Firmware customization guide for REF_MTR_48V30A_GaN

About this document

Scope and purpose

This document provides a background of the firmware of the REF_MTR_48V30A_GaN reference board and a guide on running motors with this firmware. This document focuses on the firmware changes that are specific to REF_MTR_48V30A_GaN, and provides some hands-on guidance on motor tuning.

Intended audience

This document is intended for design engineers, technicians, users who want to get familiar with the firmware of REF_MTR_48V30A_GaN and use it as a design reference.

Attention

There are some limits that **must be strictly followed before powering up or any tuning of firmware** on this board. Please read carefully before performing any tests on this board.

Table of contents

About this document **1**
Table of contents **2**
1 (Read first!) Limitations **3**
2 Hardware **4**
2.1 Temperature sensing and protection 4
2.2 OCD protection (motor overcurrent protection) 4
3 Firmware **5**
3.1 Preparation and preliminaries 5
3.1.1 Prepare tools and install software 5
3.2 Firmware structure 6
3.2.1 MCU pin assignment 6
3.2.2 Performance data 7
3.3 Tuning motor 8
3.3.1 Motor parameters 9
3.3.1.1 Stator resistance 10
3.3.1.2 q-axis inductance and d-axis inductance 11
3.3.1.3 Permanent magnet's flux linkage and pole pairs 12
3.3.1.4 Maximum torque 13
3.3.1.5 Mechanical parameters 13
3.3.1.6 Motor parameters set in firmware 20
3.3.2 Motor control mode 20
3.3.2.1 Speed_Mode_FOC_Sensorless_ Volt_Startup 21
3.3.2.2 Speed_Mode_FOC_Sensorless_Align_Startup 26
3.3.2.3 Speed_Mode_FOC_Sensorless_SixPulse_Startup 28
3.3.2.4 Speed_Mode_FOC_Sensorless_ HighFreq_Startup 30
3.4 GUI 33
References **34**
Revision history **35**
Disclaimer **36**

1 (Read first!) Limitations

Attention: ***The following limits must be strictly followed before powering up or tuning the firmware. Failure to do so will result in damage to the hardware board or even cause loss and injuries.***

1. Short the low-side precharge MOSFET (Q5) on the REF_MTR_48V30A_GaN board. See , “Short the precharge FET” of the user guide [8].

   If it is not shorted on the board, the DC input power supply to this board **must power up slowly**. In other words, do not power up the board with 48 V directly.

2. The **maximum duty cycle** of each high-side of the three phases **must** be limited to **90 percent**. This comes from actual test results according to the adopted bootstrap components.

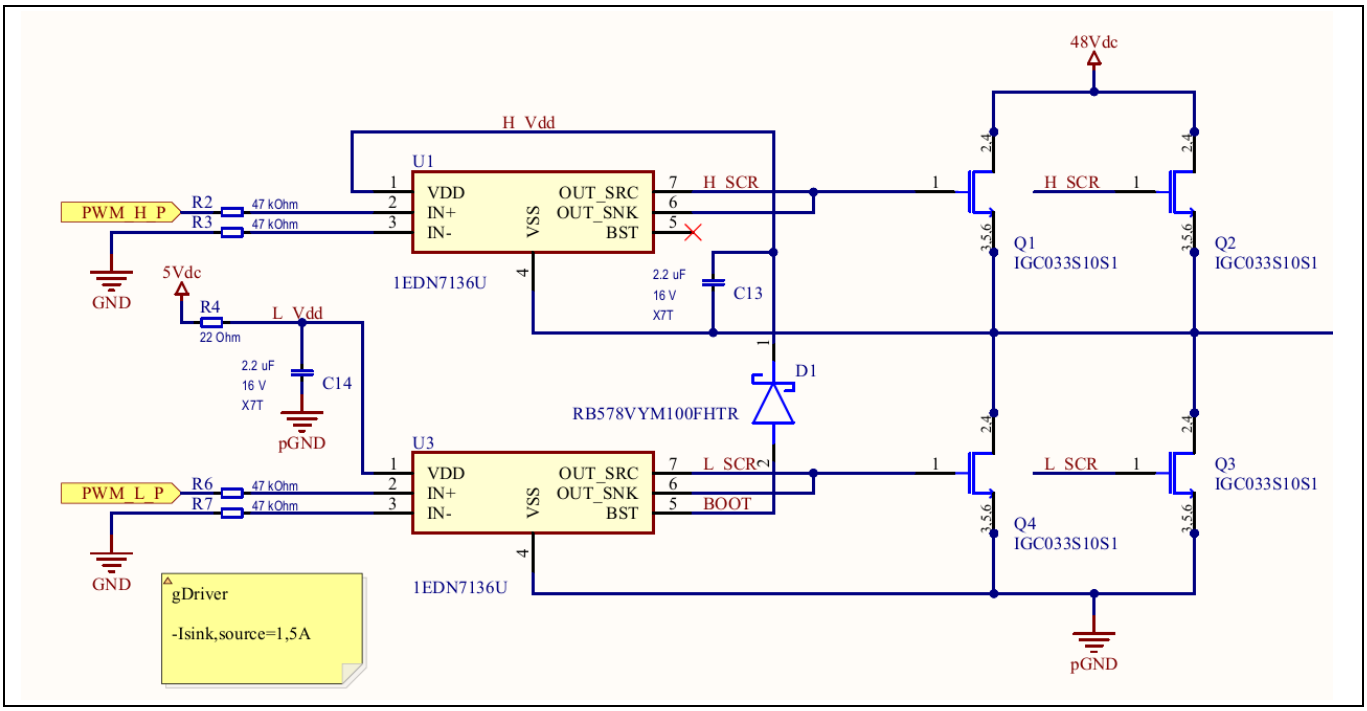

Figure 1 Bootstrap schematic

In the firmware, the maximum duty cycle has been limited to 90 percent as the following code shows. Do not change this limit.

Code listing 1 Maximum duty cycle limit in MCU_RunISR0() in MCU.c

```
vars.d_uvw_cmd.u = SAT(0.0f, 0.90f, vars.d_uvw_cmd.u);
vars.d_uvw_cmd.v = SAT(0.0f, 0.90f, vars.d_uvw_cmd.v);
vars.d_uvw_cmd.w = SAT(0.0f, 0.90f, vars.d_uvw_cmd.w);
```

2 Hardware

2.1 Temperature sensing and protection

Depending on the mounted temperature sensor, MCP9700AT-E/LT (top mark is AXNN) or MAX6612 (top mark is ADG), the offset and scaling of temperature sensing may need to be changed. To do so, select one and comment the other in the following code:

Code listing 2 Select the temperature sensor in MotorCtrlHWConfig.h. Example: MAX6612 is mounted

```
// #define TEMP_SENSOR_OFFSET (0.5f)        // MCP9700AT-E/LT: 500mV is 0 ℃
// #define TEMP_SENSOR_SCALE  (0.010f)      // MCP9700AT-E/LT: 10mV/℃
#define TEMP_SENSOR_OFFSET (0.4f)         // MAX6612: 400mV is 0 ℃
#define TEMP_SENSOR_SCALE  (0.01953f)     // MAX6612: 19.53mV/℃
```

The overtemperature threshold can be changed in the following code. For example, the threshold can be increased to get more headroom for performance.

Code listing 3 Overtemperature threshold setting in PARAMS_InitManual() in Params.c

```
params.sys.faults.temp_ps_thresh = 90.0f; // [Celsius]
```

2.2 OCD protection (motor overcurrent protection)

The TLI4971 current sensor (50 A full scale) outputs two pins for the MCU to detect motor overcurrent condition: OCD1 and OCD2. The protection threshold is 62.5 A and 41 A respectively.

By default, OCD2 is used for protection with 41 A instantaneous motor current value as the threshold. You can change this to use OCD1 instead to get higher current capability of 62.5 A using the following code. Comment the first line and uncomment the second line.

Code listing 4 OCD1 and OCD2 protection in MCU_RunISR1() of Params.c

```
sensor_iface.digital.fault = !XMC_GPIO_GetInput(N_FAULT_HW_PORT,
N_FAULT_HW_PIN);
//sensor_iface.digital.fault = !XMC_GPIO_GetInput(OCD1_PORT, OCD1_PIN);
```

3 Firmware

3.1 Preparation and preliminaries

3.1.1 Prepare tools and install software

Before getting hands-on with the firmware, you should prepare or install the following tools:

- XMC™ Link debugger [1]. The hardware is to debug and flash the code on XMC4200 MCU.
- Download and install ModusToolbox™ V3.2 [2]. ModusToolbox™ is a set of multiplatform development tools and a comprehensive suite of GitHub-hosted firmware libraries. It is used mainly as an IDE for building and debugging firmware code. The generated built output files (.hex, .elf files) can also be imported in Micrium µC/Probe™ XMC™ software or Ozone for GUI-based tuning.
- Download and install Micrium µC/Probe™ XMC™ software [3]. It is used to open the GUI workspace file (*REF_MTR_48V30A_GaN_GUI.wspx*) and provide the GUI to tune and run the motor.
- Download and install Ozone – The J-Link Debugger [4]. This tool from Segger is used for tuning and debugging. It is especially useful to watch the variable values in real time. The Timeline window can show the graphs of variable values on the fly.

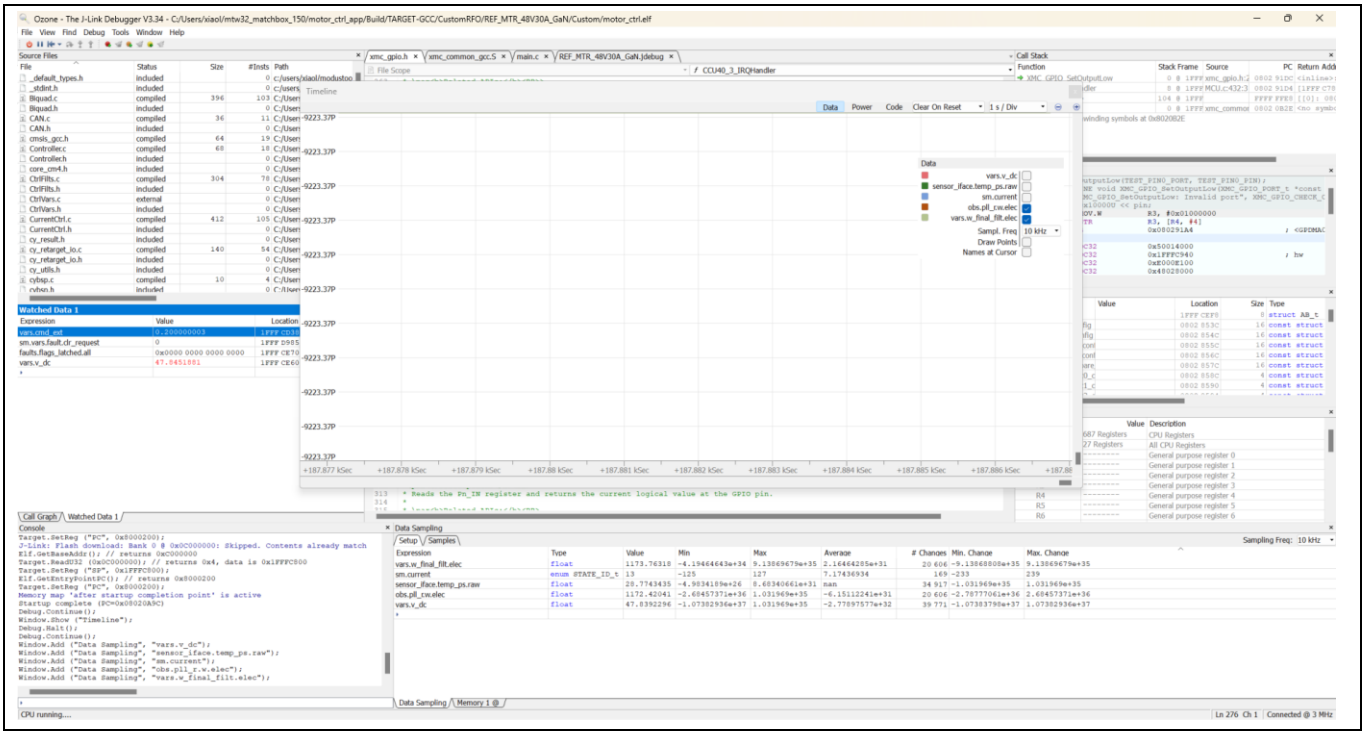

Figure 2 Ozone for debugging

Note: For commercial use, J-Link PLUS, ULTRA+, and PRO models come with a built-in license for Ozone. XMC™ Link is a J-Link base model; it can be used for free on Ozone only for evaluation purposes and non-commercial use.

- Download and install Infineon Developer Center [5]. This is a software platform that integrates many software tools from Infineon.

3.2 Firmware structure

The firmware of the REF_MTR_48V30A_GaN board is based on motor control firmware V1.5.0, which is a flexible and versatile platform. See the reference manual [6] that describes the detailed design and algorithms.

3.2.1 MCU pin assignment

REF_MTR_48V30A_GaN adopts XMC4200-Q48x256 (VQFN-48) as the MCU. The pin assignment is listed in the following table:

Table 1 XMC4200-Q48x256 (VQFN-48) MCU pin assignment

MCU pin	Input/ output	MCU setting	Schematic name	Device configurator name	Usage
P0.0	Output	CCU80.OUT21	PWM_L_P1	PWMUL	CCU80 slice 2 UL PWM
P0.1	Output	CCU80.OUT11	PWM_L_P2	PWMVL	CCU80 slice 1 VL PWM
P0.2	Output	CCU80.OUT01	PWM_L_P3	PWMWL	CCU80 slice 0 WL PWM
P0.3	Output	CCU80.OUT20	PWM_H_P1	PWMUH	CCU80 slice 2 UH PWM
P0.4	Output	CCU80.OUT10	PWM_H_P2	PWMVH	CCU80 slice 1 VH PWM
P0.5	Output	CCU80.OUT00	PWM_H_P3	PWMWH	CCU80 slice 0 WH PWM
P0.6	Output	GPIO output	–	PRE_CHARGE	PRE_charge
P0.7	Input	TDI	TDI	–	Emulator
P0.8	Output	GPIO output	N/A	TEST_PIN0	TEST_PIN0
P1.0	–	–	–	–	Not used
P1.1	Input	–	ENC_3	HALL_2, ENC_Z	Encoder input
P1.2	Input	–	ENC_2	HALL_1, ENC_B	Encoder input
P1.3	Input	–	ENC_1	HALL_0, ENC_A	Encoder input
P1.4	Output	GPIO output	SB_CAN	SB_CAN	CAN receiver standby input. High level: standby
P1.5	Input	CAN Node0 RXD	CAN_RX_3.3Vdc	CAN_RX	CAN RX
P2.0	Output	CAN Node0 TXD	CAN_TX_3.3Vdc	CAN_TX	CAN TX
P2.1	–	TDO	TDO	–	Emulator
P2.2	Input	USIC0_CH1 dx0	RX_UART	CYBSP_DEBUG_UART_RX	UART RX
P2.3	Input	–	OVER_CURR_2	N_FAULT_HW	OCD input 2
P2.4	Input	–	OVER_CURR_1	OCD1	OCD input 1

MCU pin	Input/ output	MCU setting	Schematic name	Device configurator name	Usage
P2.5	Output	USIC0_CH1 dout0	TX_UART	CYBSP_DEBUG_UART_TX	UART TX
P14.0	Output	GPIO output	N/A	TEST_PIN1	TEST_PIN1
P14.3	Input	G0CH3/G1CH3	I_Phase1	IUP	U phase motor current
P14.4	Input	G0CH4	I_Phase2	IVP	V phase motor current
P14.5	Input	G0CH5	I_Phase3	IWP	W phase motor current
P14.6	Input	G0CH6	Temp	T_POWER	Temperature
P14.7	Input	G0CH7	Vmes	VBUS	DC BUS voltage
P14.8	Output	GPIO output	N/A	TEST_PIN2	TEST_PIN2
P14.9	Output	GPIO output	N/A	TEST_PIN3	TEST_PIN3

3.2.2 Performance data

The typical execution time of fast isr0 and slow isr1 when the motor is running with speed loop control is as follows.

Table 2 Performance in speed loop control

	Execution period	Actual execution time	Remark
isr0	20 kHz	~39.225 µs	Fast control loop
isr1	1 kHz	~12.5 µs (when not occupied by isr0) ~57.25 µs (when occupied by isr0)	Slow control loop

3.3 Tuning motor

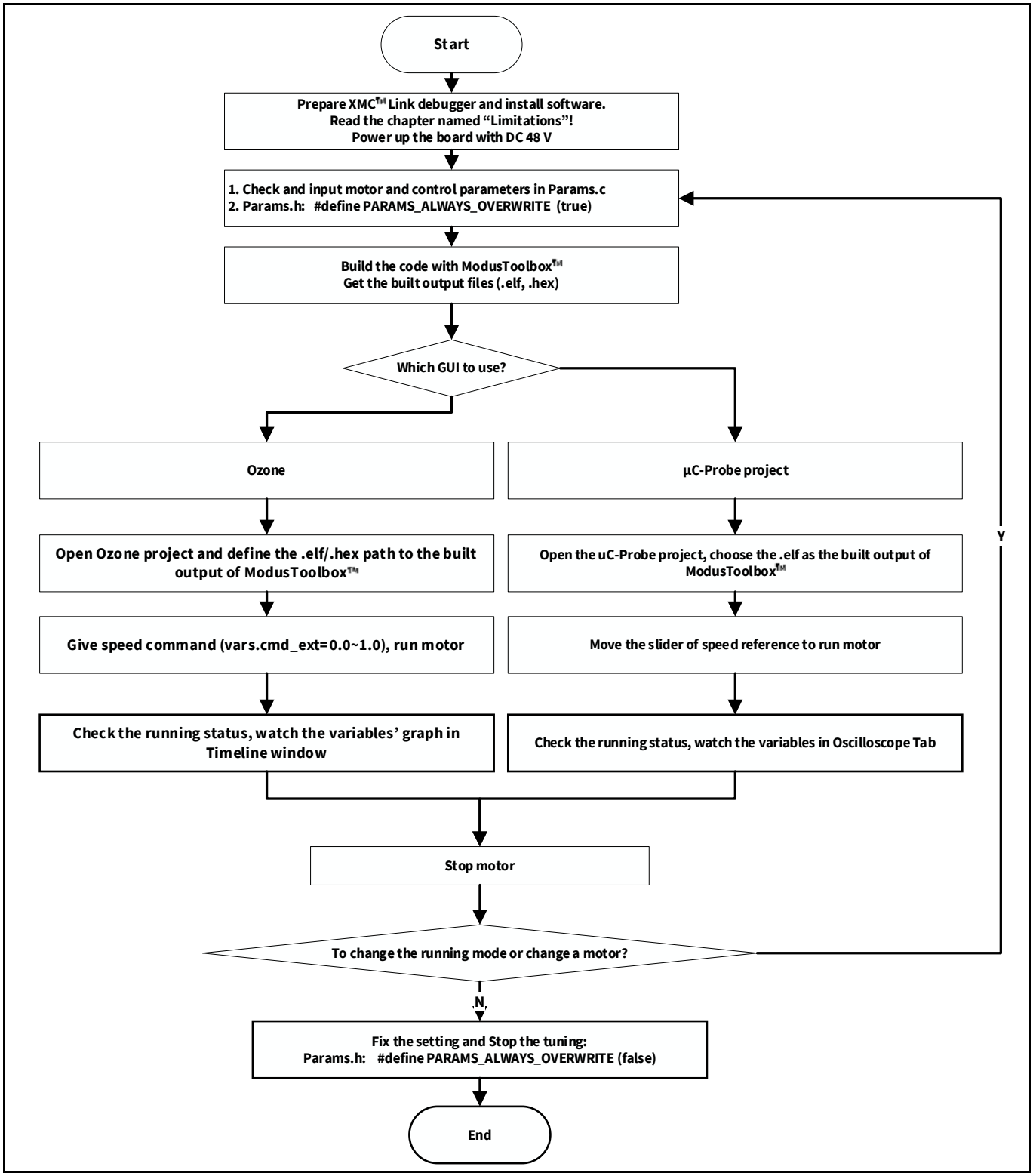

Figure 3 Overview of the motor running process

The P60 KV170 motor is tested and tuned to run. The following tuning methods are related to this motor unless otherwise mentioned. However, the same methods can be followed to tune other PMSM motors that this board can drive.

3.3.1 Motor parameters

The website of the P60 KV170 motor mentions the following motor parameters and load testing data.

Specification

Test Report			
Test Item	P60 KV170	Report NO.	P.00002
Specifications			
Internal Resistance	80mΩ	Configuration	24N28P
Shaft Diameter	6mm	Motor Dimensions	Φ69×37mm
Stator Diameter	62mm	Stator Height	15mm
AWG	16#	Cable Length	600mm
Weight Including Cables	375g	Weight Excluding Cables	345g
No.of Cells(Lipo)	6-14S	Idle Current@10v	1A
Max Continuous Power 180S	1800W	Max Continuous current 180S	38A

Figure 4 P60 KV170 motor specification

Load Testing Data

Ambient Temperature			18℃		Voltage			DC Power Supplier	
Item No.	Voltage (V)	Prop	Throttle	Current (A)	Power (W)	Thrust (G)	RPM	Efficiency (G/W)	Operating Temperature (℃)
P60 KV170	48	T-motor 20*6CF	50%	5.4	259.20	2116	4152	8.16	55
			55%	6.4	307.20	2371	4425	7.72	
			60%	7.7	369.60	2762	4709	7.47	
			65%	9.3	446.40	3125	5014	7.00	
			75%	13.2	633.60	4002	5626	6.32	
			85%	17.3	830.40	4821	6177	5.81	
			100%	25.4	1219.20	6246	6992	5.12	
		T-motor 22*6.6CF	50%	6.6	316.8	2801	3703	8.84	85
			55%	8.6	412.8	3312	4005	8.02	
			60%	9.9	475.2	3763	4289	7.92	
			65%	12.4	595.2	4356	4575	7.32	
			75%	17.1	820.8	5372	5091	6.54	
			85%	23.2	1113.6	6582	5635	5.91	
			100%	34	1632	8414	6374	5.16	

Notes:The test condition of temperature is motor surface temperature in 100% throttle while the motor run 10min.

Figure 5 P60 KV170 motor load testing data

Firmware

This info does not cover all the information that the firmware requires. It is recommended to use the following process to obtain the required information on motor parameters.

Note: The motor is fastened before performing the following tests.



Figure 6 P60 KV170 motor, fastened for testing

3.3.1.1 Stator resistance

Measure the phase-phase resistance with an LCR meter, and then get the phase resistance (Rs) from the average value.

– Rs = (68.78673+68.67645+67.91724) mΩ/3/2 = 68.46/2 mΩ = 34.23 mΩ

Code listing 5 Stator resistance

```
params.motor.r = 34.23E-3f; // [Ohm]
```



Figure 7 P60 KV170 motor stator resistance

3.3.1.2 q-axis inductance and d-axis inductance

Measure the phase-phase inductance with an LCR meter (rotate the motor rotor slowly and record the maximum and minimum inductance):

- Ld = 36.93929 µH/2 = 18.47 µH
- Lq = 53.23208 µH/2 = 26.616 µH

Code listing 6 q-axis inductance and d-axis inductance

```
params.motor.lq = 26.616E-6f; // [H]
params.motor.ld = 18.47E-6f; // [H]
```



Figure 8 P60 KV170 motor q-axis inductance and d-axis inductance

3.3.1.3 Permanent magnet's flux linkage and pole pairs

Connect a drill mechanically to the outer rotor of the motor, and then push the drill trigger to make the motor run. Use a speed meter to measure the actual motor speed. Use an oscilloscope probe to measure the phase-to-phase voltage.

When the motor running speed is 1681 rpm, the back emf (phase-to-phase voltage) is measured as shown in the following waveform. From the waveform, note that the electrical frequency is 391.6 Hz and the peak-peak value is 20.57 V when the motor is running at 1681 rpm.

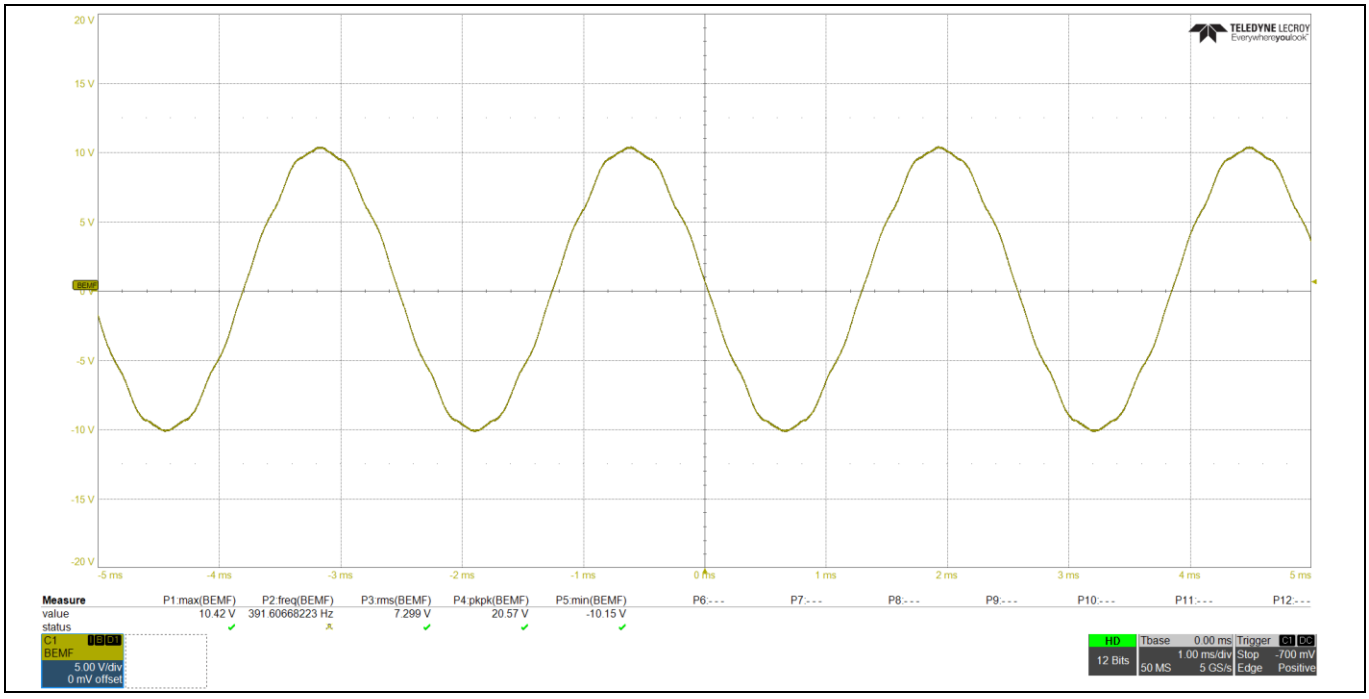

Figure 9 P60 KV170 motor back emf and pole pairs

Motor pole pairs:

– P = 60*f/n = 60*391.6 Hz/1681 rpm = 14

Permanent magnet's flux linkage (such as the back emf constant):

lam = Vphase-peak/ ω(Electrical speed at that rpm)

$$lam = \frac{\frac{20.57/2}{\sqrt{3}}}{2*\pi*391.6 rad/s} = 0.00241\ Wb$$

Equation 1 Calculating the permanent magnet's flux linkage

Code listing 7 Motor pole pairs and permanent magnet's flux linkage

```
params.motor.P = 28.0f; // [#]
params.motor.lam = 2.41E-3f; // [Wb]
```

3.3.1.4 Maximum torque

As the maximum torque is not provided from the website, you can calculate it from the power and speed:

$$T_{max} = \frac{9550*P}{n} = \frac{9550*1.632}{6374} = 2.44518 Nm$$

Equation 2 Calculating the maximum torque

Code listing 8 Motor maximum torque

```
params.motor.T_max = 2.4452f; // [Nm]
```

3.3.1.5 Mechanical parameters

There are three parameters.

Code listing 9 Mechanical parameters

```
params.mech.inertia = 2.757E-4f; // [kg.m^2]=[(N.m)/(Ra/sec-mech).sec],
=1/2*m*r^2
params.mech.viscous = 7.4525E-6f; // [kg.m^2/sec]=[(N.m)/(Ra/sec-mech)]
params.mech.friction = 3.3E-2f; // [kg.m^2/sec^2]=[N.m]
```

These parameters may not be available in the motor datasheet. Before getting these parameters, you should first run the motor at different speeds. Therefore, you should set the initial values into these parameters.

Note: Skip this section (Mechanical parameters), and follow the instructions in the next sections. Return to this section to fine tune these after motor can run stably at different speeds.

Firmware

Use the following example to obtain these parameters:

1. Set three speed references by setting `vars.cmd_ext` to three different values. Watch the graph of `vars.w_final_filt.elec` and `vars.T_est_filt` at each speed.

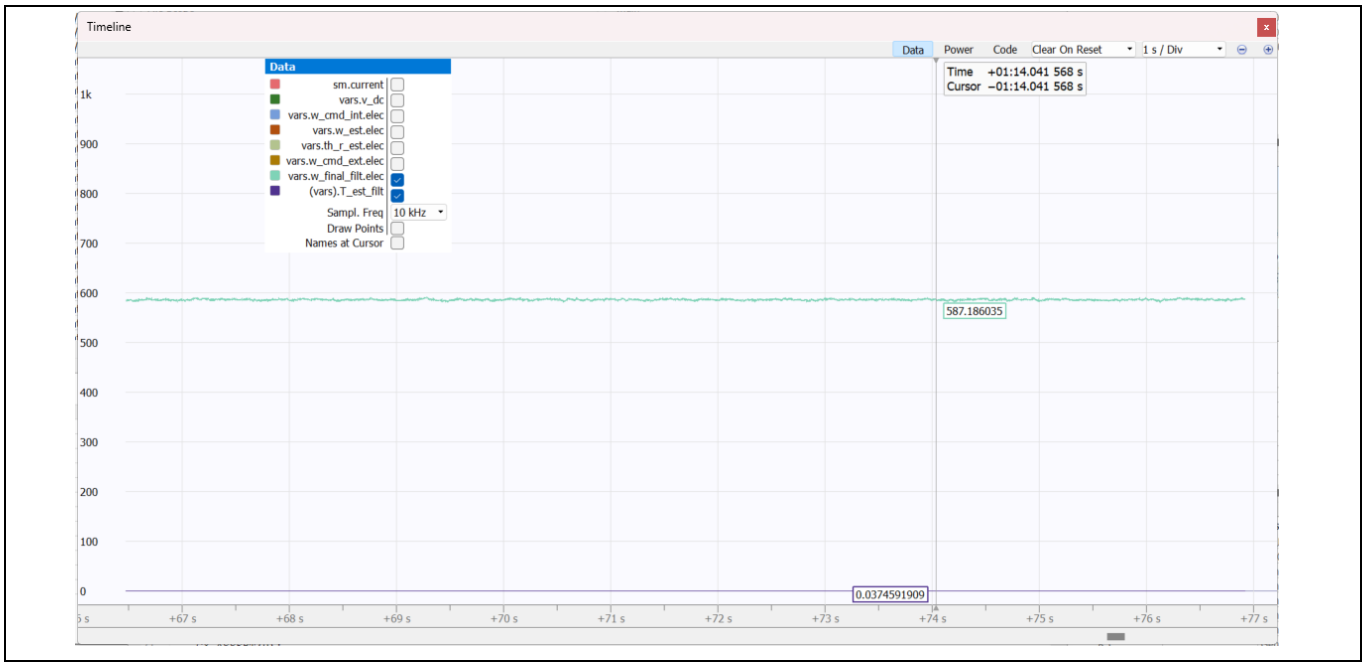

Figure 10 Motor running when vars.cmd_ext is set as 0.1

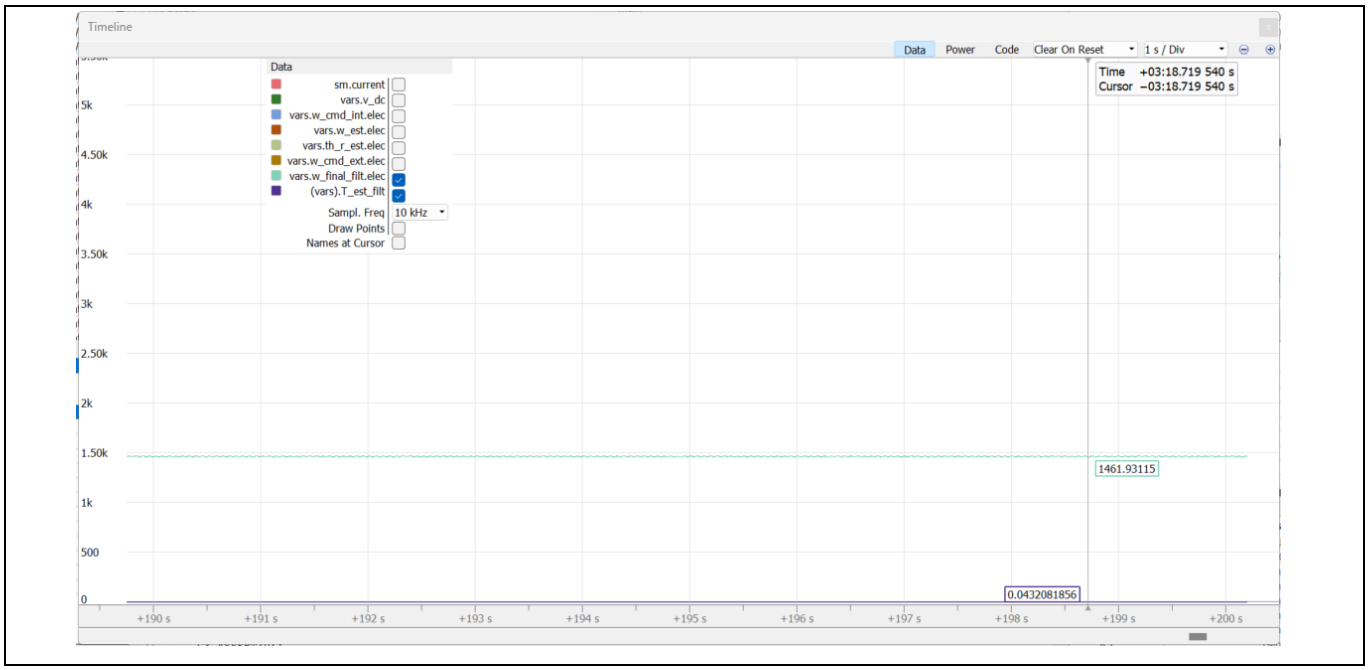

Figure 11 Motor running when vars.cmd_ext is set as 0.25

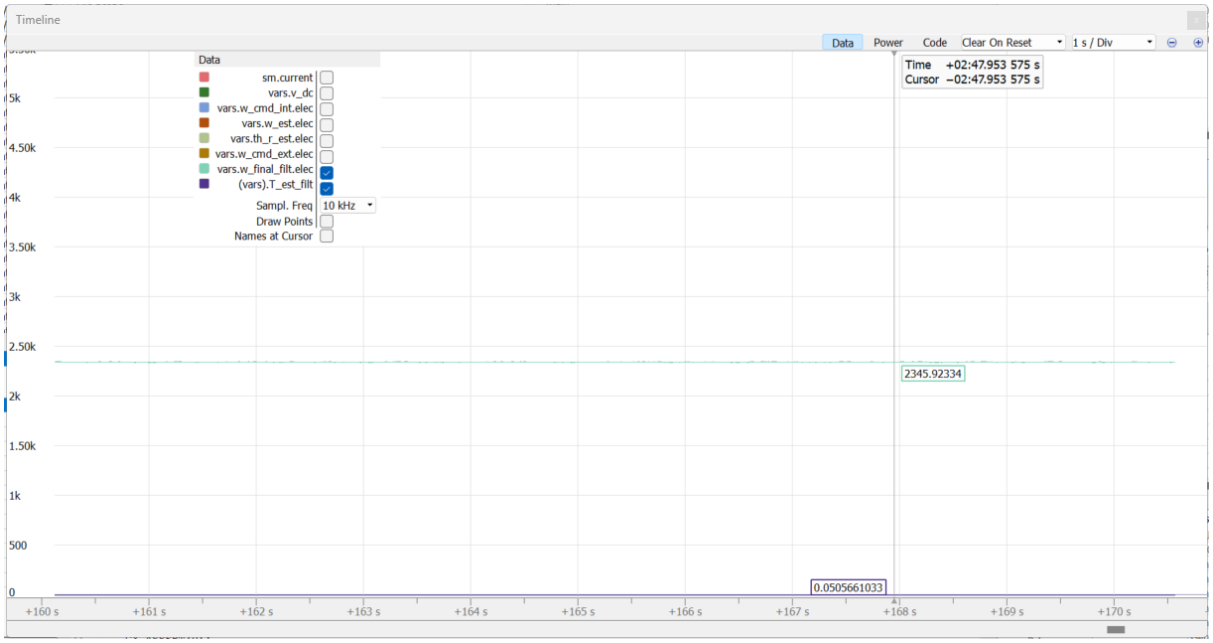

Figure 12 Motor running when vars.cmd_ext is set as 0.4

2. Pick the values at minimum and maximum speeds to calculate friction (Tf) and viscosity (B).

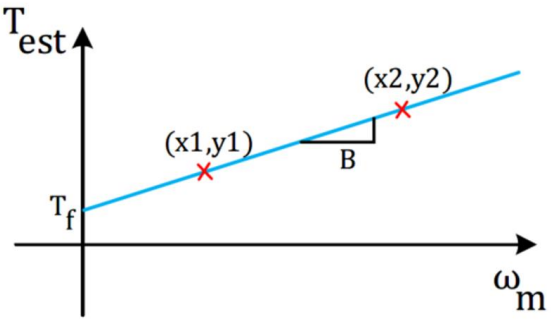

Figure 13 Graph necessary to extract Tf and B. Refer to Motor Control Firmware Reference Manual [6]

x1 = 587.186035 , y1 = 0.0374591909

X2 = 2345.92334 , y2 = 0.0505661033

$$\frac{y-y1}{x-x1}=\frac{y2-y1}{x2-x1}$$

$$\frac{y-0.0374591909}{x-587.186035}=\frac{0.0505661033-0.0374591909}{2345.92334-587.186035}$$

$$y=kx+b=\frac{y2-y1}{x2-x1}x+y1-x1*\frac{y2-y1}{x2-x1}$$

$$\frac{y-0.0374591909}{x-587.186035}=\frac{0.0505661033-0.0374591909}{2345.92334-587.186035}$$

$$y=kx+b=\frac{y2-y1}{x2-x1}x+y1-x1*\frac{y2-y1}{x2-x1}$$

Firmware

$$k = \frac{y2 - y1}{x2 - x1} = \frac{0.0505661033 - 0.0374591909}{2345.92334 - 587.186035} = \frac{0.0131069124}{1{,}758.737305}$$
$$= 7.4524560107627898414311510837032e - 6 \approx \mathbf{7.4525e - 6}$$
$$b = y1 - x1 * \frac{y2 - y1}{x2 - x1} = 0.0374591909 - 587.186035 * 7.4524560107627898414311510837032e - 6$$
$$= 0.0374591909 - 0.004375978095971719892528236330 33$$
$$= 0.033083212804028280107471763669 67 \approx \mathbf{0.033}$$

Then we get

$$y = (\mathbf{7.4525e - 6})\boldsymbol{x} + \mathbf{0.033}$$

Equation 3 Calculating Coulombic friction Tf and coefficient of viscous friction B

Therefore, **Tf = 0.033, B = 7.4525e-6**

Tf: Coulombic friction. In the firmware, it is represented as `params.mech.friction`.

B: Coefficient of viscous friction. In the firmware, it is represented as `params.mech.viscous`.

3. (Optional) Verify the above calculations by using the previous middle speed (`vars.cmd_ext` = 0.25) into the derived formula and calculate the torque. Compare the result and the actual torque to determine if these are nearly the same.

$$y = (\mathbf{7.4525e - 6})\boldsymbol{x} + \mathbf{0.033} = (\mathbf{7.4525e - 6}) * \mathbf{1461.93115} + \mathbf{0.033} = \mathbf{0.010895041895375} + \mathbf{0.033} = \mathbf{0.043895041895375}$$

Equation 4 Verification of the results

The actual value is 0.0432081856; this value is close to the calculated one.

4. Obtain inertia.

Follow this sequence and watch the graph of `vars.w_final_filt.elec` and `vars.T_est_filt` during this whole process.

1. Let the motor run at a low speed.
2. Enter a higher speed reference into `vars.cmd_ext` to let the motor accelerate.
3. Enter a lower speed reference into `vars.cmd_ext` to let motor decelerate.

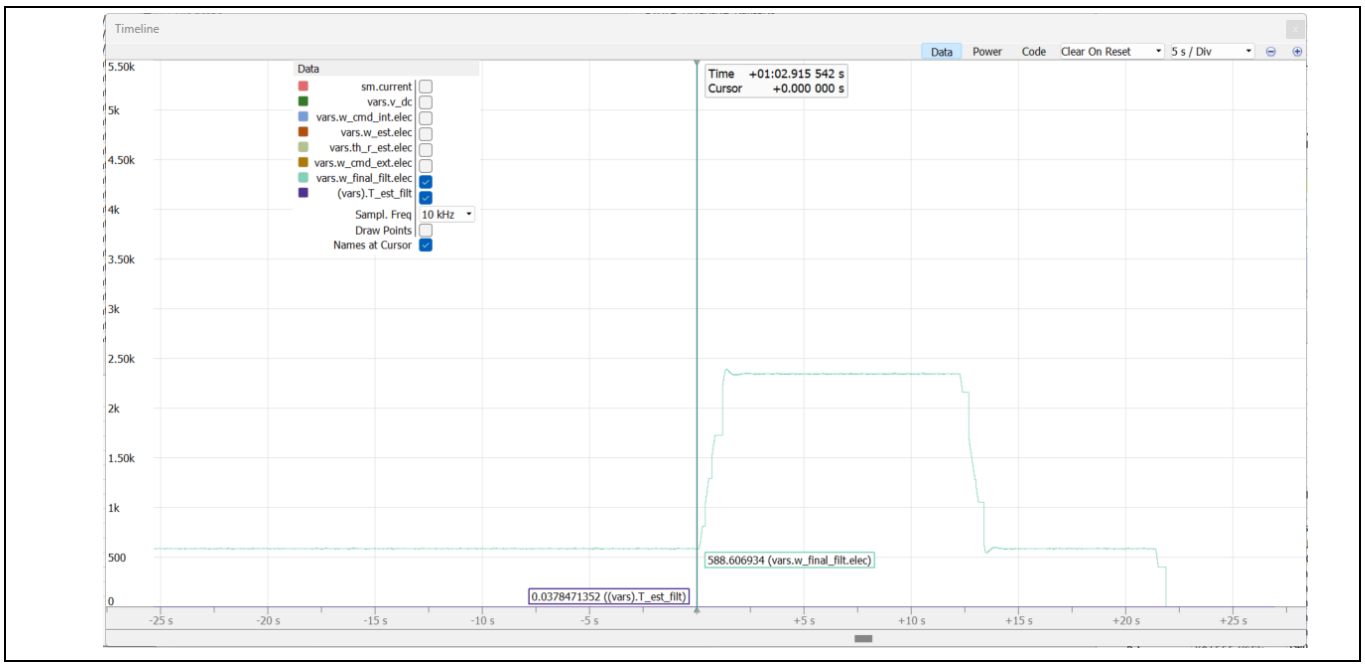

Figure 14 Acceleration starting point



Figure 15 Acceleration ending point

Firmware

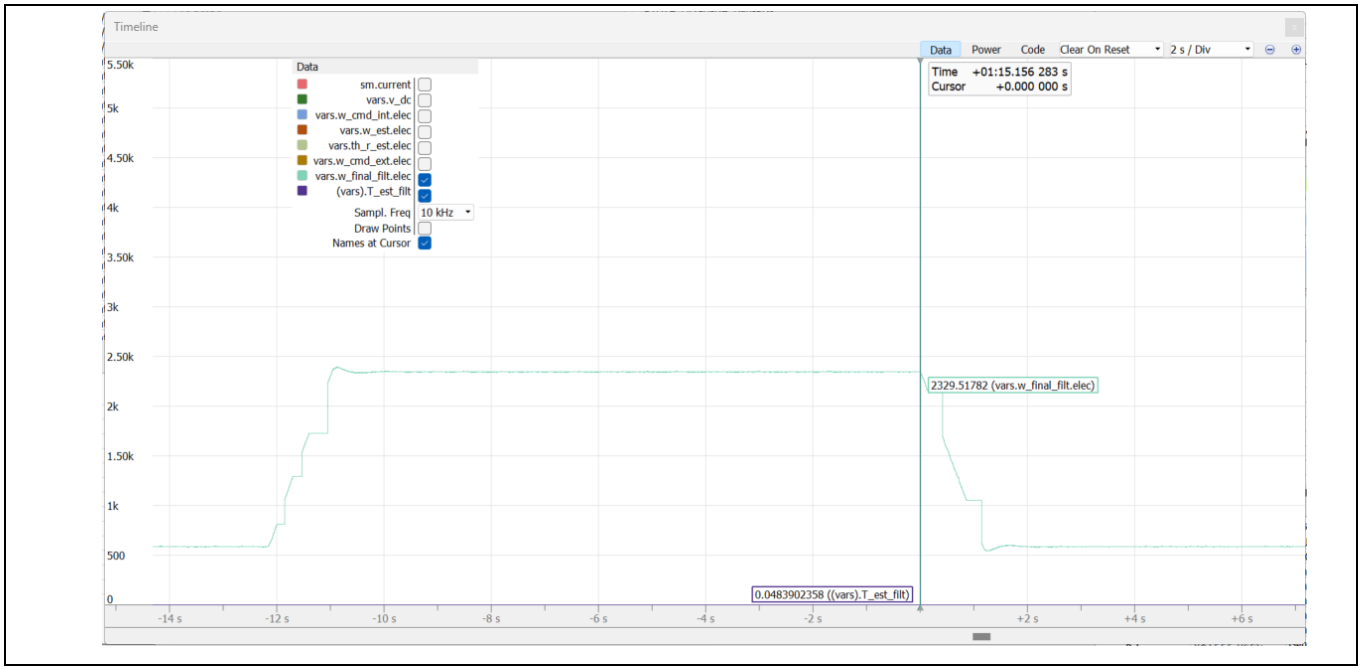

Figure 16 Deceleration starting point

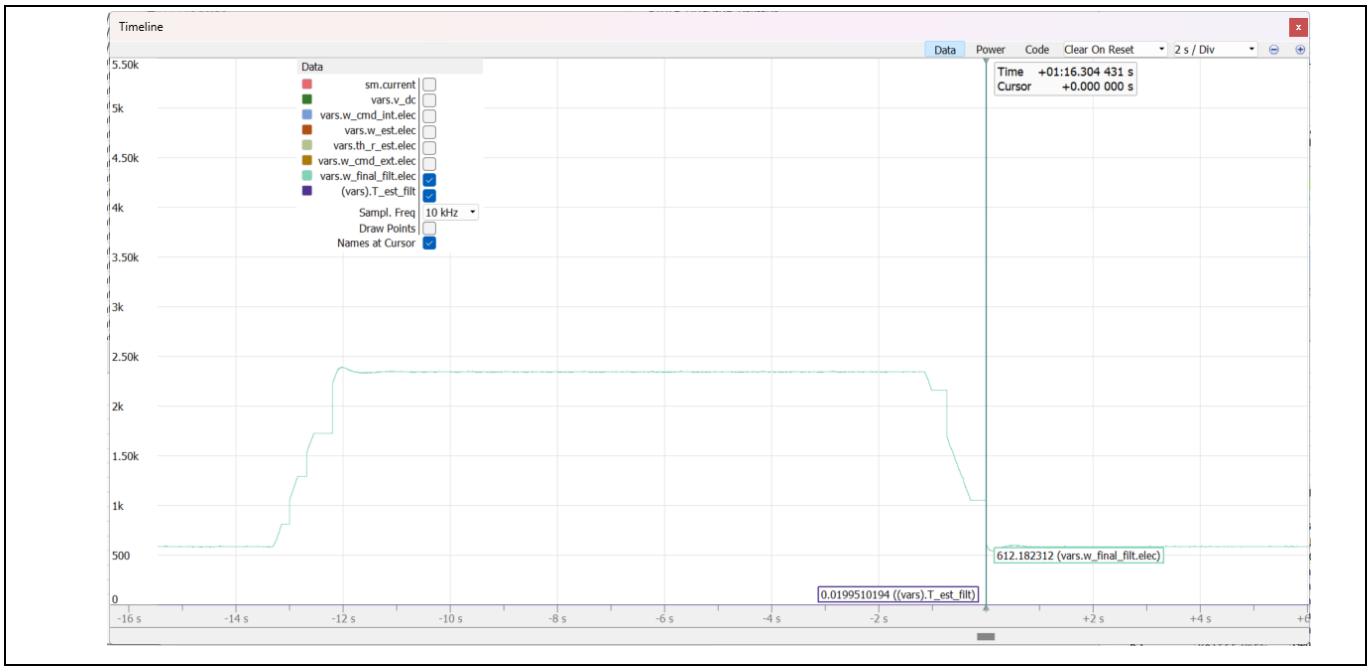

Figure 17 Deceleration ending point

Firmware

Calculate the inertia from the acceleration and deceleration separately and get the average result from these two values. Taking the values from Figure 14 through Figure 17 and substituting pole pairs = 14 into the formula:

1. Acceleration phase

$$t_2 = 4.150110s$$
$$t_1 = 2.915542s$$
$$\omega_{m2} = 2366.80518$$
$$\omega_{m1} = 588.606934$$
$$T_{est1} = 0.0378471352$$
$$T_{est2} = 0.0671903864$$

Calculating:

$$\frac{d\omega_m}{dt} = \frac{2366.80518 - 588.606934}{4.150110 - 2.915542s} * \frac{1}{14}$$

$$\boldsymbol{J_{acc}} = \frac{T_{est2} - T_{est1}}{\frac{d\omega_m}{dt}} = \frac{0.0671903864 - 0.0378471352}{\frac{2366.80518 - 588.606934}{4.150110 - 2.915542} * \frac{1}{14}} = \frac{0.0293432512 * 1.234568 * 14}{1{,}778.198246}$$
$$= 2.852141747443509737890046259780 2e - 4 \approx \boldsymbol{2.852E - 4f}$$

Equation 5 Calculating the acceleration phase inertia

2. Deceleration phase

$$t_1 = 15.156283s$$
$$t_2 = 16.304431s$$
$$\omega_{m2} = 612.182312$$
$$\omega_{m1} = 2329.51782$$
$$T_{est1} = 0.0483902358$$
$$T_{est2} = 0.0199510194$$

Calculating:

$$\frac{d\omega_m}{dt} = \frac{612.182312 - 2329.51782}{16.304431 - 15.156283} * \frac{1}{14}$$

$$\boldsymbol{J_{dec}} = \frac{T_{est2} - T_{est1}}{\frac{d\omega_m}{dt}} = \frac{0.0199510194 - 0.0483902358}{\frac{612.182312 - 2329.51782}{16.304431 - 15.156283} * \frac{1}{14}} = \frac{-0.0284392164 * 1.148148 * 14}{-1717.335508}$$
$$= 2.661879463317896994184784537 7e - 4 \approx \boldsymbol{2.662E - 4f}$$

Equation 6 Calculating the deceleration phase inertia

3. Calculate the final inertia by averaging J_{acc} and J_{dec}

$$J = \frac{J_{acc}+J_{dec}}{2} = \frac{2.852141747443509737890046259780 2e-4+2.661879463317896994184784537 7e-4}{2}$$

$$= 2.757010605380703366037415398 7e\text{-}4$$

$$\approx \mathbf{2.757}E-4f$$

Equation 7 Calculating the final inertia

This **J** is `params.mech.inertia` in the firmware.

3.3.1.6 Motor parameters set in firmware

By combining the output from above steps, the motor parameters are set into the firmware as shown in the following code. This snippet is in the `PARAMS_InitManual()` function in *Params.c*.

Code listing 10 Motor parameters set in PARAMS_InitManual() in Params.c

```
params.motor.P = 28.0f; // [#]
params.motor.lq = 26.616E-6f; // [H]
params.motor.ld = 18.47E-6f; // [H]
params.motor.lam = 2.41E-3f; // [Wb]
params.motor.r = 34.23E-3f; // [Ohm]
params.motor.T_max = 2.4452f; // [Nm]
params.motor.i_peak = 38.0f;    //38.0f; // [A]
params.motor.i_cont = 34.0f;    //34.0f; // [A]
params.motor.id_max = 17.0f; // [A]
params.motor.v_nom = LINE_TO_PHASE(48.0f); // [Vpk-ln]
params.motor.w_nom.elec = MECH_TO_ELEC(HZ_TO_RADSEC(RPM_TO_HZ(6374.0f)),
params.motor.P); // [Ra/sec-elec] 945Hz elec, 19.6Vdc
        params.motor.w_max.elec =
    MECH_TO_ELEC(HZ_TO_RADSEC(RPM_TO_HZ(7000.0f)), params.motor.P); //
    [Ra/sec-elec]
params.mech.inertia = 2.757E-4f;
params.mech.viscous = 7.4525E-6f;
params.mech.friction = 3.3E-2f;
```

3.3.2 Motor control mode

For Rotor Field Oriented (RFO) and speed mode, the firmware provides several different control modes that provide different startup methods and low-speed control. These control mode options are listed in *Params.h*.

Code listing 11 Control modes of RFO speed control in *Params.h*

```
Speed_Mode_FOC_Sensorless_Align_Startup,     // Closed-loop sensorless-foc
speed control with pre-alignment at startup
```

```
Speed_Mode_FOC_Sensorless_SixPulse_Startup,  // Closed-loop sensorless-foc
speed control with six pulse injection at startup
Speed_Mode_FOC_Sensorless_HighFreq_Startup,  // Closed-loop sensorless-foc
speed control high frequency injection at startup
Speed_Mode_FOC_Sensorless_Volt_Startup,            // Closed-loop
sensorless-foc speed control with open-loop V/Hz at startup
```

Set `params.ctrl.mode` to the mode you want. For example, the following code sets the mode to Volt startup:

Code listing 12 Set the control mode in PARAMS_InitManual() in *Params.c*

```
// Control Mode:
#if defined(CTRL_METHOD_RFO) || defined(CTRL_METHOD_SFO)
    params.ctrl.mode = Speed_Mode_FOC_Sensorless_Volt_Startup;
```

It is recommended to start tuning the motor from Volt startup (`Speed_Mode_FOC_Sensorless_Volt_Startup`), which starts the motor in the open-loop mode and transitions to FOC sensorless mode after the speed reaches a threshold.

3.3.2.1 Speed_Mode_FOC_Sensorless_Volt_Startup

In this mode, you can check the basic running of the motor in the open loop, and tune the speed-observer algorithm for sensorless FOC. Once the speed-observer is tuned to a good state, the tuning of other control modes becomes easier.

Specifically, the tuning can be divided into several steps:

1. Tune the basic running in an open loop.
   a) Set the appropriate parameters for startup and stable running at low speed.
   b) Check the motor phase current sampling.
2. Check the observer behavior.
3. Fine-tune the open loop to FOC sensorless transition.
4. Fine-tune the parameters.

3.3.2.1.1 Tune the open loop

For tuning the open loop, the target is to let the motor start rotating while ensuring that the motor current is under a safe value to avoid overheating.

Code listing 13 Open-loop parameters in PARAMS_InitManual() of Params.h

```
params.ctrl.volt.v_min = 4.0E-2f * 2.0f; // [Vpk]
params.ctrl.volt.v_to_f_ratio = 2.97E-3f;//5.8E-3f * 1.5f; // [Vpk/(Ra/sec-
elec)]
```

To start with, the initial values of these two parameters can be set as follows:

$$\text{params.ctrl.volt.v_min} = \frac{(\text{params.motor.r} * \text{params.motor.i_cont})}{k}$$
$$k: 3\sim10$$

$$\text{params.ctrl.volt.v_to_f_ratio} = \frac{\text{params.motor.v_nom}}{\text{params.motor.w_nom.elec}}$$

Figure 18 Tentative starting values of open loop

It is recommended to fix the motor to run in an open loop and prevent it from transitioning to sensorless FOC mode because the observer is not fine-tuned yet. This can be realized by two methods:

- Set the target speed to be lower than `params.obs.w_thresh.elec`.
  - Set `vars.cmd_ext` to a smaller percentage of the nominal speed compared to `params.obs.w_thresh.elec`.
- Fix the state machine to `Volt_Hz_OL`.
  - Comment out the code of transition to `Speed_OL_To_CL`; see the following code:

Code listing 14 Fix state machine to *Volt_Hz_OL* in *ConditionCheck()* in StateMachine.c

```
else if (w_cmd_int_above_thresh_high &&
(Mode(Speed_Mode_FOC_Sensorless_Volt_Startup) ||
Mode(Motor_Profiler_Mode)))
{
//sm.vars.speed_reset_required = true;
      //next = Speed_OL_To_CL;
}
```

When tuning, set the final speed reference to a relatively low value. For example, set `vars.cmd_ext` to 0.1 (its range is 0 ~ 1.0) to avoid a larger motor current at high speed. Run the motor and keep watching the actual motor phase current. If the motor cannot start or cannot overcome the initial friction and load, increase `params.ctrl.volt.v_min`. Once the motor starts successfully, this value can be decreased to an appropriate value depending on the load.

In the same way, if the motor cannot accelerate smoothly, increase `params.ctrl.volt.v_to_f_ratio` slightly. If the motor phase current is too high, decrease this value.

Firmware

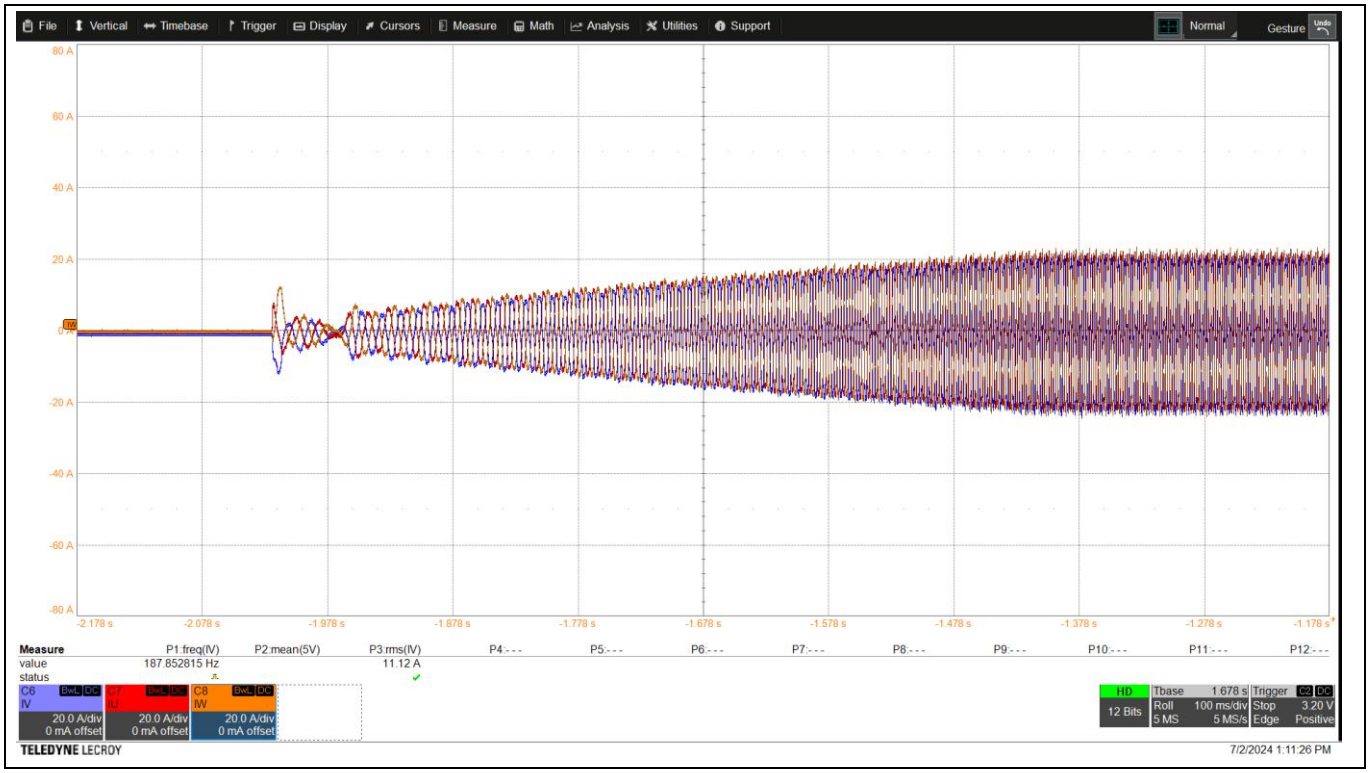

Figure 19 Example of Speed_Mode_FOC_Sensorless_Volt_Startup

Check the motor phase current after the motor runs stably at the speed reference, and then compare the actual motor phase current with the sampled values in firmware. These should be the same. The following is an example.

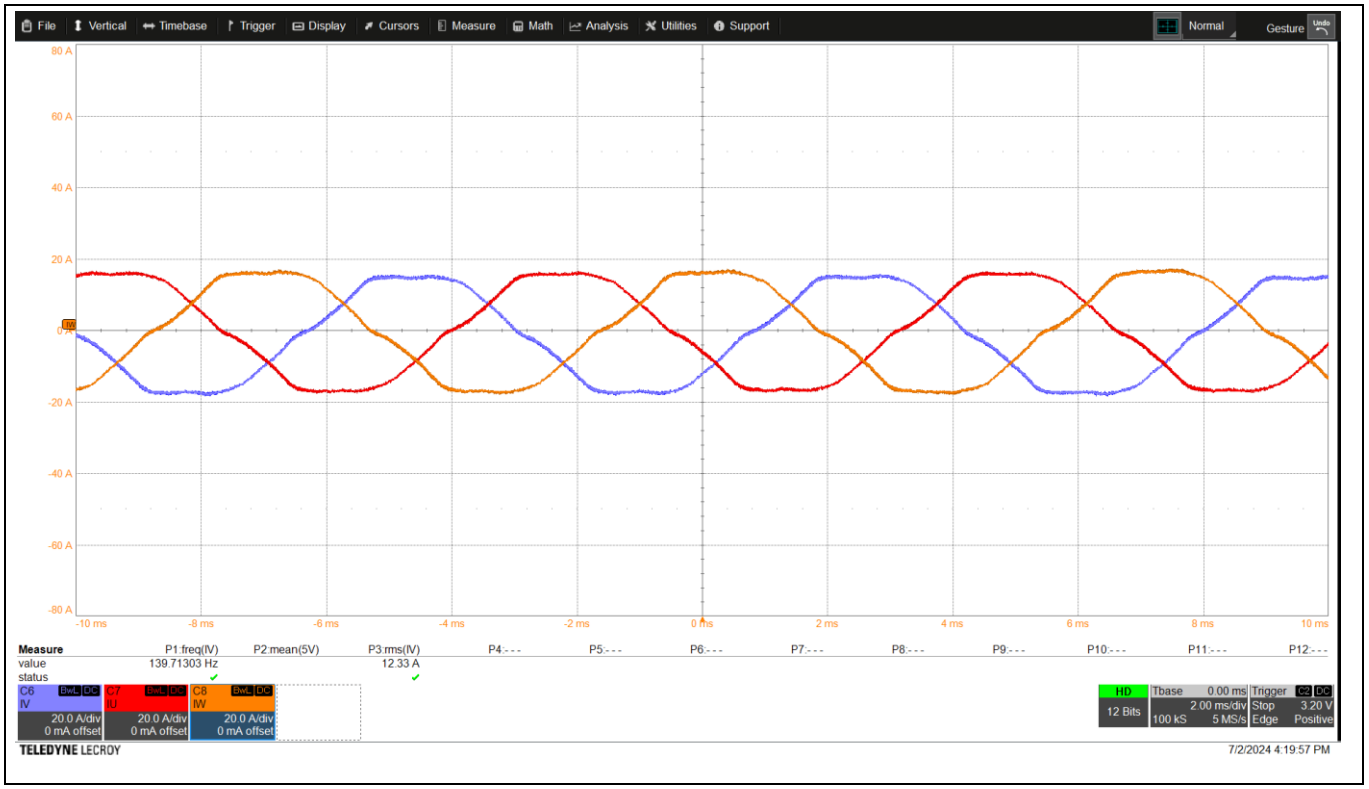

Figure 20 Example of actual motor phase current running in open loop

The sampled currents are in the following variables:

- `sensor_iface.i_u.raw`
- `sensor_iface.i_v.raw`
- `sensor_iface.i_w.raw`

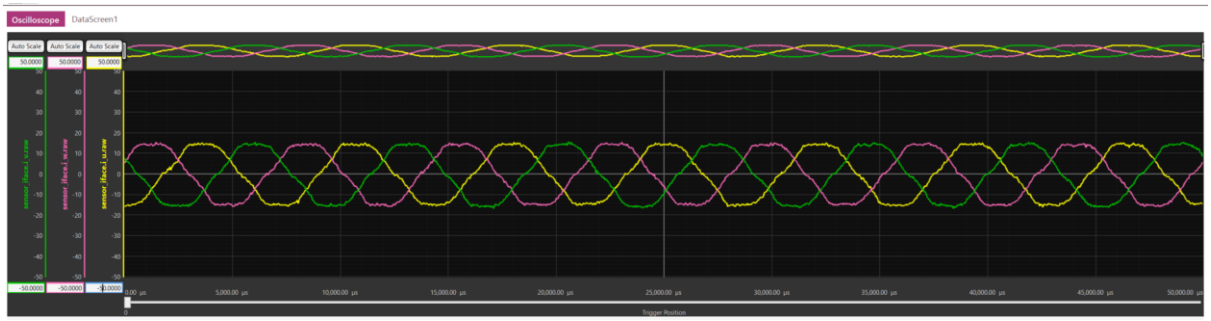

Figure 21 Example of sampled motor phase current in firmware when running in open loop

3.3.2.1.2 Check observer behavior

If the speed reference is set higher than `params.obs.w_thresh.elec`, normally the state machine transition of `Speed_Mode_FOC_Sensorless_Volt_Startup` would behave as follows:

Brake_Boot -> Volt_Hz_OL -> Speed_OL_To_CL-> Speed_CL

There is an intermediate state of `Speed_OL_To_CL` before transiting from open loop to speed close loop. During this state, the speed observer starts to observe the rotor angle and speed concurrently with the motor running in an open loop. It is perfect for checking and tuning the observer in this state.

Fix the state machine in `Speed_OL_To_CL` state by commenting out the following code:

Code listing 15 Fix state machine to *Speed_OL_To_CL* in *ConditionCheck()* in StateMachine.c

```
else if (StopWatchIsDone(&sm.vars.speed_ol_to_cl.timer))
{
//next = Speed_CL;
}
```

From the actual current running stably at `Speed_OL_To_CL` state as follows, the frequency of current (or electrical frequency) is 112 Hz.

$$\omega = 2 * \pi * 112 = 703.7\, rad/s$$

Equation 8 Stable current frequency

Firmware

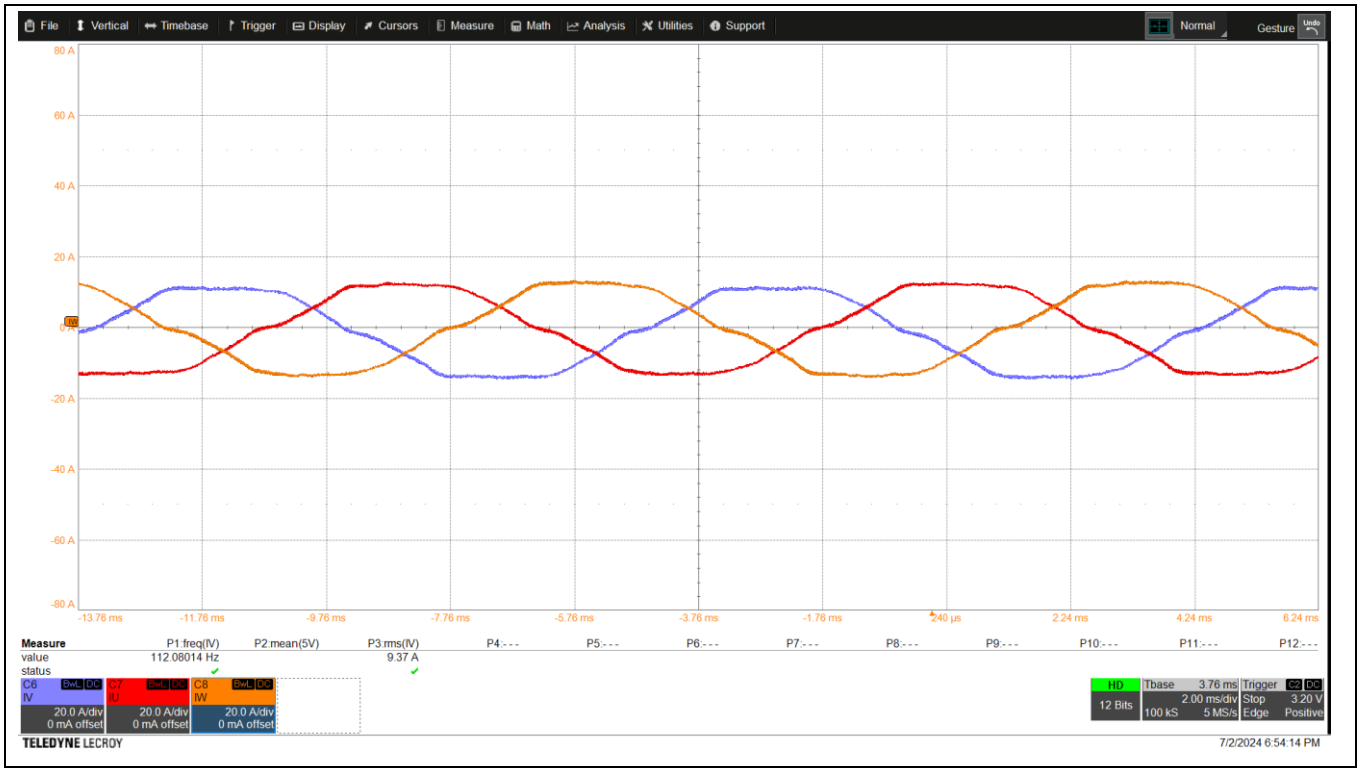

Figure 22 Actual motor phase current running in *Speed_OL_To_CL*

The following waveform is the estimated angle (`vars.th_r_est.elec`) and estimated speed (`obs.pll_r.w.elec`) in the firmware. The estimated angle is a regular sawtooth wave; the average of the estimated speed is 703.73, which is close to the actual speed. Thus, the motor is ready to transition to the speed loop.

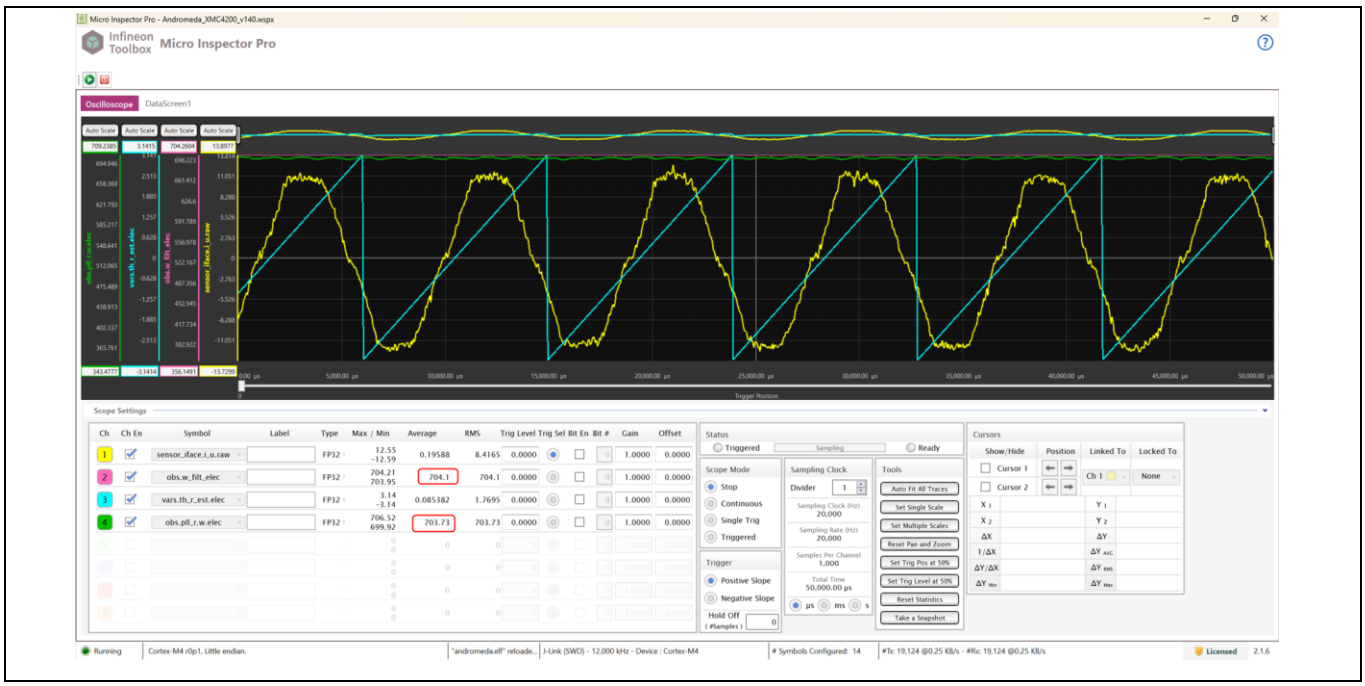

Figure 23 Observed angle and speed running in *Speed_OL_To_CL*

Uncomment the following code to let the state machine go to the `Speed_CL` state after the `Speed_OL_To_CL` state completes.

Code listing 16 Transition to speed loop in *ConditionCheck()* in StateMachine.c

```
Else if (StopWatchIsDone(&sm.vars.speed_ol_to_cl.timer))
{
next = Speed_CL;
}
```

`params.obs.lock_time` can be adjusted to set the running time in the `Speed_OL_To_CL` state; this parameter ensures that the observer gets enough time to work normally before the transition to speed loop.

Code listing 17 Tune the observer locking time in *PARAMS_InitManual()* in Params.c

```
params.obs.lock_time = 0.300f; // [sec], increase when transition to
closed-loop is bumpy, especially when using single shunt in SFO
```

3.3.2.2 Speed_Mode_FOC_Sensorless_Align_Startup

For this mode, the key tuning target is to let the motor rotor align to a known position before transitioning to the speed loop with the feedback speed from the observer. Normally, increase the `params.ctrl.align.time` to a suitable value to ensure that the motor rotor indeed goes to the defined position and stays at that position before transition; see the following code:

Code listing 18 Tune the observer locking time in *PARAMS_InitManual()* in Params.c

```
// Align (Pre-Positioning) Parameters:
params.ctrl.align.time = 0.25f; // [sec]
params.ctrl.align.voltage = (params.motor.r * params.motor.i_peak); //
[Vpk]
```

After the motor starts rotating, the observer works and the previous tuning is applied.

Firmware

An example of the motor phase current waveform for this mode is as follows:

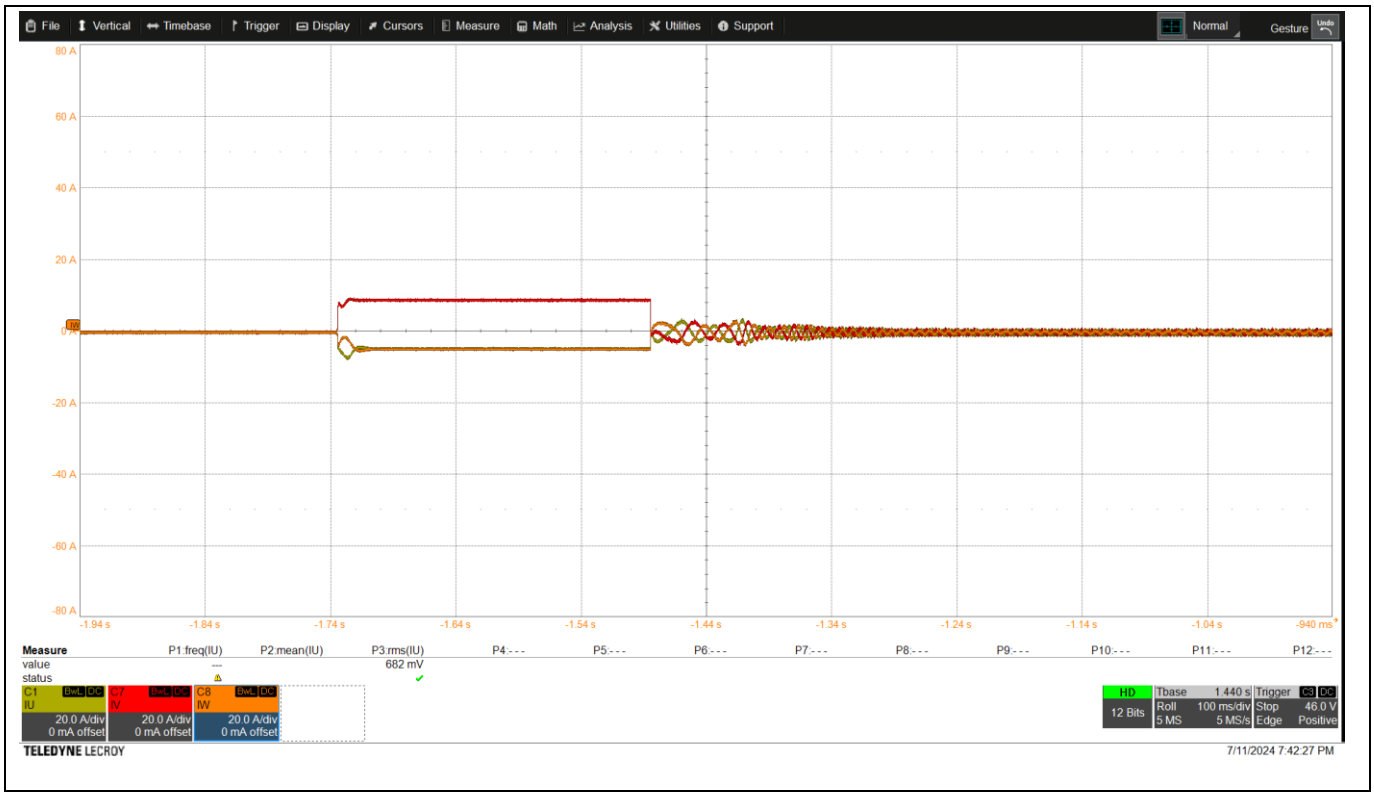

Figure 24 Example of motor phase current in *Speed_Mode_FOC_Sensorless_Align_Startup*

3.3.2.3 Speed_Mode_FOC_Sensorless_SixPulse_Startup

This mode is similar to `Speed_Mode_FOC_Sensorless_Align_Startup`. The difference is that for six-pulse startup, the rotor initial angle is identified rather than being pulled to a given position.

For this mode to work successfully, the applied voltage and the time duration to the motor (thus the current) for each of the six pulses should be appropriate. Ensure that the motor rotor does not move whereas the current is high enough during the six-pulse process to identify the initial angle successfully. The key is tuning the following parameter:

Code listing 19 Tune the six-pulse injection in *PARAMS_InitManual()* in Params.c

```
// Six Pulse Injection Parameters:
params.ctrl.six_pulse_inj.i_peak = 17.0f;//1.75f; // [A], this is the
target value, not the actual value
```

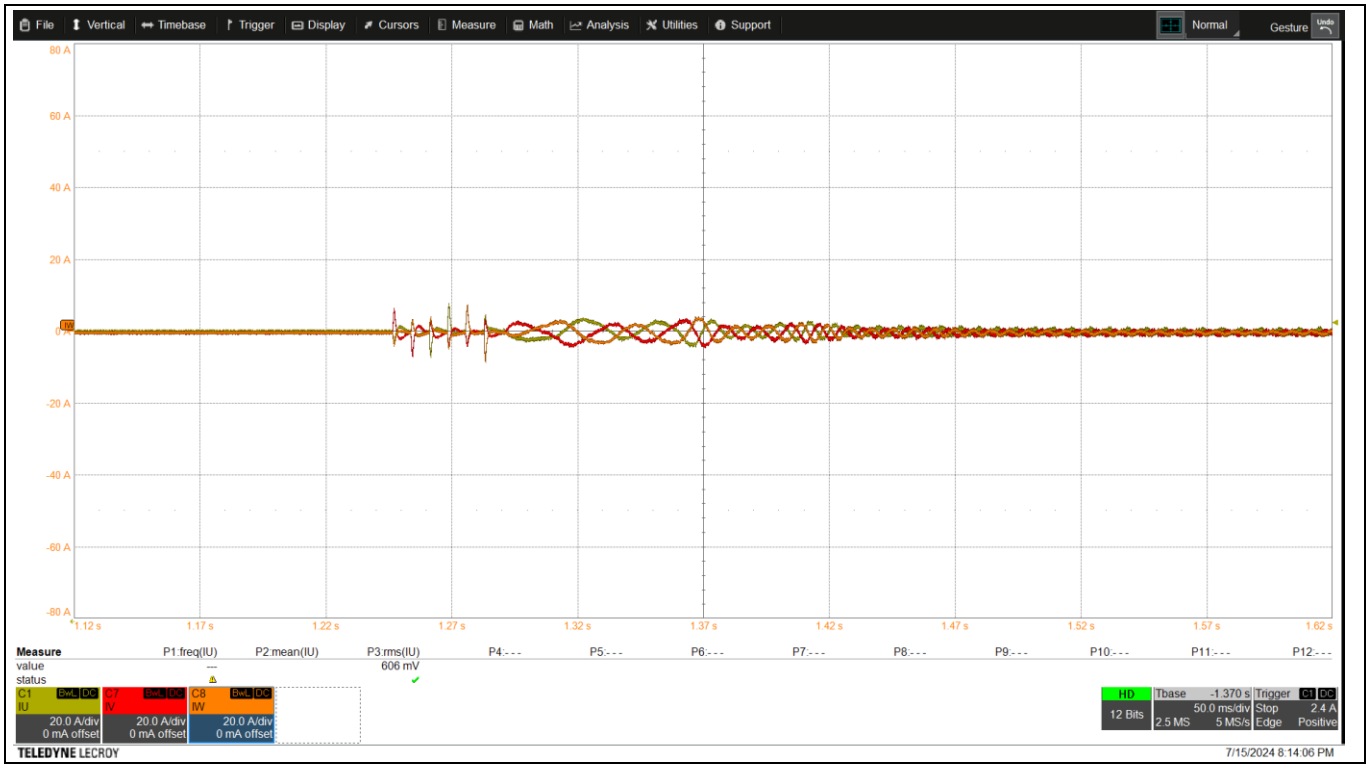

Figure 25 Example of motor phase current in Speed_Mode_FOC_Sensorless_SixPulse_Startup

Firmware

The following is the related change when `params.ctrl.six_pulse_inj.i_peak` is changed to 1.75 A, 5 A, 15 A, and 20 A. When the injected current is increased, the `v_pulse` (thus the applied voltage) is increased, `d_pos` (positive duty cycle) is increased, and `d_neg` (negative duty cycle) is decreased. `t_on` and `t_off` are not changed; this is automatically calculated and determined by the motor time constant.

1.75 A:

Expression	Value
sensor_iface.i_u.raw	0.167846695
vars.cmd_ext	0
vars.v_dc	47.8572807
sm.vars.fault.clr_request	0
⊞ vars.d_uvw_cmd	
faults.flags_latched.all	0
ctrl.six_pulse_inj.d_pos	0.501480699
params.ctrl.six_pulse_inj.v_pulse	0.142146558
params.ctrl.six_pulse_inj.t_off	0.00659999996
params.ctrl.six_pulse_inj.t_on	0.000650000002
ctrl.six_pulse_inj.d_neg	0.498519301

5 A:

Watched Data 1

Expression	Value	Location	Refresh
sensor_iface.i_u.raw	-0.772094786	1FFF E02C	off
vars.cmd_ext	0	1FFF D608	off
vars.v_dc	47.8539124	1FFF D730	off
sm.vars.fault.clr_request	0	1FFF E251	off
vars.d_uvw_cmd		1FFF D71C	off
faults.flags_latched.all	0	1FFF D740	off
ctrl.six_pulse_inj.d_pos	0.504230559	1FFF D45C	off
params.ctrl.six_pulse_inj.v_pulse	0.406132966	1FFF DEEC	off
params.ctrl.six_pulse_inj.t_off	0.00659999996	1FFF DEE8	off
params.ctrl.six_pulse_inj.t_on	0.000650000002	1FFF DEE4	off
ctrl.six_pulse_inj.d_neg	0.495769441	1FFF D460	off

15 A:

ctrl.six_pulse_inj.d_pos	0.512691677
params.ctrl.six_pulse_inj.v_pulse	1.21839905
params.ctrl.six_pulse_inj.t_off	0.00659999996
params.ctrl.six_pulse_inj.t_on	0.000650000002
ctrl.six_pulse_inj.d_neg	0.487308353

20 A:

sensor_iface.i_u.raw	0.234985381
vars.cmd_ext	0
vars.v_dc	47.8553391
sm.vars.fault.clr_request	0
⊞ vars.d_uvw_cmd	
faults.flags_latched.all	0
ctrl.six_pulse_inj.d_pos	0.516922235
params.ctrl.six_pulse_inj.v_pulse	1.62453187
params.ctrl.six_pulse_inj.t_off	0.00659999996
params.ctrl.six_pulse_inj.t_on	0.000650000002
ctrl.six_pulse_inj.d_neg	0.483077794

Figure 26 Variables' change when increasing *params.ctrl.six_pulse_inj.i_peak*

See the following code for auto calculation of these parameters:

Code listing 20 Auto calculation of six pulse in *PARAMS_InitAutoCalc()* in Params.c

```
// Six Pulse Injection Parameters:
float motor_l_ave = 0.5f * (params.motor.lq + params.motor.ld); // [H]
float motor_tau_ave = motor_l_ave / params.motor.r; // [sec]
params.ctrl.six_pulse_inj.t_on = QUANTIZE_FLOAT(motor_tau_ave,
params.sys.samp.ts0); // [sec], to maximize the difference between currents
params.ctrl.six_pulse_inj.t_off = QUANTIZE_FLOAT(10.0f * motor_tau_ave,
params.sys.samp.ts0); // [sec], enough time for current to go to zero
params.ctrl.six_pulse_inj.v_pulse = (1.5f / (1.0f - EXP_MINUS_ONE)) *
params.motor.r * params.ctrl.six_pulse_inj.i_peak; // [V], dc+ to dc-
```

When `params.ctrl.six_pulse_inj.i_peak` is set to 20 A, the motor rotor would move during the six-pulse process, making the identification of the initial rotor position fail. Thus, the final value of this parameter is set to 17 A.

3.3.2.4 Speed_Mode_FOC_Sensorless_HighFreq_Startup

Before setting the control mode to `Speed_Mode_FOC_Sensorless_HighFreq_Startup`**, you must tune** `Speed_Mode_FOC_Sensorless_ SixPulse_Startup` **successfully**. This is because the HighFreq startup would also inject six pulses initially to get the initial rotor angle and use that result as a seed to process the HighFreq algorithm.

Normally the state machine transition of `Speed_Mode_FOC_Sensorless_HighFreq_Startup` would behave as follows:

Brake_Boot -> Six_Pulse -> High_Freq -> Speed_CL

This state of `High_Freq` lasts for a duration of `params.ctrl.high_freq_inj.lock_time`, which is determined by `params.ctrl.high_freq_inj.pll.w0` in the `PARAMS_InitAutoCalc()` function in *Params.c*.

After the state transitions to *Speed_CL*, there is an additional transition inside this state to estimate the angle and speed for the feedback in the speed loop, those states are: high frequency injection or observer.
It can be divided into two phases:

Phase 1:

`sm.vars.high_freq.used` is true. In this phase, both high-frequency injection and observer are running. Both generate angle and speed, but only the angle and speed of high-frequency injection are used.

```
vars.w_est.elec = high_freq_inj->pi_pll_r.output;
vars.th_r_est.elec = high_freq_inj->integ_pll_r.integ;
```

Firmware

Phase 2:

`sm.vars.high_freq.used` is false. After the estimated speed (`vars.w_final_filt.elec`) exceeds the observer working speed threshold (`params.obs.w_thresh.elec`), `sm.vars.high_freq.used` is set to false, then high-frequency injection stops and only the observer keeps working.

```
vars.th_r_est.elec = obs.pll_r.th.elec;
vars.w_est.elec = obs.pll_r.w.elec;
```

Code listing 21 Angle/speed transition in Speed_CL state in StateMachine.c

```
static void SensorlessFeedbackISR0()
{
// Using threshold hysteresis
if ((sm.vars.high_freq.used) && ABS_ABOVE_LIM(vars.w_final_filt.elec,
params.obs.w_thresh.elec))
{
sm.vars.high_freq.used = false;
#if defined(CTRL_METHOD_RFO)
CURRENT_CTRL_Init(1.0f);
#elif defined(CTRL_METHOD_SFO)
FLUX_CTRL_Init(1.0f);
ctrl.delta.bw_red_coeff = 1.0f;
#endif
}

if (!sm.vars.high_freq.used)
{
OBS_RunISR0();
}
else // running both observer and high frequency injection
{
HIGH_FREQ_INJ_RunFiltISR0();
OBS_RunISR0();
HIGH_FREQ_INJ_RunCtrlISR0();
}
…
…
}
```

To start tuning, the parameters related to high-frequency injection are as follows. Pay special attention to the tuning of `params.ctrl.high_freq_inj.pll.w0` and `params.ctrl.high_freq_inj.i_qd_r_peak.d` to improve the correctness of the estimated angle/speed from high-frequency injection.

Code listing 22 High-frequency injection parameters in *PARAMS_InitManual()* in Params.c

```
// High Frequency Injection Parameters:
params.ctrl.high_freq_inj.w_h = HZ_TO_RADSEC(1500.0f); // [Ra/sec], at
least 1 decade below switching frequency
params.ctrl.high_freq_inj.w_sep = HZ_TO_RADSEC(150.0f); // [Ra/sec], at
least 1 decade below switching frequency
params.ctrl.high_freq_inj.pll.w0 = HZ_TO_RADSEC(40.0f); // [Ra/sec]
params.ctrl.high_freq_inj.pll.w_max.elec = 0.5f *
(params.ctrl.high_freq_inj.w_h - params.ctrl.high_freq_inj.w_sep); //
[Ra/sec-elec]
params.ctrl.high_freq_inj.i_qd_r_peak.d = 6.0f; //0.25f; // [A], excitation
current peak (d-axis), limiting factor (larger than q-axis)
```

Also, the observer starting speed and hysteresis may need to be fine-tuned to get a smooth transition from high-frequency injection to observer when accelerating.

Code listing 23 Observer starting speed threshold in *PARAMS_InitManual()* in Params.c

```
params.obs.w_thresh.elec = params.motor.w_nom.elec * 0.05f; // [Ra/sec-
elec]
params.obs.w_hyst.elec = params.obs.w_thresh.elec * 0.5f; //0.08f; //
[Ra/sec-elec]
```

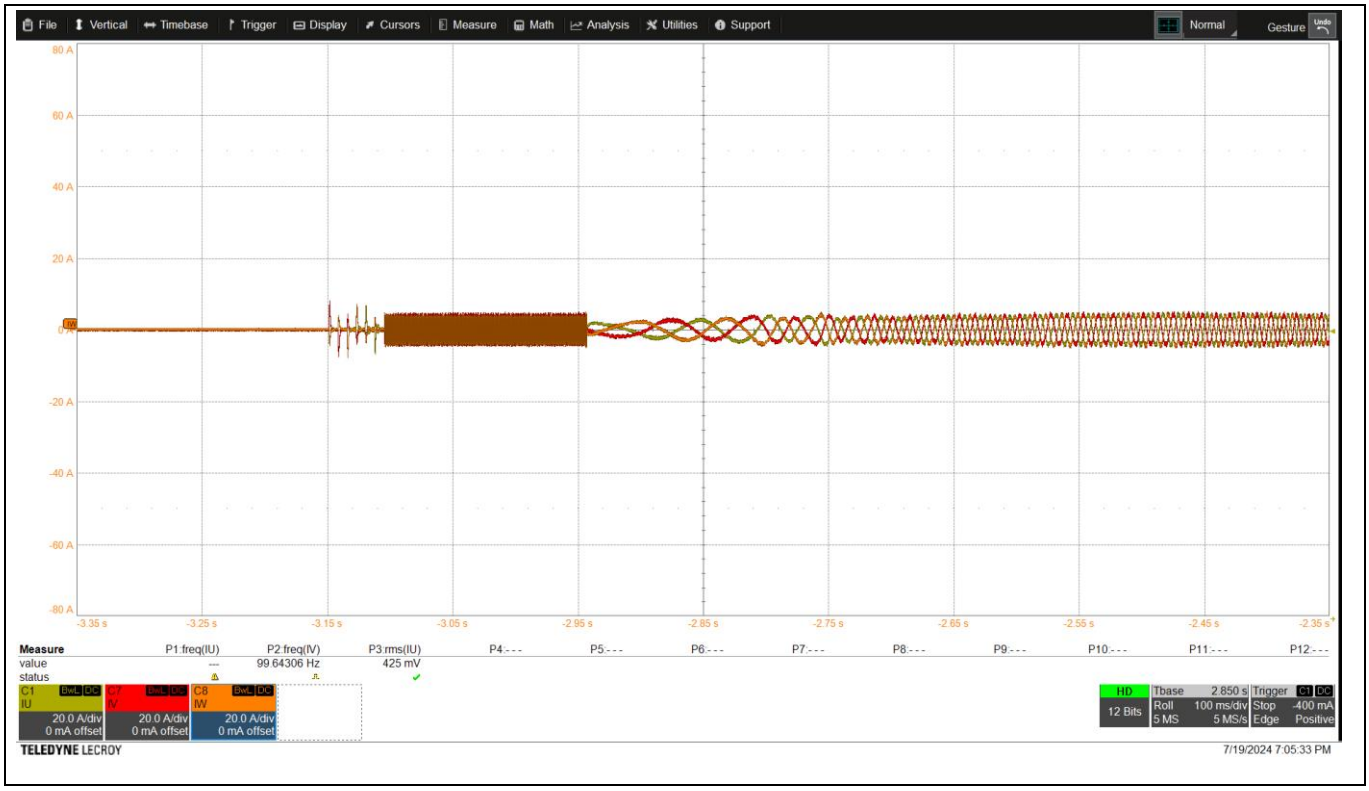

Figure 27 Example of Speed_Mode_FOC_Sensorless_HighFreq_Startup

3.4 GUI

Note: Ensure that the board is powered up and XMC™ Link is connected.

1. Open the workspace file (.wspx) with Micrium µC/Probe.
2. Click the green run button to start the GUI application.

There are two tabs: **Dashboard** and **Oscilloscope**.

3. Move the slider under **Speed reference** or just input a percentage to change the speed. You can monitor the status and variables in the **Oscilloscope** tab.

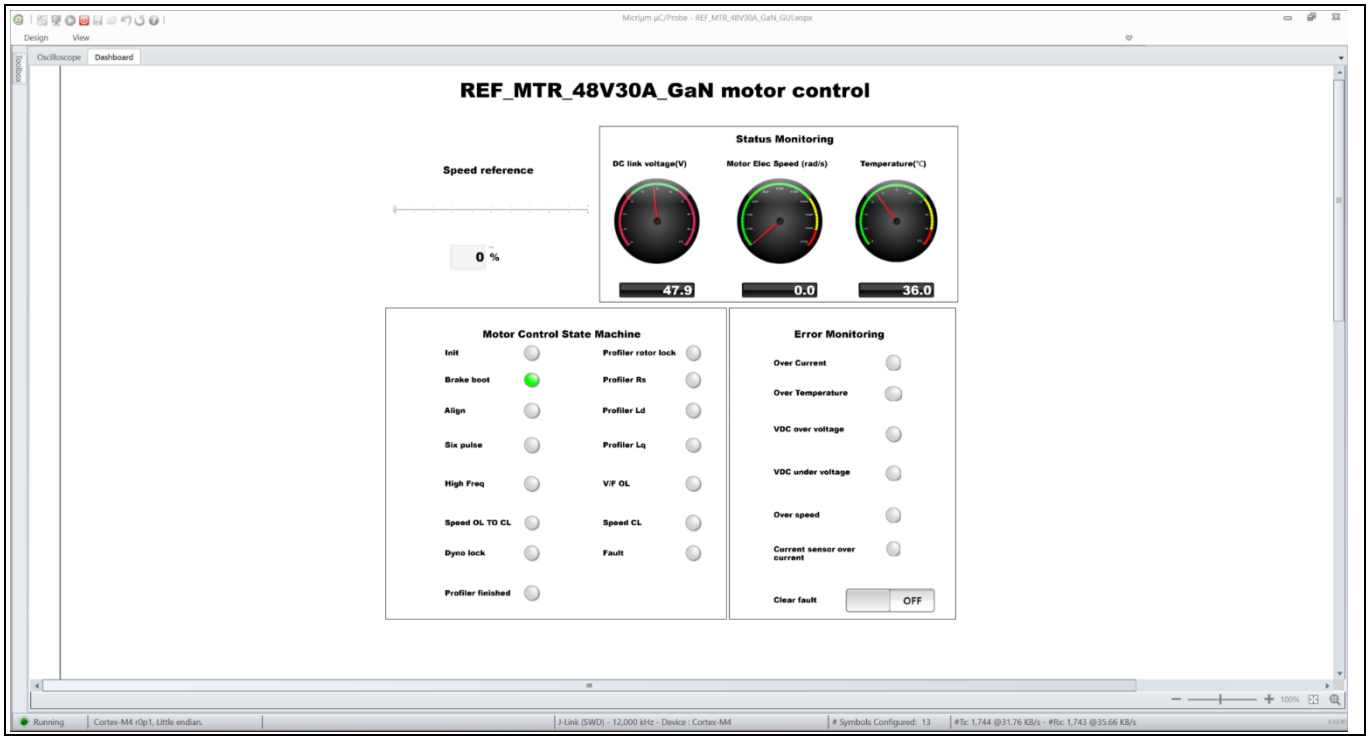

Figure 28 REF_MTR_48V30A_GaN GUI

References

[1] Infineon Technologies AG: *XMC™ Link debugger*; Available online

[2] Infineon Technologies AG: *ModusToolbox™ V3.2*; Available online

[3] Infineon Technologies AG: *µC/Probe™ XMC™ software*; Available online

[4] Infineon Technologies AG: *Ozone – The J-Link Debugger*; Available online

[5] Infineon Technologies AG: *Infineon Developer Center*; Available online

[6] Infineon Technologies AG: *Motor control firmware – Reference manual*; Revision 01; 05-20-2024

[7] T-motor: *P60 KV170 motor*; Available online

[8] Infineon Technologies AG: *CoolGaN™ motor drive reference board for battery powered FOC application*

Revision history

Document revision	Date	Description of changes
V 1.0	2024-09-10	Initial version

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

Edition 2024-09-10

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

UG151804

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

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.