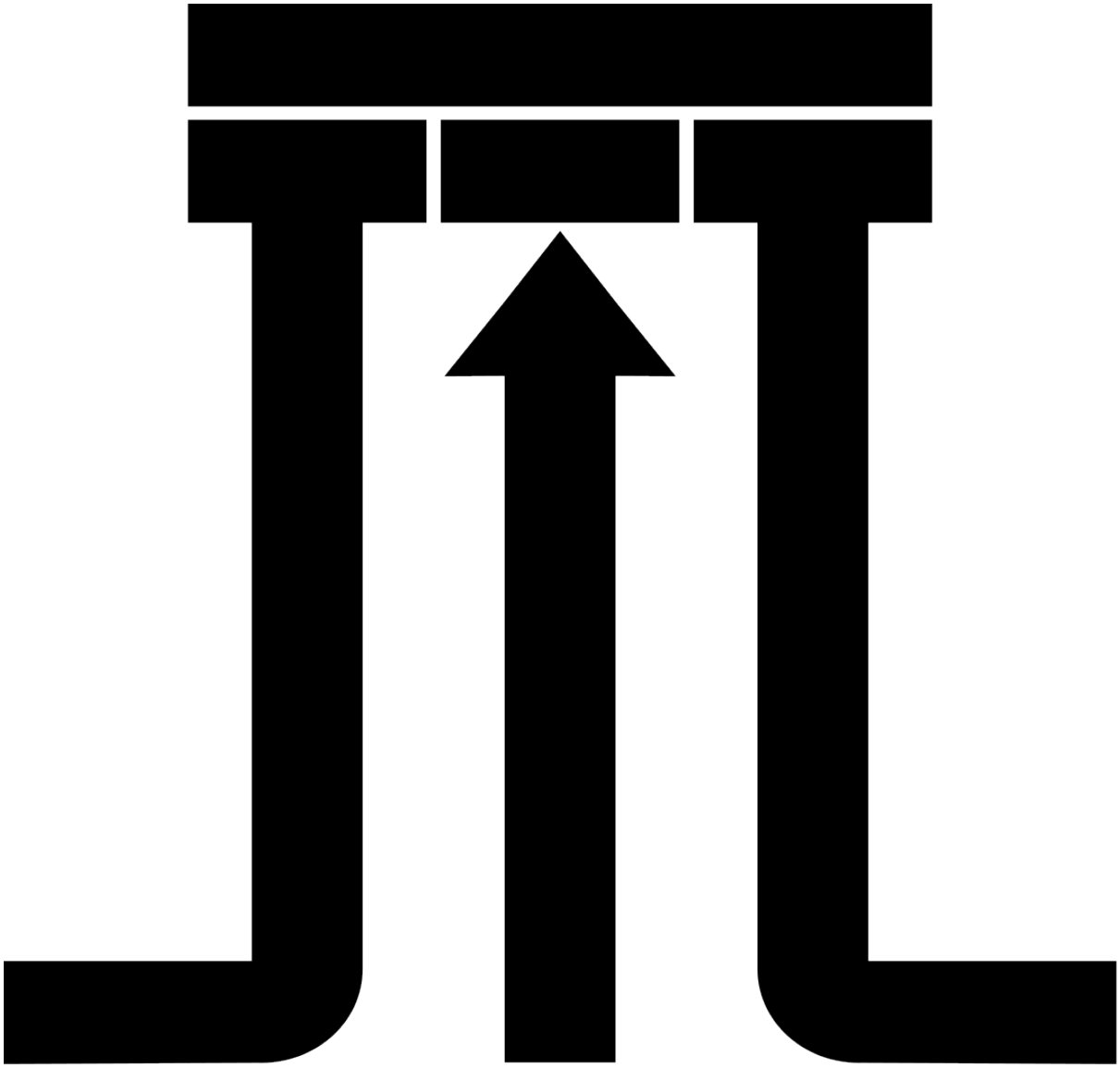


Steinmetz Motor Controllers

Release 1.5

Steinmetz, Inc.

June 03, 2026

SOFTWARE

1 Software Documentation **2**

1.1 Getting Started - Software . . . 2
1.1.1 Validate Interfaces . . . 2
1.1.2 Validate Internal Sensors . . . 2
1.1.3 Write Initial EEPROM values . . . 2
1.1.3.1 Calculating initial i_d & i_q PI gains . . . 3
1.1.3.2 Tuning velocity PI gains . . . 4
1.1.4 Validate Position Sensing . . . 4
1.1.4.1 Tune Resolver Offset . . . 4
1.2 State Machine . . . 5
1.3 CAN Protocol . . . 6
1.3.1 TX (Emitted from Motor Controller) . . . 7
1.3.1.1 Summary . . . 7
1.3.1.2 0x012 (Resolver) . . . 8
1.3.1.3 0x016 (State ID) . . . 9
1.3.1.4 0x01c (EEPROM Set Response) . . . 11
1.3.1.5 0x01d (EEPROM Get Response) . . . 11
1.3.1.6 0x020 (DQ Currents) . . . 12
1.3.1.7 0x021 (PWM Duty Cycles) . . . 13
1.3.1.8 0x022 (DQ PID Integral + DC Current estimate) . . . 15
1.3.1.9 0x023 (Torque/Velocity Feedback) . . . 16
1.3.1.10 0x025 (Electrical Angle) . . . 17
1.3.1.11 0x029 (Switching Misc Data) . . . 18
1.3.1.12 0x031 (Power Data) . . . 19
1.3.1.13 0x306 (GPIO State) . . . 20
1.3.1.14 0x310-0x31B (Temperature Data) . . . 20
1.3.1.15 0x31C (External RTD interface) . . . 22
1.3.1.16 0x31D (Fault Trigger NP Cap Voltages) . . . 22
1.3.1.17 0x31E (Fault Trigger Current/Temp) . . . 24
1.3.1.18 0x31F (Fault Trigger HV Undervoltage) . . . 24
1.3.1.19 0x320 (Current Sense Phase C) . . . 26
1.3.1.20 0x321 (Current Sense Phase B) . . . 27
1.3.1.21 0x322 (Current Sense Phase A) . . . 28
1.3.1.22 0x323 (Voltage Sense) . . . 29
1.3.1.23 0x324 (Current Sense ADC Debug) . . . 30
1.3.2 RX (Received by Motor Controller) . . . 30
1.3.2.1 Summary . . . 30
1.3.2.2 0x019 (Command Packet) . . . 31
1.3.2.3 0x202 (Set DC Current Limit Packet) . . . 32

1.3.2.4 0x01a (EEPROM Set Parameter) . . . 34
1.3.2.5 0x01b (EEPROM Get Parameter) . . . 35
1.4 EEPROM Configuration . . . 36
1.4.1 Fields . . . 36
1.4.2 Reading/Writing to EEPROM . . . 40
1.4.2.1 Using Provided Tool . . . 40
1.4.2.1.1 Reading . . . 40
1.4.2.1.2 Writing . . . 42
1.4.2.1.3 Dumping . . . 42
1.4.2.2 Over CAN . . . 45
1.4.2.2.1 Writing . . . 45
1.4.2.2.2 Reading. . . . 47

2 MC0-500 Hardware Documentation **50**
2.1 Getting Started - Hardware . . . 50
2.1.1 Mounting . . . 50
2.1.2 Quickstart Harnessing . . . 51
2.2 Cooling . . . 52
2.2.1 Fittings . . . 52
2.2.2 Coolant . . . 52
2.2.3 Flow Requirements . . . 53
2.3 Connectors . . . 53
2.3.1 LV Connector . . . 54
2.3.2 LV Connector Pinout . . . 55
2.3.3 Ethernet Connector . . . 55
2.3.4 HV Connectors . . . 56
2.3.5 Grounding . . . 56
2.4 Environmental Sealing . . . 56
2.5 Mechanical . . . 56
2.5.1 I/O . . . 56
2.5.2 Drawing . . . 56
2.5.3 3D CAD . . . 57

3 MC0-250 Hardware Documentation **58**
3.1 MC0-250 Coming Soon . . . 58

Welcome to the **Steinmetz** documentation. This website is designed to be a comprehensive source of truth This site covers everything from high-voltage hardware and assembly procedures to low-level firmware, control software, and testing workflows.

CHAPTER

ONE

SOFTWARE DOCUMENTATION

1.1 Getting Started - Software

This guide should get you from 0 to 1 for getting a motor to spin assuming all of the hardware is setup per *Getting Started - Hardware*.

In addition to setting up the hardware, before proceeding through this guide, it is recommended to understand the fundamentals of how the software works. Consider skimming through these chapters to get a high-level understanding of the system.

- *State Machine*
- *CAN Protocol*
- *EEPROM Configuration*

1.1.1 Validate Interfaces

First, apply LV power to the motor controller through the *LV Connector Pinout*. After a short (<=5 second) delay, the unit will start emitting telemetry packets over the CAN bus. Before attempting to spin under HV power, it is prudent to ensure that all sensors are operational

1.1.2 Validate Internal Sensors

Read the data from the *0x016 (State ID)*. Both `cangaroo` and `wireshark` are excellent tools to use for this purpose on Linux. When the motor controller first turns on, it will be in the Init state (learn more here: *State Machine*). It should then quickly and automatically progress to the ReadyToSwitch state if the temperature, current, and resolver sensors are healthy. Additionally, none of the fault bits should be active.

If one or more fault bits are set, it could indicate that the resolver's external signals are not properly connected, or that there is an issue internal to the unit.

1.1.3 Write Initial EEPROM values

For proper commutation, there are a few EEPROM values for which the default value is unlikely to be appropriate for your usecase. You will almost certainly need to update these values from the default See *EEPROM Configuration* for details on *how* to set configuration values.

Table 1: EEPROM Fields

Field Name	Default	Description	What will happen if it is incorrect?
ResolverOffsetDegrees	0.0	This value should be the electrical angle read by the resolver when the rotor is aligned to phase A of the motor.	Motor may not spin, or if it does, commutation efficiency may be severely degraded
LoopIqPGain	0.0	See below	Improper commutation, or poor performance under high speeds/torques
LoopIqIGain	0.005	See below	Improper commutation, or poor performance under high speeds/torques
LoopIdPGain	0.0	See below	Improper commutation, or poor performance under high speeds/torques
LoopIdIGain	0.005	See below	Improper commutation, or poor performance under high speeds/torques
ResolverPolesPerMotorPole	1.0	This value should be the `motor pole count / resolver pole count`. For example, with a 10 pole motor and a 5 pole resolver, this value should be `2.0`. The sign of this value indicates whether the resolver is expected to read forwards or backwards relative to the motor.	Motor will not spin, reported i_d/i_q/torque values will be incorrect
TorqueConstant	1.0 $\frac{\text{Nm}}{\text{A}_{\text{peak}}}$	This value should be the torque constant from the motor's datasheet (rescaled to $\frac{\text{Nm}}{\text{A}_{\text{peak}}}$)	Torque values to/from the motor will be incorrect
MaxTorqueSlewRate	65.0 $\frac{\text{Nm}}{\text{sec}}$	Maximum allowable change in torque per second	Aggressive motoring/regen action may sag/boost HV DC bus beyond system limits.

1.1.3.1 Calculating initial i_d & i_q PI gains

We have found these to be a good initial guess for PI gains when manually tuning:

$$K_p = 0.8 \cdot \frac{L_{ph}}{0.0002} \qquad K_i = 0.8 \cdot \frac{R_{ph}}{0.0002}$$

where L_{ph} is the motor's phase-to-neutral inductance (in Henries) and R_{ph} is the motor's phase-to-neutral resistance (in Ohms).

The quality of your chosen gains can be evaluated by using the Torque Control mode in the *0x019 (Command Packet)*.

Note

Autotuning, plug and play operation of the motor controller units is coming via a firmware update in 2026.

1.1.3.2 Tuning velocity PI gains

The ideal velocity gains of the system depends on the rotational inertia/damping of the rest of the system. The following *EEPROM Configuration* parameters can be used to tune the velocity response.

Table 2: EEPROM Fields

Field Name	Default	Description	Unit
LoopVelPGain	0.006	P (proportional) gain in the velocity control loop.	$\mathrm{Nm} \cdot \mathrm{s}/\mathrm{E}^\circ$
LoopVelIGain	0.001	I (integral) gain in the velocity control loop.	$\mathrm{Nm}/\mathrm{E}^\circ$
LoopVelDGain	0.0	D (derivative) gain in the velocity control loop.	$\mathrm{Nm} \cdot \mathrm{s}^2/\mathrm{E}^\circ$

The velocity controller is best tuned by setting `LoopVelIGain` to 0 initially and adjusting `LoopVelPGain` by a small increment (e.g by `0.001`) until a sufficiently fast response is achieved between the reported electrical velocity (in *0x025 (Electrical Angle)*) and the desired electrical velocity (in *0x019 (Command Packet)*). A small increment is required as E°/s is a small unit – e.g on a 10 pole pair motor, 10,000 E°/s is equivalent to ~166 mechanical RPM.

Once an appropriate response speed has been achieved, `LoopVelIGain` can be slowly increased to reduce the steady-state error.

Note that, while the current control loop currently runs at 10kHz, the velocity control loop runs at 1kHz.

Note

Autotuning, plug and play operation of the motor controller units is coming via a firmware update in 2026.

1.1.4 Validate Position Sensing

Read back the data from *0x012 (Resolver)* and *0x025 (Electrical Angle)*. Validate that in *0x012 (Resolver)* there are no fault bits (FAULT should equal 0x00).

Manually spin the motor by hand to complete 1 mechanical rotation. The position reported in *0x025 (Electrical Angle)* should increment or decrement depending on the direction of rotation. Ensure that the angle returns to it's initial angle N times, where N is the number of poles in your motor.

1.1.4.1 Tune Resolver Offset

There are a few ways to tune the resolver offset. In the future there will be more robust control algorithms to automatically determine the optimal resolver offset.

For now, it's possible to guess and check it. Set ResolverOffsetDegrees in the *EEPROM Configuration* and apply torque using the *0x019 (Command Packet)*. If you see rotation begin to occur, you are close.

You should be able to tell it's roughly tuned when applying a torque causes desired rotation and the DQ currents are reasonably stable.

Complete!

Once you're spinning with reasonable PI gains and a working resolver offset, you're off to the races!

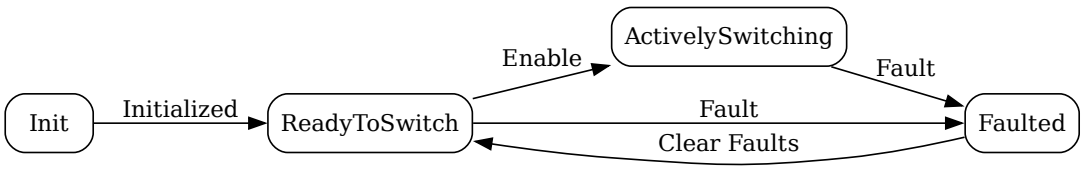
1.2 State Machine

The firmware is based on a custom state machine designed in house. This page will discuss the overall system state machine.

Table 3: States

State Number	State	Description
0	Init	Awaiting Initialization of all Required Sensors and Interfaces
1	ReadyToSwitch	The controller is ready to actuate. Motor controller is disabled but has no faults.
2	ActivelySwitching	The controller is actively switching in some standard control mode.
3	Faulted	Some fault has occured and the motor controller should be in a disabled state.
4-7, 17-18	Reserved	These are reserved debug states.

The high-level state machine is depicted below. Detailed transitions descriptions are found in the table below.



The transitions are described in greater detail here.

Table 4: Transitions

Transition Name	Description
Initialized	All sensors are setup and providing good data. Current Sense, Temperature Sense and Position Sense.
Enable	Valid Enable command is provided over CAN with a valid actuation command request.
Fault	The motor controller has detected a critical fault. Detailed fault descriptions are found below.
Clear Faults	The motor controller has received over CAN a ClearFaults message and has determined that the previous fault case is no longer applicable.

The faults are described here.

Table 5: Faults

Fault Name	Description
Over-Temperature	Detected over temperature on transistor. This is potentially hardware damaging.
Over-Current	Detected over current on phase output. This is potentially hardware damaging.
Missed Heartbeat	Failed to receive command packet in last 1s. Only applicable in ActivelySwitching state.
Current Comms Loss	Lost communication to current sense ADCs.
Temperature Comms Loss	Lost communication to thermistor sense ADCs
Neutral Point Balance Loss	The controller has failed to maintain proper balance across the neutral point, risking overvoltage stress to the transistors.

1.3 CAN Protocol

This page describes the CAN protocol the user can use to interact with the motor controller. This includes TX and RX. The Steinmetz motor controller uses Standard CAN IDs. In future versions, the ability to customize the communication protocol will be possible.

Note

By default all words are packed as little-endian.

1.3.1 TX (Emitted from Motor Controller)

1.3.1.1 Summary

Table 6: TX CAN Messages

CAN ID	CAN Message Name	Brief Description
0x012	Resolver	Raw data from resolver to digital converter.
0x016	State ID	The state of the state machine see: *State Machine*
0x01c	EEPROM Set Response	Response/ACK when writing to EEPROM.
0x01d	EEPROM Get Response	Response/ACK when reading from EEPROM.
0x020	DQ Currents	Computed DQ current data.
0x021	PWM Duty Cycles	Phase A, B and C switching duty cycles.
0x022	DQ PID Integral	Computed integral term for DQ PID loop.
0x023	Torque/Velocity Feedback	Torque request and feedback, velocity setpoint and feedback.
0x025	Electrical Angle	Processed information on Electrical Angle
0x029	Switching Misc Data	Extra telemetry data from switching.
0x031	Power Data	Info about the amount of power flowing through the device.
0x306	GPIO State	External GPIO Data
0x310-0x31B	Temperature Data	Individual Thermistor Data
0x31D	Fault Trigger NP Cap Voltages	NP cap voltage readings that triggered latched overvoltage faults. Only sent when one of those faults is latched.
0x31E	Fault Trigger Current/Temp	Current and temperature readings that triggered latched current/overtemp faults. Only sent when a relevant fault is latched.
0x31F	Fault Trigger HV Undervoltage	HV bus voltage reading that triggered the HV undervoltage fault. Only sent when that fault is latched.
0x320-0x324	Current and Voltage Sensor Readings	Phase Current and Voltage Sense Readings

1.3.1.2 0x012 (Resolver)

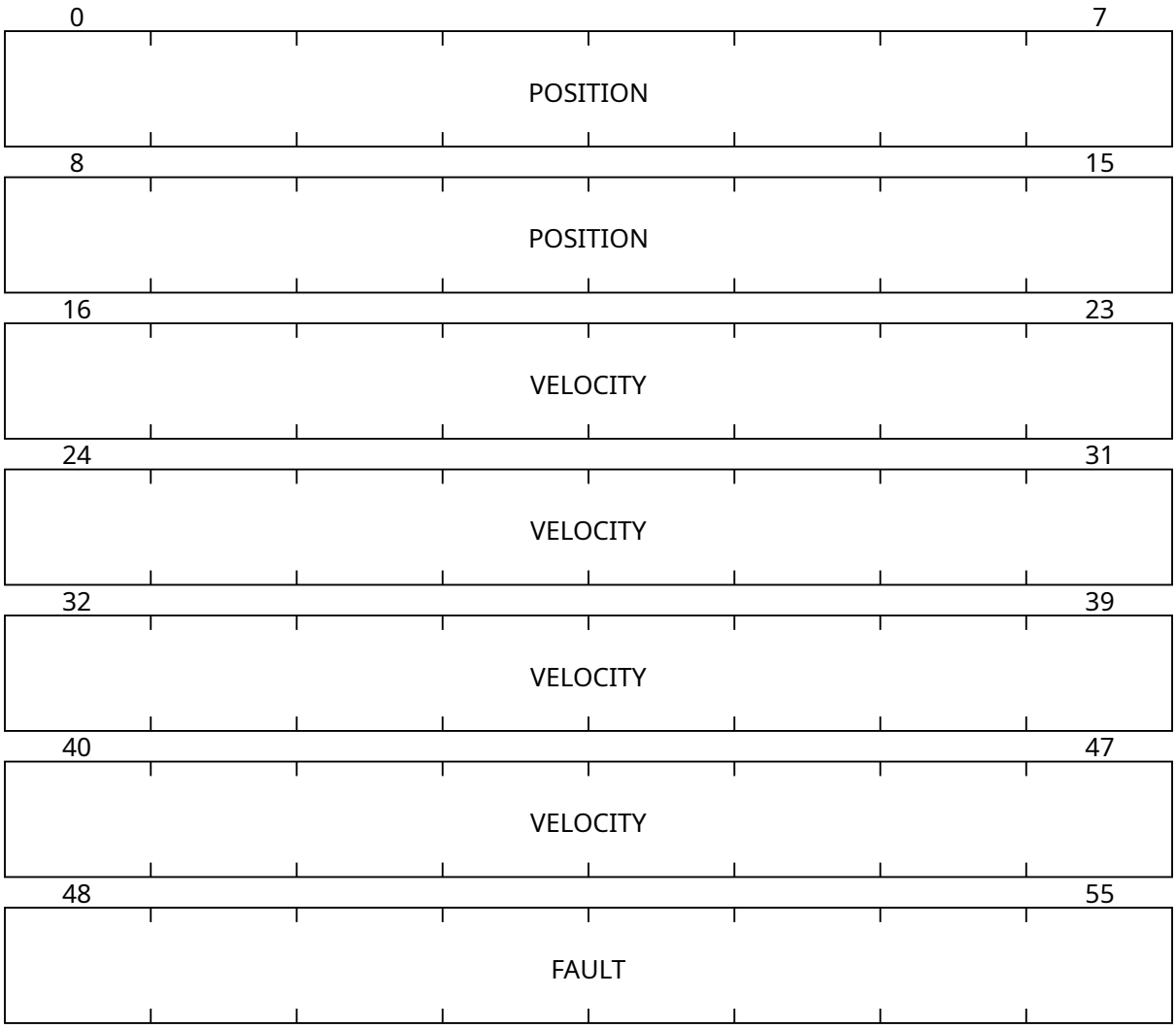


Table 7: Resolver Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
POSITION	Resolver Position in Degrees	100	u16
VELOCITY	Resolver Rotational Velocity in Degrees per second	100	i32
FAULT	The Fault bits in the resolver to digital converter. The resolver used is the AD2S1210, see AD2S1210 datasheet, for detailed description of the fault register.	N/A	u8

1.3.1.3 0x016 (State ID)

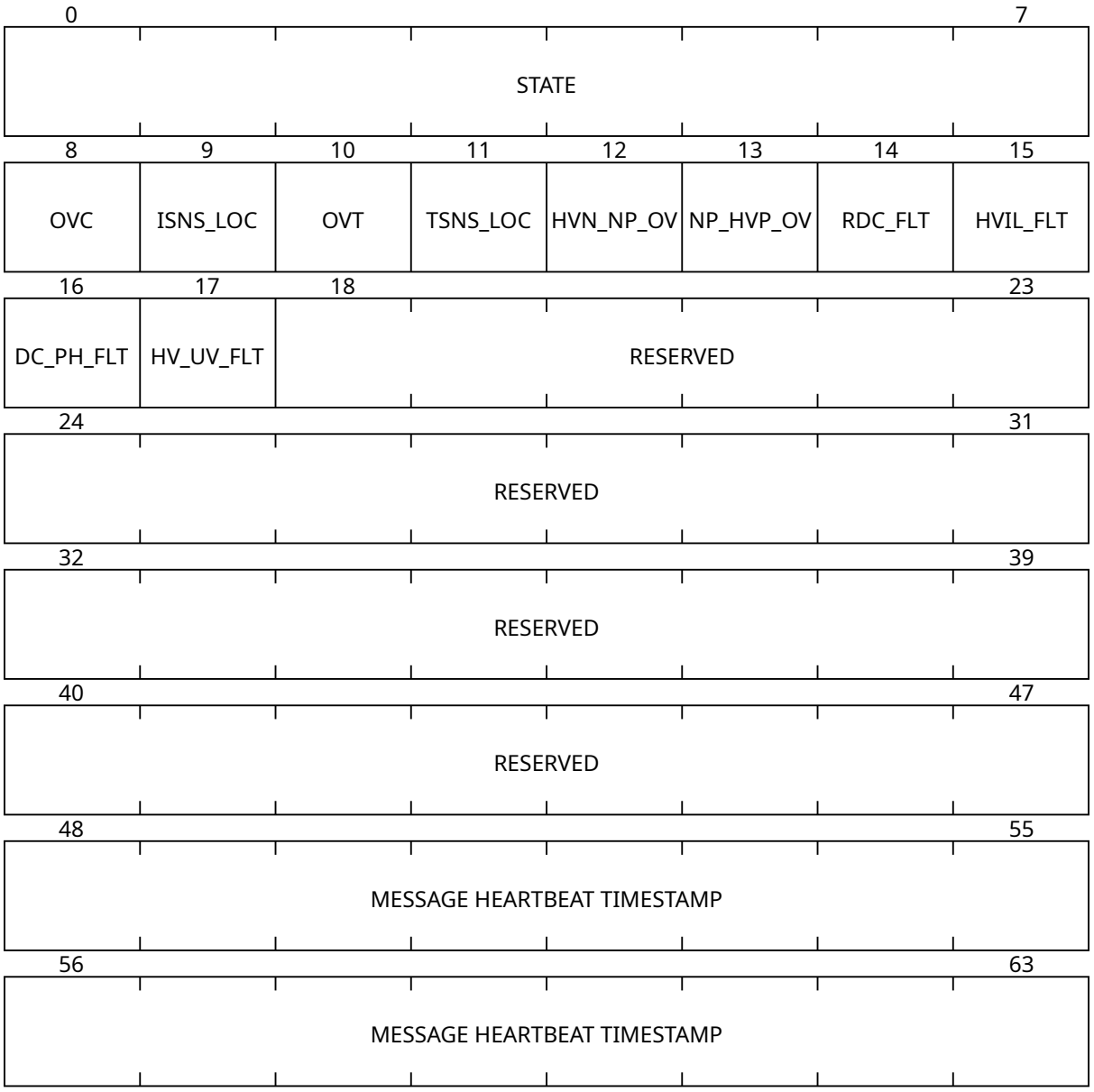


Table 8: State ID Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
STATE	State ID. See *State Machine* for details	N/A	u8
OVC	Phase current magnitude exceeded configured overcurrent threshold	N/A	N/A
ISNS_LOC	Internal communication issue to current sense ADC	N/A	N/A
OVT	Transistor temperature exceeded configured overtemperature threshold	N/A	N/A
TSNS_LOC	Internal communication issue to temperature sense ADC	N/A	N/A
HVN_NP_OV	Voltage across capacitor from HV- to neutral point exceeded configured threshold, indicates risk of FET damage	N/A	N/A
NP_HVP_OV	Voltage across capacitor from neutral point to HV+ exceeded configured threshold, indicates risk of FET damage	N/A	N/A
RDC_FLT	The resolver has a fault, specific details in *0x012 (Resolver)*	N/A	N/A
HVIL_FLT	One or more HV connectors are not properly seated. This check can be enabled/disabled in the *EEPROM Configuration*	N/A	N/A
DC_PH_FLT	The controller thinks a motor phase is not properly connected to the motor controller. The relevant thresholds can be modified in the *EEPROM Configuration*	N/A	N/A
HV_UV_FLT	Undervoltage on the HV bus. The threshold can be altered in the *EEPROM Configuration*	N/A	N/A
RESERVED	Reserved	N/A	f32
MESSAGE HEARTBEAT TIMESTAMP	Milliseconds since last heartbeat. Saturates at 65535ms.	1	u16

Note

When a fault occurs, the corresponding bit will latch high. The bit will not clear until a Clear Faults command is issued via the *0x019 (Command Packet)*.

Additionally, while a fault bit is latched high, the motor controller will transmit packets indicating the worst-case value seen while the fault was latched (see *0x31D (Fault Trigger NP Cap Voltages)*, *0x31E (Fault Trigger Current/Temp)*, *0x31F (Fault Trigger HV Undervoltage)*).

1.3.1.4 0x01c (EEPROM Set Response)

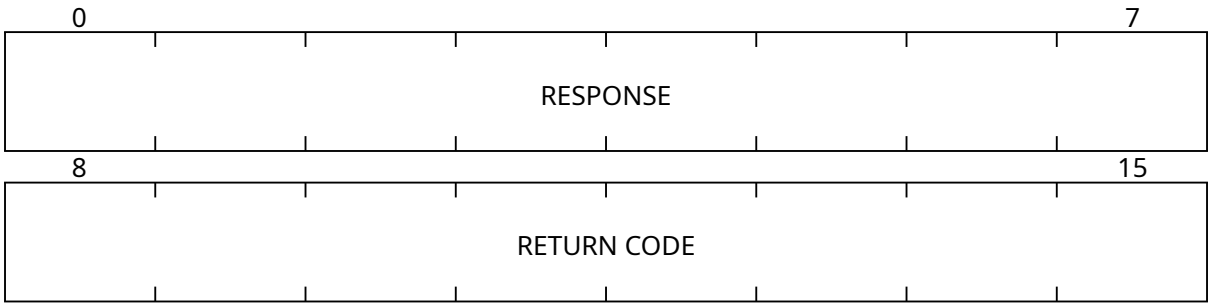


Table 9: EEPROM Set Response Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
FIELD	Returns the field that was attempted to be written to. If the request was reset to default this field will be 0xFF.	N/A	u8
RETURN CODE	A RETURN CODE = 0x00 indicates success. Typical failure codes: 0xFF - The motor controller is not in a safe state to alter configs (must be ReadyToSwitch or Faulted) 0xFE - Unrecognized config key 0xFD - The value requires a special unlock to modify 0xFC - Value is invalid or out of bounds	N/A	u8

1.3.1.5 0x01d (EEPROM Get Response)

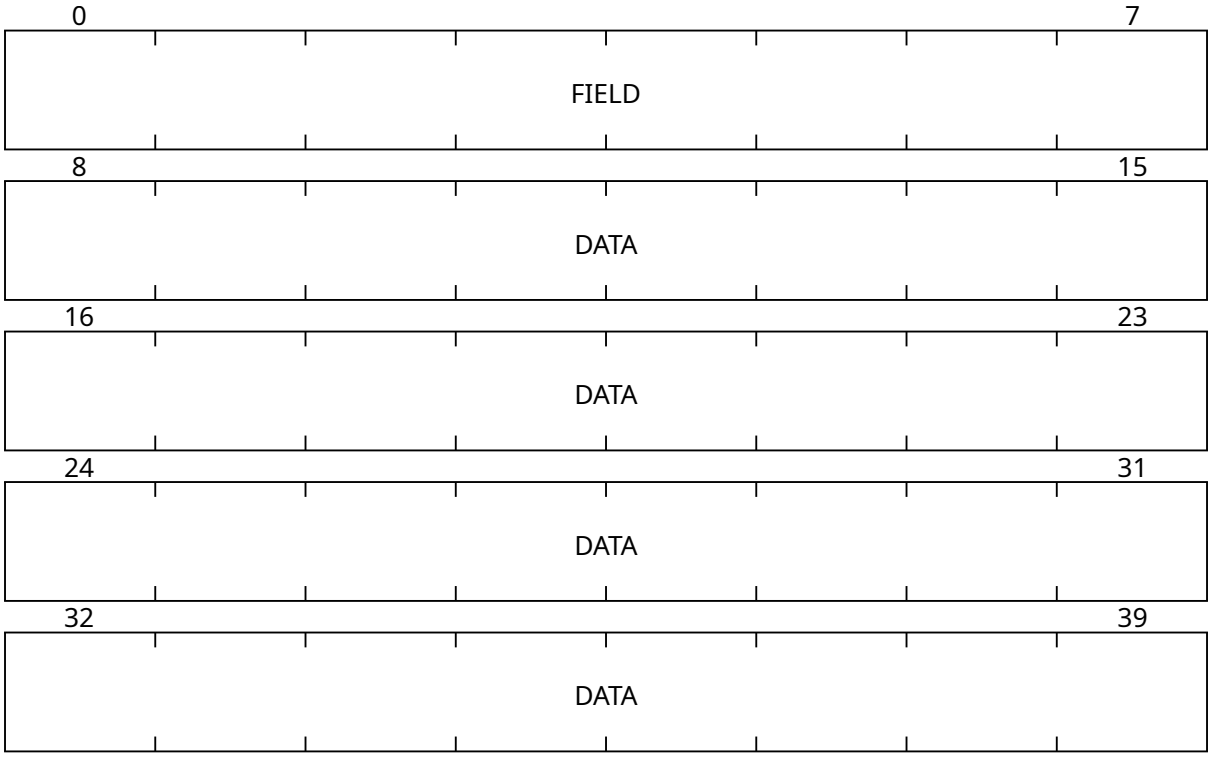


Table 10: EEPROM Get Response Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
FIELD	Returns the ID of the field that was requested	N/A	u8
DATA	Returns the data from the field.	N/A	field-dependent (f32 default)

1.3.1.6 0x020 (DQ Currents)

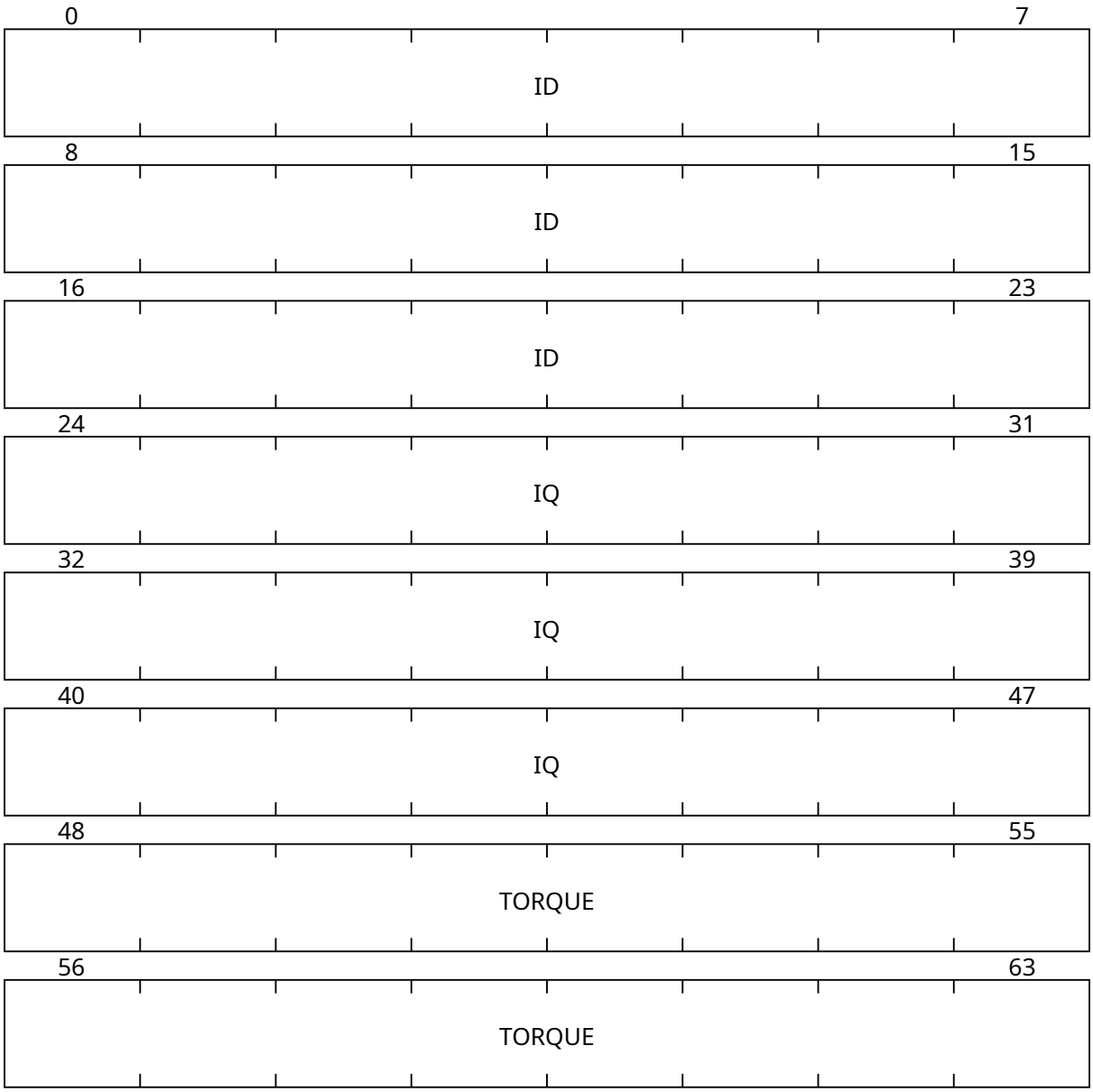


Table 11: DQ Currents Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
ID	Current in the D axis in amperes (signed)	1000	i24
IQ	Current in the Q axis in amperes (signed)	1000	i24
TORQUE	Estimated Torque being delivered based on Torque Constant and Q current. (Newton-Metres). (Currently this saturates, will be improved in next firmware revision, use IQ for a better torque estimation)	1000	i24

1.3.1.7 0x021 (PWM Duty Cycles)

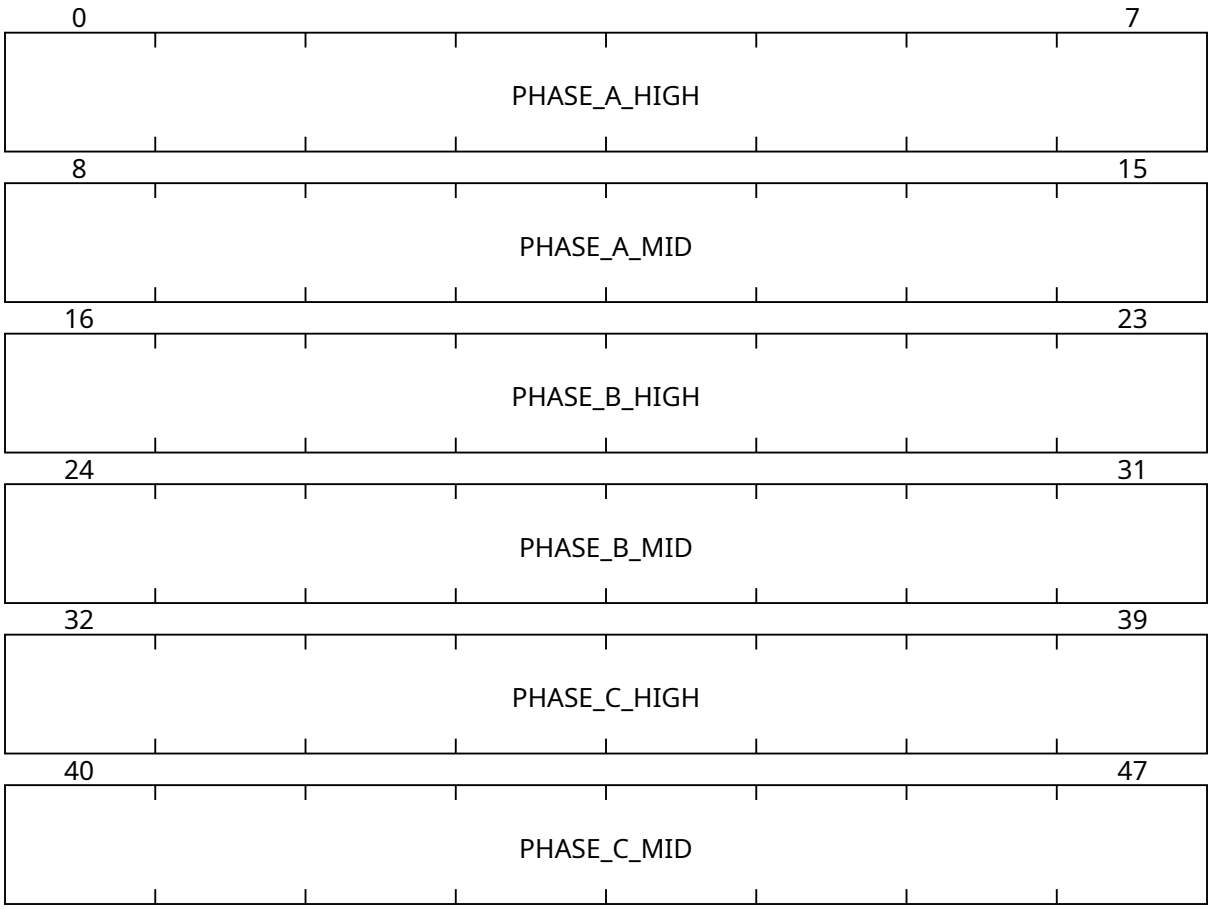


Table 12: PWM Duty Cycles Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
PHASE_A_HIGH	Percentage of switching cycle for which phase A is in the high switching state (connected to HV+)	255	u8
PHASE_A_MID	Percentage of switching cycle for which phase A is in the mid switching state (connected to inverter's neutral point)	255	u8
PHASE_B_HIGH	Percentage of switching cycle for which phase B is in the high switching state (connected to HV+)	255	u8
PHASE_B_MID	Percentage of switching cycle for which phase B is in the mid switching state (connected to inverter's neutral point)	255	u8
PHASE_C_HIGH	Percentage of switching cycle for which phase C is in the high switching state (connected to HV+)	255	u8
PHASE_C_MID	Percentage of switching cycle for which phase C is in the mid switching state (connected to inverter's neutral point)	255	u8

As all of the percentages must add up to 100%, the values for `PHASE_A_LOW`, `PHASE_B_LOW`, and `PHASE_C_LOW` can be calculated. For example, using pre-scaling values, `PHASE_A_LOW = 255 - PHASE_A_HIGH - PHASE_A_MID`

1.3.1.8 0x022 (DQ PID Integral + DC Current estimate)

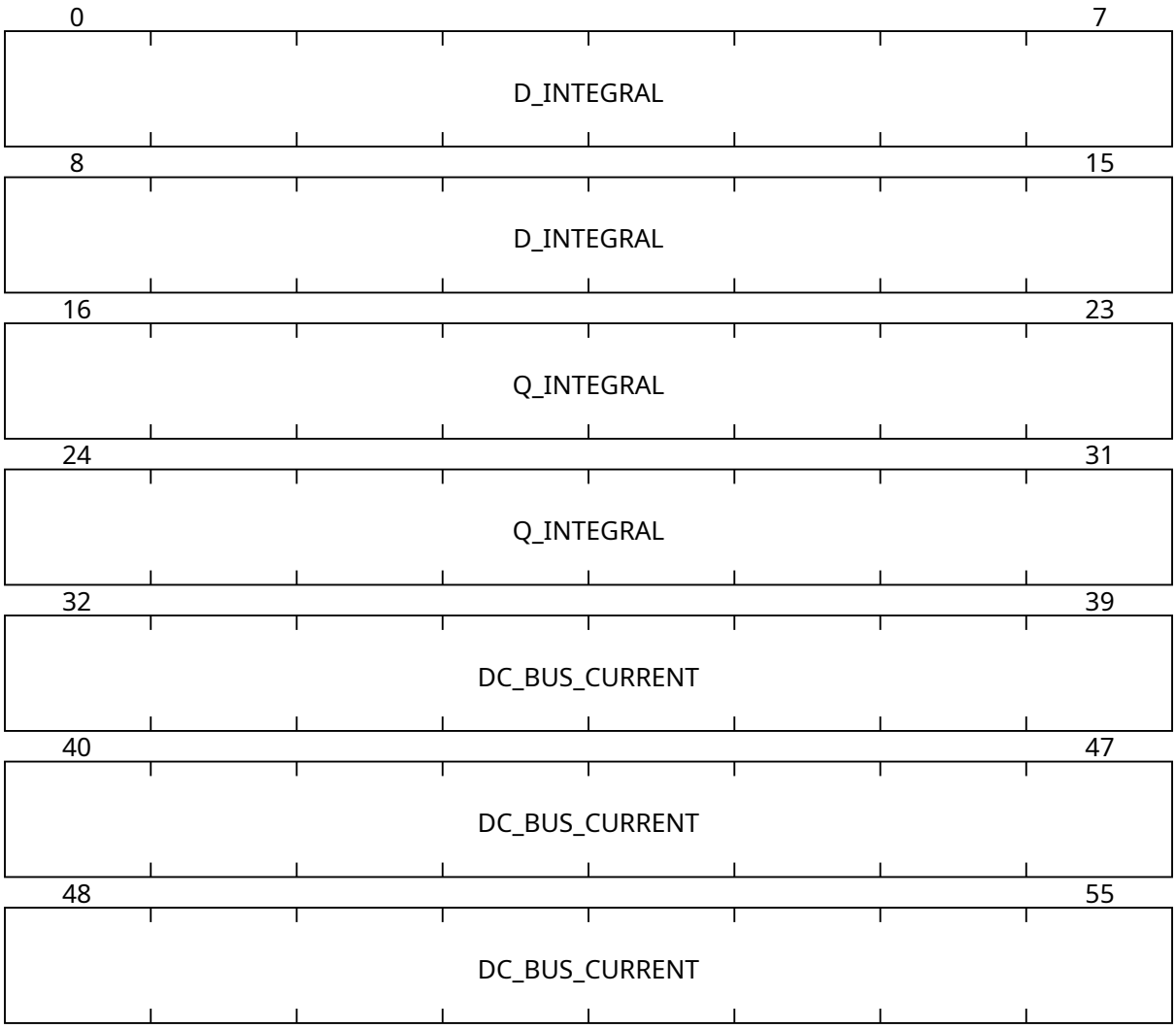


Table 13: DQ PID Integral Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
D_INTEGRAL	The accumulated integral term for the D side of the PID loop (in Volts)	10	i16
Q_INTEGRAL	The accumulated integral term for the Q side of the PID loop (in Volts)	10	i16
DC_BUS_CURRE	Estimated DC link current draw in Amperes	1000	i24

1.3.1.9 0x023 (Torque/Velocity Feedback)

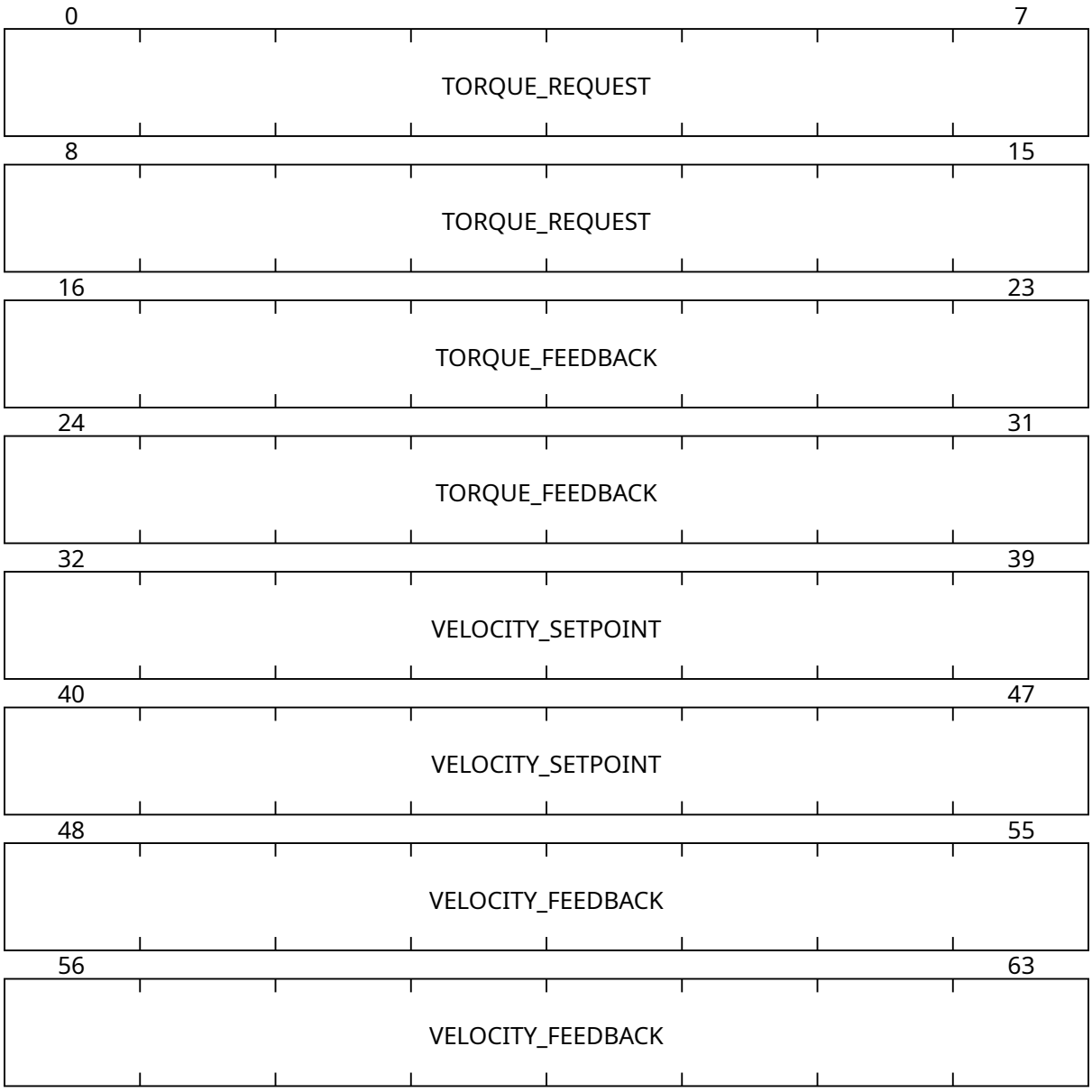


Table 14: Torque/Velocity Feedback Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
TORQUE_REQUE	Requested torque in Newton-Metres.	10	i16
TORQUE_FEEDB	Estimated actual torque in Newton-Metres.	10	i16
VELOC-ITY_SETPOINT	Velocity setpoint in mechanical RPM.	1	i16
VELOC-ITY_FEEDBACK	Measured velocity in mechanical RPM.	1	i16

Note

For the mechanical RPM value to be accurate, the *EEPROM Configuration* must be configured with the correct `NumberOfMotorPolePairs`.

1.3.1.10 0x025 (Electrical Angle)

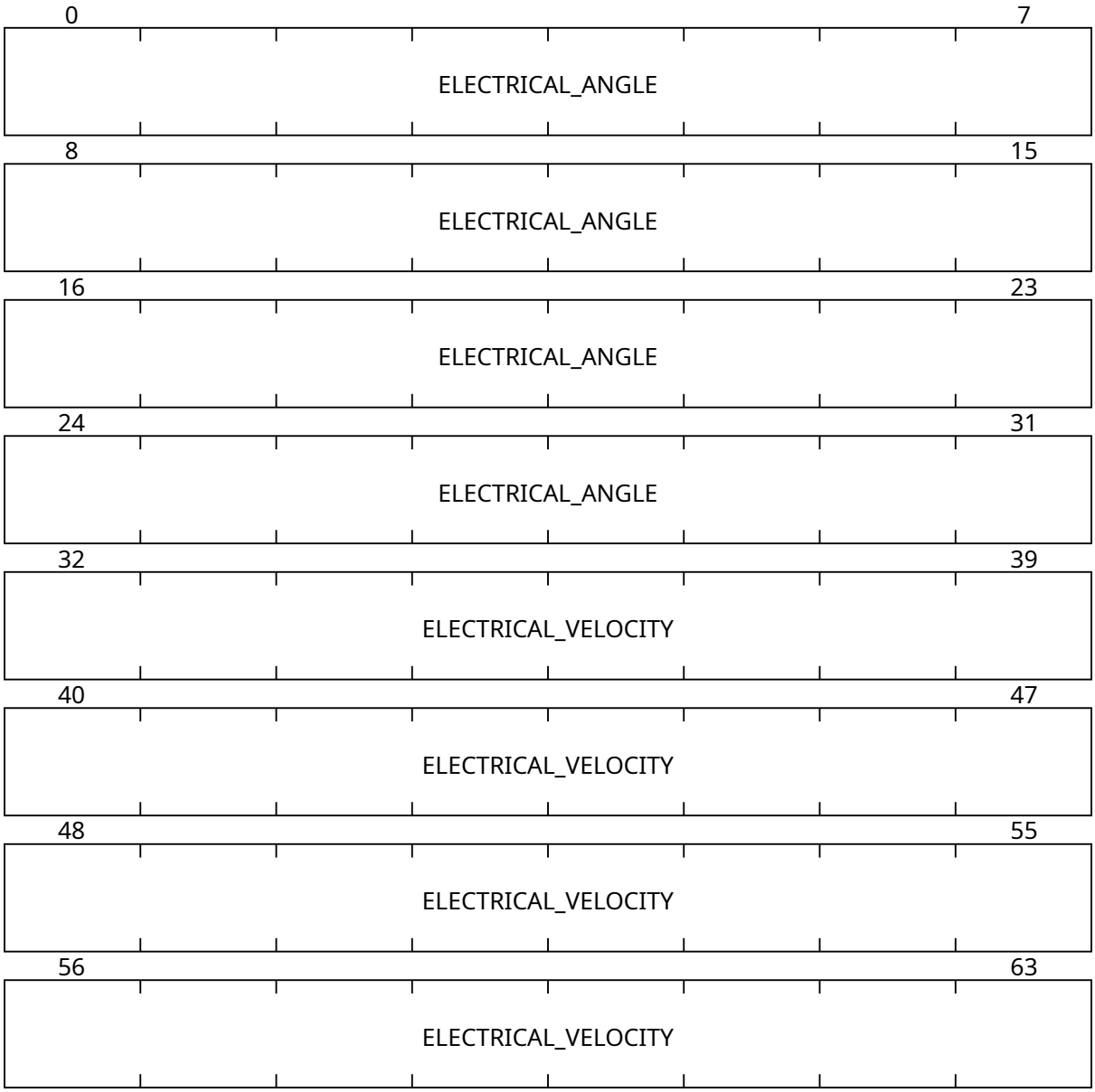


Table 15: Electrical Angle Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
ELECTRICAL_ANGLE	The electrical angle of the motor. Used in the DQZ transforms. Stored as degrees.	100	i32
ELECTRICAL_VELOCITY	The electrical velocity of the motor. Used in the DQZ transforms. Stored in degrees per seconds.	100	i32

1.3.1.11 0x029 (Switching Misc Data)

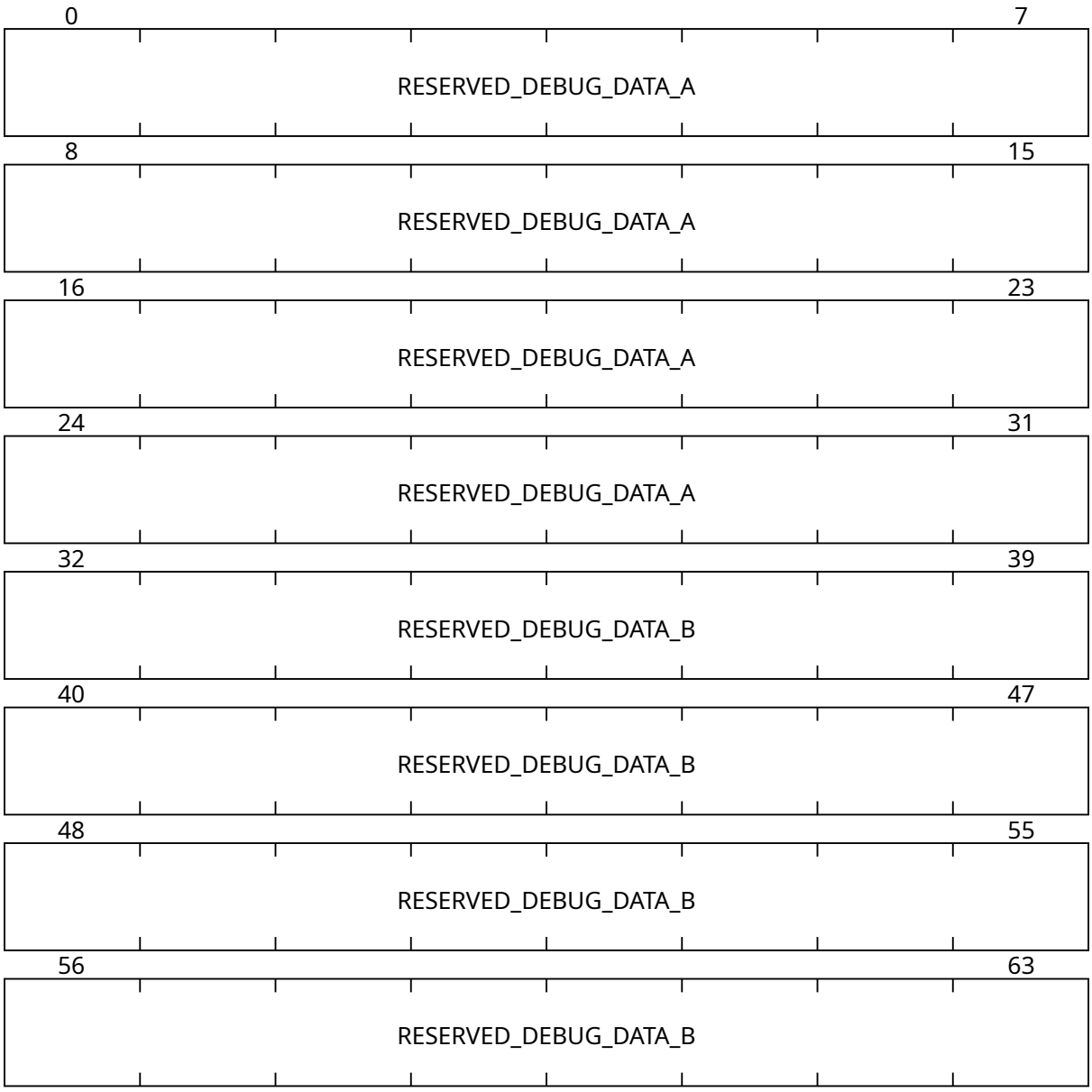


Table 16: Switching Misc Data

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
RE-SERVED_DEBUG			
RE-SERVED_DEBUG			

1.3.1.12 0x031 (Power Data)

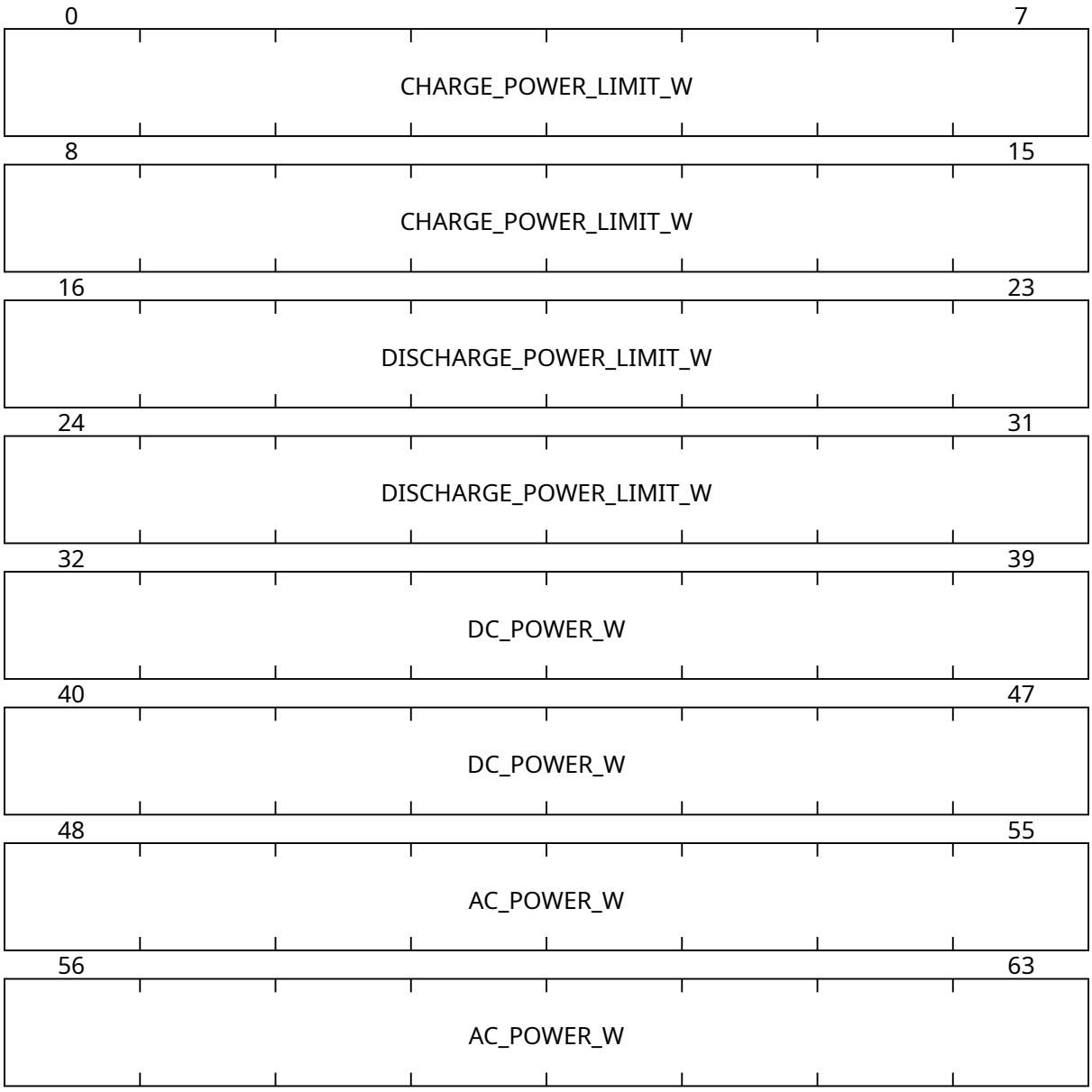


Table 17: Switching Misc Data

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
CHARGE_POWEI	Limit on amount of regen power (in Watts)	100	i16
DIS-CHARGE_POWEI	Limit on amount of motoring power (in Watts)	100	i16
DC_POWER_W	Estimated power consumption on the DC bus	100	i16
AC_POWER_W	Estimated instaneous power driven into the motor	100	i16

Note

Currently, the only factors influencing the power limits are the DC bus current limits.

1.3.1.13 0x306 (GPIO State)

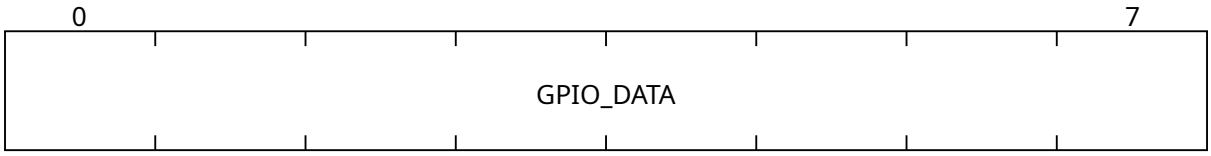


Table 18: GPIO State

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
GPIO_DATA	To be implemented	N/A	N/A

1.3.1.14 0x310-0x31B (Temperature Data)

Indexed as (0x310 + n) where n is the nth group of temperature readings.

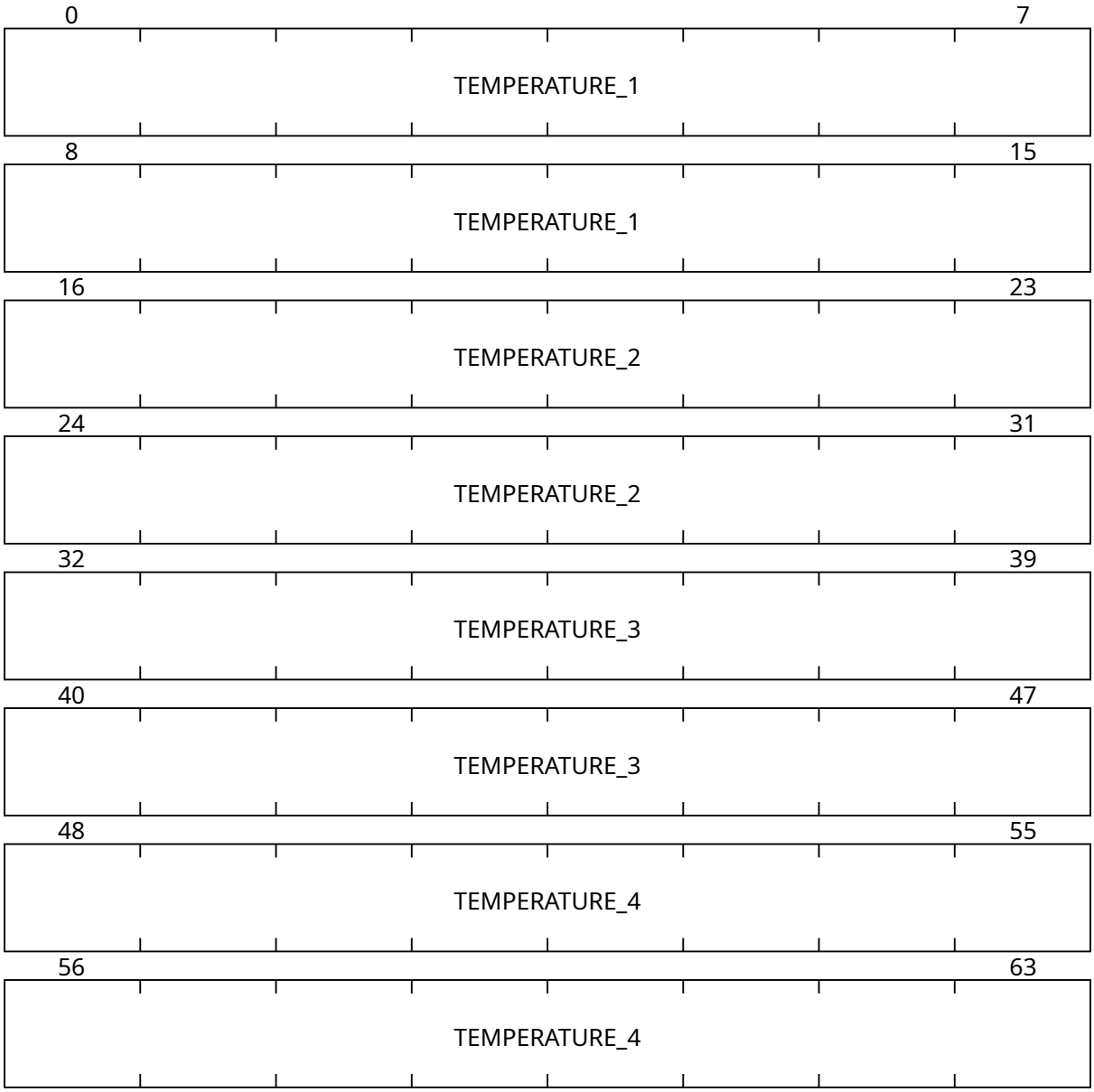


Table 19: Temperature Data

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
TEMPERATURE_1	Temperature reading from 1 thermistor in Celsius	100	i16
TEMPERATURE_2	Temperature reading from 1 thermistor in Celsius	100	i16
TEMPERATURE_3	Temperature reading from 1 thermistor in Celsius	100	i16
TEMPERATURE_4	Temperature reading from 1 thermistor in Celsius	100	i16

1.3.1.15 0x31C (External RTD interface)

Indexed as (0x310 + n) where n is the nth group of temperature readings.

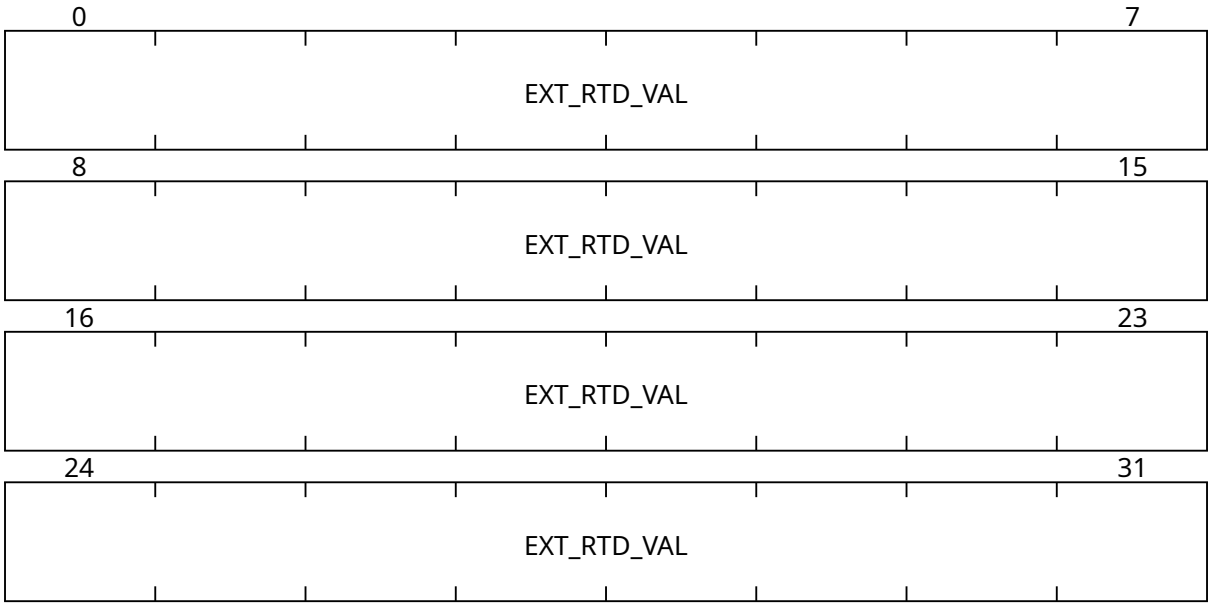


Table 20: Temperature Data

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
EXT_RTD_VAL	Temperature reading from external PT100/PT1000 RTD in Celsius. Can alternately report resistance – configurable in *EEPROM Configuration*.	100	i32

1.3.1.16 0x31D (Fault Trigger NP Cap Voltages)

Only transmitted when at least one NP cap overvoltage fault (`HVN_NP_OV` or `NP_HVP_OV` in *0x016 (State ID)*) is latched. Fields for faults that have not triggered are reported as 0.

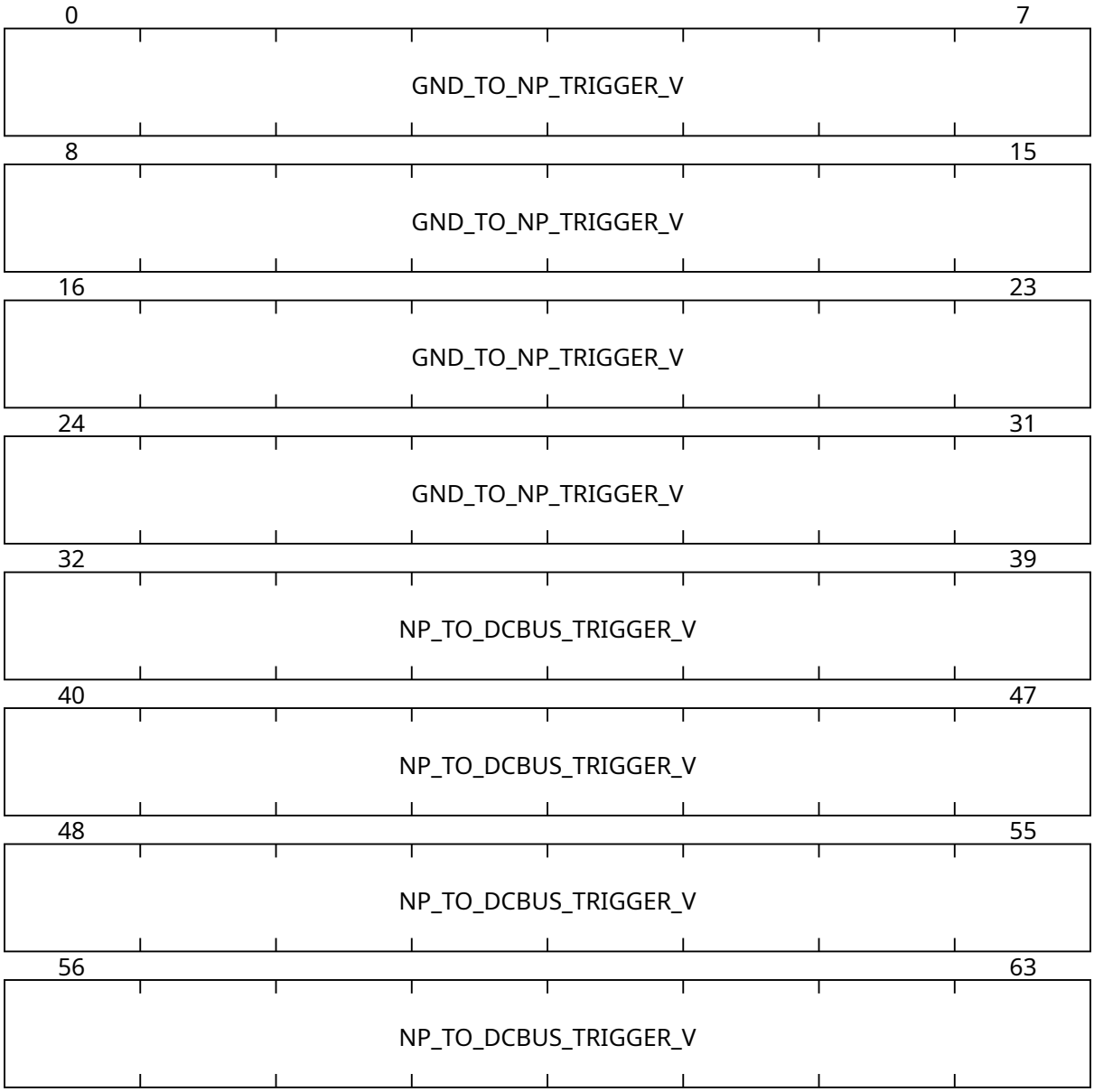


Table 21: Fault Trigger NP Cap Voltages Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
GND_TO_NP_TRI	Highest voltage across the capacitor from GND to neutral point that triggered the HVN_NP_OV fault, in Volts. Corresponds to `V_NEUTRAL_POINT` in *0x323 (Voltage Sense)*.	1,000,000	i32
NP_TO_DCBUS_T	Highest voltage across the capacitor from neutral point to HV+ that triggered the NP_HVP_OV fault, in Volts. Corresponds to `V_BUS - V_NEUTRAL_POINT` in *0x323 (Voltage Sense)*.	1,000,000	i32

1.3.1.17 0x31E (Fault Trigger Current/Temp)

Only transmitted when at either of the `OVC` or `OVT` faults are latched in *0x016 (State ID)*. Fields for faults that have not triggered are reported as 0.

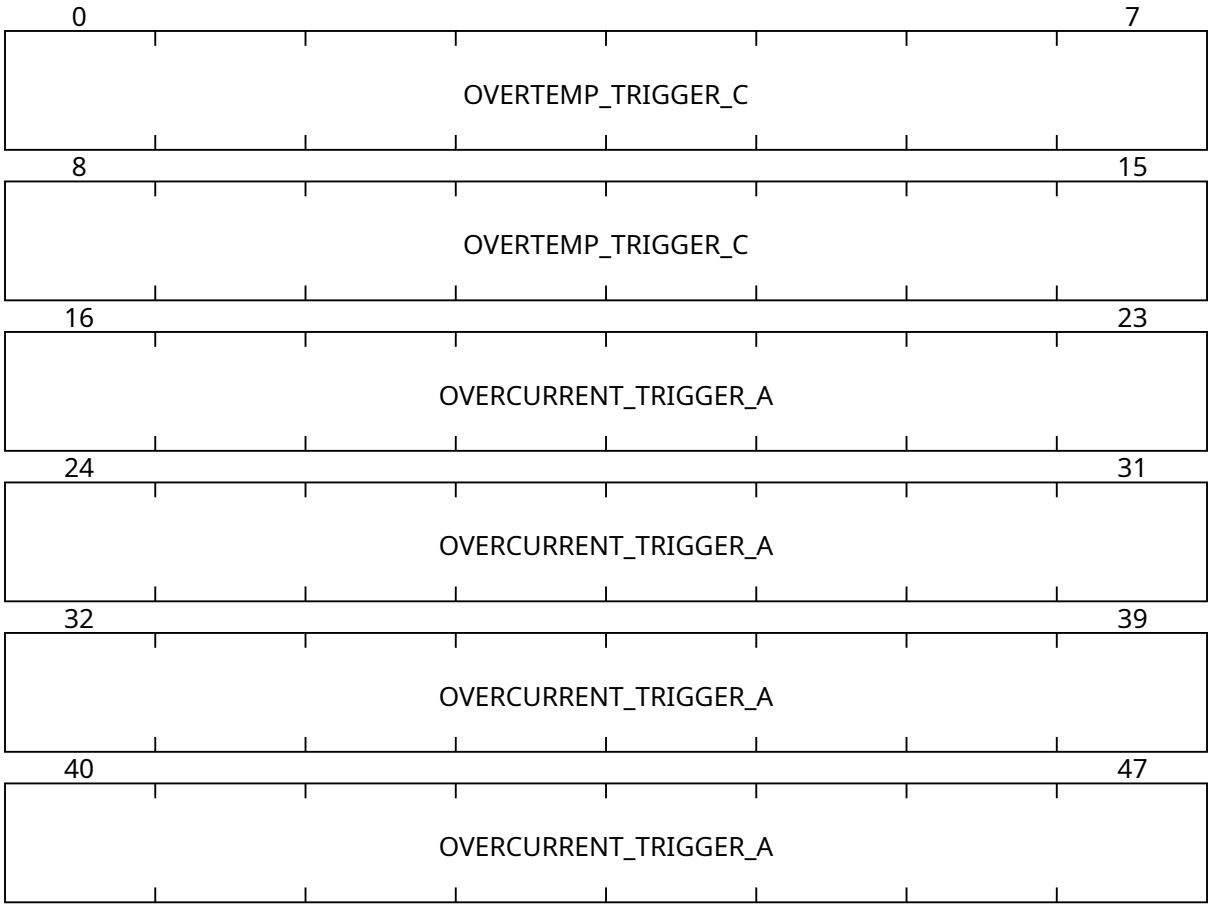


Table 22: Fault Trigger Current/Temp Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
OVERTEMP_TRIC	Highest transistor temperature that triggered the OVT fault, in Celsius.	100	i16
OVERCUR-RENT_TRIGGER_	Highest magnitude phase current that triggered the OVC fault, in Amperes.	1,000,000	i32

1.3.1.18 0x31F (Fault Trigger HV Undervoltage)

Only transmitted when the HV undervoltage fault (`HV_UV_FLT` in *0x016 (State ID)*) is latched.

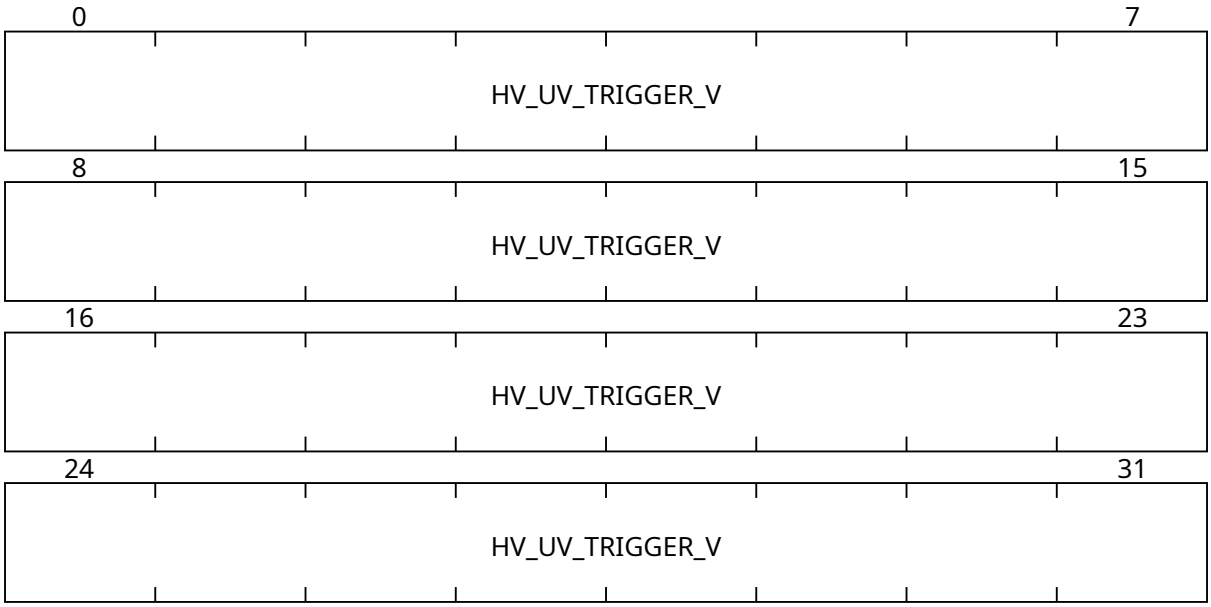


Table 23: Fault Trigger HV Undervoltage Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
HV_UV_TRIGGE	Lowest HV bus voltage that triggered the HV_UV_FLT fault, in Volts. Corresponds to `V_BUS` in *0x323 (Voltage Sense)*.	1,000,000	i32

1.3.1.19 0x320 (Current Sense Phase C)

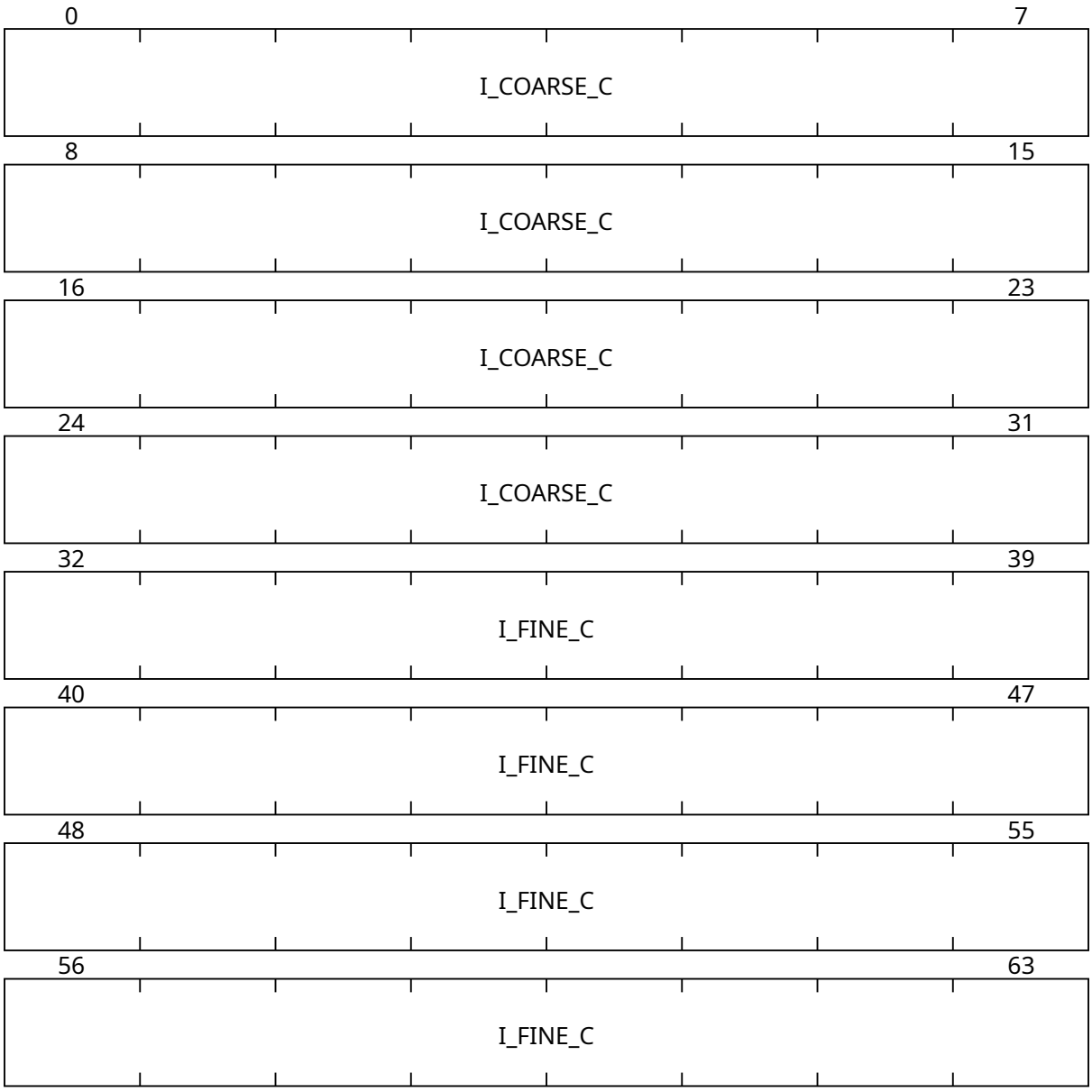


Table 24: Current Sense Phase C

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
I_COARSE_C	Coarse current readings from Phase C in Amperes.	1,000,000	i32
I_FINE_C	Fine current readings from Phase C in Amperes.	1,000,000	i32

1.3.1.20 0x321 (Current Sense Phase B)

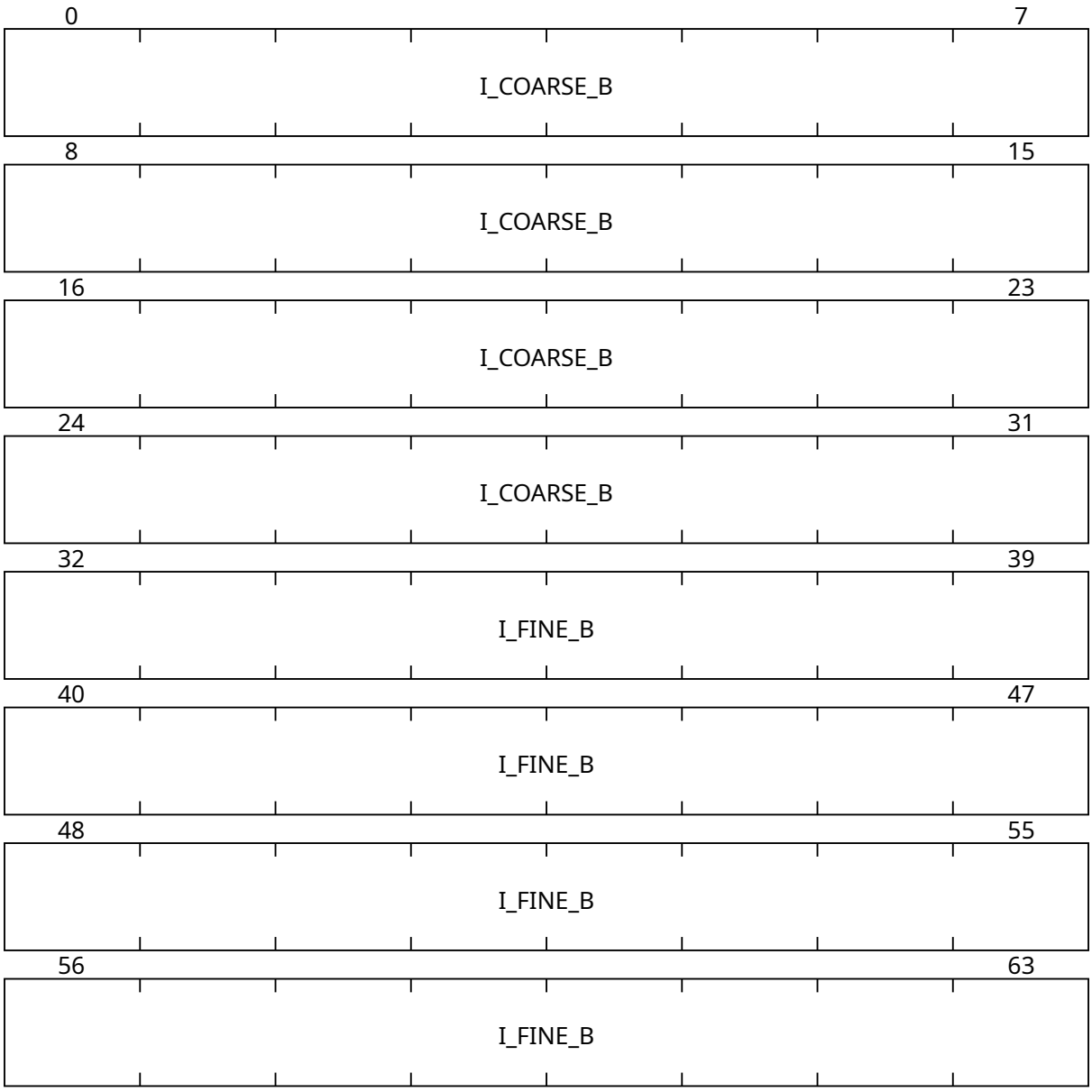


Table 25: Current Sense Phase B

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
I_COARSE_B	Coarse current readings from Phase B in Amperes.	1,000,000	i32
I_FINE_B	Fine current readings from Phase B in Amperes.	1,000,000	i32

1.3.1.21 0x322 (Current Sense Phase A)

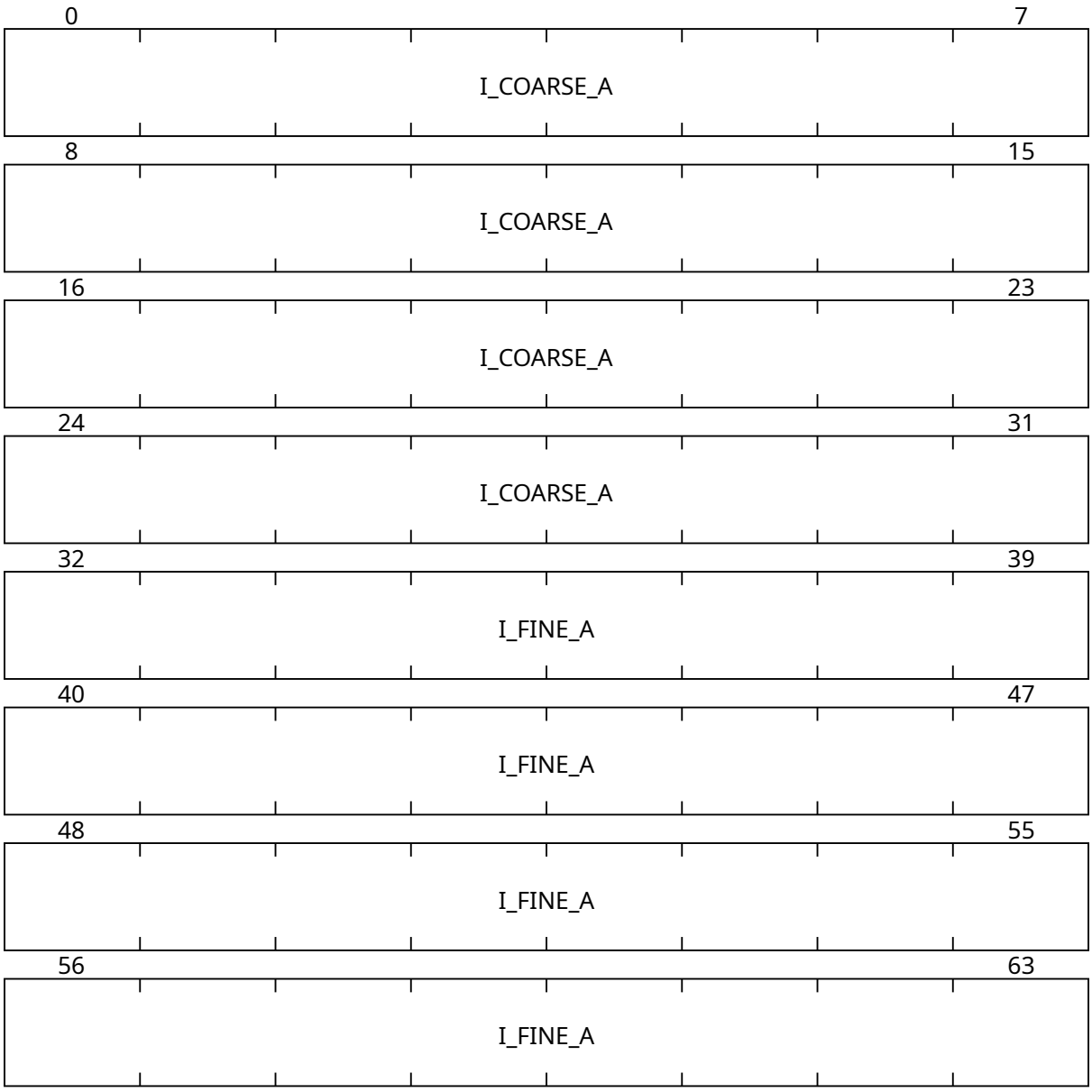


Table 26: Current Sense Phase A

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
I_COARSE_A	Coarse current readings from Phase A in Amperes.	1,000,000	i32
I_FINE_A	Fine current readings from Phase A in Amperes.	1,000,000	i32

1.3.1.22 0x323 (Voltage Sense)

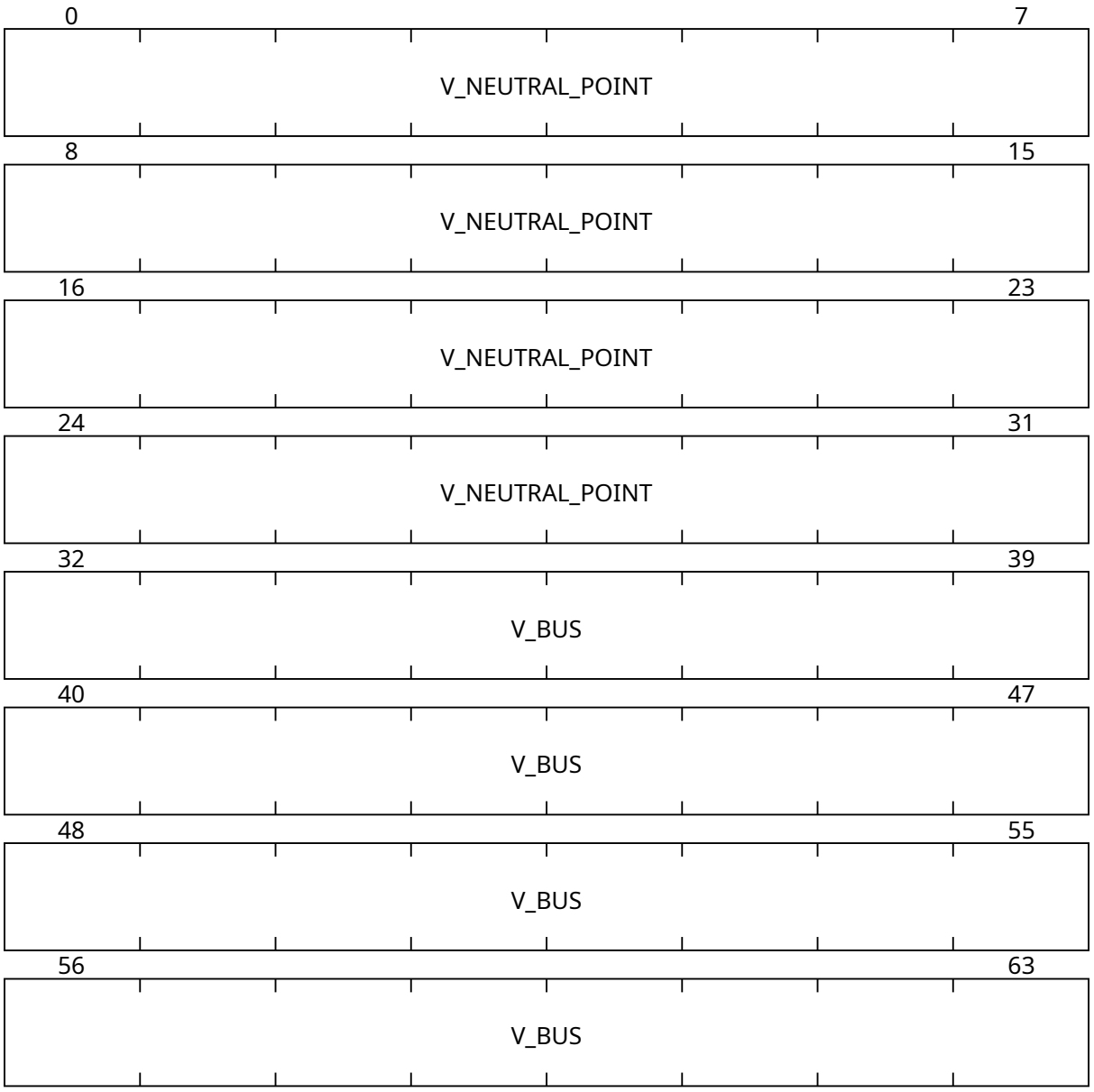


Table 27: Voltage Sense

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
V_NEUTRAL_PO	Voltage from V- to Neutral Point in Volts.	1,000,000	i32
V_BUS	Voltage from V- to V+ in Volts.	1,000,000	i32

1.3.1.23 0x324 (Current Sense ADC Debug)

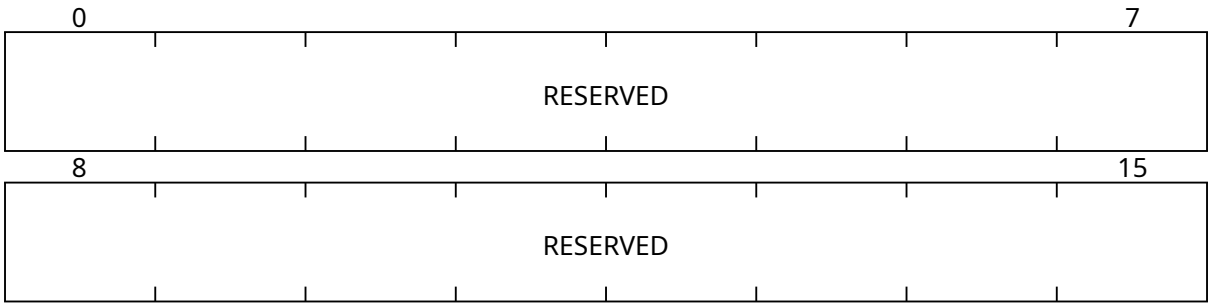


Table 28: Current Sense ADC Debug

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
RESERVED		N/A	

1.3.2 RX (Received by Motor Controller)

1.3.2.1 Summary

Table 29: RX CAN Messages

CAN ID	CAN Message Name	Brief Description
0x019	Command Packet	Command packet. Used to traverse state machine and command torque/speed.
0x01a	EEPROM Set Parameter	Write parameter to EEPROM.
0x01b	EEPROM Get Parameter	Read parameter to EEPROM.
0x202	Set DC Current Limit Packet	Used to update the DC current limit in real-time

1.3.2.2 0x019 (Command Packet)



Table 30: Command Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
COMMAND	Command sent from user. See Command Table below.	N/A	u8
DATA	The data supplement used with the command packet. Interpretation depends on the COMMAND byte.	Command-dependent	i32

Table 31: Command Table

Command ID	Command Description	Data Description	Data Scaling (Divide by X to get real value)
0x00	DISABLE. This tells the motor controller to enter a disable state. This should take the motor controller into the ReadyToSwitch state.	DATA unused, leave as 0x00	N/A
0x01	Enable (Torque Control). This instructs the motor controller to go into torque control mode. It will transition to ActivelySwitching and attempt to drive a desired torque.	DATA is desired torque in Newton-Metres	10
0x02	Enable (Velocity Control — edeg/s). This instructs the motor controller to go into velocity control mode commanding in electrical degrees per second.	DATA is desired velocity in electrical degrees per second.	1
0x04	Enable (Velocity Control — RPM). This instructs the motor controller to go into velocity control mode commanding in mechanical RPM. The conversion to electrical degrees per second uses the configured *EEPROM Configuration* value for number of motor pole pairs.	DATA is desired velocity in mechanical RPM.	1
0xFF	Clear Faults. Attempt to clear non-latching faults of motor controller. See *State Machine*	DATA unused, leave as 0x00	N/A

1.3.2.3 0x202 (Set DC Current Limit Packet)

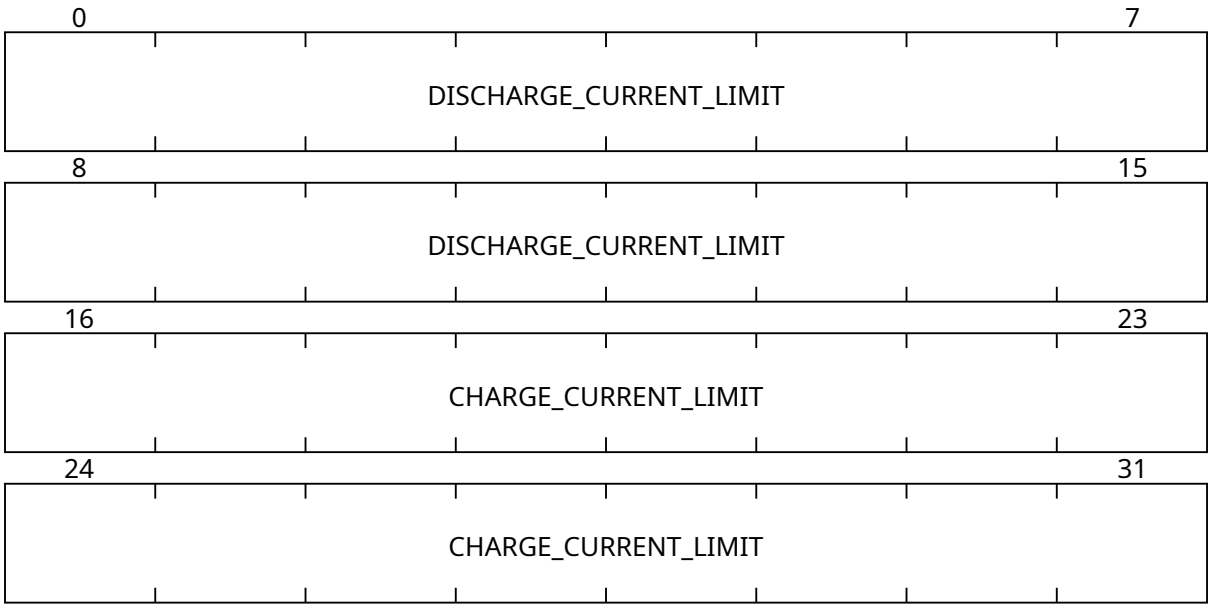


Table 32: Command Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
DIS-CHARGE_CURRE	Maximum amount of current the inverter is allowed to draw from the DC bus (in Amps)	10	i16
CHARGE_CURRE	Maximum amount of current the inverter is allowed to supply into the DC bus.	10	i16

Note

The new limit supplied by this packet will remain in place until either

a) Another packet is received

b) The device is power-cycled, in which case the limit from the *EEPROM Configuration* is used.

1.3.2.4 0x01a (EEPROM Set Parameter)

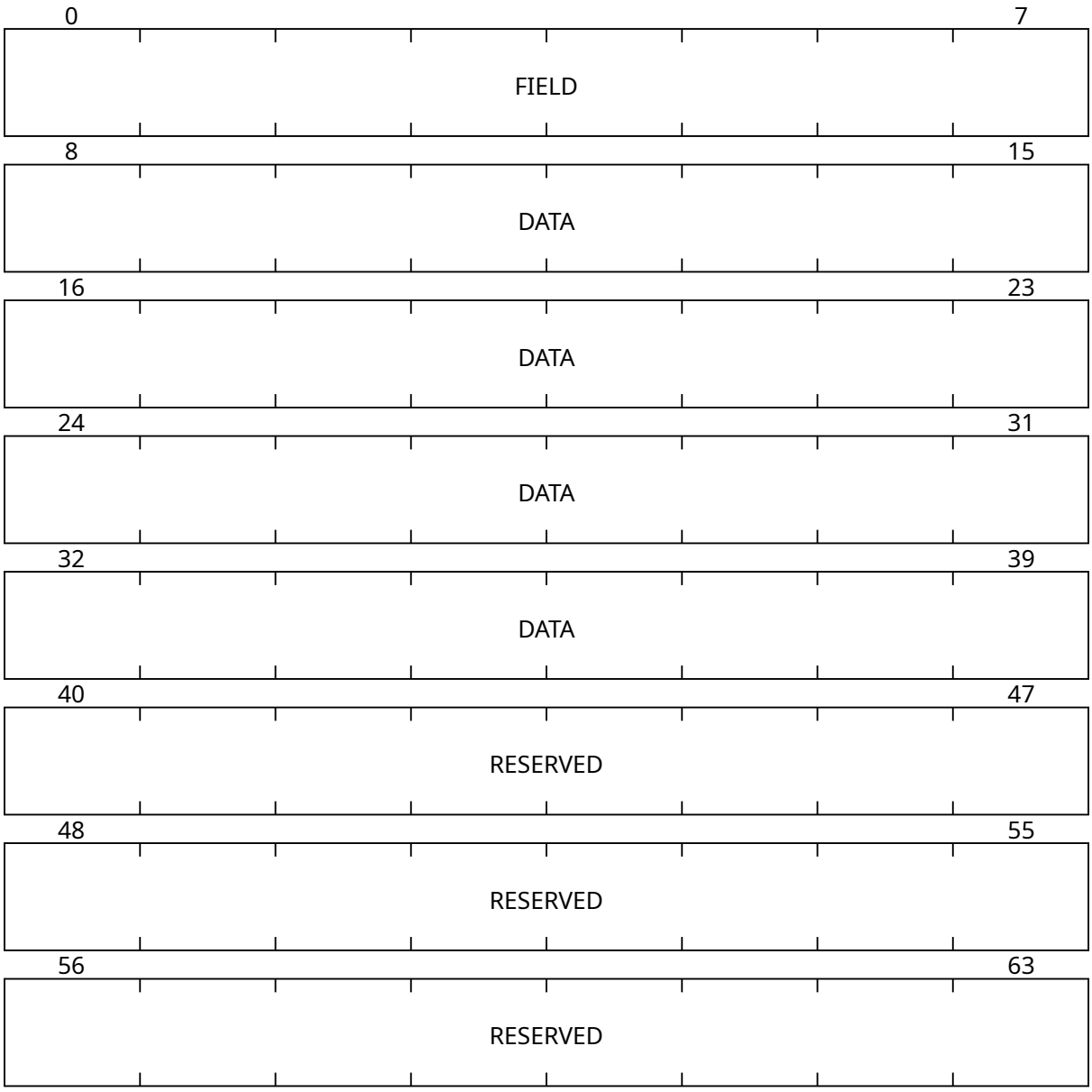


Table 33: EEPROM Set Parameter Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
FIELD	ID of the desired parameter to modify	N/A	u8
DATA	Data to be written to EEPROM parameter.	N/A	field-dependent, default f32
RESERVED	Reserved data, leave as 0x00.	N/A	N/A

1.3.2.5 0x01b (EEPROM Get Parameter)

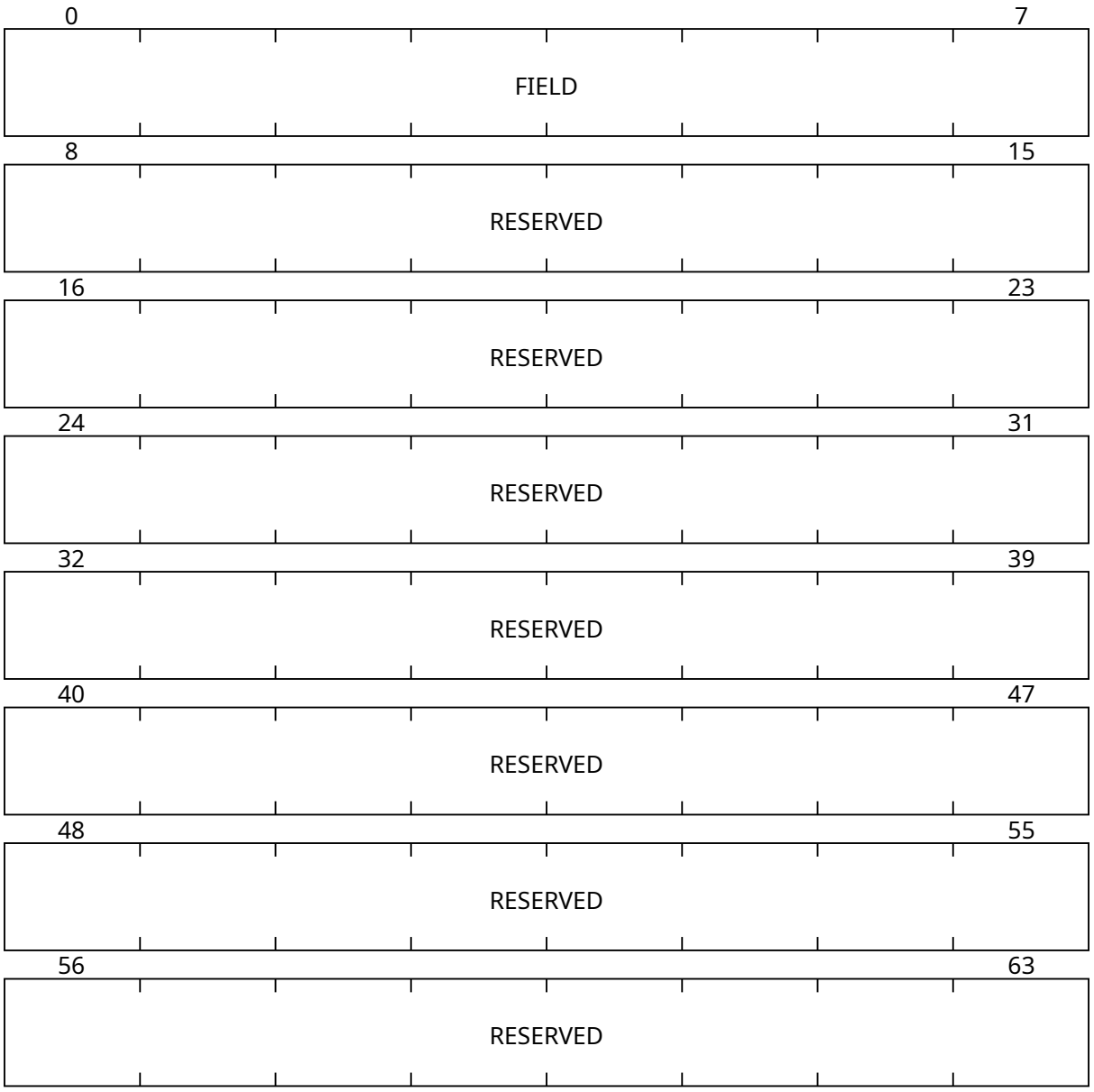


Table 34: EEPROM Set Parameter Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
FIELD	ID of the desired parameter to read	N/A	u8
RESERVED	Reserved data, leave as 0x00.	N/A	N/A

1.4 EEPROM Configuration

The EEPROM allows for persistent configuration of the motor controller.

1.4.1 Fields

All values (including f32 and u32 values) are little-endian. Divide values by their scaling in order to retrieve the true value.

Table 35: EEPROM Fields

Field ID	Field Name	Field Description	Datatype	Scaling	Bounds	Units
0	NumberOfBoots	Number of boots of the device total. (pending implementation)	f32	1		
1	ResolverOffsetDegrees	Offset from measured resolver angle to electrical angle in degrees.	f32	1	[-360.0, 360.0]	E°
2	LoopIqPGain	P (proportional) gain in the Iq PID loop	f32	1		$\mathrm{A/V}$
3	LoopIqIGain	I (integral) gain in the Iq PID loop	f32	1		$\mathrm{A \cdot s/V}$
4	LoopIqDGain	D (derivative) gain in the Iq PID loop	f32	1		$\mathrm{A/\left(V \cdot s\right)}$
5	LoopIqTGain	Gain for integral anti windup in the Iq PID loop	f32	1		
6	LoopIdPGain	P (proportional) gain in the Id PID loop	f32	1		$\mathrm{A/V}$
7	LoopIdIGain	I (integral) gain in the Id PID loop	f32	1		$\mathrm{A \cdot s/V}$
8	LoopIdDGain	D (derivative) gain in the Id PID loop	f32	1		$\mathrm{A/\left(V \cdot s\right)}$
9	LoopIdTGain	Gain for integral anti windup in the Id PID loop	f32	1		
10	TorqueConst	Torque constant for the motor. Units Newton-Metres per Ampere. This torque constant is relative to Q-Axis current. The physical interpretation is roughly Nm/A where current is the amplitude of the torque generating phase current.	f32	1		$\mathrm{Nm/A}$
11-22	Current Scaling and Offsets	The user should not touch these, calibrated at factory.	f32	1		
23	LoopVelPGain	P (proportional) gain in the velocity control loop.	f32	1		$\mathrm{Nm \cdot s/E^\circ}$
24	LoopVelIGain	I (integral) gain in the velocity control loop.	f32	1		$\mathrm{Nm/E^\circ}$
25	LoopVelDGain	D (derivative) gain in the velocity control loop.	f32	1		$\mathrm{Nm \cdot s^2/E^\circ}$
26	LoopVelTGain	Gain for integral anti windup in the velocity control loop.	f32	1		
27	MaxTorqueSlewRate	Maximum torque slew rate in Newton-Metres per s.	f32	1	>= 0.0	$\mathrm{Nm/s}$
28	OvercurrentThresholdAmps	Maximum allowable phase current magnitude (in Apk)	f32	1	[0.0, 1500.0]	$\mathrm{A_{pk}}$
29	OvertempThresholdCelsius	Maximum allowable temperature in the inverter	f32	1	[0.0, 200.0]	$^\circ\mathrm{C}$
30	ResolverPolesPerMotorPole	Number of resolver poles per motor pole. Sign indicates resolver inversion.	f32	1		

continues on next page

Table 35 – continued from previous page

Field ID	Field Name	Field Description	Datatype	Scaling	Bounds	Units
31	MaxNpCapacitorVoltage	Maximum allowable voltage across either neutral-point capacitor	f32	1	<= 700.0	V
32	EnableHvilAlarm	Set to 1 to prevent operation if any HV connectors are unseated. Set to 0 to disable this check.	u32	1	[0, 1]	
33	ExtTempSensorType	This value controls what is reported in the *0x31C (External RTD interface)*. Options: • 0 - voltage (millivolts) • 1 - resistance (Ohms) • 2 - PT1000 temperature (100x Celsius) • 3 - PT100 temperature (100x Celsius)	u32	N/A	[0, 3]	
34	DeviceIdentifier	Lower 7 bytes of the 20-byte device identifier (little endian). The device identifier as printed on the unit is shown in base 36.	7 bytes	N/A		
35	DeviceIdentifier2	Middle 7 bytes of the 20-byte device identifier (little endian)	7 bytes	N/A		
36	DeviceIdentifier3	Upper 6 bytes of the 20-byte device identifier (little endian)	6 bytes	N/A		
37	HardwareRevision	Internal revision number for sensing and control circuitry.	u32	N/A		
38	DisconnectedPhaseVoltageThresholdVolts	If the applied phase-to-neutral voltage exceeds this threshold…	f32	N/A		V
39	DisconnectedPhaseCurrentThresholdAmps	… and if the measured current magnitude is below this threshold, a disconnected phase fault will be triggered	f32	N/A		A
40	UndervoltageThresholdVolts	If the DC bus voltage drops below this value while the motor controller is enabled, an undervoltage fault will be triggered.	f32	N/A		V
41	CanTimeoutMs	If more than this many milliseconds elapses between command packets on the CANBus, the motor controller will fault.	u32	N/A		ms
42	DefaultMaxDcDischargeCurrent	Maximum DC current allowed into the inverter (positive, in Amps). The inverter will automatically limit torque in order to remain within the DC bus current limits.	f32	N/A	>= 0	A

continues on next page

Table 35 – continued from previous page

Field ID	Field Name	Field Description	Datatype	Scaling	Bounds	Units
43	DefaultMaxDcChargeCurrent	Maximum DC current allowed out of the inverter (negative, in Amps)	f32	N/A	<= 0	A
44	NumberOfMotorPolePairs	Number of motor pole pairs in the motor. This is strictly used for converting from mechanical RPM in CAN packets to electrical RPM	u32	N/A	>= 1	

1.4.2 Reading/Writing to EEPROM

There are 2 ways to configure the EEPROM. You can manually send CAN messages using whatever tool you prefer, or you can use Steinmetz's tool which will autoformat and set that for you. The mechanism of configuration is the same for both manually sending CAN messages or the tool, the benefit of the tool is it will perform checks and formatting for you.

1.4.2.1 Using Provided Tool

Steinmetz provides a command-line tool to configure Avalanche EEPROM as well as manage firmware updates. At this time, Steinmetz only provides a tool that works on Linux systems. By default, the tool communicates over the `can0` socketcan interface, although this can be changed with the `--can` flag.

1.4.2.1.1 Reading

The config tool can be used to read individual config values, or to dump all config values. Individual config values can be read like so:

```
$ ./steinmetz_tool config read ResolverOffsetDegrees
ResolverOffsetDegrees: 20
```

The list of all config values can be show with the `--help` flag:

```
$ ./steinmetz_tool config read --help
Read the value of a single configuration parameter

Usage: steinmetz_tool config read [OPTIONS] <KEY>

Arguments:
  <KEY>
          Possible values:
          - NumberOfBoots
          - ResolverOffsetDegrees:                  Resolver offset (in␣
↪degrees)
          - LoopIqPGain:                            kP for the q-axis current␣
↪controller
          - LoopIqIGain:                            kI for the q-axis current␣
↪controller
          - LoopIqDGain:                            kD for the q-axis current␣
↪controller
          - LoopIqTGain
          - LoopIdPGain:                            kP for the d-axis current␣
↪controller
          - LoopIdIGain:                            kI for the d-axis current␣
↪controller
          - LoopIdDGain:                            kD for the d-axis current␣
↪controller
          - LoopIdTGain
          - TorqueConst:                            Torque constant (in Nm/
↪Amps)
          - PhaseACoarseHallOffsetVolts
          - PhaseBCoarseHallOffsetVolts
```

(continues on next page)

(continued from previous page)

```
          - PhaseCCoarseOffsetVolts
          - PhaseACoarseVoltsPerAmp
          - PhaseAFineHallOffsetVolts
          - PhaseBFineHallOffsetVolts
          - PhaseCFineOffsetVolts
          - PhaseAFineVoltsPerAmp
          - PhaseBCoarseVoltsPerAmp
          - PhaseCCoarseVoltsPerAmp
          - PhaseBFineVoltsPerAmp
          - PhaseCFineVoltsPerAmp
          - LoopVelPGain:                          kP for closed-loop␣
↪velocity controller
          - LoopVelIGain:                          kI for closed-loop␣
↪velocity controller
          - LoopVelDGain:                          kD for closed-loop␣
↪velocity controller
          - LoopVelTGain
          - MaxTorqueSlewRate:                     Maximum allowable slew␣
↪rate for torque controller (in Nm/s)
          - OvercurrentThresholdAmps:              Maximum allowable phase␣
↪current magnitude (in Apk)
          - OvertempThresholdCelsius:              Maximum allowable␣
↪temperature in the inverter
          - ResolverPolesPerMotorPole:             Number of resolver poles␣
↪per motor pole. Sign indicates resolver inversion
          - MaxNpCapacitorVoltage:                 Maximum allowable voltage␣
↪across either neutral-point capacitor
          - EnableHvilAlarm:                       Enable HVIL alarm
          - ExtTempSensorType:                     Type of temperature␣
↪sensor connected to external temp port
          - DeviceIdentifier:                      The device identifier␣
↪uniquely identifies a device and ties it back to pre-delivery acceptance␣
↪testing data
          - HardwareRevision
          - DisconnectedPhaseVoltageThresholdVolts: If a phase has more than␣
↪this phase-to-neutral voltage applied, then it should have more current␣
↪than [`DisconnectedPhaseCurrentThresholdA`] flowing through it. Otherwise,␣
↪it may be disconnected
          - DisconnectedPhaseCurrentThresholdAmps:  Threshold for phase␣
↪current to be considered abnormally low (in A)
          - UndervoltageThresholdVolts:            Threshold for␣
↪undervoltage alarm (in Volts)
          - CanTimeoutMs:                          CAN bus communications␣
↪timeout (in milliseconds)
          - DefaultMaxDcDischargeCurrent:          Maximum DC current␣
↪allowed into the inverter (positive, in Amps)
          - DefaultMaxDcChargeCurrent:             Maximum DC current␣
↪allowed out of the inverter (negative, in Amps)
          - NumberOfMotorPolePairs:                Number of motor pole␣
```

(continues on next page)

(continued from previous page)

```
↪pairs (i.e. half the number of magnetic poles)

Options:
      --can <CAN>
          [default: can0]

  -h, --help
          Print help (see a summary with '-h')
```

1.4.2.1.2 Writing

The tool can similarly be used to write values to EEPROM:

```
$ ./steinmetz_tool config read ResolverOffsetDegrees
ResolverOffsetDegrees: 10
$ ./steinmetz_tool config write ResolverOffsetDegrees 20.0
Successfully Written ResolverOffsetDegrees: 20
$ ./steinmetz_tool config read ResolverOffsetDegrees
ResolverOffsetDegrees: 20
```

1.4.2.1.3 Dumping

For efficiently reading the entire configuration (e.g in preparation to provision backup units), one may also dump the entire config:

```
$ ./steinmetz_tool config dump
NumberOfBoots: 0
ResolverOffsetDegrees: 285
LoopIqPGain: 0.414
LoopIqIGain: 46.88
LoopIqDGain: 0
LoopIqTGain: 0
LoopIdPGain: 0.414
LoopIdIGain: 46.88
LoopIdDGain: 0
LoopIdTGain: 0
TorqueConst: 1
PhaseACoarseHallOffsetVolts: 0
PhaseBCoarseHallOffsetVolts: 0
PhaseCCoarseOffsetVolts: 0
PhaseACoarseVoltsPerAmp: 0.0012143924
PhaseAFineHallOffsetVolts: 0
PhaseBFineHallOffsetVolts: 0
PhaseCFineOffsetVolts: 0
PhaseAFineVoltsPerAmp: 0.005004027
PhaseBCoarseVoltsPerAmp: 0.0012751862
PhaseCCoarseVoltsPerAmp: 0.0010704114
PhaseBFineVoltsPerAmp: 0.005007173
PhaseCFineVoltsPerAmp: 0.0049246317
```

(continues on next page)

(continued from previous page)

```
LoopVelPGain: 0.004
LoopVelIGain: 0.001
LoopVelDGain: 0
LoopVelTGain: 0
MaxTorqueSlewRate: 65
OvercurrentThresholdAmps: 580
OvertempThresholdCelsius: 88
ResolverPolesPerMotorPole: 2
MaxNpCapacitorVoltage: 500
EnableHvilAlarm: 1
ExtTempSensorType: 2
DeviceIdentifier: ea4355b14d8b488c8829fae4e080912
HardwareRevision: 2
DisconnectedPhaseVoltageThresholdVolts: 30
DisconnectedPhaseCurrentThresholdAmps: 5
UndervoltageThresholdVolts: 5
CanTimeoutMs: 1000
DefaultMaxDcDischargeCurrent: 500
DefaultMaxDcChargeCurrent: -500
NumberOfMotorPolePairs: 10
```

To simplify the provisioning of fresh units, it is possible dump config to a file and restore it later.

```
$ ./steinmetz_tool config dump --output eeprom_contents.txt
$ ./steinmetz_tool config restore-dump  eeprom_contents.txt
Warning: skipping 'NumberOfBoots' (factory-only field)
Restored ResolverOffsetDegrees = 285
Restored LoopIqPGain = 0.414
Restored LoopIqIGain = 46.88
Restored LoopIqDGain = 0
Restored LoopIqTGain = 0
Restored LoopIdPGain = 0.414
Restored LoopIdIGain = 46.88
Restored LoopIdDGain = 0
Restored LoopIdTGain = 0
Restored TorqueConst = 1
Warning: skipping 'PhaseACoarseHallOffsetVolts' (factory-only field)
Warning: skipping 'PhaseBCoarseHallOffsetVolts' (factory-only field)
Warning: skipping 'PhaseCCoarseOffsetVolts' (factory-only field)
Warning: skipping 'PhaseACoarseVoltsPerAmp' (factory-only field)
Warning: skipping 'PhaseAFineHallOffsetVolts' (factory-only field)
Warning: skipping 'PhaseBFineHallOffsetVolts' (factory-only field)
Warning: skipping 'PhaseCFineOffsetVolts' (factory-only field)
Warning: skipping 'PhaseAFineVoltsPerAmp' (factory-only field)
Warning: skipping 'PhaseBCoarseVoltsPerAmp' (factory-only field)
Warning: skipping 'PhaseCCoarseVoltsPerAmp' (factory-only field)
Warning: skipping 'PhaseBFineVoltsPerAmp' (factory-only field)
Warning: skipping 'PhaseCFineVoltsPerAmp' (factory-only field)
Restored LoopVelPGain = 0.004
Restored LoopVelIGain = 0.001
```

(continues on next page)

(continued from previous page)

```
Restored LoopVelDGain = 0
Restored LoopVelTGain = 0
Restored MaxTorqueSlewRate = 65
Restored OvercurrentThresholdAmps = 580
Restored OvertempThresholdCelsius = 88
Restored ResolverPolesPerMotorPole = 2
Warning: skipping 'MaxNpCapacitorVoltage' (factory-only field)
Restored EnableHvilAlarm = 1
Restored ExtTempSensorType = 2
Warning: skipping 'DeviceIdentifier' (factory-only field)
Warning: skipping 'HardwareRevision' (factory-only field)
Restored DisconnectedPhaseVoltageThresholdVolts = 30
Restored DisconnectedPhaseCurrentThresholdAmps = 5
Restored UndervoltageThresholdVolts = 5
Restored CanTimeoutMs = 1000
Restored DefaultMaxDcDischargeCurrent = 500
Restored DefaultMaxDcChargeCurrent = -500
Restored NumberOfMotorPolePairs = 10

       hint: run `target/debug/steinmetz_tool config unlock` first to␣
↪restore factory-only fields, then re-run `restore-dump`
       hint: run `./steinmetz_tool_x86_64 config unlock` first to restore␣
↪factory-only fields, then re-run `restore-dump`
```

The text format of the dump contains a umber of <key>: <value> pairs, where <key> is one of the possible values emitted by `./steinmetz_tool_x86_64 config read --help`. The file does not need to exhaustively list every key.

Example:

```
NumberOfBoots: 0
ResolverOffsetDegrees: 285
LoopIqPGain: 0.414
LoopIqIGain: 46.88
LoopIqDGain: 0
LoopIqTGain: 0
LoopIdPGain: 0.414
LoopIdIGain: 46.88
LoopIdDGain: 0
LoopIdTGain: 0
TorqueConst: 1
PhaseACoarseHallOffsetVolts: 0
PhaseBCoarseHallOffsetVolts: 0
PhaseCCoarseOffsetVolts: 0
PhaseACoarseVoltsPerAmp: 0.0012143924
PhaseAFineHallOffsetVolts: 0
PhaseBFineHallOffsetVolts: 0
PhaseCFineOffsetVolts: 0
PhaseAFineVoltsPerAmp: 0.005004027
PhaseBCoarseVoltsPerAmp: 0.0012751862
```

(continues on next page)

(continued from previous page)

```
PhaseCCoarseVoltsPerAmp: 0.0010704114
PhaseBFineVoltsPerAmp: 0.005007173
PhaseCFineVoltsPerAmp: 0.0049246317
LoopVelPGain: 0.004
LoopVelIGain: 0.001
LoopVelDGain: 0
LoopVelTGain: 0
MaxTorqueSlewRate: 65
OvercurrentThresholdAmps: 580
OvertempThresholdCelsius: 88
ResolverPolesPerMotorPole: -2
MaxNpCapacitorVoltage: 500
EnableHvilAlarm: 1
ExtTempSensorType: 2
DeviceIdentifier: ea4355b14d8b488c8829fae4e080912
HardwareRevision: 2
DisconnectedPhaseVoltageThresholdVolts: 30
DisconnectedPhaseCurrentThresholdAmps: 5
UndervoltageThresholdVolts: 100
CanTimeoutMs: 1000
DefaultMaxDcDischargeCurrent: 500
DefaultMaxDcChargeCurrent: -500
NumberOfMotorPolePairs: 10
```

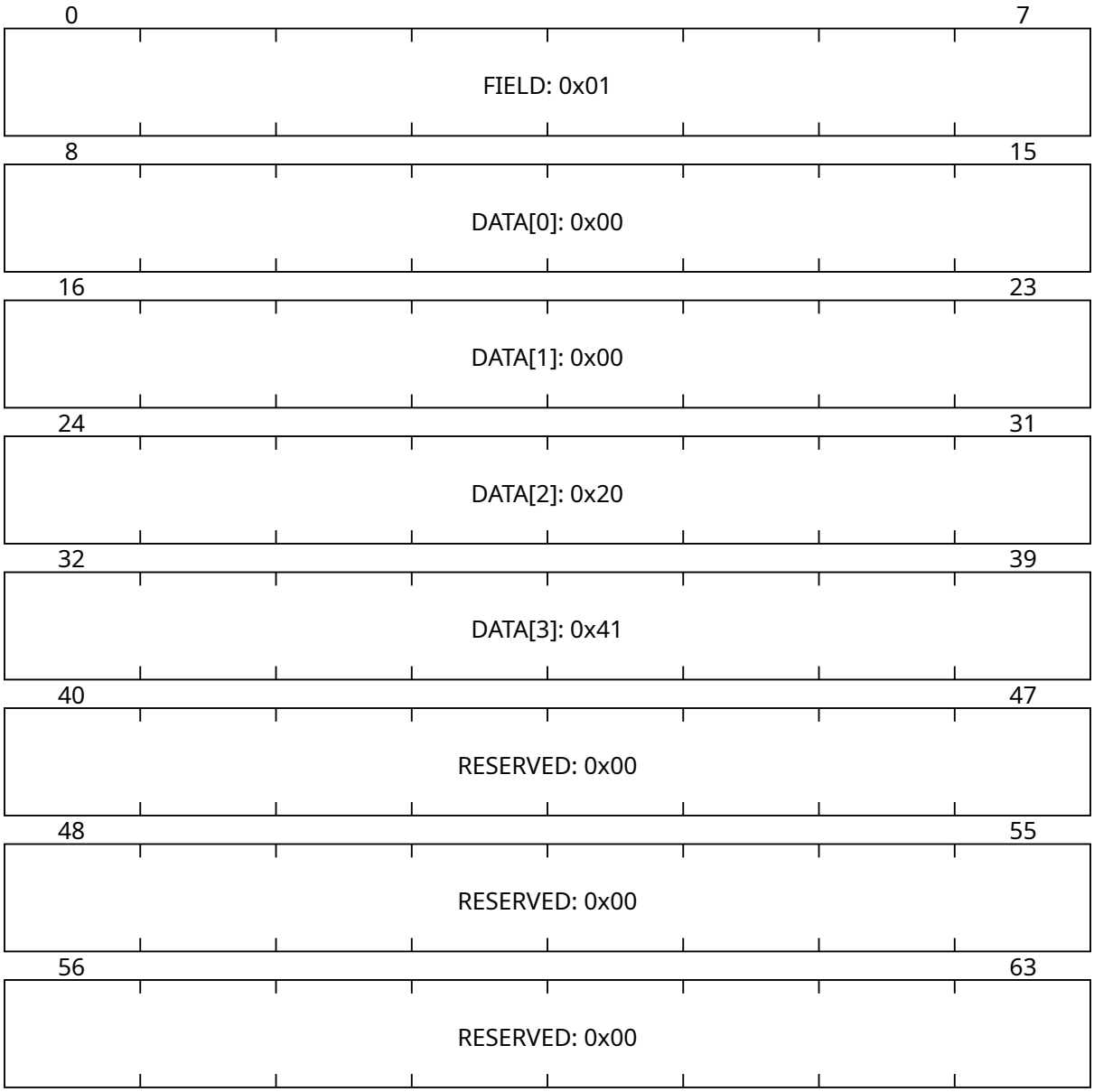
1.4.2.2 Over CAN

1.4.2.2.1 Writing

We will go through the example of writing the ResolverOffsetDegrees to be 10 degrees.

Step 1. Prepare CAN Packet

Begin by crafting the Write EEPROM CAN Packet as per the format described in *0x01a (EEPROM Set Parameter)*. ResolverOffsetDegrees is Field ID: 1. We want to write 10 degrees. If convert 10 to little-endian f32 we get [0x00, 0x00, 0x20, 0x41]. So we want to transmit a Standard CAN packet with ID 0x1a and data field as shown below:

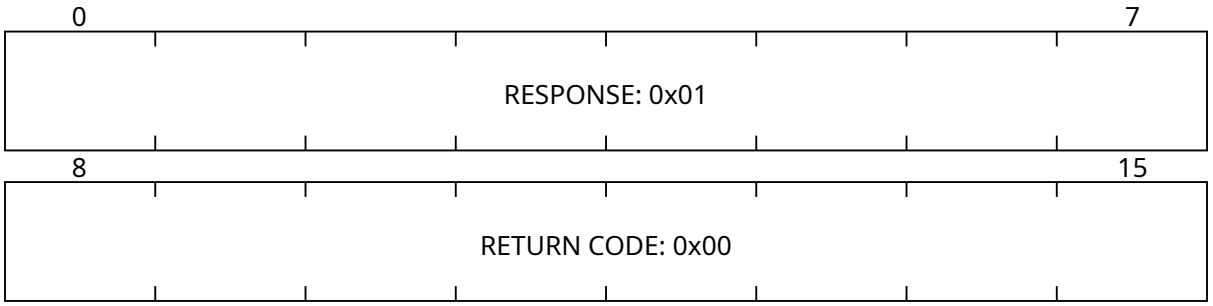


Step 2. Transmit CAN Packet

Ensure the motor controller is powered on and in the ReadyToSwitch state, see *State Machine*. Transmit the CAN packet.

Step 3. Validate ACK Packet

When you recieve the ACK packet as described in *0x01c (EEPROM Set Response)*, you should check it to validate the write was performed correctly. The data we should recieve back per this example is shown below.

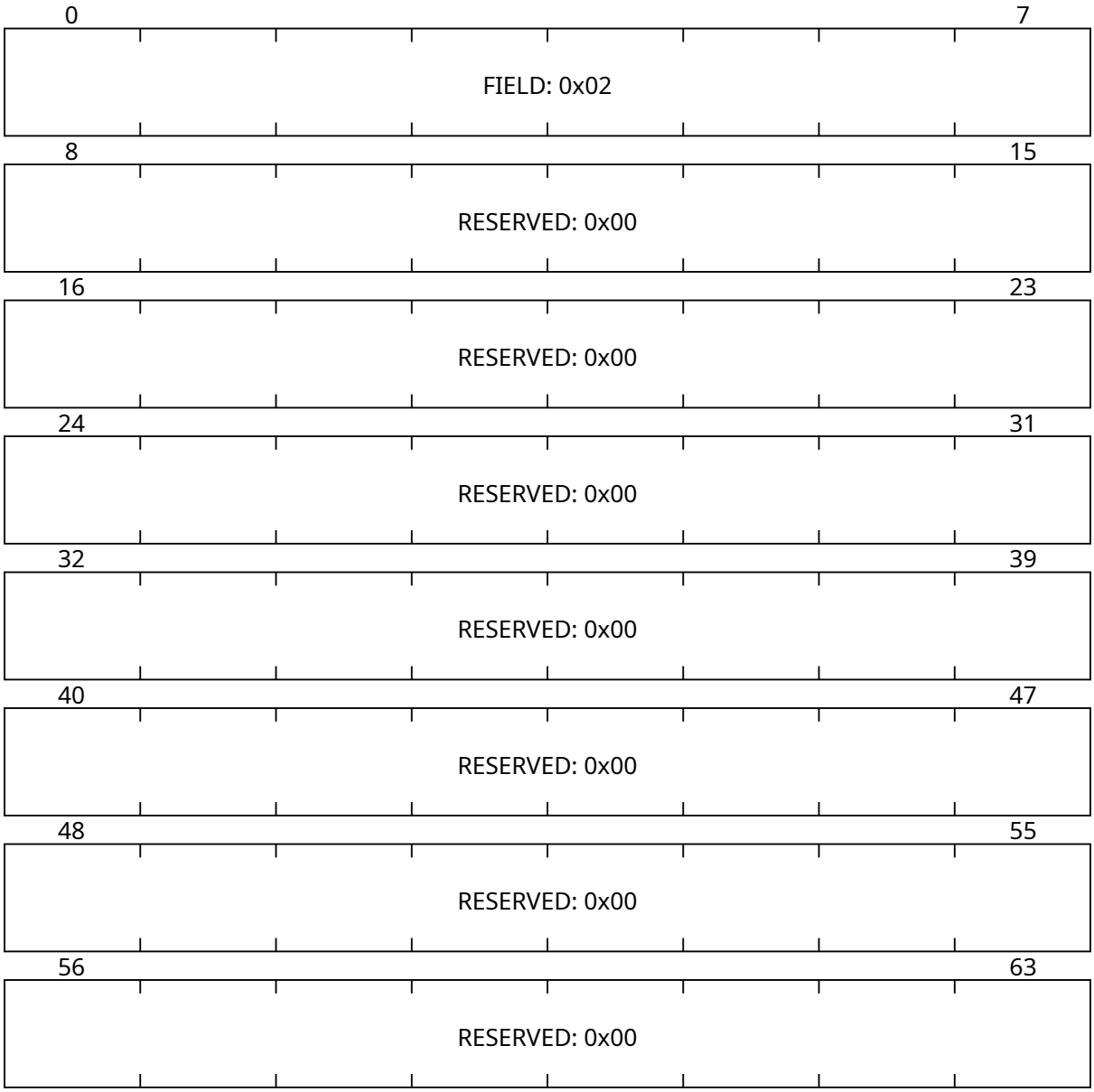


1.4.2.2.2 Reading.

We will go through the example of reading back the LoopIqPGain, for our case we will assume LoopIqPGain is 0.5.

Step 1. Prepare CAN Packet

Begin by crafting the Read EEPROM CAN Packet as per the format described in *0x01b (EEPROM Get Parameter)*. LoopIqPGain is field: 2. So we prepare the CAN packet data as shown below.

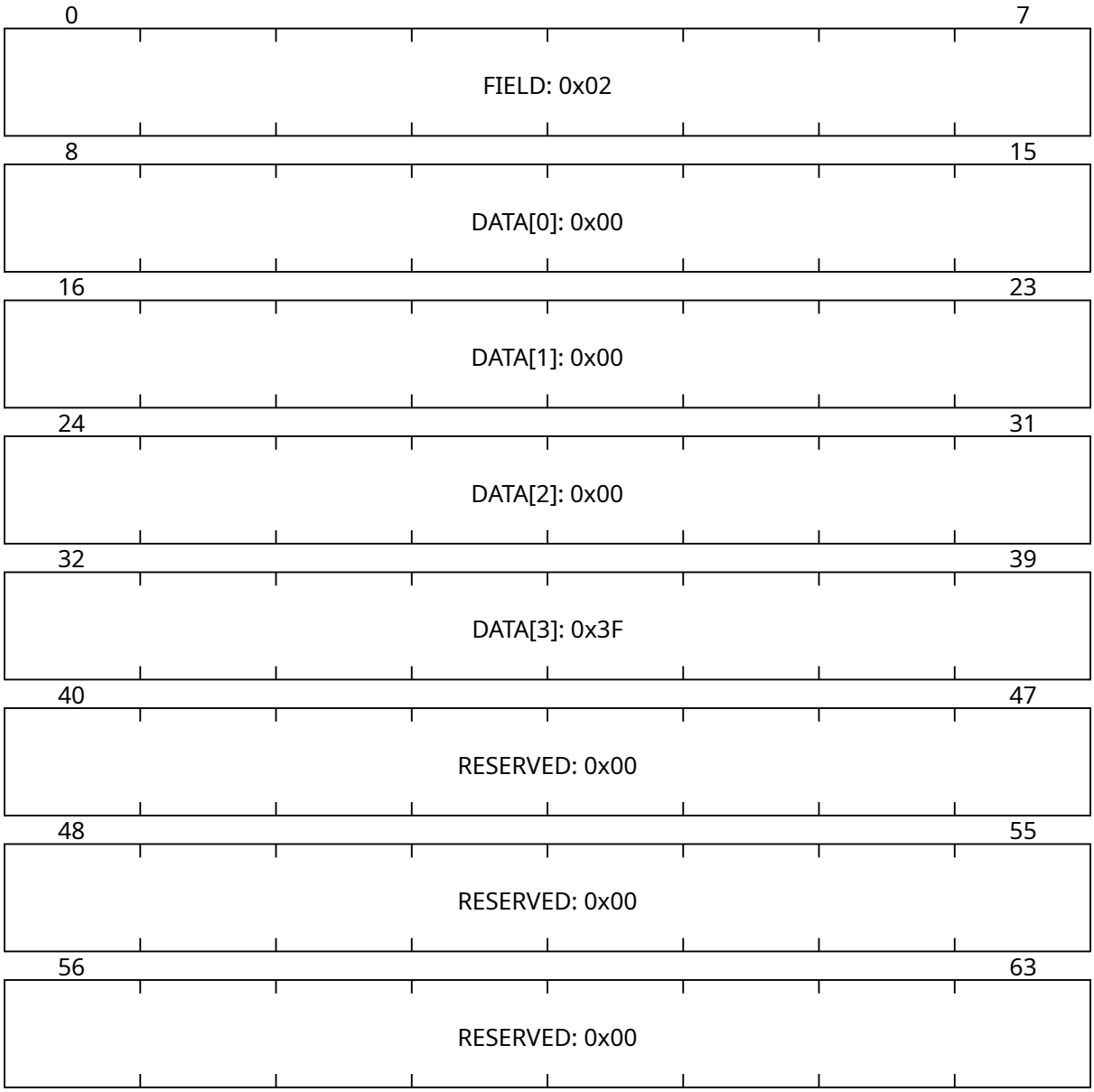


Step 2. Transmit CAN Packet

Ensure the motor controller is powered on and in the ReadyToSwitch state, see *State Machine*. Transmit the CAN packet.

Step 3. Validate Response Packet

When you recieve the response packet as described in *0x01d (EEPROM Get Response)*, you should check it to validate the write was performed correctly. The data we should recieve back per this example is shown below.


0
7
FIELD: 0x02
8
15
DATA[0]: 0x00
16
23
DATA[1]: 0x00
24
31
DATA[2]: 0x00
32
39
DATA[3]: 0x3F
40
47
RESERVED: 0x00
48
55
RESERVED: 0x00
56
63
RESERVED: 0x00

CHAPTER

TWO

MC0-500 HARDWARE DOCUMENTATION

2.1 Getting Started - Hardware

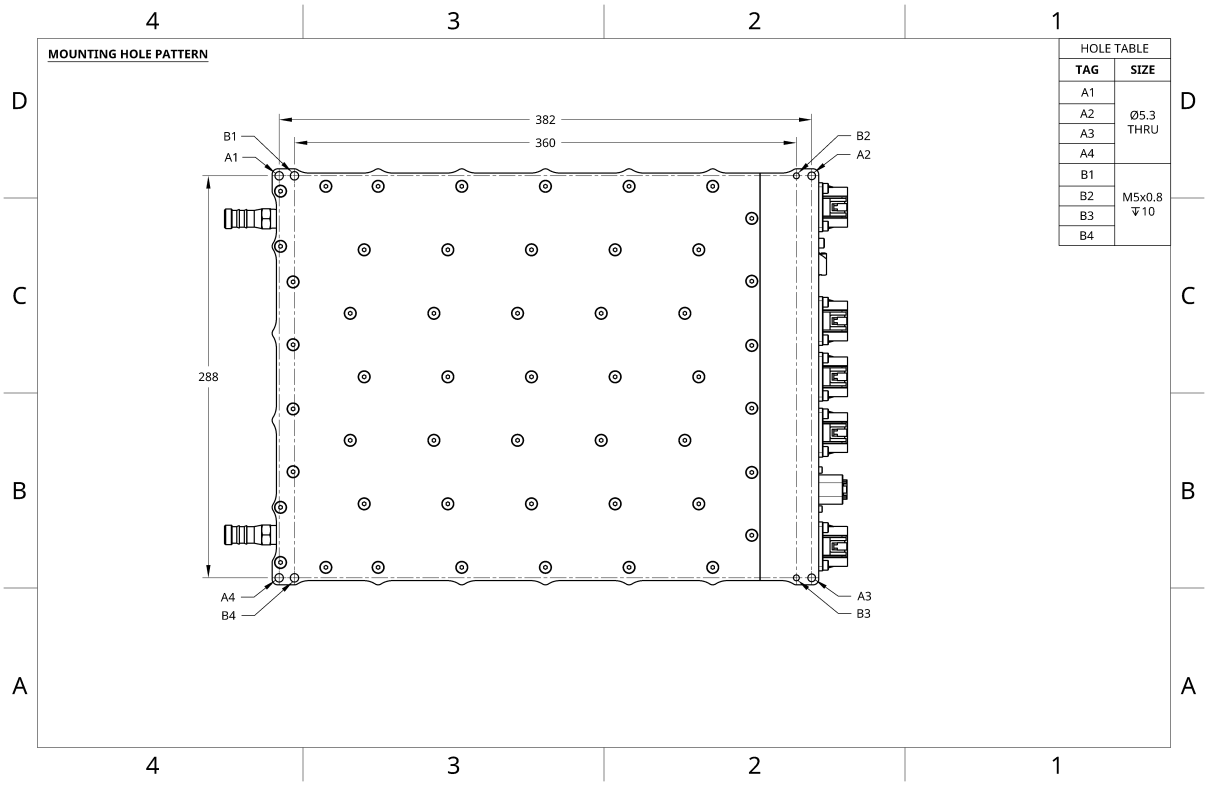
Welcome to the **Steinmetz HW - MC0-500** documentation.

This section should help you prepare your new MC0-500 unit for *Getting Started - Software*.

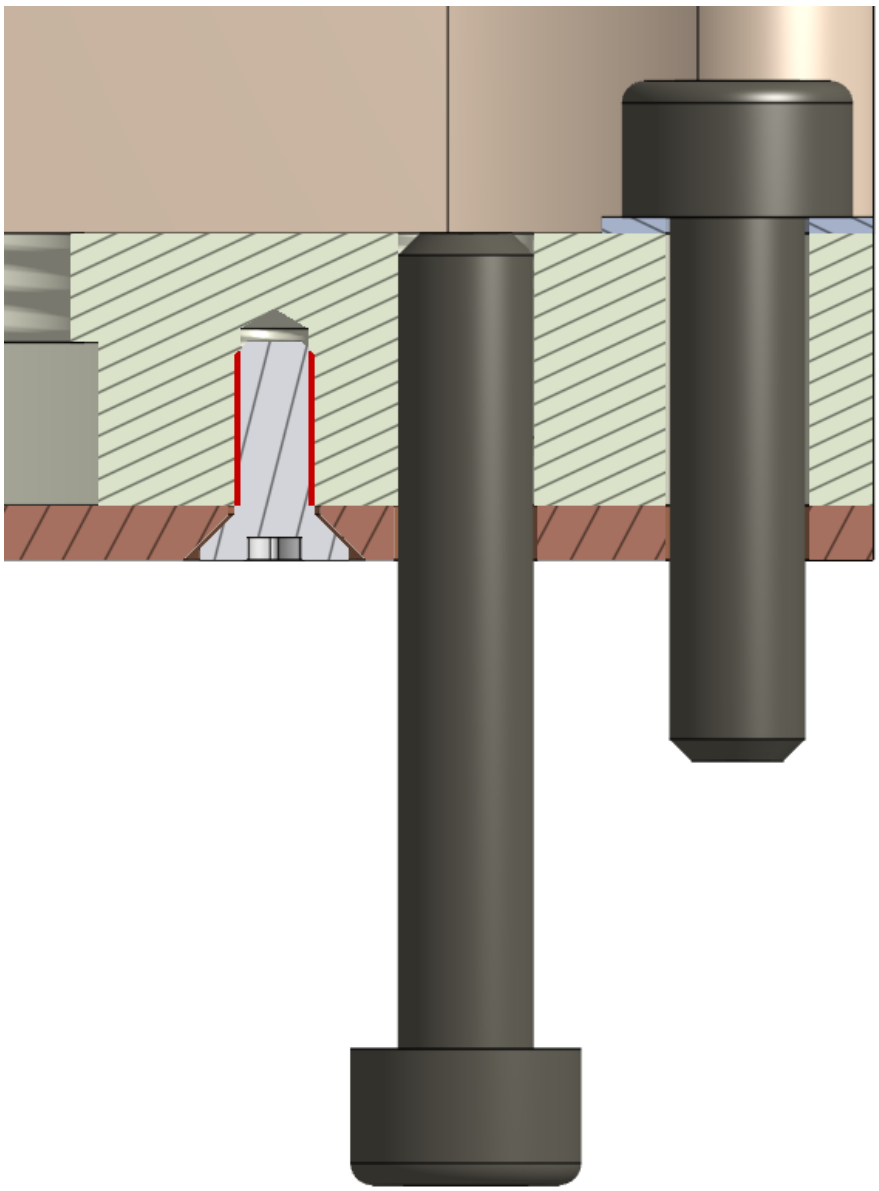
2.1.1 Mounting

MC0-500 should be securely fastened to a rigid, electrically conductive surface. Use a flat surface, avoid any point loading as it could damage the integrated cooling plate.

MC0-500 is designed to be mounted via four M5x0.80 fasteners. Both threaded and through-hole mounting options are provided. The corresponding bolt pattern as viewed from the enclosure bottom is:



The inner bolt pattern features threaded qty-4 M5x0.80 holes. The outer rectangular pattern features qty-4 ø5.3mm through holes. This allows for M5x0.80 bolts to be used in either orientation:



The use of a threadlocker is strongly recommended when using the threaded holes.

A torque of at least 4 N-m is recommended for all fasteners.

For additional mechanical information refer to *Mechanical*.

2.1.2 Quickstart Harnessing

The bare minimum required to run the unit is connecting all front-panel HV connections to the HV inputs and motor outputs. On the LV side, the following connections must be made:

- LV ground (black wires with banana jack on provided harness)
- +LV_IN, 11 V to 52 V (red wire with banana jack on provided harness)
- CAN (green and yellow wires on provided harness)
- Resolver (group of 3 sets of twisted pairs EXC, COS, SIN on provided harness) For additional information on pinout refer to *Connectors*.

Cooling should be attached and provided with 25 L/min of flow rate, and inlet temperatures should be limited to 60 °C for maximum performance.

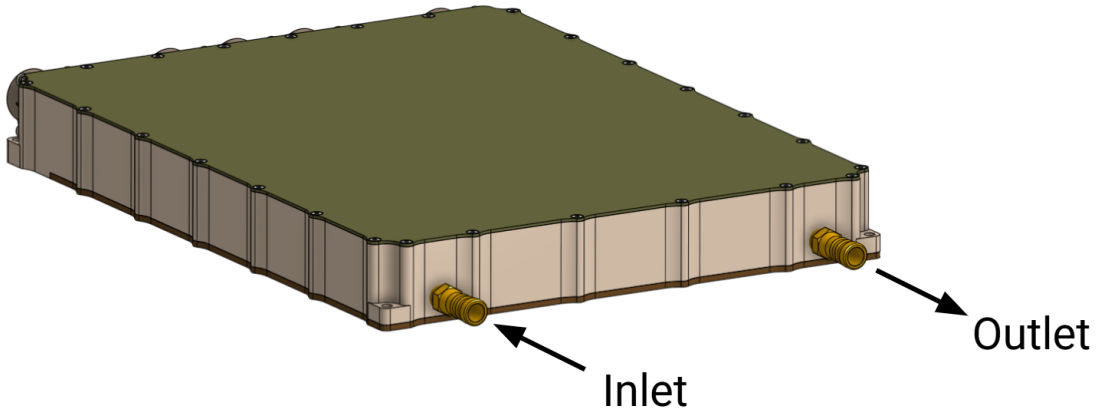
2.2 Cooling

2.2.1 Fittings

Two coolant ports are fitted to the rear of the MC0-500 enclosure. These ports are used for motor controller cooling under normal operation.

The coolant ports and cooling geometry are symmetric and coolant flow is allowable in either direction.

However, due to internal heating characteristics it is recommended that when viewed from the rear, the left port is used as the inlet, and right port as the outlet.



As-provided, MC0-500 ships with two straight 1/2” barbed tube fittings (McMaster Carr 5357K39). This is meant to be used with a flexible hose with an ID of 1/2”.

When selecting tubing, ensure that it is rated for the expected coolant pressure at elevated coolant temperatures during normal operation.

The stock fittings can be replaced with any 1/4”-18 NPT threaded fitting. Use care when tightening fittings, it is easy to overtighten the threads. It is recommended to first fully hand tighten, then use a wrench to tighten an additional half turn.

It is recommended to use only plastic or aluminum-based components within the MC0-500’s loop. Mixing of other metals can cause galvanic corrosion.

2.2.2 Coolant

The coolant plate is rated for a maximum inlet pressure of 3.1 bar (45 psi) relative to ambient. Pressures in excess of the maximum may cause leakage of coolant and degraded performance.

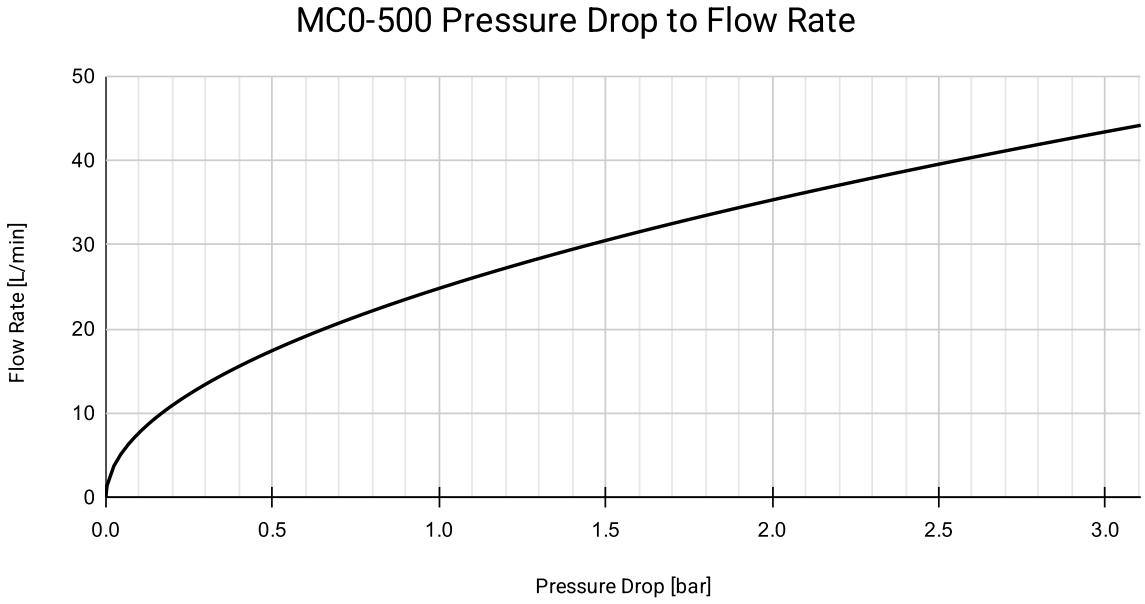
Coolant consisting of a commercially available 50/50 mix of water and glycol is required for operation. Do not use plain water as this will cause corrosion. The maximum recommended inlet temperature is 60C at full load operation.

2.2.3 Flow Requirements

The recommended flow rate to MC0-500 is 25 LPM. This corresponds to a pressure drop across the inverter of about 1 bar (15psi).

The maximum junction temperature of the internal FETs is 150C. At full load, the temperature delta between the inlet temperature and maximum junction is approximately 90C.

The pressure drop to flow rate relation of the MC0-500 coldplate is:



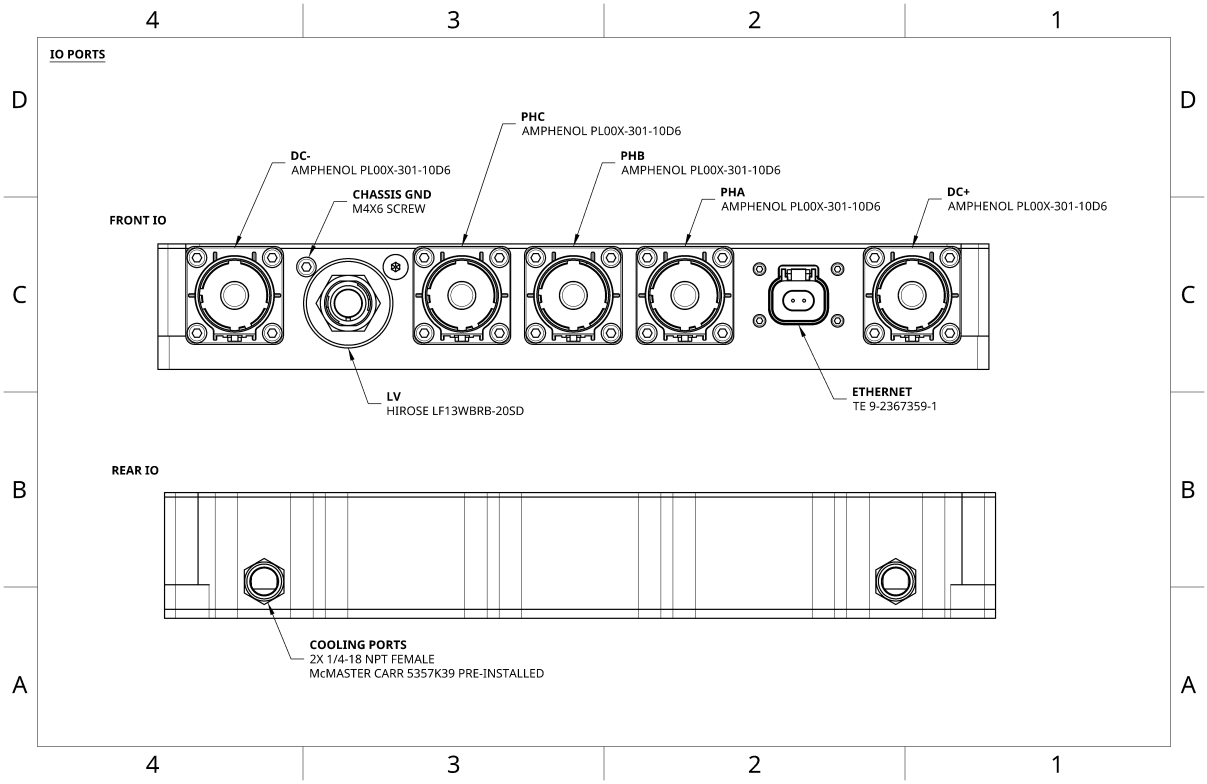
2.3 Connectors

The front face of MC0-500 has all of the device's connectors. When facing the front of the device, the connectors are, from left to right:

- HV Minus
- LV Connector
- PHC
- PHB
- PHA
- Automotive Ethernet
- HV Positive

Name	Location	Connector
DC +	Front left	Amphenol PL00X-301
Phase C	Centre left	Amphenol PL00X-301
Phase B	Centre	Amphenol PL00X-301
Phase A	Centre right	Amphenol PL00X-301
DC -	Front right	Amphenol PL00X-301
Low Voltage IO	Front left	Hirose F13WBRB-20SD
Automotive Ethernet	Front right	TE 9-2367359-1

The I/O ports are as follows:



All connectors on MC0-500 have an IP67 seal or better when mated with manufacturer-specified mating parts and properly sized wiring.

2.3.1 LV Connector

The LV connector on MC0-500 is the `LF13WBRB-20SD` from HIROSE. When mated, this connector achieves an IPX7/IPX8 rating. The mating part is either `LF13WBP-20P` or `LF13WBLP-20P`; both come in crimp and solder varieties and support shielding. The provided harness follows the diagram below, which also includes pinout information for the LV connector. Refer to the HIROSE documentation for additional information.

The pinout is as follows. The wire color and twisted pair group columns refer to the included shielded pigtail harness.

2.3.2 LV Connector Pinout

Pin	Signal	Description	Wire Color (included harness)	Twisted Pair Group (included harness)
1	+LV_IN	Main LV supply input	Red	Red / Blue
2	CAN_L	CAN Low	Green	Yellow / Green
3	CAN_H	CAN High	Yellow	Yellow / Green
4	EXT_GPIO_0	External GPIO	Purple	Black / Purple
5	EXT_GPIO_1	External GPIO	White	White / Pink
6	GND	Ground	Blue	Red / Blue
7	GND	Ground	Black	Black / Purple
8	ETHERNET_N	100BASE-T1 Automotive Ethernet -	Brown	Brown / Gray
9	RESOLVER_COS_P / ENCODER_A_P	Resolver COS+ / Encoder A+	Pink-Gray	Pink-Gray / Blue-Red
10	EXT_RTD_P	External RTD Positive	White-Green	White-Green / Green-Brown
11	EXT_GPIO_2	External GPIO	Pink	White / Pink
12	RESOLVER_SIN_N / ENCODER_B_N	Resolver SIN- / Encoder B-	Pink-White	Pink-Brown / Pink-White
13	ETHERNET_P	100BASE-T1 Automotive Ethernet +	Gray	Brown / Gray
14	RESOLVER_SIN_P / ENCODER_B_P	Resolver SIN+ / Encoder B+	Pink-Brown	Pink-Brown / Pink-White
15	RESOLVER_EXC_P / ENCODER_Z_P	Resolver Excitation+ / Encoder Z+	White-Gray	White-Gray / Gray-Brown
16	EXT_RTD_N	External RTD Negative	Green-Brown	White-Green / Green-Brown
17	RESOLVER_COS_N / ENCODER_A_N	Resolver COS- / Encoder A-	Blue-Red	Pink-Gray / Blue-Red
18	+LV_IN	Main LV supply input	White-Yellow	White-Yellow / Brown-Yellow
19	ENCODER_POWER	Encoder supply (5V)	Brown-Yellow	White-Yellow / Brown-Yellow
20	RESOLVER_EXC_N / ENCODER_Z_N	Resolver Excitation- / Encoder Z-	Gray-Brown	White-Gray / Gray-Brown

The LV input supports 11 V to 52 V operation on `+LV_IN`.

2.3.3 Ethernet Connector

The Ethernet connector on MC0-500 is `9-2367359-1` from TE. This connector and the associated hardware in MC0-500 are configured for 100BASE-T. When mated, this connector achieves IP6K9K sealing. At this time, the firmware interface is not implemented.

The recommended mating connector is `9-2367337`, which is available in a variety of forms.

2.3.4 HV Connectors

Warning

If the correct OD wire is not used with the mated connector, the IP67 rating will not be achieved. Pay close attention to the required wire diameter when sourcing the harness.

MC0-500 has five high-voltage connectors, all of which are `PL00X-301` connectors from Amphenol Industrial. Each has its HVIL (High Voltage Interlock) daisy-chained internally, and a fault is detected if any connector is improperly seated. To run MC0-500 at its maximum rated current, the mating part should be crimped with 70 mm2 or larger wire. When mated, this connector achieves an IP67 seal.

The mating parts are `PL18X-301`, `PL28X-301`, `PL10X-301`, and `PL20X-301`.

2.3.5 Grounding

An M4x0.7x6 socket head cap screw is provided on the front of the enclosure, located above the low voltage connector.

Provide a continuous low impedance connection between MC0-500's grounding point, chassis ground, and the motor's ground.

2.4 Environmental Sealing

MC0-500 is IP67 rated when *all* connectors and coolant fittings are correctly mated or installed, and the harness uses wiring that meets the connector manufacturer's wire diameter, crimp, and seal requirements.

If the wiring is not correctly sized or assembled to seal to its connectors, the IP67 rating will not be achieved. Carefully follow the connector manufacturer specifications when building or sourcing harnesses.

2.5 Mechanical

MC0-500 has dimensions of 45 mm x 298 mm x 392 mm (HxWxD), with a dry mass of 7.4 kg and a displacement volume of 4.8 L.

2.5.1 I/O

The front of the device features all the major electrical interfaces, including five high-voltage connectors, a low-voltage connector, and an automotive Ethernet connector. A threaded hole for chassis grounding is located next to the LV connector. Refer to *Connectors* for additional information.

M5 clearance and M5x0.8 threaded holes are provided on each corner of the inverter for mounting.

At the rear of the unit are the cooling ports, which are preinstalled with two 1/2" barb fittings. These can be swapped for fittings of the customer's choosing; refer to *Cooling* for additional information.

2.5.2 Drawing

`Download PDF`

2.5.3 3D CAD

Download 3D .step

CHAPTER

THREE

MC0-250 HARDWARE DOCUMENTATION

3.1 MC0-250 Coming Soon