ModusToolbox™ Motor Suite Motor Control Library tuning guide

About this document

Scope and purpose

ModusToolbox™ Motor Suite Motor Control Library (motor-ctrl-lib) software to drive 3-phase permanent magnet synchronous motors (PMSM) or brush-less DC (BLDC) motors. This document describes how to measure/identify the motor and load parameter required for the Motor Control Library and how to configure/ tune the control loop parameters.

Intended audience.

This document is intended for the ModusToolbox™ Motor Suite Motor Control Library users who wants to develop motor control driver applications.

Software version

- ModusToolbox™ software 3.3 or above
- Motor Control Library V3.0.0 or above
- ModusToolbox™ Motor Suite 2.6.1 or above

Table of contents

Table of contents

Table of contents

Table of contents

1 Introduction

Motor control software library provides an advanced sensorless or sensor-based field-oriented control (FOC) algorithm to drive 3-phase permanent magnet synchronous motor (PMSM) loads including constant air-gap surface mounted permanent magnet (SM) motor and interior permanent magnet (IPM) motor. Also, this library supports a hall sensor-based block/trapezoidal commutation algorithm to drive Brush-less DC motor (BLDC). This library supports the following control methods:

- Rotor frame-oriented field-oriented control (RFO): Sensor-less and sensor (hall and encoder) based rotor position estimation.
- Stator frame-oriented field-oriented control (SFO): Sensor-less rotor-based rotor position estimation
- Trapezoidal /block commutation (TBC): Hall sensor (3-digital hall) based rotor position estimation.

This document describes how to measure/identify the motor parameters that are required for the Motor Control Library and how to configure/ tune the control loop parameters. Motor control engineers need to adapt the default library configuration to match their specific motor, inverter hardware, and application requirements.

1.1 Key features

A list of key features supported in the Motor Control Library are given here:

- Adaptive sensorless observer that provides minimal phase distortion. It can observe rotor angle, speed, stator angle, stator flux magnitude, and load angle.
- Hall sensor (3-Digital Hall) or incremental encoder-based field-oriented control.
- Supports leg shunt and single shunt-based motor phase current measurement.
- 3-Phaseand 2-Phase PWM modulation schemes. 2ph SVPWM that allows reduction of the switching losses compared with three-phase SVPWM (symmetrical placement of zero vectors).
- Supports flux weakening, over modulation and maximum torque per amp for IPM motors.
- Start-up methods for sensor-less field-oriented control
- Pre alignment rotor into knows position.
- Constant V/F control
- I/F control
- Inductance based initial rotor angle estimation.
- Dyno mode to catch the free running motor.
- High Frequency Injection for rotor angle estimation (only for IPM)
- Motor Control Library provides the following protection/Fault handling.
- Under/Over voltage protection
- Overcurrent protection
- Over speed and overtemperature protection
- Motor I2T protection
- Control mode supported
- Speed control for RFO, SFO, and TBC control methods.
- Current control for RFO and TBC control methods
- Torque control for SFO control method.

Introduction

- Voltage control
- Profiler to automatically identify motor and load parameters that are used in speed, current controller, and rotor angle estimation.
- Algorithm is implemented in Floating Point Base (compliant with the ANSI/IEEE Std 754-2008) - No extra bit-shift or overflow & underflow checks.

The Motor Control Library supports many permutations of control type, controlled entity, feedback type, and startup method as listed in the following table.

Table 1 Motor Control Library support control type

Control type	Controlled entity	Position feedback	Startup method
Open loop	Voltage	-NA-	-NA-
FOC in RFO	Current	Sensorless	Rotor Pre-Alignment
FOC in RFO	Current	Sensorless	Six Pulse Injection
FOC in RFO	Current	Sensorless	High Frequency Injection
FOC in RFO	Current	Sensorless	Dyno Mode
FOC in RFO	Current	Encoder	Rotor Pre-Alignment
FOC in RFO	Current	Hall Sensor	-NA-
BC in TBC	Current	Hall Sensor	-NA-
TC in TBC	Current	Hall Sensor	-NA-
FOC in SFO	Torque	Sensorless	Rotor Pre-Alignment
FOC in SFO	Torque	Sensorless	Six Pulse Injection
FOC in SFO	Torque	Sensorless	High Frequency Injection
FOC in SFO	Torque	Sensorless	Dyno Mode
FOC in RFO or SFO	Speed	Sensorless	Rotor Pre-Alignment
FOC in RFO or SFO	Speed	Sensorless	Six Pulse Injection
FOC in RFO or SFO	Speed	Sensorless	High Frequency Injection
FOC in RFO or SFO	Speed	Sensorless	Open-loop voltage
FOC in RFO	Speed	Sensorless	Open-loop current
FOC in RFO	Speed	Encoder	Rotor Pre-Alignment
FOC in RFO	Speed	Hall Sensor	-NA-
FOC in RFO	Position	Encoder	Rotor Pre-Alignment
BC in TBC	Speed	Hall Sensor	-NA-
TC in TBC	Speed	Hall Sensor	-NA-

Control mode is configured using params[x].ctrl.mode parameter.

1.2 Rotor field-oriented control

RFO is a variant of FOC where the motor's three-phase sinusoidal currents are decomposed into q-axis and d-axis DC currents using Clark and Park transformations. These transformations reduce the complexity of the control system for AC machines. Figure 1 illustrates the overall block diagram of the RFO control method which is composed of modules such as a speed control loop, q- and d-axis current control loops, and position feedback.

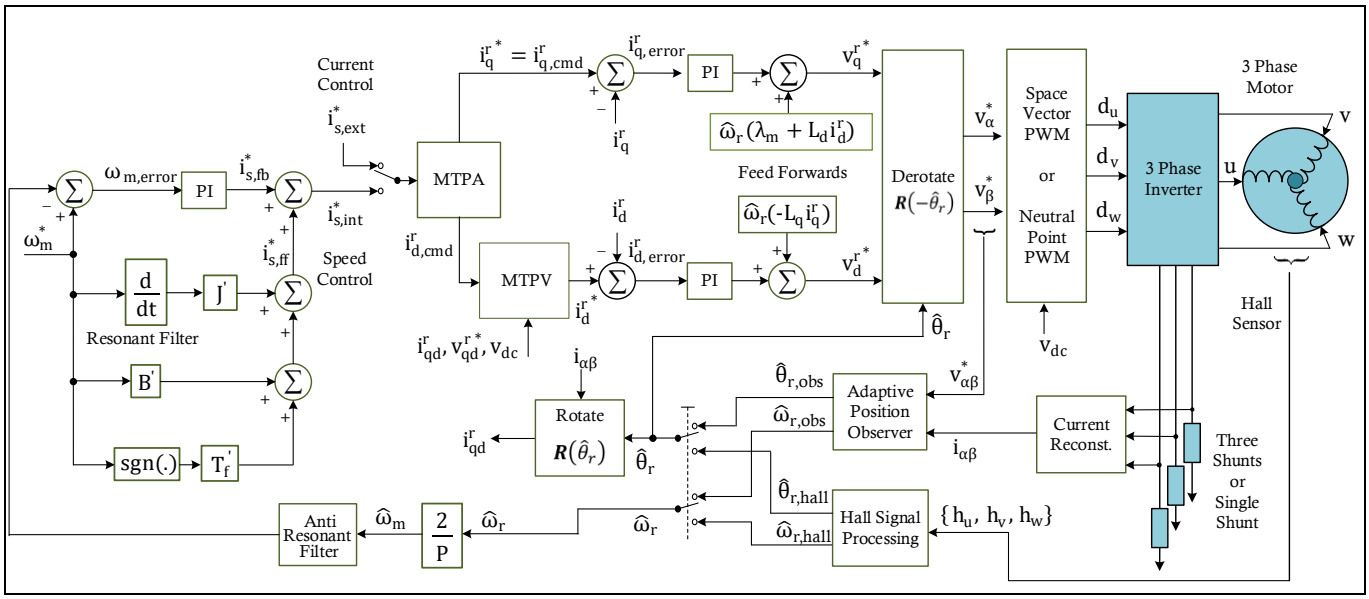


Figure 1 RFO block diagram

Position and speed information can be obtained using position sensors (such as encoders or hall sensors) or through sensorless approaches. The speed controller utilizes this speed information to generate current references via a PI controller.

The Maximum Torque Per Ampere (MTPA) block generates optimal d-axis and q-axis currents to maximize torque output. This block is specifically used for Interior Permanent Magnet motors to effectively utilize reluctance torque. For operations above base speed, the Maximum Torque Per Volt (MTPV) algorithm can be employed to achieve higher operating speeds.

Both MTPA and MTPV algorithms create reference values for q-axis and d-axis current control. The current controllers process these references to produce voltage commands, which are then applied to the inverter through selectable modulation schemes.

1.3 Motor control tuning flow

Motor control tuning involves the following steps.

- Step #1: Parameter Identification and Configuration (Refer: Parameter configuration)
- Configure basic motor parameters from nameplate/datasheet.
- Set up hardware-specific parameters (power board, controller)
- Configure system (such as PWM) and control-related parameters (such as Bandwidth)
- Step #2: Verification of Voltage / Current Measurement (Refer: Verification of ADC measurement)
- Verify that system configuration matches hardware setup.
- Validate protection thresholds (overcurrent, overvoltage)
- Step #3: Motor and Load Parameter Identification (Optional) (Refer: Motor parameter identification)
- Configured I/F and Speed loop parameters.
- Run profiler to identification motor parameters.
- Update configuration with identified values.
- Step #4: Startup Parameter Adjustment (Refer: Startup method tuning)
- Configure V/F or I/F startup parameters.
- Adjust voltage offset and slope ratios for V/F startup.
- Set appropriate startup current command for I/F startup.
- Optimize acceleration ramps for smooth startup.
- Step #5: Control Loop Tuning - Current Control Loop (Refer: Current)
- Set current loop bandwidth and calculate initial gains.
- Step #6: Control Loop Tuning - Speed Control Loop (Refer: Speed)
- Configure speed controller gains and bandwidth.
- Optimize for application-specific performance requirements.

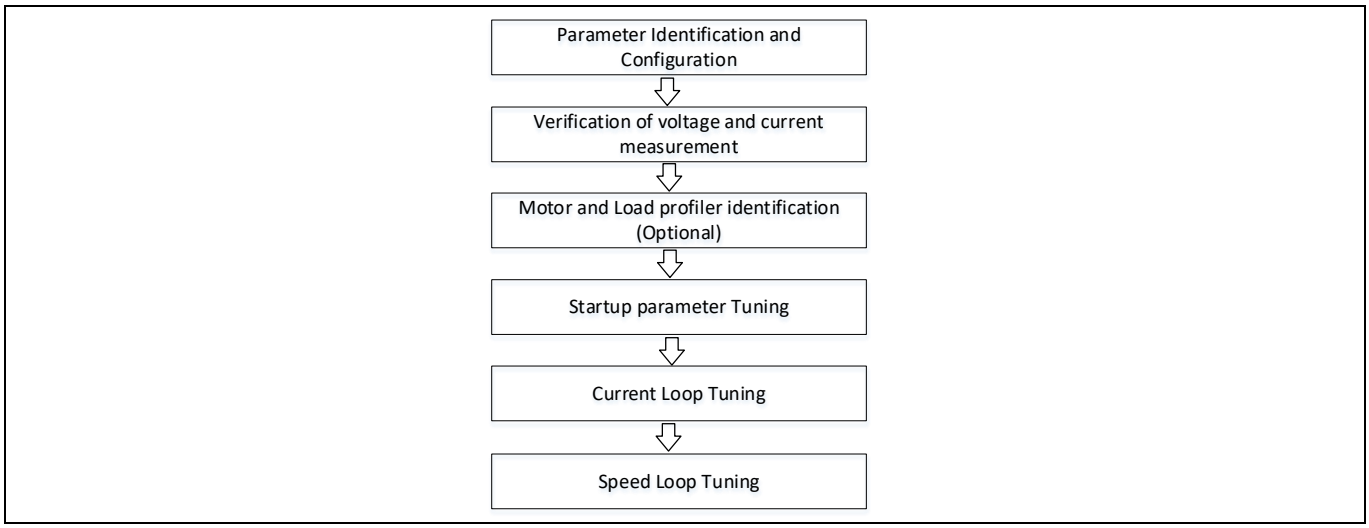


Figure 2 Motor control tuning flow

2 Motor parameters required for Motor Suite Motor Control Library tuning

This chapter describes the required motor and load parameters for the Motor Control Library.

2.1 Motor/load parameters

Table 2 shows the motor parameters required for Motor Control Library.

Table 2 Motor/load parameters required for Motor Control Library

Name	Unit	Parameter Name	Descriptions	Where it is used
Motor poles	#	params[x].motor.P	Number of magnetic poles, not pole pairs.	▪ Convert electrical speed & angle to mechanical speed & angle or conversely. ▪ Speed control K_p, K_i and Feed forward parameter calculation. ▪ Hall and Incremental Encoder interface
Motor L_q	H	params[x].motor.lq	Stator q-axis (Torque current) inductance of each phase winding.	▪ Current control K_p and Feed forward calculation ▪ Rotor angle estimation-Sensorless FOC ▪ MTPA for IPM motor ▪ Torque Calculation
Motor L_d	H	params[x].motor.ld	Stator d-axis (Flux current) inductance of each phase winding.	▪ Current control K_p and Feed forward calculation ▪ Rotor angle estimation-Sensorless FOC ▪ MTPA for IPM motor ▪ Torque Calculation
Motor λ_m	wb	params[x].motor.lam	Rotor flux linkage, this can be calculated from B_{emf} Constant (K_e [$V_{peak(Line\ to\ Line)}$ / kRPM]) $\lambda m = \frac{K_e}{\sqrt{3}} * \frac{2 * 60}{1000 * P * 2\pi}$ Where P – Number of Motor Poles	▪ MTPA for IPM motor ▪ Torque Calculation
Motor Resistance	Ω	params[x].motor.r	Motor phase resistance	▪ Current control K_i calculation ▪ Rotor angle estimation-Sensorless FOC

Motor parameters required for Motor Suite Motor Control Library tuning

Name	Unit	Parameter Name	Descriptions	Where it is used
Motor Maximum Torque	Nm	params[x].motor.T_max	Maximum torque of the motor	▪ Torque Control [SFO]
Motor Peak Current	A	params[x].motor.i_peak	Maximum allowed motor current. Set this value to 2.5 to 3x of motor continuous current, if not specified in the motor datasheet.	▪ I2T protection ▪ Speed control PI output limit
Motor Continuous Current	A	params[x].motor.i_cont	Continuous motor current in rms value	▪ I2T protection
Motor d-axis current maximum	A	params[x].motor.id_max	Allowed maximum d-axis current rating of the motor that does not result in demagnetization of the permanent magnet. Set this value to 25% of motor continuous current, if not specified in the motor datasheet.	▪ Field weakening
Motor Voltage	V	params[x].motor.v_nom	This parameter specifies the maximum terminal voltage.	▪ Not used
Nominal Speed	RPM	params[x].motor.w_nom.elec	Nominal speed of the motor.	▪ Hall sensor for zero speed detection
Maximum Speed	RPM	params[x].motor.w_max.elec	Maximum speed of the motor.	▪ Limit for sensorless rotor angle estimator ▪ Over speed threshold
Inertia	$kg.m^2$	params[x].motor.mech.inertia	Inertia of the motor and load system	▪ Speed control K_p and Feed forward parameter calculation
viscous	$kg.m^2/sec^2$	params[x].motor.mech.viscous	Viscous damping of the motor and load system	▪ Speed control K_i and Feed forward parameter calculation
friction	$kg.m^2/sec^2$	params[x].motor.mech.friction	Friction of the motor and load system	▪ Speed control Feed forward parameter calculation

Note: Each parameter or variable contains [x] notation in this document, where the x value represents motor instances. To access Motor 0 parameters, the x value should be 0, and for Motor 1, the value should be 1.

2.2 Parameter configuration hints

- System Parameters
  - params[x].sys.cmd.w_max.mech is the maximum speed the user wants to command via potentiometer. This can be different from maximum speed of the motor.
  - params[x].sys.vdc_nom is the DC bus voltage., this value is the same as params[x].motor.v_nom. However, if the user may want to apply 24 V to a 48 V motor. So, vdc_nom needs to be set as 24 V rather than 48 V in this case.
- Control Parameters
  - Set the value of params[x].ctrl.curr.bw at least 10 times lower than switching frequency. In addition, it must be set higher than params[x].ctrl.speed.bw

3 Parameter configuration

Motor Suite Motor Control Library parameters are centrally defined and configured in the *ParamConfig.h* file. This header file serves as the primary configuration interface for all parameters required by the Motor Control Library. The ParamConfig.h file is in the project directory structure at: *\configuration\motor-ctrl-lib-config\ParamConfig.h.*

Values configured using macros in *ParamConfig.h* are assigned into actual parameters in the *ParamConfig.c* file through the PARAMS_InitManual() function, which is called during state machine initialization to transfer all macro-defined configuration values from the header file into the runtime parameter structure. Refer 12.1 Parameter and 12.3 Initialization and interrupt handling in motor control for parameter handling and initialization sequence.

Alternatively, ModusToolbox™ Motor Suite GUI provides a comprehensive Graphical User Interface (GUI) for configuring Motor Control Library parameters without manually editing code files.

3.1 How to configure motor and load parameters

Configuration of motor and load parameters in the motor control code example.

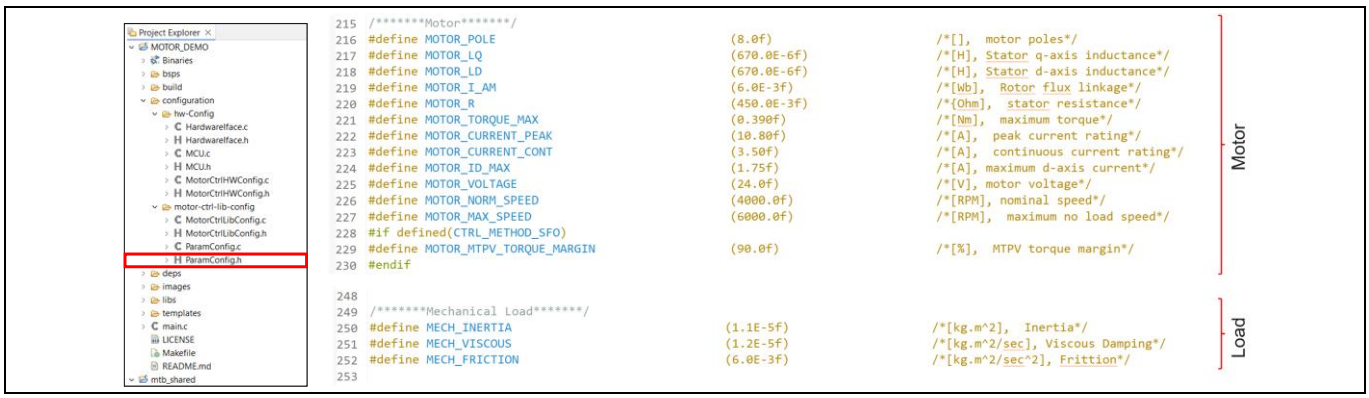


Figure 3 Motor and load parameter configuration in ModusToolbox™ code example

Attention: When updating the motor and load parameter make sure the following macro in this file is set true "#define PARAMS_ALWAYS_OVERWRITE (true)"

Parameter configuration

Configuration of motor and load parameter from ModusToolbox™ Motor Suite GUI.

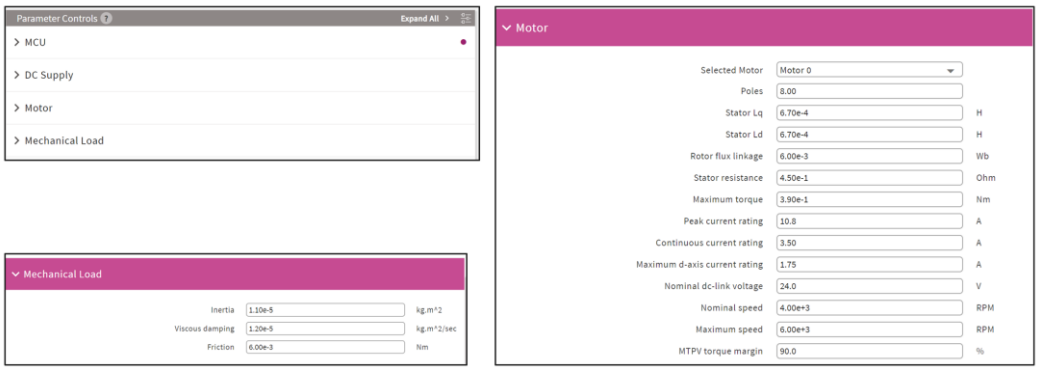


Figure 4 Motor and load parameter configuration in ModusToolbox™ Motor Suite GUI

After updating the parameter in the motor suite GUI, write the parameter into the target board from Motor Suite GUI .

3.2 How to configure voltage and current measurement parameter

Configure the following power board parameters using ModusToolbox™ Motor Control code example in *ParamConfig.h* file, *\configuration\motor-ctrl-lib-config\ParamConfig.h.*

- Current shunt and external amplifier gain for current measurement
- Voltage measurement resistive network circuit used to convert DC bus voltage range into controller measurable range

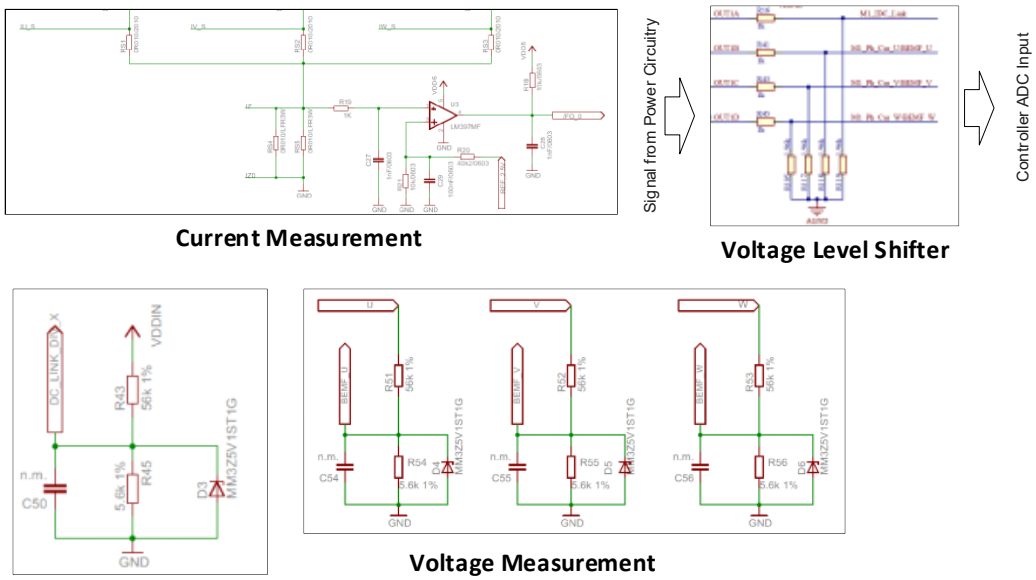


Figure 5 Current and voltage measurement circuit

Parameter configuration

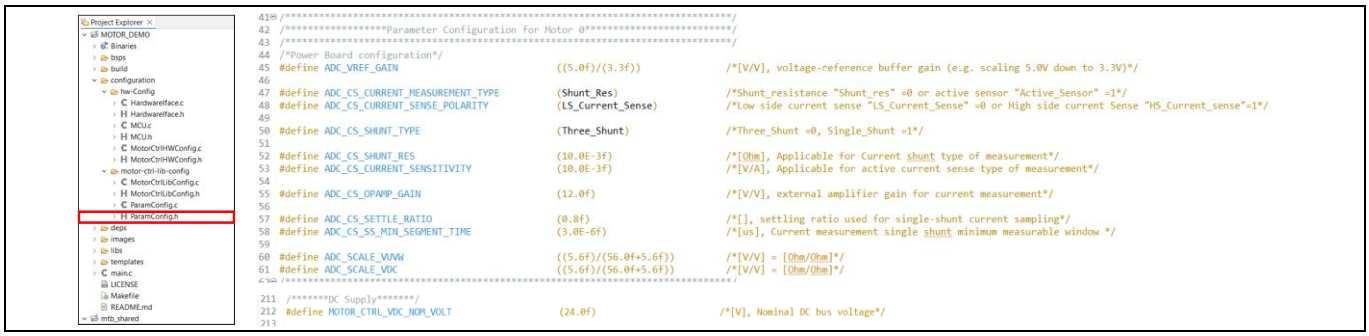


Figure 6 Power board-related configuration in ModusToolbox™ IDE

The "ADC_VREF_GAIN" macro defined in ParamConfig.h holds the configuration for the voltage level shifter network in the controller board. For example, in "KIT_PSC3M5_CC2", a resistor network (by default) is used to convert the 5V signal from the power board to 3.3V; in this case, “ADC_VREF_GAIN” = 5.0/3.3. If no voltage level shifter network is used to or is manually removed from the controller board, this macro value should be 1.

The “ADC_CS_OPAMP_GAIN” macro defined in ParamConfig.h holds the gain in the current input path for shunt or active current sensor based current measurement. This includes external gain(amplifier gain and any attenuation in the external circuit) and configured internal ADC sampler gain(default sampler gain value is 1).

ADC_CS_OPAMP_GAIN = External Gain * Internal Gain = 12*1 = 12 (default configured value)

The "ADC_CS_SHUNT_RES" macro holds the current measurement shunt resistance value in ohms for shunt-based current measurement systems, while the "ADC_CS_CURRENT_SENSITIVITY" macro is used for active current sensor-based current measurement and holds the current sensitivity of the active sensor defined in V/A units - for example, a 1mΩ shunt resistor would use ADC_CS_SHUNT_RES = 0.001f, whereas an active sensor with 100mV/A output would use ADC_CS_CURRENT_SENSITIVITY = 0.1f, and the appropriate macro should be configured based on whether the hardware design uses shunt resistors or active current sensors for motor current feedback measurement.

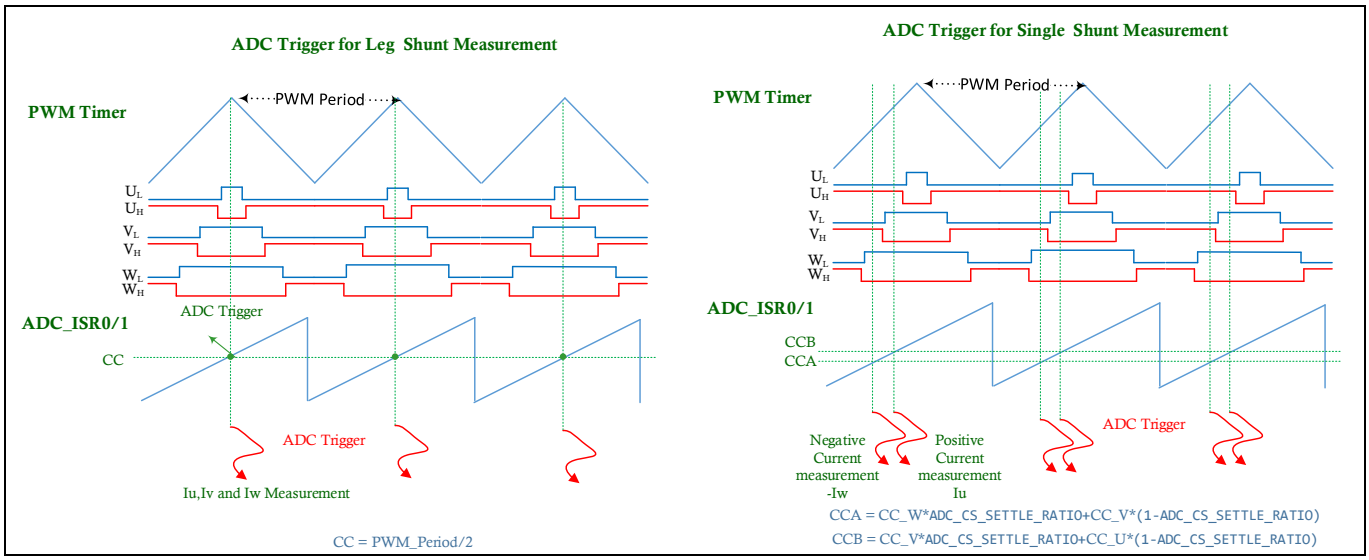


Figure 7 Current trigger – PWM for sector 0

Also, some of the current measurement-related parameters can be configured using the ModusToolbox™ Motor Suite GUI.

Parameter configuration

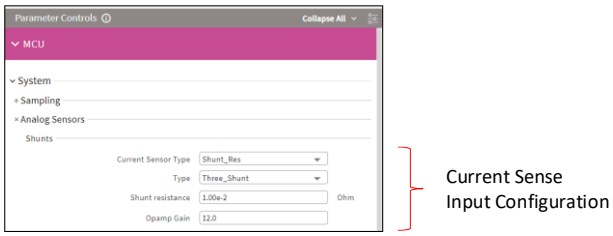


Figure 8 Current measurement configuration in Motor Suite GUI

In case of multiple motors, configure for other motors as well.

3.3 How to configure key system parameters

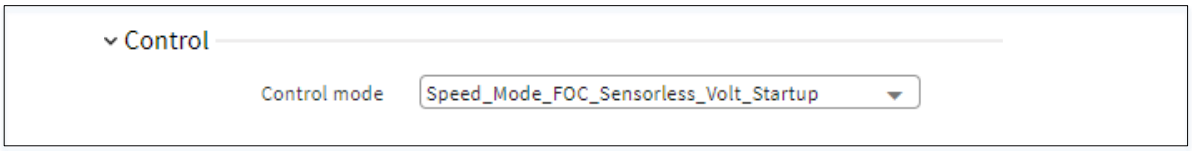
This chapter describes the key system parameters of the Motor Control Library and how to configure those parameters.

3.3.1 Control mode

The Motor Control Library supports many permutations of control type, controlled entity, feedback type, and startup methods that are all listed in Table 1. The control mode selection is managed through the params[x].ctrl.mode parameter, which serves as the primary configuration interface for determining the operational behavior of the motor control system. This parameter can be configured through the ModusToolbox™ IDE or Motor Suite GUI.

```
127 /*******Control*******/
128 #if defined(CTRL_METHOD_RFO) || defined(CTRL_METHOD_SFO)
129 #define MOTOR_CTRL_CTRL_MODE                    (Speed_Mode_FOC_Sensorless_Volt_Startup)/*Control mode*/
130 #elif defined(CTRL_METHOD_TBC)
131 #define MOTOR_CTRL_CTRL_MODE                    (Speed_Mode_Block_Comm_Hall)  /*Control mode*/
132 #endif
```

Configure from Modus Toolbox IDE , File name : ParamConfig.h



Configure from Modus Toolbox Motor Suite GUI

Figure 9 Control mode configuration

Parameter configuration

Table 3 Supported control mode

Control mode	Build			Description
	RFO	SFO	TBC	
Volt_Mode_Open_Loop	0	0	0	Open-loop V/Hz control
Current_Mode_Open_Loop	1			Open-loop I-F control
Curr_Mode_FOC_Sensorless_Align_Startup	2			Closed-loop sensorless-FOC current control with pre-alignment at startup
Curr_Mode_FOC_Sensorless_SixPulse_Startup	3			Closed-loop sensorless- FOC current control with six pulse injection at startup
Curr_Mode_FOC_Sensorless_HighFreq_Startup	4			Closed-loop sensorless- FOC current control with high frequency injection at startup
Curr_Mode_FOC_Sensorless_Dyno	5			Closed-loop sensorless- FOC current control in dyno mode (waiting for observer lock to start up)
Curr_Mode_FOC_Encoder_Align_Startup	6			Closed-loop sensored- FOC current control with encoder feedback and pre-alignment at startup
Curr_Mode_FOC_Hall	7			Closed-loop sensored- FOC current control with hall sensor feedback
Curr_Mode_Block_Comm_Hall			1	Closed-loop block-commutation current control with hall sensor feedback
Trq_Mode_FOC_Sensorless_Align_Startup		1		Closed-loop sensorless- FOC torque control with pre-alignment at startup
Trq_Mode_FOC_Sensorless_SixPulse_Startup		2		Closed-loop sensorless- FOC torque control with six pulse injections at startup
Trq_Mode_FOC_Sensorless_HighFreq_Startup		3		Closed-loop sensorless- FOC torque control with high frequency injection at startup
Trq_Mode_FOC_Sensorless_Dyno		4		Closed-loop sensorless- FOC torque control in dyno mode (waiting for observer lock to start up)
Speed_Mode_FOC_Sensorless_Align_Startup	8	5		Closed-loop sensorless- FOC speed control with pre-alignment at startup

Parameter configuration

Control mode	Build			Description
	RFO	SFO	TBC	
Speed_Mode_FOC_Sensorless_SixPulse_Startup	9	6		Closed-loop sensorless- FOC speed control with six pulse injection at startup
Speed_Mode_FOC_Sensorless_HighFreq_Startup	10	7		Closed-loop sensorless- FOC speed control high frequency injection at startup
Speed_Mode_FOC_Sensorless_Volt_Startup	11	8		Closed-loop sensorless- FOC speed control with open-loop V/Hz at startup
Speed_Mode_FOC_Sensorless_Current_Startup	12			Closed-loop sensorless- FOC speed control with open-loop current at startup
Speed_Mode_FOC_Encoder_Align_Startup	13			Closed-loop sensored- FOC speed control with encoder feedback and pre-alignment at startup
Speed_Mode_FOC_Hall	14			Closed-loop sensored- FOC speed control with hall sensor feedback
Speed_Mode_Block_Comm_Hall			2	Closed-loop block-commutation speed control with hall sensor feedback
Profiler_Mode	15	9		Profiler mode

Each configured control mode operates through a structured state machine architecture, where the system progresses through a series of well-defined operational states. The current system state is stored in the sm[x].current variable. The list of states is shown in Figure 10.

Parameter configuration

State	Build			Description
	RFO	SFO	TBC	
Init	0	0	0	Initialization
Brake_Boot	1	1	1	Brake and Bootstrap
Align	2	2		Aligning (pre-positioning)
Six_Pulse	3	3		Six pulse injection
High_Freq	4	4		High-frequency-injection locking
Speed_OL_TO_CL	5	5		Transition from open-loop to closed-loop
Dyno_Lock	6	6		Waiting for obsever lock to start up in dyno mode
Prof_Finished	7	7		Profiler, finished
Prof_Rot_Lock	8	8		Profiler, rotor locking
Prof_R	9	9		Profiler, stator resistance estimation
Prof_Ld	10	10		Profiler, d-axis inductance estimation
Prof_Lq	11	11		Profiler, q-axis inductance estimation
Current_OL	12			Open-loop current control
Volt_HZ_OL	13	12	3	Open-loop Volt/Hz control
Speed_CL	14	13	4	Closed-loop speed control
Fault	15	14	5	Fault
Torque_CL		15		Closed-loop torque control
Current_CL	16		6	Closed-loop Current control

Figure 10 List of state in Motor Control

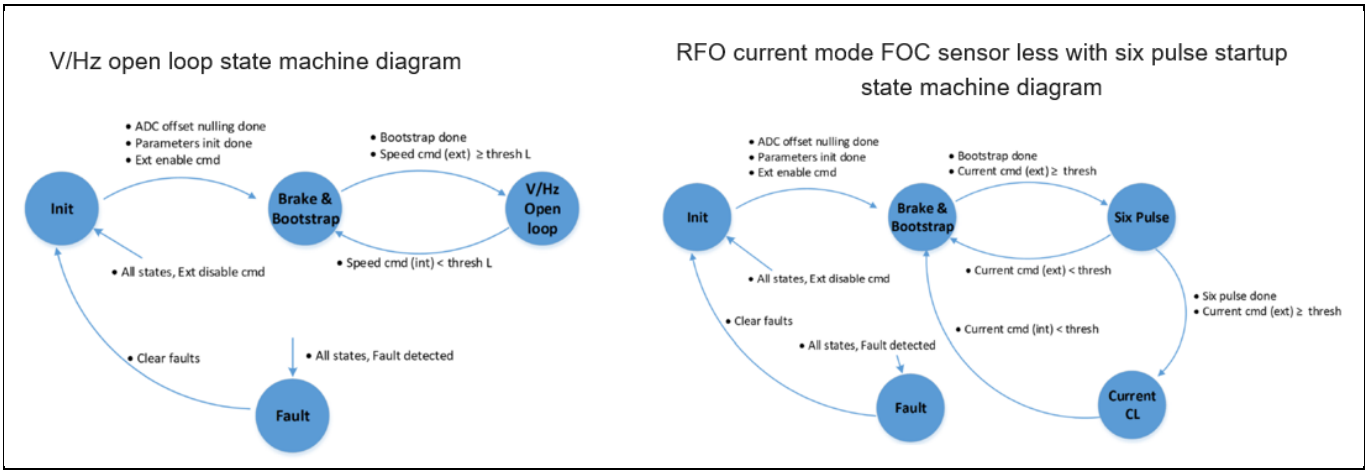


Figure 11 Control mode types and associated state machines – Example

- Control mode is configured using “params[x].ctrl.mode” parameter and configuration macro is defined in *paramconfig.h* file
- “sm[x].current” variable holds the current state machine state

3.3.2 Control loop frequency

The motor control system implements two primary control loops with interdependent frequencies:

- Fast Control Loop: Executes current control algorithms and operates at the highest frequency
- Slow Control Loop: Handles speed control and system management functions

The PWM frequency is determined by the fast control loop frequency multiplied by the configured ratio between PWM frequency and fast loop frequency.

The slow control loop frequency is derived from the fast control loop frequency using the configured ratio between slow loop frequency and fast loop frequency.

Control loop frequency-related parameters can be configured through the ModusToolbox™ IDE or Motor Suite GUI.

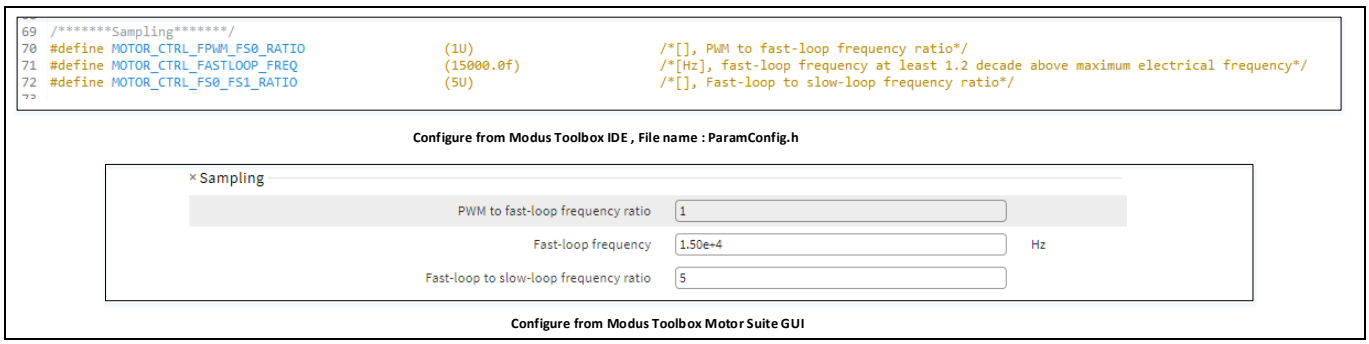


Figure 12 Control loop frequency configuration

Parameter configuration

Table 4 Control loop frequency - Default configured values

Input (Macro defines in ParamConfig.h)	MOTOR_CTRL_FASTLOOP_FREQ	15000.0f	[Hz], fast-loop frequency Param name: params[x]. sys.samp.fs0 = MOTOR_CTRL_FASTLOOP_FREQ params[x]. sys.samp.ts0[sec]= 1/ MOTOR_CTRL_FASTLOOP_FREQ = 66.6E-6f
	MOTOR_CTRL_FS0_FS1_RATIO	5	[], Fast-loop to slow-loop frequency ratio Param name: params[x]. sys.samp.fs0_fs1_ratio = MOTOR_CTRL_FS0_FS1_RATIO params[x].sys.samp.ts1[sec] = params[x].sys.samp.fs0_fs1_ratio/ params[x].sys.samp.fs0 = 333.33E-6f
	MOTOR_CTRL_FS0_FS1_RATIO	1	[#], PWM to Fast-loop frequency ratio Param name: Params[x].sys.samp.fs0_fs1_ratio = MOTOR_CTRL_FS0_FS1_RATIO
Calculated Parameters (calculated in ParamConfig.c)	Params[x].sys.samp.ts0	66.66E-6	[sec], fast-loop period =1/ params[x]. sys.samp.fs0 = 66.66E-6
	Params[x].sys.samp.fpwm	15000.0f	[Hz], PWM frequency =params[x].sys.samp.fs0 * params [x].sys.samp.fpwm_fs0_ratio = 15000.0f*1 = 15000.0f
	Params[x].sys.samp.tpwm	66.66E-6	[sec], PWM period =1/ params[x]. sys.samp. fpwm = 66.66E-6
	Params[x].sys.samp.fs1	3000.0f	[Hz], PWM frequency =params[x].sys.samp.fs0 / params[x].sys.samp.fs0_fs1_ratio = 15000.0f/5 = 3000.0f
	Params[x].sys.samp.ts1	333.33E-6	[sec], slow-loop period =1/ params[x]. sys.samp.fs1 = 1/3000.0f = 333.33E-6

3.3.3 State transition threshold

State transition thresholds determine when the system moves from one state to another and can be configured based on motor speed/ specific events/motor current/ time. When the state machine transitions back to a previous state, a hysteresis value is applied using the formula (Threshold - Hysteresis) to prevent oscillation between states.

3.3.3.1 Start-up threshold parameters

Params[x].ctrl.volt.w_thresh.elec and Params[x].volt.w_hyst.elec parameters control open-loop threshold values. The Params[x].ctrl.volt.w_thresh.elec parameter defines the threshold for transitioning the system from "Brake_Boot" state to startup states (for example, Current_OL, Volt_Hz_OL). The Params[x].volt.w_hyst.elec parameter defines the threshold for transitioning the system from startup states (for example, Current_OL, Volt_Hz_OL) back to "Brake_Boot" state.

3.3.3.2 Observer threshold parameters (speed control):

Params[x].obs.w_thresh.elec and Params[x].obs.w_hyst.elec parameters control observer threshold values. The Params[x].obs.w_thresh.elec parameter defines the threshold for transitioning the system from startup states (for example, Current_OL, Volt_Hz_OL) to Speed_CL state. The Params[x].obs.w_hyst.elec parameter defines the threshold for transitioning the system from Speed_CL state back to startup states (for example, Current_OL, Volt_Hz_OL).

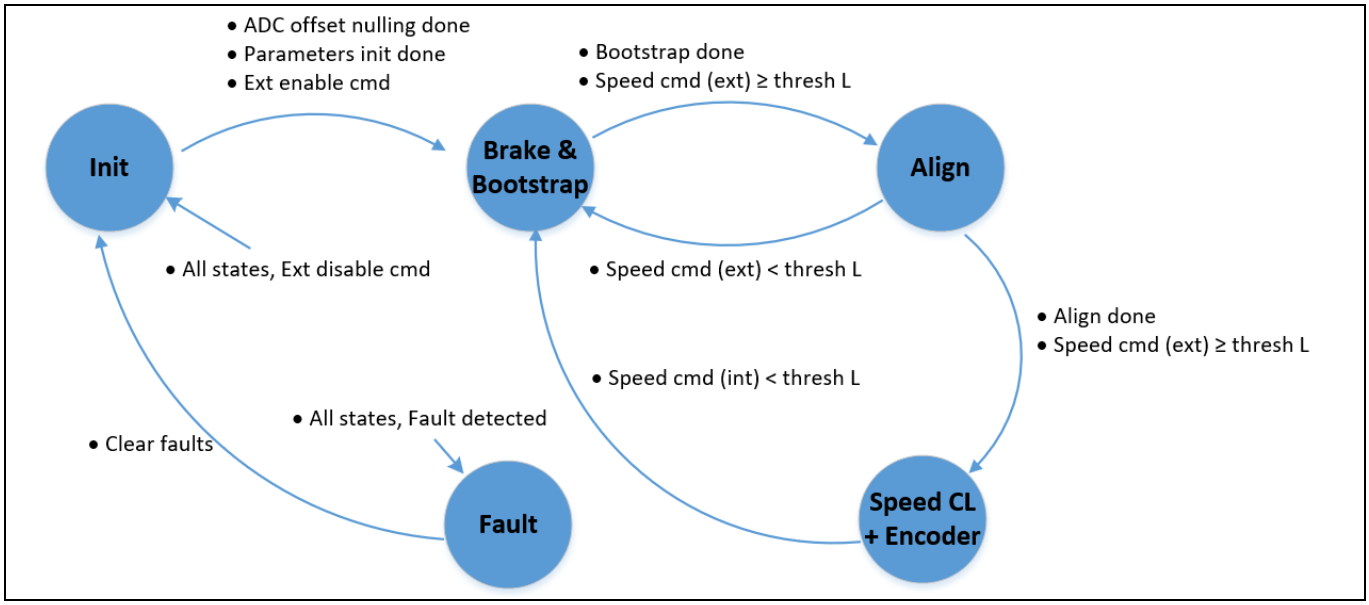


Figure 13 RFO & SFO speed control FOC sensorless with align startup

Parameter configuration

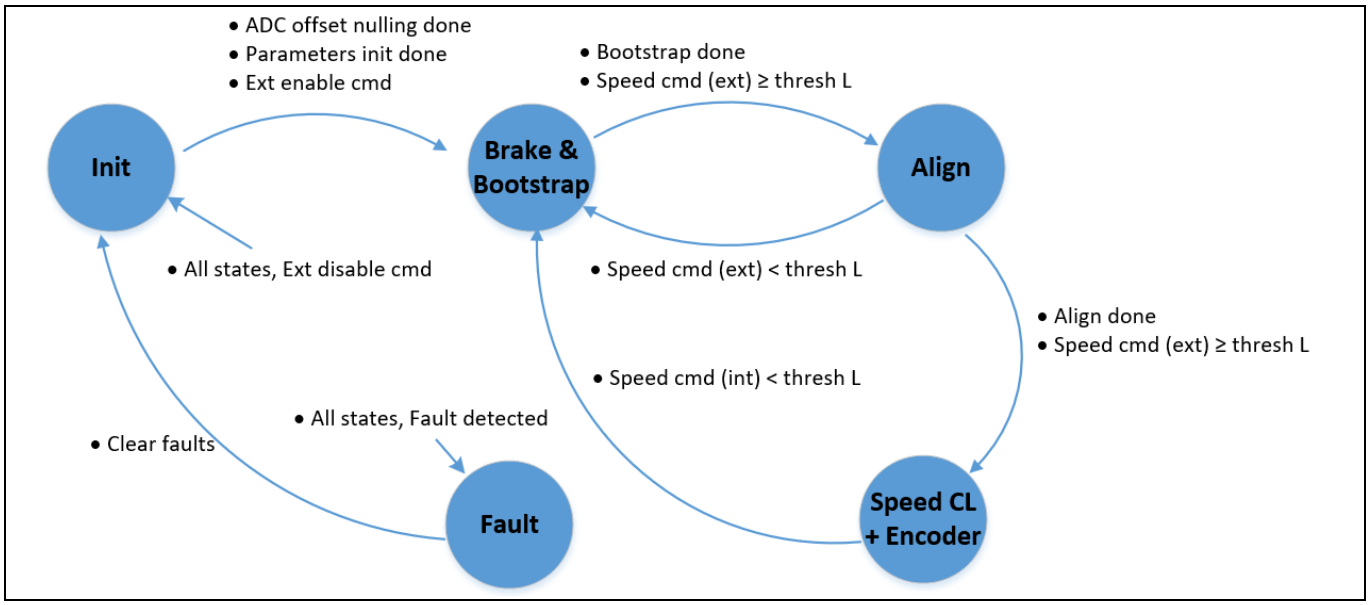


Figure 14 RFO speed control FOC encoder with rotor pre-alignment startup

3.3.3.3 Current control threshold parameters:

Params[x]. ctrl.curr.i_cmd_thresh and Params[x]. ctrl.curr.i_cmd_hyst parameters control observer threshold values. The Params[x]. ctrl.curr.i_cmd_thresh parameter defines the threshold for transitioning the system from "Brake_Boot" state to Current_CL state. The Params[x]. ctrl.curr.i_cmd_hyst parameter defines the threshold for transitioning the system from Current _CL state back to "Brake_Boot" state.

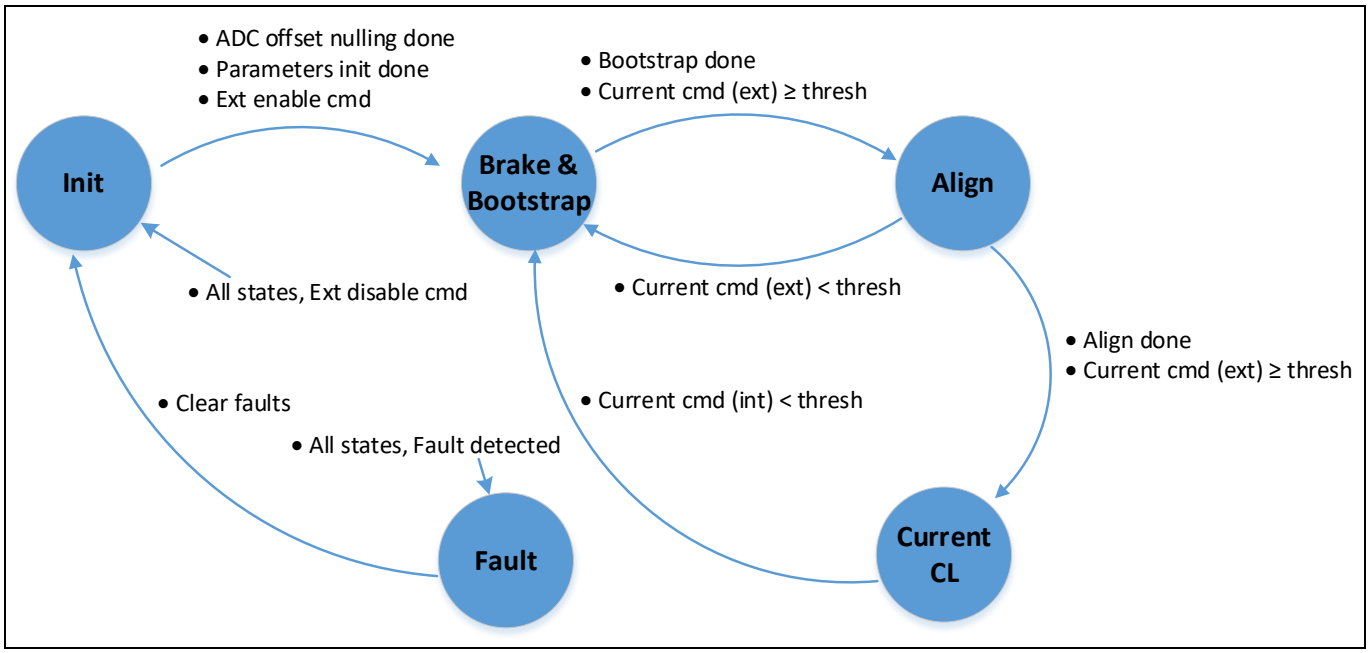


Figure 15 RFO current mode FOC sensorless with align startup state machine

Parameter configuration

Table 5 Startup transition - Default configured values

Input (Macro defines in ParamConfig.h)	MOTOR_CTRL_VOLT_STARTUP_THRESH	200.0f	[RPM], startup threshold. Default value is 5% of motor nominal speed Param name: params[x].ctrl.volt.w_thresh.elec [Ra/sec-elec] = MECH_TO_ELEC(HZ_TO_RADSEC (RPM_TO_HZ(MOTOR_CTRL_VOLT_STARTUP_THRESH)), MOTOR_POLE) Params[x].ctrl.volt.w_hyst.elec = params[x].ctrl.volt.w_thresh.elec * 0.5f
	MOTOR_CTRL_OBS_SPEED_THRESH	800.0f	[RPM], Observer threshold. Default value is 20% of motor nominal speed Param name: params[x]. obs.w_thresh.elec [Ra/sec-elec] = MECH_TO_ELEC(HZ_TO_RADSEC (RPM_TO_HZ(MOTOR_CTRL_OBS_STARTUP_THRESH), MOTOR_POLE) Params[x]. obs.w_hyst.elec = params[x]. obs.w_thresh.elec * 0.5f
	MOTOR_CTRL_CURRENT_STARTUP_THRESH	0.525f	[A], MOTOR_CURRENT_CONT*0.15f, Default value is 15% of motor continuous current Param name: params[x]. ctrl.curr.i_cmd_thresh [A] = MOTOR_CTRL_CURRENT_STARTUP_THRESH Params[x]. ctrl.curr.i_cmd_hyst = params[x]. ctrl.curr.i_cmd_thresh * 0.8f

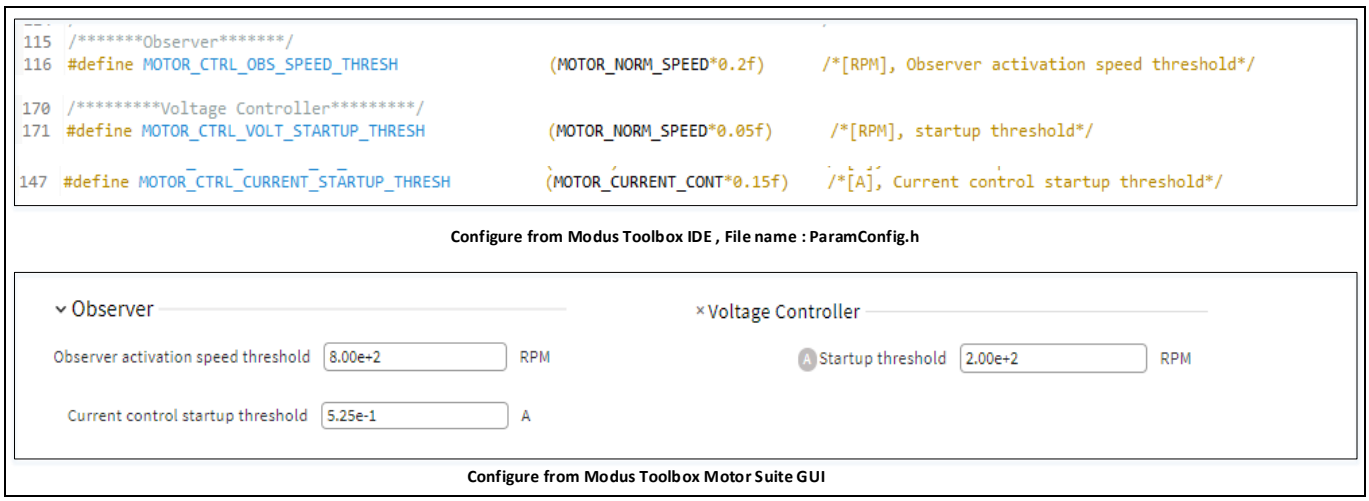


Figure 16 How to configure state threshold parameters

Parameter configuration

3.3.4 Rate limiter

Rate limiters provide a controllable soft start by limiting how quickly speed and current commands can change, preventing abrupt system responses.

Parameters:

- Speed rate limit: params[x].sys.rate_lim.w_cmd.elec [rad/(sec)^2]
- Current rate limit: params[x].sys.rate_lim.i_cmd [A/sec]

Examples:

- With a 10 A/sec current rate limit, it takes 1 second for current to ramp from 0A to 10A, ensuring smooth startup without sudden torque changes.
- With a 100 rad/(sec)^2 speed rate limit, it takes 1 second for speed to accelerate from 0 to 100 rad/sec, providing gradual speed transitions.

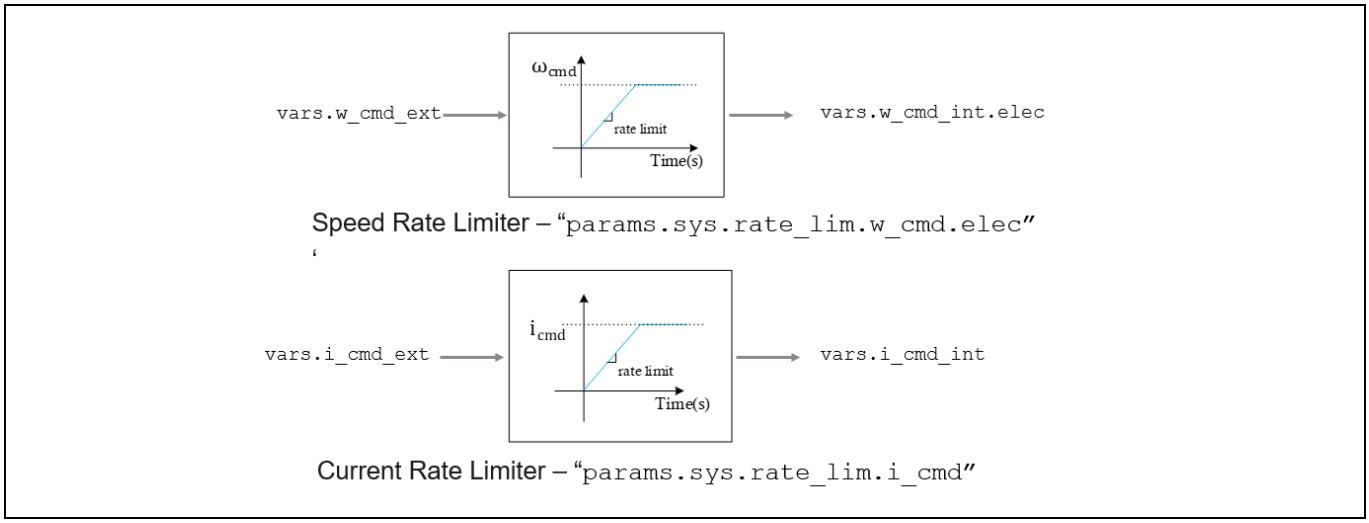


Figure 17 Ramp limiter

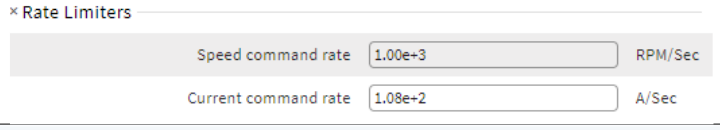
Table 6 Startup transition - Default configured values

Input (Macro defines in ParamConfig.h)	MOTOR_CTRL_SPEED_CMD_RATE	1000.0f	[RPM/sec], startup threshold. Default value is 5% of motor nominal speed Param name: params[x].sys.rate_lim.w_cmd.elec [Ra/sec-elec] = MECH_TO_ELEC(HZ_TO_RADSEC(RPM_TO_HZ(MOTOR_CTRL_SPEED_CMD_RATE)), MOTOR_POLE)
	MOTOR_CTRL_CURRENT_CMD_RATE	108.0f	[A/sec], 10.0f*MOTOR_CURRENT_PEAK. Default value is 10-time motor peak current Param name: params[x].sys.rate_lim.i_cmd [A/sec] = MOTOR_CTRL_CURRENT_CMD_RATE

Parameter configuration

```
79  /*********Rate Limiters*********/
80  #define MOTOR_CTRL_SPEED_CMD_RATE                (1000.0f)                  /*[RPM/sec], Speed command rate*/
81  #if defined(CTRL_METHOD_RFO) || defined(CTRL_METHOD_TBC)
82  #define MOTOR_CTRL_CURRENT_CMD_RATE              (10.0f*MOTOR_CURRENT_PEAK)   /*[A/sec], Current command rate*/
```

Configure from Modus Toolbox IDE , File name : ParamConfig.h



Configure from Modus Toolbox Motor Suite GUI

Figure 18 Rate limiter parameter configuration

4 Verification of ADC measurement

Verification of DC bus voltage and motor current measurement if these values are not matching with actual values how to adjust it or fix.

4.1 DC bus voltage measurement

Read the DC bus voltage using ModusToolbox™ Motor Suite GUI Test Bench, the measured voltage should match with actual DC bus voltage applied to power board with ±2% tolerance. If the measured DC bus voltage does not match with actual DC bus voltage, check the following configurations in code example.

- Voltage resistor divider configuration (“ADC_SCALE_VDC” macro in *Paramconfig.h* file), voltage measurement low side and high side resistance value are configured as per power board.
- ADC voltage reference configuration (“ADC_VREF_GAIN” macro in *Paramconfig.h* file)

It is possible to read the DC bus voltage ADC value directly from “mcu[x].dma_results[2]” variable using Motor Suite GUI Builder. ADC count value should be $4095*V_{in}/V_{ADCREF}$, Where V_{in} is voltage at MCU voltage measurement pin and V_{ADCREF} is 3.3 V.

4.2 Motor phase current measurement

Current sensing is one of the most critical aspects of motor control systems, serving as the foundation for both control performance and system protection. In sensorless control methods particularly, current information becomes the primary feedback source from which rotor position and speed are estimated. Field Oriented Control (FOC) is entirely dependent on accurate current measurements, as the current control loops require precise feedback to regulate d-axis and q-axis currents effectively. Therefore, it is critical to verify current measurement accuracy and quality before running FOC algorithms.

Step 1: Read all three-phase current values without running the motor using Motor Suite GUI Oscilloscope. All three phase current values should be zero or close to zero.

Optionally read the current ADC count value using Motor suite GUI Builder. In case of leg shunt configuration, “mcu[x].dma_results[0]”, “mcu[x].dma_results[5]” and “mcu[x].dma_results[1]”variables hold phase U (Iu), phase V (Iv), and phase W (Iw) currents, respectively. ADC count value should match with the current input-offset value. In case of single shunt configuration, read “mcu[x].dma_results[0]” and “mcu[x].dma_results[5]” variables value should match with the current input offset.

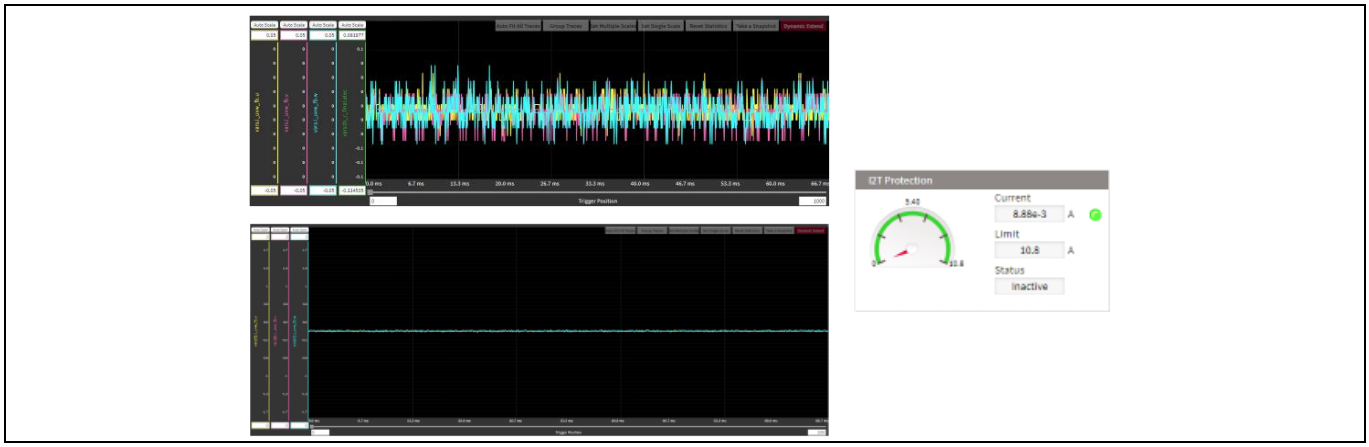


Figure 19 Phase current waveform- While the motor is not running

Verification of ADC measurement

Step 2: Run the motor in V/F open loop mode and read motor phase currents using Motor Suite GUI Oscilloscope. The measured value should match with the actual current drawn by the motor.

If the measured current values are not matching with actual current, check the following parameters configurations are matched with the power board.

- ADC voltage reference configuration (“ADC_VREF_GAIN” macro in Paramconfig.h file)
- Current measurement shunt value configuration (“ADC_CS_SHUNT_RES” macro in Paramconfig.h file)
- Current measurement external amplifier configuration (“ADC_CS_OPAMP_GAIN” macro in Paramconfig.h file)
- Configured shunt type (“ADC_CS_SHUNT_TYPE”), polarity(“ADC_CS_CURRENT_SENSE_POLARITY”) and measurement type (“ADC_CS_CURRENT_MEASUREMENT_TYPE”)

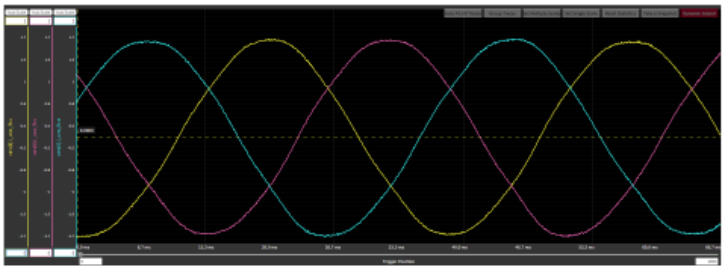


Figure 20 Phase current waveform - V/F open-loop mode

In the case of single shunt current measurement, three parameters directly influence current measurement. These parameters are defined in configuration/motor-ctrl-lib-config/ParamConfig.h and require proper configuration for accurate current sensing.

1. “ADC_CS_SS_MIN_SEGMENT_TIME” defines the minimum measurable window for single shunt current measurement, specified in microseconds [µs]. This parameter defines the minimum PWM on-time for current measurement, so it has the most significant impact on measurement accuracy, with a default value of 3µs. Optimization of this value can improve system performance and accurate current measurement.
2. “ADC_CS_SETTLE_RATIO” represents the settling ratio used for single-shunt current sampling timing. This parameter allows adjustment of the measurement timing to avoid current sensing during switching transients or ringing periods. It can be fine-tuned based on board-specific propagation delays and switching characteristics, with default values typically working well for most standard applications.
3. “MOTOR_CTRL_SS_HMOD_KI” serves as the harmonic modification parameter for single shunt current measurement. This parameter helps optimize current reconstruction algorithms and typically functions effectively with default settings without requiring adjustment in most applications.

Begin optimization with the minimum segment time parameter as it provides the most significant improvement potential. Example, start by reducing ADC_CS_SS_MIN_SEGMENT_TIME from 3µs to 1-1.25µs and verify current sensing. If further refinement is needed, adjust ADC_CS_SETTLE_RATIO to account for board-specific characteristics. Monitor current measurement quality throughout the tuning process to ensure stable and accurate reading across the complete operating range.

5 Motor parameter identification

Motor electrical and mechanical parameters are critical for precise motor operation. This chapter explains how to identify these parameters.

5.1 Parameter identification using Motor Suite motor profiler

The Profiler feature automates extraction of motor parameters so users can tune the system when using a new motor and/or mechanical load. The profiler extracts the following parameter by aligning the motor into a known position and running the motor in speed control mode.

- Motor parameters
  - Stator resistance, r (“params[x].motor.r“)
  - Stator q-axis inductance, L_q (“params[x].motor.lq“)
  - Stator d-axis inductance, L_d (“params[x].motor.ld“)
  - Rotor permanent-magnet flux linkage, λ_m (“params[x].motor.lam “)
- Mechanical Parameters (used in speed controller)
  - Inertia, J (“params[x].mech.inertia”)
  - Viscous, B (“params[x].mech.viscous”)
  - Friction, Tf (“params[x].mech.viscous”)

The profiler is not a substitute for engineering judgment or expertise in tuning motor control systems; it is a tool to facilitate that process. It assumes a simplified first-order mechanical model (inertia and friction), whereas some loads exhibit resonances, for example, when there is a significant inertia mismatch between the motor and the load. Therefore, the user should fine-tune the parameters based on the application’s characteristics. There are temperature variations, aging effects, and other environmental factors, so these values are estimated and can vary with operating conditions and time.

Note: In Profiler mode, it is required to run the motor in both current open loop and speed closed loop modes to identify the load parameters. This identification process can only be executed successfully when both underlying control modes are already functioning properly - therefore, when the motor is not operating correctly in current open loop mode, the Current Open Loop section (6.2 Open-loop I/F) can be referenced for adjusting the appropriate open loop parameters, and when the motor is not working properly in speed closed loop mode, the Speed Loop section (7.2 Speed) can be referenced for tuning the speed controller parameters.

5.1.1 Motor Suite motor profiler execution steps

The profiler executes the following steps to estimate parameters:

1. Lock the rotor (electrical identification)

- Align and lock the rotor at known electrical degrees by commanding Iα = params[x].profiler.i_cmd_dc and Iβ = 0 for “params[x].profiler.time_rot_lock”.
- While locked, estimate phase resistance Rs from the DC injection.
- Still at lock, estimate inductances Ld and Lq by injecting a high-frequency current (1 kHz) with amplitude “params[x].profiler.i_cmd_ac“and measuring the response.
- Use the identified Rs, Ld, and Lq to initialize the sensorless rotor estimator for the next step.

Motor parameter identification

2. Run the motor (mechanical and flux identification)

- Run the motor with speed command from 0 to “params[x].profiler.w_cmd_elec.max“. The ramp rate is set by “params[x].sys.rate_lim.w_cmd.elec“.
- Start the motor in open-loop current control with current command “params[x].ctrl.curr.i_cmd_ol” and switch to closed-loop speed control at motor speed reaches “params[x].obs.w_thresh.elec.”
- Begin mechanical/flux estimation once the speed reaches params[x].profiler.w_cmd_elec.min.”
- Perform estimation over multiple speed steps. At each step, hold speed for “params[x].profiler.time_spd”and estimate:
  - B: viscous friction coefficient
  - Tf: Coulomb (dry) friction torque
  - λ_m : Rotor flux linkage
- Finalize B, Tf, and λ_m when the motor speed reaches “params[x].profiler.w_cmd_elec.max.”

3. Ramp down (inertia identification)

- Ramp the commanded speed down to “params[x].profiler.w_cmd_elec.min.”
- Estimate inertia J from the speed response during ramp-down (using measured acceleration/torque)

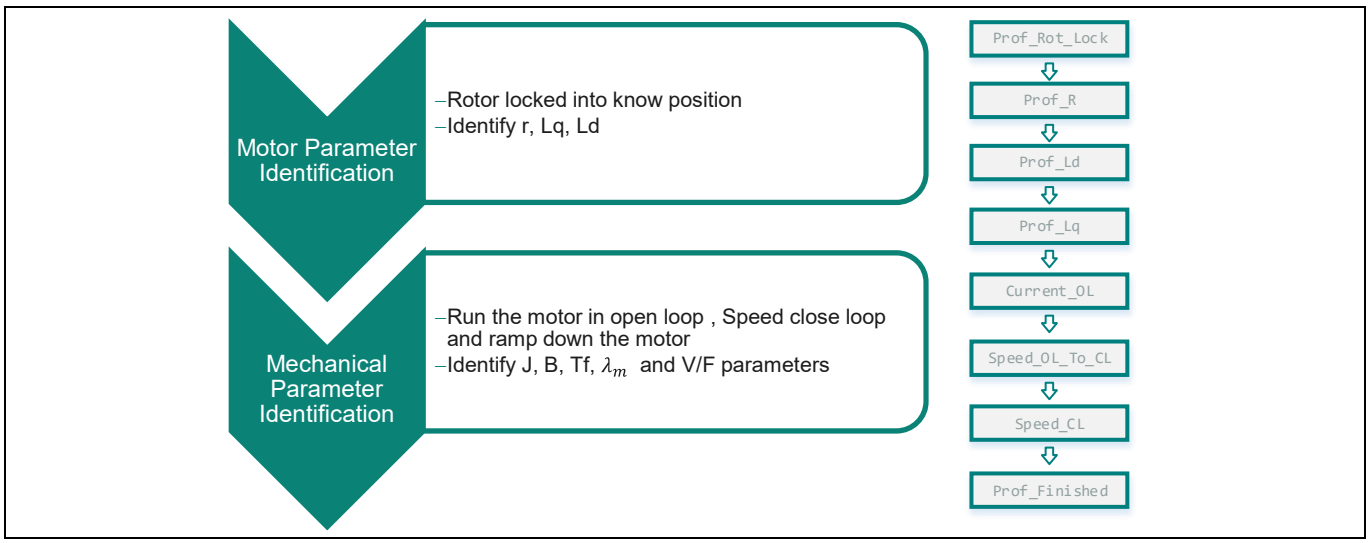


Figure 21 Profiler execution steps

Note: Refer to the motor control firmware reference manual for estimation details.

As the Profiler runs the motor in speed-control mode to estimate parameters, it is essential to preconfigure the following motor and mechanical parameters using the motor’s nameplate, datasheet, or best-known values.

5.1.2 Profiler parameters

The Motor Control Library includes configurable input parameters for profiler setup that list of parameters is described in Table 7.

Motor parameter identification

Table 7 Profile input parameters

Parameter Name	Description
params[x].profiler.overwrite	Enable: After the profiler estimates values, it replaces the configured motor and load parameters with its results. Disable: Keep existing parameters; do not apply profiler estimates. Config macro: MOTOR_CTRL_PROFILER_PARAM_OVERWRITE
params[x].profiler.cmd_thresh	[%], Activation command threshold for profiler. The profiler starts only if the speed command value exceeds this threshold value. Config macro: MOTOR_CTRL_PROFILER_CMD_THRESH
params[x].profiler.cmd_hyst	[%], Activation command hysteresis for profiler Config macro: MOTOR_CTRL_PROFILER_CMD_HYST
params[x].profiler.i_cmd_dc	[A], Target DC current applied to lock the rotor and resistance estimation Config macro: MOTOR_CTRL_PROFILER_I_CMD_DC
params[x].profiler.i_cmd_ac	[A], Target AC current applied to inductance estimation Config macro: MOTOR_CTRL_PROFILER_I_CMD_AC
params[x].profiler.w_cmd_elec.min	[Ra/sec-elec], Start threshold to start the mechanical parameter estimation Config macro: MOTOR_CTRL_PROFILER_SPEED_CMD_MIN
params[x].profiler.w_cmd_elec.max	[Ra/sec-elec], Maximum motor speed in profiler mode Config macro: MOTOR_CTRL_PROFILER_SPEED_CMD_MAX
params[x].profiler.time_rot_lock	[sec], Rotor locking time Config macro: MOTOR_CTRL_PROFILER_ROTOR_LOCK_TIME
params[x].profiler.time_spd	[sec], Duration to maintain constant motor speed in each step for mechanical parameter estimation Config macro: MOTOR_CTRL_PROFILER_FLUX_EST_TIME

Advanced profiler parameters are derived from profile input parameters.

```
/*******Profiler*******/
#if defined(CTRL_METHOD_RFO) || defined(CTRL_METHOD_SFO)
#define MOTOR_CTRL_PROFILER_PARAM_OVERWRITE        (Dis)                      /*Write the parameter value calculated from profiler*/
#define MOTOR_CTRL_PROFILER_CMD_THRESH             (5.0f)                      /*[%], Activation command threshold*/
#define MOTOR_CTRL_PROFILER_CMD_HYST               (2.5f)                      /*[%], Activation command hysteresis*/
#define MOTOR_CTRL_PROFILER_I_CMD_DC    (MOTOR_CURRENT_CONT * 0.5f)            /*[A], Target DC current*/
#define MOTOR_CTRL_PROFILER_I_CMD_AC    (MOTOR_CURRENT_CONT * 0.25f)           /*[A], Target AC current*/
#define MOTOR_CTRL_PROFILER_SPEED_CMD_MIN  (2.0f * MOTOR_CTRL_OBS_SPEED_THRESH) /*[RPM], Initial electrical speed*/
#define MOTOR_CTRL_PROFILER_SPEED_CMD_MAX   (0.75f * MOTOR_NORM_SPEED)         /*[RPM], Final electrical speed*/
#define MOTOR_CTRL_PROFILER_ROTOR_LOCK_TIME        (1.0f)                      /*[sec], Rotor locking time*/
#define MOTOR_CTRL_PROFILER_FLUX_EST_TIME          (1.5f)                      /*[sec], Flux estimation time*/
#endif
/*****************************************************************************/
```

Figure 22 Profiler input parameter config macros in *ParamConfig.h*

Motor parameter identification

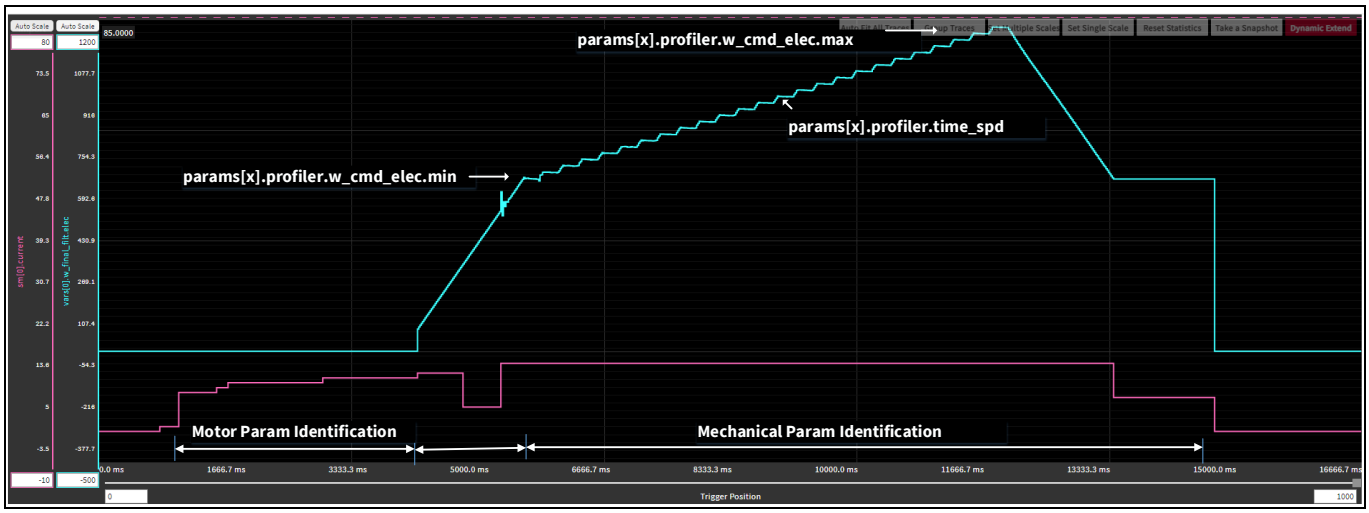


Figure 23 Motor profiler: State and motor speed

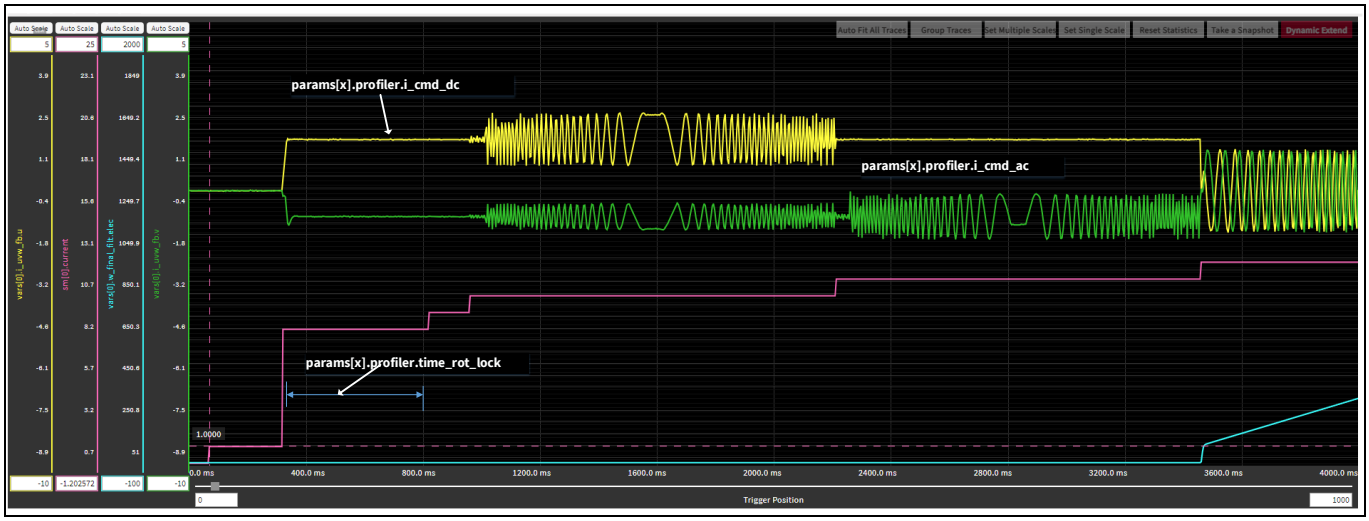


Figure 24 Motor profiler: Motor parameter identification stage

5.1.3 Parameter tuning in profiler mode

This section explains the parameters that influence each stage of the profile.

5.1.3.1 Rotor lock state

To extract motor parameters, first lock the rotor at a known electrical angle by applying sufficient DC current and allowing time for it to rotate and settle.

By default, the current command (“params[x].profiler.i_cmd_dc”) is set to 40% of the motor’s continuous current, that is typically adequate for most cases; Increase this value as needed based on the motor load condition.

The default rotor lock time (“params[x].profiler.time_rot_lock”) is 1 sec. If required, adjust the lock time based on the load condition.

Below diagram depicts the rotor locking stage for a typical IPM with an initial rotor angle of $\theta = \pi/4,$ by regulating and applying a constant dc current along the α-axis, the rotor flux linkage will align itself with the α-axis and after settling, the rotor angle would be $\theta = /2$.

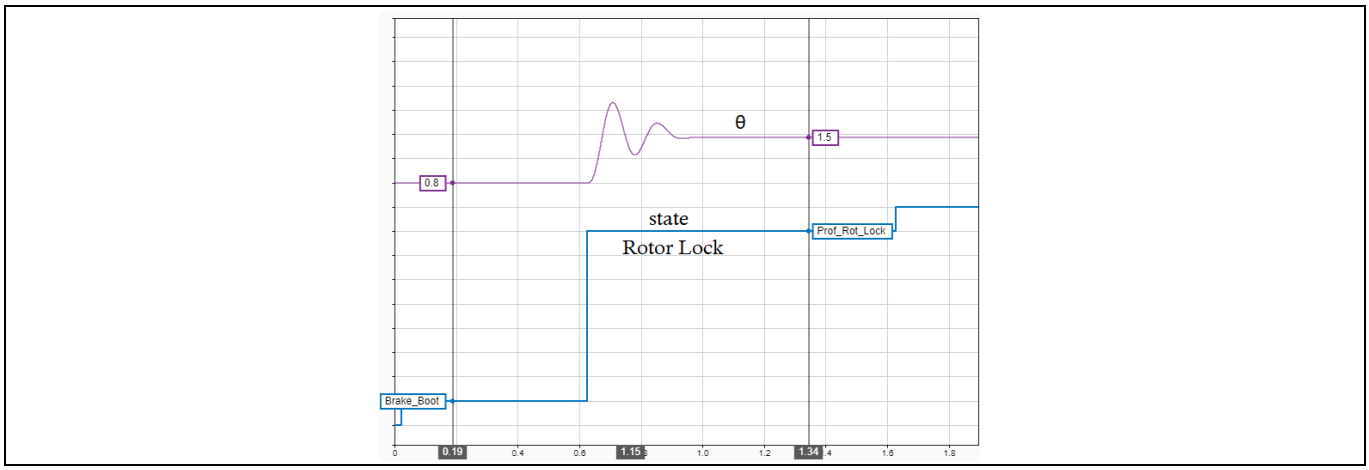


Figure 25 Rotor locking stage, starting from $\theta = \pi/4$ as the initial rotor angle and locking at $\theta = \pi/2$

5.1.3.1 Resistance estimation

When the rotor is locked and the dc-currents reach their steady-state value, the resistance of the motor is estimated using the applied DC voltage, V_α, and the commanded DC current, I_α^*, as follows:

$$r = \frac{V_\alpha}{I_\alpha} = \frac{V_\alpha}{I_\alpha^*}$$

During motor resistance estimation, the system measures the total resistance path including motor windings, connecting cables, and power switching device on-resistance. This comprehensive measurement may show slightly higher resistance values than an LCR meter reading of just the motor windings, especially for very low-resistance motors (<10 mΩ). This is normal behavior and will not impact motor control performance.

The following diagram illustrates the resistance-estimation stage, filtered V_α used in estimated of resistance.

Motor parameter identification

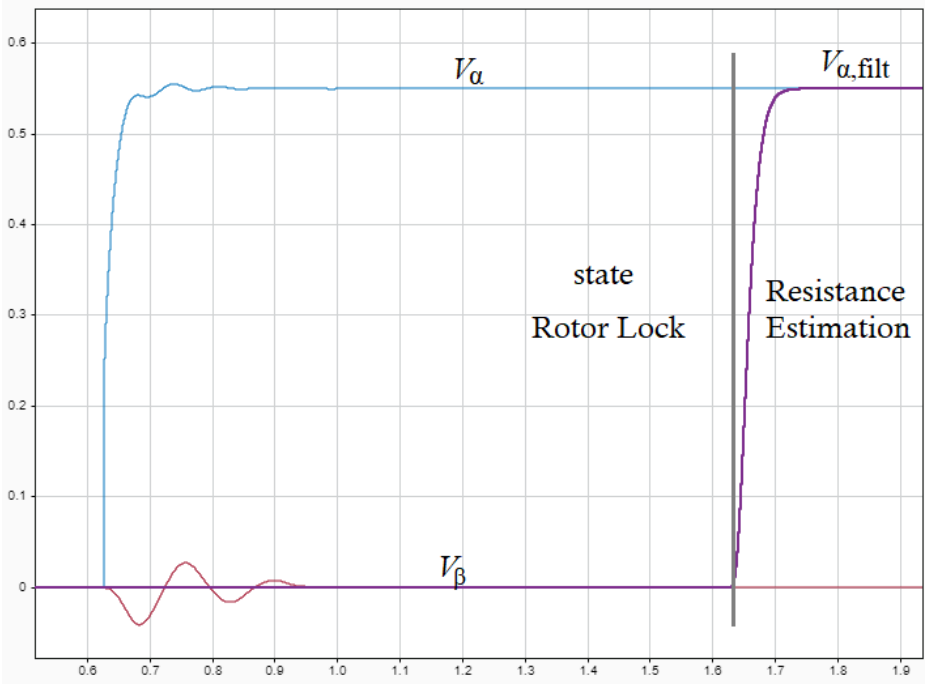


Figure 26 Stator resistance estimation stage

5.1.3.2 Inductance estimation

In motor inductance value estimation, high-frequency components are also injected in the $\alpha\beta$-axes in addition to the dc component applied to α-axis lock the rotor. First, inject the high-frequency voltage to the α-axis and measure its corresponding current. This value is used to estimate motor inductance Ld. Then inject the high-frequency current to the β-axis and measure its corresponding current. This value is used to estimate motor inductance Lq.

To improve estimation accuracy, profiler uses multiple injection frequencies around 1 kHz (900–1100 Hz) when estimating inductance. When validating estimated values against LCR meter readings, ensure frequency consistency. As inductance estimation uses 1 kHz injection, configure the LCR meter to 1kHz for meaningful comparison

- High Frequency Component

High-frequency current components are defined in “params[x]profiler.i_cmd_ac” parameter, high-frequency component value should be less than DC component “params[x].profiler.i_cmd_dc,” to avoid torque generation from the high- frequency component. Default high frequency component value is 25% of motor continuous current, this is half the value of dc component. Also injected dc and ac currents must be lower than the peak current rating of the motor otherwise the measurements would not be accurate due to saturation of the inductances along both axes (L_q and L_d). It is recommended that the injected currents do not go above 25% of the peak current rating of the motor.

- Zero Inductance Estimation issue:

This occurs when the motor's electrical time constant (L/R) approaches or falls below the sampling period (1/PWM_frequency). For low-inductance motors, the system cannot resolve the L/R dynamics within the sampling window. Increase the PWM frequency to provide sufficient time resolution for accurate inductance estimation.

Motor parameter identification

Below diagram depicts the iterative current control and inductance estimation. L_q and L_d estimations will converge to their corresponding real values while the current magnitudes converge to the commanded current ($|\hat{\imath}|^*$).

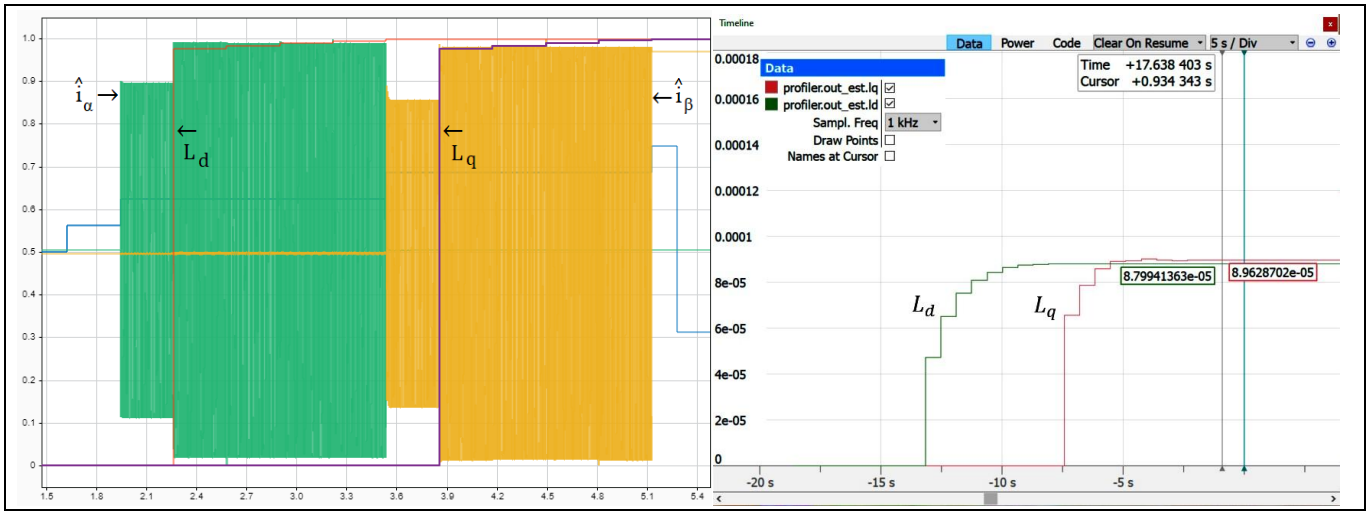


Figure 27 Iterative current control and inductance estimation in the inductance-measurement stage

5.1.3.3 Mechanical parameter and flux linkage estimation

After resistance and inductance value estimation, these estimated values are used for sensorless rotor angle estimation instead of configured value.

It is necessary to run the motor to estimate mechanical parameters and flux linkage values. Start the motor in the current open loop with current command value “params[x].ctrl.curr.i_cmd_ol.” Once the speed command reaches the observer threshold (“params[x].obs.w_thresh.elec”), state moves from current open loop to speed close loop. In speed close loop operation, observer estimation rotor angle is used. Speed Ramp up to the configured maximum speed, “params[x].profiler.w_cmd_elec.max.” Begin mechanical/flux estimation once the speed reaches “params[x].profiler.w_cmd_elec.min.”

- Motor is not running in current open loop mode or over-current:

  Make sure “params[x].ctrl.curr.i_cmd_ol” is configured correctly as per the system. Default value of “params[x].ctrl.curr.i_cmd_ol” is 40% of motor continuous current. Increase the current command if motor is not running and reduce in case of overcurrent.

  It is essential to configure speed ramp rate is configured as per the system.
- Motor is not running in speed loop running:

  Make sure profiler min and max speed parameter configuration as per motor specification. It is required to set min value more than observer speed threshold and max value less than motor nominal speed(around 50%).

  Adjust the speed and current loop bandwidth to run the motor in speed close loop.

  Adjust the speed ramp rate based on system configuration

V/F constant and V/F offset values are also estimated in the profiler.

During mechanical parameter identification the motor should run in open loop and close loop, otherwise the estimated parameter will not be correct.

Motor parameter identification

5.1.4 How to run a motor profiler using Motor Suite GUI

- Open ModusToolbox™ Motor Suite and create project based on hardware setup used
- Make sure Motor Suite GUI is connected to the target device
- Use program button , Program the default binary file.
- Configure motor and mechanical parameters in Configurator view using the motor's nameplate, datasheet, or best-known values using Motor Suite GUI configurator.

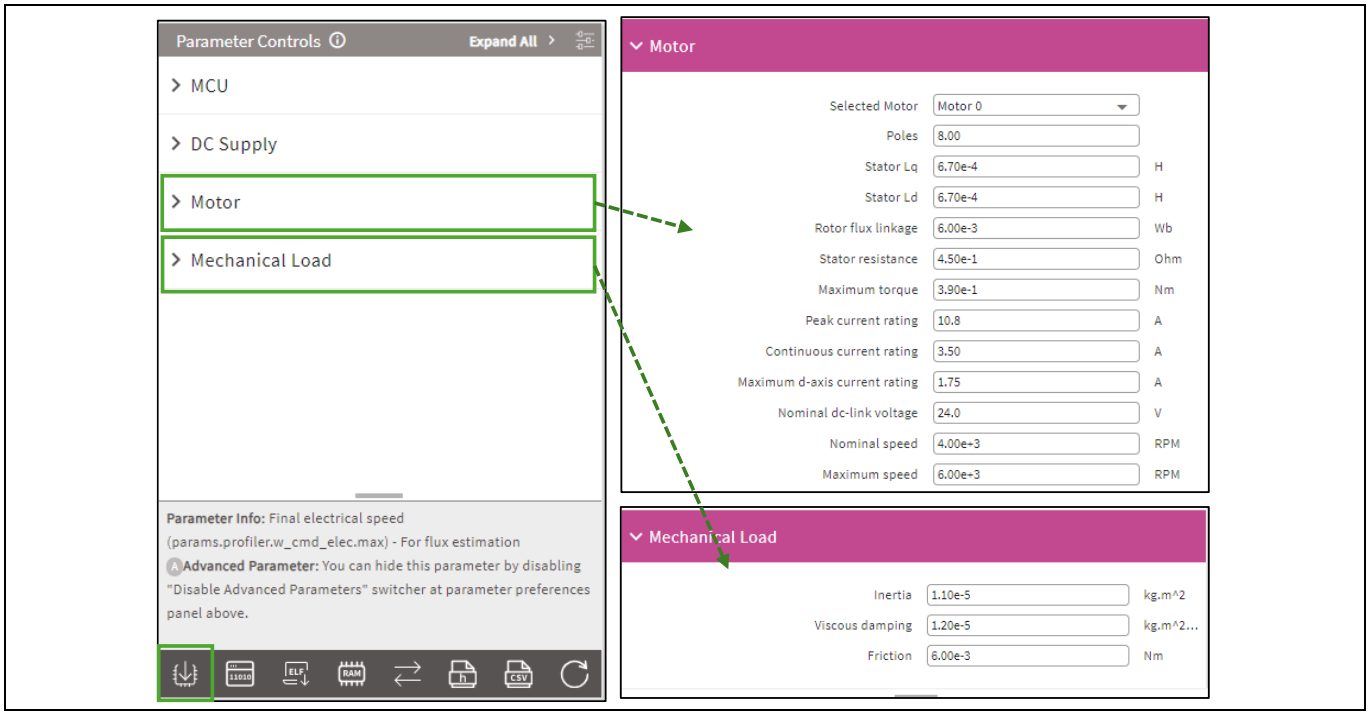


Figure 28 Motor and mechanical parameter configuration

- Configure profiler parameters (refer Table 7) in Motor Suite GUI configurator

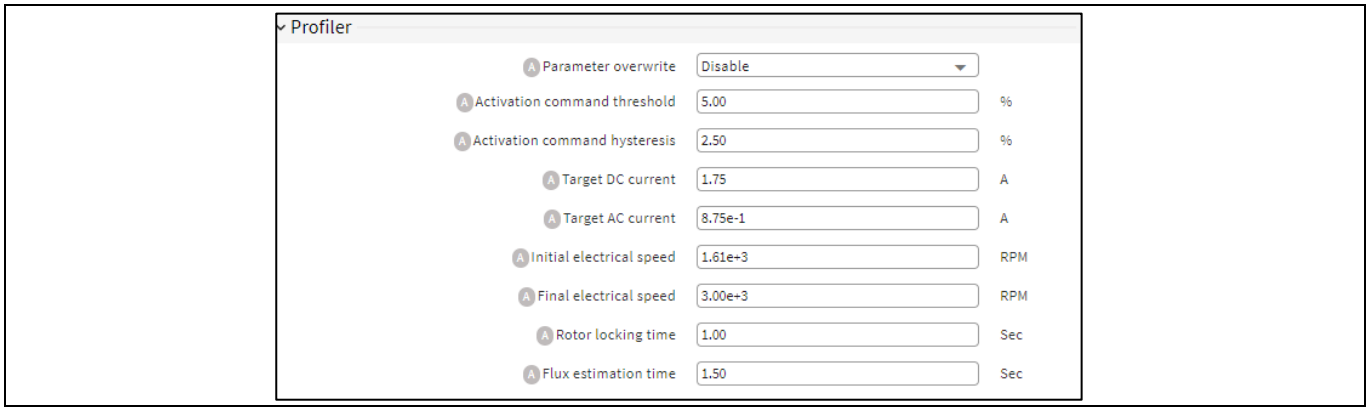


Figure 29 Profiler parameter configuration

Motor parameter identification

- Program the parameter into RAM or Flash
- Open profiler window from Test Bench view and set speed command (>5%) and start the profiler by enabling the driver. If motor is not running during mechanical parameter identification Change the “Dynamic Response” to Slow / Fast / Custom or only adjust the speed loop bandwidth (Ex. reduce speed loop bandwidth)
- Once profiler completes the estimation, update this value into Motor Suite GUI manually or automatically

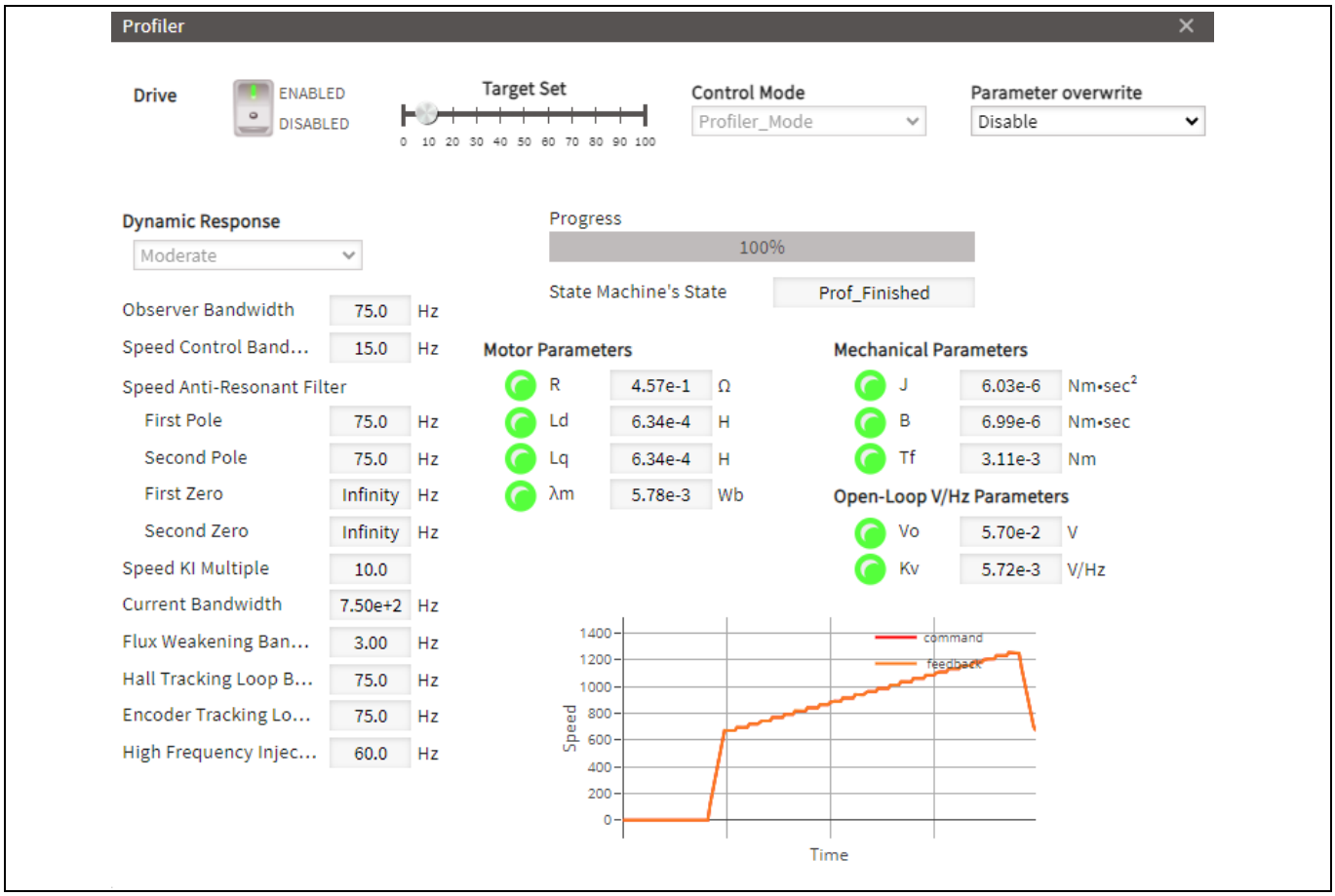


Figure 30 Profiler window

5.2 How to measure motor parameter manually

In this section, it explains how to measure basic motor parameters as listed here:

- Stator resistance per phase (Rs)
- Stator inductance per phase (lq, ld)
  - IPM motor stator lq inductance per phase
  - IPM motor stator ld inductance per phase
- Motor poles (p)

5.2.1 Stator resistance (Rs)

The measurement method of the stator inductance with the equivalent circuit of the motor is shown in Figure 31. It measures the line-to-line resistance by LCR meter, but this measurement result is the sum of the two resistances of both lines. The motor control parameter of a stator resistance parameter (R_s) represents the winding resistance of the motor per phase, so the measurement result should be divided by 2 (for star-connected windings).

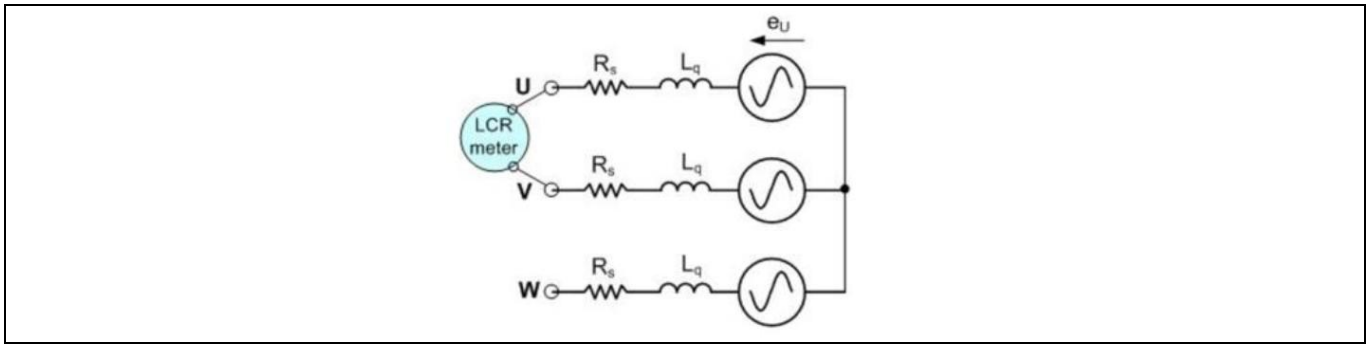


Figure 31 Measurement method of stator resistance (Rs) and stator inductance (Lq, Ld)

5.2.1.1 Measurement procedure

The measurement procedure of R_s is as follows,

1. Connect two phases to LCR meter, and leave the third phase open
2. Measure the line-to-line resistance value
3. Divide the measured resistance value by 2

5.2.1.2 Measurement example

The following diagram shows the actual measurement example of the line-to-line stator resistance. The measurement result in this example is 94.28, so, the parameter value of the R_s is 47.14 Ω.

Motor parameter identification

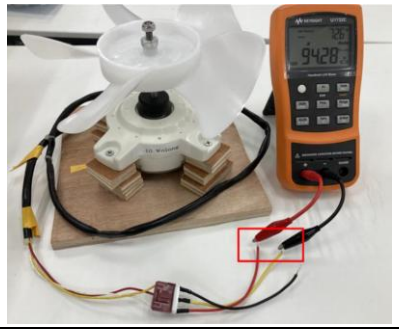


Figure 32 Measure Line to Line stator resistance (2Rs)

5.2.2 Stator inductance (Lq, Ld)

Motor Suite Motor Control Lib advanced sensorless Field Oriented Control (FOC) algorithm supports both surface mounted permanent magnet (SPM) motors and interior permanent magnet (IPM) motors.

When working with surface permanent magnet (PM) motors the winding inductances Ld and Lq will have the same value with any rotor angle as shown in Figure 33, left. However, when working with interior permanent magnet (IPM) motors the winding inductance varies with the rotor angle as shown in Figure 33, right, and the Lq inductance is greater than the Ld inductance.

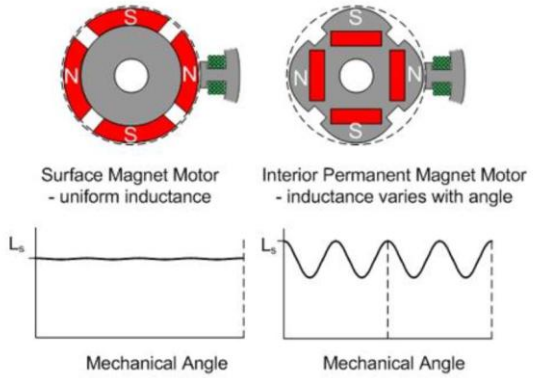


Figure 33 Motor type (PM motor, IPM motor)

This section explains how to measure the winding inductance values Lq and Ld when using a 4-pole IPM motor.

When using an LCR meter to measure inductance, the result is the sum of two-phase inductances (line-to-line inductance) because the measurement path goes through two windings in series. To obtain the actual phase inductance value, you need to divide the measured line-to-line inductance by 2 (for star-connected windings).

And since the inductance value of the IPM motor varies with the rotor angle, it is necessary to adjust the angle to measure the Lq and Ld value. Figure 34 shows the inductance value of the 4 pole IPM motor with respect to the electrical angle and the mechanical angle. With respect to the 4-pole motor, one cycle of the electrical angle is half the cycle of the mechanical angle. And there are 4 peaks (maximum and minimum) in inductance value in one electrical cycle. So, it means that the inductance value is changed from maximum to bottom between 45 degrees in the mechanical angle. Therefore, to measure the inductance value by changing the motor angle gradually by hand within 45 degrees in the mechanical angle, and the maximum value is the Lq, and the minimum value is the Ld.

Motor parameter identification

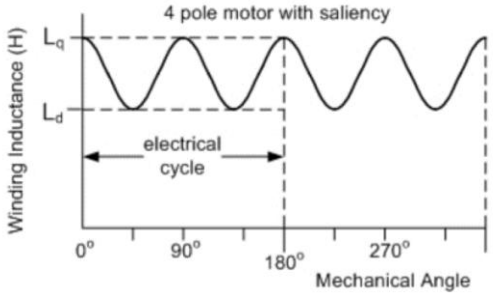


Figure 34 Stator inductance (Lq, Ld)

5.2.2.1 Measurement procedure

The measurement procedure of L_q and L_d is as follows,

1. Connect 2 phases to LCR meter (set injection frequencies 1 kHz) and leave third phase open
2. Measure the line-to-line inductance value
3. Rotate the rotor gradually by hand and record the highest inductance value as L_q, and the lowest inductance value as L_d
4. Divide these measured inductance values by 2

5.2.2.2 Measurement example

Figure 35 shows the actual measurement example of the line-to-line stator inductance. The measurement result of Lq is shown on the right, and Ld on the left. The measurement result of the Lq is 550.0 mH, so, the stator inductance per phase is 275.0 mH. The measurement result of the Ld is the 469.0 mH, so, the stator inductance per phase is 234.5 mH.

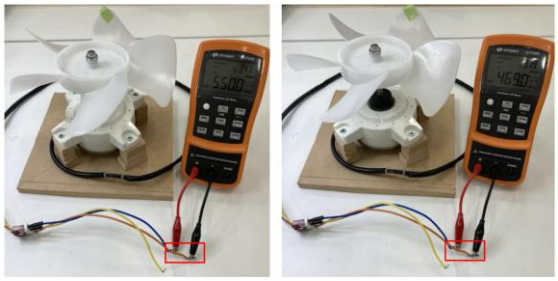


Figure 35 Measure Line to Line stator inductance (2Ld, 2Lq)

5.2.3 Motor poles number (p)

This parameter represents the number of magnetic poles in a full mechanical cycle. There is one electrical cycle for every pair of magnetic poles.

This parameter can be identified by counting the positive and negative peaks in the motor back EMF waveform over a full mechanical revolution.

The following figure shows the line-to-line voltage waveform generated by the back EMF of the 4 pole IPM motor. There are 2 peaks and 2 valleys (2 sinusoidal shaped cycles) in one mechanical cycle.

Motor parameter identification

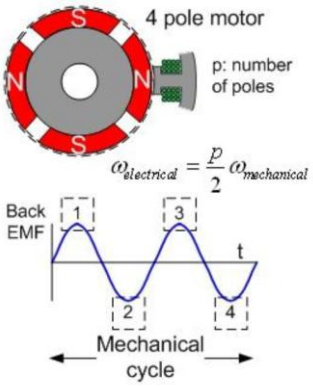


Figure 36 Measurement method of motor poles number

5.2.3.1 Measurement procedure

The measurement procedure of motor poles number is as follows,

1. Connect 2 phases to oscilloscope voltage probe, and leave third phase open
2. Move the motor by hand at a constant speed and make one mechanical revolution, and record the waveform by oscilloscope
3. Count the peaks of the sinusoid

6 Startup method tuning

Different start-up methods are used in sensorless FOC. In this chapter, tuning of start-up method parameters are described.

6.1 Open-loop V/F control

Open-loop V/F control, also referred to as scalar control or constant Volt/Hz control, is one of the methods used for driving PMSMs. The control method behind this approach is preserving a constant voltage-to-frequency ratio corresponding to the target synchronous speed. Rotor angle generated and voltage calculation are done from speed command. There is no speed or current feedback involved.

This control method serves as a startup technique for closed-loop sensorless control systems. During initial startup, back EMF signals are unavailable for determining rotor position, so the motor operates in open-loop Volt-Hz mode. Once the motor reaches adequate speed and the observer can detect EMF, the system transitions to closed-loop control.

6.1.1 V/F parameters

Two parameters define the V/F voltage command Vmin(“params[x].ctrl.volt.v_min”) and K(“params[x].ctrl.volt.v_to_f_ratio”) as shown in Figure 37.

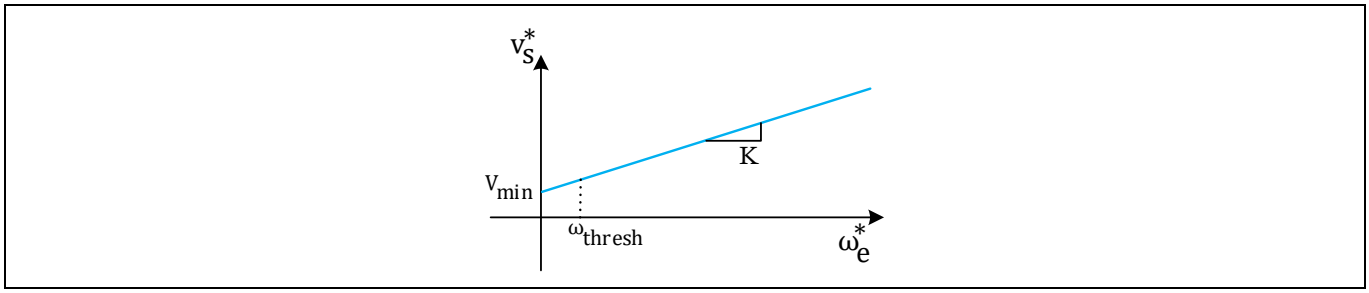


Figure 37 Commanded voltage vs command speed in V/f control

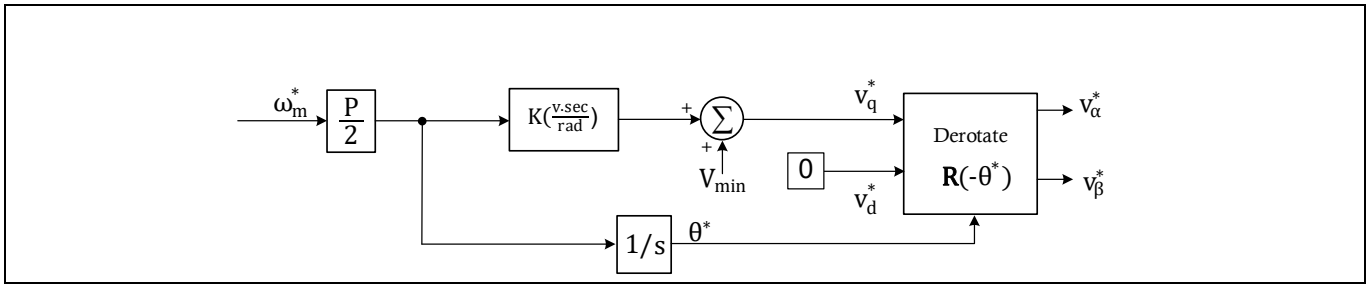


Figure 38 Open-loop V/F control block diagram

Startup method tuning

The list of parameters for the Open-loop V/F startup method are mentioned in Table 8

Table 8 Open-loop V/F control related parameters

Parameter Name	Unit	Description
params[x].ctrl.volt.v_min	V	Voltage offset - minimum voltage needed to overcome motor resistance and generate sufficient torque for startup
params[x].ctrl.volt.v_to_f_ratio	V/(Ra/sec-elec)	Ratio between voltage to frequency value
Params[x].ctrl.volt.w_thresh.elec	Ra/sec-elec	Speed threshold value to start the V/F control when system in bootstrap state
Params[x].ctrl.volt.w_hyst.elec	Ra/sec-elec	Speed threshold value to move from open loop V/F control to bootstrap state

Open-loop constant V/F control parameters are calculated using the following equations.

$$\text{params.ctrl.volt.v\_min} = \frac{(\text{params.motor.r} * \text{params.motor.i\_cont})}{k}$$
$$k: 3 \sim 10$$

$$\text{params.ctrl.volt.v\_to\_f\_ratio} = \frac{\text{params.motor.v\_nom}}{\text{params.motor.w\_nom.elec}}$$

The V_min value should be added at low stator frequency to overcome the stator resistance drop and is calculated based on motor resistance and continuous current value (typically V_min = I_cont × R_stator × safety_factor). The v_to_f_ratio is calculated from motor nominal voltage and nominal frequency/Speed. These are combined in the relationship V_output = V_min + (V_to_f_ratio × Speed) to provide adequate starting torque and maintain proper flux control across the entire operating speed range. When V_min and V_to_f_ratio values are configured optimally, it should not result in too much variation in current for different speed range in the open loop V/F mode.

If the motor current is too high during startup, the V_min value is too high and should be reduced, and if the motor current increases gradually when increasing the motor speed, the V_to_f_ratio value should be reduced to maintain constant current operation throughout the speed range.

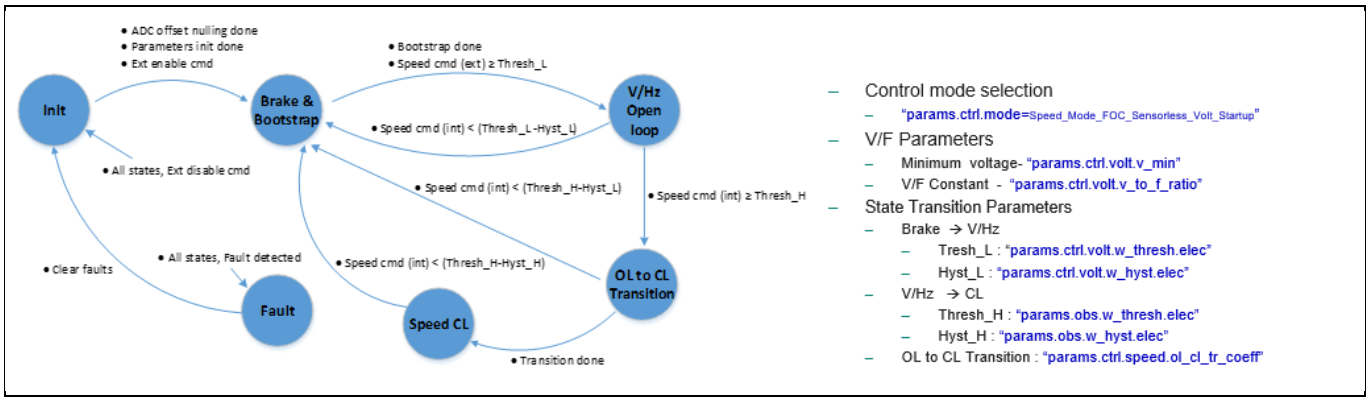


Figure 39 State machine for Speed_Mode_FOC_Sensorless_Volt_Startup

Startup method tuning

6.1.2 Troubleshooting

- Motor is not running in open loop V/F mode:
  - V/F control is sensitive to the control parameters v_min and v_to_f_ratio. If v_min and/or v_to_f_ratio are set too low, the motor will not start spinning and will shake instead. In this case, increase v_min and/or v_to_f_ratio if the motor is not starting or stops after reaching a certain speed.
  - The v_min and/or v_to_f_ratio parameters should be increased if the load is increased and decreased if the load is reduced.
  - Speed ramp rate may be high for the styme, so reduce the ramp rate.
  - Typically, motor phase current will be constant across the complete speed range in V/F control mode.
- Overcurrent triggered while running the motor in V/F mode:
  - If v_min and/or v_to_f_ratio are set too high, the motor will start but there will be an overcurrent fault. Reduce the V/F parameters or check the over the current threshold value.

6.1.3 How to configure parameters

V/F parameters can be configured using the ModusToolbox™ motor control code example or the ModusToolbox™ Motor Suite GUI.

Configuration of V/F parameter using ModusToolbox™ motor control code example in ParamConfig.h file, *\configuration\motor-ctrl-lib-config\ParamConfig.h*

```
/*********Voltage Controller*********/
#define MOTOR_CTRL_VOLT_STARTUP_THRESH          (MOTOR_NORM_SPEED*0.05f)     /*[RPM], startup threshold*/
#define MOTOR_CTRL_VOLT_VF_OFFSET               (0.15f)                      /*[V], V/F ramp voltage offset*/
#define MOTOR_CTRL_VOLT_VF_RATIO                (7.5E-3f)                    /*[V/(Rad/secz], V/f ramp slope*/

```

Figure 40 V/F parameter configuration using ModusToolbox™ code example

Attention: When updating the motor and load parameter make sure the following macro in this file is set true "#define PARAMS_ALWAYS_OVERWRITE (true)"

Configuration of current control parameter using ModusToolbox™ Motor Suite GUI

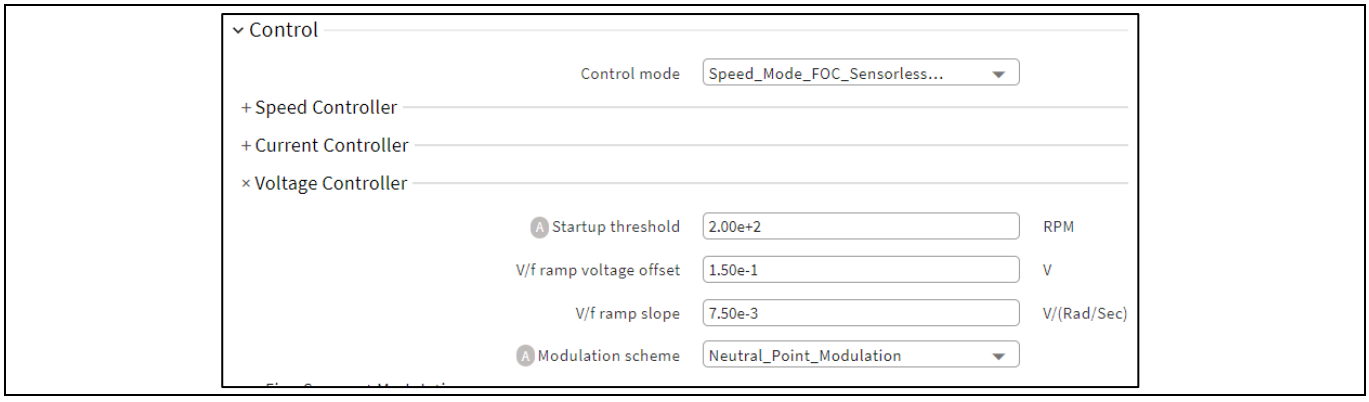


Figure 41 V/F parameter configuration in ModusToolbox™ Motor Suite GUI

Startup method tuning

6.2 Open-loop I/F control

In open-loop I/F control, the current magnitude is regulated using a closed-loop controller with current feedback, while the angle is determined in open-loop fashion based on the commanded speed. The current command value i_q^{r*} (“params[x].ctrl.curr.i_cmd_ol”) sets the magnitude of current injected into the motor for open-loop I/F control. Since this control mode uses motor phase current for current regulation, accurate phase current measurement and proper current loop bandwidth configuration are critical.

During the startup when the back-EMF information is not available for rotor position estimation, the motor can run in open loop I/F mode. Once the motor speed goes up and back-EMF can be estimated by the observer, the closed loop kicks in and the motor exits I/F control.

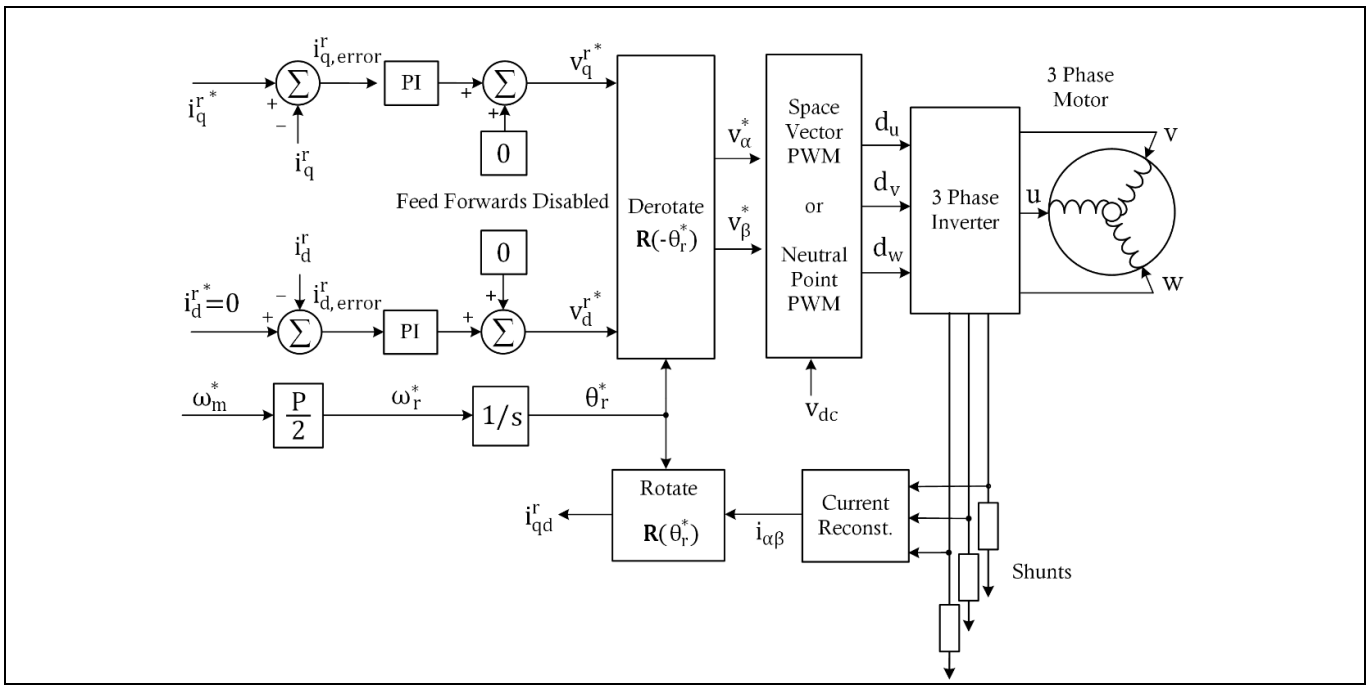


Figure 42 Open-loop I/F control block diagram

6.2.1 I/F parameters

The list of parameters for the open-loop I/F startup method are mentioned in Table 9.

Table 9 Open-Loop I/F control related parameters

Parameter name	Unit	Description
params[x].ctrl.curr.i_cmd_ol	A	Current command value in I/F control
Params[x].ctrl.volt.w_thresh.elec	Ra/sec-elec	Speed threshold value to start the V/F control when system in bootstrap state
Params[x].ctrl.volt.w_hyst.elec	Ra/sec-elec	Speed threshold value to move from open loop V/F control to bootstrap state

Startup method tuning

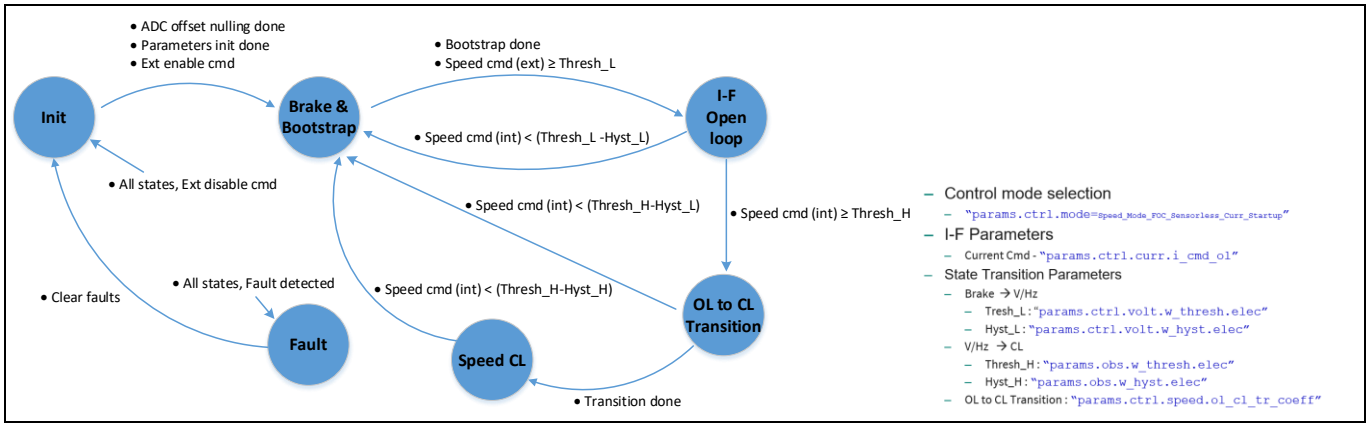


Figure 43 State machine for Speed_Mode_FOC_Sensorless_Curr_Startup

6.2.2 Troubleshooting

- Motor is not running in open loop I/F mode
  - The angle between the rotor flux and the stator current vector is determined by the load. It is important to set i_cmd_ol correctly because it determines the maximum torque the motor can deliver (when the angle between rotor flux and stator current is 90° in SPMs). The actual torque is dictated by the load if it is below the maximum torque achievable by "i_cmd_ol." If the actual torque tries to go above the maximum achievable torque, the motor will lose synchronization and will stall.
  - If "i_cmd_ol" is set too low for that target speed or loading condition, the motor will not start spinning and will shake instead. In this case, increase "i_cmd_ol."
  - If the motor is not running even after adjusting the current command, the speed ramp rate may be too high for the system, so reduce the ramp rate. Also adjust the current loop bandwidth.
- Overcurrent triggered while running the motor in V/F mode
  - In case of overcurrent fault, reduce the "i_cmd_ol" command and/ or reduce the speed ramp rate or adjust the current loop bandwidth.

Tuning approach:

- Start with lower current, slower ramp rate
- Gradually increase until smooth operation is achieved
- Monitor for overcurrent faults or stalling
- Fine-tune based on actual performance requirements

Startup method tuning

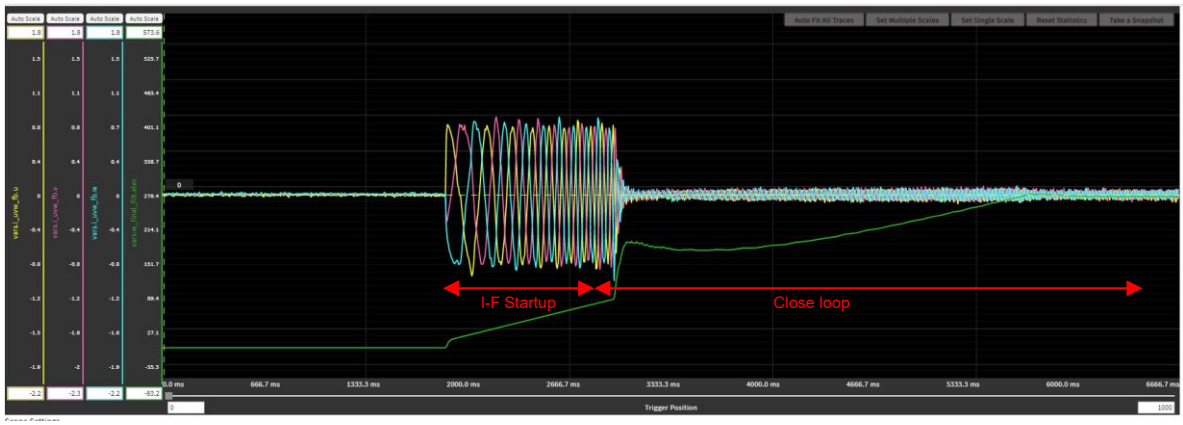


Figure 44 Phase current in Speed_Mode_FOC_Sensorless_Curr_Startup

Note: Open-loop I/F in contrast to open-loop V/F control requires current sensors and more complex controller implementation.

6.2.3 How to configure parameters

Configure the open loop current command value in ParamConfig.c file, \configuration\motor-ctrl-lib-config\ParamConfig.h. Parameter name : “params[x].ctrl.curr.i_cmd_ol,” default value is 40% of “params[x].motor.i_cont”.

7 Control loop tuning

7.1 Current controller

The d and q current commands are used as an input for the current controllers, and the output would be the voltage references. The voltage references v_d^{r*} and v_q^{r*} are eventually applied to the motor using the inverter to control the current.

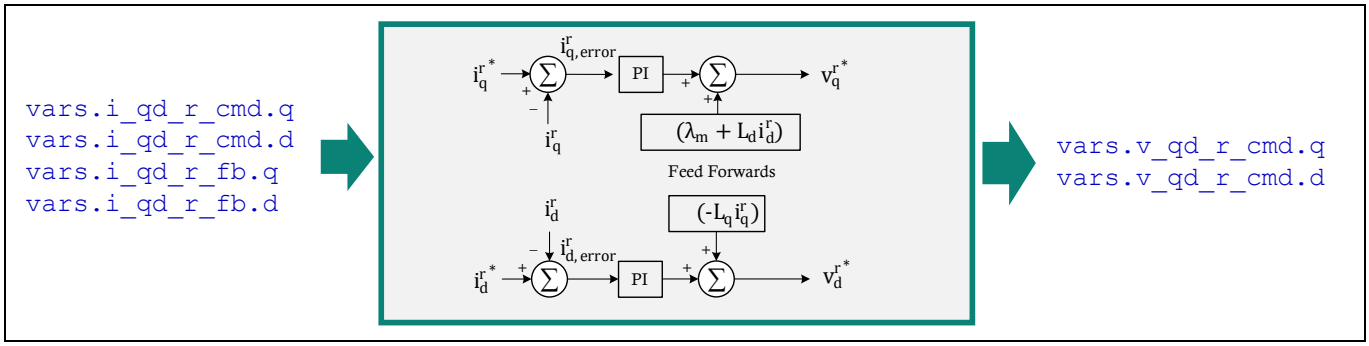


Figure 45 Current controller

Current controller's K_p , K_i and feedforward terms are directly derived from motor electrical parameters. Pole-zero cancellation technique is used to find the PI controller integral and proportional gain.

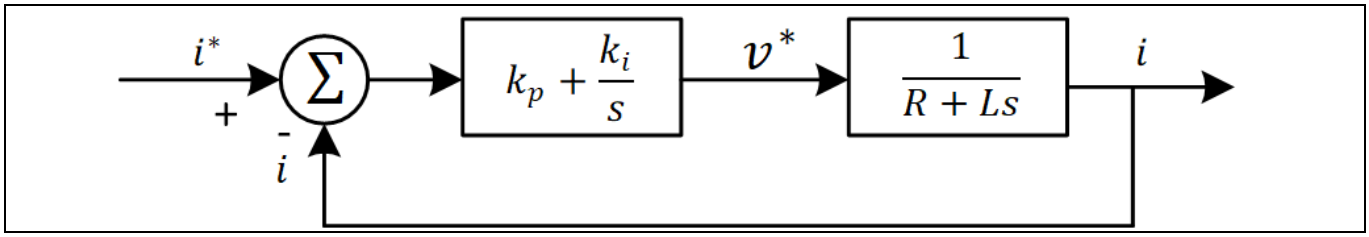


Figure 46 Current controller block diagram

In the closed loop system in Figure 46, it is desired to cancel the pole of the system with the zero of the PI controllers to reduce the order of closed loop system, i.e.

$$\frac{k_p}{k_i} = \frac{L}{R}$$

This will reduce the closed loop transfer function of the system to,

$$H_{cl}(s) = \frac{k_i}{Rs} = \frac{k_p}{Ls}$$

which can be used to calculate the PI controller coefficients for a given system bandwidth ω_c as,

$$k_p = \omega_c L$$

$$k_i = \omega_c R$$

Control loop tuning

7.1.1 Current control parameters

As mentioned in the previous section, current control K_p and K_i values are calculated from motor resistance, inductance, and bandwidth values. Current control K_p and K_i values are calculated in the "PARAMS_InitAutoCalc()" function, which executes during startup (after input parameter initialization) and when input parameters are updated from the Motor Suite GUI. The Kp and Ki value calculations are shown here:

params[x].ctrl.curr.kp.q = params[x].ctrl.curr.bw * params[x].motor.lq

params[x].ctrl.curr.kp.d = params[x].ctrl.curr.bw * params[x].motor.ld

params[x].ctrl.curr.ki.q = params[x].ctrl.curr.bw * params[x].motor.r

params[x].ctrl.curr.ki.d = params[x].ctrl.curr.bw * params[x].motor.r

It is possible to manually enter these values (e.g., in the main.c/paramconf.c file or from the Motor Suite GUI) by disabling auto calculation and setting the params[x].autocal_disable.current_control variable to 1.

K_p and Ki parameter values are used to calculate the control K_p and Ki value that are used in the current control PI function. Control K_p and Ki variable are updated only when the system enters “init” state (CURRENT_CTRL_Init() function)

Ctrl[x].curr.pi_q.kp = params[x].ctrl.curr.kp.q

Ctrl[x].curr.pi_q.ki = params[x].ctrl.curr.ki.q* params[x].sys.samp.ts0

Ctrl[x].curr.pi_d.kp = params[x].ctrl.curr.kp.d

Ctrl[x].curr.pi_d.ki = params[x].ctrl.curr.ki.q* params[x].sys.samp.ts0

Ctrl[x].curr.pi_q.output_min= -params[x].ctrl.curr.v_max.q

Ctrl[x].curr.pi_q.output_max= params[x].ctrl.curr.v_max.q

Ctrl[x].curr.pi_d.output_min= -params[x].ctrl.curr.v_max.d

Ctrl[x].curr.pi_d.output_max= params[x].ctrl.curr.v_max.d

PI output limit is calculated from dc bus nominal voltage value, “params[x].sys.vdc_nom”/√3 and defined in “params[x].ctrl.curr.v_max.q” and “params[x].ctrl.curr.v_max.d”.

The amount of feedforward in the current controller is defined in MOTOR_CTRL_CURRENT_FF_COEFF macro (“params[0].ctrl.curr.ff_coef”). Setting this value to zero will disable the current controller feedforward term. The default value is 100%, meaning the complete feedforward term is added.

Control loop tuning

7.1.1.1 Troubleshooting

Motor Suite GUI – PID Tuner directly update “Ctrl[x].curr.pi_q and Ctrl[x].curr.pi_d” variables and when save the tunned value corresponding parameter values in “params[x].ctrl.curr" are updated

- Motor is not running in Close loop (speed or current) mode: Verification of current control in open loop mode
  - Configure the current control open loop using “params[x].ctrl.mode” parameter, set the appropriate current command using Parameter name: “params[x].ctrl.curr.i_cmd_ol,” default value is 40% of “params[x].motor.i_cont”
  - Run to make sure the open loop and check whether the current is controlled correctly
  - Start with a low current control bandwidth if it is not running correctly
  - Reduce the ramp rate and output limit

Check the system response and adjust the current control bandwidth.

7.1.2 Current controller parameter calculation – Example

The following table depicts the input for the current controller and how the derived parameters are calculated.

Input (These macros are used to initialize parameters in Paramconfig.c file)	MOTOR_CTRL_FASTLOOP_FREQ	15000.0f	[Hz], fast-loop frequency Param name: params[x]. sys.samp.fs0 = MOTOR_CTRL_FASTLOOP_FREQ params[x]. sys.samp.ts0[sec]= 1/ MOTOR_CTRL_FASTLOOP_FREQ = 66.6E-6f
	MOTOR_LQ	670.0E-6f	[H], Stator q-axis inductance, Param name: params[x].motor.lq = MOTOR_LQ
	MOTOR_LD	670.0E-6f	[H], Stator d-axis inductance, Param name: params[x].motor.ld = MOTOR_LD
	MOTOR_R	450.0E-3f	[Ω], Stator resistance, Param name: params[x].motor.r = MOTOR_R
	MOTOR_CTRL_CURRENT_BW	750.0f	[Hz], Current loop bandwidth Param name: params[x].ctrl.curr.bw [Ra/sec]= HZ_TO_RADSEC(MOTOR_CTRL_CURRENT_BW) = TWO_PI*750.0f = 4712.4f
	MOTOR_CTRL_VDC_NOM_VOLT	24.0f	[V], Nominal DC bus voltage Param name: params[x]sys.vdc_nom = MOTOR_CTRL_VDC_NOM_VOLT
Calculated Parameters (calculated in ParamConfig.c and CurrentCtrl.c)	params[x].ctrl.curr.kp.q	3.16f	[V/A], params[x].ctrl.curr.bw * params[x].motor.lq 4712.4f * 670.0E-6f = 3.16f
	params[x].ctrl.curr.kp.d	3.16f	[V/A], params[x].ctrl.curr.bw * params[x].motor.ld 4712.4f * 570.0E-6f = 2.69f
	params[x].ctrl.curr.ki.q	2120.6f	[V/A.(Ra/sec)], params[x].ctrl.curr.bw * params[x].motor.r 4712.4f * 450.0E-3f = 2120.6f
	params[x].ctrl.curr.ki.d	2120.6f	[V/A.(Ra/sec)], params[x].ctrl.curr.bw * params[x].motor.r 4712.4f * 450.0E-3f = 2120.6f
	params[x].ctrl.curr.v_max.q	13.856f	[V], params[x]sys.vdc_nom /√3 = 13.856f
	params[x].ctrl.curr.v_max.d	13.856f	[V], params[x]sys.vdc_nom /√3= 13.856f
	Ctrl[x].curr.pi_q.kp	3.16f	[V/A], params[x].ctrl.curr.kp.q
	Ctrl[x].curr.pi_d.kp	3.16f	[V/A], params[x].ctrl.curr.kp.d
	Ctrl[x].curr.pi_q.ki	141.4E-3f	[V/A.(Ra/sec).sec], params[x].ctrl.curr.ki.q* params[x]. sys.samp.ts0 2120.6f *66.6E-6f = 141.4E-3f
	Ctrl[x].curr.pi_d.ki	141.4E-3f	[V/A.(Ra/sec).sec],params[x].ctrl.curr.ki.d* params[x]. sys.samp.ts0 2120.6f *66.6E-6f = 141.4E-3f
	Ctrl[x].curr.pi_q.output_min	-13.856f	[V], -1* params[x].ctrl.curr.v_max.q
	Ctrl[x].curr.pi_q.output_max	13.856f	[V],params[x].ctrl.curr.v_max.q
	Ctrl[x].curr.pi_d.output_min	-13.856f	[V],-1* params[x].ctrl.curr.v_max.d
	Ctrl[x].curr.pi_d.output_max	13.856f	[V], params[x].ctrl.curr.v_max.d

Control loop tuning

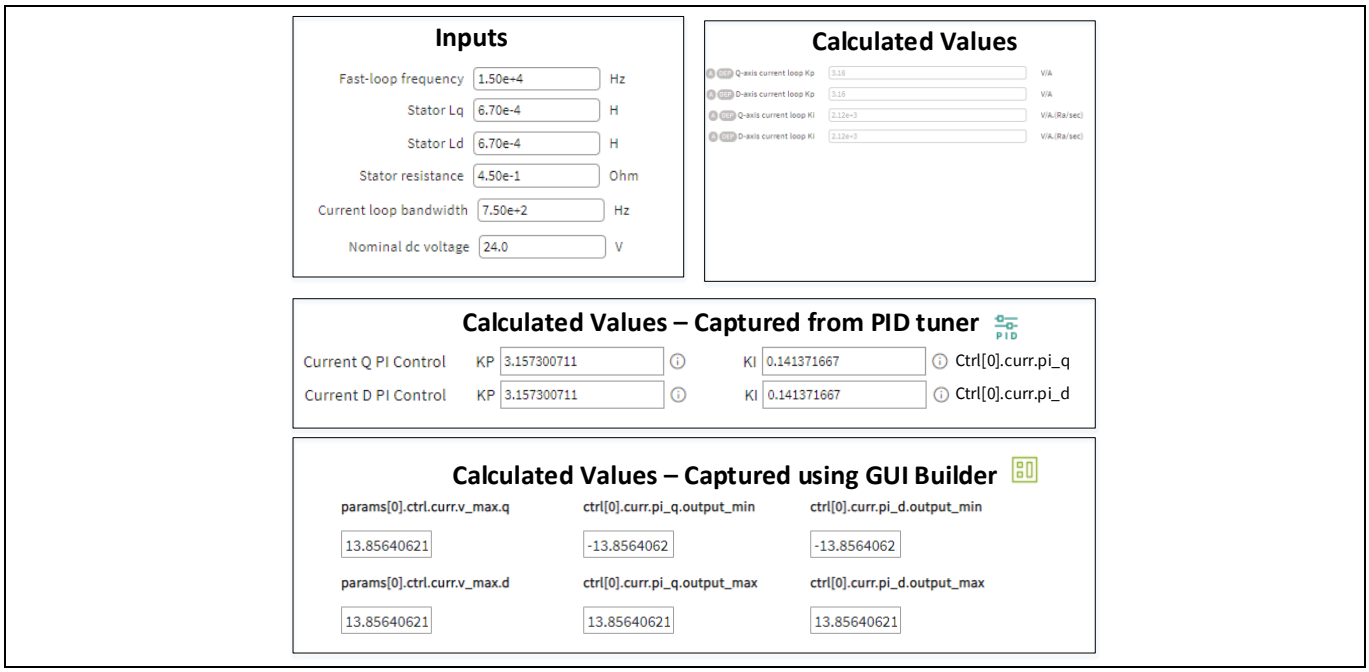


Figure 47 Current controller parameter default values captured from Motor Suite GUI

7.1.3 How to configure parameters

Current control parameters can be configured using the ModusToolbox™ motor control code example or the ModusToolbox™ Motor Suite GUI.

Configuration of current control parameter using ModusToolbox™ motor control code example in ParamConfig.h file, *\configuration\motor-ctrl-lib-config\ParamConfig.h*

```
144 /*********Current Controller*********/
145 #define MOTOR_CTRL_CURRENT_BW                (750.0f)                /*[Hz], Current loop bandwidth*/
146 #define MOTOR_CTRL_CURRENT_FF_COEFF          (100.0f)                /*[%], Current loop feed-forward coefficient*/
147 #define MOTOR_CTRL_CURRENT_STARTUP_THRESH    (MOTOR_CURRENT_CONT*0.15f)   /*[A], Current control startup threshold*/
148
```

Figure 48 Current control parameter configuration using ModusToolbox™ code example

Attention: When updating the current control parameter make sure the following macro in this file is set true "#define PARAMS_ALWAYS_OVERWRITE (true)"

Configuration of current control parameter using ModusToolbox™ Motor suite

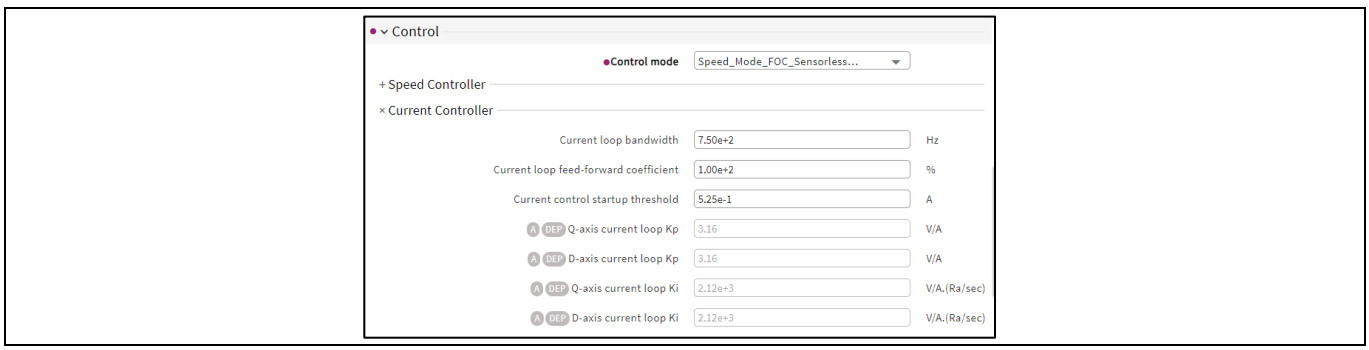


Figure 49 Current control parameter configuration in ModusToolbox™ Motor Suite GUI

Control loop tuning

7.1.4 Update the current control parameter directly

Current control k_p and k_i parameters can be directly modified in the ParamConfig.c file by editing the PARAMS_InitAutoCalc() function.

```
726     // Current Control Parameters:
727     if(!params_ptr->autocal_disable.current_control) /*Execute calculation if this bit is not set*/
728     {
729       params_ptr->ctrl.curr.kp.q = params_ptr->ctrl.curr.bw * params_ptr->motor.lq; // [V/A]
730       params_ptr->ctrl.curr.kp.d = params_ptr->ctrl.curr.bw * params_ptr->motor.ld; // [V/A]
731       params_ptr->ctrl.curr.ki.q = params_ptr->ctrl.curr.bw * params_ptr->motor.r; // [V/A.(Ra/sec)]
732       params_ptr->ctrl.curr.ki.d = params_ptr->ctrl.curr.bw * params_ptr->motor.r; // [V/A.(Ra/sec)]
733     } // if !current_control
```

```
726     // Current Control Parameters:
727     if(!params_ptr->autocal_disable.current_control) /*Execute calculation if this bit is not set*/
728     {
729       params_ptr->ctrl.curr.kp.q = 3.16f; // [V/A]
730       params_ptr->ctrl.curr.kp.d =3.16f; // [V/A]
731       params_ptr->ctrl.curr.ki.q = 2120.6f; // [V/A.(Ra/sec)]
732       params_ptr->ctrl.curr.ki.d = 2120.6; // [V/A.(Ra/sec)]
733     } // if !current_control
```

Figure 50 Speed control parameter direct updated

Also, current control k_p *and* k_i values can be directly configuration using the Motor Suite GUI – PID Tuner that is described in 7.3 PID Tuner

7.2 Speed controller

The speed command is used as an input for the speed controller, and the output would be the current references.

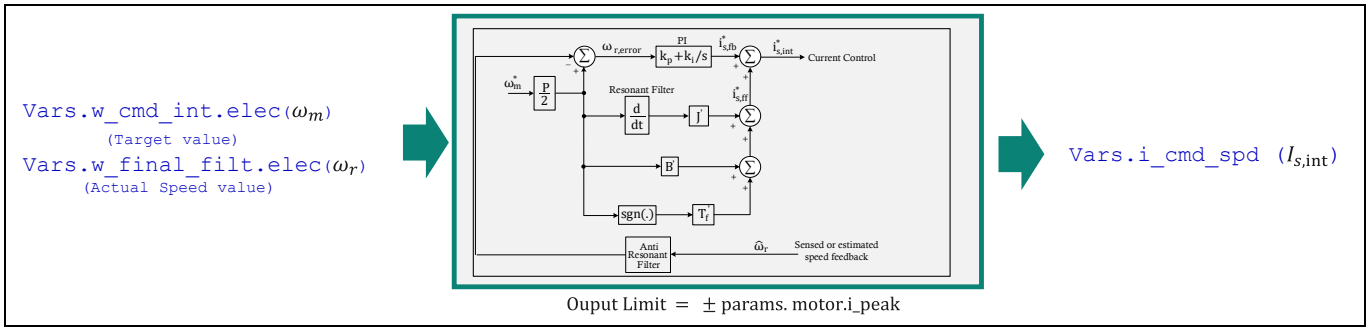


Figure 51 Speed controller

The parameters of the speed controller are significantly impacted by the mechanical load. The mathematical model of the mechanical load driven by the motor and electrical drive system is illustrated in Figure 52 . This model includes T_f, representing coulombic friction, B as the coefficient of viscous friction, and J denoting the inertia. When utilizing the software to operate any motor, it is crucial to accurately measure or estimate these three parameters. The speed controller's k_p, k_i, and feedforward terms are directly derived from these parameters.

Control loop tuning

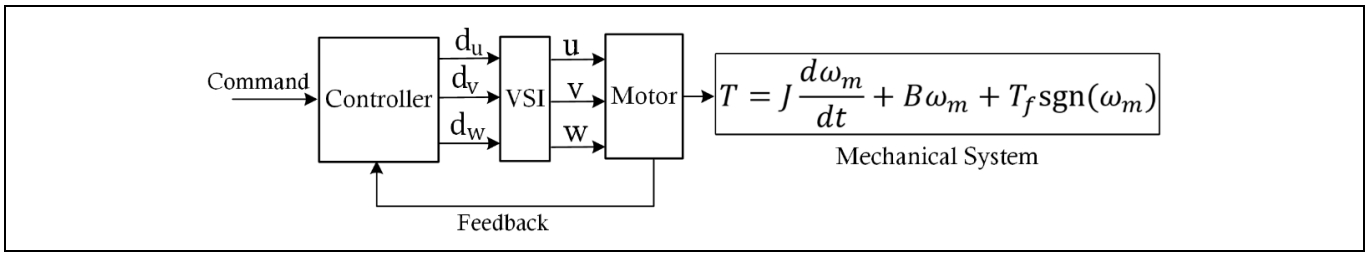


Figure 52 Mechanical load run by a motor-drive system

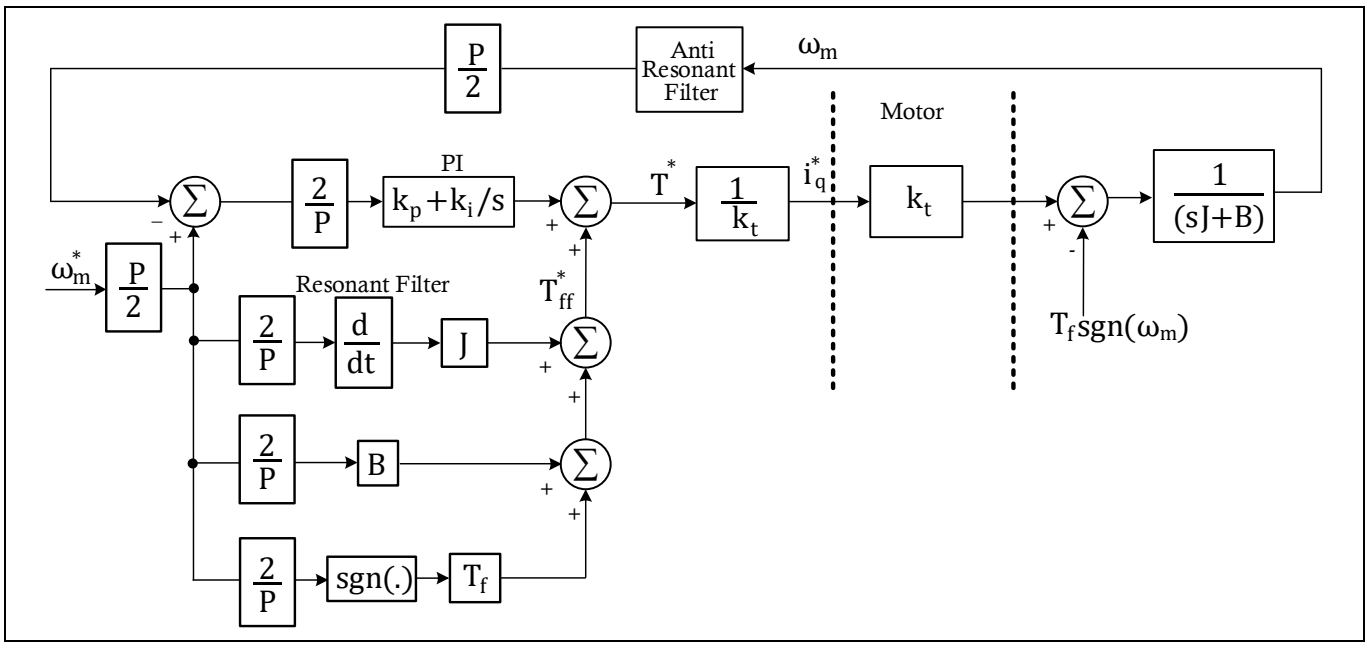


Figure 53 RFO speed loop block diagram before rescaling the parameters

To derive the value of proportional, integral, and feedforward terms of speed controller, the speed loop block diagram along with a simple model of motor and mechanical load will be used as shown in Figure 53 where:

$$k_t \approx \left(\frac{3}{2}\right)\left(\frac{P}{2}\right)\lambda_m$$

Pole-zero cancellation technique is used to find the PI controller integral and proportional gain. By having

$$\frac{k_i}{k_p} = \frac{B}{J}$$

Control zero will cancel the mechanical load's pole. The PI controller coefficients are also proportional to speed loop bandwidth as shown here:

$$k_i \propto B\omega_{BW}$$

$$k_p \propto J\omega_{BW}$$

Control loop tuning

The proportional and integral gains are determined after being rescaled to account for the motor parameters k_t and P shown in Figure 53

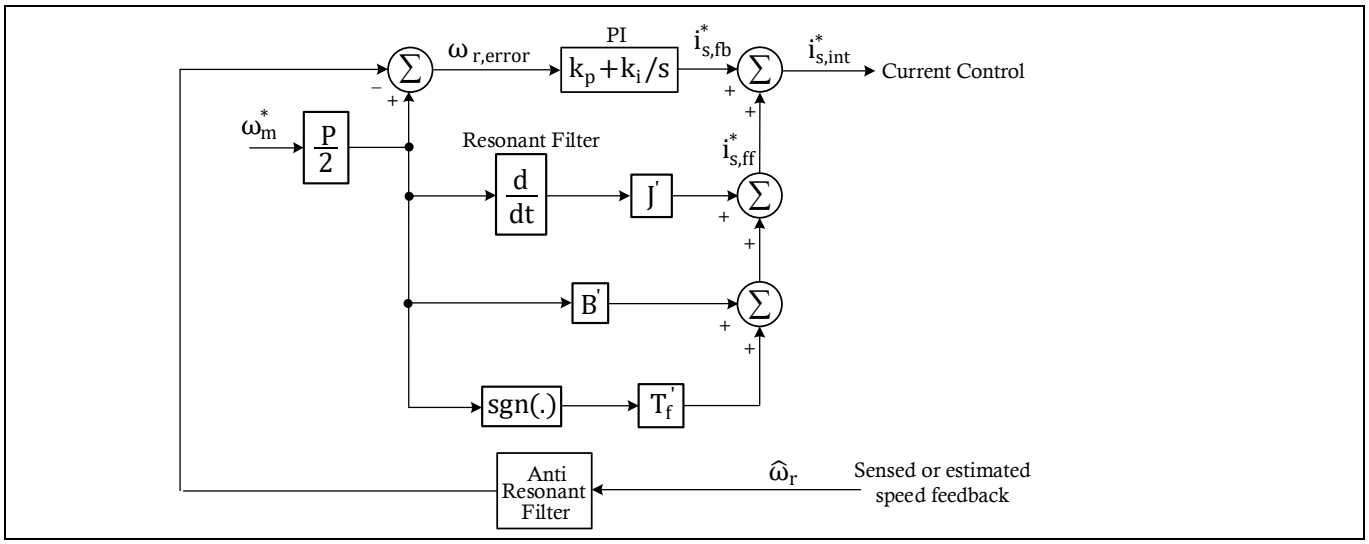


Figure 54 RFO speed loop block diagram as implemented

The speed contoller parameters after rescaling are as follows:

$$k_i = \left(\frac{1}{k_t}\right)\left(\frac{2}{P}\right)B\omega_{BW} = \left(\frac{8}{3}\right)\left(\frac{1}{P^2}\right)\left(\frac{1}{\lambda_m}\right)B\omega_{BW}$$

$$k_p = \left(\frac{1}{k_t}\right)\left(\frac{2}{P}\right)J\omega_{BW} = \left(\frac{8}{3}\right)\left(\frac{1}{P^2}\right)\left(\frac{1}{\lambda_m}\right)J\omega_{BW}$$

The feedforward terms can also be updated as depicted in the following manner:

$$B' = \left(\frac{1}{k_t}\right)\left(\frac{2}{P}\right)B = \left(\frac{8}{3}\right)\left(\frac{1}{P^2}\right)\left(\frac{1}{\lambda_m}\right)B$$

$$J' = \left(\frac{1}{k_t}\right)\left(\frac{2}{P}\right)J = \left(\frac{8}{3}\right)\left(\frac{1}{P^2}\right)\left(\frac{1}{\lambda_m}\right)J$$

$$T_f' = \left(\frac{1}{k_t}\right)T_f = \left(\frac{4}{3}\right)\left(\frac{1}{P}\right)\left(\frac{1}{\lambda_m}\right)T_f$$

The feedforward terms in the speed loop are affected by both mechanical load and motor parameters. These three feedforward terms play a role in enhancing the dynamic performance of the speed loop. The inertia term utilizes a second-order resonant filter to estimate the acceleration, aiming to mitigate the potential impact of noise that could arise if a direct derivation method had been employed.

Control loop tuning

7.2.1 Speed control parameters

As mentioned in the previous section, speed control Kp , Ki and feedforward coefficient values are calculated from motor, load parameter, and bandwidth values. Speed control Kp, Ki and feedforward values are calculated in the "PARAMS_InitAutoCalc()" function, which executes during startup (after input parameter initialization) and when input parameters are updated from the Motor Suite GUI. The Kp and Ki value calculations are shown here:

params[x].ctrl.speed.kp	=	((8.0f / 3.0f) / (POW_TWO(params[x].motor.P) * params[x].motor.lam)) * params[x].mech.inertia * params[x].ctrl.speed.bw
params[x].ctrl.speed.ki	=	((8.0f / 3.0f) / (POW_TWO(params[x].motor.P) * params[x].motor.lam)) * params[x].mech.viscous * params[x].ctrl.speed.bw * params[x].ctrl.speed.ki_multiple
params[x].ctrl.speed.ff_k_inertia	=	((8.0f / 3.0f) / (POW_TWO(params[x].motor.P) * params[x].motor.lam)) * params[x].mech.inertia
params[x].ctrl.speed.ff_k_viscous	=	((8.0f / 3.0f) / (POW_TWO(params[x].motor.P) * params[x].motor.lam)) * params[x].mech.viscous
params[x].ctrl.speed.ff_k_friction	=	((4.0f / 3.0f) / (params[x].motor.P * params[x].motor.lam)) * params[x].mech.friction

It is possible to manually enter Kp and Ki values (for example, in the main.c/paramconf.c file or from the Motor Suite GUI) by disabling auto calculation and setting the “params[x].autocal_disable.speed_control” variable to 1.

Kp and Ki parameter values are used to calculate the control Kp and Ki value that is used in the current control PI function. Control Kp and Ki variable are updated only when the system enters “init” state (SPEED_CTRL_Init() function)

Ctrl[x].speed.pi.kp = params[x].ctrl.speed.kp

Ctrl[x].speed.pi.ki = params[x].ctrl.speed.ki * params[x].sys.samp.ts1

PI output limit is set from maximum motor current, params[x]. motor.i_peak

Motor Suite GUI – PID Tuner directly update “Ctrl[x].speed.pi” variables and when save the tunned value corresponding parameter values in “params[x].speed.curr” are updated

7.2.1.1 Troubleshooting

- Motor is not running in Close loop mode properly
  - Make sure that current controller is working as per expectation
  - Start with low-speed control bandwidth
  - Reduce the ramp rate and output limit
  - Check the system response

7.2.2 Speed controller parameter calculation – Example

The following table depicts the input for the current controller and how the derived parameters are calculated.

Input (Macros are used to initialize parameters in Paramconfig.c file)	MOTOR_CTRL_FASTLOOP_FREQ	y	[Hz], fast-loop frequency Param name: params[x]. sys.samp.fs0 = MOTOR_CTRL_FASTLOOP_FREQ
	MOTOR_CTRL_FS0_FS1_RATIO	5	[#], Fast-loop to slow-loop frequency ratio Param name: params[x]. sys.samp.fs0_fs1_ratio = MOTOR_CTRL_FS0_FS1_RATIO params[x].sys.samp.ts1[sec] = params[x].sys.samp.fs0_fs1_ratio/ params[x].sys.samp.fs0 = 333.33E-6f
	MOTOR_POLE	8.0f	[#], Motor poles Param name: params[x].P= MOTOR_POLE
	MOTOR_I_AM	6.0E-3f	[Wb], Rotor flux linkage Param name: params[x].lam = MOTOR_I_AM
	MECH_INERTIA	1.1E-5f	[kg.m^2], Inertia Param name: params[x].mech.inertia = MECH_INERTIA
	MECH_VISCOUS	1.2E-5f	[kg.m^2/sec], Viscous Damping Param name: params[x].mech.viscous = MECH_VISCOUS
	MECH_FRICTION	6.0E-3f	[kg.m^2/sec^2], Friction Param name: params[x].mech.friction = MECH_FRICTION
	MOTOR_CTRL_SPEED_BW	15.0f	[Hz], Speed loop bandwidth Param name: params[x].ctrl. speed.bw [Ra/sec]= HZ_TO_RADSEC(MOTOR_CTRL_SPEED_BW) = TWO_PI*15.0f = 94.29f
	MOTOR_CTRL_SPEED_KI_MULTIPLE	10.0f	[#], Ki multiple for speed loop Param name: params[x].ctrl.speed.ki_multiple = MOTOR_CTRL_SPEED_KI_MULTIPLE
Calculated Parameters (calculated in ParamConfig.c and CurrentCtrl.c)	params[x]. ctrl.speed.kp	1.2E-3f	[A/(Ra/sec-elec)], ((8.0f / 3.0f) / (POW_TWO(params[x].motor.P) * params[x].motor.lam)) * params[x].mech.inertia * params[x].ctrl.speed.bw = ((8.0f / 3.0f) / (POW_TWO(8) * 6.0E-3f) * 1.1E-5f *94.29f = 7.2E-3f
	params[x]. ctrl.speed.ki	7.2E-3f	[A/(Ra/sec-elec).(Ra/sec)], ((8.0f / 3.0f) / (POW_TWO(params[x].motor.P) * params[x].motor.lam)) * params[x].mech.viscous * params[x].ctrl.speed.bw * params[x].ctrl.speed.ki_multiple = ((8.0f / 3.0f) / (POW_TWO(8) * 6.0E-3f) * 1.2E-5f *94.29f *10.0f = 7.85E-2f
	params[x].ctrl.speed.ff_k_inertia	7.64E-5f	[A/(Ra/sec-elec).sec], ((8.0f / 3.0f) / (POW_TWO(params[x].motor.P) * params[x].motor.lam)) * params[x].mech.inertia = ((8.0f / 3.0f) / (POW_TWO(8) * 6.0E-3f)) * 1.1E-5f = 7.64E-5f
	params[x].ctrl.speed.ff_k_viscous	8.33E-5f	[A/(Ra/sec-elec)], ((8.0f / 3.0f) / (POW_TWO(params[x].motor.P) * params[x].motor.lam)) * params[x].mech.viscous = ((8.0f / 3.0f) / (POW_TWO(8) * 6.0E-3f)) * 1.2E-5f = 8.33E-5f
	params[x].ctrl.speed.ff_k_friction	4.17E-2f	[A], ((4.0f / 3.0f) / (params[x].motor.P * params[x].motor.lam)) * params[x].mech.friction = ((8.0f / 3.0f) / (POW_TWO(8) * 6.0E-3f)) * 6.0E-3f =4.17E-2f
	Ctrl[x].speed.pi.kp	7.2E-3f	[A/(Ra/sec-elec)], params[x]. ctrl.speed.kp
	Ctrl[x].speed.pi.ki	2.617E-3f	[A/(Ra/sec-elec).(Ra/sec)], params[x].ctrl.speed.ki*params[x].sys.samp.ts1 = 7.85E-2f *333.33E-6f = 26.17E-6f
	Ctrl[x].speed.pi.output_min	-10.80f	[A], -params[x].motor.i_peak
	Ctrl[x].speed.pi.output_max	10.80f	[A], params[x].motor.i_peak

Control loop tuning

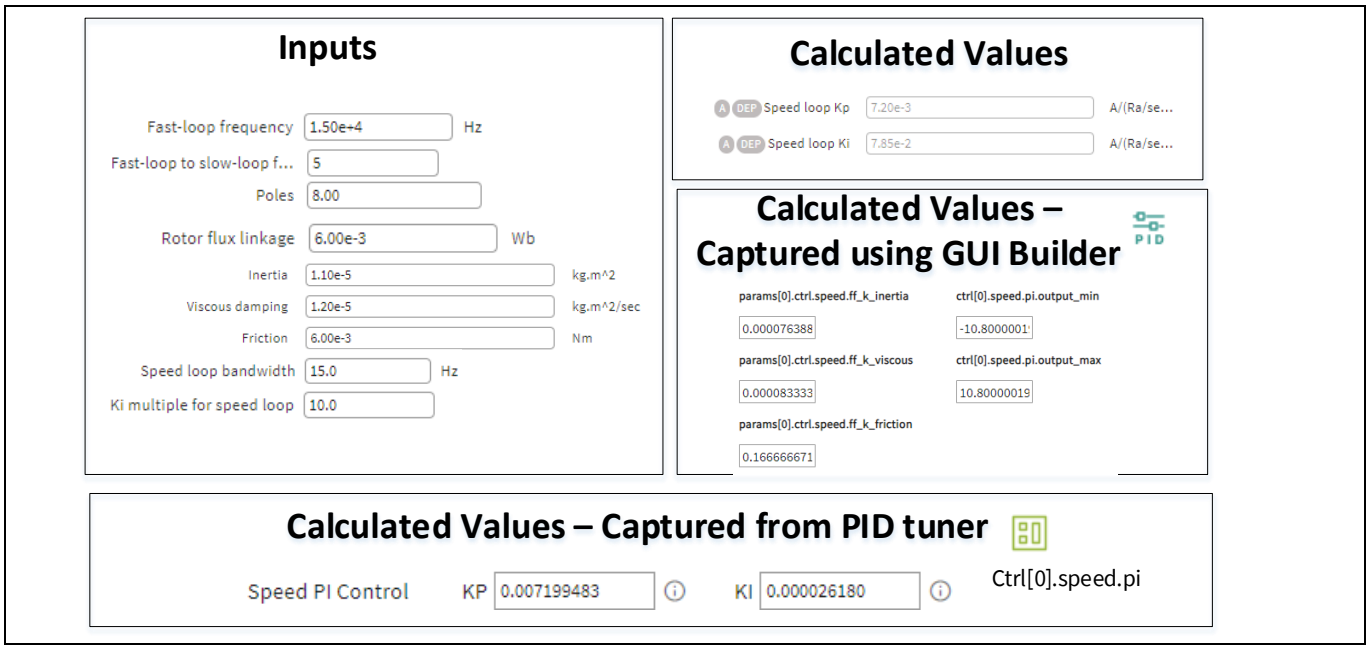


Figure 55 Speed controller parameter default values captured from Motor Suite GUI

7.2.3 Speed open-loop to closed-loop transition

The transition from open loop to closed loop speed control is a critical phase that ensures smooth motor operation without torque disturbances or speed oscillations.

The transition from open loop to closed loop speed control occurs when the motor reaches the observer threshold speed, at which point the system enters the Speed_OL_To_CL state where the motor continues running in open loop mode while the position observer simultaneously starts estimating the rotor angle and speed.

The system remains in this transition state for a predetermined lock time (params[x].obs.lock_time, typically 100-1000 ms) to allow the observer to stabilize and accurately converge on the actual rotor position.

Just before switching to closed loop control, the speed PI controller is pre-initialized using the current open loop current value multiplied by a transition coefficient (MOTOR_CTRL_SPEED_OL_CL_TR_COEFF macro or parameter - params[x].ctrl.speed.ki_multiple, default 100%) to prevent current spikes or speed disturbances during the handover. Once the lock time expires and the speed controller is properly initialized with a realistic output value that matches the current motor operating conditions, the system seamlessly switches from open loop voltage/frequency control to closed loop speed control using observer feedback, ensuring smooth operation without torque bumps, speed oscillations, or performance degradation during this critical transition phase.

- Current Overshoot Issue

During the transition from open loop to closed loop, if there is current overshoot, it indicates that the speed controller is being initialized with too high a value, causing excessive torque demand. In this case reduce MOTOR_CTRL_SPEED_OL_CL_TR_COEFF value 50-80% based on the amount of overshoot.

7.2.4 How to configure parameters

Speed control parameters can be configured using the ModusToolbox™ motor control code example or the ModusToolbox™ Motor Suite GUI.

Configuration of speed control parameter using ModusToolbox™ motor control code example in ParamConfig.h file, *\configuration\motor-ctrl-lib-config\ParamConfig.h*.

```
/*********SPEED Controller*********/
#define MOTOR_CTRL_SPEED_BW                    (15.0f)                 /*[Hz], Speed loop bandwidth*/ //(15.0f)
#define MOTOR_CTRL_SPEED_OL_CL_TR_COEFF        (100.0f)                /*[%], Open-loop to closed-loop transition coefficient*/
#define MOTOR_CTRL_SPEED_KI_MULTIPLE           (10.0f)                 /*[], Ki multiple for speed looop*/

```

Figure 56 Speed control parameter configuration using ModusToolbox™ code example

Configuration of speed control parameter using ModusToolbox™ Motor Suite GUI

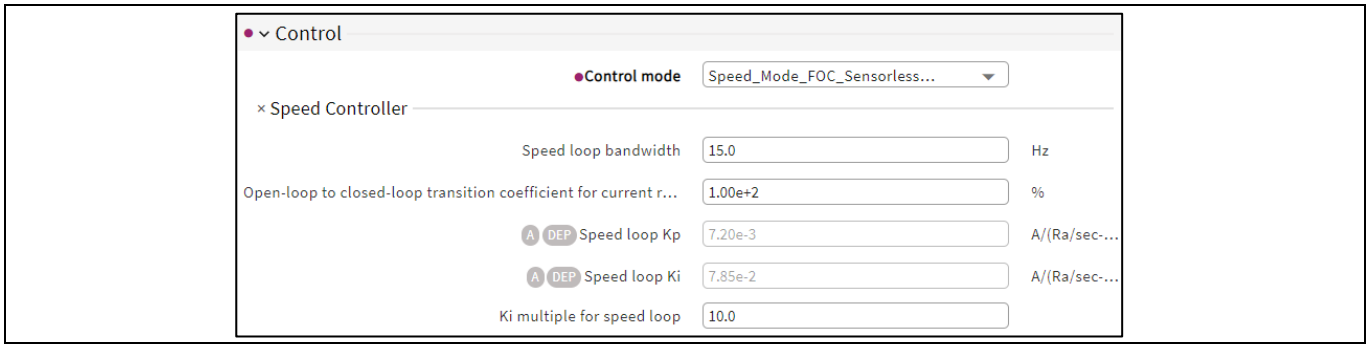


Figure 57 Speed control parameter configuration in ModusToolbox™ Motor Suite GUI

7.2.5 Update the speed control parameter directly

As mentioned in the previous section, speed controller's k_p, k_i, and feedforward terms are directly derived from includes coulombic friction (T_f), viscous friction (B), and inertia (J). This section describes the configuration of speed control parameters and how to configure these parameters directly.

It is possible to directly update the speed control k_p, k_i, and feedforward terms in the *ParamConfig.c* file in PARAMS_InitAutoCalc() function.

```
    // Speed Control Parameters:
#if defined(CTRL_METHOD_RFO)
    if(!params_ptr->autocal_disable.speed_control) /*Skip the calculation if this bit is set*/
    {
      params_ptr->ctrl.speed.kp = ((8.0f / 3.0f) / (POW_TWO(params_ptr->motor.P) * params_ptr->motor.lam)) * params_ptr->mech.inertia * params_ptr->ctrl.speed.bw; // [A/(Ra/sec-elec)]
      params_ptr->ctrl.speed.ki = ((8.0f / 3.0f) / (POW_TWO(params_ptr->motor.P) * params_ptr->motor.lam)) * params_ptr->mech.viscous * params_ptr->ctrl.speed.bw * params_ptr->ctrl.sp
    } // if !speed_control
    params_ptr->ctrl.speed.ff_k_inertia = ((8.0f / 3.0f) / (POW_TWO(params_ptr->motor.P) * params_ptr->motor.lam)) * params_ptr->mech.inertia; // [A/(Ra/sec-elec).sec]
    params_ptr->ctrl.speed.ff_k_viscous = ((8.0f / 3.0f) / (POW_TWO(params_ptr->motor.P) * params_ptr->motor.lam)) * params_ptr->mech.viscous; // [A/(Ra/sec-elec)]
    params_ptr->ctrl.speed.ff_k_friction = ((4.0f / 3.0f) / (params_ptr->motor.P * params_ptr->motor.lam)) * params_ptr->mech.friction; // [A]
```

```
    // Speed Control Parameters:
#if defined(CTRL_METHOD_RFO)
    if(!params_ptr->autocal_disable.speed_control) /*Skip the calculation if this bit is set*/
    {
      params_ptr->ctrl.speed.kp = 1.2E-3f // [A/(Ra/sec-elec)]
      params_ptr->ctrl.speed.ki = 7.2E-3f; // [A/(Ra/sec-elec).(Ra/sec)]
    } // if !speed_control
    params_ptr->ctrl.speed.ff_k_inertia = 7.64E-5f; // [A/(Ra/sec-elec).sec]
    params_ptr->ctrl.speed.ff_k_viscous = 8.33E-5f; // [A/(Ra/sec-elec)]
    params_ptr->ctrl.speed.ff_k_friction = 4.17E-2f; // [A]
```

Figure 58 Speed control parameter direct updated

Control loop tuning

Also speed control k_p *and* k_i values can be directly configuration using the Motor Suite GUI – PID Tuner that is described in 7.3 PID Tuner

7.3 PID Tuner

Motor Suite GUI supports a PID tuner to adjust speed control and/or current control Kp and Ki values directly during runtime to optimize system performance. These values will override the auto-calculated Kp and Ki values that are calculated based on bandwidth, motor, and load parameters. Tuned values from the PID tuner can be saved into the main Motor Suite GUI Configurator. Once these values are available in the Configurator, they can be written to the target or exported as parameter files in .h or .csv format for integration in code example or reporting.

The PID Tuner can be launched from Motor Suite GUI Test Bench using the toolbar or from the menu bar (Tools → PID Tuner).

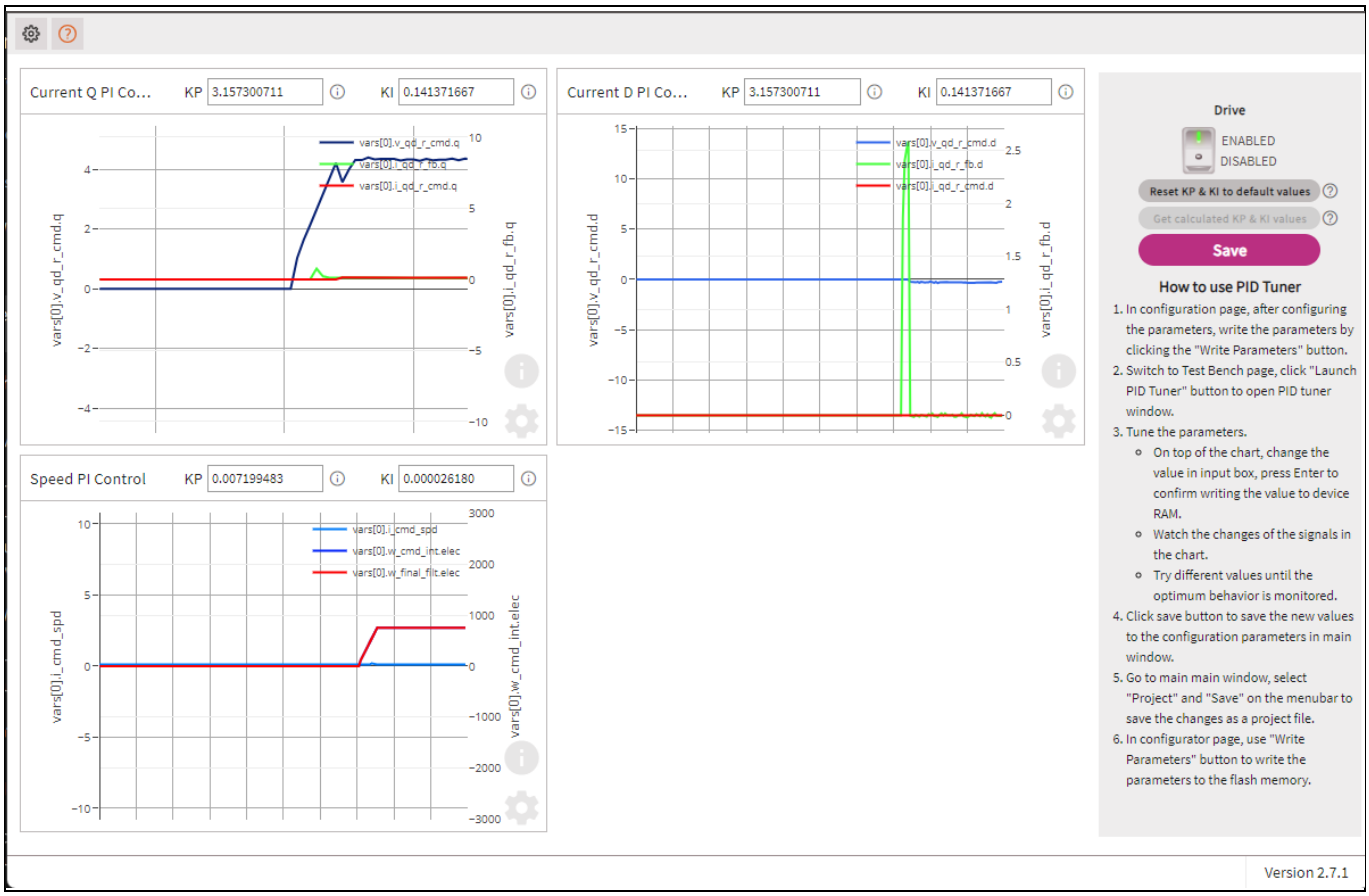


Figure 59 PID Tuner

Ctrl[x] Kp and Ctrl[x] Ki variables can be directly adjusted from the PID tuner based on system response (raise time, overshoot, steady state error etc.). When pressing the save button in the PID tuner, Ctrl[x] variables are converted to params[x] format and the params[x] variables are updated in Motor Suite GUI Configurator. Ctrl[x] and params[x] mapping is mentioned in Current control (7.1.1) and Speed control (7.2.1).

8 How to configure a new board and motor

In the Motor Control Library, default parameter values are configured for the "DB42M03" motor + KIT_PSC3M5_CC2_V2 + EVAL_24V_250W. This chapter describes how to configure Motor Control Library parameters for a new motor or when changing the power board (no change in ADC and PWM pins) or different control boards.

8.1 Configuration for new motor

It is necessary to configure all new motor/load-related parameters in the ModusToolbox™ code example or in the ModusToolbox™ Motor Suite GUI. Refer: Section 3.1 How to configure motor and load

Motor profiler can be used to find the motor and load parameters. Before starting the profiler, configure motor and mechanical parameters using the new motor's nameplate, datasheet, or best-known values in the Motor Suite GUI configurator. Refer: Section 5.1.4 How to run a motor profiler using Motor Suite GUI

Parameter	Macro	DB42M03	DB42S03	Unit	Description
Motor Poles	MOTOR_POLE	8	8	-	Motor poles
Q-axis Inductance	MOTOR_LQ	0.00067	0.001196	H	Stator q-axis inductance
D-axis Inductance	MOTOR_LD	0.00067	0.001196	H	Stator d-axis inductance
Rotor Flux Linkage	MOTOR_I_AM	0.006	0.005	Wb	Rotor flux linkage
Stator Resistance	MOTOR_R	0.45	0.843	Ohm	Stator resistance
Maximum Torque	MOTOR_TORQUE_MAX	0.39	0.19	Nm	Maximum torque
Peak Current	MOTOR_CURRENT_PEAK	10.8	5.4	A	Peak current rating
Continuous Current	MOTOR_CURRENT_CONT	3.5	1.79	A	Continuous current rating
Max D-axis Current	MOTOR_ID_MAX	1.75	0.75	A	Maximum d-axis current
Motor Voltage	MOTOR_VOLTAGE	24	24	V	Motor voltage
Nominal Speed	MOTOR_NORM_SPEED	4000	4000	RPM	Nominal speed
Maximum Speed	MOTOR_MAX_SPEED	6000	6000	RPM	Maximum no load speed
Inertia	MECH_INERTIA	0.000011	0.0000027	$kg{\cdot}m^2$	Inertia
Viscous Damping	MECH_VISCOUS	0.000012	0.0000037	$kg{\cdot}m^2/sec$	Viscous Damping
Friction	MECH_FRICTION	0.006	0.0015	$kg{\cdot}m^2/sec^2$	Friction
V/F Voltage Offset	MOTOR_CTRL_VOLT_VF_OFFSET	0.15	0.15	V	V/F ramp voltage offset
V/F Voltage Ratio	MOTOR_CTRL_VOLT_VF_RATIO	0.0075	0.0075	V/(rad/sec)	V/F ramp slope

Figure 60 Example- Motor parameter configuration for “DB42S03” motor

8.2 Configuration for new power board

In case of using any different power board from the default, it is required to configure voltage, current, and temperature measurement configurations.

8.2.1 Voltage and current measurement

Voltage and current measurement-related configuration, Refer Section 3.2 How to configure voltage and current measurement parameter. Configure the ADC-related macros based on the new power board. Example required ADC macro configuration for REF_80VDC_3.5KW board.

When configuring ADC-related macros for the REF_80VDC_3.5KW power board, several keyboard-specific parameters must be updated to match the hardware.

The ADC_VREF_GAIN macro should be set to 1 because this board uses a direct 3.3V reference without any voltage level shifter circuitry.

The ADC_CS_SHUNT_RES macro must be configured to 0.001f to reflect the 1mΩ shunt resistor used for current measurement on this board.

How to configure a new board and motor

The ADC_SCALE_VUVW and ADC_SCALE_VDC macros require updates because this board supports up to 80V operation, that means the voltage divider networks in the measurement circuits are different from default values - these scaling factors must be calculated based on the actual high-side and low-side resistor values in the voltage divider network using the formula R_low /(R_high + R_low) to ensure accurate voltage measurements, and the specific resistor values should be obtained from the board schematic to determine the precise scaling coefficients needed for proper ADC voltage conversion.

	KIT_PSC3M5_CC2_V2+EVAL_24V_250W	REF_80VDC_3.5KW
#define ADC_VREF_GAIN	((5.0f)/(3.3f))	((3.3f)/(3.3f))
#define ADC_CS_OPAMP_GAIN	(12.0f)	(12.0f)
#define ADC_CS_SHUNT_RES	(10.0E-3f)	(1.0E-3f)
#define ADC_SCALE_VUVW	((5.6f)/(56.0f+5.6f))	((4.87f)/(154.0f+4.87f))
#define ADC_SCALE_VDC	((5.6f)/(56.0f+5.6f))	((4.87f)/(154.0f+4.87f))

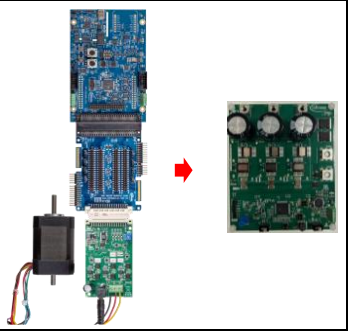


Figure 61 Example- Power board configuration for REF_80VDC_3.5KW board

Some power boards do not use external amplifiers (for example, MADK power boards) and instead use internal gain for current measurement. In this case, the internal gain for current input can be configured using Device Configurator. It is also required to configure the "ADC_CS_OPAM_GAIN" macro using the following relation:

ADC_CS_OPAM_GAIN = Configured_Internal_gain × External_gain_attenuation

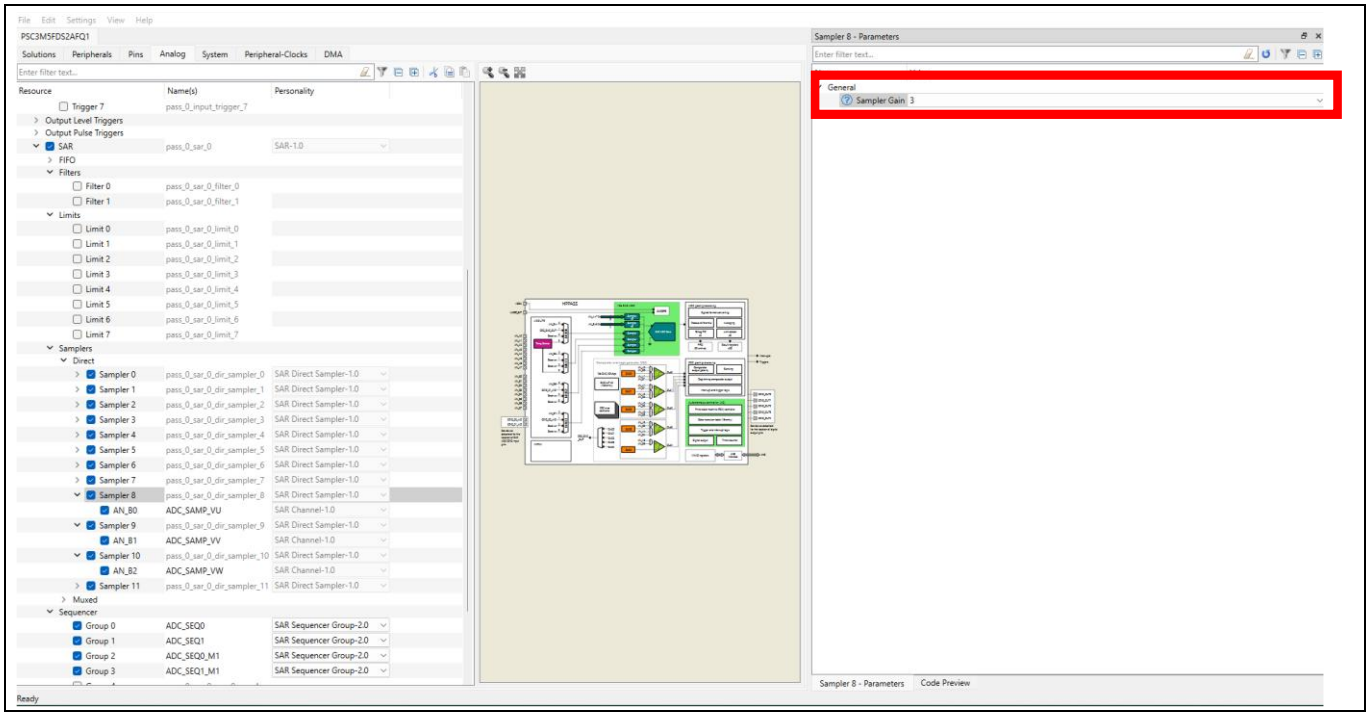


Figure 62 Current measurement -Internal gain configuration using Device Configurator

Note: The current input-offset value is calculated automatically by Motor Control Library when the system is in "init" state, so no need configured current input-offset value.

8.2.2 Temperature measurement

When using a different passive temperature input, it is required to configure the mapping between voltage output and corresponding temperature in the “MotorCtrlHWConfig.c” file.

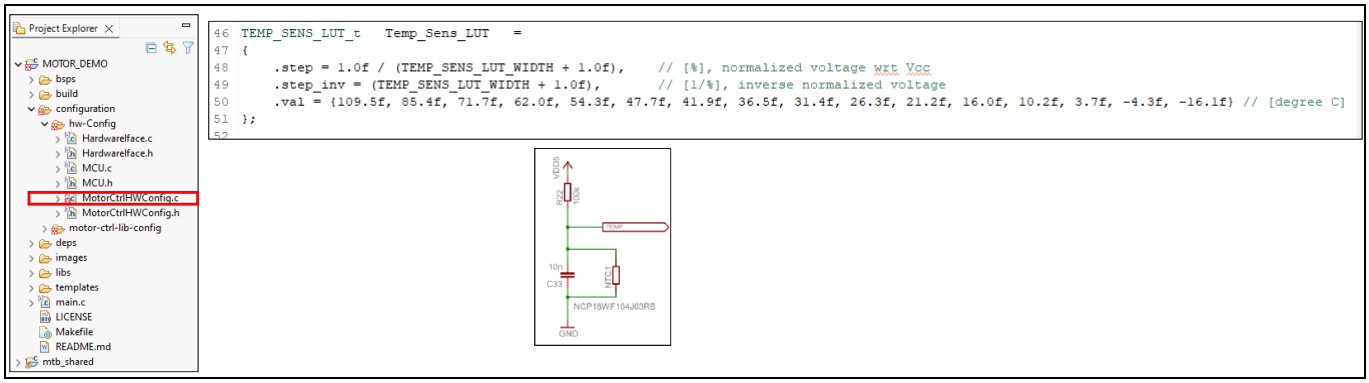


Figure 63 Passive temperature configuration

The temperature sense lookup table size is 16. The mapping of voltage to temperature is defined in this table, where each step corresponds to “Vadcref/17”. The temperature mapping for the default board (EVAL_24V_250W+KIT_PSC3M5_CC2) is mentioned in Figure 64.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
STEP	0.0588	0.1176	0.1765	0.2353	0.2941	0.3529	0.4118	0.4706	0.5294	0.5882	0.6471	0.7059	0.7647	0.8235	0.8824	0.9412
Voltage [V]	0.1941	0.3882	0.5824	0.7765	0.9706	1.1647	1.3588	1.5529	1.7471	1.9412	2.1353	2.3294	2.5235	2.7176	2.9118	3.1059
Temperature [℃]	109.521	85.373	71.748	62.038	54.315	47.750	41.902	36.501	31.357	26.317	21.235	15.946	10.228	3.717	-4.329	-16.064

Figure 64 Temperature mapping for the default board (EVAL_24V_250W+KIT_PSC3M5_CC2)

8.3 Configuration for new control board

When using a controller board with different pinouts (PWM, ADC, Gatekill, direction, fault LED) from the default board, use Device Configurator to change the default pins for the different controller board.

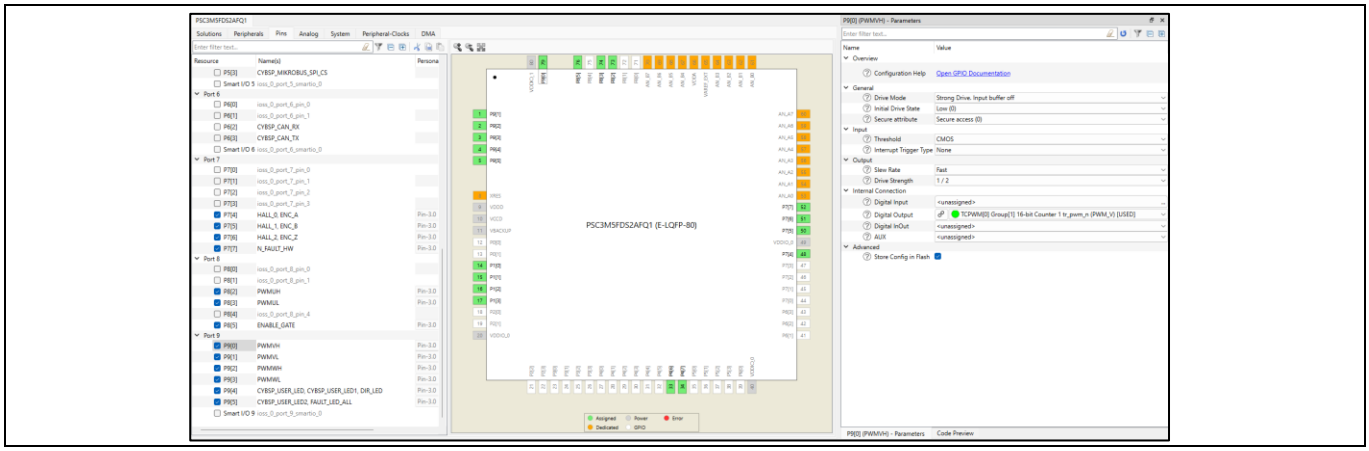


Figure 65 Device Configurator for changing pinout

How to configure a new board and motor

Steps to Modify Pinout Configuration:

1. Open Device Configurator from ModusToolbox™ IDE
2. Modify the existing pinout according to your hardware requirements
3. Save the changes in Device Configurator
4. Configuration files are automatically updated into ModusToolbox™ IDE, code example under the *bsps* folder

8.3.1 Change in ADC pins

In PSOC™ Control C3, Group 0 and Group 1 are used to convert motor control analog input signals. If any changes are made to the Group 0 or Group 1 samplers or any channel pins, it is required to update the Code Example configuration files in “MotorCtrlHWConfig.c”.

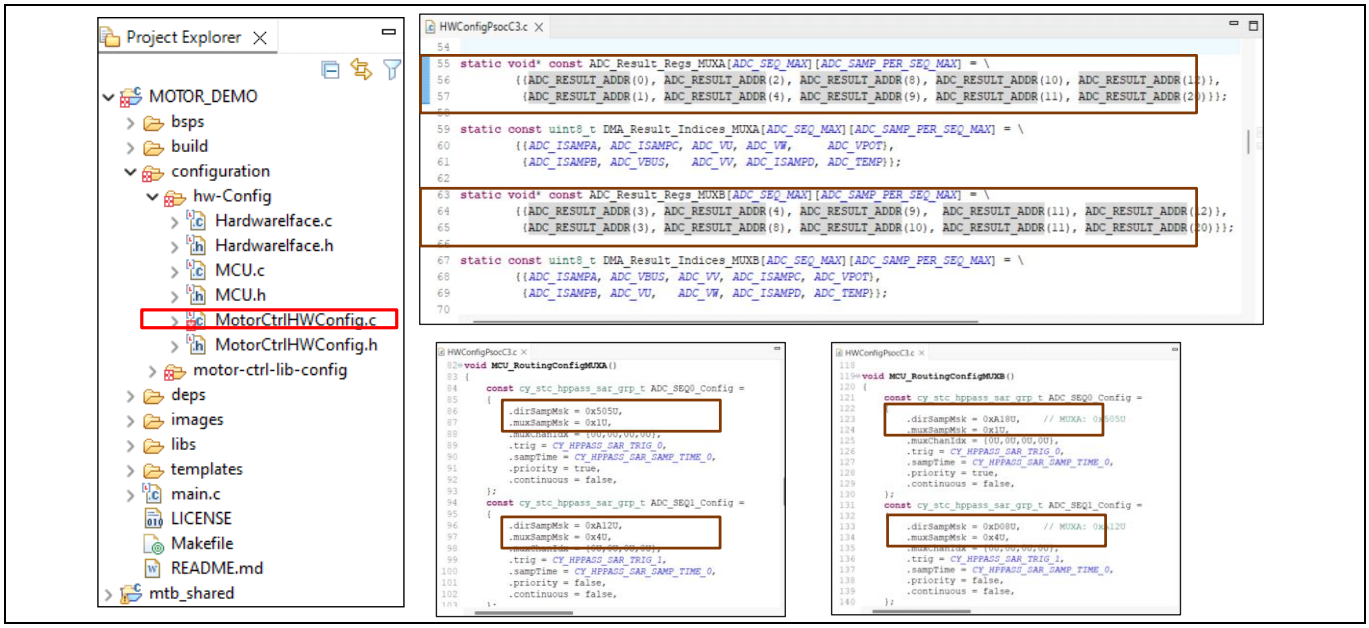


Figure 66 Code change for change in ADC pins

The motor control code uses two software multiplexers for current measurement: MUXA for leg shunt current measurement and MUXB for single shunt current measurement. When remapping ADC channels from their default pin assignments, the corresponding result register numbers must be updated in the "ADC_Result_Regs_MuxA/B" arrays. The system uses "DMA_Result_Indices_MUXA/B" to maintain the mapping between channel functionality and pin indices. To properly update the configuration, locate the specific channel function in the DMA indices mapping and update the corresponding register number in the ADC result registers array. For example, if the Vbus voltage measurement pin is remapped from Channel 4 to Channel 7, both “ADC_Result_Regs_MUXA[1][1]” and “ADC_Result_Regs_MUXB[0][1]” must be updated to ADC_RESULT_ADDR(7) to reflect this change. This ensures that the ADC results are correctly mapped to their respective measurement functions in the motor control algorithm.

If the sequencer Group0 and/or Group1 channel sequence is modified, it is required to change the sampler mask in the "MCU_RoutingConfigMUXA/B()" function. The sampler masks that need to be updated are “ADC_SEQ0/1_Config.dirSampMsk” and “ADC_SEQ0/1_Config.MuxSampMsk” to ensure proper synchronization between the modified channel sequence and the corresponding sampling configuration.

9 GUI to code parameter mapping

This chapter describes how to copy the tunned or configured parameters from the Motor Suite GUI into a motor control code example. After tuning or configuring the parameters using the motor control GUI, export .h file using export as .h file option in the GUI. This chapter explains how to transfer tuned or configured parameters from the Motor Suite GUI into the motor control code example. After completing parameter tuning or configuration using the Motor Suite GUI, the optimized parameters must be exported and integrated into the code example for implementation.

- Step 1: Complete Parameter Tuning
  - Finish tuning/configuration of Motor Control Library parameters using Motor Suite GUI
  - Verifying all parameters is optimized for your motor and load
- Step 2: Export Parameters
  - In the Motor Suite GUI, navigate to the export function
  - Select "Export as .h file" option from the GUI menu
  - Choose the destination folder and filename for the exported header file
  - The GUI will generate a .h file containing all configured parameters
- Step 3: Integration into Code Example
  - Copy the parameter definitions from the exported file
  - Replace the corresponding parameter values in your motor control code example, *ParamConfig.h*
- Step 4: Verification
  - Compile the updated code example
  - Download to target hardware
  - Launch Motor Suite GUI
  - Verify that the motor operates with the new tuned parameters
  - Confirm performance matches the GUI tuning results

This process ensures seamless transfer of optimized parameters from the GUI environment to the Motor Control Library.

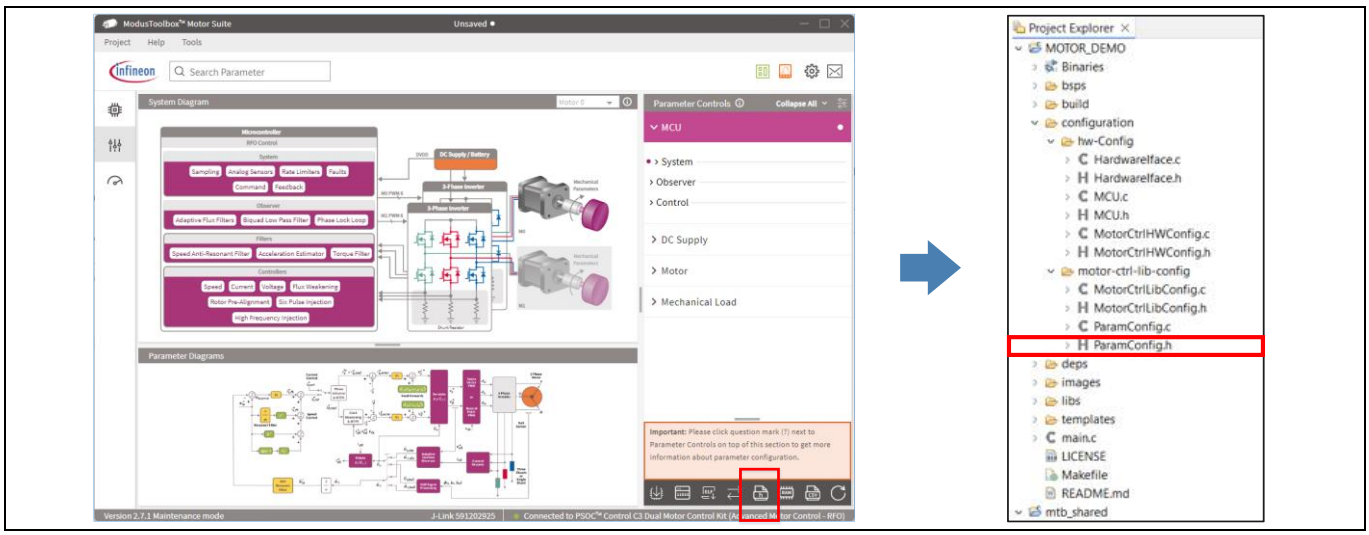


Figure 67 GUI to code parameter mapping

10 Fault handling

The Motor Control Library supports comprehensive set of protections including:

- Under/Over voltage Protection,
- Overcurrent Protection,
- Over Speed Protection,
- Overtemperature Protection, and
- Motor I^2T Protection.

When any fault condition is detected, the corresponding bitfield in the "faults[x].flags.all" variable is set to '1'. All fault flags are latched into "faults[x].flags_latched.all" for persistent fault tracking. If any fault is reported in the "faults[x].flags_latched.all" variable, the state machine moves to the fault state. Once the fault is cleared, the state machine moves to the Init state for system restart.

10.1 Fault response actions

During fault conditions, all switches are turned OFF or a zero vector is applied based on the "faults[x].react_mask" variable configuration. The software supports multiple zero vector methods, with one applied based on the configuration in “params[x].sys.faults.short_method.” The available zero vector options are:

- Low_Side_Short
- High_Side_Short
- Alternate_Short (In one PWM cycle, high-side switches are ON for 50% of the time and low-side switches are ON for the remaining 50% of the time)

This allows flexible fault response strategies depending on the application requirements and hardware protection needs.

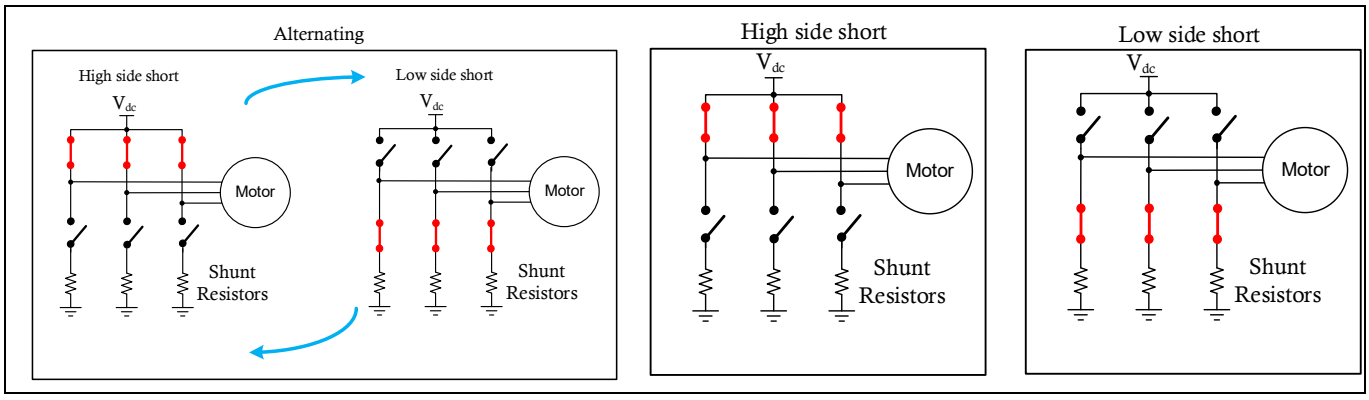


Figure 68 Inverter fault reaction - Zero vector

10.2 Fault/protection summary

Over/Under Voltage Protection

Under/over voltage faults are triggered when the actual DC bus voltage falls below the under-voltage threshold ("params[x].sys.faults.vdc_thresh.min") or exceeds the over voltage threshold ("params[x].sys.faults.vdc_thresh.max") for a defined period ("params.sys.faults.vdc_time"). When an under-voltage condition is detected, the bitfield "faults[x].flags.sw.uv_vdc"[3] is set. When an over voltage condition is detected, the bitfield "faults[x].flags.sw.ov_vdc"[2] is set.

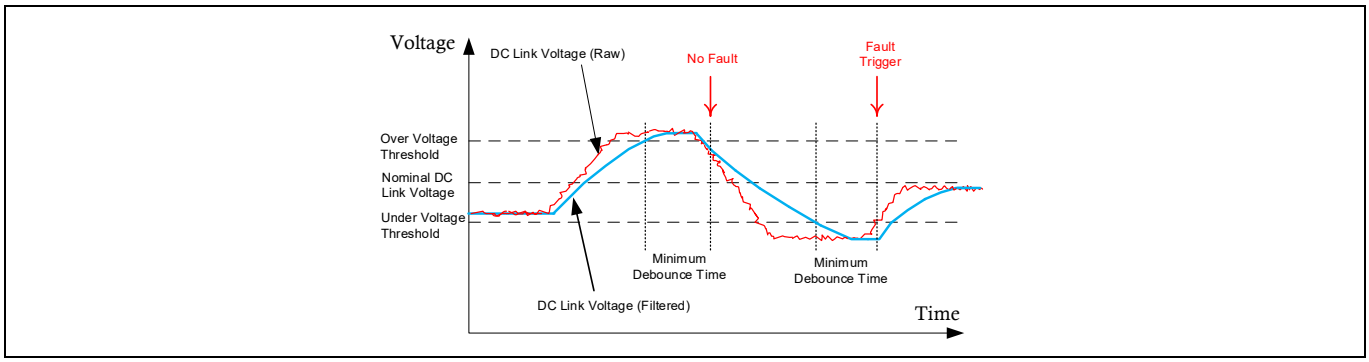


Figure 69 Fault/protection: Voltage protection

Hardware overcurrent protection:

When an overcurrent condition is detected via hardware (Trap), the bitfield "faults[x].flags.hw.cs_ocp" is set for immediate protection response.

Software overcurrent protection:

Overcurrent faults are triggered when motor current exceeds the configured threshold value "faults[x].vars.oc_thresh". When this condition is detected, the bitfield "faults[x].flags.sw.oc"[0] is set.

I²T protection:

When motor current reaches peak value, the current limit in the current controller is automatically reduced to the continuous current value to prevent thermal damage.

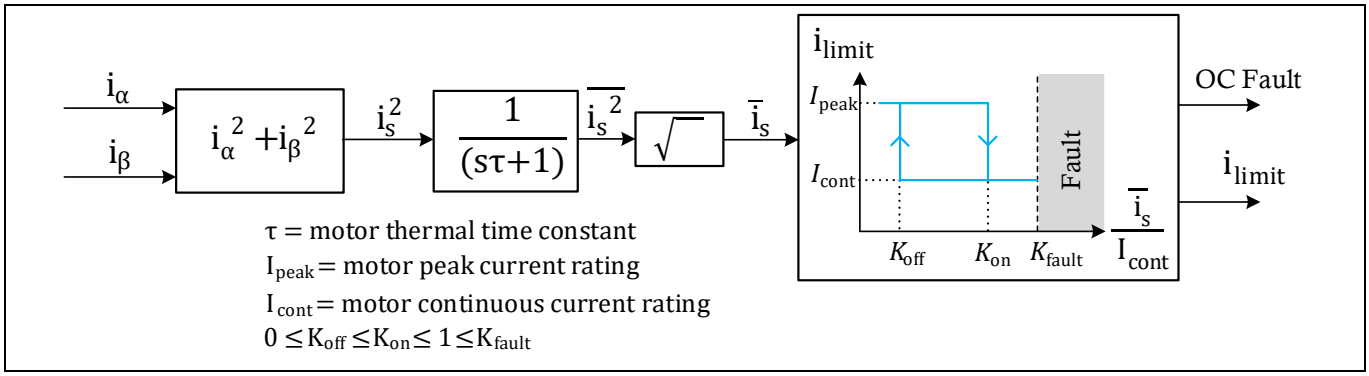


Figure 70 Fault/protection - I²T protection

Fault handling

Over speed protection:

Over speed faults are triggered when motor speed exceeds the configured threshold value "params[x].sys.faults.w_thresh.elec". When detected, the bitfield "faults[x].flags.sw.os"[4] is set.

Overtemperature protection:

Overtemperature faults are triggered when temperature input values exceed the configured threshold value "params[x].sys.faults.temp_ps_thresh". When detected, the bitfield "faults[x].flags.sw.ot_ps"[1] is set.

Refer to the Firmware reference manual for more detailed information.

10.3 Fault clear mechanism

The motor control system provides a software mechanism to clear latched fault conditions through the “sm[x].vars.fault.clr_request” variable. This variable serves as a command interface to reset the fault state machine and clear all latched fault flags. To clear faults from the system, set the “sm[x].vars.fault.clr_request” variable to 1. This action initiates the fault clearing sequence, which resets all latched fault flags stored in “faults.flags_latched.all” and allows the state machine to transition from the fault state back to the init state.

Note: The fault clear request will only be effective if the actual fault condition has been physically resolved. If the underlying fault condition still exists (such as overcurrent, overvoltage, or overtemperature), the fault will be immediately re-triggered after the clear request is processed.

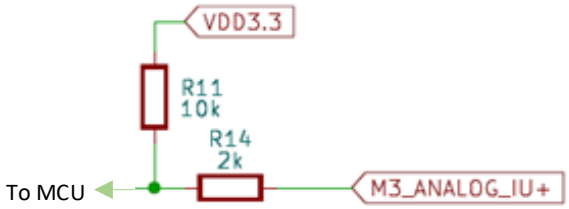
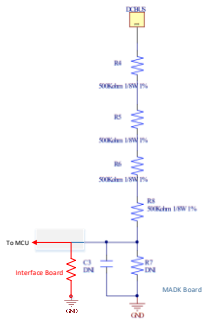
11 MADK power board configuration

This section covers the essential configuration steps required to set up the MADK power board along with PSOC™ C3 CC2 board for motor control applications.

11.1 Hardware used

- KIT_PSC3M5_CC2, PSOC™ Control C3M5 Motor Drive Control Card
- EVAL-M1-CTE620N3, iMOTION™ MADK Evaluation board High voltage
- Dual motor interface card, interface between

Table 10 Board specification input required for Motor Control (MADK+ Dual motor interface card)

Current Shunt Type	Three Shunt
Current Shunt Resistor	30 mΩ
Current Amplifier Gain	Current Gain in MADK board , A_{power} = 1 (No external amplifier in this board) Current Gain in Dual motor interface board A_{inter} = 10kΩ/(10kΩ+2kΩ) = 0.833, In the interface board 10kΩ and 2kΩ resistor used to provide offset for current input  Total External Current gain =A_{power} * A_{inter} = * 0.833 = **0.833** The resistor network provides 0.55V offset (3.3* 2/12), so it is possible to use ADC MCU internal gain up to 3. Another Ex. -If board has external amplifier(8) and no offset network in dual motor interface (removed R11 and R14 = 0Ω), Total external gain = 8*1 = 8
DC Bus Voltage Resistor Divider	High Side Resistor: 2000kΩ (in MADK board) Low Side Resistor: 15KΩ (in Dual motor interface board) Maximum measurable voltage =3.3*(2000+15)/15 = 443.3V 

Note: The current input-offset value is calculated automatically by Motor Control Library when the system is in "init" state, so no need configured current input-offset value.

MADK power board configuration

11.2 Software configuration

11.2.1 Device configuration

Pinout Configuration:

- MADK and Dual motor interface hardware setup pinout for PWM, ADC, and other peripherals matches default software pinout configuration
- No specific pinout changes required

Internal ADC Gain Configuration:

- Use Device Configurator to enable internal gain, set internal gain = 3 for all current input channels

Dual Motor Interface Board Setup:

- Connect appropriate jumpers on Dual Motor Interface Board, ensure proper signal mapping for current inputs, voltage inputs, and temperature inputs to corresponding MCU ADC channels

Verification:

- Verify all signal mappings before system power-up

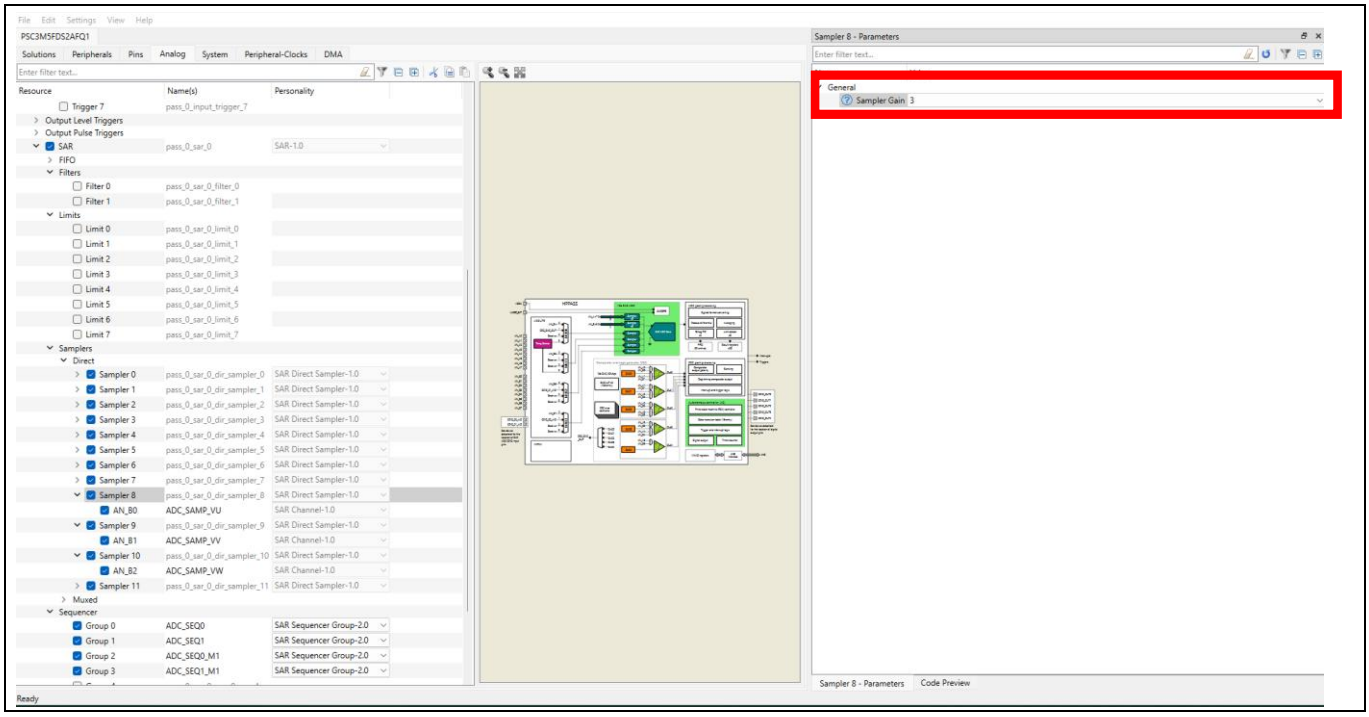


Figure 71 Current measurement -Internal gain configuration using Device Configurator

MADK power board configuration

11.2.2 Parameter configuration

Power board Parameter configuration:

- All the power board-related parameters are defined in configuration/motor-ctrl-lib-config/ParamConfig.h

Table 11 Power board specific parameter

ADC_CS_CURRENT_MEASUREMENT_TYPE	Shunt_Res
ADC_CS_CURRENT_SENSE_POLARITY	LS_Current_Sense
ADC_CS_SHUNT_TYPE	Three_Shunt
ADC_CS_SHUNT_RES	30E-3f [Ω]
ADC_CS_OPAMP_GAIN	Internal Gain * External Gain(or Attenuation) = 3*0.833
ADC_SCALE_VDC	((15.0f)/(2000.0f+15.0f))
ADC_VREF_GAIN	((5.0f)/(3.3f)) if no change made in the controller board ((3.3f)/(3.3f)) if voltage level shifters are removed in the controller board Refer: How to configure voltage and current measurement parameter 3.2
MOTOR_CTRL_VDC_NOM_VOLT	170V

```
/*****************Parameter Configuration for Motor 0************************/
/*****************************************************************************/
/*Power Board configuration*/
#define ADC_VREF_GAIN                           ((5.0f)/(3.3f))             /*[V/V], voltage-reference buffer gain (e.g. scaling 5.0V down to 3.3V)*/

#define ADC_CS_CURRENT_MEASUREMENT_TYPE         (Shunt_Res)                 /*Shunt_resistance "Shunt_res" =0 or active sensor "Active_Sensor" =1*/
#define ADC_CS_CURRENT_SENSE_POLARITY           (LS_Current_Sense)          /*Low side current sense "LS_Current_Sense" =0 or High side current Sense "HS_Current_sense"=1*/

#define ADC_CS_SHUNT_TYPE                       (Three_Shunt)               /*Three_Shunt =0, Single_Shunt =1*/

#define ADC_CS_SHUNT_RES                        (30.0E-3f)                  /*[Ohm], Applicable for Current shunt type of measurement*/
#define ADC_CS_CURRENT_SENSITIVITY              (10.0E-3f)                  /*[V/A], Applicable for active current sense type of measurement*/

#define ADC_CS_OPAMP_GAIN                       (3.0f *0.833f)              /*[V/V], external amplifier gain for current measurement*/

#define ADC_CS_SETTLE_RATIO                     (0.8f)                      /*[], settling ratio used for single-shunt current sampling*/
#define ADC_CS_SS_MIN_SEGMENT_TIME              (3.0E-6f)                   /*[us], Current measurement single shunt minimum measurable window */

#define ADC_SCALE_VUVW                          ((5.6f)/(56.0f+5.6f))       /*[V/V] = [Ohm/Ohm]*/
#define ADC_SCALE_VDC                           ((15.0f)/(2000f+15.0f))      /*[V/V] = [Ohm/Ohm]*/
/*****************************************************************************/
/*****************************************************************************/
/*******DC Supply*******/
#define MOTOR_CTRL_VDC_NOM_VOLT                 (150.0f)                    /*[V], Nominal DC bus voltage*/

/*****************************************************************************/
```

Figure 72 Power board-specific parameter using ModusToolbox™ IDE

Make require change in motor parameter and control parameter. Tuning of V/F or I/F startup method or speed/current control loop refer respectively chapter.

12 Appendix

12.1 Parameter handling

- Parameter initialization is done in “Init” state entry function if parameters are not initialized.
  - If no valid data EEPROM or “PARAMS_ALWAYS_OVERWRITE” macro is set to true, the following functions are called
  - “PARAMS_InitManual()” → All the input parameter variables assignment
  - “PARAMS_InitAutoCalc()” → Calculate all the derived parameters from input parameters
  - All parameter values are stored into EEPROM
  - If EEPROM has valid parameters and the “PARAMS_ALWAYS_OVERWRITE” macro set to false (default value), parameters variables are updated from EEPROM
- Parameter values can be updated from external interface (Ex. GUI) using FcnExeHandler.c functions
  - Derived parameter calculations can be triggered by setting “fcn_exe_handler.req.Auto_Calc_Params” bitfield
  - Store all parameters value from RAM to EEPROM by setting “fcn_exe_handler.req.Flash_Params” bitfield
  - Reset all the module and reinitialize the peripheral by setting “fcn_exe_handler.req.Reset_Modules” bitfield
  - During execution of the above request, the FcnExe handler stops all the peripherals and restarts after handling the request
  - FcnExe Handler function only execute the request when SM is in “init” or “fault” state, the request is ignored if SM is not in “init” state

12.2 State machine handling

Each state in the motor control system has four functions: entry, exit, RunISR0, and RunISR1. The state entry function is called when the state machine enters a state, while the state exit function is called when the state machine exits from a state. The state ISR0 function is called every fast loop, and the state ISR1 function is called every slow loop. State transitions are handled in the slow loop ISR. If a state change is requested, the exit function of the current state is executed, followed by the entry function of the requested state, and finally the current state is set as the requested state.

“sm[x]. current” variable holds the current state machine state.

Appendix

12.3 Initialization and interrupt handling in motor control

Initialization sets up the motor control system through state machine setup, parameter initialization, and configuring hardware peripherals (PWM, ADC, GPIO, DMA, ISR).

Interrupt handling provides real-time control execution using two ISRs in motor control: Fast loop ISR (triggered after all ADC conversions) executed every PWM period or every multiple PWM period based on configuration, and Slow loop ISR (timer-based ISR) executed every fast loop period or every multiple fast loop period based on configuration.

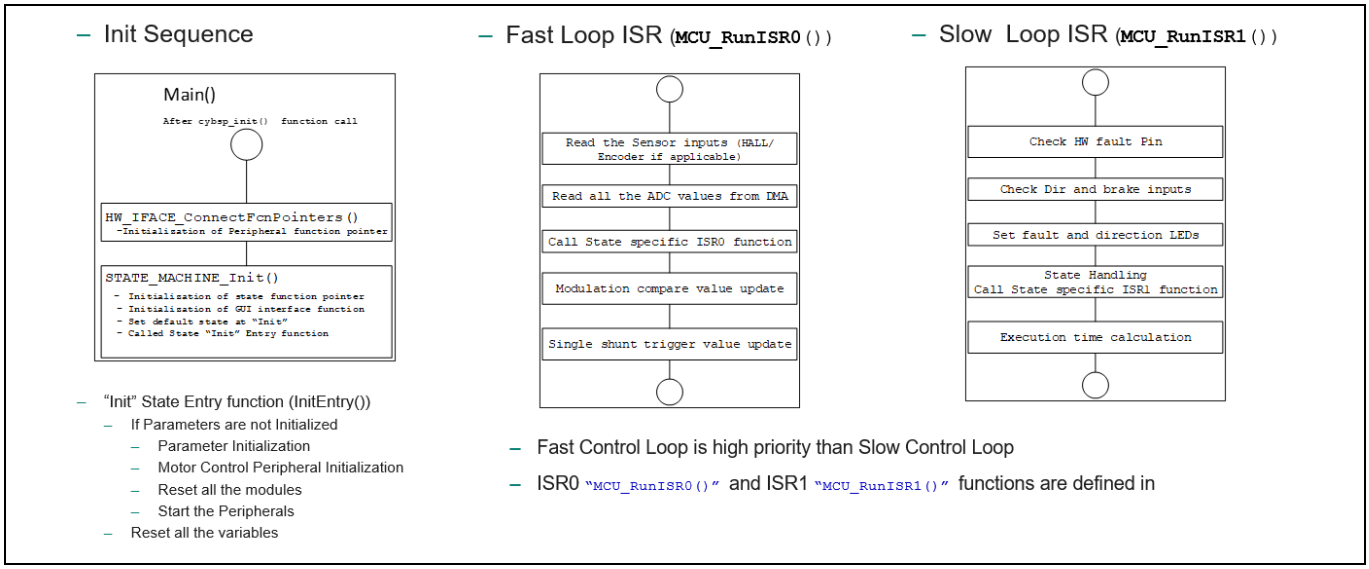


Figure 73 Initialization and interrupt handling in Motor control

12.4 ModusToolbox™ file structure

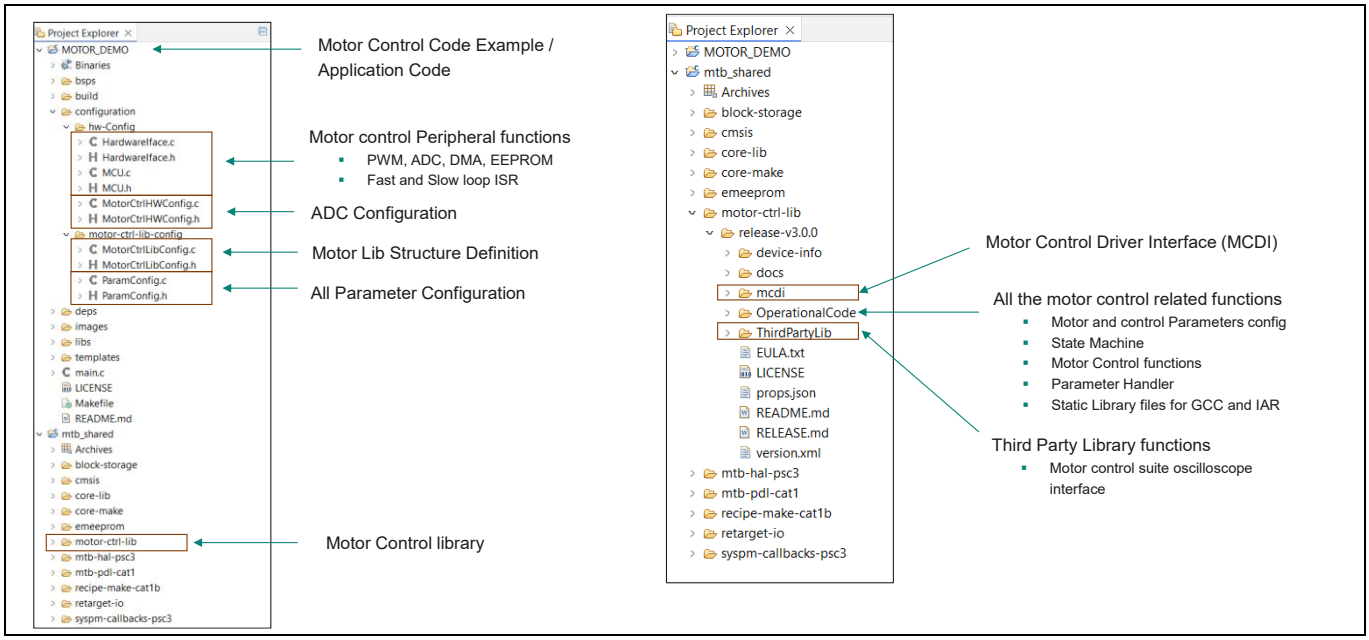


Figure 74 ModusToolbox™ file structure (Motor Control Library V3.0.0)

Appendix

12.5 How to override library function with a user-defined function

This chapter provides instructions for replacing existing Motor Control Library functions with user-defined functions without modifying the core motor library. This approach allows for customization while maintaining library integrity.

- Key benefits
  - Preserve original library functionality
  - Enable system-specific customizations
  - Maintain upgrade compatibility
  - Isolate custom code in application layer
- General process

  Step 1: Exclude Library Function from Build
  - Use the make file to exclude the original library function from the build process by adding it to the ignore list.

  Step 2: Implement User-Defined Function
  - Create a custom implementation in your application code that maintains the same interface as the original function.

  Step 3: Ensure Proper Integration
  - Verify that all output variables and function signatures match the expected library interface.
- Example to replace the Speed control function with user-defined function
  - Modify the make file in ModusToolbox™ to exclude the Speed control file form build
  - CY_IGNORE+=$(wildcard../mtb_shared/motor-ctrl-lib/…/SpeedCtrl.c)
  - Define user-defined functions
    - Copy the "*SpeedCtrl.c*" file from the library and place into the Application code
    - Modify the copied "*SpeedCtrl.c*" file based on system requirements
    - Make sure that Speed Control output variable is updated correctly.

Note: Complete functions can be redefined in a new file without copy "SpeedCtrl.c" file, should not modify the function name, definition (Content) of the function can be modified.

- Functions executed in each state can be modified in "*StateMachine.c*" file. Follow the same step given above to modify the "*StateMachine.c*" file

13 Abbreviations and definitions

Table 12 Abbreviations and definitions

BC	Block Commutation
BLDC	Brushless DC
FOC	Field Oriented Control
FPU	Floating Point Unit
IPM	Interior Permanent Magnet
ISR	Interrupt Service Routine
MCU	Microcontroller Unit
MTPA	Maximum Torque per Amp
MTPV	Maximum Torque Per Volt
PMSM	Permanent Magnet Synchronous Motor
PWM	Pulse Width Modulation
RFO	Rotor frame-oriented Field Oriented control
RRF	Rotating Reference Frame
SFO	Stator frame-oriented Field Oriented control
SRF	Stationary Reference Frame
SM	Surface Mounted
SVM	Space Vector Modulation
TBC	Trapezoidal or block commutation
TC	Trapezoidal Commutation

References

[1] Infineon Technologies AG: PSOC™ Control C3 Documentation; Available online

[2] Infineon Technologies AG: AN238329 - Getting started with PSOC™ Control C3 MCU on ModusToolbox™ software; Available online

[3] Infineon Technologies AG: KIT_PSC3M5_MC1 - PSOC™ Control C3 motor drive card; Available online

[4] Infineon Technologies AG: Motor Control Ecosystem Introduction-Motor Suite; Available online

Revision history

Document revision	Date	Description of changes
**	2025-12-10	Initial release.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Edition 2025-12-10

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference number

002-42330 Rev. **