**CAN Stuff –**

**Pedal Box** – Will transmit to address 0x3C

Byte 1 – torque request (0 – 100) in units of % (int)

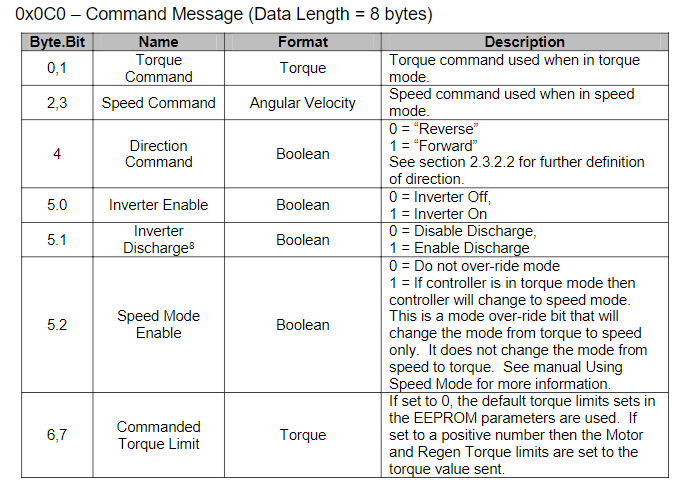
Byte 2 – Reserved in case we need higher precision in torque request

Byte 3 - Brake pressure (0 – 100) in units of % (int)

Byte 4 – Implausibility event occurred (bool)

**Inverter –**

Send commands to 0x0C0 at least once every 500ms (DO NOT SEND OLD TORQUE REQUESTS, ONLY SEND THEM AS THEY ARE RECEIVED FROM THE PEDAL BOX)



For an example, see Appendix 1.

**Power Distribution Module (PDM) –**

Send commands to 0x28C:

For now just worry about sending commands to turn off and on the coolant pump and fan:

DLC is 1 byte, 0b1XXXXXXX for coolant fan and pump on and 0b0XXXXXXX for fan and pump off. X’s indicate don’t care, they will be masked off at the PDM.

**States:**

**Initial:**

Move to state Standby (state = standby)

**Standby:**

Set PA[3] High

Every time new brake data is received, check to see if PB[3] is LOW AND PA[4] is high AND Brake > 5%,

if conditions met, state = RTD Sound

Check coolant temp (PB[0]) is below a preset maximum, if it has changed, send 0b00000000 to PDM, else send 0b100000000 (only send CAN message if it has gone from below max -> above max or vice versa, don’t want the VCU to send 0b00000000 along the CAN bus every 10ms)

DO NOT have any communication to the inverter

**Ready to Drive Sound:**

Set PB[5] High

Send 0b1XXXXXXX to PDM

Wait 2 Sec

Set PB[5] Low

Send 0x00 (all bytes) to inverter (has a lockout feature where the inverter wont be enabled unless it has been sent a disable command first)

State = Ready to Drive

**Ready to Drive:**

If new torque data from pedal box is available – relay torque request to inverter where:

0% from pedal box = 0 N.m

100% from Pedal box = 140 N.m (would be great if

this max torque could be changed through a variable)

Byte 0 and 1 has torque command, byte 2 and 3 should be 0 as we are not in speed mode, byte 4 should be 1 and byte 5, bit 0 (5.0) should be 1, bit 1 (5.1) should be 0 and bit 2 (5.2 should be 0)

Bytes 6 and 7 should be set to 0.

Monitor Coolant Pressure (PA[1]), if it gets below a preset threshold, set PA[3] LOW and state = Emergency Stop

Monitor Coolant Temp, if it gets above a preset threshold, set PA[3] LOW and state = Emergency Stop

**Emergency Stop:**

Send CAN Message to Inverter – Byte 0 – 3 (inclusive) 0x00, byte 4 - 0xFF, byte 5 – 0x40, byte 6 and 7 0x00

Enter a forever loop {

Check coolant temp (PB[0]) is below a preset maximum, if it has changed, send 0b00000000 to PDM, else send 0b100000000 (only send CAN message if it has gone from below max -> above max or vice versa, don’t want the VCU to send 0b00000000 along the CAN bus every 10ms)

**Interrupts:**

**Falling Edge on Safety Circuit (PA[4]):**

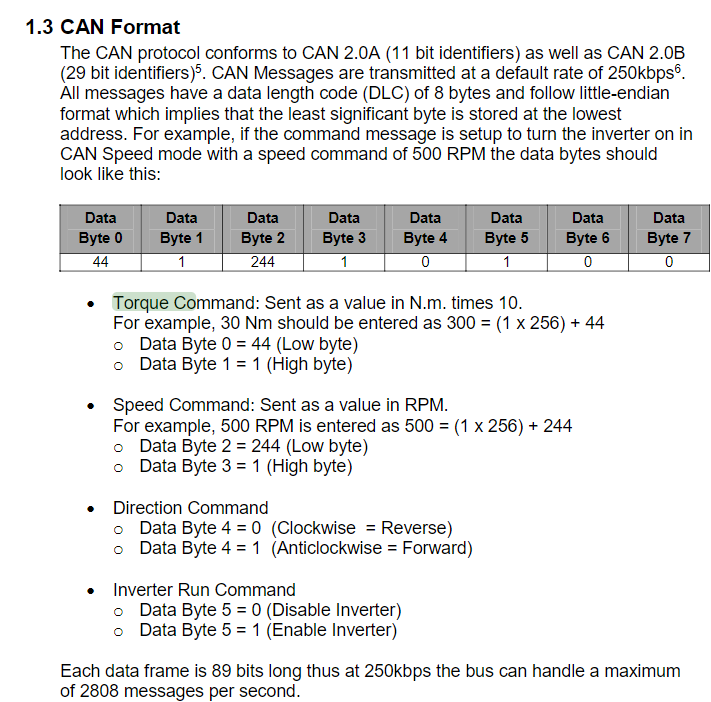
State = Emergency stop

**Timer set to trigger every 3ms (For watchdog):**

Invert PA[2] (Watchdog will reset on a rising edge or falling edge)

**APPENDIX:**

**APPENDIX 1:**



(Page 11 of <https://app.box.com/s/4fb49r9p6lzfz4uwcb5izkxpcwh768vc>)

Pinout

PA10[PA12] CAN TX PA9[PA11] CANRX PB[7] UARTRX (UART incase we need to debug) PB[6] UARTTX (UART incase we need to debug) PB[5] (OUTPUT) Vehicle Activation Sound (High activates buzzer) PB[3] (INPUT) Car Start Button PA[4] (INPUT) Current Shunt on safety circuit (HIGH if there is current going thru safety circuit) PB[0] (INPUT) Coolant Temp (Analog Value) PA[3] (OUTPUT) Safety Circuit Control (A LOW value stops current in the safety circuit, shutting down the car) PA[2] (OUTPUT) Watchdog pin (transmit a rising or falling edge every ~6ms) PA[1] (INPUT) Coolant Pressure (Analog Value)