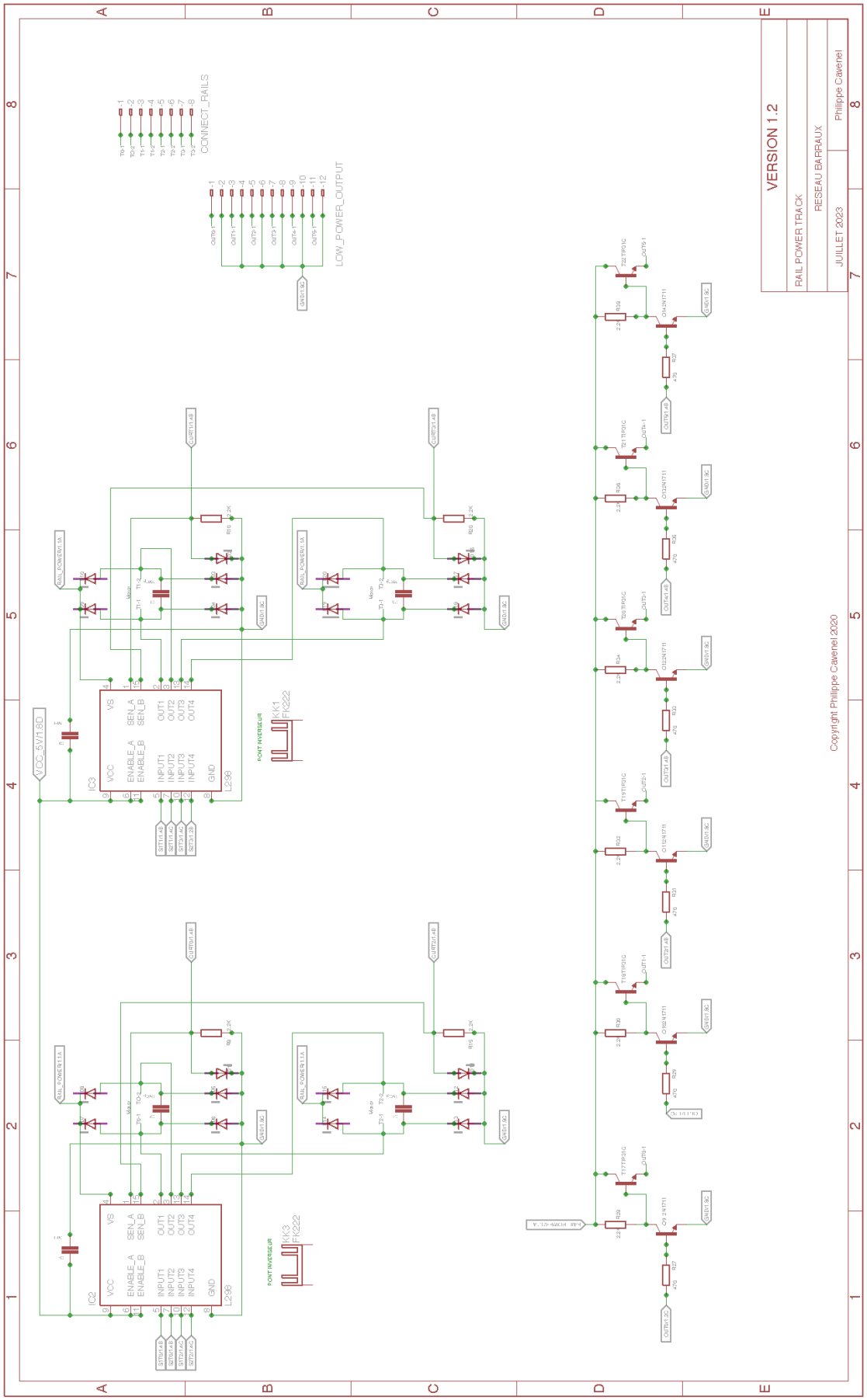


Figure 12 vehicle presence detection


VERSION 1.2
CONTROL UNIT
RESEAU BARRAUX
JUILLET 2023
Philippe Cavenel
Copyright Philippe Cavenel 2019
PIC18F4585-I/P
MAX232
MCP2551
ADDRESS
TRACK_STATUS
GPIO_IN(OUT)
POWER
CAN_TERM
PICKIT2


VERSION 1.2
RAIL POWER TRACK
RESEAU BARRAUX
JUILLET 2023
Philippe Cavenel
CONNECT_RAILS
LOW_POWER_OUTPUT
Copyright Philippe Cavenel 2020

4.5 Pin assignment

PIN	SIGNAL NAME	SIGNAL USED	SCHEMATIC	DIR	FONCTION
1	MCLR/Vpp/RE3	MCLR/Vpp	MCLR/Vpp	IN	PIC programming via PICkit2
2	RA0/AN0/Cvref	AN0	CURT0	IN	Track voltage measurement 0
3	RA1/AN1	AN1	CURT1	IN	Track voltage measurement1
4	RA2/AN2/Vref-	AN2	CURT2	IN	Track voltage measurement2
5	RA3/AN3/Vref+	AN3	CURT3	IN	Track voltage measurement
6	RA4/T0CKI	RA4	S1T0	OUT	S1 Track 0
7	RA5/AN4/SS/HLVDIN	AN4	I/O_4	IN	KNOB 0 analog value
8	RE0/RD/AN5	AN5	I/O_5	IN	KNOB 1 analog value
9	RE1/WR/AN6/C1OUT	RE1	I/O_6	OUT	TM1637 SCK I2C bus (mode)
10	RE2/CS/AN7/C2OUT	RE2	I/O_7	OUT	TM1637 SDA I2C bus (mode)
11	Vdd	Vdd	VCC_5V	IN	+5V
12	Vss	Vss	GND	IN	GND
13	OSC1/CLKI/RA7	RA7	S2T0	OUT	S2 Track 0
14	OSC2/CLKO/RA6	RA6	S1T1	OUT	S1 Track 1
15	RC0/T1OSO/T13CKI	RC0	S2T1	OUT	S2 Track 1
16	RC1/T1OSI	RC1	S1T2	OUT	S1 Track 2
17	RC2/CCP1	RC2	S2T2	OUT	S2 Track 2
18	RC3/SCK/SCL	RC3	S1T3	OUT	S1 Track 3
19	RD0/PSP0/C1IN+	RD0	S2T3	OUT	S2 Track 3
20	RD1/PSP1/C1IN-	RD1	I/O_0	IN	Global Purpose Input 0

PIN	SIGNAL NAME	SIGNAL USED	SCHEMATIC	DIR	FONCTION
21	RD2/PSP2/C2IN+	RD2	I/O_1	IN	Global Purpose Input 1
22	RD3/PSP3/C2IN-	RD3	I/O_2	IN	Global Purpose Input 2
23	RC4/SDI/SDA	RC4	I/O_3	IN	Global Purpose Input 3
24	RC5/SDO	RC5	IDENTBIT2	IN	BIT 2 card identifier
25	RC6/TX/CK	RC6	IDENTBIT3	IN	BIT 3 card identifier or TX line if MAX232 installed
26	RC7/RX/DT	RC7	IDENTBIT4	IN	BIT 4 card identifier or RX line if MAX232 installed
27	RD4/PSP4/ECCP1/P1A	RD4	IDENTBIT1	IN	BIT 1 card identifier
28	RD5/PSP5/P1B	RD5	IDENTBIT0	IN	BIT 0 card identifier
29	RD6/PSP6/P1C	RD6	OUT0	OUT	Low-power control (points, low beam, lighting, level crossings, various motors)
30	RD7/PSP7/P1D	RD7	OUT1	OUT	Low-power control (points, low beam, lighting, level crossings, various motors)
31	Vss	Vss	VCC_5V	IN	+5V
32	Vdd	Vdd	GND	IN	GND
33	RB0/INT0/FLT0/AN10	RB0	OUT2	OUT	Low-power control (points, low beam, lighting, level crossings, various motors)
34	RB1/INT1/AN8	RB1	OUT3	OUT	Low-power control (points, low beam, lighting, level crossings, various motors)
35	RB2/INT2/CANTX	CANTX	CANTX	IN/OUT	Bus CAN Transmission
36	RB3/CANRX	CANRX	CANRX	IN/OUT	Bus CAN Reception
37	RB4/KBI0/AN9	RB4	OUT4	OUT	Low-power control (points, low beam, lighting, level crossings, various motors)
38	RB5/KBI1/PGM	RB5	OUT5	OUT	Low-power control (points, low beam, lighting, level crossings, various motors)
39	RB6/KBI2/PGC	PGC	PGC	IN/OUT	PIC programming via PICkit2
40	RB7/KBI3/PGD	PGD	PGD	IN/OUT	PIC programming via PICkit2

4.6 Place and route

Components are placed as close as possible to each other to limit the distance between related components, especially for decoupling capabilities. Routing is performed automatically.

Top view

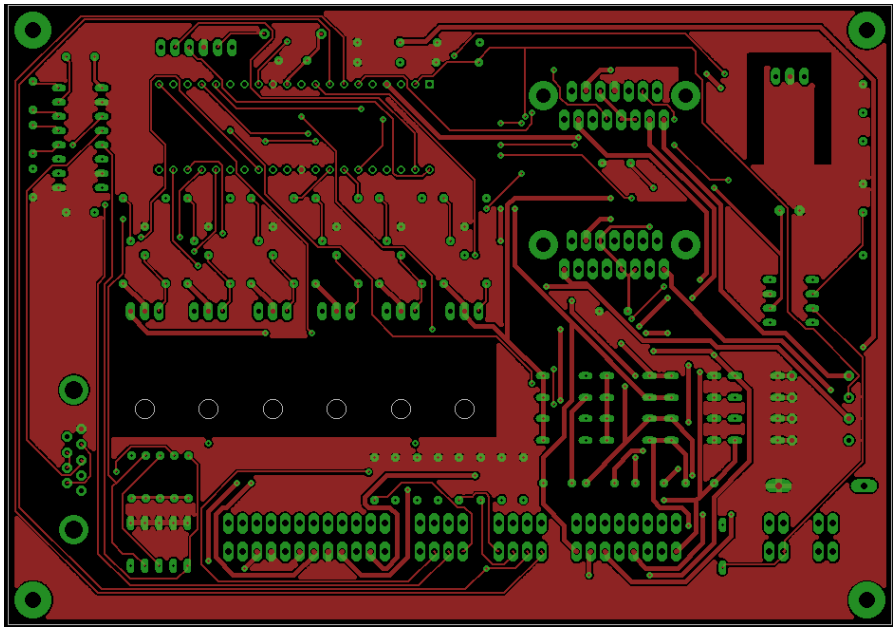


Figure 13 Top, Pads, Vias

Bottom view

Figure 14 Bottom, Pads, Vias

Components view

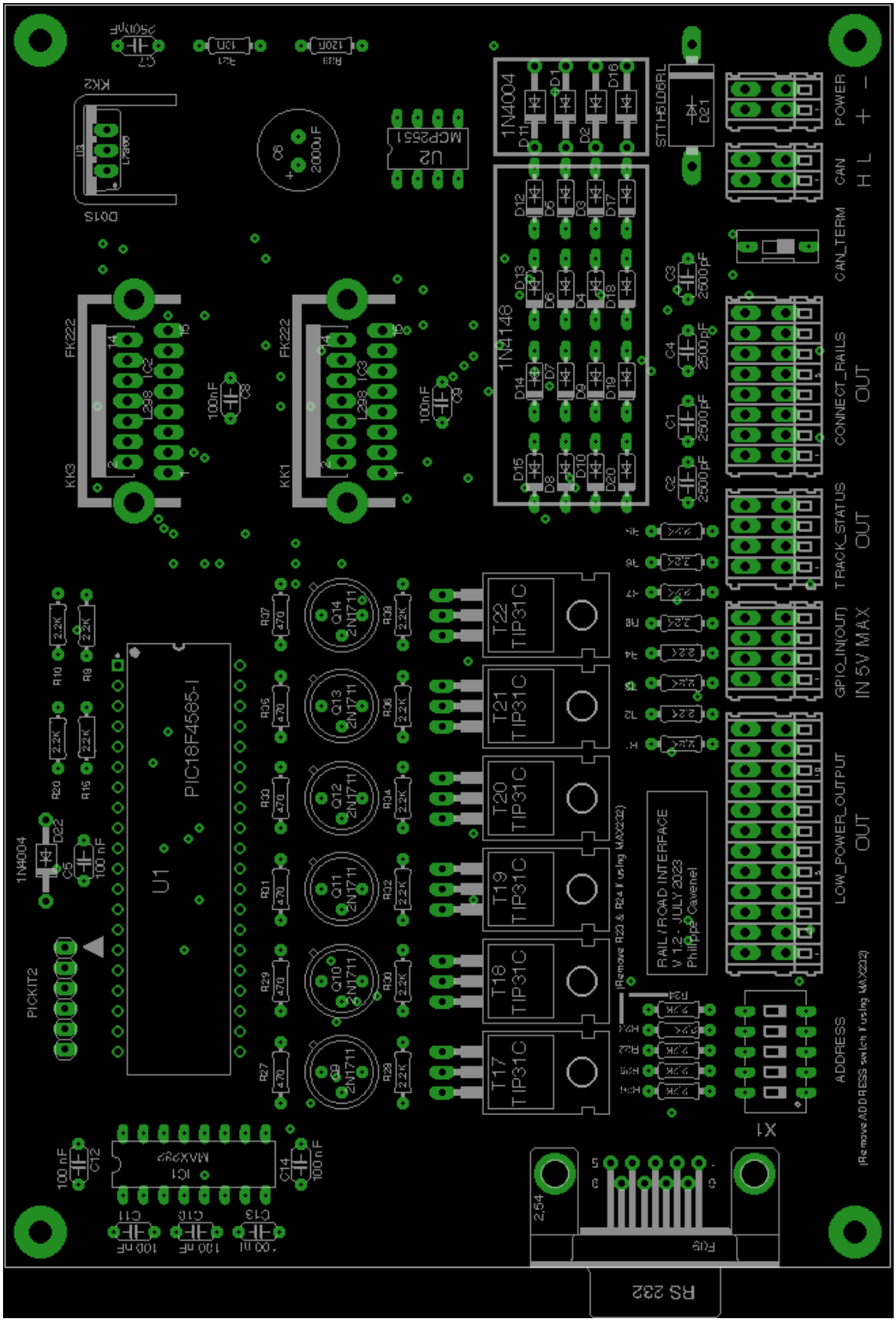


Figure 15 Pads, Vias, tPlace, tValues

3D view

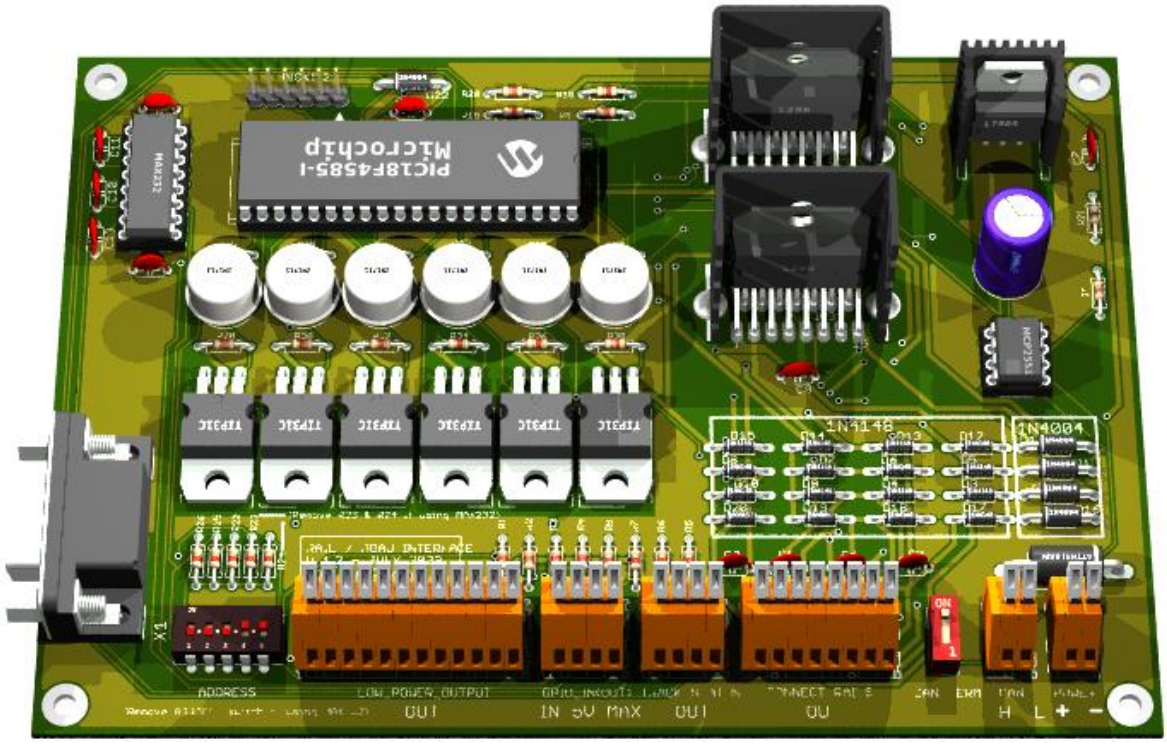


Figure 16 Top View (800 x 600)

Figure 17 Bottom View (800 x 600)

4.7 BOM

Qty	VALUE	DESCRIPTION	Farnell
1	Connecteurs D-Sub standards AMPL PLUG HD20, R/A 9P, B/L,4-40 INS	X1	2857982
8	100nF	C5, C8, C9, C10, C11, C12, C13, C14	2309064
1	MAX 232N	IC1	3121260
1	2000uF	C6	2766923
1	10R	R21	2329993
1	120R	R39	2329862
16	1N4148	D3, D4, D5, D6, D7, D8, D9, D10, D12, D13, D14, D15, D17, D18, D19, D20	9843680
5	1N4004	D22, D1, D11, D2, D16	1843708
1	STTH5L06RL	D21	2353682
23	2,2K	R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R15, R20, R22, R23, R24, R25, R26, R28, R30, R32, R34, R36, R38	9341536
5	2500pF	C1, C2, C3, C4, C7	2860175
6	2N1711	Q9, Q10, Q11, Q12, Q13, Q14	1611558
6	470	R27, R29, R31, R33, R35, R37	3496822
1	Commutateur DIP / SIP, 5 Circuit(s), Glissière, Traversant, SPST, 24 VDC, 100 mA	ADDRESS	3397711
1	Bornier x2	CAN	1777096
1	Bornier x2	POWER	1777096
1	Commutateur DIP / SIP, 1 Circuit	CAN TERMINATOR	1960919
1	Bornier x12	LOW_POWER_OUTPUT	1777102
1	Bornier x8	CONNECT_RAILS	1777101
2	Bornier x4	TRACK_STATUS, GPIO_IN(OUT)	1777098
1	L7805 or DollaTek 5V 1A	U3	1467758
1	MCP2551P	U2	1439745
1	PIC18F4585-I	U1	1439547
1	PICKIT2	PICKIT2	1187827
6	TIP31C	T17, T18, T19, T20, T21, T22	9804145
2	L298N	IC2, IC3	403295
2	FK222	KK1, KK3	4621281
1	274-1AB	KK2	1611445

4.8 Generate the manufacturing files

First step: ERC on Eagle

- Generate schematic and validate circuit with ERC
- Pay particular attention to net class, especially for the power section (larger surface area required)

Step two: DRC and silkscreen on Eagle

- Component placement and automatic routing.
- To fill the top and bottom polygons, draw a polygon around the map, selecting the right layer (Top or Bottom) in the top left-hand corner after routing (otherwise automatic routing won't work), then restart routing.
- Validate with a DRC, selecting only the Top, Bottom, Via and Pad layers.
- Finalize silkscreen (layer 21) by placing all indications correctly, then generate drill legend on layer 144 by launching the ULP icon in the layout screen of Eagle on `drillegend-stack.ulp`

Step three: CAM processor on Eagle

- Generate the files for CAM Processor production by clicking on the `CAM_JOB/RailDriverCamJob.cam` file.
- In the top-left File menu, select the `.brd` file for the corresponding design.
- The directories and file names must be correct in the various tabs (check and correct if necessary).
- All that's left to do is run PROCESS JOB to produce the files.

Fourth stage: Final rendering

- To modify the selection of unknown boxes, modify the `3D_RENDERING/eagle3d/ulp/3dusrpac.dat` file.
- Update the configuration files directory in `3dconf.dat`
- Generate a 3D rendering of the design by launching the ULP icon in the layout screen of Eagle on `3D_RENDERING/eagle3d/ulp/3d50.ulp`
- Specify the `.brd` file, which will generate a `.pov` file that can be processed by POV-Ray to obtain the 3D rendering.
- Select "User-defined model" in the General tab
- Add layer 25 in Miscellaneous Case Design
- Modify Writing on plate to display layers 21 and 25 only.
- Add layers 25 and 27 in Case Reference
- Click on Create POVRay file and Exit.
- Copy the generated `EAGLE_FILES/RailDriver/RailDriver.pov` file into `3D_RENDERING/eagle3d/povray`
- Repeat the same operation, moving the camera to shoot from below (using Y) and/or from the other sides (using X and Z).
- Click on the `3D_RENDERING/eagle3d/povray/RailDriver.pov` file to start building the 3D view with POV-Ray (automatic launch on Windows). Use 800 x 600 AA 0.3 for a reasonable image creation time, or 1600 x 1200 A 0.3 for a better resolution, but the image creation time will take several tens of minutes (On old PC).

- In Eagle, make a screen copy of the various layers built during the construction of the manufacturing files. Place all views in a `PNGVIEW` directory

How to add a 3d model

- Retrieve the model in STL format from the net (snapeda for example on https://www.snapeda.com/). If the model is in STEP format, it must first be converted to STL, you can do that for example at on https://polyd.com/fr/convertir-step-en-stl-en-ligne
- Use the **stl2pov.exe** utility (launch a Windows shell)

```
stl2pov.exe myModel.stl > myModel.inc.
```

- Copy it to `3D_RENDERING/eagle3d/povray`. For example:

```
stl2pov.exe "TO220HeatSink\TO220_Heatsink.stp">"TO220HeatSink\TO220HeatSink.inc"
```

- Modify the `myModel.inc` file so that it is understood by POV-Ray, taking as an example `fk_222_sa.inc`

- Add the following include to the `3D_RENDERING/eagle3d/povray/e3d_user.inc` file:

```
#include "myModel.inc".
```

- Add the following information to the `3D_RENDERING/eagle3d/ulp/3dusrpac.dat` file:

```
MYMODEL:0:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0: MYMODEL
(:MYMODEL_GRND
```

[00] Eagle component package name
[01] Output name
[02] Output value
[03] Define color bands
[04] SMD offset (parts will be moved pcb_cuhight up/down)
[05] LED options (LED options dialog box will be displayed)
[06] Ready for sockets (see explanation)
[07] Quartz height request
[08] A part of a macro (e.g. SMD jumpers)
[09] SMD resistor, generates a combination of numbers
[10] Socket macro
[11] Socket height in 1/10mm
[12] Comments on socket
[13] Internal for administration (not currently used)
[14] Y-axis rotation correction
[15] Correction offset x
[16] Offset correction y
[17] Correction offset z
[18] Use Prefix from Part?
[19] Shunt on pin header (a dialog box will be displayed)
[20] Logo selection dialog box is displayed
[21] Reserved
[29] Minimum encircling box
[30] Maximum bounding box
[31] POV-Ray macro (Name of pov macro and left parenthesis)
[32] Package comments (German)
[33] Package comments (English)

Fifth stage: BOM

- Update the BOM to purchase components on Farnell or Mouser or other dealers on the net.

Sixth step: send files for manufacturing

- Compress `CAM_JOB` and `PNGVIEW` directories to start manufacturing on a PCB manufacturing site such as AllPcb: https://www.allpcb.com/ or other manufacturer on the net.

5 Firmware design

5.1 Set PIC18F frequency to 32MHz

```
    OSCCON = 0x70;          // no pre-divider => 8MHz // comment to set 8MHz
    OSCTUNE = 0x40;         //activate PLL *4 => 32MHz // comment to set 8MHz
```

5.2 CAN bus control library

Introduction

This document describes programming interface to ECAN (Enhanced Controller Area Network) module. At the time of writing this document, only PIC18F8680/6680 family of microcontroller contained ECAN module. This module provides access to ECAN module in polling fashion. This module is completely written in 'C' language. It provides many customization options that may result in significant code reduction. To utilize this module, one must understand all options offered by ECAN module. This module is also available in Microchip Application Note AN878.

(source ECANPoll.ReadMe file)

Module Features

- Out-of-box support for Microchip C18 and HI-TECH PICC-18™ C compilers
- Offers simple abstract interface to ECAN module for most applications
- Additional functions/macros are available for advanced applications
- Supports all three functional modes
- Provides access to all ECAN features in Polling mode
- Easily modifiable to Interrupt-driven mode
- Operates in two main modes:
- Run-time Library Mode and Fixed Library Mode
- Various compile-time options to customization routines to a specific application
- Also available as Microchip Application Note AN878

(source ECANPoll.ReadMe file)

List of Component Modules

`ECAN.ex.txt`	This is main test file developed to demonstrate use of the library functions.
`ECANPoll.c`	This is ECAN code implementation file. One needs to include this file in their project.
`ECANPoll.h`	This file contains prototypes of functions and macros. One needs to include this file in every source file where ECAN functions will be called.
`ECANPoll.def`	This file contains all compile-time options for ECAN module. If you are using Maestro, this file will be created as per your option selections. This file is automatically included by `ECANPoll.h` file.

Functions

Refer to the `ECANPoll.ReadMe` file

CAN Initialization

In order to improve performance, data reception is interrupted.

```
// ECAN
    TRISBbits.TRISB3 = 1; // CANRX input setting
        gl_InputBufferPointer=0;
        gl_getDataCANPointer=0;
        gl_canMode=CAN_UNKNOWN;

    ECANInitialize(); // init ECAN
        PIE3bits.RXB0IE=1; // enable interrupt for CAN
        PIE3bits.RXB1IE=1; // enable interrupt for CAN
```

CAN Reception

Can reception is shared with RS232 reception:

```
/*****************************************************************************/
/* high_isr */
/*****************************************************************************/
#pragma interrupt high_isr
void high_isr(void){

        BYTE dataLen;                           // Number of bytes transmitted in the message
        ECAN_RX_MSG_FLAGS flags;  // Flags
        unsigned long    id;                    // Id of sender

        if(PIR3bits.RXB0IF ||PIR3bits.RXB1IF) {
                while(ECANReceiveMessage(&id, &gl_inputBuffer[gl_InputBufferPointer], &dataLen, &flags)) {
                        gl_InputBufferPointer+=dataLen;
                        if(gl_InputBufferPointer>=MAXTRAMESIZE)gl_InputBufferPointer-=MAXTRAMESIZE;
                }
                gl_synchroSend=(gl_boardNumber+1)*SYNCHROSENDDELAY;
        }

        if (gl_master==FALSE)return;

    // Check if interrupt originates from USART reception
    if (PIR1bits.RCIF)
    {
        // Read received data
        gl_receivedUSARTData[gl_receivedUSARTPointer++] = RCREG;
                if (gl_receivedUSARTPointer>=USARTBUFFERSIZE)gl_receivedUSARTPointer=0;
    }
}
#pragma code
```

Can transmission is performed either by the user_putc() function when an ASCII message is to be sent to the master, or by a specific CAN routine to send compressed data.

```
/*****************************************************************************/
/* CANsendDelay*/
/*****************************************************************************/
void CANsendDelay() {
        unsigned short   delay;
        for(delay=0;delay<WAITDELAYTRAMECAN;delay++);
}

/*****************************************************************************/
/* _user_putc*/
/*****************************************************************************/
int _user_putc (char c) {

        unsigned long    id;                    // Id of sender
    unsigned char        dataOut[8];      // DATA to CAN
        unsigned char    dataCounter;
        unsigned char    dataOutCounter;
        unsigned char    trameComplete;

        BYTE dataLen;                           // Number of bytes transmitted in the gl_message
        ECAN_RX_MSG_FLAGS flags;  // Flags

        // On master board send to UART via standart putc()
```

```
        if (gl_master==TRUE){
                if (gl_inputCounter==0)sendUSART(c);
        }
        else {
                if (gl_outputBufferCounter<MAXMESSAGESIZE)gl_outputBuffer[gl_outputBufferCounter++]=c;

                // Send on CAN bus


                if (gl_outputBufferCounter==MAXMESSAGESIZE || c==ENDOFPRINTFTRAME) {
                        dataLen=8;
                        flags=ECAN_TX_STD_FRAME;
                        id=gl_boardNumber;

                        // header trame
                        for(dataOutCounter=0;dataOutCounter<8;dataOutCounter++)
dataOut[dataOutCounter]=TRAMEPRINTHEADER;

                        // Synchro send
                        while(gl_synchroSend >0);

                        while(!ECANSendMessage(id,dataOut,dataLen,flags));
                        CANsendDelay();

                        // trame
                        trameComplete=FALSE;
                        dataCounter=0;
                        while(dataCounter<MAXMESSAGESIZE) {
                                for(dataOutCounter=0;dataOutCounter<8;dataOutCounter++) {
                                        if (dataCounter<gl_outputBufferCounter) {
                                                dataOut[dataOutCounter]=gl_outputBuffer[dataCounter++];
                                                if (dataOut[dataOutCounter]==ENDOFPRINTFTRAME)
trameComplete=TRUE;
                                        }
                                        else {
                                                dataOut[dataOutCounter]=ENDOFPRINTFTRAME;
                                                if (dataCounter<MAXMESSAGESIZE)dataCounter++;
                                                trameComplete=TRUE;
                                        }
                                }
                                while(!ECANSendMessage(id,dataOut,dataLen,flags));
                                CANsendDelay();

                                if (trameComplete==TRUE) break;
                        }
                        gl_outputBufferCounter=0;

                        // footer trame
                        for(dataOutCounter=0;dataOutCounter<8;dataOutCounter++)
dataOut[dataOutCounter]=TRAMEPRINTFOOTER;
                        while(!ECANSendMessage(id,dataOut,dataLen,flags));
                        CANsendDelay();
                }
        }
        return(c);
}

/*****************************************************************************/
/* sendRequestToCAN() */
/*****************************************************************************/
void sendRequestToCAN(unsigned char* request) {

         unsigned long   id;                     // Id of sender
     unsigned char       dataOut[8];      // DATA to CAN
         unsigned char   dataCounter;
         unsigned char   dataOutCounter;
        BYTE dataLen;                           // Number of bytes transmitted in the gl_message
        ECAN_RX_MSG_FLAGS flags;  // Flags
        unsigned char trameSize;

        // Convert request to dataOut using gl_dataStructure
        trameSize=compressData(request);

        dataLen=8;
        flags=ECAN_TX_STD_FRAME;
        dataCounter=0;
        id=gl_boardNumber;

        // header trame
        for(dataOutCounter=0;dataOutCounter<8;dataOutCounter++) {
                dataOut[dataOutCounter]=TRAMEREQUESTHEADER;
        }

        // Synchro send
        while(gl_synchroSend >0);
        CANsendDelay(); // Delay to avoid sending to fast after receiving a trame
```

```
        while(!ECANSendMessage(id,dataOut,dataLen,flags));
        CANsendDelay();

        // trame
        while(dataCounter<trameSize) {
                for(dataOutCounter=0;dataOutCounter<8;dataOutCounter++) {
                        if (dataCounter<trameSize) {
                                dataOut[dataOutCounter]=request[dataCounter++];
                                if(dataCounter>=MAXTRAMESIZE) return; // Something wrong happened
                        }
                        else {
                                dataOut[dataOutCounter]=0;
                        }
                }
                while(!ECANSendMessage(id,dataOut,dataLen,flags));
                CANsendDelay();
        }

        // footer trame
        for(dataOutCounter=0;dataOutCounter<8;dataOutCounter++) {
                dataOut[dataOutCounter]=TRAMEREQUESTFOOTER;
        }
        while(!ECANSendMessage(id,dataOut,dataLen,flags));
        uncompressData(request); // get back to initial data for other action in manageRequest()
}
/*****************************************************************************/
/* getInputRequestFromCAN() */
/*****************************************************************************/
unsigned char getInputRequestFromCAN(unsigned char* request) {

        unsigned char    requestHeaderTrameDetected=0;
        unsigned char    printHeaderTrameDetected=0;
        unsigned char    requestFooterTrameDetected=0;
        unsigned char    printFooterTrameDetected=0;
        char                    requestTrameEnd;
        char                    printTrameEnd;

        unsigned char    dataInCounter;
        unsigned char    dataStructureCounter;

        sprintf(gl_message,"");
        while (gl_getDataCANPointer!=gl_InputBufferPointer) {

                        if (gl_inputBuffer[gl_getDataCANPointer]==TRAMEREQUESTHEADER)
requestHeaderTrameDetected++;     else requestHeaderTrameDetected=0;
                        if (gl_inputBuffer[gl_getDataCANPointer]==TRAMEREQUESTFOOTER)
        requestFooterTrameDetected++;     else requestFooterTrameDetected=0;

                        if (gl_inputBuffer[gl_getDataCANPointer]==TRAMEPRINTHEADER)
        printHeaderTrameDetected++;               else printHeaderTrameDetected=0;
                        if (gl_inputBuffer[gl_getDataCANPointer]==TRAMEPRINTFOOTER)
        printFooterTrameDetected++;               else printFooterTrameDetected=0;

                        // REQUEST HEADER
                        if (requestHeaderTrameDetected==8) {
                                gl_canMode=CAN_REQUEST;
                                requestHeaderTrameDetected=0;
                                gl_requestTrameStart=gl_getDataCANPointer+1;
                                if (gl_requestTrameStart>=MAXTRAMESIZE)gl_requestTrameStart=0;
                        }

                        // PRINT HEADER
                        else if (printHeaderTrameDetected==8) {
                                gl_canMode=CAN_PRINT;
                                printHeaderTrameDetected=0;
                                gl_printTrameStart=gl_getDataCANPointer+1;
                                if (gl_printTrameStart>=MAXTRAMESIZE)gl_printTrameStart=0;
                        }

                        // REQUEST FOOTER
                        else if (requestFooterTrameDetected==8 && gl_canMode==CAN_REQUEST) {
                                requestTrameEnd=gl_getDataCANPointer-7;
                                if (requestTrameEnd<0)requestTrameEnd+=MAXTRAMESIZE;
                                dataInCounter=gl_requestTrameStart;
                                dataStructureCounter=0;
                                while(dataInCounter!=requestTrameEnd) {
                                        request[dataStructureCounter++]=gl_inputBuffer[dataInCounter++];
                                        if (dataInCounter>=MAXTRAMESIZE)dataInCounter=0;
                                }
                                gl_canMode=CAN_UNKNOWN; // If more data arrive.... we delete this trame
                                gl_getDataCANPointer++;
                                if (gl_getDataCANPointer>=MAXTRAMESIZE)gl_getDataCANPointer=0;
                                uncompressData(request);
                                return(TRUE); // Mean request available to proceed
                        }
```

```
                        // PRINT FOOTER
                        else if (printFooterTrameDetected==8 && gl_canMode==CAN_PRINT) {
                                printTrameEnd=gl_getDataCANPointer-7;
                                if (printTrameEnd<0)printTrameEnd+=MAXTRAMESIZE;
                                dataInCounter=gl_printTrameStart;

                                while(dataInCounter!=printTrameEnd) {
                                        if (gl_master==TRUE)printf("%c",gl_inputBuffer[dataInCounter]);
                                        dataInCounter++;
                                        if (dataInCounter>=MAXTRAMESIZE)dataInCounter=0;
                                }
                                if (gl_master==TRUE) prompt(gl_message);
                                gl_canMode=CAN_UNKNOWN; // If more data arrive.... we continue to get data
                                gl_getDataCANPointer++;
                                if (gl_getDataCANPointer>=MAXTRAMESIZE)gl_getDataCANPointer=0;
                                return(FALSE); // Mean no more data to print
                        }

                        // read new data
                        gl_getDataCANPointer++;
                        if (gl_getDataCANPointer>=MAXTRAMESIZE)gl_getDataCANPointer=0;

        }
        return(FALSE); // Nothing to do
}
```

5.3 PWM management in analog mode, DCC management in NMRA digital mode and TM1637 device management

PWM or DCC signal management is performed under interrupt, including TM1637 7 segment display management

```
////////////////////////////////////////////////////////////////////////////
// INTERRUPT AND SIGNAL MANAGEMENT
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// function set7segmentPort
////////////////////////////////////////////////////////////////////////////
void set7segmentPort(unsigned char CLK, unsigned char DIO) {

        unsigned char myPortE; // used to better synchronised output updates
        unsigned char delay;

    myPortE= (CLK<<1) + (DIO<<2);
    LATE=myPortE;
}

////////////////////////////////////////////////////////////////////////////
// function twoWire_init()
////////////////////////////////////////////////////////////////////////////
void twoWire_init() {
        set7segmentPort(0,0);
}
////////////////////////////////////////////////////////////////////////////
// function twoWire_start()
////////////////////////////////////////////////////////////////////////////
void twoWire_start(){

        set7segmentPort(1,1);
        set7segmentPort(1,0);
}
////////////////////////////////////////////////////////////////////////////
// function twoWire_stop()
////////////////////////////////////////////////////////////////////////////
void twoWire_stop(){
        set7segmentPort(0,0);
        set7segmentPort(1,0);
        set7segmentPort(1,1);
}
////////////////////////////////////////////////////////////////////////////
// function twoWire_ack()
////////////////////////////////////////////////////////////////////////////
void twoWire_ack(){

        set7segmentPort(0,0);
        set7segmentPort(1,0);
```

```
}
////////////////////////////////////////////////////////////////////////////
// function twoWire_write(char data)
////////////////////////////////////////////////////////////////////////////
char twoWire_write(char data){

        unsigned char tx;
        unsigned char DIO;
        for(tx = 0 ; tx < 8 ; tx++) {
                DIO = ((data >> tx) & 0x01) ? 1 : 0 ; //LSB first (Real 12c sends MSB first)
                set7segmentPort(0,DIO);
                set7segmentPort(1,DIO);
                set7segmentPort(0,DIO);
        }
}
////////////////////////////////////////////////////////////////////////////
// function
////////////////////////////////////////////////////////////////////////////
void TM1637_init(void){
    twoWire_init();
}
////////////////////////////////////////////////////////////////////////////
// function TM1637_write(unsigned char number1, unsigned char number2)
////////////////////////////////////////////////////////////////////////////
void TM1637_write(short number1,short number2){


    char str1Num[4];
    char str2Num[4];
    char strNum[8];
    char size;

         sprintf(str1Num,"%3d",number1);
         sprintf(str2Num,"%3d",number2); // 3 characters
         sprintf(strNum,"%s%s",str2Num,str1Num);

    for(size=5;size>=0;size--) {
                if (strNum[size]==' ')twoWire_write(digits[11]);
                else if (strNum[size]=='-')twoWire_write(digits[10]);
            else {
                        unsigned char i = strNum[size] - '0';  //Get index 0 - 9
         twoWire_write(digits[i]);
                }
        twoWire_ack();
    }
}

////////////////////////////////////////////////////////////////////////////
// main function for display TM1637_display(unsigned char number1, unsigned char number2)
////////////////////////////////////////////////////////////////////////////
void TM1637_display(short number1,short number2){

    twoWire_start();
    twoWire_write(0x40);
    twoWire_ack();
    twoWire_stop();

    twoWire_start();
    twoWire_write(0xC0);
    twoWire_ack();
    TM1637_write(number1,number2);

    twoWire_stop();

}
////////////////////////////////////////////////////////////////////////////
// Valid brightness values: 0 - 8.
// 0 = display off.
// main function for display TM1637_setBrightness(char level)
////////////////////////////////////////////////////////////////////////////
void TM1637_setBrightness(char level){

         gl_mutex=1;
    twoWire_start();
    twoWire_write(0x87 + level);
    twoWire_ack();
    twoWire_stop();
         gl_mutex=0;
}

////////////////////////////////////////////////////////////////////////////
// function SetPort
////////////////////////////////////////////////////////////////////////////
void setPort(){

    unsigned char OUTCounter;
```

```
        unsigned char myPortA; // used to better synchronised output updates
        unsigned char myPortB; // used to better synchronised output updates
        unsigned char myPortC; // used to better synchronised output updates
        unsigned char myPortD; // used to better synchronised output updates


        if (gl_stopAll==TRUE) {
                gl_S1T0char=0; gl_S2T0char=0;
                gl_S1T1char=0; gl_S2T1char=0;
                gl_S1T2char=0; gl_S2T2char=0;
                gl_S1T3char=0; gl_S2T3char=0;
                for(OUTCounter=0;OUTCounter<6;OUTCounter++) gl_OUTchar[OUTCounter]=1;
        }

    myPortA=(gl_S1T0char<<4) + (gl_S1T1char<<6) + (gl_S2T0char<<7);
    myPortB=(gl_OUTchar[2]) + (gl_OUTchar[3]<<1) + (gl_OUTchar[4]<<4) + (gl_OUTchar[5]<<5);
        myPortC=(gl_S2T1char) + (gl_S1T2char<<1) + (gl_S2T2char<<2) + (gl_S1T3char<<3) +(TRISCbits.RC4==0 ?
gl_GPIOchar[3] <<4 : 0);
        myPortD=(gl_S2T3char) + (TRISDbits.RD1==0 ? gl_GPIOchar[0] <<1 :0) + (TRISDbits.RD2==0 ? gl_GPIOchar[1]
<<2:0) +(TRISDbits.RD3==0 ? gl_GPIOchar[2] <<3:0) + (gl_OUTchar[0]<<6) +  (gl_OUTchar[1]<<7);



        LATA=myPortA;
    LATB=myPortB;
    LATC=myPortC;
    LATD=myPortD;

}

/******************************************************************************/
/* interrupt_at_high_vector */
/******************************************************************************/
#pragma code high_vector=0x08
void interrupt_at_high_vector(void){
    _asm goto high_isr _endasm
}
#pragma code
/******************************************************************************/
/* high_isr */
/******************************************************************************/
#pragma interrupt high_isr
void high_isr(void){

        BYTE dataLen;                           // Number of bytes transmitted in the message
        ECAN_RX_MSG_FLAGS flags;  // Flags
        unsigned long    id;                    // Id of sender

        if(PIR3bits.RXB0IF ||PIR3bits.RXB1IF) {
                while(ECANReceiveMessage(&id, &gl_inputBuffer[gl_InputBufferPointer], &dataLen, &flags)) {
                        gl_InputBufferPointer+=dataLen;
                        if(gl_InputBufferPointer>=MAXTRAMESIZE)gl_InputBufferPointer-=MAXTRAMESIZE;
                }
                gl_synchroSend=(gl_boardNumber+1)*SYNCHROSENDDELAY;
        }

        if (gl_master==FALSE)return;

    // Check if interrupt originates from USART reception
    if (PIR1bits.RCIF)
    {
        // Read received data
        gl_receivedUSARTData[gl_receivedUSARTPointer++] = RCREG;
                if (gl_receivedUSARTPointer>=USARTBUFFERSIZE)gl_receivedUSARTPointer=0;
    }
}
#pragma code
/******************************************************************************/
/* low_interrupt */
/******************************************************************************/
#pragma code low_vector=0x18
void low_interrupt (){
    _asm goto low_isr _endasm
}
#pragma code
/******************************************************************************/
/* low_isr */
/******************************************************************************/
#pragma interruptlow low_isr
void low_isr(void){
          unsigned char  bitStateCounter;
          unsigned char  delay;
          unsigned char  selectBitDelay;
          unsigned char  bitNumber;
          unsigned char  bitValue;
          short                  ADC;
```

```
        // Synchro send
        if(gl_synchroSend>0)gl_synchroSend--;

        if (!gl_mutex) {

                // KNOB VALUE
                ADCON0=INKNOB0;
                ADCON0bits.GO = 1;                          // ADCON0.GODONE = 1
                while(ADCON0bits.GO == 1);                  // wait till GODONE bit is zero
                ADC = ADRESH;    //Read converted result
                ADC = (ADC<<8) + ADRESL;
                gl_adcKnobValue0=((9*gl_adcKnobValue0)+ADC)/10;
                gl_knobValue0=(9*gl_knobValue0+(gl_adcKnobValue0/20)-25)/10;

                if (gl_knobValue0<-15)gl_knobValue0=-15;
                if (gl_knobValue0>15)gl_knobValue0=15;


                ADCON0=INKNOB1;
                ADCON0bits.GO = 1;                          // ADCON0.GODONE = 1
                while(ADCON0bits.GO == 1);                  // wait till GODONE bit is zero
                ADC = ADRESH;    //Read converted result
                ADC = (ADC<<8) + ADRESL;
                gl_adcKnobValue1=((9*gl_adcKnobValue1)+ADC)/10;
                gl_knobValue1=(9*gl_knobValue1+(gl_adcKnobValue1/9))/10;

                if (gl_knobValue1<0)gl_knobValue1=0;
                if (gl_knobValue1>MAXINERTIAVALUE)gl_knobValue1=MAXINERTIAVALUE;

                // GPIO IN Detection
                if(TRISDbits.RD1==1 && PORTDbits.RD1!=gl_GPIOchar[0]){
                        gl_GPIOstabilized[0]++;
                        if (gl_GPIOstabilized[0]>GPIOTHRESHOLD) {
                                gl_GPIOchar[0]=PORTDbits.RD1;
                                if (gl_GPIOchar[0]==1)gl_GPIOcounter[0]++;
                                gl_GPIONotification[0]=TRUE;
                                gl_GPIOstabilized[0]=0;
                        }
                }
                else gl_GPIOstabilized[0]=0;

                if(TRISDbits.RD2==1 &&PORTDbits.RD2!=gl_GPIOchar[1]){
                        gl_GPIOstabilized[1]++;
                        if (gl_GPIOstabilized[1]>GPIOTHRESHOLD) {
                                gl_GPIOchar[1]=PORTDbits.RD2;
                                if (gl_GPIOchar[1]==1)gl_GPIOcounter[1]++;
                                gl_GPIONotification[1]=TRUE;
                                gl_GPIOstabilized[1]=0;
                        }
                }
                else gl_GPIOstabilized[1]=0;

                if(TRISDbits.RD3==1 & PORTDbits.RD3!=gl_GPIOchar[2]){
                        gl_GPIOstabilized[2]++;
                        if (gl_GPIOstabilized[2]>GPIOTHRESHOLD) {
                                gl_GPIOchar[2]=PORTDbits.RD3;
                                if (gl_GPIOchar[2]==1)gl_GPIOcounter[2]++;
                                gl_GPIONotification[2]=TRUE;
                                gl_GPIOstabilized[2]=0;
                        }
                }
                else gl_GPIOstabilized[2]=0;

                if(TRISCbits.RC4==1 & PORTCbits.RC4!=gl_GPIOchar[3]){
                        gl_GPIOstabilized[3]++;
                        if (gl_GPIOstabilized[3]>GPIOTHRESHOLD) {
                                gl_GPIOchar[3]=PORTCbits.RC4;
                                if (gl_GPIOchar[3]==1)gl_GPIOcounter[3]++;
                                gl_GPIONotification[3]=TRUE;
                                gl_GPIOstabilized[3]=0;
                        }
                }
                else gl_GPIOstabilized[3]=0;

                if (gl_stopAll==FALSE) {

                        // TIMER
                        gl_timer--;
                        if (gl_timer==0) {
                                gl_timer=INITTIMERVALUE;
                                if (gl_TIMERValue[gl_timerNumber]>0){
                                        gl_TIMERValue[gl_timerNumber]--;
                                        if
(gl_TIMERValue[gl_timerNumber]==0)gl_TIMERNotification[gl_timerNumber]=TRUE;
                                }
```

```
                                gl_timerNumber++;
                                if(gl_timerNumber>MAXTIMER)gl_timerNumber=0;
                        }

                        gl_trackNumber=gl_trackNumber+1;
                        if (gl_trackNumber>3)gl_trackNumber=0;

                        ////////////// MODE ANALOG /////////////////
                        if(gl_boardMode==ANAValue) {
                                gl_speedCounter++;
                                if (gl_speedCounter>MAX_INERTIA_COUNTER) {
                                        gl_speedCounter=1;
                                }

                                if (gl_curSpeed[gl_trackNumber]!=gl_setPoint[gl_trackNumber]) {
                                        if (gl_curSpeed[gl_trackNumber]>gl_setPoint[gl_trackNumber]){
                                                gl_curSpeed[gl_trackNumber]-=(MAXINERTIAVALUE-
gl_setStep[gl_trackNumber]+1)/5;

        if(gl_curSpeed[gl_trackNumber]<gl_setPoint[gl_trackNumber])gl_curSpeed[gl_trackNumber]=gl_setPoint[gl_track
Number];
                                        }
                                        else {
                                                gl_curSpeed[gl_trackNumber]+=(MAXINERTIAVALUE-
gl_setStep[gl_trackNumber]+1)/5;

        if(gl_curSpeed[gl_trackNumber]>gl_setPoint[gl_trackNumber])gl_curSpeed[gl_trackNumber]=gl_setPoint[gl_track
Number];
                                        }

                                        if (gl_curSpeed[gl_trackNumber]>0) {

        gl_speed[gl_trackNumber]=gl_curSpeed[gl_trackNumber]/(MAXINTERNALSPEED);
                                                gl_direction[gl_trackNumber]=TRACK_BACKWARD;
                                        }
                                        else if (gl_curSpeed[gl_trackNumber]<0) {
                                                gl_speed[gl_trackNumber]=-
gl_curSpeed[gl_trackNumber]/(MAXINTERNALSPEED);
                                                gl_direction[gl_trackNumber]=TRACK_FORWARD;
                                        }
                                        else {
                                                gl_direction[gl_trackNumber]==TRACK_STOP;
                                                gl_speed[gl_trackNumber]=0;
                                        }
                                }

                                if (gl_speed[gl_trackNumber]>=gl_speedCounter) {
                                        if (gl_direction[gl_trackNumber]==TRACK_FORWARD) {
                                        switch (gl_trackNumber) {
                                                case 0:gl_S1T0char=1; gl_S2T0char=0;break;
                                                case 1:gl_S1T1char=1; gl_S2T1char=0;break;
                                                case 2:gl_S1T2char=1; gl_S2T2char=0;break;
                                                case 3:gl_S1T3char=1; gl_S2T3char=0;break;
                                }
                                }
                                if (gl_direction[gl_trackNumber]==TRACK_BACKWARD) {
                                switch (gl_trackNumber) {
                                        case 0:gl_S1T0char=0; gl_S2T0char=1;break;
                                        case 1:gl_S1T1char=0; gl_S2T1char=1;break;
                                        case 2:gl_S1T2char=0; gl_S2T2char=1;break;
                                        case 3:gl_S1T3char=0; gl_S2T3char=1;break;
                                }
                                }
                                if (gl_direction[gl_trackNumber]==TRACK_STOP) {
                                switch (gl_trackNumber) {
                                        case 0:gl_S1T0char=0; gl_S2T0char=0;break;
                                        case 1:gl_S1T1char=0; gl_S2T1char=0;break;
                                        case 2:gl_S1T2char=0; gl_S2T2char=0;break;
                                        case 3:gl_S1T3char=0; gl_S2T3char=0;break;
                                }
                                }
                                }
                                else {
                                        switch (gl_trackNumber) {
                                case 0:gl_S1T0char=0; gl_S2T0char=0;break;
                                case 1:gl_S1T1char=0; gl_S2T1char=0;break;
                                case 2:gl_S1T2char=0; gl_S2T2char=0;break;
                                case 3:gl_S1T3char=0; gl_S2T3char=0;break;
                                }
                                }

                                setPort();
                        }

                        ////////////// MODE DIGITAL /////////////////
```

```
                        if(gl_boardMode==DCCValue && gl_dcc_ready==0) {

                                for (bitStateCounter=0;bitStateCounter<FRAME_SIZE;bitStateCounter++) {

                                        if (gl_dcc[bitStateCounter]==0) selectBitDelay=DCC_0;
                                        else selectBitDelay=DCC_1;

                                        gl_S1T0char=0;gl_S1T1char=0;
                                        gl_S1T2char=0;gl_S1T3char=0;
                                        gl_S2T0char=1;gl_S2T1char=1;
                                        gl_S2T2char=1;gl_S2T3char=1;
                        setPort();
                                        for (delay=0;delay<selectBitDelay;delay++);

                                gl_S2T0char=0;gl_S2T1char=0;
                                        gl_S2T2char=0;gl_S2T3char=0;
                                        gl_S1T0char=1;gl_S1T1char=1;
                                        gl_S1T2char=1;gl_S1T3char=1;
                        setPort();
                                        for (delay=0;delay<selectBitDelay;delay++);
                                }

                                gl_S1T0char=0; gl_S2T0char=1;
                                gl_S1T1char=0; gl_S2T1char=1;
                                gl_S1T2char=0; gl_S2T2char=1;
                                gl_S1T3char=0; gl_S2T3char=1;
                setPort();
                                for (delay=0;delay<selectBitDelay;delay++);
                                for
(bitStateCounter=0;bitStateCounter<FRAME_SIZE;bitStateCounter++)gl_dcc[bitStateCounter]=1;
                        }
                        gl_dcc_ready--;
                        if (gl_dcc_ready<0) gl_dcc_ready=INITWAITDCCCOUNTER;
                        setPort();

                        // TRACK DETECTION

                        switch(gl_trackNumber) {
                                case 0 : ADCON0=CURT0;break;
                        case 1 : ADCON0=CURT1;break;
                        case 2 : ADCON0=CURT2;break;
                                case 3 : ADCON0=CURT3;break;
                        }
                        // NEED TO GET LOW VOLTAGE VALUE WHEN TRACK IS OFF FOR CALIBRATION AT POWER ON
                        if (gl_calibration==TRUE) {
                                ADCON0bits.GO = 1;                              // ADCON0.GODONE = 1
                                while(ADCON0bits.GO == 1);                      // wait till GODONE bit is zero
                                ADC = 0;
                                ADC = ADRESH;    //Read converted result
                                ADC = (ADC<<8) + ADRESL;

        gl_average[gl_trackNumber]=(SAMPLEFORCALIBRATION*gl_average[gl_trackNumber]+ADC)/(SAMPLEFORCALIBRATION+1);
                                if
(gl_noVehicule[gl_trackNumber]>gl_average[gl_trackNumber])gl_noVehicule[gl_trackNumber]=gl_average[gl_trackNumber];
                        }

                        if((gl_speed[gl_trackNumber]==gl_speedCounter && gl_boardMode==ANAValue) ||
(gl_dcc_ready==INITWAITDCCCOUNTER && gl_boardMode==DCCValue)) { // ONLY WHEN POWER ON
                                ADCON0bits.GO = 1;                              // ADCON0.GODONE = 1
                                while(ADCON0bits.GO == 1);                      // wait till GODONE bit is zero
                                ADC = 0;
                                ADC = ADRESH;    //Read converted result
                                ADC = (ADC<<8) + ADRESL;

                                if (gl_average[gl_trackNumber]<ADC) gl_average[gl_trackNumber]=ADC; // TRAP THE
EVENT
                                else
gl_average[gl_trackNumber]=(SAMPLEFORAVERAGE*gl_average[gl_trackNumber]+ADC)/(SAMPLEFORAVERAGE+1);

                                if
((10*gl_average[gl_trackNumber]>(10+HYSTERERISHIGH)*gl_noVehicule[gl_trackNumber]) &&
(gl_OUTSTATchar[gl_trackNumber]==0)  && (gl_trackNotification[gl_trackNumber]==FALSE)) {
                                        gl_OUTSTATchar[gl_trackNumber]=1;
                                        gl_trackNotification[gl_trackNumber]=TRUE;
                                }
                                else if
((10*gl_average[gl_trackNumber]<(10+HYSTERERISLOW)*gl_noVehicule[gl_trackNumber]) &&
(gl_OUTSTATchar[gl_trackNumber]==1) && (gl_trackNotification[gl_trackNumber]==FALSE)) {
                                        gl_OUTSTATchar[gl_trackNumber]=0;
                                        gl_trackNotification[gl_trackNumber]=TRUE;
                                }
                        }
                }
                else setPort();
        }
```

```
    // INTERRUPT RESET
    if(INTCONbits.TMR0IF==1){
            INTCONbits.TMR0IF = 0;
                T0CONbits.PSA                   = 0;   // Timer0 prescaler is assigned
                T0CONbits.T0PS0                 = 0;   // Prescale value
                T0CONbits.T0PS1                 = 0;   // Prescale value
                T0CONbits.T0PS2                 = 0;   // Prescale value
    }
}
#pragma code
```

The DCC frame in standard NMRA format is generated by the setDcc() function and sent by `low_isr(void)` shown above . Please note that the timings have been adjusted by the following define values:

```
// DELAY FOR DCC SIGNAL
#define DCC_0 48
#define DCC_1 18
```

Any modification to the code, such as the use of `int` instead of `char` for example, may modify the timing and cause the DCC signal to become invalid. In this case, it is necessary to check the correct timing values of the DCC signal with an oscilloscope (58us for a 1 and 100us for a 0).

```
////////////////////////////////////////////////////////////////////////////
// setDcc
////////////////////////////////////////////////////////////////////////////
void setDcc(unsigned char address, unsigned char command) {

         unsigned char i;
         unsigned char bitNumber;
         unsigned char control;

        control=address ^ command; // Control
        bitNumber=0;

        // PREAMBULE
    for(i=0;i<PREAMBLE_SIZE;i++) {
                gl_dcc[bitNumber++]=1;
        }
        // 0
        gl_dcc[bitNumber++]=0;

        // ADDRESS
    for(i=0;i<8;i++) {
                gl_dcc[bitNumber++]=address >> (7-i) & 1;
        }
        // 0
        gl_dcc[bitNumber++]=0;

        // COMMAND
    for(i=0;i<8;i++) {
                gl_dcc[bitNumber++]=command >> (7-i) & 1;
        }
        // 0
        gl_dcc[bitNumber++]=0;

        // CONTROL
    for(i=0;i<8;i++) {
                gl_dcc[bitNumber++]=control >> (7-i) & 1;
        }
        // 1
        gl_dcc[bitNumber++]=1;
        gl_dcc[bitNumber++]=1; // Only one is enough, but in case of....
}
```

5.4 Flash storage of programming information

EEPROM data storage is managed by a set of flash read/write functions. In order to get more space, automations are compressed inside EEPROM.

```
////////////////////////////////////////////////////////////////////////////
// EEPROM READ / WRITE FUNCTION
////////////////////////////////////////////////////////////////////////////
```

```
/*******************************************************
        This function reads a byte at given addresse in EEPROM.
        IN:             address
        OUT:    data
        Return Value: ERROR, SUCCESS
*******************************************************/
unsigned char ReadEEPROM(unsigned int adr, unsigned char *data){

        if(adr > 0x3FF){
                return(ERROR);
        }
        else{
                EEADR = adr&0xFF;
                EEADRH = (adr>>8) & 0x3;
                EECON1bits.EEPGD = 0;     // Point to data memory
                EECON1bits.CFGS = 0;      // Access EEPROM
                EECON1bits.RD = 1;                // Read data
                *data = EEDATA;                   // Load data
                return(SUCCESS);
        }
} // end of ReadEEPROM()

/*****************************************************************************
*
* Function ResetEEPROM
*
*****************************************************************************/
void ResetEEPROM(){

         unsigned char   value;
         unsigned short adr;
         unsigned char  checkMagicNumberCounter;

        // Init EEPROM after flashing the board
        adr=(unsigned short)MAGICNUMBER_ADDRESS;
        for (checkMagicNumberCounter=0;checkMagicNumberCounter<MAGICNUMBERSIZE;checkMagicNumberCounter++) {
                WriteEEPROM(adr++,checkMagicNumberCounter);
        }
        // Set ANA mode
        adr=(unsigned short)MODE_ADDRESS;
        value=ANAValue;
        WriteEEPROM(adr,value);

        // No automation
        adr=(unsigned short)NEXTTAUTOMATION_ADDRESS;
        value=0;
        WriteEEPROM(adr,value);

        gl_mutex=1;gl_mode = ANAValue;gl_nexAvailableAutomation=0;gl_mutex=0;

        // GPIO IN
        for(adr=(unsigned short)GPIO0DIR_ADDRESS;adr<=(unsigned short)GPIO0DIR_ADDRESS+3;adr++)WriteEEPROM(adr,1);

        // Init
        initSignal();

        // Calibration
        calibration();
}

/*****************************************************************************
*
* Function ReadEEPROMConfig
*
*****************************************************************************/
void ReadEEPROMConfig(void) {

         unsigned char   value;
         unsigned short adr;
         unsigned char  automationCounter;
         unsigned char  automationDataCounter;
         unsigned char  checkMagicNumberCounter;

        // Read MAGIC NUMBER
        adr=(unsigned char)MAGICNUMBER_ADDRESS;
        for (checkMagicNumberCounter=0;checkMagicNumberCounter<MAGICNUMBERSIZE;checkMagicNumberCounter++) {
                ReadEEPROM(adr++,&value);
                if (value!=checkMagicNumberCounter) {

                        ResetEEPROM();
                        return;
                }
        }

        // Read in EEPROM MODE
        adr=(unsigned short)MODE_ADDRESS;
```

```
        ReadEEPROM(adr,&value);
        gl_mutex=1;gl_mode=value;gl_mutex=0;

        // Read in GPIO dir
        adr=(unsigned short)GPIO0DIR_ADDRESS;
        ReadEEPROM(adr++,&value);
        TRISDbits.RD1=value;
        ReadEEPROM(adr++,&value);
        TRISDbits.RD2=value;
        ReadEEPROM(adr++,&value);
        TRISDbits.RD3=value;
        ReadEEPROM(adr,&value);
        TRISCbits.RC4=value;

        // Read in EEPROM last automation
        uncompressAutomation();
}

/*********************************************************
        This function informs on if write to EEPROM is completed.
        IN:             None
        OUT:    None
        Return Value: IN_PROGRESS, SUCCESS
*********************************************************/
unsigned char WriteCompletedEEPROM(void) {

        if(PIR2bits.EEIF){
                PIR2bits.EEIF=0;          // Clear write complete flag
                EECON1bits.WREN = 0;      // Disable write
                return(SUCCESS);          // Write to EEPROM completed
        }
        else {
                return(ERROR);                    // Write to EEPROM not completed
        }
}

/*********************************************************
        This function informs on if it is possible to write in EEPROM.
        IN:             None
        OUT:    None
        Return Value: ERROR, SUCCESS
*********************************************************/
unsigned char WriteRdyEEPROM(void){

        if(!EECON1bits.WR) {
                return(SUCCESS); // New Write Enabled
        }
        else {
                return(ERROR);            // new Write Disabled
        }
}

/*********************************************************
        This function writes a byte at given addresse in EEPROM.
        IN:             addresse, data
        Return Value: ERROR, SUCCESS
*********************************************************/
unsigned char WriteEEPROM(unsigned short adr, unsigned char data){
        if(adr > 0x3FF){
                return(ERROR);
        }
        else{

                // Wait eeprom ready to be written
                while (WriteRdyEEPROM()==(unsigned char) ERROR);

                EEADR = adr&0xFF;                 // Address of the data in EEPROM
                EEADRH = (adr>>8) & 0x3;  // Address of the data in EEPROM
                EEDATA = data;                                  // Data to write in EEPROM
                EECON1bits.EEPGD = 0;             // Point to data memory
                EECON1bits.CFGS = 0;              // Access EEPROM
                EECON1bits.WREN = 1;              // Enable write
                INTCONbits.GIE   = 0;             // Disable Interrupt
                EECON2 = 0x55;
                EECON2 = 0x0AA;
                EECON1bits.WR = 1;                        // Begin write

                // Wait data written
                while (WriteCompletedEEPROM()==(unsigned char) IN_PROGRESS);
                while (WriteRdyEEPROM()==(unsigned char) ERROR);

                INTCONbits.GIE            = 1;    // Enable Interrupt

                return(SUCCESS);
        }
} // end of WriteEEPROM()
```

```
////////////////////////////////////////////////////////////////////////////
// memAvailable
////////////////////////////////////////////////////////////////////////////
void memAvailable() {

        sprintf(gl_message,"Memory available %d %",uncompressAutomation());
        prompt(gl_message);
        if (gl_nexAvailableAutomation>0) {
                if (gl_nexAvailableAutomation==1)sprintf(gl_message,"%d automation",gl_nexAvailableAutomation);
                else sprintf(gl_message,"%d automations",gl_nexAvailableAutomation);
                prompt(gl_message);
        }
        sprintf(gl_message,"");
        prompt(gl_message);
}

////////////////////////////////////////////////////////////////////////////
// uncompressAutomation
////////////////////////////////////////////////////////////////////////////
unsigned char uncompressAutomation() {

        unsigned short quantityValue;
    unsigned char  automationCounter;
    unsigned char  automationDataCounter;
        unsigned char  value;
        unsigned short adr;
        long                percentage;

        // Get next automation address
        adr=(unsigned short)NEXTTAUTOMATION_ADDRESS;
        ReadEEPROM(adr,&value);
        gl_nexAvailableAutomation=value;
        if (gl_nexAvailableAutomation==0) return((double)100); // nothing to do

        // Read and uncompress automation from EEPROM
        adr=(unsigned short)AUTOMATION_ADDRESS;
        ReadEEPROM(adr++,&value);
        quantityValue=value;
        ReadEEPROM(adr++,&value);

        for(automationCounter=0;automationCounter<gl_nexAvailableAutomation;automationCounter++) {
                for(automationDataCounter=0;automationDataCounter<AUTOMATIONSIZE;automationDataCounter++) {
                        gl_automation[automationCounter][automationDataCounter]=value;
                        if (quantityValue>0)quantityValue--;
                        if (quantityValue==0) {
                                ReadEEPROM(adr++,&value);
                                quantityValue=value;
                                ReadEEPROM(adr++,&value);
                        }
                        if (quantityValue==0 || adr>=1024) {
                                percentage=100*(1024-(long)adr)/(1024-(long)AUTOMATION_ADDRESS);
                                return((unsigned char)percentage);
                        }
                }
        }
        percentage=100*(1024-(long)adr)/(1024-(long)AUTOMATION_ADDRESS);
        return((unsigned char)percentage);
}

////////////////////////////////////////////////////////////////////////////
// compressAutomation
////////////////////////////////////////////////////////////////////////////
unsigned char compressAutomation() {

    unsigned char  automationCounter;
    unsigned char  automationDataCounter;
        unsigned short dataCounter;
        unsigned short dataEeprom;
        unsigned short quantityValue;
        unsigned char  curValue;
        unsigned short adr;
        unsigned char  value;
        unsigned char  checkValue;


        // Codage is simply a list of (X,Y) where X is the number of Y.
        dataCounter=0;
        curValue=gl_automation[0][0];
        quantityValue=0;

        // Check size
        for(automationCounter=0;automationCounter<gl_nexAvailableAutomation;automationCounter++) {
                for(automationDataCounter=0;automationDataCounter<AUTOMATIONSIZE;automationDataCounter++) {
                        if (gl_automation[automationCounter][automationDataCounter]==curValue) {
                                quantityValue++;
```

```
                                if (quantityValue>=256) {
                                        gl_parserErrorCode=AUTOMATIONSIZELIMIT;
                                        return(FALSE);
                                }
                        }
                        else {
                                if (dataCounter<=(unsigned short)(1021-AUTOMATION_ADDRESS)) {
                                        dataCounter+=2;
                                }
                                else {
                                        gl_parserErrorCode=AUTOMATIONSIZELIMIT;
                                        return(FALSE);
                                }
                                quantityValue=1;
                                curValue=gl_automation[automationCounter][automationDataCounter];
                        }
                }
        }

        // Write to eeprom
        curValue=gl_automation[0][0];
        quantityValue=0;
        adr=(unsigned short)AUTOMATION_ADDRESS;

        for(automationCounter=0;automationCounter<gl_nexAvailableAutomation;automationCounter++) {
                for(automationDataCounter=0;automationDataCounter<AUTOMATIONSIZE;automationDataCounter++) {
                        if (gl_automation[automationCounter][automationDataCounter]==curValue) {
                                quantityValue++;
                        }
                        else {
                                value=quantityValue;
                                ReadEEPROM(adr,&checkValue);
                                if (checkValue!=value)WriteEEPROM(adr++,value);else adr++;
                                value=curValue;
                                ReadEEPROM(adr,&checkValue);
                                if (checkValue!=value)WriteEEPROM(adr++,value);else adr++;
                                quantityValue=1;
                                curValue=gl_automation[automationCounter][automationDataCounter];
                        }
                }
        }
        // Write end of automation list (no more data)
        if (adr<=1021) {

                value=quantityValue;
                ReadEEPROM(adr,&checkValue);
                if (checkValue!=value)WriteEEPROM(adr++,value);else adr++;
                value=curValue;
                ReadEEPROM(adr,&checkValue);
                if (checkValue!=value)WriteEEPROM(adr++,value);else adr++;

                value=0;
                ReadEEPROM(adr,&checkValue);
                if (checkValue!=value)WriteEEPROM(adr++,value);else adr++;
        }

        // update in EEPROM next automation value
        adr=(unsigned short)NEXTTAUTOMATION_ADDRESS;
        value=gl_nexAvailableAutomation;
        ReadEEPROM(adr,&checkValue);
        if (checkValue!=value)WriteEEPROM(adr++,value);
        return(TRUE);
}
```

5.5 Automation management

The system is automated using a language that allows commands to be given or actions to be programmed on receipt of events. Syntax is parsed using a parser() function, and semantics and automation management are handled by a function for managing the system's various requests (from the RS232 link or the CAN bus).

A main loop in the `main()` function analyzes data from the RS232 link, CAN bus, internal timers, GPIOs or vehicle presence detection on the tracks.

5.6 Software design

5.7 Protocol on serial link

The protocol describes all the data exchanged between a PC and a master board. All data exchanged on the serial bus are in ASCII format, so the value 25 is transmitted with the letter “2” followed by the letter “5”. This allows the board to be controlled directly from a terminal. Each transmission could be followed by one or more return messages.

- If message sent to the master needs a specific response
  - a status of a board (DCC or ANA)
  - a list of automation (value and name)
  - a status of a track
  - a status of GPIO
  - a status of a board
- Error: the board number with an error message

5.8 BNF Grammar

Sending a command on the serial bus uses the following BNF grammar:

```
Command         ::=  <Mode> <Board Number> <Control> |
                     <Global Command>
Mode            ::=  PROG | COM
Board Number    ::=  <between 0 and 31>
Control         ::=  <Program> | <Command>
Global Command ::=  STOP | RUNALL | RUN <Board Number> |RESET <Board Number>
```

For Programing Mode (P)

```
Program         ::= <Board Mode> |
                    <GPIO Setting> |
                    <Automation> |
                    <Del Automation>
Board Mode      ::= DCC | ANA
GPIO Setting    ::= GPIO <GPIO Number> <GPIO Dir>
GPIO number     ::= 0 | 1 | 2 | 3
GPIO Dir        ::= IN | OUT
Automation      ::= AUT <Identifier> <Manual Status> <Event> ACT <Action>
Manual Status   ::= AUTOFF | AUTON
Identifier      ::= <List of 2 characters max between 2 spaces>
Event           ::= BOARD < Board Number> TIMER <Timer Number>
                    BOARD < Board Number> GPIO < Board Number> <GPIO Level>|
                    BOARD < Board Number> TRACK < Track Number> STA <vehicle status>
Board Number    ::= <between 0 and 31>
Timer number    ::= <A value between 0 and 15>
GPIO Level      ::= <0 or 1 for output, a value between 0 and 255 and input, counter mode>
vehicle status ::= ONTRACK | OFFTRACK
Action          ::= TIMER <Timer Number> <Timer Delay>
                    GPIO <GPIO Number> <GPIO Level> |
                    LPO <LPO Number> <LPO Level> |
                    TRACK <Track Number> SPEED <Track Speed> <Track Dir> INERTIA <Inertia>|
                    DCC <DCC NMRA>|
                    MANUAL | AUTOMATIC |
                    AUTOFF <identifier> | AUTON <identifier>
Timer delay     ::= <A value between 0 and 255>
LPO Number      ::= 0 | 1 | 2 | 3 | 4 | 5
LPO Level       ::= 0 | 1
Track Number    ::= 0 | 1 | 2 | 3 | 4
Track Speed     ::= <A value between 0 and 99> | KNOB0 | KNOB1
Track Dir       ::= FORW | BACK
Inertia         ::= <A value between 0 and 99> | KNOB0 | KNOB1
Del Automation ::= DEL <Automation Number>
```

For Command Mode (C)

```
Command         ::= < Action > | <GPIO Status> | <LPO Status> | <Track Status> |
                    <Board Status> |<Automation List> | <Dump Memory> | <Knob Calib>
Action          ::= TIMER<Timer Number> <Timer delay>
                    GPIO <GPIO Number> <GPIO Level> |
                    LPO <LPO Number> <LPO Level> |
                    TRACK <Track Number> SPEED <Track Speed> <Track Dir> INERTIA <Inertia>|
                    DCC <DCC NMRA> |
                    MANUAL | AUTOMATIC
                    AUTOFF <identifier> | AUTON <identifier>
Timer number    ::= <A value between 0 and 15>
Timer delay     ::= <A value between 0 and 255>
GPIO Number     ::= 0 | 1 | 2 | 3
GPIO Level      ::= <0 or 1 for output, a value between 0 and 255 and input, counter mode>
LPO Number      ::= 0 | 1 | 2 | 3 | 4 | 5
LPO Level       ::= 0 | 1
Track Number    ::= 0 | 1 | 2 | 3 | 4
Track Speed     ::= <A value between 0 and 99> | KNOB0 | KNOB1
Inertia         ::= <A value between 0 and 99> | KNOB0 | KNOB1
DCC NMRA        ::= <See NMRA standard specification>
GPIO Status     ::= GSTAT
LPO Status      ::= LSTAT
Track Status    ::= TSTAT
Board Status    ::= BSTAT
Automation List::= AUTLIST
Dump Memory     ::= DUMP
Knob Calib      ::= CALIB
```

Any other syntax in command or programing mode should produce an error.

Response from the master

```
Response        ::= <Acknowledge> |
                    Board <Board Number> : GPIO <GPIO Number> VAL <GPIO Level> <GPIO Dir> |
                    Board <Board Number> : LPO <LPO Number> VAL <LPO Level> |
                    Board <Board Number> : KNOB0 VAL <knob 0 value> |
                    Board <Board Number> : KNOB1 VAL <knob 1 value> |
                    Board <Board Number> : TRACK <Track Number> SPEED <Track Speed> <Track Dir><vehicle status>
                    Board <Board Number> : DCC |
                    Board <Board Number> : ANA |
                    Board <Board Number> : AUT <Number> <Identifier> ON | AUT <Number> <Identifier> OFF |
                    Board <Board Number> : <Memory content>
Memory content  ::= (<Automation number>,<Data index>,<Data>)<Memory Content>| <empty>
vehicle status  ::= ONTRACK | OFFTRACK
Acknowledge     ::= Board <Board Number> : <Error Message>
```

5.9 Protocol on CAN bus

Each data exchange on the CAN bus is made up of a frame containing the sender identifier (address value defined with the switch) and a set of data grouped on 8 bytes (Data Field); the other frame information is of no importance here. These fields are described below:

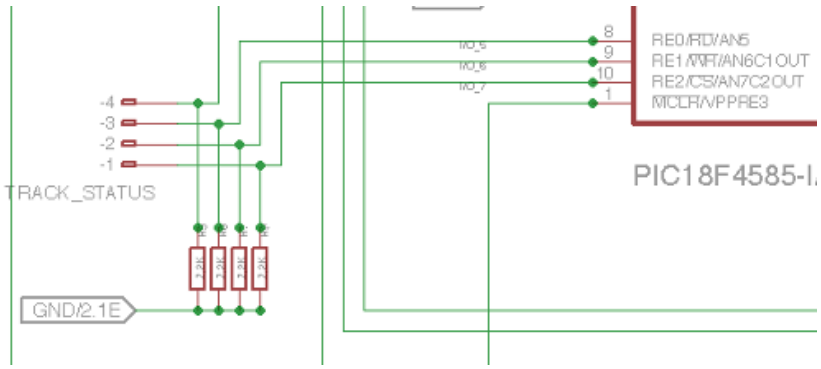
```
SOF (START OF FRAME) : 1 BIT.
Frame start field always equal to 0.

IDENTIFIER : 11 BITS.
Identifies the message sender.

RTR (REMOTE TRANSMISSION REQUEST) : 1 BIT.
Usually 0 except in the case of a request frame.

COMMAND : 6 BITS.
Contains the DLC (Data Length Code), the length of data transmitted in of bytes. For
example, if 4 bytes of data: DLC = 001000.

DATA : FROM 0 TO 8 BYTES.
```

```
CRC (CYCLIC REDUNDANCY CHECK) : 16 BITS.
A calculation algorithm is used to check for transmission errors.

ACK (ACKNOWLEDGE) : 2 BITS.
Acknowledges whether the frame has been read by a node.

EOF (END OF FRAME) : 7 BITS.
Indicates the end of message transmission, 7 bits to 1: 1111111.
```

The data exchanged on the CAN bus is either text between two 8-byte frames (header 0xCC and footer 0xDD) or compressed data between two 8-byte frames (header 0xEE and footer 0xFF).

5.10 Output display and knob controls

We can connect 2 knobs and an output display of 6 7-segments LEDS compliant with TM1637 protocol on track status GPIO



- GPIO 4 is used for KNOB0 where a value between 0 and 5V must be read.
- GPIO 3 is used for KNOB1 where a value between 0 and 5V must be read.
- GPIO 2 is used for CLK (or SCK) line.
- GPIO 1 is used for DIO (or SDA) line.

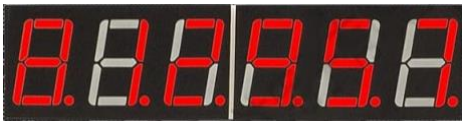
On the SPEED and INERTIA command or program, the numerical value can be replaced by KNOB0 or KNOB1, in which case the value read when the command is executed on GPIO 4 or 3 is used.

KNOB0 has a value between -15 and 15, where -15 is 0V and 15 is 5V
KNOB1 has a value between 0 and 100, where 0 is 0V and 100 is 5V

Negative value used on SPEED reverses the running direction. The lowest inertia value corresponds to the shortest time needed to change speed.

Warning: A FORW command with a negative value will result in a BACK side and a BACK command with a negative value will result in a FORW command

The corresponding values are sent to the 7-segments LEDS display



The left side shows the value of KNOB0 and the right side of KNOB1
In MANUAL mode, speed and inertia have an immediate action on all the tracks
.

To calibrate the knobs, turn the min to max values of the two knobs and run the CALIB command. Calibration is saved in EEPROM**. Need to be done once after firmware update**.

Command

Programming DCC mode on a board

Description	This command allows you to configure a board to operate at the NMRA DCC standard. This action remains memorized when the power is off
Syntax	`PROG <Board Number> DCC`
Response	`No response`
Example	To configure board number 5 in DCC mode: **`PROG 5 DCC`**

Programming ANA mode on a board

Description	This command allows you to configure a board to operate in analogic mode using PWM on track for speed setting. This action remains memorized when the power is off
Syntax	`PROG <Board Number> ANA`
Response	`No response`
Example	To configure board number 5 in ANA mode: **`PROG 5 ANA`**

Programming AUTOMATIC mode on a board

Description	This command reactivates all AUTOFF-type commands and disables track control from the speed and inertia knobs.
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> <Event> ACT AUTOMATIC`
Response	`No response`
Example	To configure board number 2 in AUTOMATIC mode based on GPIO 3 status **`PROG 2 AUT MC AUTON BOARD 2 GPIO 3 0 ACT AUTOMATIC`**

Programming MANUAL mode on a board

Description	This command deactivates all AUTOFF-type commands and enables track control from the speed and inertia knobs. MANUAL0 also allows controlling only channel 0; otherwise, all four channels will be controlled by the knobs in this mode.
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> <Event> ACT MANUAL`
Response	`No response`
Example	To configure board number 2 in MANUAL mode based on GPIO 3 status **`PROG 2 AUT MC AUTON BOARD 2 GPIO 3 0 ACT MANUAL`**

Initialize a GPIO as output on a board

Description	This command allows you to configure a GPIO in output mode. This action remains memorized when the power is off (By default, the 4 GPIO are in input mode). Default value in output mode is 1.
Syntax	`PROG <Board Number> GPIO <GPIO Number> OUT`
Response	`No response`
Example	To configure GPIO 3 on board 5 in output mode **`PROG 5 GPIO 3 OUT`**

Initialize a GPIO as input on a board

Description	This command allows you to configure a GPIO in input mode (default mode). This action remains memorized when the power is off In input mode, the GPIO counts level changes from 0 to 255. To reset this counter to 0, simply initialize the GPIO to 0 as if it were an output GPIO, and the counter will be reset to zero.
Syntax	`PROG <Board Number> GPIO <GPIO Number> IN`
Response	`No response`
Example	To configure GPIO 3 on board 5 in input mode **`PROG 5 GPIO 3 IN`**

Program an automatic action on a GPIO driven by a timer

Description	This command allows you to program an automatic change on GPIO output level when a timer is triggered. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TIMER <Timer Number> ACT GPIO <GPIO Number> <GPIO Level>`
Response	`No response`
Example	To automatically switch the level of GPIO 3 of board 5 to level 1 when the TIMER 2 of board 2 is triggered **`PROG 5 AUT Test AUTOFF BOARD 2 TIMER 2 ACT GPIO 3 1`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test AUTOFF BOARD 2 TIMER 2`** `=> when TIMER 2 of board 2 is triggered. This action is not active in manual mode` **`ACT GPIO 3 1`** `=> GPIO 3 of board 5 is set to 1`

Program an automatic action on a LPO driven by a timer

Description	This command allows you to program an automatic change on LPO output level when a timer is triggered. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TIMER <Timer Number> ACT LPO <LPO Number > <LPO Level>`
Response	`No response`
Example	To automatically switch the level of GPIO 3 of board 5 to level 1 when the TIMER 2 of board 2 is triggered **`PROG 5 AUT Test AUTON BOARD 2 TIMER 2 ACT LPO 3 1`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test AUTON BOARD 2 TIMER 2`** `=> when TIMER 2 of board 2 is triggered. This action is active in manual mode` **`ACT LPO 3 1`** `=> LPO 3 of board 5 is set to 1`

Program an automatic action on a track driven by a timer

Description	This command allows you to program an automatic change of track speed and direction when a timer is triggered. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TIMER <TIMER Number> ACT TRACK <Track Number> SPEED <Track Speed> <Track Dir> INERTIA <Inertia>`
Response	`No response`
Example	To slowly change speed and direction of track 2 of board 5 to speed 10 and travel forward when the TIMER 2 of board 2 is triggered **`PROG 5 AUT Test1 AUTOFF BOARD 2 TIMER 2 ACT TRACK 2 SPEED 10 FORW INERTIA 15`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test1 AUTOFF BOARD 2 TIMER 2`** `=> when TIMER 2 of board 2 is triggered. This action is not active in manual mode` **`ACT TRACK 2 SPEED 10 FORW INERTIA 15`** `=> set track 2 for board 5 at speed 10 forward, 15 steps inertia to change speed`
Comment	**Speed and/or Inertia value(s) could be replaced by KNOB0 or KNOB1, in this case the value of the knob is read when the action is performed**

Program on a timer driven by a timer

Description	This command allows you to program a timer when a timer (the same or another) is triggered. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TIMER <TIMER Number> ACT TIMER <TIMER Number> <TIMER delay>`
Response	`No response`
Example	To set TIMER 3 on board 5 with a delay of 100 when the TIMER 2 of board 2 is triggered **`PROG 5 AUT Test AUTOFF BOARD 2 TIMER 2 ACT TIMER 3 100`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test AUTOFF BOARD 2 TIMER 2`** `=> when TIMER 2 of board 2 is triggered. This action is not active in manual mode` **`ACT TIMER 3 100`** `=> TIMER 3 is set up at value 100`

Turn On or Off an automation driven by a timer

Description	This command allows you to turn on or off an automation when a timer (the same or another) is triggered. This action remains memorized when the power is off

Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TIMER <TIMER Number> ACT AUTOFF <Identifier>`
	
	`PROG <Board Number> AUT <Identifier> BOARD <Board Number> TIMER <TIMER Number> ACT AUTON <Identifier>`

Response	`No response`
Example	To turn on automation Test1 on board 5 when the TIMER 2 of board 2 is triggered **`PROG 5 AUT Test AUTOFF BOARD 2 TIMER 2 ACT AUTON Test1`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test AUTOFF BOARD 2 TIMER 2`** `=> when TIMER 2 of board 2 is triggered. This action is not active in manual mode` **`ACT AUTON Test1`** `=> turn on Test1`

Program an automatic action on a GPIO driven by a change of level on a GPIO

Description	This command allows you to program an automatic change on GPIO output level when the level of a GPIO input has changed on a board. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> GPIO <GPIO Number> <GPIO Level> ACT GPIO <GPIO Number> <GPIO Level>`
Response	`No response`
Example	To automatically switch the level of GPIO 3 of board 5 to level 1 when the GPIO 2 of board 2 changes to level 0 **`PROG 5 AUT Test AUTOFF BOARD 2 GPIO 2 0 ACT GPIO 3 VAL 1`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test AUTOFF BOARD 2 GPIO 2 0`** `=> when GPIO 2 of board 2 is set to 0. This action is not active in manual mode` **`ACT GPIO 3 1`** `=> GPIO 3 of board 5 is set to 1`

Program an automatic action on a LPO driven by a change of level on a GPIO

Description	This command allows you to program an automatic change on LPO output level when the level of a GPIO input has changed on a board. This action remains memorized when the power is off.
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> GPIO <GPIO Number> <GPIO Level> ACT LPO <LPO Number> VAL <LPO Level>`
Response	`No response`
Example	To automatically switch the level of LPO 3 of board 5 to level 1 when the GPIO 1 of board 2 changes to level 0 **`PROG 5 AUT Test AUTOFF BOARD 2 GPIO 1 0 ACT LPO 3 VAL 1`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test AUTOFF BOARD 2 GPIO 1 VAL 0`** `=> when GPIO 1 of board 2 is set to 0. This action is not active in manual mode` **`ACT LPO 3 1`** `=> LPO 3 of board 5 is set to 1`

Program an automatic action on a track driven by a level change on a GPIO

Description	This command allows you to program an automatic change of track speed and direction when the level of a GPIO input has changed on a board. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> GPIO <GPIO Number> <GPIO Level> ACT TRACK <Track Number> SPEED <Track Speed> <Track Dir> INERTIA <Inertia>`
Response	`No response`
Example	To quickly change speed and direction of track 2 of board 5 to speed 10 and travel forward when the GPIO 1 of board 2 changes to level 0 and reverse when the GPIO 1 of board 2 changes to level 1 **`PROG 5 AUT Test1 AUTOFF BOARD 2 GPIO 1 0 ACT TRACK 2 SPEED 10 FORW INERTIA 0`** **`PROG 5 AUT Test2 AUTOFF BOARD 2 GPIO 1 1 ACT TRACK 2 SPEED 10 BACK INERTIA 0`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test1 AUTOFF BOARD 2 GPIO 1 0`** `=> Automation Test1 when GPIO 1 of board 2 is set to 0. Not active in manual mode` **`AUT Test2 AUTOFF BOARD 2 GPIO 1 1`** `=> Automation Test2 when GPIO 1 of board 2 is set to 1. Not active in manual mode` **`ACT TRACK 2 SPEED 10 FORW INERTIA 0`** `=> set track 2 for board 5 at speed 10 forward, immediate change` **`ACT TRACK 2 SPEED 10 BACK INERTIA 0`** `=> set track 2 for board 5 at speed 10 backward, immediate change`

Program an automatic action on a loco (DCC) driven by a level change on a GPIO

Description	This command allows you to program an automatic change of loco setting when the level of a GPIO input has changed on a board. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> GPIO <GPIO Number> <GPIO Level> ACT DCC <DCC Standard Command>`
Response	`No response`
Example	To speed up loco 3 at speed "0xF" forward on board 8 when the GPIO 1 of board 2 changes to level 0 **`PROG 8 AUT Test AUTOFF BOARD 2 GPIO 1 0 ACT DCC 0x03 0x6F`**
Detail	**`PROG 8`** `=> programming board number 8` **`AUT Test AUTOFF BOARD 2 GPIO 1 0`** `=> Automation Test when GPIO 1 of board 2 is set to 0. Not active in manual mode.` **`ACT DCC 0x03 0x6F`** `=> to speed up loco "0x3" at speed "0xF" forward`

Program on a timer driven by a level change on a GPIO

Description	This command allows you to program a timer when the level of a GPIO input has changed on a board. This action remains memorized when the power is off

Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status>`
	`BOARD <Board Number> GPIO <GPIO Number> <GPIO Level>`
	`ACT TIMER <TIMER Number> <TIMER delay>`

Response	`No response`
Example	To set TIMER 3 on board 5 with a delay of 100 when the GPIO 1 of board 2 changes to level 0 **`PROG 5 AUT Test1 AUTOFF BOARD 2 GPIO 1 0 ACT TIMER 3 VAL 100`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test1 AUTOFF BOARD 2 GPIO 1 0`** `=> Automation Test1 when GPIO 1 of board 2 is set to 0. Not active in manual mode.` **`ACT TIMER 3 100`** `=> TIMER 3 is set up at value 100`

Turn on or off an automation by a level change on a GPIO

Description	This command allows you to turn on or off an automation when the level of a GPIO input has changed on a board. This action remains memorized when the power is off

Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> GPIO <GPIO Number> <GPIO Level> ACT AUTOFF <Identifier>`
	
	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> GPIO <GPIO Number> <GPIO Level> ACT AUTON <Identifier>`

Response	`No response`
Example	To turn off Test2 when the GPIO 1 of board 2 changes to level 0 **`PROG 5 AUT Test1 AUTOFF BOARD 2 GPIO 1 0 ACT AUTOFF Test2`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test1 AUTOFF BOARD 2 GPIO 1 0`** `=> Automation Test1 when GPIO 1 of board 2 is set to 0. Not active in manual mode` **`ACT AUTOFF Test2`** `=> turn off Test2`

Program an automatic action on a GPIO driven by a change of vehicle presence on a track

Description	This command allows you to program an automatic change on GPIO output when the presence of a vehicle has changed on a track. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TRACK <Track Number> STA <Vehicle Status> ACT GPIO <GPIO Number> <GPIO Level>`
Response	`No response`
Example	To switch the level of GPIO 3 of board 5 to 1 when a vehicle is on track 1 of board 2 and to 0 when a vehicle is no more on track 1 of board 2 **`PROG 5 AUT Test1 AUTOFF BOARD 2 TRACK 1 STA ONTRACK ACT GPIO 3 1`** **`PROG 5 AUT Test2 AUTOFF BOARD 2 TRACK 1 STA OFFTRACK ACT GPIO 3 0`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test1 AUTOFF BOARD 2 TRACK 1 STA ONTRACK`** `=> Automation Test1 when a vehicle is on track 1 of board 2. Not active in manual mode` **`AUT Test2 AUTOFF BOARD 2 TRACK 1 STA OFFTRACK`** `=> Automation Test2 when a vehicle is no more on track 1 of board 2. Not active in manual mode` **`ACT GPIO 3 1`** `=> GPIO 3 of board 5 is set to 1`

Program an automatic action on a LPO driven by a change of vehicle presence on a track

Description	This command allows you to program an automatic change on LPO output when the presence of a vehicle has changed on a track. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TRACK <Track Number> STA <Vehicle Status> ACT LPO <LPO Number> <LPO Level>`
Response	`No response`
Example	To switch the level of LPO 3 of board 5 to 1 when a vehicle is on track 1 of board 2 and to 0 when a vehicle is no more on track 1 of board 2 **`PROG 5 Test1 AUTOFF AUT BOARD 2 TRACK 1 STA ONTRACK ACT LPO 3 1`** **`PROG 5 Test2 AUTOFF AUT BOARD 2 TRACK 1 STA OFFTRACK ACT LPO 3 0`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test1 AUTOFF BOARD 2 TRACK 1 STA ONTRACK`** `=> Automation Test1 when a vehicle is on track 1 of board 2. Not active in manual mode` **`AUT Test2 AUTOFF BOARD 2 TRACK 1 STA OFFTRACK`** `=> Automation Test2 when a vehicle is no more on track 1 of board 2. Not active in manual mode` **`ACT LPO 3 1`** `=> LPO 3 of board 5 is set to 1` **`ACT LPO 3 0`** `=> LPO 3 of board 5 is set to 0`

Program an automatic action on a track driven by a change of vehicle presence on a track

Description	This command allows you to program an automatic change of track speed and direction when the presence of a vehicle has changed on a track. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TRACK <Track Number> STA <Vehicle Status> ACT TRACK <Track Number> SPEED <Track Speed> <Track Direction> INERTIA <Inertia>`
Response	`No response`
Example	To change the speed and direction of track 3 to 10 and forward when a vehicle is on track 1 of board 2 **`PROG 5 AUT Test AUTOFF BOARD 2 TRACK 1 STA ONTRACK ACT TRACK 3 SPEED 10 DIR FORW INERTIA 5`**
Detail	**`PROG 5`** `=> programming board number 5` **`AUT Test AUTOFF BOARD 2 TRACK 1 STA ONTRACK`** `=> Automation Test when a vehicle is on track 1 of board 2. Not active in manual mode` **`ACT TRACK 3 SPEED 10 FORW INERTIA 5`**`=> set speed of track 3 to 10 forward, 5 steps inertia to change speed`
Comment	**Speed and/or Inertia value(s) could be replaced by KNOB0 or KNOB1, in this case the value of the knob is read when the action is performed**

Program an automatic action on a loco (DCC) by a change of vehicle presence on a track

Description	This command allows you to program an automatic change of loco setting when the presence of a vehicle has changed on a track. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TRACK <Track Number> STA <Vehicle Status> ACT DCC <DCC Standard Command>`
Response	`No response`
Example	To speed up loco 3 at speed "0xF" forward on board 8 when a vehicle is on track 1 of board 2 **`PROG 8 AUT Test AUTOFF BOARD 2 TRACK 1 STA ONTRACK ACT DCC 0x03 0x6F`**
Detail	**`PROG 8`** `=> programming board number 8` **`AUT Test AUTOFF BOARD 2 TRACK 1 STA ONTRACK`** `=> Automation Test when a vehicle is on track 1 of board 2. Not active in manual mode` **`ACT DCC 0x03 0x6F`** `=> to speed up loco "0x3" at speed "0xF" forward`

Program on a timer driven by a change of vehicle presence on a track

Description	This command allows you to program a timer when the presence of a vehicle has changed on a track. This action remains memorized when the power is off
Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TRACK <Track Number> STA <Vehicle Status> ACT TIMER <TIMER Number> <TIMER delay>`
Response	`No response`
Example	To set TIMER 3 on board 5 with a delay of 100 when a vehicle is on track 1 of board 2 **`PROG 8 AUT Test AUTOFF BOARD 2 TRACK 1 STA ONTRACK ACT TIMER 3 100`**
Detail	**`PROG 8`** `=> programming board number 8` **`AUT Test AUTOFF BOARD 2 TRACK 1 STA ONTRACK`** `=> Automation Test when a vehicle is on track 1 of board 2. Not active in manual mode` **`ACT TIMER 3 100`** `=> TIMER 3 is set up at value 100`

Turn on or off an automation by a change of vehicle presence on a track

Description	This command allows you to turn on or off an automation when the presence of a vehicle has changed on a track. This action remains memorized when the power is off

Syntax	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TRACK <Track Number> STA <Vehicle Status> ACT AUTOFF <Identifier>`
	
	`PROG <Board Number> AUT <Identifier> <Manual Status> BOARD <Board Number> TRACK <Track Number> STA <Vehicle Status> ACT AUTON <Identifier>`

Response	`No response`
Example	To turn off Test1 when a vehicle is on track 1 of board 2 **`PROG 8 AUT Test AUTOFF BOARD 2 TRACK 1 STA ONTRACK ACT AUTOFF Test1`**
Detail	**`PROG 8`** `=> programming board number 8` **`AUT Test AUTOFF BOARD 2 TRACK 1 STA ONTRACK`** `=> Automation Test when a vehicle is on track 1 of board 2. Not active in manual mode` **`ACT AUTOFF Test1`** `=> Turn off Test1`

Delete an automation on a board

Description	This command allows you to remove an automation on a board. This action remains memorized when the power is off
Syntax	`PROG <Board Number> DEL <Number>`
Response	`No response`
Example	To remove automation 3 on board 5 (List of automation could be retrieve using the Automation list command COM <Board Number> AUTLIST) **`PROG 5 DEL 3`**
Warning	Deleting an automation changes the number of the other automations, so you need to perform once again an AUTLIST command to know the new numbers

Force the value on a GPIO on a board

Description	This command allows you to set the level of a GPIO. This action remains valid until the board is stopped or an automatic action is triggered. When GPIO is in input mode, this command reset the counter when set to 0
Syntax	`COM <Board Number> GPIO <GPIO Number> <GPIO Level>`
Response if GPIO in output mode	`No response`
Example	To set GPIO 3 at level 1 on board 4 **`COM 4 GPIO 3 1`**

Create a timer on a board

Description	This command allows you to set a timer. This action remains valid until the board is stopped or the timer is triggered
Syntax	`COM <Board Number> TIMER <Timer Number> <Timer delay>`
Response if GPIO in output mode	`No response`
Example	To set TIMER 3 with a delay of 10 on board 4 **`COM 4 TIMER 3 10`**

Force the value on a LPO on a board

Description	This command allows you to set the level of a LPO. This action remains valid until the board is stopped or an automatic action is triggered
Syntax	`COM <Board Number> LPO <LPO Number> <LPO Level>`
Response	`No response`
Example	To set LPO 3 at level 1 on board 4 **`COM 4 LPO 3 1`**

Turn on an automation on a board

Description	This command allows you to turn on an automation. This action remains valid until the board is stopped or an automatic action is triggered
Syntax	`COM <Board Number> AUTON <Identifier>`
Response	`No response`
Example	To turn on automation Test On board 4 **`COM 4 AUTON Test`**

Turn off an automation on a board

Description	This command allows you to turn off an automation. This action remains valid until the board is stopped or an automatic action is triggered
Syntax	`COM <Board Number> AUTOFF <Identifier>`
Response	`No response`
Example	To turn off automation Test On board 4 **`COM 4 AUTOFF Test`**

Force speed and direction on a track

Description	This command allows you to set the speed and direction on a track. This action remains valid until the board is stopped or an automatic action is triggered
Syntax	`COM <Board Number> TRACK <Track Number> SPEED <Track Speed> <Dir> INERTIA <Inertia>`
Response	`No response`
Response if board in DCC mode	`Board <Board Number> Bad mode`
Example	To set speed at 4 and direction forward on track 2 board 8 **`COM 8 TRACK 2 SPEED 4 FORW INERTIA 2`**
Comment	**Speed and/or Inertia value(s) could be replaced by KNOB0 or KNOB1, in this case the value of the knob is read when the action is performed**

Send a DCC command to a board

Description	This command allows you to send a DCC NMRA command to a board (all tracks). This action remains valid until the board is stopped or an automatic action is triggered
Syntax	`COM <Board Number> DCC <DCC Stand Command>`
Response	`No response`
Response if board in ANA mode	`Board <Board Number> Bad mode`
Example	To send a command to speed up loco 3 at speed “0xF” forward on board 8 **`COM 8 DCC 0x03 0x6F`** To send a command to speed up loco 3 at speed “0xF” backward on board 8 **`COM 8 DCC 0x03 0x4F`**

Request the status of all the GPIOs on a board

Description	This command allows you to get the direction and the level of all the GPIOs on a board
Syntax	`COM <Board Number> GSTAT`

Response	`Board <Board Number> GPIO 0 <GPIO Dir> VAL <GPIO Level> COUNT <GPIO Counter>`
	`Board <Board Number> GPIO 1 <GPIO Dir> VAL <GPIO Level> COUNT <GPIO Counter>`
	`Board <Board Number> GPIO 2 <GPIO Dir> VAL <GPIO Level> COUNT <GPIO Counter>`
	`Board <Board Number> GPIO 3 <GPIO Dir> VAL <GPIO Level> COUNT <GPIO Counter>`
	`Board <Board Number> KNOB0 VAL <knob value>`
	`Board <Board Number> KNOB1 VAL <knob value>`

Example	To get the status of all the GPIO of board 5 **`COM 5 GSTAT`**

Request the status of all the LPOs on a board

Description	This command allows you to get the direction and the level of all the LPOs on a board
Syntax	`COM <Board Number> LSTAT`

Response	`Board <Board Number> LPO 0 VAL <LPO Level>`
	`Board <Board Number> LPO 1 VAL <LPO Level>`
	`Board <Board Number> LPO 2 VAL <LPO Level>`
	`Board <Board Number> LPO 3 VAL <LPO Level>`
	`Board <Board Number> LPO 4 VAL <LPO Level>`
	`Board <Board Number> LPO 5 VAL <LPO Level>`

Example	To get the status of all the LPO of board 5 **`COM 5 LSTAT`**

Request track status on a board

Description	This command allows you to get the direction and the speed and direction of all the tracks on a board if the board is in ANA mode
Syntax	`COM <Board Number> TSTAT`

Response	`Board <Board Number> TRACK 0 SPEED <Track Speed> <Track Dir> <vehicle status>`
	`Board <Board Number> TRACK 1 SPEED <Track Speed> <Track Dir> <vehicle status>`
	`Board <Board Number> TRACK 2 SPEED <Track Speed> <Track Dir> <vehicle status>`
	`Board <Board Number> TRACK 3 SPEED <Track Speed> <Track Dir> <vehicle status>`

Example	To get the status of all the tracks of board 5 **`COM 5 TSTAT`**

Request board status

Description	This command allows you to get the status of a board to know if the running mode is DCC or ANA
Syntax	`COM <Board Number> BSTAT`
Response if DCC mode	`Board <Board Number> DCC`
Response if ANA mode	`Board <Board Number> ANA`
Example	To get the status of the board 5 **`COM 5 BSTAT`**

Request the list of actions programmed on a board

Description	This command allows you to get all the Automations stored in a board
Syntax	`COM <Board Number> AUTLIST`
Response if no data	`No response`
Response for each automation	`Board <Board Number> <Number> <Identifier> ON` `Board <Board Number> <Number> <Identifier> OFF`
Example	To get the list of actions programmed on board 5 **`COM 5 AUTLIST`**

Request a dump of memory on a board

Description	This command allows you to get the content of Automations stored in a board in binary mode (reserved for PC application)
Syntax	`COM <Board Number> DUMP`
Response if no data	`No response`
Response for each automation	**`Board <Board Number> <Number> (<Number>,<Index>,<Data>)`**…
Example	To get the dump of board 5 **`COM 5 DUMP`**

Request a calibration of knobs

Description	This command is used to save the min and max values of knobs 0 and 1. To do this, turn both knobs to the min value and then to the max value and execute this command.
Syntax	`CALIB <Board Number>`
Response	`No response`
Example	To calibrate knobs of board 5 **`CALIB 5`**

5.11 Global command

Stop all

Description	This command will stop the activity on all the boards of the network
Syntax	`STOP`
Response	`No response`
Example	To stop all **`STOP`**

Run all

Description	This command will start the activity on all the boards of the network
Syntax	`RUNALL`
Response	`No response`
Example	To run all **`RUNALL`**

Run a specific board

Description	This command will start the activity on a specific board
Syntax	`RUN <Board Number>`
Response	`No response`
Example	To run board 5 **`RUN 5`**

Reset all automation of a specific board

Description	This command will delete all the automation on a specific board. **Warning this action is performed immediately without any notification!**

Syntax	`RESET <Board Number>`
Response	`No response`
Example	To reset board 5 **`RESET 5`**

Inconsistent programming

If an impossible automatic action is triggered, such as setting the speed and direction on the track of a board in DCC mode, or configuring a locomotive on a map in ANA mode, the action is simply ignored and a message is sent to the serial console:

- Unknown token
- Number missing
- Incomplete request
- Bad number
- Mode is missing
- Bad GPIO number
- Bad TIMER number
- Bad TIMER value
- Bad LPO number
- Bad Automation status
- Bad Automation name
- Automation name missing
- Bad GPIO direction
- Bad GPIO level
- Bad Low Power Output level
- Bad track speed
- Bad track direction
- Bad track number
- Bad mode
- Wrong board number
- No more automation available
- Space missing
- Identifier is too long
- Automation already defined
- Wrong automation number
- Bad inertia value
- Bad user mode