# 36 W BLDC ceiling fan tuning guide

## REF_XMC13_36W_LV_Ceiling_Fan

## About this document

### Scope and purpose

This tuning guide describes how to fine-tune a 36 W brushless DC (BLDC) ceiling fan based on the current XMC1302 field-oriented control (FOC) single-shunt firmware. It contains configurable parameters you can tune as well as some additional parameters.

All parameters include values you can use as a reference guide.

### Intended audience

This guide is intended for those wanting to use Infineon's 36 W low voltage BLDC ceiling fan reference (REF) solution.

# Table of contents

**About this document** ..... **1**
**Table of contents** ..... **2**
**1 Introduction** ..... **4**
1.1 Key features that need tuning ..... 4
**2 Getting started** ..... **5**
2.1 BMI setting ..... 5
2.2 Motor specification parameters ..... 5
**3 Ceiling fan state machine flow** ..... **6**
**4 Catch-free running** ..... **7**
**5 Brake bootstrap: brake timing** ..... **8**
**6 Pre-positioning** ..... **9**
**7 Rotor initial position detection (IPD)** ..... **10**
**8 FOC closed loop** ..... **11**
8.1 Normal start-up ..... 11
8.2 CFR startup ..... 11
8.3 Remote control (RF/IR) ..... 12
**9 Stop motor** ..... **13**
9.1 Advance stop conditional ..... 13
9.2 Stall detection ..... 13
9.3 Software overcurrent protection ..... 13
9.4 Over/undervoltage protection ..... 13
9.5 Auto-retry ..... 14
9.6 Deep sleep mode ..... 14
9.7 Power Limit ..... 14
**10 Ceiling Fan GUI** ..... **15**
10.1 GUI Setup ..... 15
**11 Overall rough motor tuning process** ..... **17**
11.1 Tuning of the PI controller gains ..... 18
11.1.1 Tuning of the PI parameters ..... 18
**Revision history** ..... **25**
**Disclaimer** ..... **26**

## Table of contents

# 1 Introduction

The ceiling fan board uses Infineon's XMC™ XMC1302 (TSSOP-38) MCU. This firmware project code is based on the 100 W FOC XMC13 software. Several new features have been added to the existing firmware and each one can be fine-tuned to best suit your needs.

## 1.1 Key features that need tuning

The following features in the FOC software can be tuned.

- Catch-free running (CFR)
- Bootstrap brake time
- Pre-positioning
- Initial detection position (IPD)
- RF remote control/IR remote
- Auto-restart
- Stall detection
- Software overcurrent protection (OCP)
- Over/Undervoltage protection (OVP/UVP)
- Deep sleep mode
- Power limit

# 2 Getting started

## 2.1 BMI setting

For the new board with the new XMC1302 MCU, you need to set the boot mode index (BMI). Do not connect the board to power just yet, simply connect the debugger to the board. Go to DAVE™ > BMI setting and set the BMI to (debug) SWD0 (SWDIO = P0.14, SWDCLK = P0.15).

## 2.2 Motor specification parameters

Every new motor will have different specifications. Change the parameters below before attempting to run the motor.

- USER_MOTOR_R_PER_PHASE_OHM 0.9 f
  - Motor resistance per phase in Ω
  - Divide by 2 to get the value for this parameter. Refer to the motor specification datasheet that is being used.
- USER_MOTOR_L_PER_PHASE_uH 4500.0 f
  - Motor inductance per phase in µH
  - Divide by 2 to get the value for this parameter. See the motor specification datasheet.
- USER_MOTOR_POLE_PAIR 4
  - Motor pole pair
  - Take the number of poles and divide by 2 to get the pole pair value. See the motor specification datasheet.

# 3 Ceiling fan state machine flow

The whole tuning guide follows the state machine flow shown below and the features related to it.

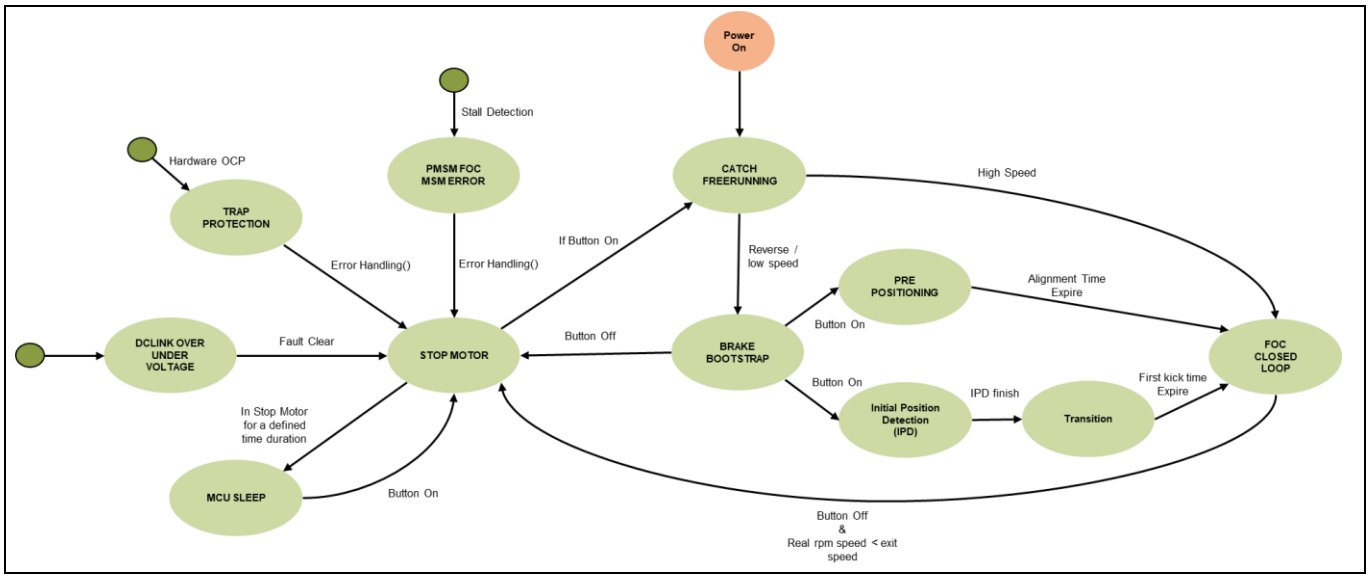


**Figure 1 Ceiling fan state machine**

# 4 Catch-free running

Following are the main parameters to determine whether the fan goes into the normal startup process or into the CFR startup process, depending on the starting fan situation.

- BEMF_THRESHOLD_V 13.0 f
  - Threshold BEMF in V to enter CFR. If BEMF voltage is lower than this value, the fan enters normal startup, otherwise it enters CFR startup.
  - Try 10.0 f and higher. 13.0 f is about 0.23 BEMF volts taken from the oscilloscope.
- CFR_ADC_BEMF_UVW_HIGH_LIMIT 300 U
  - ADC BEMF threshold. If the ADC_BEMF value is higher than this threshold, it is used to determine that the fan motor is spinning at a certain higher speed.
  - Used to determine the different brake timing during the motor state bootstrap brake.

# 5 Brake bootstrap: brake timing

This is where the brake timing is tuned. This brake timing is determined by the CFR state. In the actual situation, during a complete stop, the flag might be swaying between positive and reverse condition. All timings are in ms. Low/high speed is determined by being below/above the ADC BEMF threshold.

- CFR_BRAKE_TIME_3_MS 2500 U
  - This timing is used if CFR is normal reverse at high speed.
- CFR_BRAKE_TIME_2_MS 1000 U
  - This timing is used if CFR is normal reverse at low speed.
- CFR_BRAKE_TIME_1_MS 500 U
  - This timing is used if CFR is normal non-reverse at high speed.
- USER_BOOTSTRAP_PRECHARGE_TIME_MS 200 U
  - The default bootstrap timing.
  - This timing is used if CFR is normal non-reverse at low speed/stop.

# 6 Pre-positioning

This is where you can determine whether to include pre-alignment for the motor startup.

- USER_ROTOR_PREPOSITION_TIME_MS 1500 U
  - Rotor startup pre-alignment time in ms
- VREF_INC_STEP 1 U
  - Step to increase voltage. Default is 4 U until it reaches the pre-alignment voltage set.
- USER_STARTUP_VF_OFFSET_V 2.0 f
  - Pre-alignment voltage; default is 1.0 f.
  - Tune this higher or lower. The lower value might not have any impact. The higher value also might not work, causing unstable startup.

# 7 Rotor initial position detection (IPD)

Here, you can determine whether to include IPD inductive sense method for the motor startup. Six voltage pulses will be injected. The IPD algorithm will estimate the initial position of the rotor with the settings of the phase current on time and decay time.

- USER_ROTOR_IPD_IND_SENSE_PATTERN_ONTIME 0.8f
  - Turn on time of the injected pulse. Time in millisecond (mS).
- USER_ROTOR_IPD_IND_SENSE_CURRENT_DECAY_TIME 1 U
  - Turn off time of the injected pulse. Time in millisecond (mS).

# 8 FOC closed loop

There are different types of startup: normal startup and CFR startup. This is determined by the returned CFR status.

## 8.1 Normal start-up

This is the normal start-up after bootstrap state. For a normal start-up in speed control mode, take note of the following configurations used:

- MY_FOC_CONTROL_SCHEME SPEED_CONTROLLED_DIRECT_FOC
  - For a normal start-up in speed control mode
- USER_SPEED_RAMPUPDOWN_RATE_RPM_PER_S 30U
  - Speed ramp up rate
  - 30U means 30 rpm per second
  - For the motor to ramp up from still to max speed of 350rpm, it takes about 11.7 second.
- Remote_Control ENABLED
  - For the normal use case, Remote_Control is set as enabled so you can control the ceiling fan on or off and the speed upon startup.
- auto_start ENABLED
  - Auto-startup upon power restoration after during power disruption
  - If auto_start is ENABLED, the target motor speed level is based on what was last saved in the user flash area.
  - If auto_start is DISABLED, the target motor speed level is set to Speed level 1.
  - Speed level 1 = 150 rpm, Speed level 2 = 200 rpm, Speed level 3 = 250 rpm, Speed level 4 = 300 rpm and Speed level 5 = 350 rpm

## 8.2 CFR startup

To enable the ceiling fan to go into CFR startup, you needs to first configure the following MACRO to ENABLED:

#define CATCH_FREE_RUNNING_WITH_BEMF ENABLED

However, this is just to enable the CFR-related routine to take place to in order to evaluate if the catch free running conditions are met at start time. The ceiling fan commences with CFR startup only if the CFR-related routine determines that the catch free running conditions have been met.

The typical scenario of CFR startup is when the ceiling fan is turning in forward direction while being switched on. If the prevailing ceiling fan speed is high enough, the CFR library function returns a positive return status of AMCLIB_CFR_STARTUP_CFR to its calling function. At the application code level, instead of letting it start from zero speed (i.e. normal startup), the application starts with a CFR startup process.

## 8.3 Remote control (RF/IR)

Here, both RF and IR remote functions use the same parameters.

- remote_counter_time 50 U
  - Time to determine the button pressed; 250 ~ 1 s timing |200 ~ 0.8 s|.
  - The longer this timing, the longer you must press to determine “button pressed”.
- start_delay_time 0 U
  - Delay time before the motor starts; set to 0 if not using
- stop_delay_time 0 U
  - Delay time before the motor stops; set to 0 if not using

# 9 Stop motor

## 9.1 Advance stop conditional

- Advanced_stop_exit_speed 80 U
  - When the button stops, it will ramp down, and once the actual RPM reaches this value, it will go to motor stop.

## 9.2 Stall detection

- STALL_RATIO 2.2 f
  - Value to be greater than or equal to 1. The smaller the value, the lower the speed to detect stall condition. For example, 1.0 f means the stall is only checked at zero speed, so when the motor is spinning, the stall will not be checked.
- STALL_CURRENT_A 0.35 f
  - Torque current ref_iq is being fed into; this value cannot be too high and has to be at least below the current speed value of 1. This allows the stall condition to be fulfilled at any current above this.

## 9.3 Software overcurrent protection

- I_OCP_SW 30000 U
  - This the OCP threshold limit. If any of the phase current exceeds this limit, OCP occurs.
  - 30000 / 32768 * 4.54 A = 4.156 A.

## 9.4 Over/undervoltage protection

- VDC_OVER_LIMIT VADC_DCLINK * 115 / 100
  - VADC_DCLINK + 15%, DC-link voltage V DC maximum limit
- VDC_MIN_LIMIT VADC_DCLINK * 75 / 100
  - VADC_DCLINK - 25%, DC-link voltage V DC maximum limit

## 9.5 Auto-retry

- FAULT_MAX_RETRY_COUNT 3 U
  - Auto-retry attempt count
- fault_timing 25
  - 0.1 s; the time before entering the auto-retry function

## 9.6 Deep sleep mode

- sleepmode_countertime 5333
  - Timing before entering Deep Sleep mode
  - About 3 s (800), 15 s (4000), 20 s (5333) to enter sleep mode |16 kHz - (4000) 16 s|

## 9.7 Power Limit

- POWER_LIMIT_MAX_W (30U)
  - You can enable or disable it
  - Power limit max in watt

# 10 Ceiling Fan GUI

The dedicated ceiling fan GUI is implemented as an application in the “GUI builder” as part of the ModusToolbox™ Motor Suite.

## 10.1 GUI Setup

The following steps will guide you through the process of setting up the GUI builder.

1. Open ModusToolbox™ Motor Suite and find and open **GUI Builder.**

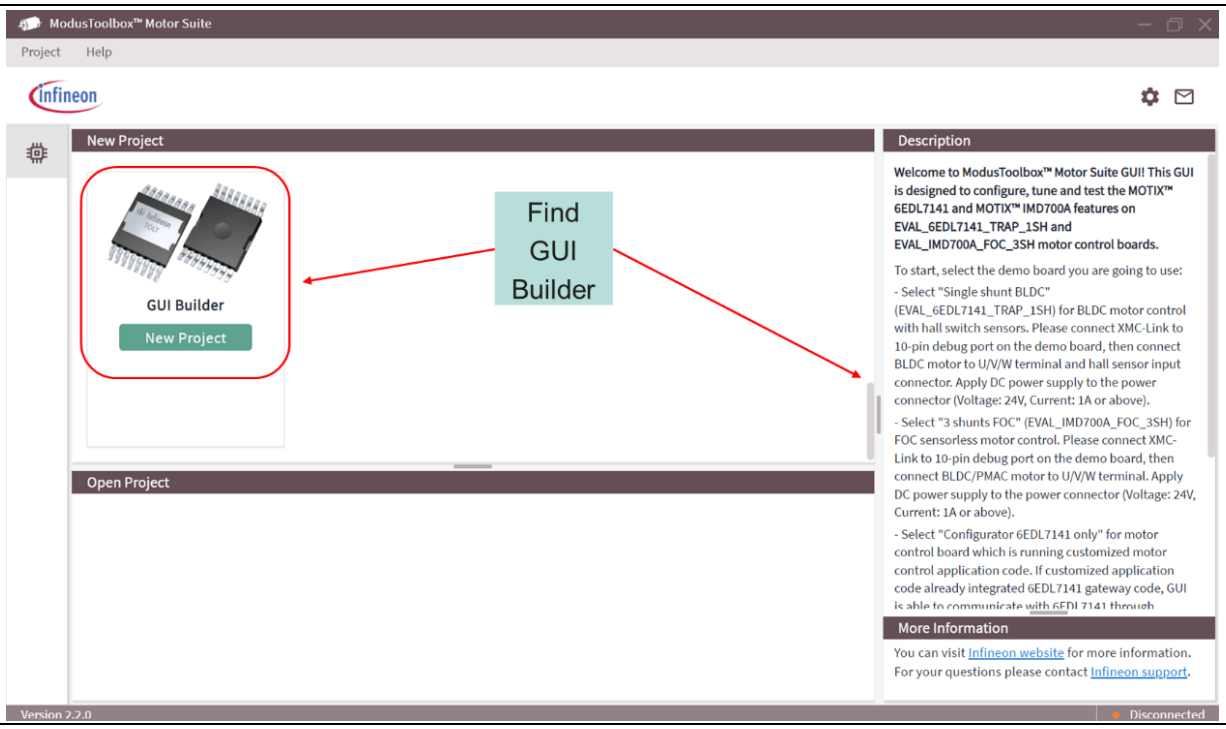


**Figure 2 GUI Builder**

2. Go to **Open project** and select the project *Ceiling_Fan_GUI_Builder.mcws* file.

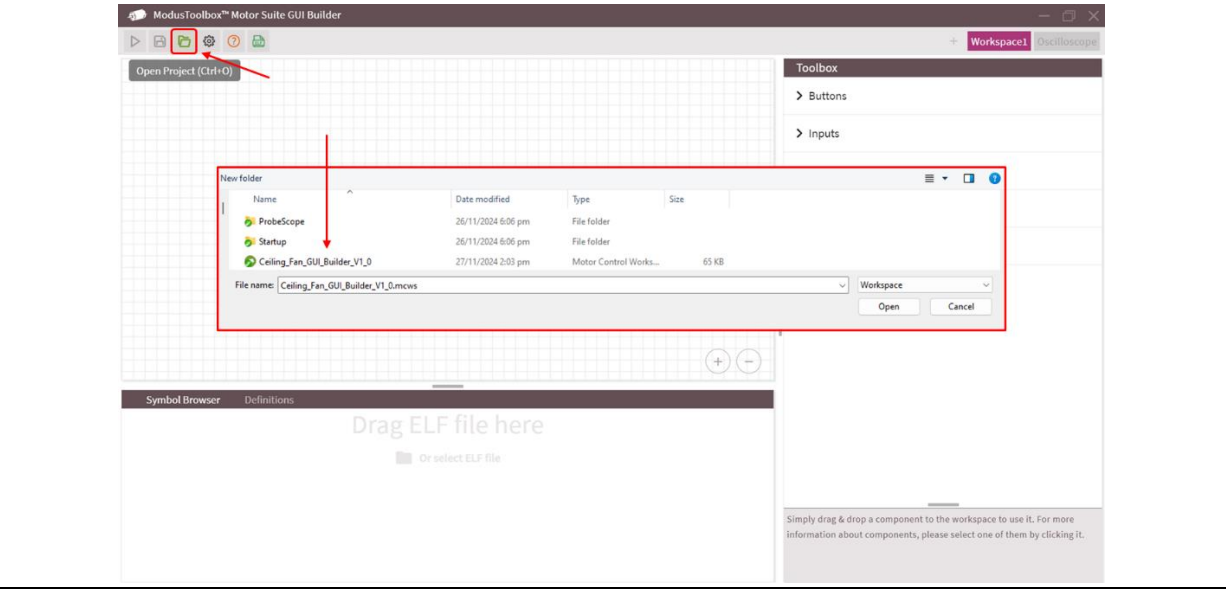


**Figure 3 Project GUI file selection**

3. Select the ELF file found in "*…/project folder/Debug/ELF*".

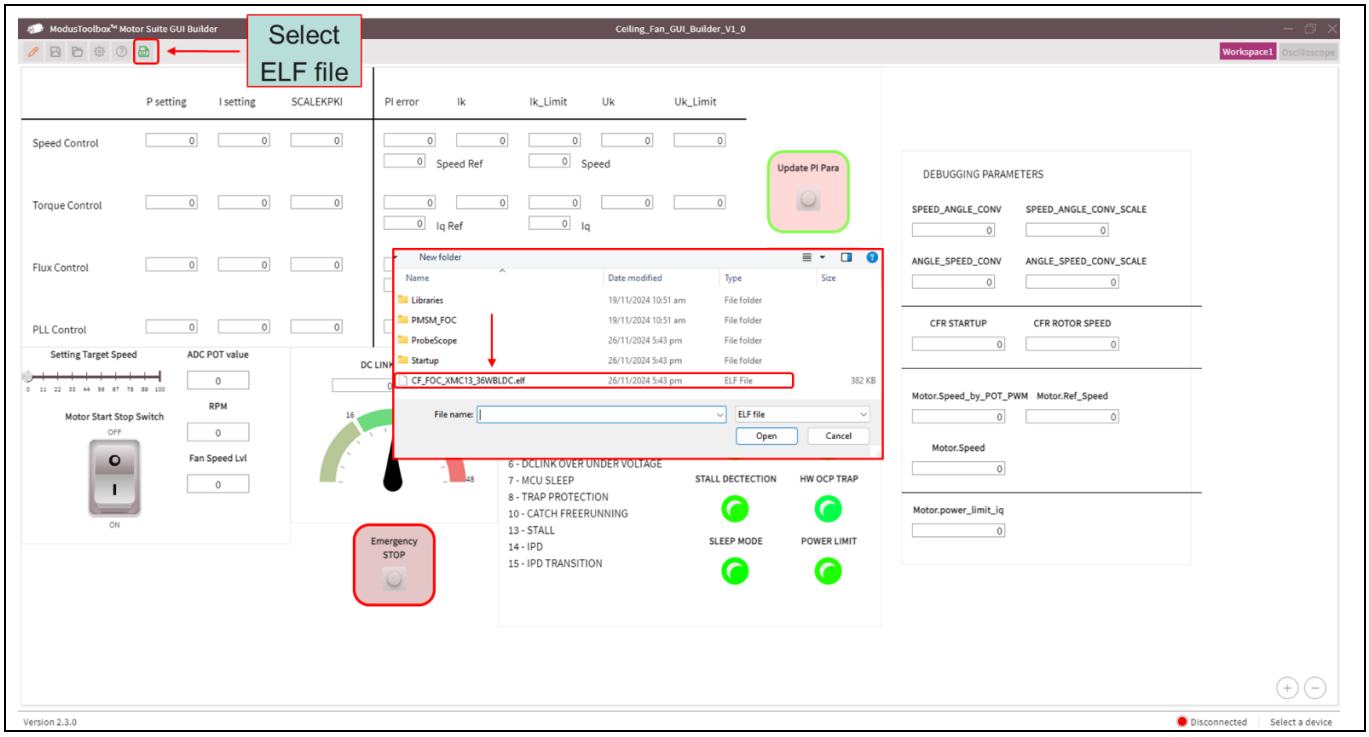


**Figure 4 Selecting the ELF file**

# 11 Overall rough motor tuning process

This section describes a rough motor tuning process. For any of the parameters mentioned, you can see the following information, which provides a better description and explanation of adjusting the values.

1. Set motor parameters.
2. Disable all the other features/functions that affect motor startup.
   a) Disable **Stall detection**.
   b) Disable **Stall check RPM** by setting it to **zero**.
   c) Disable **Pre-alignment**.
3. Run through and check on how the fan is spinning.
   a) Shape of waveform, noise, distortion.
      - If waveform is irregular/noisy/distorted or there is startup failure, go to Step 4.
4. PI tuning
   a) This is a repeating process and will require more time fine-tuning.
5. Pre-alignment
   a) Add in pre-alignment if needed. Adjusting of the alignment voltage, timing.
      - If startup and current waveform are looking good, there is no distortion or noise, and the motor spinning is good, go to Step 6.
6. CFR startup (manually spin the fan and power on to test the CFR feature)
   b) Adjust CFR related parameters only if needed.
      - If fan motor is running well including the CFR, proceed to the next step.
7. Enable and check stall detection.
      - OCP ADC value depending on the top speed (speed 5).

## 11.1 Tuning of the PI controller gains

Figure 5 gives a rough guide to tuning the PI controller gains and shows the effect of increasing and decreasing the $K_p$ or $K_i$.

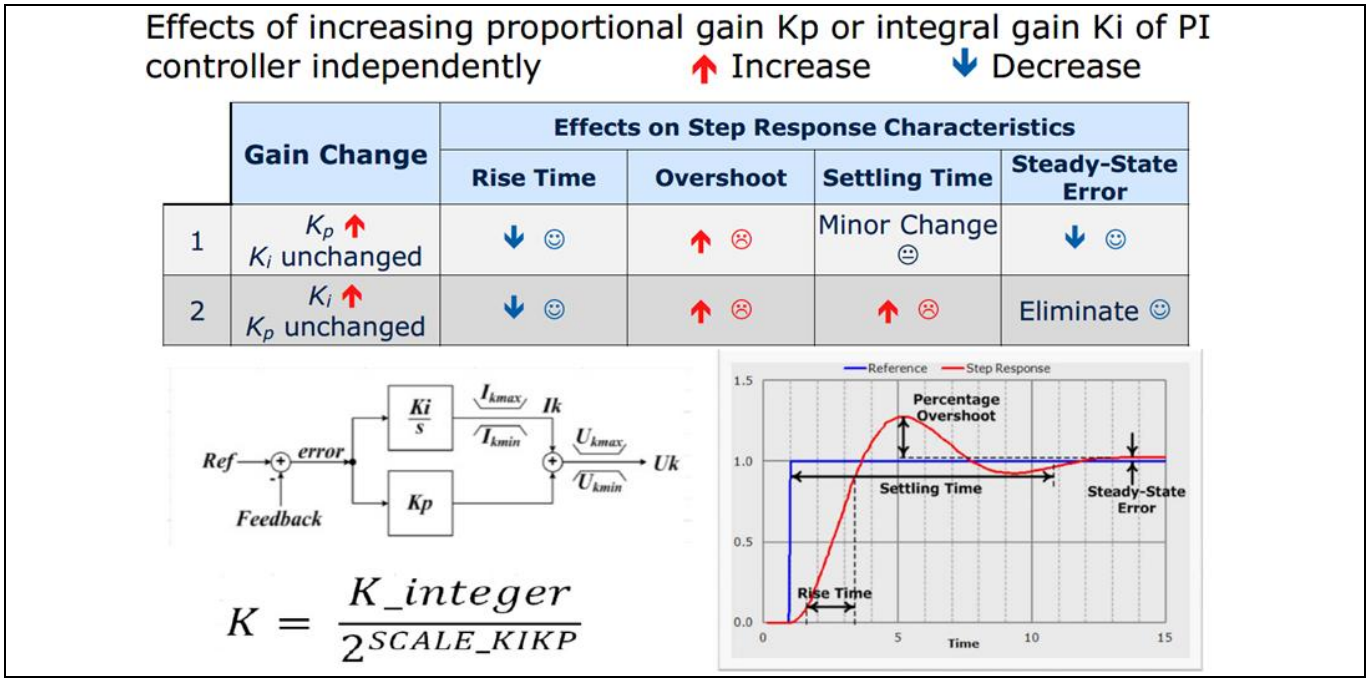


**Figure 5 Tuning of PI controller gains**

## 11.1.1 Tuning of the PI parameters

Figure 6 shows the corresponding $K_i$, $K_p$, $I_k$, and $U_k$ from the control scheme on the GUI. You can fine-tune the $K_i$ and $K_p$ values, checking the $I_k$ output and the PI controller output $U_k$ respectively.

One can also fine-tune the SCALEKPKI, which according to the formula:

- Increases value by 1, decreasing the gain of controller by half (1/2)
- Decreases value by 1, doubling the gain of controller (2x)

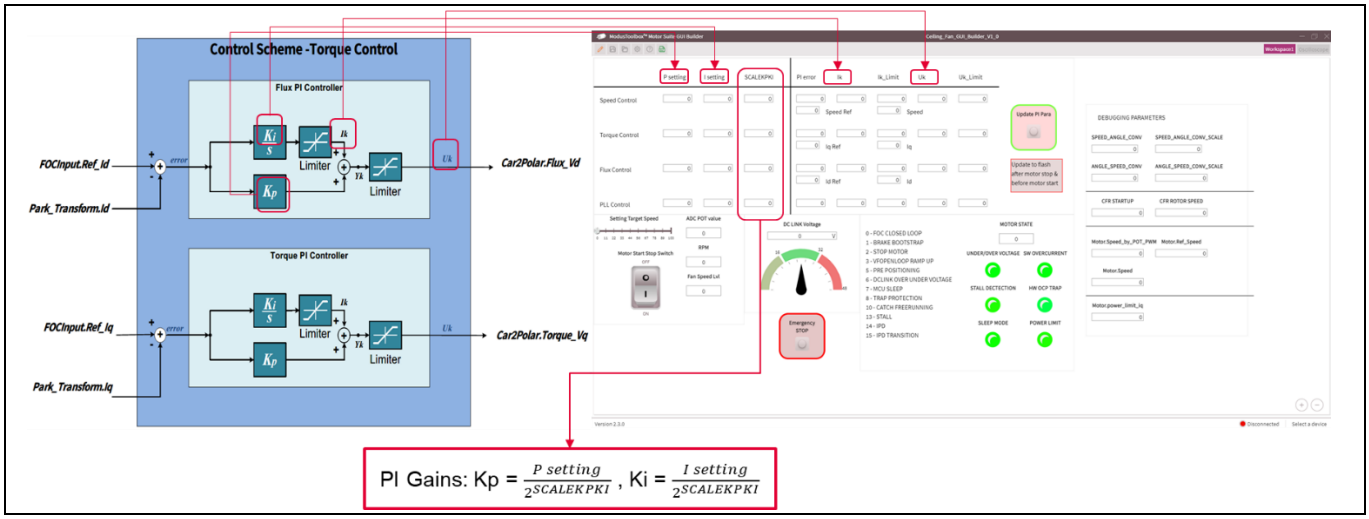


**Figure 6 Tuning of PI parameters**

### Start-up of motor using speed control scheme and tuning of different PI parameters

- In this control scheme, you need to tune the phase-locked loop (PLL), the current loop (torque control and flux control), and the speed loop (speed control).

### Recommended incremental tuning process

To enable incremental tuning process, it is recommended the following process.

1. Set the control scheme to VQ control:
   - Focus on the tunings on PLL control and Flux control $K_p$, $K_i$, and SCALEKPKI values.
2. Set the control scheme to Torque control:
   - Focus on the tunings on Torque control $K_p$, $K_i$, and SCALEKPKI values, while keeping an eye on PLL control and Flux control $K_p$, $K_i$, and SCALEKPKI values.
3. Set the control scheme to Speed control:
   - Focus on the tunings on Speed control $K_p$, $K_i$, and SCALEKPKI values, while keeping an eye on Torque control, PLL control, and Flux control $K_p$, $K_i$, and SCALEKPKI values.

### VQ control PI tuning

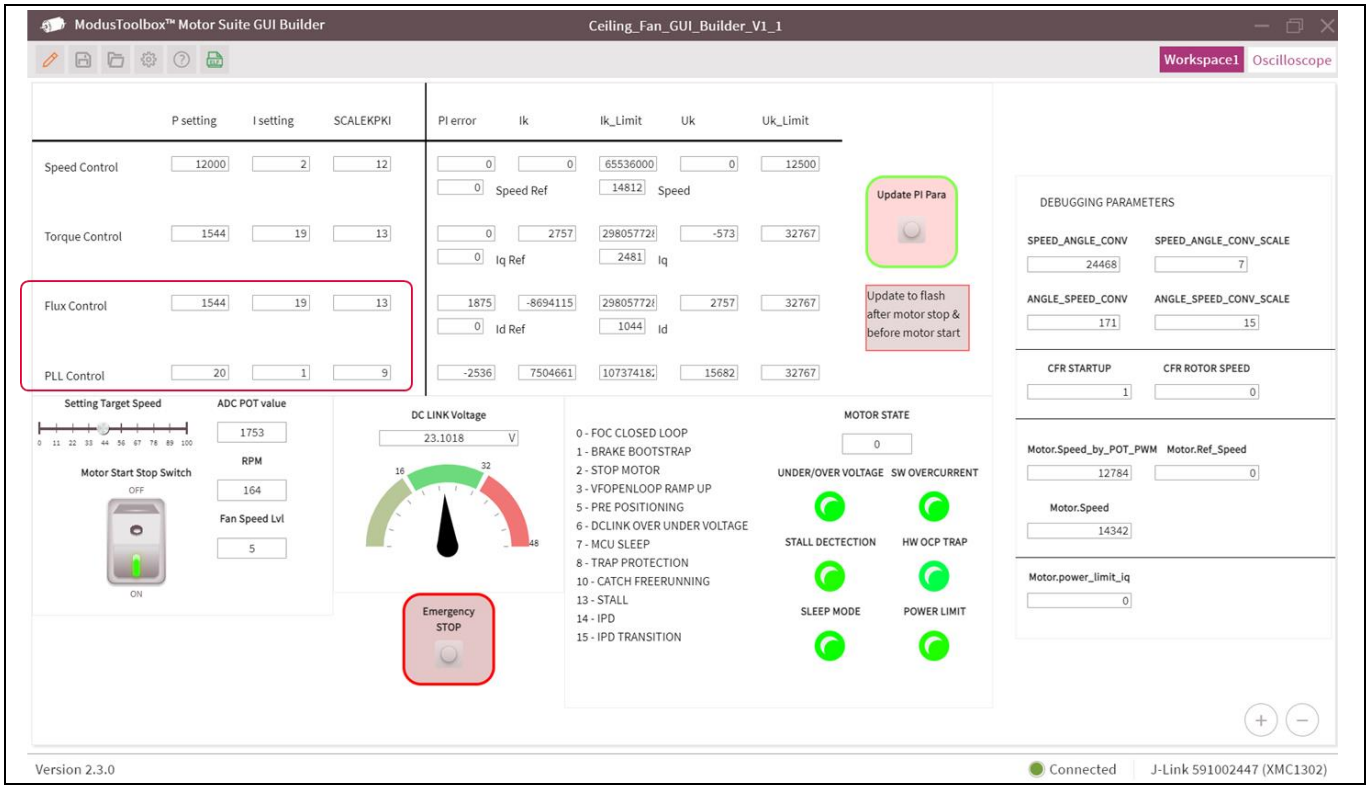


**Figure 7 VQ control PI tuning area**




**Figure 8 OSC display while motor running close to configured USER_VQ_REF_HIGH_LIMIT**

1. Focus on the tunings on PLL control and Flux control $K_p$, $K_i$, and SCALEKPKI values.
2. Check if the motor can start running smoothly.
3. If unable to startup, tune the PLL first.
   - Start by decreasing the PLL gain by increasing SCALEKPKI and decreasing the PLL PI setting. If you are unable to get it running, try the other way.
4. During the motor startup, the motor may be briefly stuck before running properly again; increasing the PLL $K_i$ value helps.
5. When the speed is unstable, decreasing the PLL $K_p$ value helps.
6. $I_k$ limit values for both PLL control and Flux control should not be too high in case of oversaturation.
7. Check that the phase current frequency running close to the configured USER_VQ_REF_HIGH_LIMIT (captured from OSC) makes sense with the Motor.Speed value dynamically displayed on the MicroInspector.
   - 22.58Hz * 60 / 4 /350 *32767 = 31709
8. Corresponding default values in *pmsm_foc_motor_parameters.h*.

```
#define USER_PI_FLUX_KP          (1544U)
#define USER_PI_FLUX_KI          (19U)
#define USER_PI_FLUX_SCALE_KPKI  (13U)
#define USER_PI_PLL_KP           (20U)
#define USER_PI_PLL_KI           (1U)
#define USER_PI_PLL_SCALE_KPKI   (9U)
```

## Torque control PI tuning

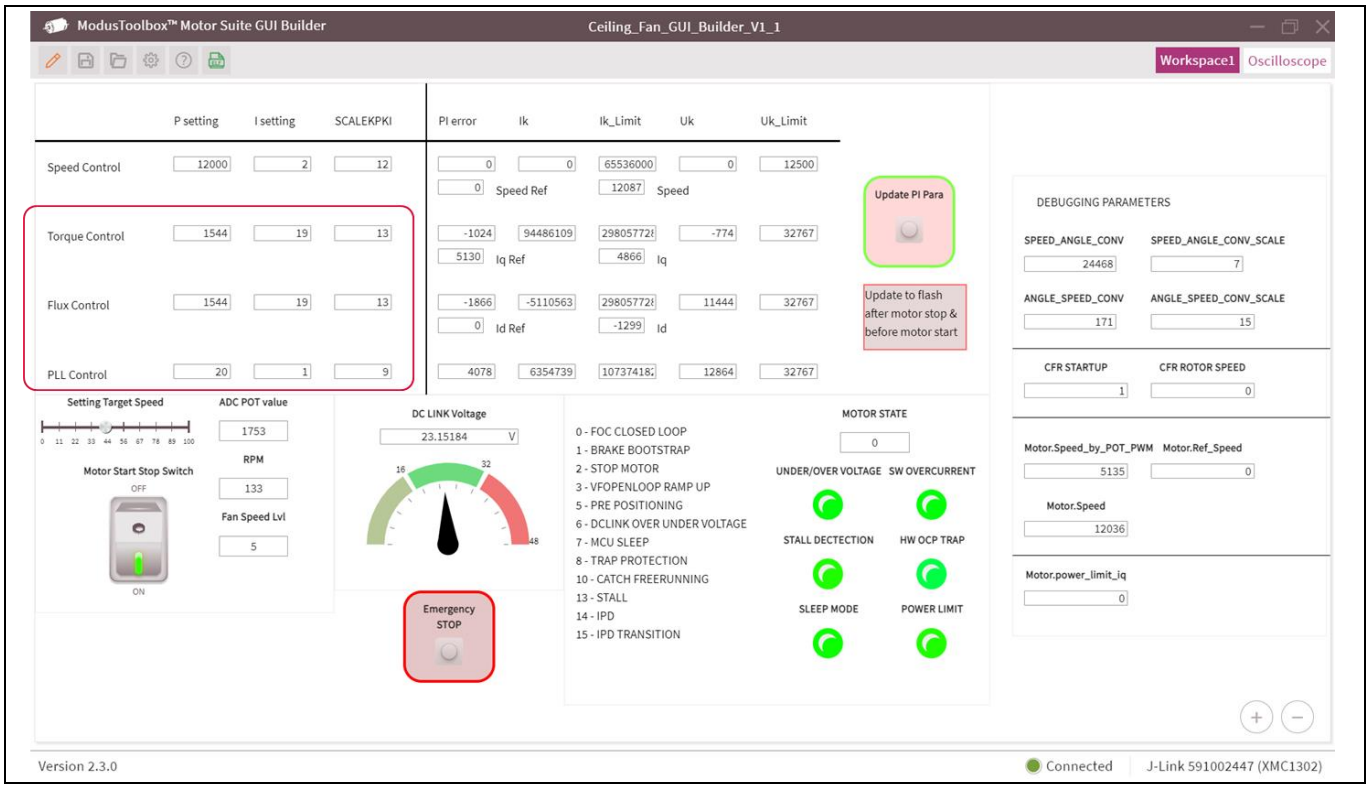


**Figure 9 Torque control PI tuning area**

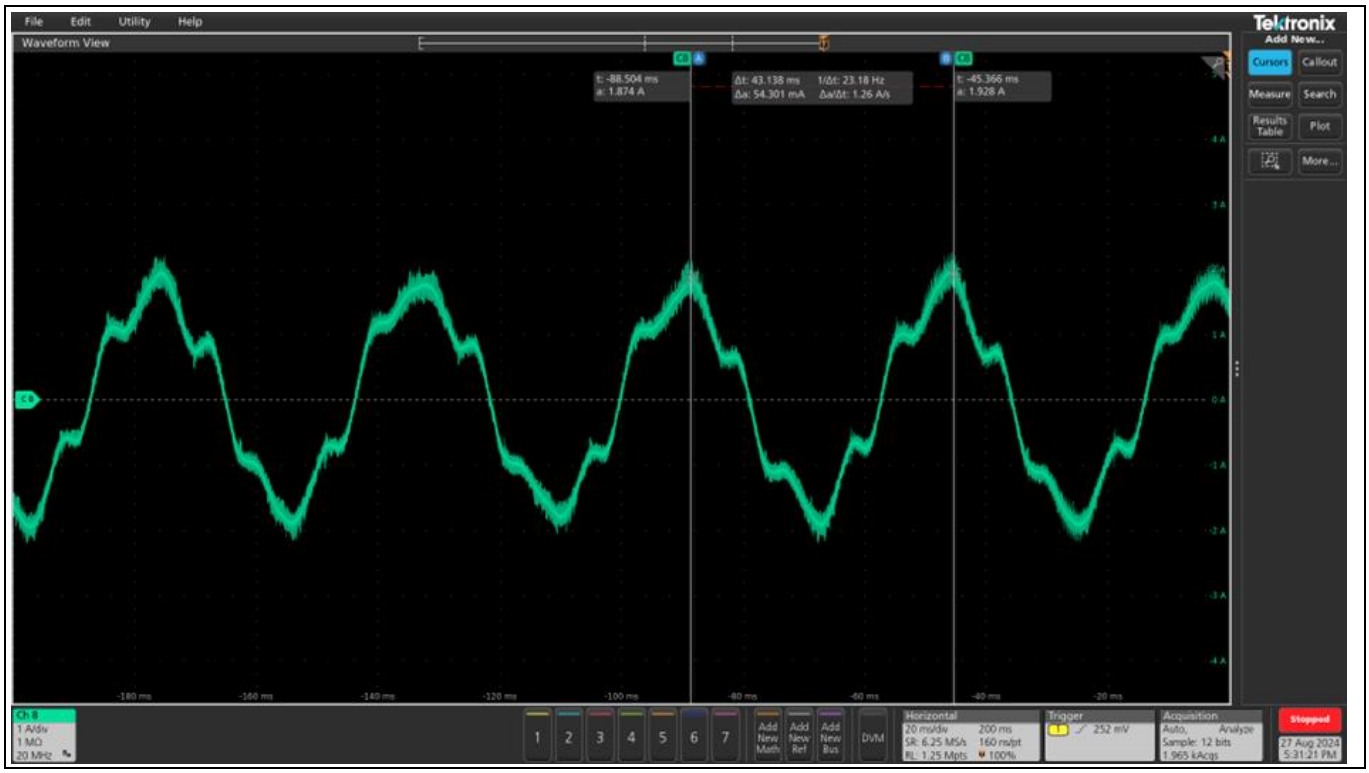


**Figure 10 OSC display while motor running close to the configured USER_IQ_REF_HIGH_LIMIT**

## Overall rough motor tuning process

1. Focus on the tunings on Torque control $K_p$, $K_i$, and SCALEKPKI values, while keeping an eye on PLL control and Flux control $K_p$, $K_i$, and SCALEKPKI values.
2. Check if the motor can start running smoothly.
3. Ensure the Ik limit values for PLL control, Flux control, and Torque control are not too high in case of oversaturation.
4. Check that the phase current frequency running close to the configured USER_IQ_REF_HIGH_LIMIT (captured from OSC) makes sense with the Motor.Speed value dynamically displayed on the MicroInspector.
   - 23.18Hz * 60 / 4 /350 *32767 = 32551
5. Corresponding default values in *pmsm_foc_motor_parameters.h*

```
#define USER_PI_TORQUE_KP          (1544U)
#define USER_PI_TORQUE_KI          (19U)
#define USER_PI_TORQUE_SCALE_KPKI  (13U)
#define USER_PI_FLUX_KP            (1544U)
#define USER_PI_FLUX_KI            (19U)
#define USER_PI_FLUX_SCALE_KPKI    (13U)
#define USER_PI_PLL_KP             (20U)
#define USER_PI_PLL_KI             (1U)
#define USER_PI_PLL_SCALE_KPKI     (9U)
```

### Speed control PI tuning

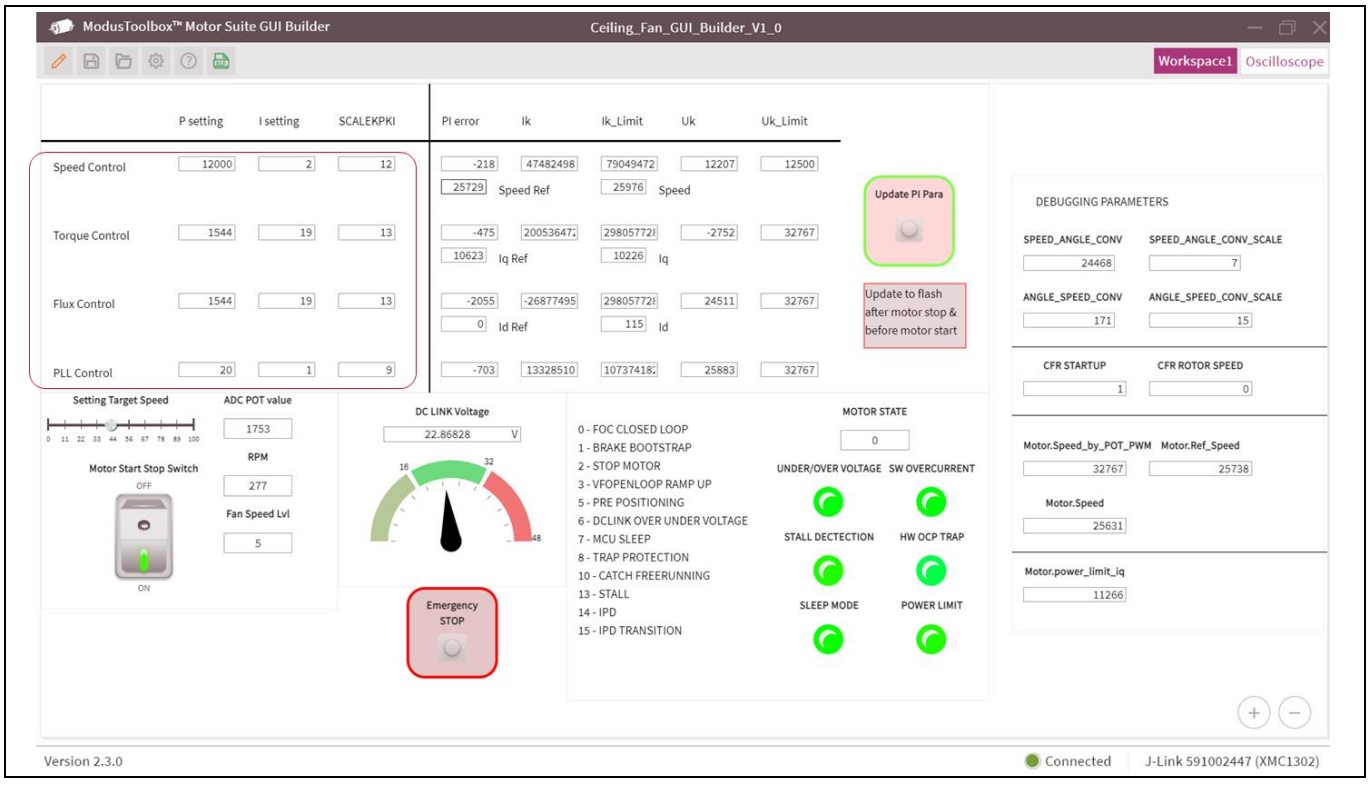


**Figure 11 Speed control PI tuning area**

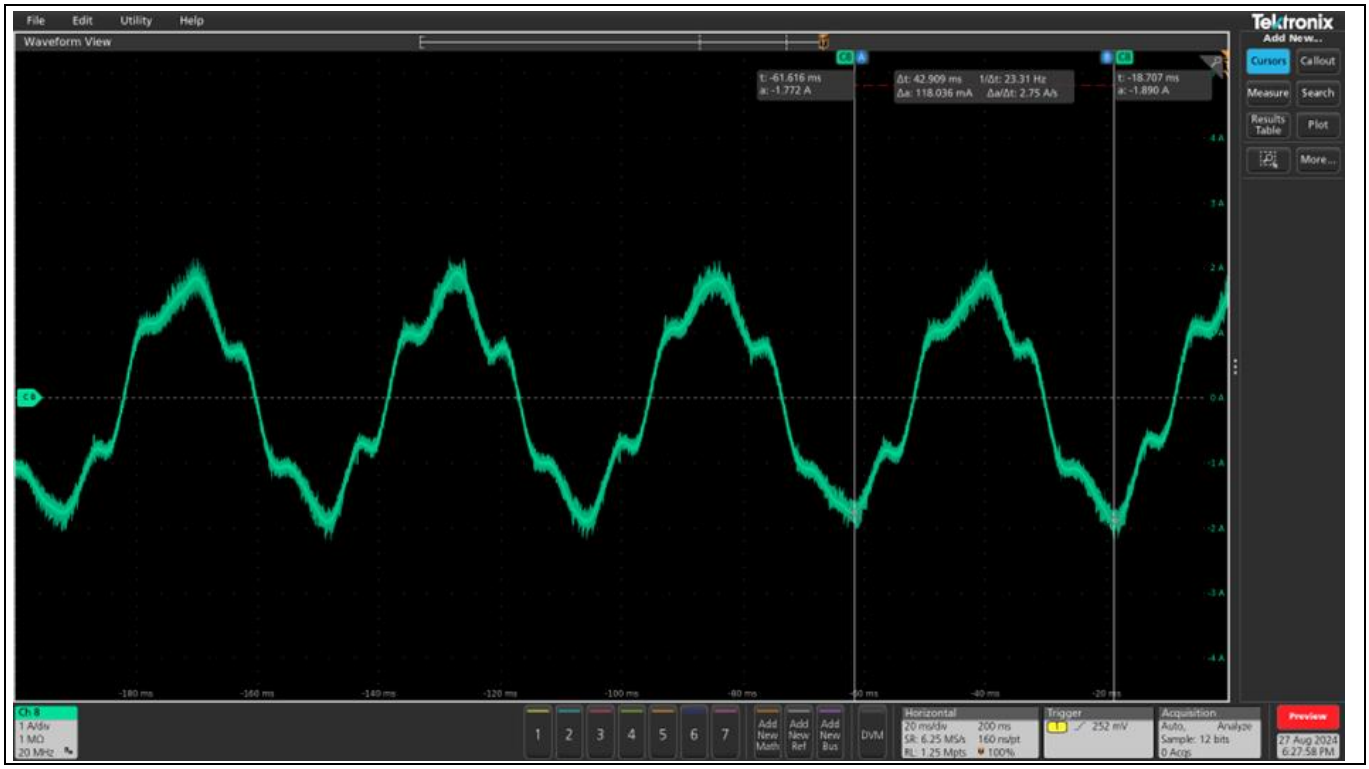


**Figure 12 OSC display while motor running at speed level 5 (350rpm)**

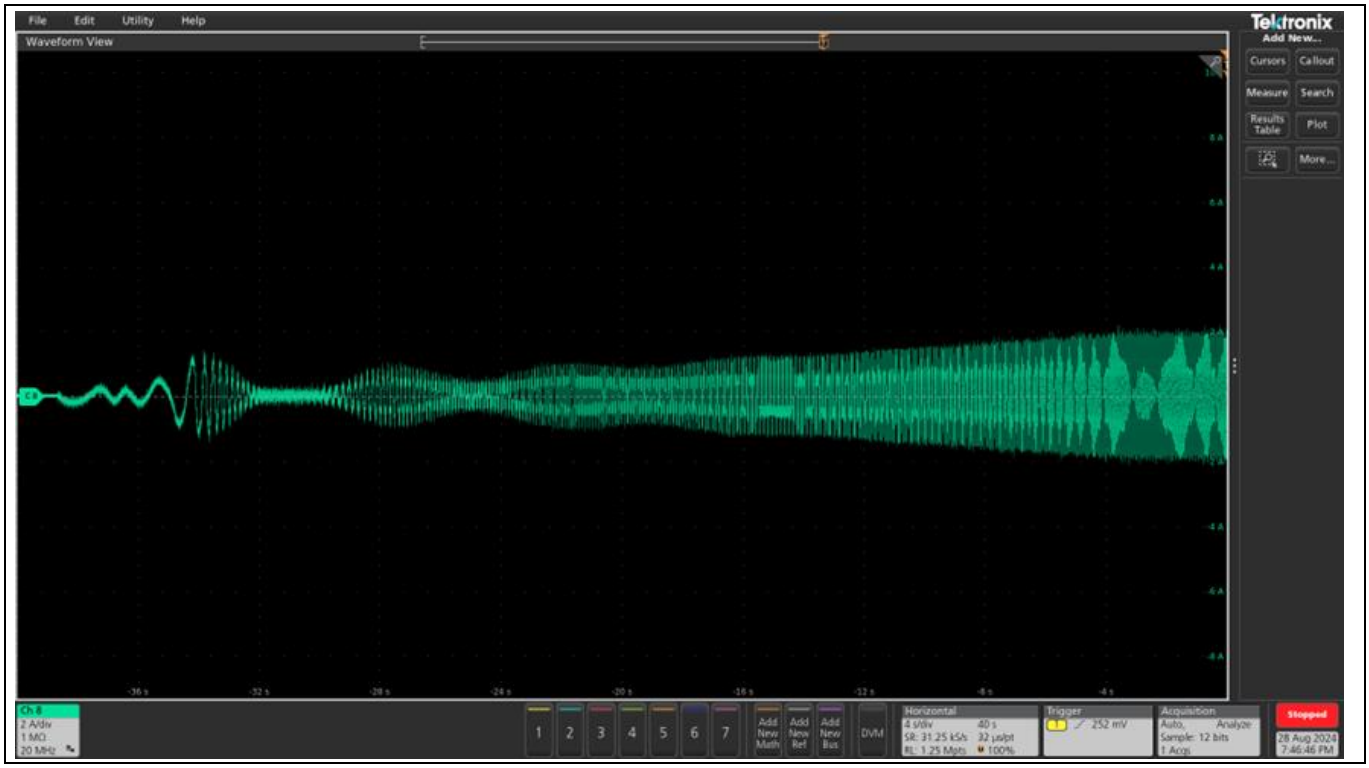


**Figure 13 Observation that motor is ramping up to the intended speed rather smoothly**

1. Focus on the tunings on Speed control $K_p$, $K_i$, and SCALEKPKI values, while keeping an eye on Torque control, PLL control, and Flux control $K_p$, $K_i$, and SCALEKPKI values.
2. Check if the motor can start running smoothly.

3. Tune the speed control loop. Higher speed Ki value will result in higher overshoot. Lower speed Ki value will result in lower dynamic. Fine-tune according to the overshoot result during of the dynamic change and static error during steady operation.
4. Ensure the Ik limit values for PLL control, Flux control, and Torque control are not too high in case of oversaturation.
5. Check that the phase current frequency running close to configured speed level 5 (captured from OSC) make sense with Motor.Speed value dynamically displayed on the MicroInspector.
   - Speed in RPM = 23.31Hz * 60 / 4 = 349.65 RPM
6. Corresponding default values in *pmsm_foc_motor_parameters.h*

```
#define USER_PI_SPEED_KP          (2000U)
#define USER_PI_SPEED_KI          (2U)
#define USER_PI_SPEED_SCALE_KPKI  (14U)
#define USER_PI_TORQUE_KP         (1544U)
#define USER_PI_TORQUE_KI         (19U)
#define USER_PI_TORQUE_SCALE_KPKI (13U)
#define USER_PI_FLUX_KP           (1544U)
#define USER_PI_FLUX_KI           (19U)
#define USER_PI_FLUX_SCALE_KPKI   (13U)
#define USER_PI_PLL_KP            (20U)
#define USER_PI_PLL_KI            (1U)
#define USER_PI_PLL_SCALE_KPKI    (9U)
```

7. Consider that after close observations of numerous motor startup attempts, you see the following Speed control $K_p$, $K_i$, and SCALEKPKI values are ideal for a smoother startup:

```
#define USER_PI_SPEED_KP          (12000U)
#define USER_PI_SPEED_KI          (2U)
#define USER_PI_SPEED_SCALE_KPKI  (12U)
```

8. Corresponding tuned values in *pmsm_foc_motor_parameters.h*

```
#define USER_PI_SPEED_KP          (12000U)
#define USER_PI_SPEED_KI          (2U)
#define USER_PI_SPEED_SCALE_KPKI  (12U)
#define USER_PI_TORQUE_KP         (1544U)
#define USER_PI_TORQUE_KI         (19U)
#define USER_PI_TORQUE_SCALE_KPKI (13U)
#define USER_PI_FLUX_KP           (1544U)
#define USER_PI_FLUX_KI           (19U)
#define USER_PI_FLUX_SCALE_KPKI   (13U)
#define USER_PI_PLL_KP            (20U)
#define USER_PI_PLL_KI            (1U)
#define USER_PI_PLL_SCALE_KPKI    (9U)
```

## Revision history

| Document revision | Date | Description of changes |
|---|---|---|
| V 1.0 | 2023-11-30 | Initial release |
| V 2.0 | 2024-12-20 | Updated to reflect scaling fixes and the startup scheme being changed from torque control to speed control |

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

**Edition 2023-12-20**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**

**© 2024 Infineon Technologies AG. All Rights Reserved.**

**Do you have a question about this document?**

**Email:**

erratum@infineon.com

**Document reference**

**UG080514**

**Warnings**

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.