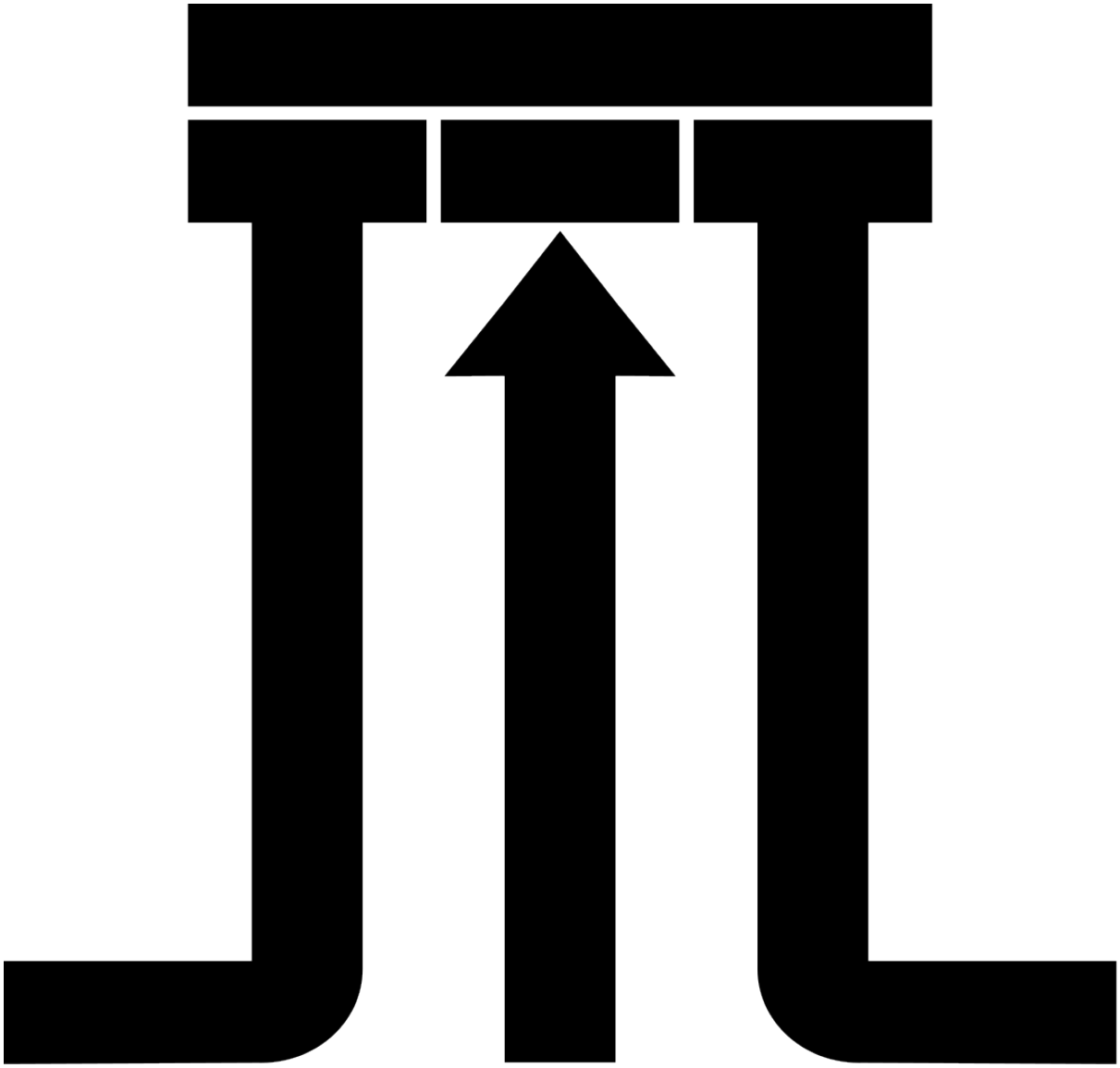


Steinmetz Motor Controllers

Steinmetz, Inc.

December 22, 2025

SOFTWARE

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 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.

1.1.4 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 2: 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 3: 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 4: 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 5: 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.
0x025	Electrical Angle	Processed information on Electrical Angle
0x029	Switching Misc Data	Extra telemetry data from switching.
0x306	GPIO State	External GPIO Data
0x310-0x31B	Temperature Data	Individual Thermistor Data
0x320-0x324	Current and Voltage Sensor Readings	Phase Current and Voltage Sense Readings

1.3.1.2 0x012 (Resolver)

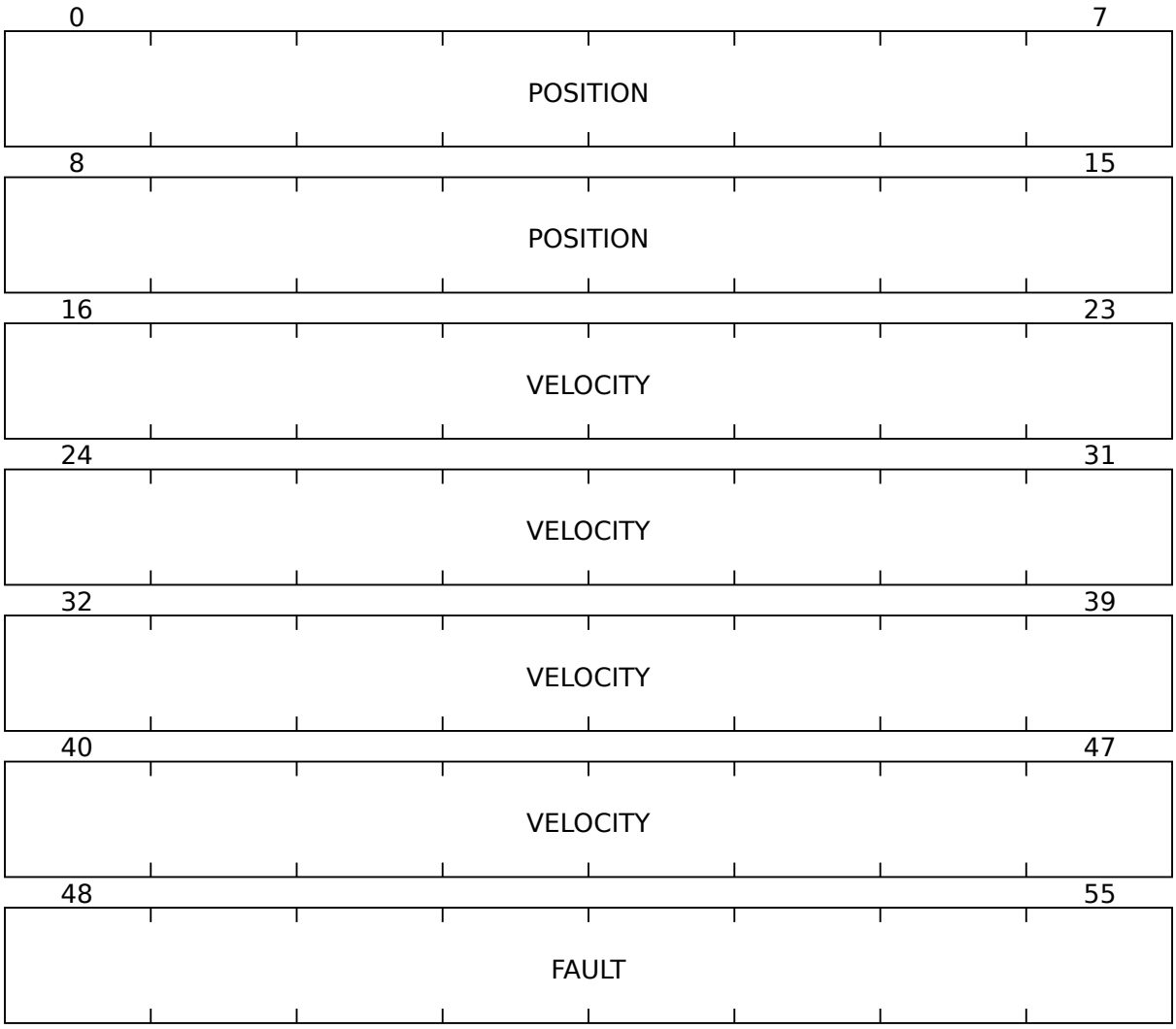


Table 6: Resolver Packet

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)

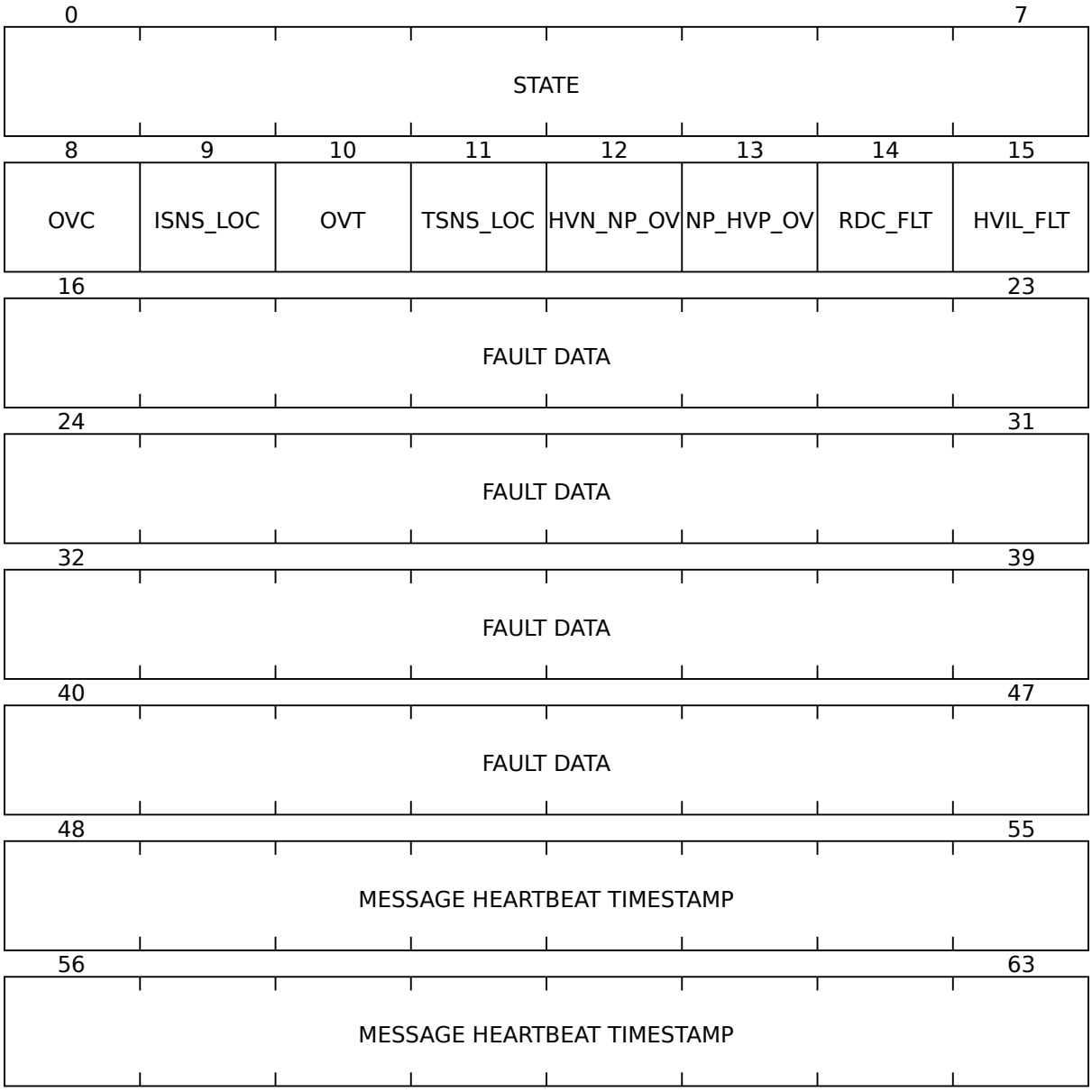


Table 7: State ID Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
STATE	State ID	N/A	u8
FAULT BITS	Bitfield describing fault state	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
FAULT DATA	Extra debugging data for fault	N/A	f32
MESSAGE HEARTBEAT TIMESTAMP	Milliseconds since last heartbeat. Saturates at 65535ms.	1	u16

1.3.1.4 0x01c (EEPROM Set Response)

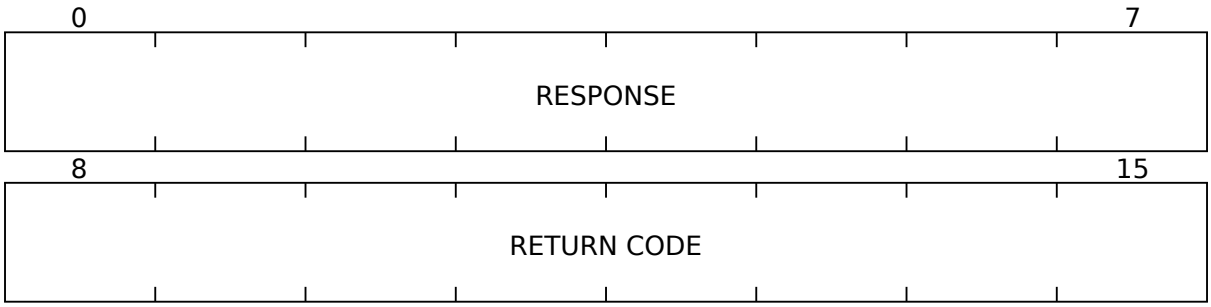


Table 8: EEPROM Set Response Packet

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	If RETURN CODE = 0x00 then successfully written, else it failed. Typical failure code is 0xFF	N/A	u8

1.3.1.5 0x01d (EEPROM Get Response)

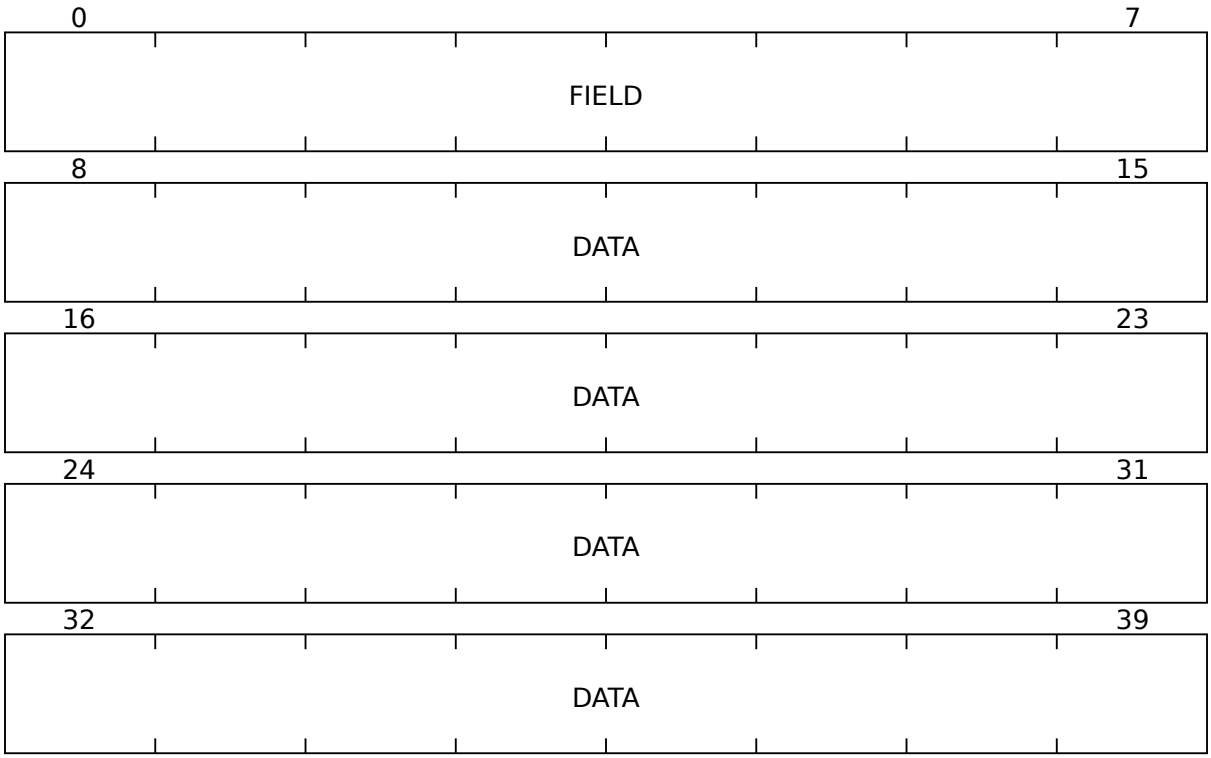


Table 9: EEPROM Get Response Packet

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)

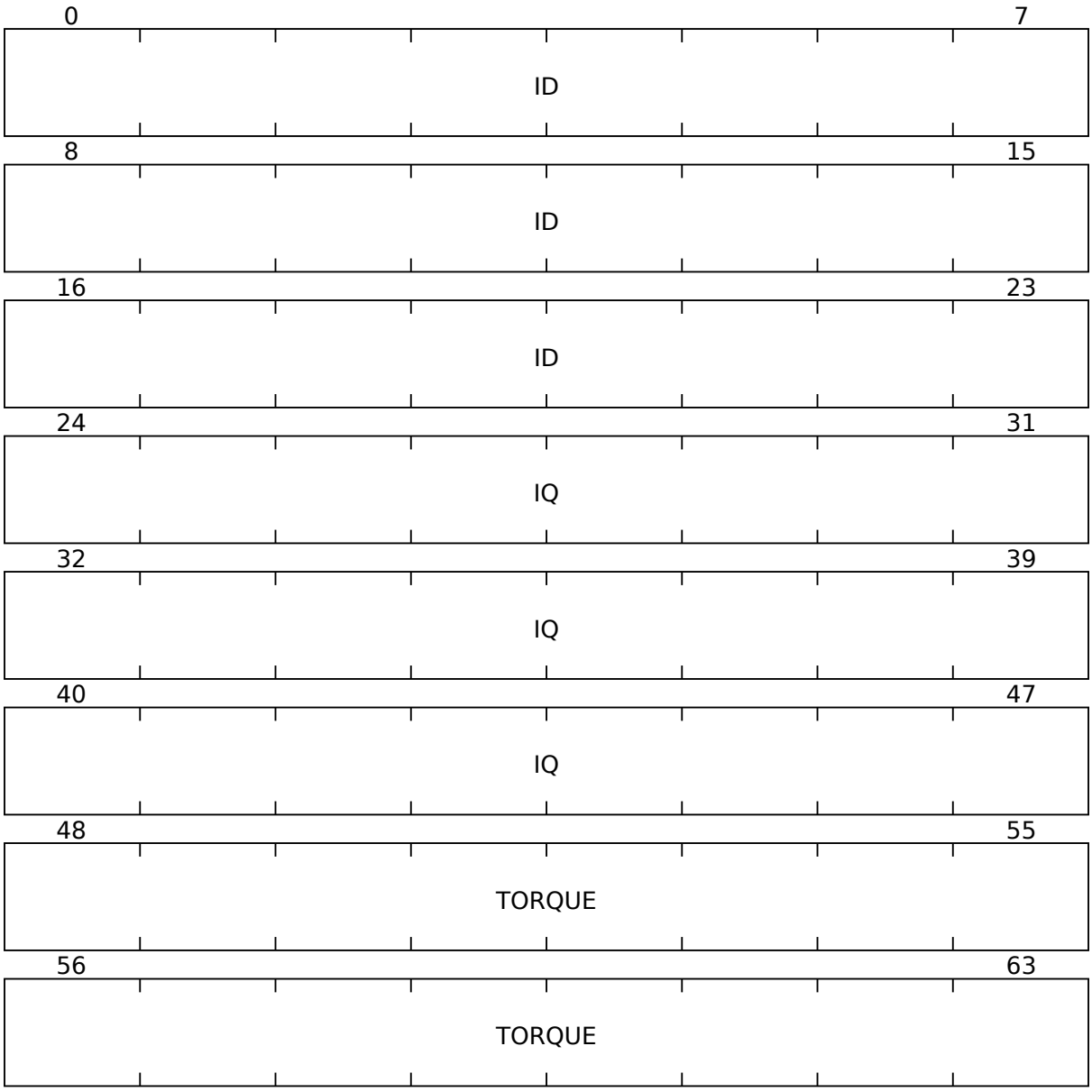


Table 10: DQ Currents Packet

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)

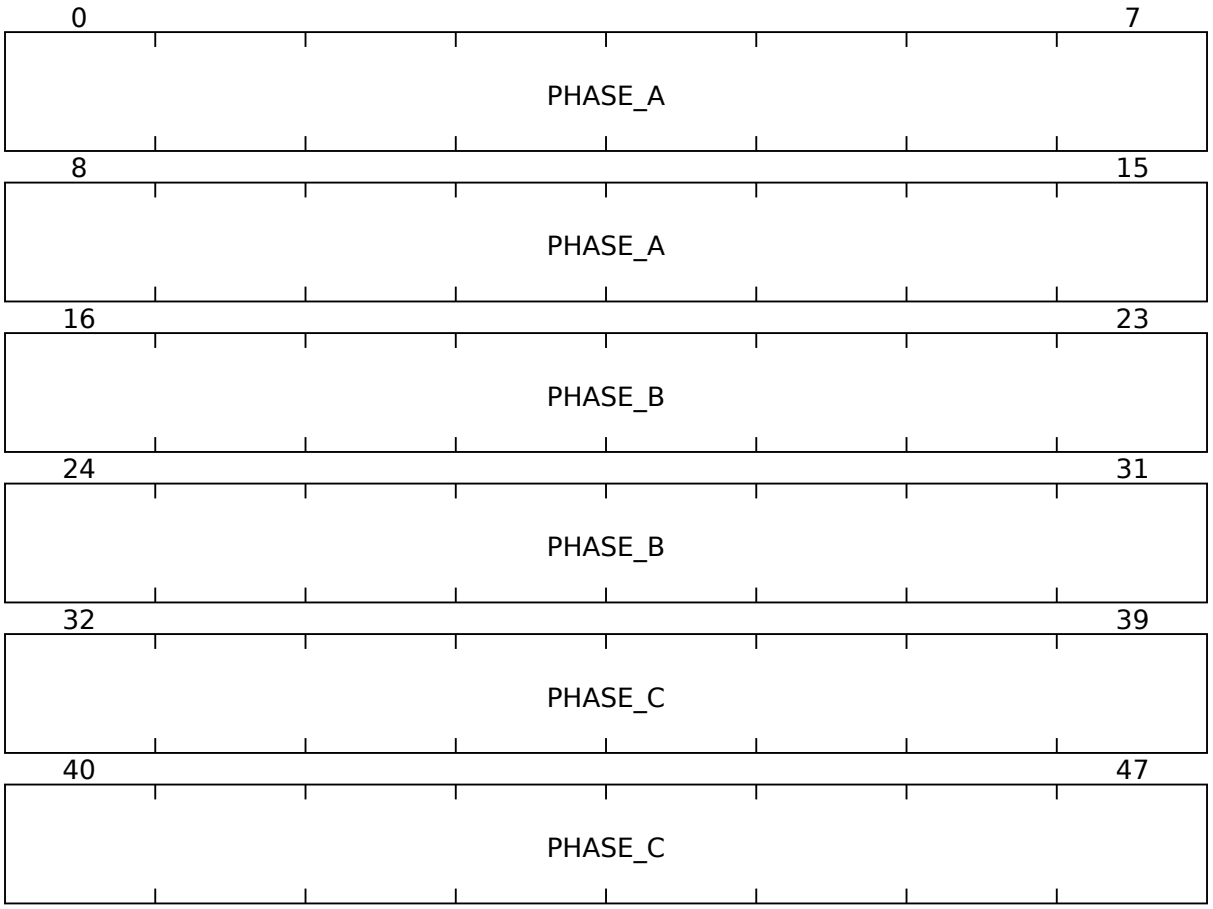


Table 11: PWM Duty Cycles Packet

Data Word	Description	Scaling (Divide by X to get real value)	Datatype
PHASE_A	Duty cycle of Phase A. Duty cycle is the percentage of time that the high FET is on in every switching cycle.	100	i16
PHASE_B	Duty cycle of Phase B. Duty cycle is the percentage of time that the high FET is on in every switching cycle.	100	i16
PHASE_C	Duty cycle of Phase C. Duty cycle is the percentage of time that the high FET is on in every switching cycle.	100	i16

1.3.1.8 0x022 (DQ PID Integral)

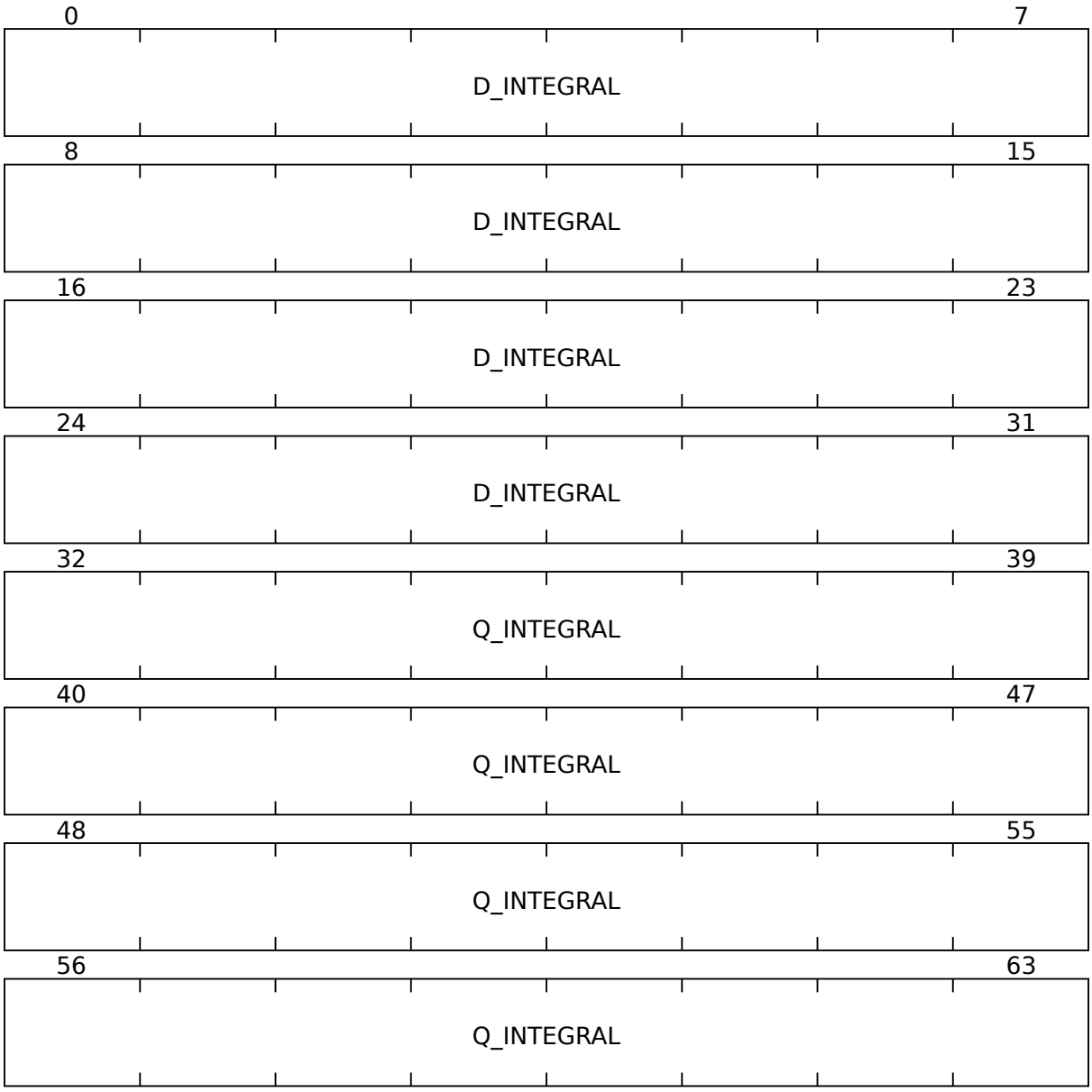


Table 12: DQ PID Integral Packet

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	1	f32
Q_INTEGRAL	The accumulated integral term for the Q side of the PID loop	1	f32

1.3.1.9 0x025 (Electrical Angle)

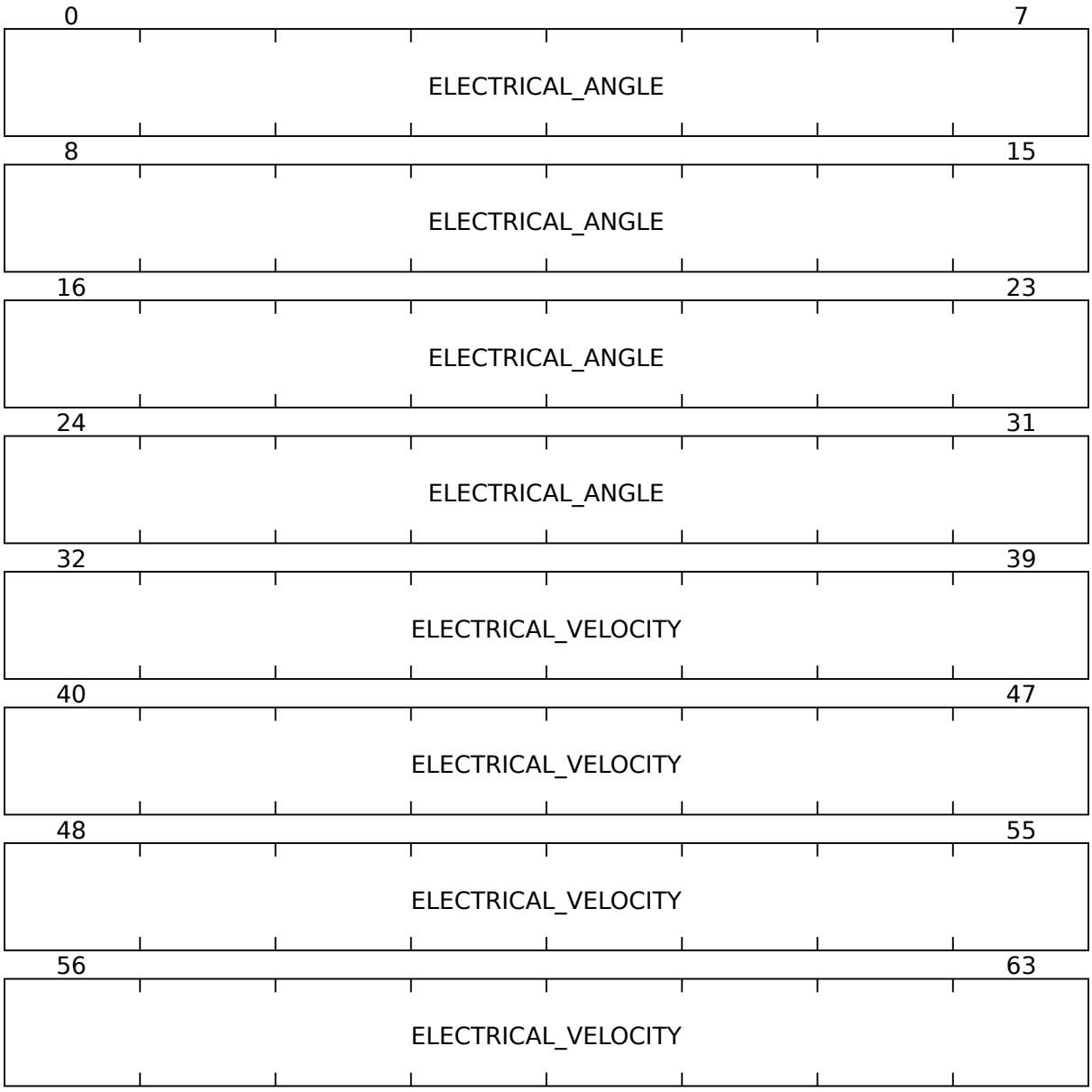


Table 13: Electrical Angle Packet

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

1.3.1.10 0x029 (Switching Misc Data)

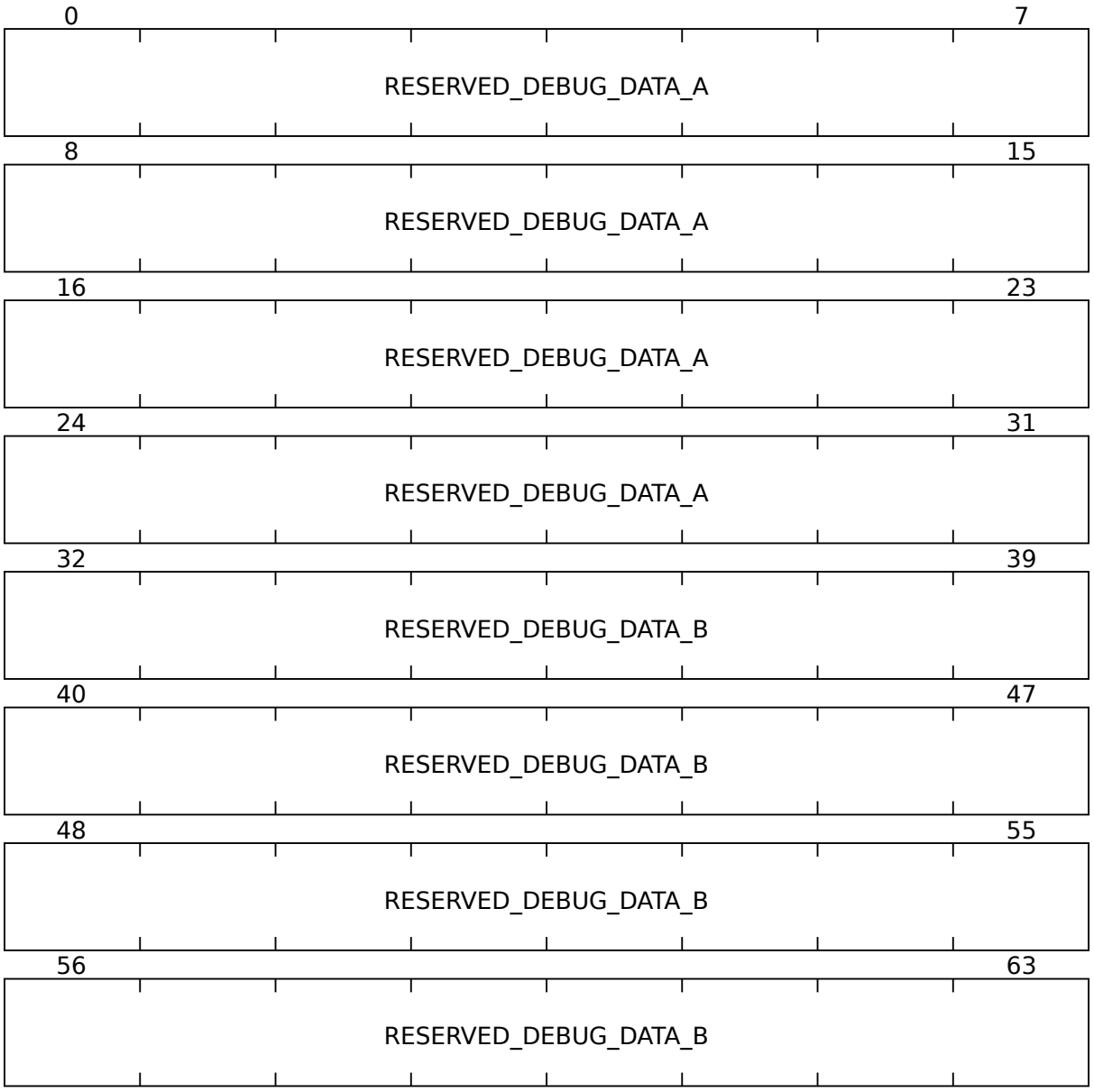


Table 14: Switching Misc Data

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

1.3.1.11 0x306 (GPIO State)

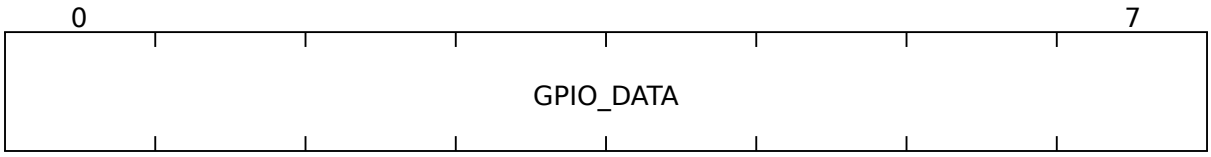


Table 15: GPIO State

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

1.3.1.12 0x310-0x31B (Temperature Data)

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

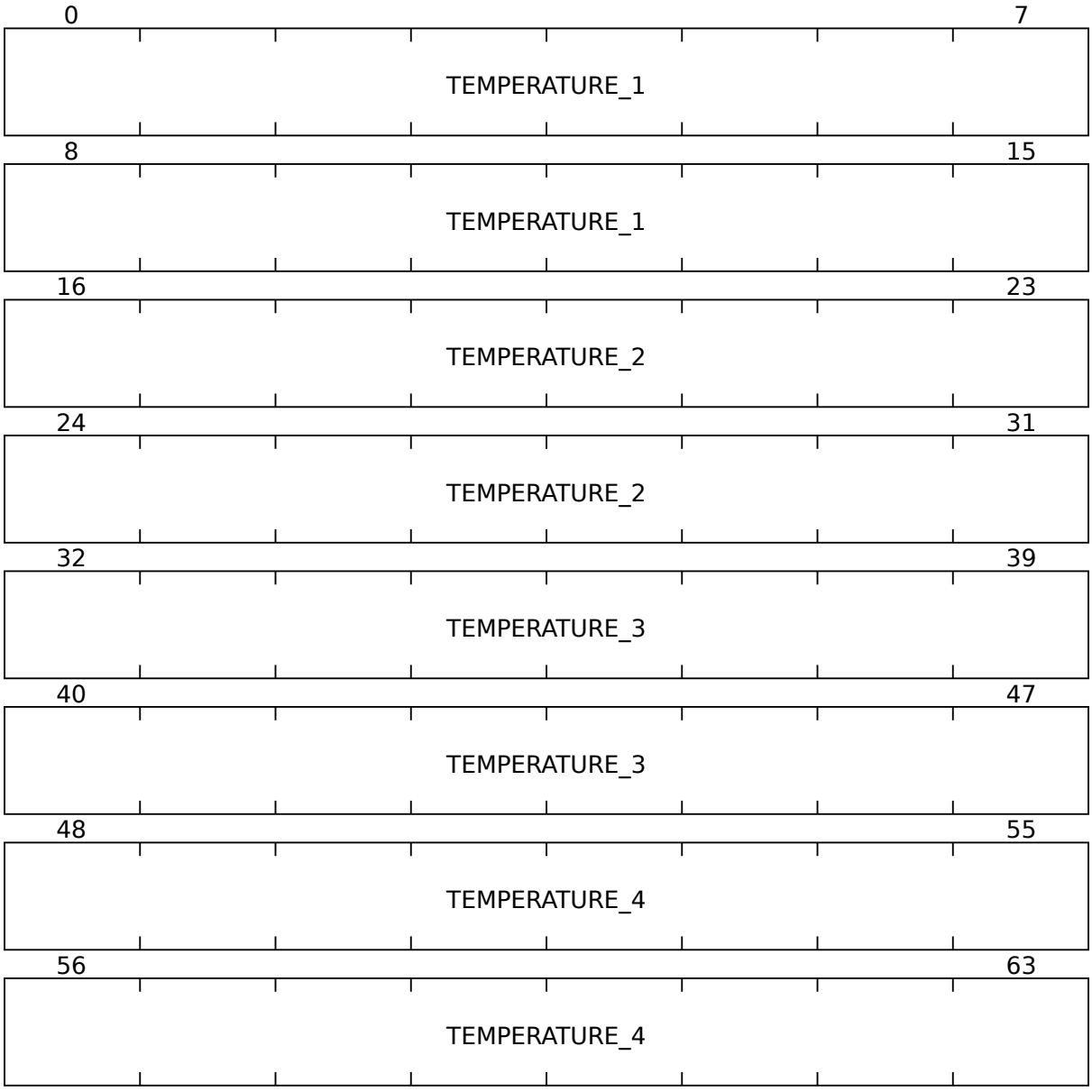


Table 16: Temperature Data

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

1.3.1.13 0x31C (External RTD interface)

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

Table 17: Temperature Data

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

1.3.1.14 0x320 (Current Sense Phase C)

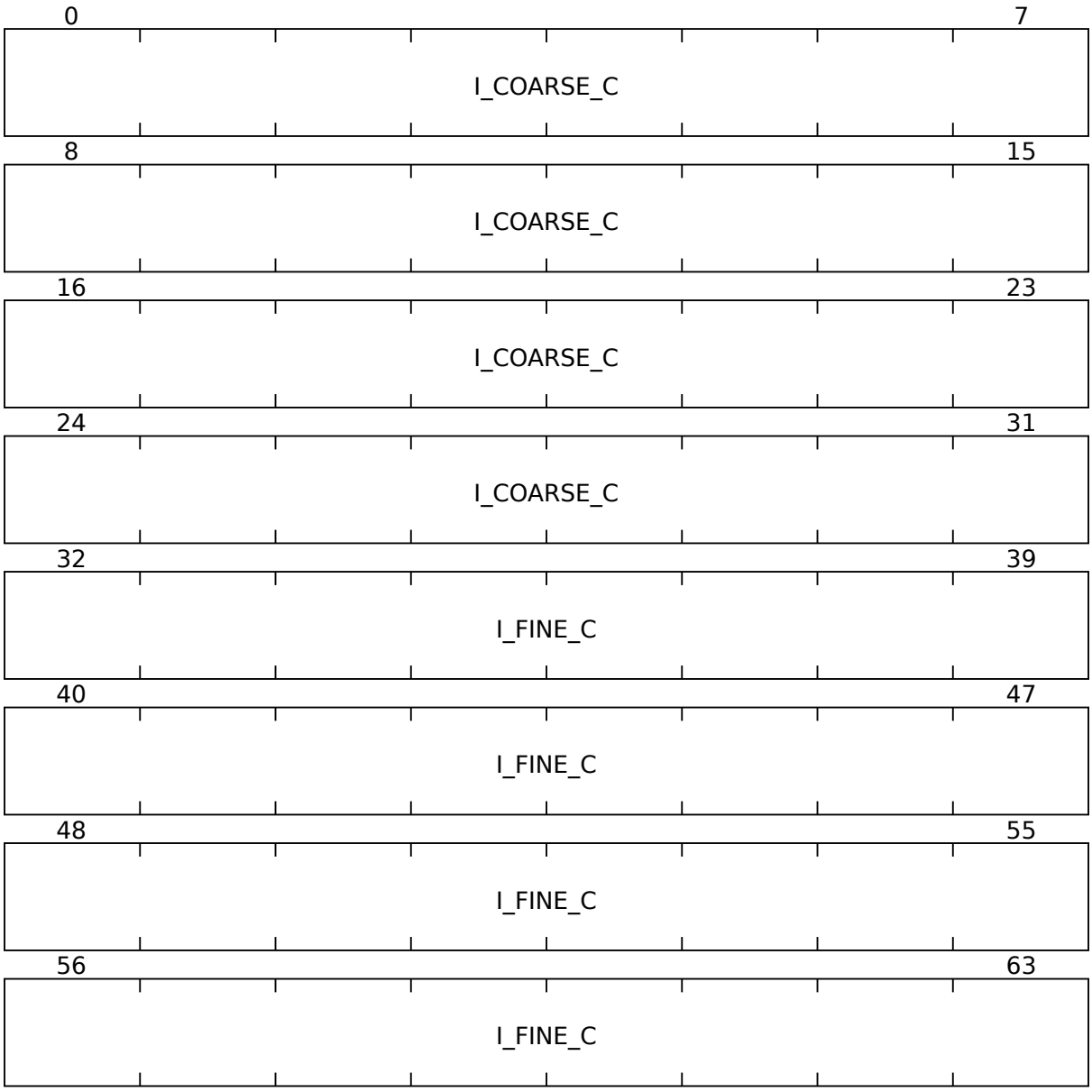


Table 18: Current Sense Phase C

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

1.3.1.15 0x321 (Current Sense Phase B)

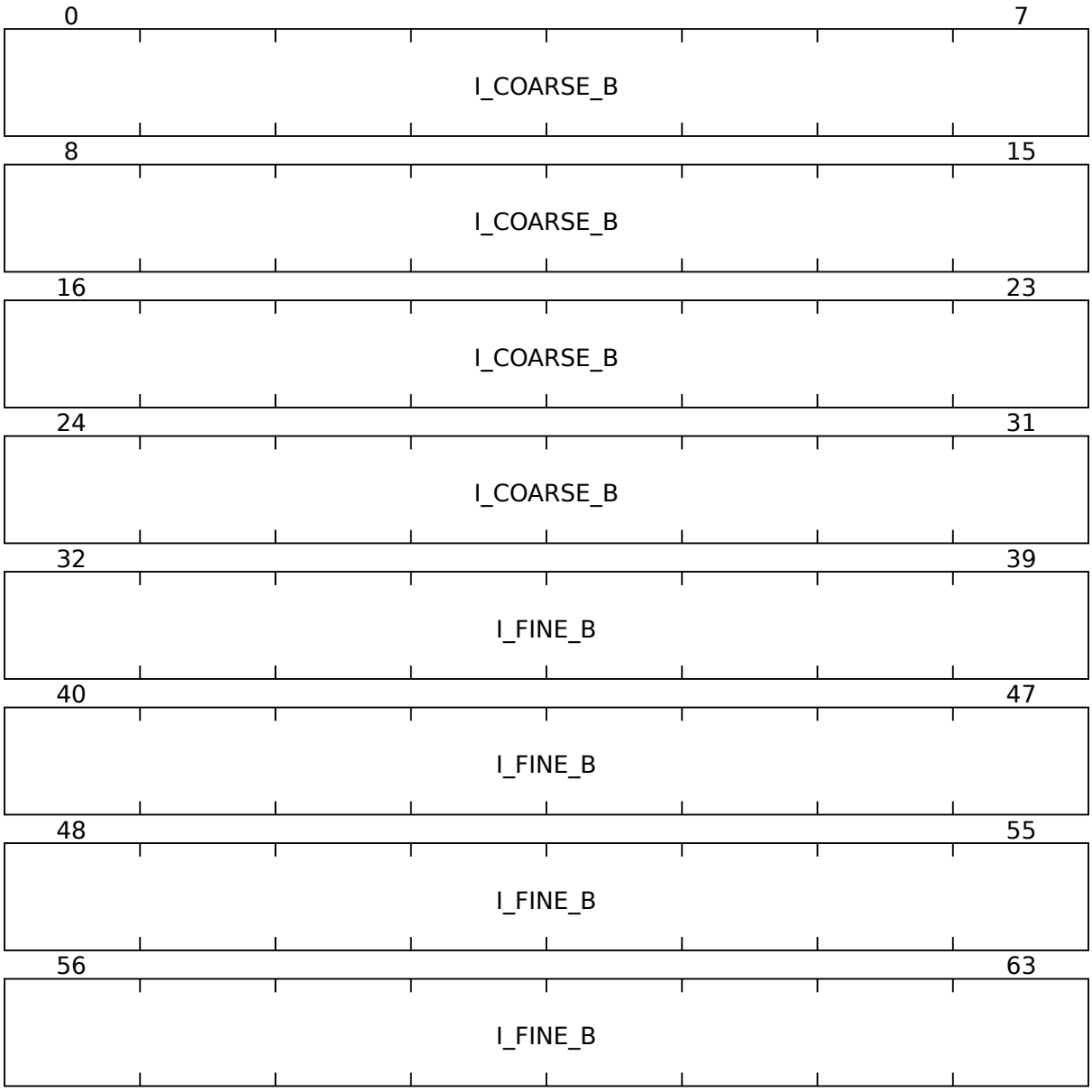


Table 19: Current Sense Phase B

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

1.3.1.16 0x322 (Current Sense Phase A)

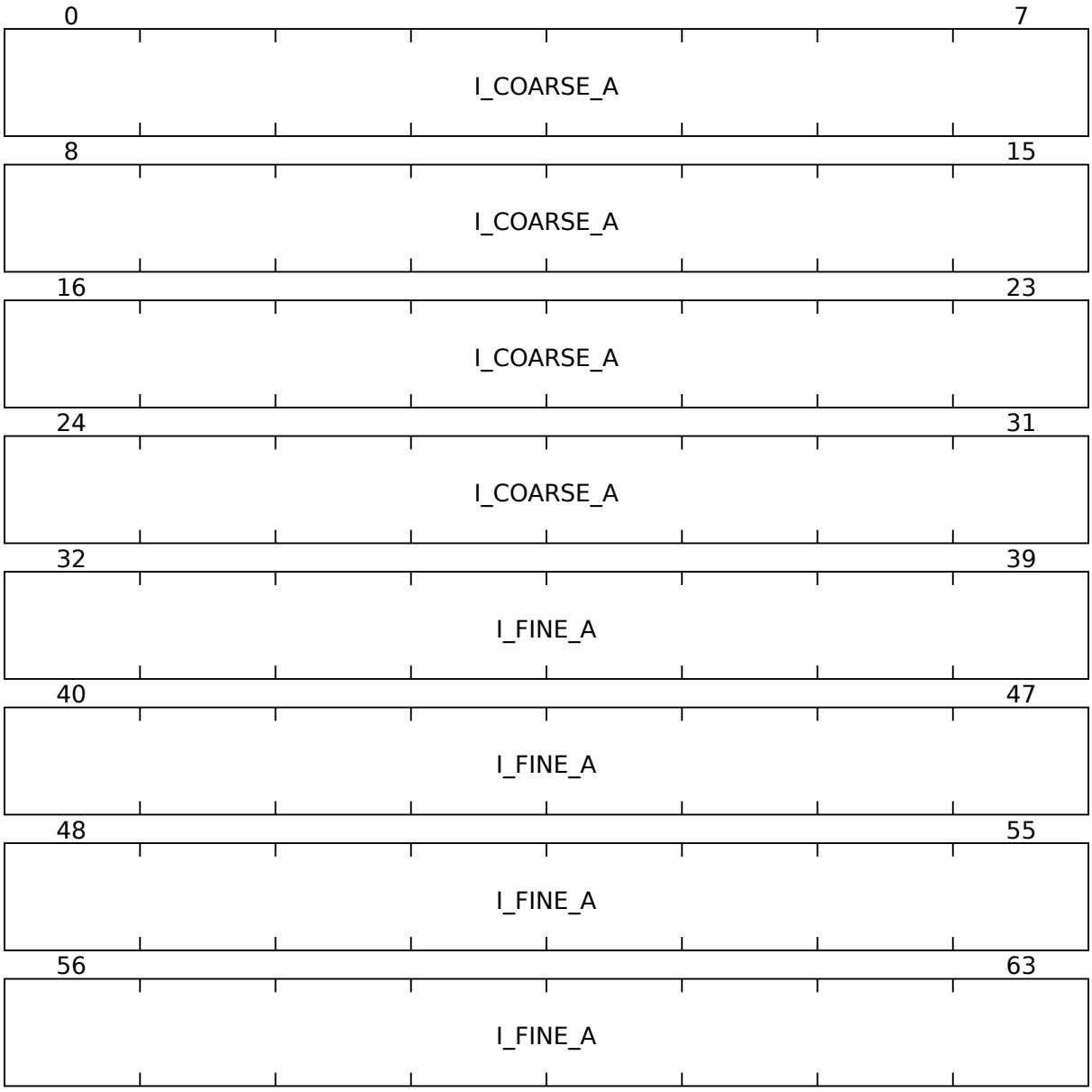


Table 20: Current Sense Phase A

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

1.3.1.17 0x323 (Voltage Sense)

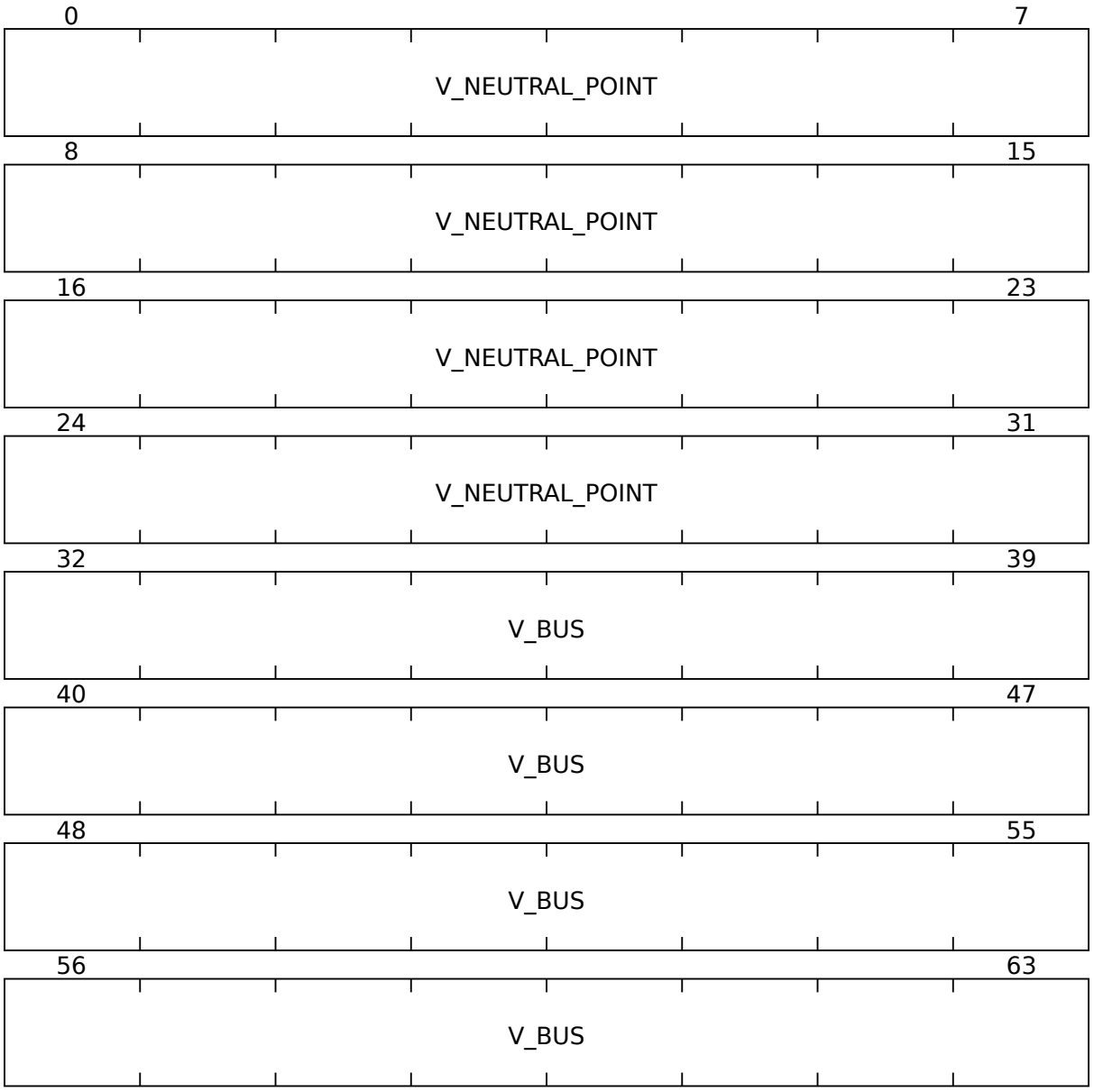


Table 21: Voltage Sense

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

1.3.1.18 0x324 (Current Sense ADC Debug)

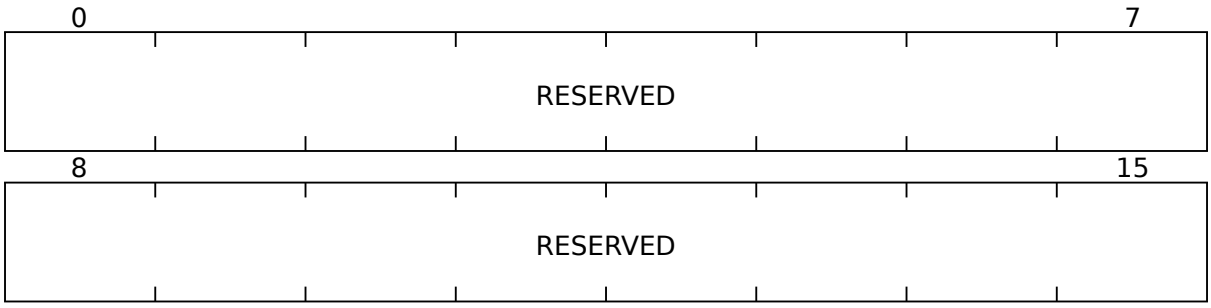


Table 22: Current Sense ADC Debug

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 23: 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.

1.3.2.2 0x019 (Command Packet)

Table 24: 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. i.e in torque control the desired torque.	0.1 Nm (torque control) or 1 degree per second (velocity control)	i32

Table 25: 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	1
0x02	Enable (Velocity Control). This instructs the motor controller to go into velocity control mode. It will transition to ActivelySwitching and attempt to drive a desired velocity.	DATA is desired velocity in degrees per second.	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 0x01a (EEPROM Set Parameter)

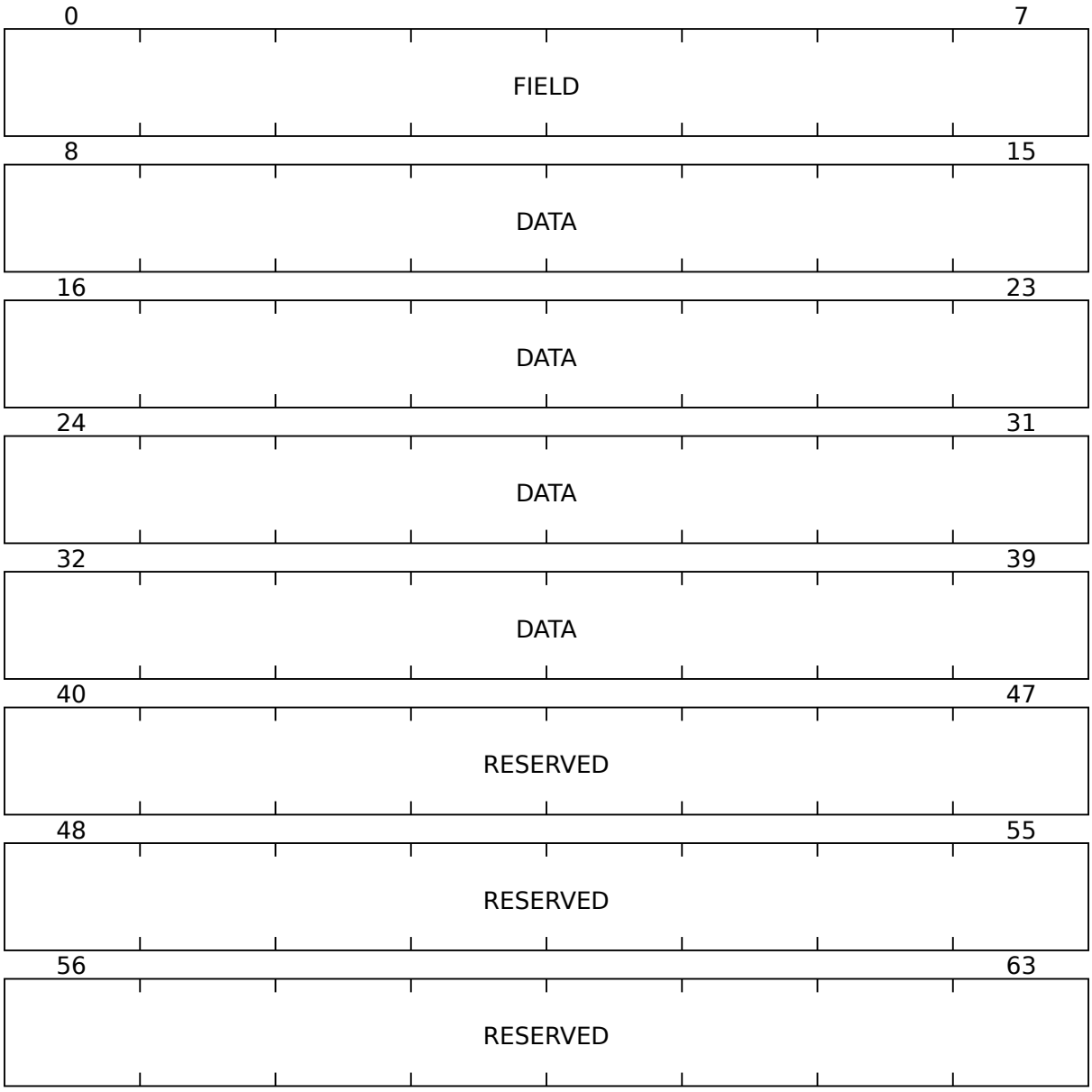


Table 26: EEPROM Set Parameter Packet

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

1.3.2.4 0x01b (EEPROM Get Parameter)

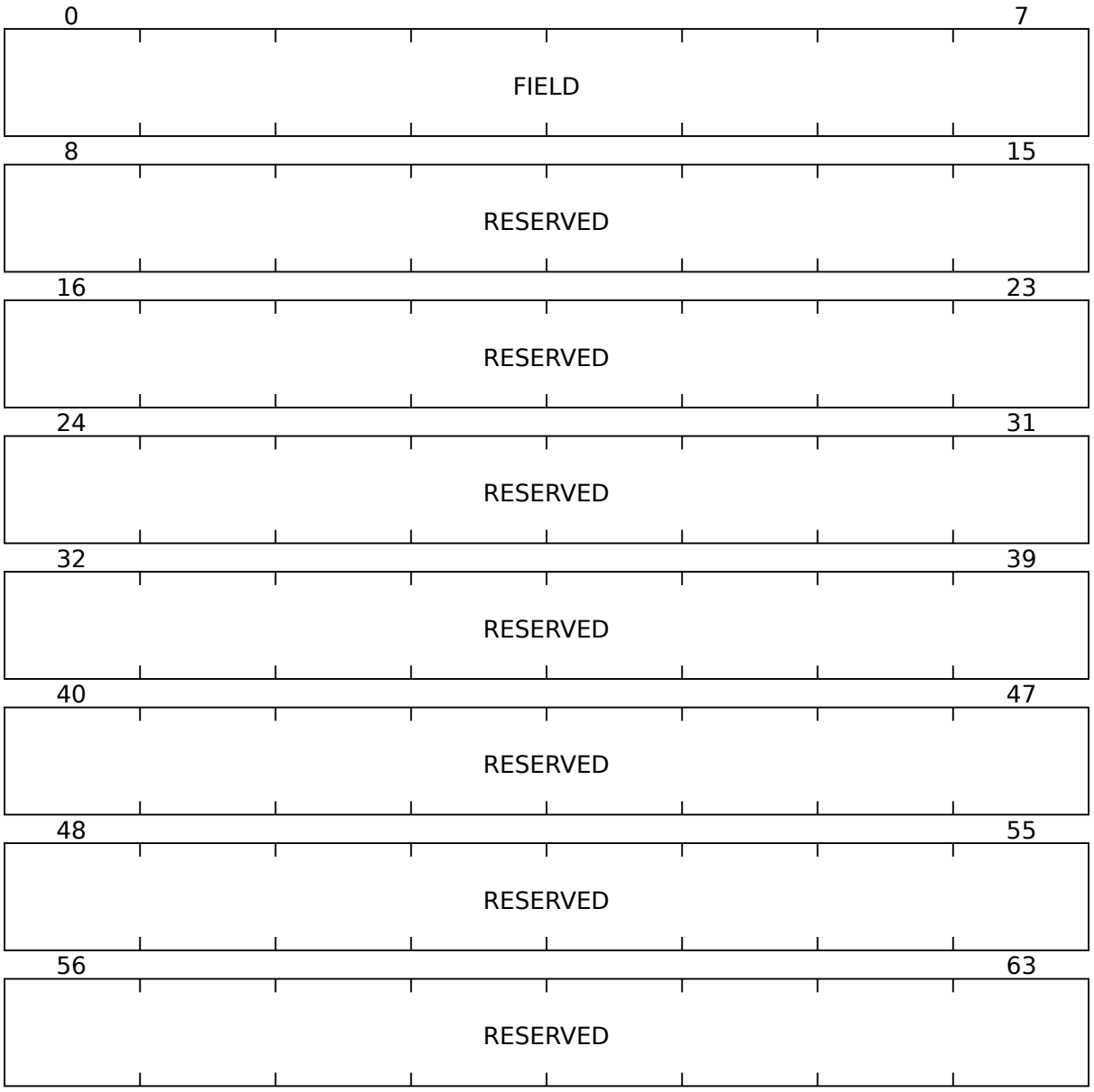


Table 27: EEPROM Set Parameter Packet

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.

Table 28: EEPROM Fields

Field ID	Field Name	Field Description	Datatype	Scaling (Divide by X to get real value)
0	NumberOf-Boots	Number of boots of the device total. (pending implementation)	f32	1
1	ResolverOffset-Degrees	Resolver offset from mechanical resolver angle to electrical angle in degrees.	f32	1
2	LoopIqPGain	P (proportional) gain in the Iq PID loop	f32	1
3	LoopIqIGain	I (integral) gain in the Iq PID loop	f32	1
4	LoopIqDGain	D (derivative) gain in the Iq PID loop	f32	1
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
7	LoopIdIGain	I (integral) gain in the Id PID loop	f32	1
8	LoopIdDGain	D (derivative) gain in the Id PID loop	f32	1
9	LoopIdTGain	Gain for integral anti windup in the Id PID loop	f32	1
10	TorqueConstant	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
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
24	LoopVelIGain	I (integral) gain in the velocity control loop.	f32	1
25	LoopVelDGain	D (derivative) gain in the velocity control loop.	f32	1
26	LoopVelTGain	Gain for integral anti windup in the velocity control loop.	f32	1
27	Max-TorqueSlewRate	Maximum torque slew rate in Newton-Metres per s.	f32	1
28	Overcurrent-ThresholdAmps	Maximum allowable phase current magnitude (in Apk)	f32	1
29	OvertempThresh-oldCelsius	Maximum allowable temperature in the inverter	f32	1
30	ResolverPoles-PerMotorPole	Number of resolver poles per motor pole. Sign indicates resolver inversion.	f32	1
31	MaxNPCapaci-torVoltage	Maximum allowable voltage across either neutral-point capacitor	f32	1
32	EnableHVI-LAlarm	Set to 1 to prevent operation if any HV connectors are unseated. Set to 0 to disable this check.	u32	1
33	ExtTempSen-sorType	This value controls what is reported in the *0x31C (External RTD interface)*. 0 - Report voltage measured across voltage divider by the internal ADC (in millivolts)1 - Report measured resistance (in Ohms)2 - Report temperature of an attached PT1000 RTD (in 100x Celsius)	u32	N/A
34	DeviceIdentifier	Lower 7 bytes of the 20-byte device identifier	7 bytes	N/A

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
          - PhaseCCoarseOffsetVolts
          - PhaseACoarseVoltsPerAmp
          - PhaseAFineHallOffsetVolts
          - PhaseBFineHallOffsetVolts
          - PhaseCFineOffsetVolts
          - PhaseAFineVoltsPerAmp
          - PhaseBCoarseVoltsPerAmp
          - PhaseCCoarseVoltsPerAmp
```

(continues on next page)

(continued from previous page)

```
          - 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

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: 178.5
LoopIqPGain: 0.03
LoopIqIGain: 6.0225
LoopIqDGain: 0
LoopIqTGain: 0
LoopIdPGain: 0.03
LoopIdIGain: 6.0225
LoopIdDGain: 0
LoopIdTGain: 0
TorqueConst: 1
PhaseACoarseHallOffsetVolts: 0
PhaseBCoarseHallOffsetVolts: 0
PhaseCCoarseOffsetVolts: 0
PhaseACoarseVoltsPerAmp: 0.0011173621
PhaseAFineHallOffsetVolts: 0
PhaseBFineHallOffsetVolts: 0
PhaseCFineOffsetVolts: 0
PhaseAFineVoltsPerAmp: 0.004703646
PhaseBCoarseVoltsPerAmp: 0.0013163976
PhaseCCoarseVoltsPerAmp: 0.00114652
PhaseBFineVoltsPerAmp: 0.0053415336
PhaseCFineVoltsPerAmp: 0.0058397744
LoopVelPGain: 0.006
LoopVelIGain: 0.001
LoopVelDGain: 0
LoopVelTGain: 0
MaxTorqueSlewRate: 65
OvercurrentThresholdAmps: 680
OvertempThresholdCelsius: 88
ResolverPolesPerMotorPole: 2
MaxNpCapacitorVoltage: 450
EnableHvilAlarm: 0
ExtTempSensorType: 2
DeviceIdentifier: a7hvgw1n2ah94bmsmj8f1dfg70
HardwareRevision: 1
```

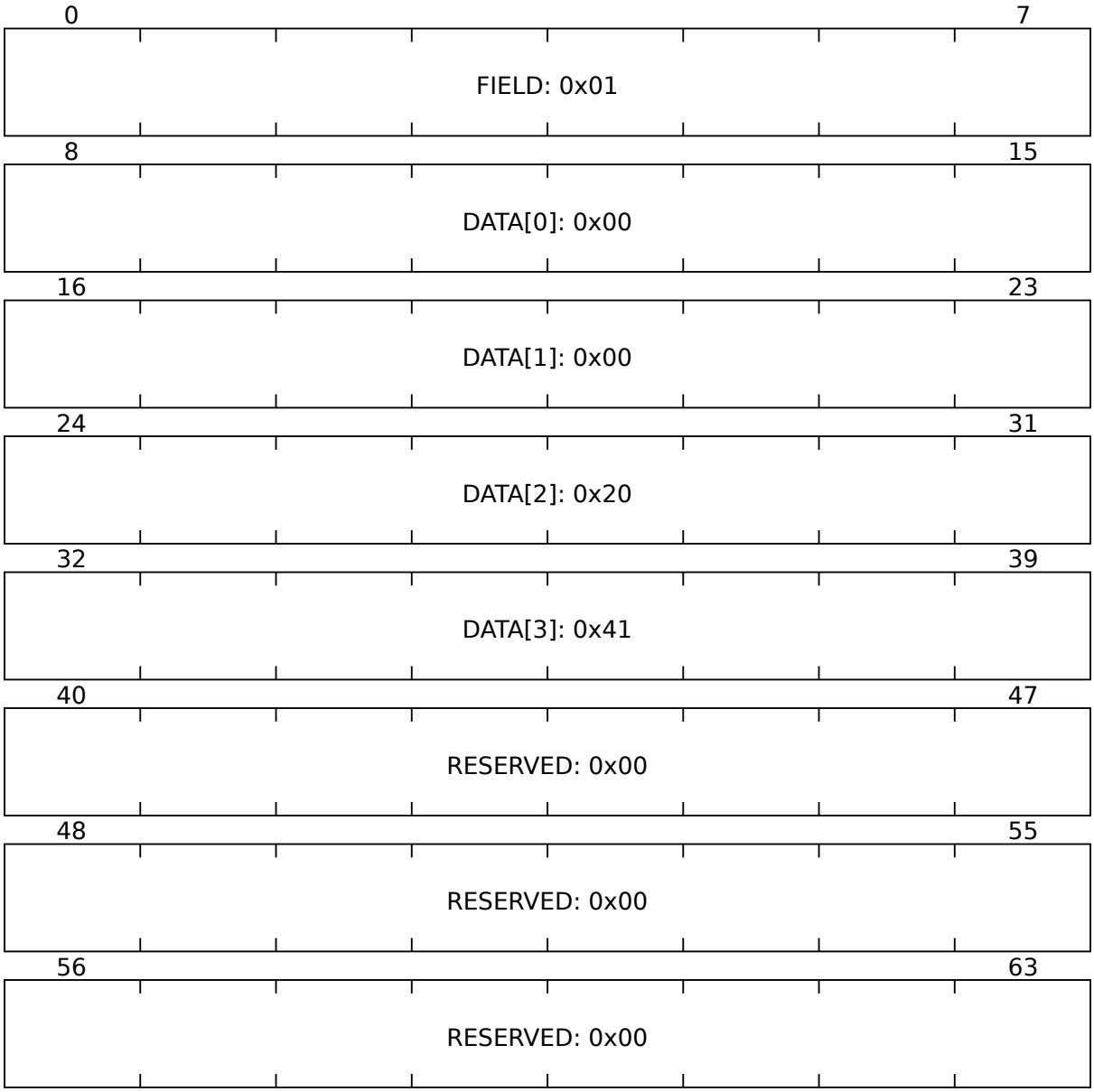
1.4.2.2 Over CAN

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:

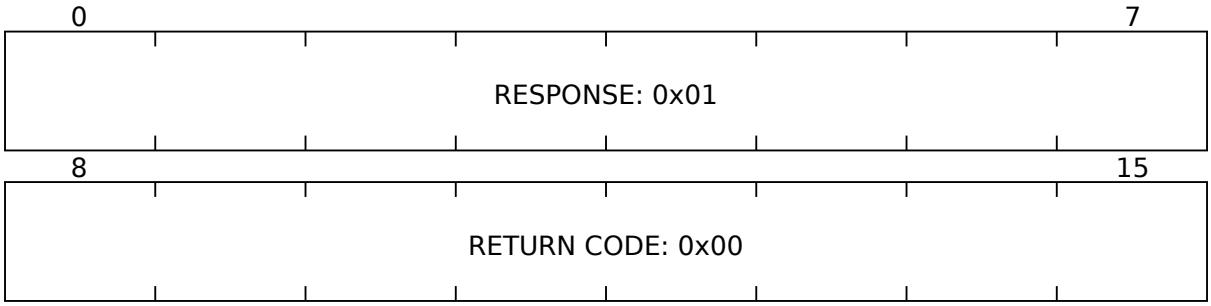


Step 2. Transmit CAN Packet

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.

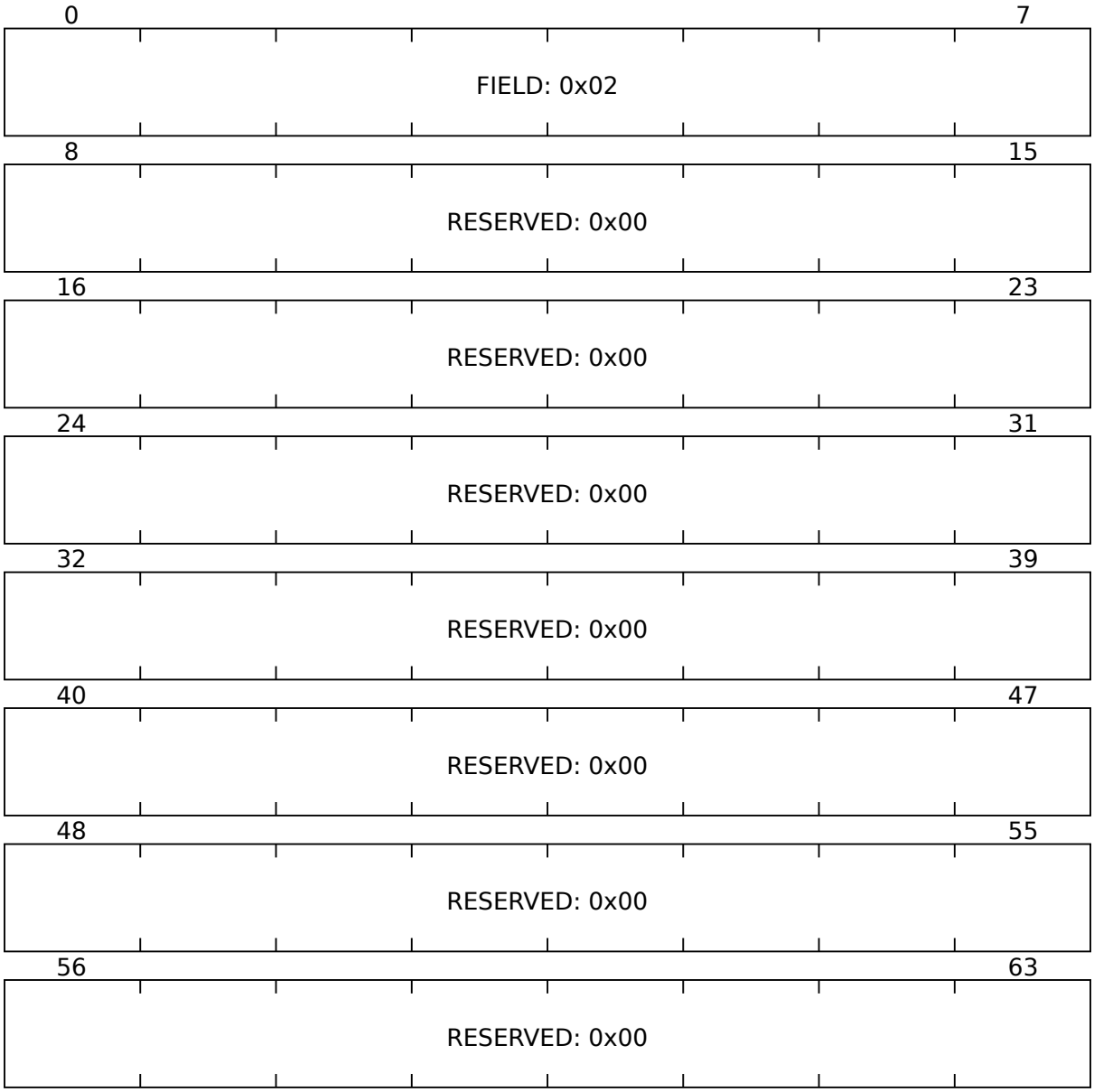


1.4.2.2.2 Reading.

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.

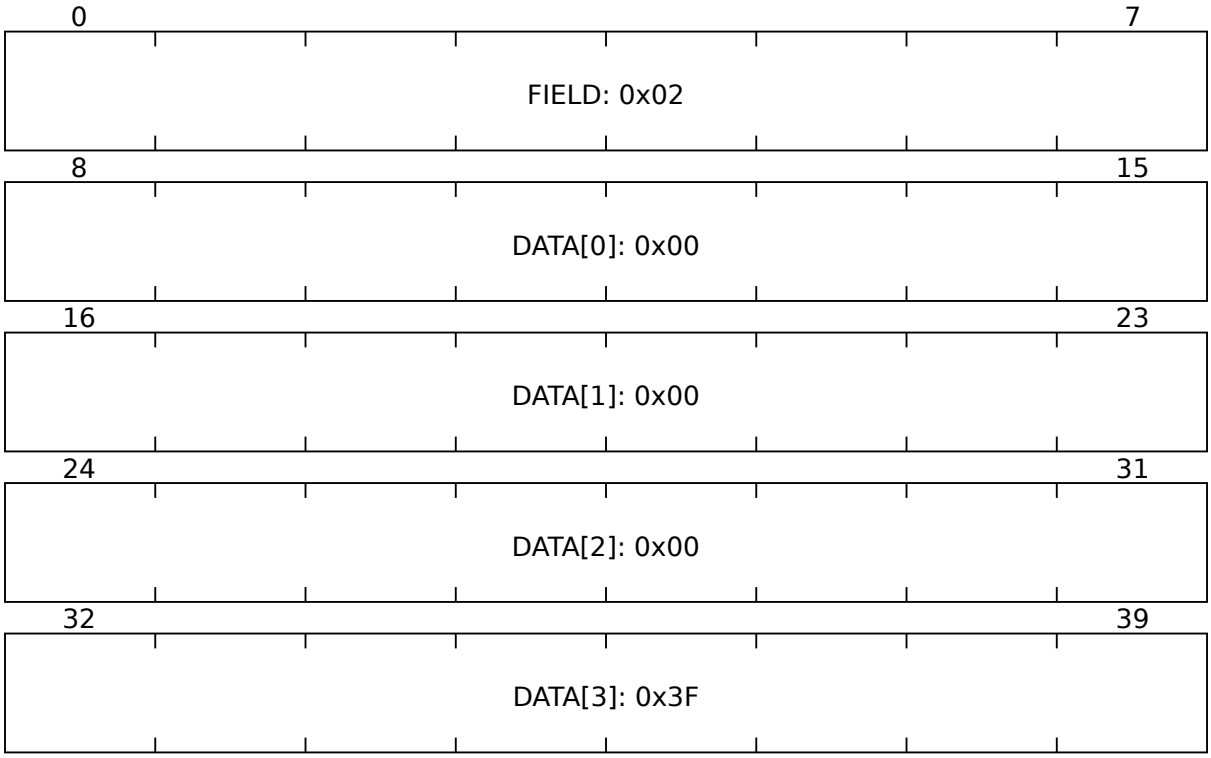


Step 2. Transmit CAN Packet

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

CHAPTER

TWO

MC0-500 HARDWARE DOCUMENTATION

2.1 Getting Started - Hardware

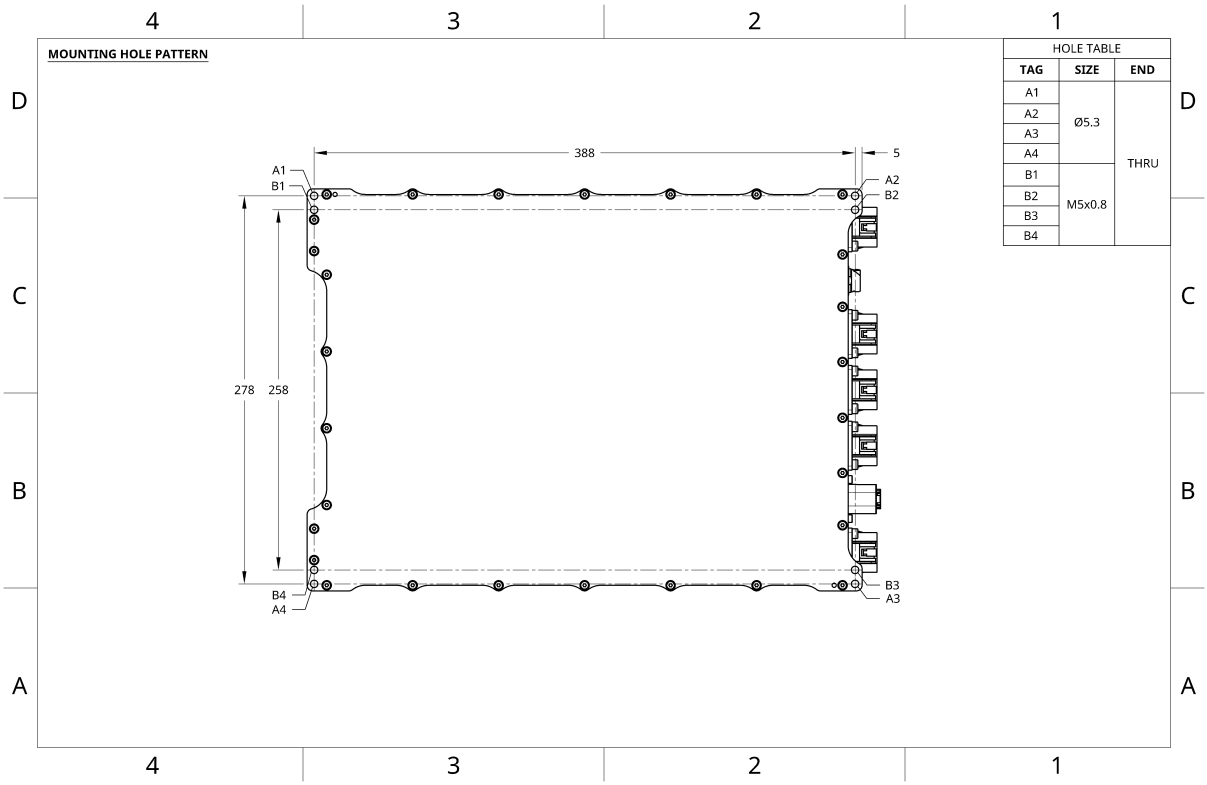
Welcome to the **Steinmetz HW - MC0-500** documentation.

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

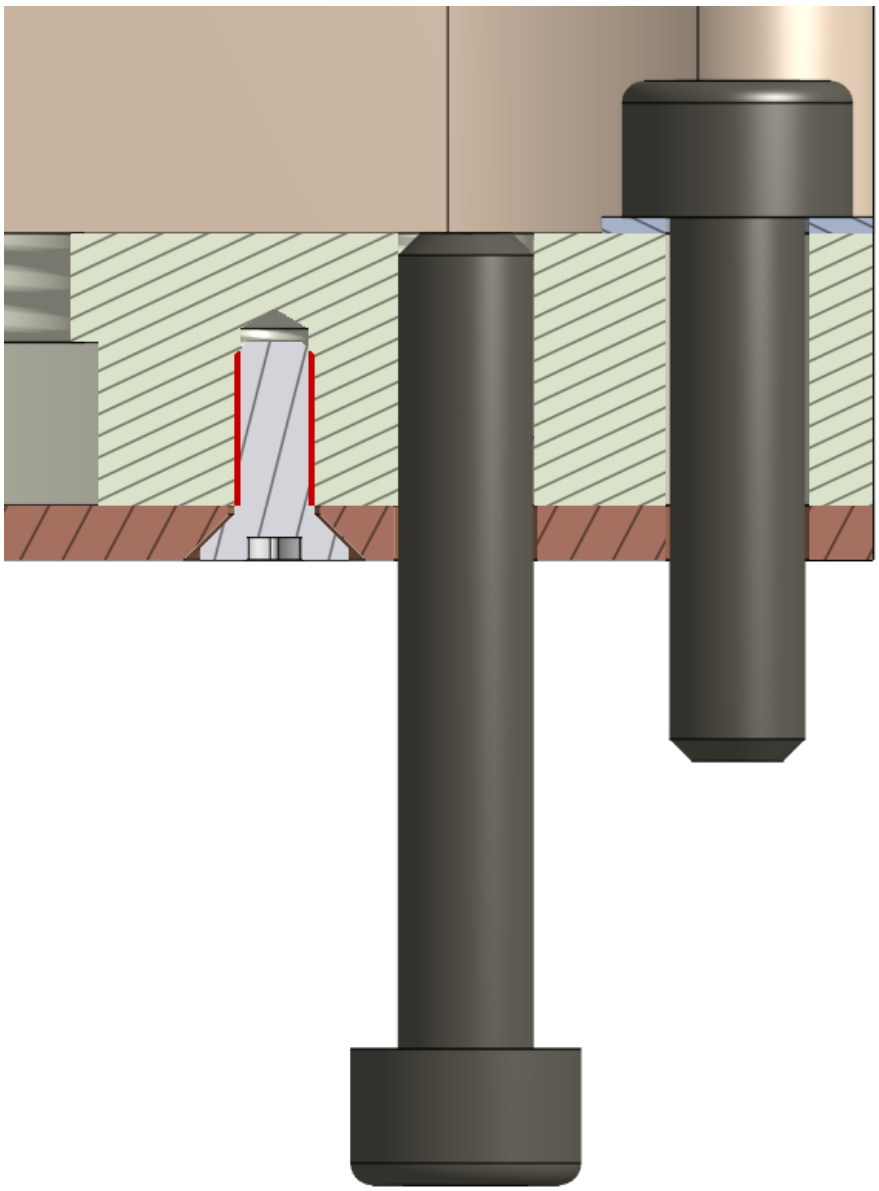
2.1.1 Mounting

MC0-500 should be securely fastened to a rigid 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 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 5 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)
- +12 Vin (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 20 L/min of flow rate, and inlet temperatures should be limited to 60 °C for maximum performance.

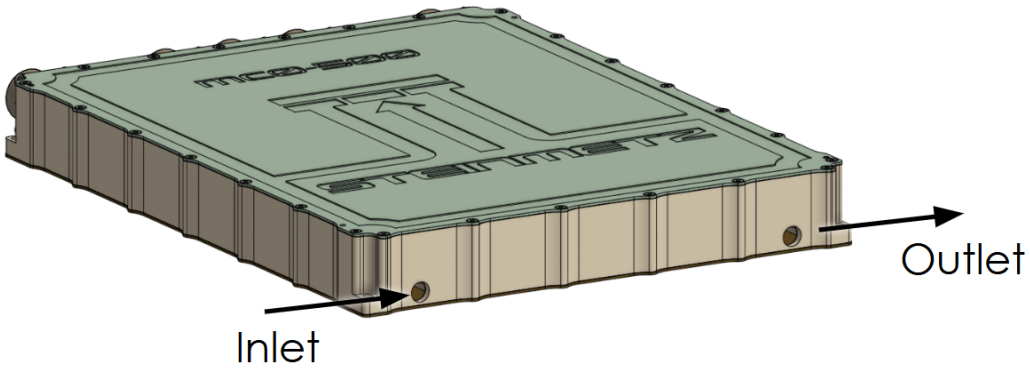
2.2 Cooling

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 right port is used as the inlet, and left port as the outlet.



It is recommended to use a semi-rigid tube with:

$$\text{OD} = 8 \text{ mm}, \text{ID} = 6 \text{ mm}$$

As-provided, MC0-500 ships with two straight barbed tube fittings. However these can be replaced with any 1/4”-18 NPT threaded fitting.

The tubing should be rated for at least the maximum coolant pressure.

2.2.2 Coolant

The coolant plate is rated for a maximum inlet pressure of **310kPa (45 psi)** relative to ambient. Pressures in excess of the maximum may cause leakage of coolant and degraded performance.

Internal testing was performed with a coolant consisting of a 50/50 mix of water and glycol. The maximum recommended inlet temperature is 60C at full load operation.

2.2.3 Flow Requirements

The recommended flow rate to MC0-500 is 20 LPM.

2.3 Connectors

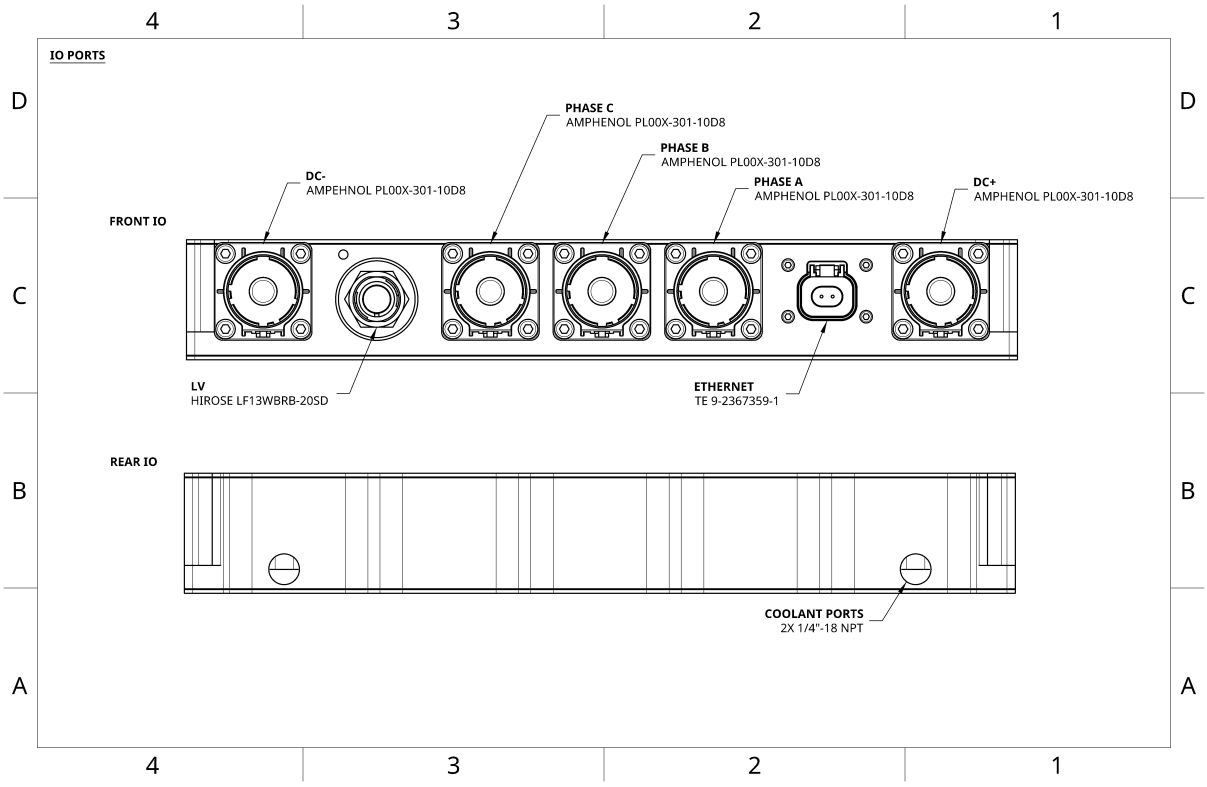
The front face of MC0-500 has all of the device’s connectors. When facing the front of the device, the connectors are, from left to right:

- 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 a IP67 seal or better when mated.

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:

2.3.2 LV Connector Pinout

Pin	Signal	Description
1	+LV_IN	Main LV supply input
2	CAN_L	CAN Low
3	CAN_H	CAN High
4	EXT_GPIO_0	External GPIO
5	EXT_GPIO_1	External GPIO
6	GND	Ground
7	GND	Ground
8	GND	Ground
9	RESOLVER_EXC_P	Resolver Excitation +
10	EXT_RTD_P	External RTD Positive
11	EXT_GPIO_2	External GPIO
12	RESOLVER_SIN_N	Resolver SIN-
13	GND	Ground
14	RESOLVER_SIN_P	Resolver SIN+
15	RESOLVER_COS_P	Resolver COS+
16	EXT_RTD_N	External RTD Negative
17	RESOLVER_EXC_N	Resolver Excitation -
18	RESERVED	Reserved
19	RESERVED	Reserved
20	RESOLVER_COS_N	Resolver COS-

Note

On engineering sample units, only 12V input is supported. Production models have support for 24-48V+.

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

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 M4x0.7x10 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

> **Warning**
>
> Current delivered engineering samples of MC0-500 are not rated to IP67.

MC0-500 is designed for IP67 rating when all connectors are mated.

2.5 Mechanical

MC0-500 has dimensions of 43 mm x 298 mm x 381 mm (HxWxD), with a dry mass of 7.4 kg and a displacement volume of 5.25 L. 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. Each corner of the unit contains mounting holes that accommodate fasteners from both the top and bottom, using a combination of through and threaded holes.

At the rear of the unit are the cooling ports, which are preinstalled with 90° push-connect fittings. These can be swapped for fittings of the customer's choosing; refer to *Cooling* for additional information.

`Download PDF`

CHAPTER

THREE

MC0-250 HARDWARE DOCUMENTATION

3.1 MC0-250 Coming Soon