AN1292 Tuning Procedure ReadMe

This document provides a step-by-step procedure on running a motor with the algorithm described in AN1292 *"Sensorless Field Oriented Control (FOC) for a Permanent Magnet Synchronous Motor (PMSM) Using a PLL Estimator and Field Weakening (FW)"* (DS01292).

1.1 SETTING SOFTWARE PARAMETERS

All of the main configurable parameters are defined in the `userparms.h` file. The adaptation of the parameters to the internal numerical format is done using the `tuning_params.xls` Excel® spreadsheet (see Figure 1-1). This file is included with the AN1292 archive file, which is available for download from the Microchip website (www.microchip.com).

After entering the motor and hardware information into the spreadsheet, the calculated parameters need to be entered into the `userparms.h` header file, as indicated by the following steps.

FIGURE 1-1: `tuning_params.xls`

Input parameters		
Peak voltage	24	V
Peak current	5	A
PWM Period (Ts)	0.000050	s
Dead time	0.000002	s
Pole pairs	5	-
Stator resistance (Rs)	2.67	Ohm
Stator inductance (Ls)	0.00192	H
Voltage constant (Kfi)	7.24	Vp/KRPM
Nominal speed (NOMINAL_SPEED_RPM)	2800	RPM
Maximum speed (MAXIMUM_SPEED_RPM)	5500	RPM

Output parameters			
NORM_VOLTAGE_CONST (U0)	0.000406		
NORM_CURRENT_CONST (I0)	0.000153		
NORM_OMEGA_CONST (W0)	0.104720		
		Predivision	
NORM_RS	32886	**2**	16443
NORM_LSDTBASE	472965	**256**	1848
NORM_INVKFIBASE	15912	**2**	7956
NORM_DELTAT	1790		
D_ILIMIT_HS	1502		
D_ILIMIT_LS	6117		

Inverter parameter
Motor parameter
Constant to set in "userparms.h"

Motor Identification
Hurst NTDynamo Brushless
DMB0224C10002 CL B 16208
24VDC 1.00 Amp 86293
Hurst Mfg., Princeton, IN

STEP 1 – Fill in the `tuning_params.xls` Excel spreadsheet with the following parameters:

a) **Peak Voltage**

Peak voltage represents the peak voltage on the DC link capacitors. It also represents the DC voltage itself when a DC power supply is connected to the DC link. If the DC link is supplied from a single-phase rectifier bridge, the AC peak voltage is connected to the rectifier:

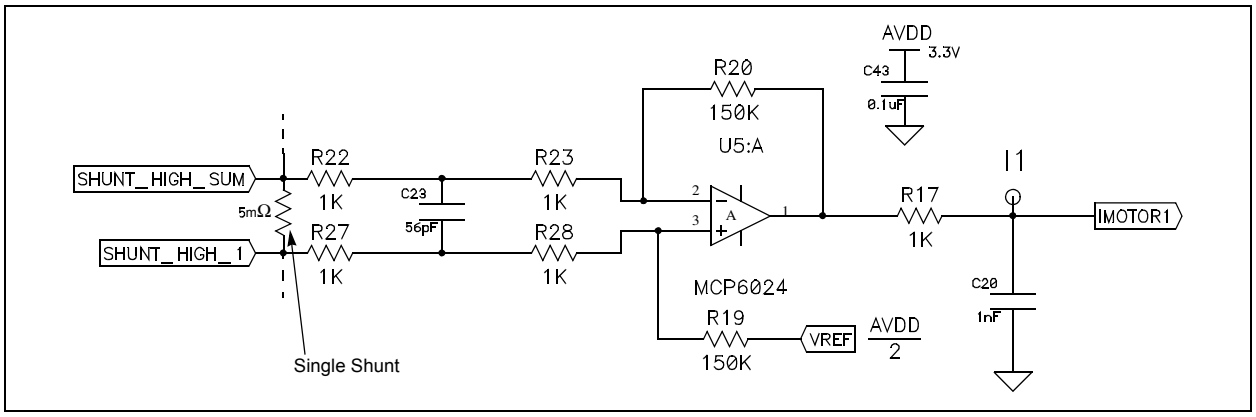
$$V_{ACpeak} = V_{ACrms} \cdot \sqrt{2}$$

A 24 VDC power supply was used in the demonstration software, so the Excel spreadsheet contains the corresponding value. If you are using a high voltage PMSM, peak voltage for 115 VAC is 163V.

b) **Peak Current**

Peak current represents the maximum real value of the current that can be internally represented, which depends on the acquisition block. Considering the maximum input to the ADC of 3.3V, the gain of the acquisition circuitry and the value of the current shunts determine the maximum value of the current that will fit to the dsPIC® DSC internal number representation. Conversely, a current of which the internal number representation is at the upper limit, represents the peak current as it may be entered in the indicated Excel spreadsheet field.

FIGURE 1-2: SIGNAL CONDITIONING CIRCUITRY



For the circuit presented in Figure 1-2 above, the current acquisition circuitry has an amplification gain of:

$$G = \frac{R20}{R22 + R23} = 75$$

The shunt resistor value for the MCLV is 5 mΩ and, with a maximum voltage accepted at the ADC input of 3.3V, results in the maximum current read of:

$$I_{max} = \frac{V_{REF}}{Shunt \cdot Gain} = \frac{\frac{3.3}{2}}{0.005 \cdot 75} = 4.4A$$

Notice that the calculated value of Peak current (I_{max}) differs from the one indicated in the Excel spreadsheet file (Figure 1-1) – the reason being that the second value is experimentally determined as it will be described later in this document (Step 3-d).

c) **PWM Period and Dead Time**

PWM Period is the sampling and control period for this algorithm (AN1292). *Dead time* represents the time needed for the power semiconductor devices to recover from the previous state so that no shoot-through occurs on any inverter leg. The values entered in these fields should coincide with the ones used.

The demonstration software included in the application note implements a value of 2 µs for dead time, and for the PWM period, a value of 50 µs is used, which is a PWM frequency of 20 kHz.

d) **Motor's Electrical Parameters**

For the parameters *Stator resistance (Rs)*, *Stator inductance (Ls)*, and *Voltage constant (Kfi)* enter them from the motor's manufacturer's information or they may be determined experimentally.

Please consult the "Tuning and Experimental Results" section of the application note, AN1292 for details on experimentally calculating *Kfi*.

e) **Nominal and Maximum Speed**

Nominal speed is a parameter provided by the manufacturer and represents the speed achievable with the nominal current and voltage provided on the motor's plate.

Maximum speed is a parameter provided by the manufacturer and depends mostly on the mechanical parameters of the motor. It may be observed that the maximum speed is higher than the nominal speed, and the region in between being covered in constant power mode, where the field weakening technique is implied.

f) **Predivision Factors**

Predivision column corresponds to a scaling constant used for bringing the resulting calculation of normalized values into the numerical representation range, [-32768, 32767]. The *Predivision* scaling should not only bring the constants into the range but also, in case of the inverse voltage constant (Kfi), to divide its initial calculated value so that when it is multiplied afterwards due to field weakening technique, it does not overflow the numerical representation range.

The Predivision factors can be found in the software code in the form of division operation term (left shift).

For example, `NORM_LSDTBASE` *Predivision* scaling is 256 in the spreadsheet, which reflects in the following line of code:

`estim.c`

```
118         EstimParm.qVIndbeta= ((long)MotorEstimParm.qLsDt * (EstimParm.qDIbeta))>>7;
```

As it may be observed, instead of shifting to the left with 15, because of previous predivision with 2^8, it is finally shifted with 7.

The same happens for `NORM_RS`, which is predivided by 2 to keep `NORM_RS` within range, which prevents a numeric overflow. This results in the `estim.c` corresponding code section to counter balance the initial predivision by a shift of 14 instead of 15:

`estim.c`

```
133     EstimParm.qEsa          =       EstimParm.qLastValpha -
134             (((long) MotorEstimParm.qRs  * (long) ParkParm.qIalpha) >>14)
135             -EstimParm.qVIndalpha;
```

In the case of `NORM_INVKFIBASE`, the *Predivision* is 2 and the reverse multiplication is done on the following line of code:

`estim.c`

```
209     EstimParm.qOmegaMr=EstimParm.qOmegaMr<<1;
```

STEP 2 – Export produced parameters to `userparms.h`.

The resulting values in the right side columns grouped as Output parameters are to be entered in the `userparms.h` file corresponding definitions.

Notice that the items on the *Output parameters* are colored differently, indicating precisely which of them are to be copied and pasted directly into the software code.

`userparms.h`

```
139  #define NORM_CURRENT_CONST     0.000153
140  /* normalized ls/dt value */
141  #define NORM_LSDTBASE 1848
142  /* normalized rs value */
143  #define NORM_RS  16433
144  /* the calculation of Rs gives a value exceeding the Q15 range so,
145  the normalized value is further divided by 2 to fit the 32768 limit */
146  /* this is taken care in the estim.c where the value is implied */
147  /* normalized inv kfi at base speed */
148  #define NORM_INVKFIBASE  7956
149  /* the calculation of InvKfi gives a value which not exceed the Q15 limit */
150  /* to assure that an increase of the term with 5 is possible in the lookup table
151  /* for high flux weakening the normalized is initially divided by 2 */
152  /* this is taken care in the estim.c where the value is implied */
153  /* normalized dt value */
154  #define NORM_DELTAT  1790
155
156  // Limitation constants
157  /* di = i(t1)-i(t2) limitation */
158  /* high speed limitation, for dt 50us */
159  /* the value can be taken from attached xls file */
160  #define D_ILIMIT_HS 1502
161  /* low speed limitation, for dt 8*50us */
162  #define D_ILIMIT_LS 6117
```

```
133  /* Nominal speed of the motor in RPM */
134  #define NOMINAL_SPEED_RPM    2800 // Value in RPM
135  /* Maximum speed of the motor in RPM - given by the motor's manufacturer */
136  #define MAXIMUM_SPEED_RPM    5500 // Value in RPM
```

```
130  //***************  Motor Parameters ***************
131  /* motor's number of pole pairs */
132  #define NOPOLESPAIRS 5
```

STEP 3 – First, tune the open loop

a) **Activate Open Loop Functioning**

The open loop tuning can be operated separately, by enabling a special `#define` in the FOC software code; otherwise, the transition to close loop control is automatically done. Make sure you disable closed loop transition for the initial tuning of open loop.

`userparms.h`

```
57  /* open loop continuous functioning */
58  /* closed loop transition disabled  */
59  #define OPEN_LOOP_FUNCTIONING
```

b) **Set Up Open Loop Parameters**

- *Current Scaling*

The prescaling constant needs to be set to adapt the ADC output to correspond to the real value in terms of sign (direction), and if necessary, to prescale it to an intermediary value, adequate for further processing.

`userparms.h`

```
122  #define     KCURRA              Q15(-0.5)  /* scaling factor for current phase A */
123  #define     KCURRB              Q15(-0.5)  /* scaling factor for current phase B */
```

The scaling factor for currents are negative because the acquisition for shunts is getting the reverse sense of the currents, and therefore, the value of Q15(-0.5) represents a (-1) multiplication of the Q15 value returned by the ADC.

- *Start-up Torque Current*

Choose nominal current for the given motor as a starting point, as indicated below (in this case, a value of 1.41 amperes was used):

`userparms.h`

```
197  /* open loop q current setup - */
198  #define Q_CURRENT_REF_OPENLOOP NORM_CURRENT(1.41)
199
```

If the start-up current is too low, the load will not move. If it is too high, the motor can overheat if it runs in open loop for a long period of time.

- *Lock Time*

In general, a lock time of a value of a few hundred milliseconds is selected.

`userparms.h`

```
188  /* open loop startup constants */
189  /* the following values depends on the PWM frequency, */
190  /* lock time is the time needed for motor's poles alignment
191   before the open loop speed ramp up */
192  #define LOCK_TIME 4000 // This number is: 20,000 is 1 second.
```

The lock time value depends on the PWM frequency. For example, at 20 kHz, the value 4000 would represent 0.2 seconds.

- *Ramp Increase Rate*

The open loop acceleration should be set as small as possible at the beginning. The smaller this value, the more capable the motor is to start with a higher resistant torque or moment of inertia.

`userparms.h`

```
195  /* open loop speed ramp up speed of increase */
196  #define OPENLOOP_RAMPSPEED_INCREASERATE 10
```

- *End Speed*

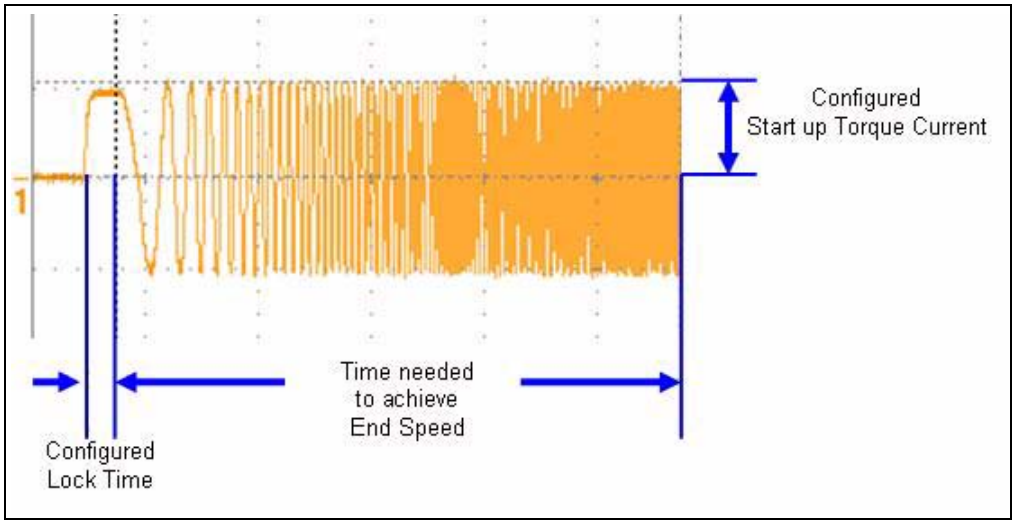
  End speed value setup is a trade-off between the efficiency of the control and the estimator's minimum speed limit to accurately estimate speed and position. Normally, the user would want to set the open loop end speed value as low as possible so that the transitions to closed loop functioning occur as soon as possible from the startup. Keeping in mind the compromise stated above, consider an end speed of one third of nominal speed of the motor under tuning for the beginning.

`userparms.h`

```
193   /* open loop speed ramp up end value */
194   #define END_SPEED_RPM 1000 // Value in RPM
```

Figure 1-3 illustrates the parameters that were previously chosen. The image represents the capture of an oscilloscope current probe attached to one of the motor's phases.

FIGURE 1-3:



- *PI Current Controllers*

Some general guidelines for effective tuning of this application's PI controllers are:

- Both controllers, on the D and Q axis, will have the same values for corresponding Proportional (`D_CURRCNTR_PTERM`, `Q_CURRCNTR_PTERM`), Integral (`D_CURRCNTR_ITERM`, `Q_CURRCNTR_ITERM`), Anti-windup Compensation (`D_CURRCNTR_CTERM`, `Q_CURRCNTR_ITERM`), and Minimum-Maximum (`D_CURRCNTR_OUTMAX`, `Q_CURRCNTR_OUTMAX`, `D_CURRCNTR_OUTMIN`, `Q_CURRCNTR_OUTMIN`) terms.
- In general, whenever current oscillation happens, lower the proportional gain term making sure integral gain is from 5 to 10 times smaller than proportional gain.

Use the values shown below as a starting point.

`userparms.h`

```
204  /* PI controllers tuning values - */
205  //******** D Control Loop Coefficients *******
206  #define     D_CURRCNTR_PTERM            Q15(0.05)
207  #define     D_CURRCNTR_ITERM            Q15(0.005)
208  #define     D_CURRCNTR_CTERM            Q15(0.999)
209  #define     D_CURRCNTR_OUTMAX           0x7FFF

211  //******** Q Control Loop Coefficients *******
212  #define     Q_CURRCNTR_PTERM            Q15(0.05)
213  #define     Q_CURRCNTR_ITERM            Q15(0.005)
214  #define     Q_CURRCNTR_CTERM            Q15(0.999)
215  #define     Q_CURRCNTR_OUTMAX           0x7FFF
```

c) **Open Loop Parameters Optimization**

The settings above would enable open loop operation. Once it has been verified that everything is working fine with the setup previously explained, try to fine tune the parameters for smoother and more efficient operation by:

- decreasing startup torque current
- increasing speedup ramp rate
- reducing the lock time
- decreasing end speed

STEP 4 – Tuning the Closed Loop Operation

a) **Enable Close Loop Transition**

Step forward to close loop tuning once the open loop is running fine, by removing the define of the `OPEN_LOOP_FUNCTIONING` macro definition.

`userparms.h`

```
57  /* open loop continuous functioning */
58  /* closed loop transition disabled  */
59  #undef OPEN_LOOP_FUNCTIONING
60
```

b) **Set Up Close Loop Parameters**

- *Initial Angle Offset Tuning*

The transition between open loop to close loop implies an initial estimation error, for which pre-selection of an initial offset angle is required:

`userparms.h`

```
183  #define INITOFFSET_TRANS_OPEN_CLSD 0x2000
```

The initial value of 0x2000 corresponds to an offset of 45 degrees. The formula for getting the value which corresponds to a particular offset angle is:

$$value = 0xFFFF \cdot \frac{angle°}{360°}$$

Depending on the resistant torque of the load, the moment of inertia, or depending on the motor's electrical constants, modify the angle to eliminate the eventual open loop/close loop transition glitches.

- *Estimator Filter Coefficients*

The default constants set up for the filters' coefficients should give good results for most motors. Nevertheless, decreasing the coefficients would decrease the phase delay, which could be particularly helpful at high speeds, where armature current variation is faster. A compromise between the filtering role and its counter back effect, the introduction of phase shift, should be achieved.

`userparms.h`

```
167  // Filters constants definitions
168  /* BEMF filter for d-q components @ low speeds */
169  #define KFILTER_ESDQ 1200
170  /* BEMF filter for d-q components @ high speed - Flux Weakening case */
171  #define KFILTER_ESDQ_FW 164
172  /* estimated speed filter constatn */
173  #define KFILTER_VELESTIM 2*374
```

- *PI Speed Controller*

For the speed controller tuning, P and I gain can be adjusted using multiple methods. For more information, search for "PID Controller" on the Wikipedia website and go to the "Loop Tuning" section.

`userparms.h`

```
217  //*** Velocity Control Loop Coefficients *****
218  #define     SPEEDCNTR_PTERM        Q15(0.5)
219  #define     SPEEDCNTR_ITERM        Q15(0.005)
220  #define     SPEEDCNTR_CTERM        Q15(0.005)
221  #define     SPEEDCNTR_OUTMAX       0x5000
```

For cases where no speed controller is needed, the torque mode can be activated by defining `TORQUE_MODE`.

`userparms.h`

```
62  /* definition for torque mode - for a separate tuning of the current PI
63   controllers, tuning mode will disable the speed PI controller */
64  #define TORQUE_MODE
```

STEP 5 – Optionally, Tune the High-Speed Field Weakening Parameters

CAUTION

Usually, the motor manufacturer indicates the maximum speed achievable by the motor without it being damaged (which could be higher than the brake point speed at rated current). If not, it is possible to run it at higher speeds but only for small periods (intermittent) assuming the risks of demagnetization or mechanical damage of the motor or of the devices attached to it.
In Field Weakening mode, if the controller becomes lost due to a miscalculation of the angle at high speed above the nominal value, the possibility of damaging the inverter is imminent. The reason is that the Back Electromotive Force (BEMF) will have a greater value than the one that would be obtained for the nominal speed, thereby exceeding the DC bus voltage value, which the inverter's power semiconductors and DC link capacitors would have to support. Since the tuning proposed implies iterative coefficient corrections until the optimum functioning is achieved, the protection of the inverter with corresponding circuitry should be modified to handle higher voltages in case of stalling at high speeds.

a) **Set Up Initial Parameters**

- *Nominal and Maximum Speed*

Start with a value for nominal speed RPM (i.e., a couple of hundred RPM less than the motor rated speed). In this example, the motor is rated for 3000 RPM; therefore, we set `NOMINAL_SPEED_RPM` to 2800. Consult the motor specification for the maximum field weakening speed, and enter this value into `MAXIMUM_SPEED_RPM`.

`userparms.h`

```
133  /* Nominal speed of the motor in RPM */
134  #define NOMINAL_SPEED_RPM    2800 // Value in RPM
135  /* Maximum speed of the motor in RPM - given by the motor's manufacturer */
136  #define MAXIMUM_SPEED_RPM    5500 // Value in RPM
```

Be aware of the fact that for these values above (over) *Nominal speed*, the field weakening strategy is enabled, and therefore, lowering the nominal speed used for smoothing this transition implies additional energy is spent on airgap flux decrease, which overall, leads to lower efficiency.

- *D-axis Current Reference*

D-axis reference current lookup table (ID) has values between 0 and the nominal stator current, distributed evenly on 18 entries of the lookup. The nominal stator current can be taken from the motor specification. If it is unknown, this value can be approximated by dividing the rated power over the rated voltage.

userparms.h

```
254     /* the following values indicate the d-current variation with speed */
255     /* please consult app note for details on tuning */
256     #define IDREF_SPEED0    NORM_CURRENT(0)       // up to 2800 RPM
257     #define IDREF_SPEED1    NORM_CURRENT(-0.24)   // ~2950 RPM
258     #define IDREF_SPEED2    NORM_CURRENT(-0.56)   // ~3110 RPM
259     #define IDREF_SPEED3    NORM_CURRENT(-0.828)  // ~3270 RPM
260     #define IDREF_SPEED4    NORM_CURRENT(-1.121)  // ~3430 RPM
261     #define IDREF_SPEED5    NORM_CURRENT(-1.385)  // ~3600 RPM
262     #define IDREF_SPEED6    NORM_CURRENT(-1.6)    // ~3750 RPM
263     #define IDREF_SPEED7    NORM_CURRENT(-1.8)    // ~3910 RPM
264     #define IDREF_SPEED8    NORM_CURRENT(-1.923)  // ~4070 RPM
265     #define IDREF_SPEED9    NORM_CURRENT(-2.055)  // ~4230 RPM
266     #define IDREF_SPEED10   NORM_CURRENT(-2.15)   // ~4380 RPM
267     #define IDREF_SPEED11   NORM_CURRENT(-2.2)    // ~4550 RPM
268     #define IDREF_SPEED12   NORM_CURRENT(-2.25)   // ~4700 RPM
269     #define IDREF_SPEED13   NORM_CURRENT(-2.3)    // ~4860 RPM
270     #define IDREF_SPEED14   NORM_CURRENT(-2.35)   // ~5020 RPM
271     #define IDREF_SPEED15   NORM_CURRENT(-2.4)    // ~5180 RPM
272     #define IDREF_SPEED16   NORM_CURRENT(-2.423)  // ~5340 RPM
273     #define IDREF_SPEED17   NORM_CURRENT(-2.442)  // ~5500 RPM
274
```

- *Voltage Constant Inverse*

The lookup table entry corresponding to the maximum speed achievable in field weakening is proportional to the percentage of increase of mechanical speed from nominal to the maximum values. In the lookup table entries, the values are evenly distributed and if the inverse voltage constant for maximum speed exceeds the numerical representation range (32,767), adjust the corresponding *Predivision* scaling factor. Note that the following numbers are predivided by 2 (see Figure 1-1).

userparms.h

```
276     /* the following values indicate the invKfi variation with speed */
277     /* please consult app note for details on tuning */
278     #define INVKFI_SPEED0   7956     // up to 2800 RPM
279     #define INVKFI_SPEED1   9300     // ~2950 RPM
280     #define INVKFI_SPEED2   10820    // ~3110 RPM
281     #define INVKFI_SPEED3   11900    // ~3270 RPM
282     #define INVKFI_SPEED4   13110    // ~3430 RPM
283     #define INVKFI_SPEED5   14000    // ~3600 RPM
284     #define INVKFI_SPEED6   14800    // ~3750 RPM
285     #define INVKFI_SPEED7   15350    // ~3910 RPM
286     #define INVKFI_SPEED8   15720    // ~4070 RPM
287     #define INVKFI_SPEED9   16120    // ~4230 RPM
288     #define INVKFI_SPEED10  16520    // ~4380 RPM
289     #define INVKFI_SPEED11  16740    // ~4550 RPM
290     #define INVKFI_SPEED12  16920    // ~4700 RPM
291     #define INVKFI_SPEED13  17140    // ~4860 RPM
292     #define INVKFI_SPEED14  17430    // ~5020 RPM
293     #define INVKFI_SPEED15  17650    // ~5180 RPM
294     #define INVKFI_SPEED16  17840    // ~5340 RPM
295     #define INVKFI_SPEED17  17950    // ~5500 RPM
```

- *Inductance Variation*

For the inductance variation (LsOver2Ls0) lookup table, the first value in the table should always be one-half since the base speed inductance is divided by its own doubled value. These values should work for most motors.

`userparms.h`

```
297   /* the following values indicate the Ls variation with speed */
298   /* please consult app note for details on tuning */
299   #define     LS_OVER2LS0_SPEED0            Q15(0.5)    // up to 2800 RPM
300   #define     LS_OVER2LS0_SPEED1            Q15(0.45)   // ~2950 RPM
301   #define     LS_OVER2LS0_SPEED2            Q15(0.4)    // ~3110 RPM
302   #define     LS_OVER2LS0_SPEED3            Q15(0.35)   // ~3270 RPM
303   #define     LS_OVER2LS0_SPEED4            Q15(0.3)    // ~3430 RPM
304   #define     LS_OVER2LS0_SPEED5            Q15(0.25)   // ~3600 RPM
305   #define     LS_OVER2LS0_SPEED6            Q15(0.25)   // ~3750 RPM
306   #define     LS_OVER2LS0_SPEED7            Q15(0.25)   // ~3910 RPM
307   #define     LS_OVER2LS0_SPEED8            Q15(0.25)   // ~4070 RPM
308   #define     LS_OVER2LS0_SPEED9            Q15(0.25)   // ~4230 RPM
309   #define     LS_OVER2LS0_SPEED10           Q15(0.25)   // ~4380 RPM
310   #define     LS_OVER2LS0_SPEED11           Q15(0.25)   // ~4550 RPM
311   #define     LS_OVER2LS0_SPEED12           Q15(0.25)   // ~4700 RPM
312   #define     LS_OVER2LS0_SPEED13           Q15(0.25)   // ~4860 RPM
313   #define     LS_OVER2LS0_SPEED14           Q15(0.25)   // ~5020 RPM
314   #define     LS_OVER2LS0_SPEED15           Q15(0.25)   // ~5180 RPM
315   #define     LS_OVER2LS0_SPEED16           Q15(0.25)   // ~5340 RPM
316   #define     LS_OVER2LS0_SPEED17           Q15(0.25)   // ~5500 RPM
```

b) **Runtime Parameters Adjustment**

If the results of running the software in these conditions will stall the motor at a speed higher than nominal, it is due to the fact that the lookup tables were filled with estimated values, which at some point do not match the real non-linearities.

Once the motor stalls, immediately halt the program execution, capturing the value of the index (`FdWeakParm.qIndex`) in the debugger watch window. The index indicates the point where the values of IDREF (see the IDREF table in Step 5a), in ascending order, were not effective and should be updated. In order to further improve the performance, the value indicated by the current index in the lookup table should be replaced by the value indicated by the next index (`FdWeakParm.qIndex` + 1) and the motor's behavior should be checked again. The achievable speed should increase and repeating this process for several times the maximum speed for the nominal current reference imposed on the d-axis will be reached.

If the maximum speed obtained for the nominal current is lower than the targeted one, the absolute value of the d-axis current reference should be increased above the nominal value. As an example, if 5500 RPM cannot be reached, change `IDREF_SPEED17` current from -1.53 to -1.60 and try again. The d-current reference increase should be started from the value denoted by the index where the motor stalled. The index value should correspond to the actual speed of the motor, measured at the shaft using a tachometer, keeping in mind that the lookup index is calculated using the reference speed, *not* the actual speed.

Once the d-current increase stops increasing the speed (increasing the current too much will generally stall the motor), the index corresponding to the stall will indicate where the value for inductance should be adjusted (increasing or decreasing its value). The inductance variation lookup table is the last to be updated.

It is always better to have a software speed reference instead of the standard potentiometer/DMCI slider control since it is more stable and it can offer better granularity.

For testing purposes, a slow software ramp is implemented as a speed reference, being activated using the following definition:

userparms.h

```
107   #define TUNING_DELAY_RAMPUP   0x3F      /* the smaller the value, the smaller the
```

The smaller the value of the delay of ramp, the slower the ramp will increase. This ramp always starts from open loop zero speed, gets to END_SPEED_RPM value in open loop, switches to closed loop and continues acceleration up to MAXIMUM_SPEED_RPM in close loop. Software speed reference is particularly important in the field weakening region for this algorithm, since it is used for referencing the index in the lookup tables.