6.4 Encoder Registers

Attention:
The encoder interface is not available in Step&Direction mode, as the encoder pins serve a different function in that mode.

ENCODER REGISTER SET (0X38…0X3C)					
R/W	**Addr**	**n**	***Register***	**Description / *bit names***	**Range [Unit]**
RW	0x38	11	***ENCMODE***	Encoder configuration and use of N channel *See separate table!*	
RW	0x39	32	*X_ENC*	Actual encoder position (signed)	-2^31… +(2^31)-1
W	0x3A	32	*ENC_CONST*	Accumulation constant (signed) 16 bit integer part, 16 bit fractional part *X_ENC* accumulates +/- *ENC_CONST* / (2^16**X_ENC*) (binary) or +/-*ENC_CONST* / (10^4**X_ENC*) (decimal) *ENCMODE* bit *enc_sel_decimal* switches between decimal and binary setting. Use the sign, to match rotation direction!	binary: ± [µsteps/2^16] ±(0 … 32767.999847) decimal: ±(0.0 … 32767.9999) *reset default = 1.0 (=65536)*
R+ WC	0x3B	2	*ENC_STATUS*	Encoder status information bit 0: *n_event* bit 1*: deviation_warn* 1: Event detected. To clear the status bit, write with a 1 bit at the corresponding position. Deviation_warn cannot be cleared while a warning still persists. Set *ENC_DEVIATION* zero to disable. Both bits are ORed to the *interrupt output* signal.	
R	0x3C	32	*ENC_LATCH*	Encoder position *X_ENC* latched on N event	
W	0x3D	20	*ENC_ DEVIATION*	Maximum number of steps deviation between encoder counter and XACTUAL for deviation warning Result in flag ENC_STATUS.*deviation_warn* 0=Function is off.	

6.4.1 *ENCMODE* – Encoder Register

0x38: *ENCMODE* – Encoder register			
Bit	**Name**	**Comment**	
10	*enc_sel_decimal*	0	Encoder prescaler divisor binary mode: Counts *ENC_CONST(fractional part)* /65536
		1	Encoder prescaler divisor decimal mode: Counts in *ENC_CONST(fractional part)* /10000
9	*latch_x_act*	1: Also latch *XACTUAL* position together with *X_ENC*. Allows latching the ramp generator position upon an N channel event as selected by *pos_edge* and *neg_edge*.	
8	*clr_enc_x*	0	Upon N event, *X_ENC* becomes latched to *ENC_LATCH* only
		1	Latch and additionally clear encoder counter *X_ENC* at N-event
7	*neg_edge*	***n p***	**N channel event sensitivity**
6	*pos_edge*	0 0	N channel event is active during an active N event level
		0 1	N channel is valid upon active going N event
		1 0	N channel is valid upon inactive going N event
		1 1	N channel is valid upon active going and inactive going N event
5	*clr_once*	1: Latch or latch and clear *X_ENC* on the next N event following the write access	
4	*clr_cont*	1: Always latch or latch and clear *X_ENC* upon an N event (once per revolution, it is recommended to combine this setting with edge sensitive N event)	
3	*ignore_AB*	0	An N event occurs only when polarities given by *pol_N*, *pol_A* and *pol_B* match.
		1	Ignore A and B polarity for N channel event
2	*pol_N*	Defines active polarity of N (0=low active, 1=high active)	
1	*pol_B*	Required B polarity for an N channel event (0=neg., 1=pos.)	
0	*pol_A*	Required A polarity for an N channel event (0=neg., 1=pos.)	

20 ABN Incremental Encoder Interface

The TMC5160 is equipped with an incremental encoder interface for ABN encoders. The encoder inputs are multiplexed with other signals in order to keep the pin count of the device low. The basic selection of the peripheral configuration is set by the register *GCONF*. The use of the N channel is optional, as some applications might use a reference switch or stall detection rather than an encoder N channel for position referencing. The encoders give positions via digital incremental quadrature signals (usually named A and B) and a clear signal (usually named N for null or Z for zero).

N Signal

The N signal can be used to clear the position counter or to take a snapshot. To continuously monitor the N channel and trigger clearing of the encoder position or latching of the position, where the N channel event has been detected, set the flag *clr_cont*. Alternatively it is possible to react to the next encoder N channel event only, and automatically disable the clearing or latching of the encoder position after the first N signal event (flag *clr_once*). This might be desired because the encoder gives this signal once for each revolution.

Some encoders require a validation of the N signal by a certain configuration of A and B polarity. This can be controlled by *pol_A* and *pol_B* flags in the *ENCMODE* register. For example, when both *pol_A* and *pol_B* are set, an active N-event is only accepted during a high polarity of both, A and B channel.

For clearing the encoder position *ENC_POS* with the next active N event set *clr_enc_x* = 1 and *clr_once* = 1 or *clr_cont* = 1.

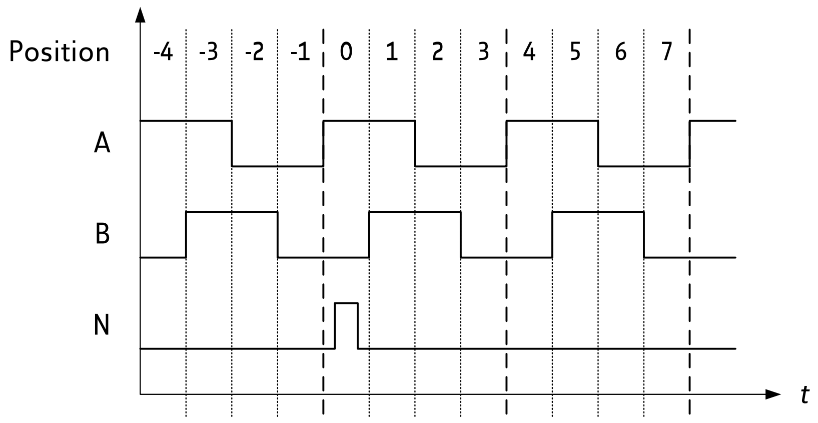


Figure 20.1 Outline of ABN signals of an incremental encoder

The Encoder Constant *ENC_CONST*

The encoder constant *ENC_CONST* is added to or subtracted from the encoder counter on each polarity change of the quadrature signals AB of the incremental encoder. The encoder constant *ENC_CONST* represents a signed fixed point number (16.16) to facilitate the generic adaption between motors and encoders. In decimal mode, the lower 16 bits represent a number between 0 and 9999. For stepper motors equipped with incremental encoders the fixed number representation allows very comfortable parameterization. Additionally, mechanical gearing can easily be taken into account. Negating the sign of *ENC_CONST* allows inversion of the counting direction to match motor and encoder direction.

Examples:

- Encoder factor of 1.0: *ENC_CONST* = 0x0001.0x0000 = FACTOR.FRACTION
- Encoder factor of -1.0: *ENC_CONST* = 0xFFFF.0x0000. This is the two's complement of 0x00010000. It equals (2^16-(FACTOR+1)).(2^16-FRACTION)
- Decimal mode encoder factor 25.6: 00025.6000 = 0x0019.0x1770 = FACTOR.DECIMALS

- Decimal mode encoder factor -25.6: 0xFFE6.4000 = 0xFFE6.0x0FA0. This equals (2^16-(FACTOR+1)).(10000-DECIMALS)

The Encoder Counter *X_ENC*

The encoder counter *X_ENC* holds the current encoder position ready for read out. Different modes concerning handling of the signals A, B, and N take into account active low and active high signals found with different types of encoders. For more details please refer to the register mapping in section 6.4.

The Register *ENC_STATUS*

The register *ENC_STATUS* holds the status concerning the event of an encoder clear upon an N channel signals. The register *ENC_LATCH* stores the actual encoder position on an N signal event.

20.1 Encoder Timing

The encoder inputs use analog and digital filtering to ensure reliable operation even with increased cable length. The maximum continuous counting rate is limited by input filtering to 2/3 of f_{CLK}.

Encoder interface timing	**AC-Characteristics** clock period is t_{CLK}					
Parameter	**Symbol**	**Conditions**	**Min**	**Typ**	**Max**	**Unit**
Encoder counting frequency	f_{CNT}			<2/3 f_{CLK}	f_{CLK}	
A/B/N input low time	t_{ABNL}		3 t_{CLK}+20			ns
A/B/N input high time	t_{ABNH}		3 t_{CLK}+20			ns
A/B/N spike filtering time	$t_{FILTABN}$	Rising and falling edge		3 t_{CLK}		

20.2 Setting the Encoder to Match Motor Resolution

Encoder example settings for motor parameters: USC=256 µsteps, 200 fullstep motor
Factor = FSC*USC / encoder resolution

Encoder example settings for a 200 fullstep motor with 256 microsteps		
Encoder resolution	**Required encoder factor**	**Comment**
200	256	
360	142.2222 = 9320675.5555 / 2^16 = 1422222.2222 / 10000	No exact match possible!
500	102.4 = 6710886.4 / 2^16 = 1024000 / 10000	Exact match with decimal setting
1000	51.2	Exact match with decimal setting
1024	50	
4000	12.8	Exact match with decimal setting
4096	12.5	
16384	3.125	

Example:

The encoder constant register shall be programmed to 51.2 in decimal mode. Therefore, set $ENC\_CONST = 51 * 2^{16} + 0.2 * 10000$

20.3 Closing the Loop

Depending on the application, an encoder can be used for different purposes. Medical applications often require an additional and independent monitoring to detect hard or soft failure. Upon failure, the machine can be stopped and restarted manually. Use *ENC_DEVIATION* setting and interrupt to safely detect a step loss failure / mismatch between motor and encoder.

Less critical applications may use the encoder to detect failure, stop the motors upon step loss and restart automatically. A different use of the encoder allows increased positioning precision by positioning directly to encoder positions. The application can modify target positions based on the deviation, or even regularly update the actual position with the encoder position.

To realize a directly encoder based commutation, TRINAMIC offers the new S-ramp closed loop motion controller TMC4361.

21 DC Motor or Solenoid

The TMC5160 can drive one or two DC motors using one coil output per DC motor. Either a torque limited operation, or a voltage based velocity control with optional torque limit is possible.

Configuration and Control

Set the flag *direct_mode* in the *GCONF* register. In direct mode, the coil current polarity and coil current, respectively the PWM duty cycle become controlled by register *XTARGET* (0x2D). Bits 8..0 control motor A and Bits 24..16 control motor B PWM. Additionally to this setting, the current limit is scaled by *IHOLD*. The STEP/DIR inputs and the motion controller are not used in this mode.

PWM Duty Cycle Velocity Control

In order to operate the motor at different velocities, use the StealthChop voltage PWM mode in the following configuration:
en_pwm_mode = 1, *pwm_autoscale* = 0, *PWM_OFS* = 255, *PWM_GRAD* = 4, *IHOLD* = 31
Set *TOFF* > 0 to enable the driver.
In this mode the driver behaves like a 4-quadrant power supply. The direct mode setting of PWM A and PWM B using *XTARGET* controls motor voltage, and thus the motor velocity. Setting the corresponding PWM bits between -255 and +255 (signed, two's complement numbers) will vary motor voltage from -100% to 100%. With *pwm_autoscale* = 0, current sensing is not used and the sense resistors should be eliminated or 150mΩ or less to avoid excessive voltage drop when the motor becomes heavily loaded up to 2.5A. Especially for higher current motors, make sure to slowly accelerate and decelerate the motor in order to avoid overcurrent or triggering driver overcurrent detection.

To activate optional motor freewheeling, set *IHOLD* = 0 and *FREEWHEEL* = %01.

Additional Torque Limit

In order to additionally take advantage of the motor current limitation (and thus torque controlled operation) in StealthChop mode, use automatic current scaling (*pwm_autoscale* = 1, *PWM_OFS* = 30). The actual current limit is given by *IHOLD* and scaled by the respective motor PWM amplitude, e.g. PWM = 128 yields in 50% motor velocity and 50% of the current limit set by *IHOLD*. In case two DC motors are driven in voltage PWM mode, note that the automatic current regulation will work only for the motor which has the higher absolute PWM setting. The PWM of the second motor also will be scaled down in case the motor with higher PWM setting reaches its current limitation.

Purely Torque Limited Operation

For a purely torque limited operation of one or two motors, spread cycle chopper individually regulates motor current for both full bridge motor outputs. When using SpreadCycle, the upper motor velocity is limited by the supply voltage only (or as determined by the load on the motor).

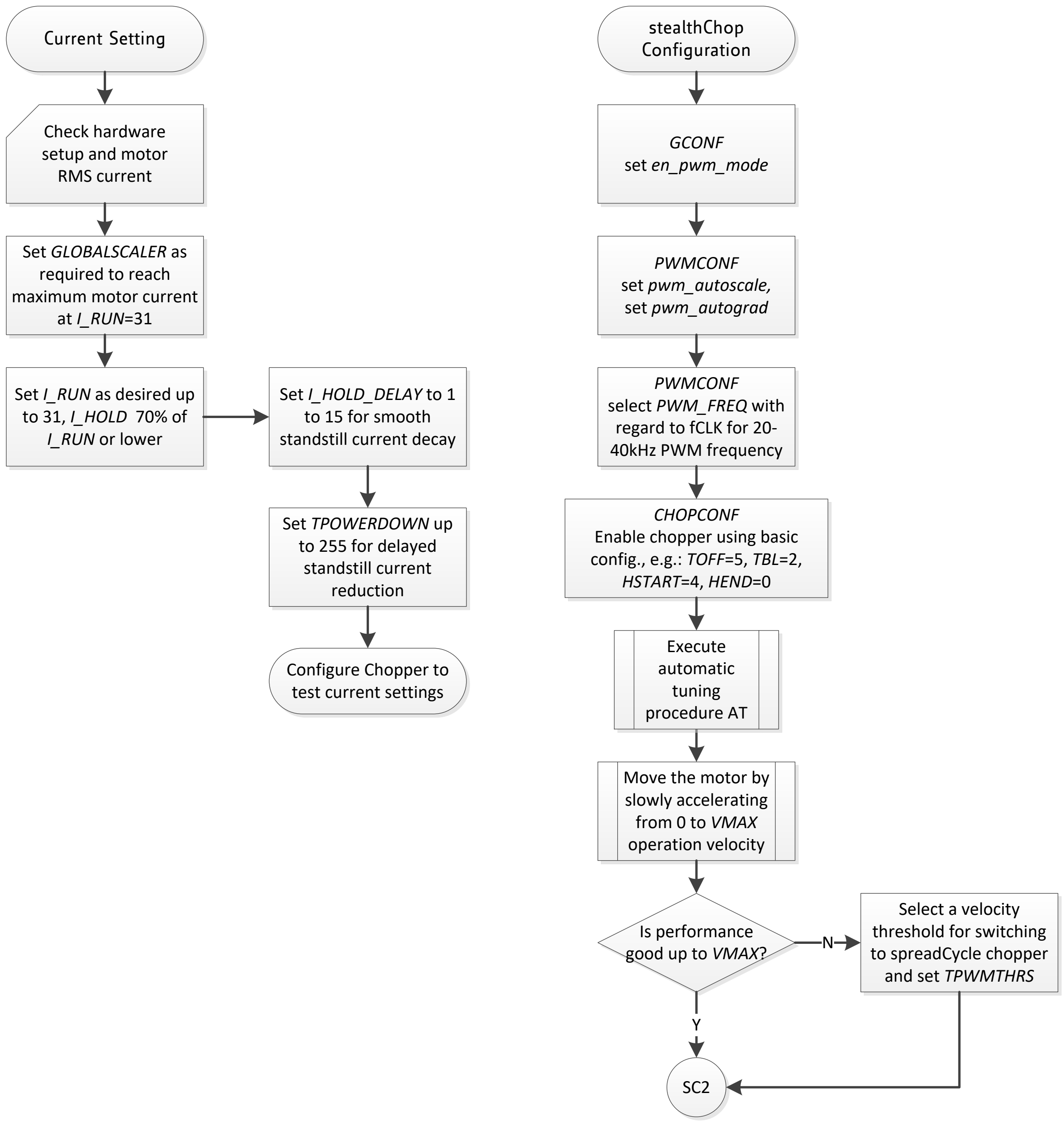
21.1 Solenoid Operation

The same way, one or two solenoids (i.e. magnetic coil actuators) can be operated using SpreadCycle chopper. For solenoids, it is often desired to have an increased current for a short time after switching on, and reduce the current once the magnetic element has switched. This is automatically possible by taking advantage of the automatic current scaling (*IRUN*, *IHOLD*, *IHOLDDELAY* and *TPOWERDOWN*). The current scaling in *direct_mode* is still active, but will not be triggered if no step impulse is supplied. Therefore, a step impulse must be given to the STEP input whenever one of the coils shall be switched on. This will increase the current for both coils at the same time.

22 Quick Configuration Guide

This guide is meant as a practical tool to come to a first configuration and do a minimum set of measurements and decisions for tuning the driver. It does not cover all advanced functionalities, but concentrates on the basic function set to make a motor run smoothly. Once the motor runs, you may decide to explore additional features, e.g. freewheeling and further functionality in more detail. A current probe on one motor coil is a good aid to find the best settings, but it is not a must.

Current Setting and First Steps with StealthChop



Current Setting
Check hardware setup and motor RMS current
Set *GLOBALSCALER* as required to reach maximum motor current at *I_RUN*=31
Set *I_RUN* as desired up to 31, *I_HOLD* 70% of *I_RUN* or lower
Set *I_HOLD_DELAY* to 1 to 15 for smooth standstill current decay
Set *TPOWERDOWN* up to 255 for delayed standstill current reduction
Configure Chopper to test current settings
stealthChop Configuration
GCONF set *en_pwm_mode*
PWMCONF set *pwm_autoscale,* set *pwm_autograd*
PWMCONF select *PWM_FREQ* with regard to fCLK for 20-40kHz PWM frequency
CHOPCONF Enable chopper using basic config., e.g.: *TOFF*=5, *TBL*=2, *HSTART*=4, *HEND*=0
Execute automatic tuning procedure AT
Move the motor by slowly accelerating from 0 to *VMAX* operation velocity
Is performance good up to *VMAX*?
N
Select a velocity threshold for switching to spreadCycle chopper and set *TPWMTHRS*
Y
SC2

Figure 22.1 Current setting and first steps with StealthChop

Tuning StealthChop and SpreadCycle

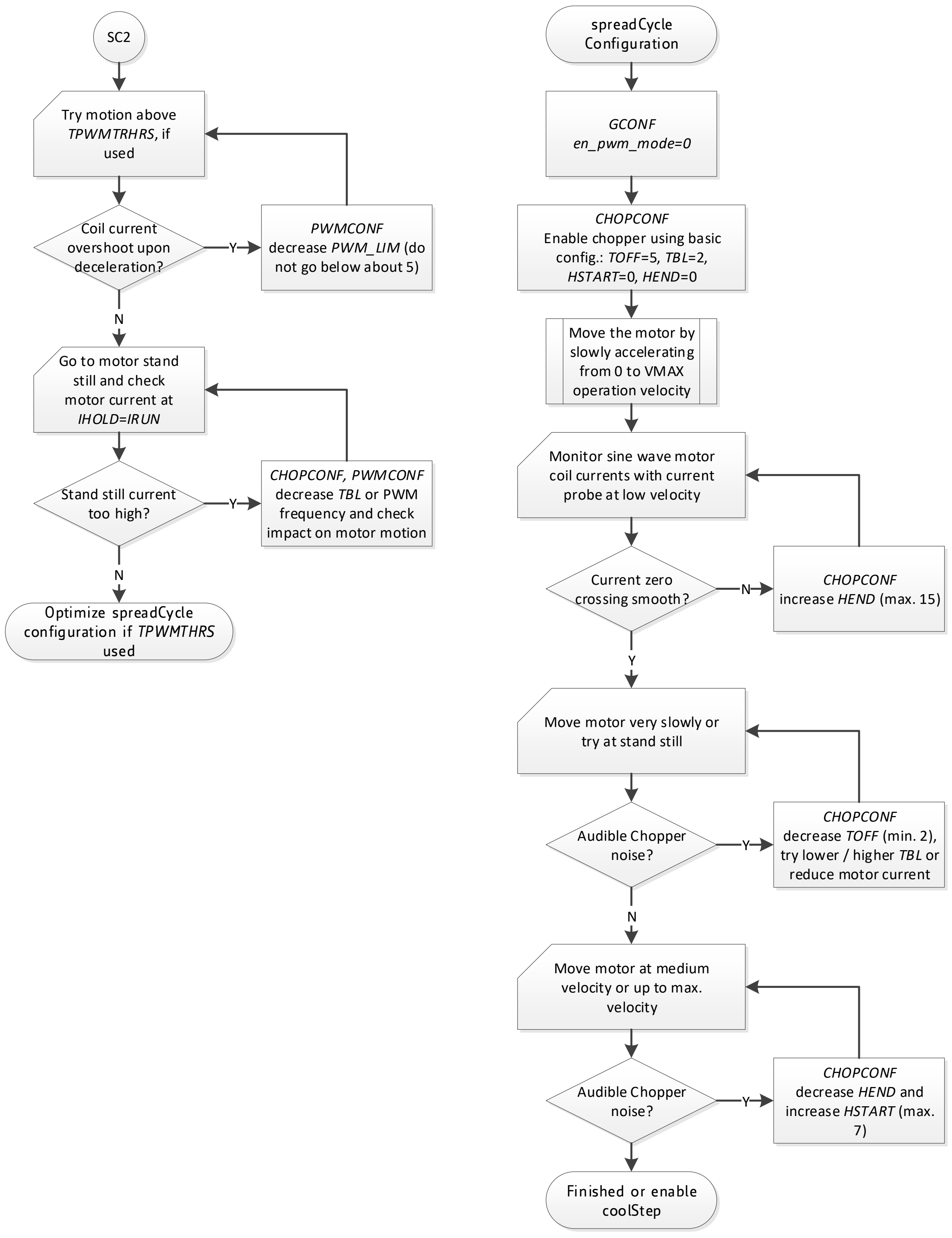

SC2
Try motion above *TPWMTRHRS*, if used
Coil current overshoot upon deceleration?
Y
PWMCONF decrease *PWM_LIM* (do not go below about 5)
N
Go to motor stand still and check motor current at *IHOLD=IRUN*
Stand still current too high?
Y
CHOPCONF, PWMCONF decrease *TBL* or PWM frequency and check impact on motor motion
N
Optimize spreadCycle configuration if *TPWMTHRS* used
spreadCycle Configuration
GCONF *en_pwm_mode=0*
CHOPCONF Enable chopper using basic config.: *TOFF*=5, *TBL*=2, *HSTART*=0, *HEND*=0
Move the motor by slowly accelerating from 0 to VMAX operation velocity
Monitor sine wave motor coil currents with current probe at low velocity
Current zero crossing smooth?
N
CHOPCONF increase *HEND* (max. 15)
Y
Move motor very slowly or try at stand still
Audible Chopper noise?
Y
CHOPCONF decrease *TOFF* (min. 2), try lower / higher *TBL* or reduce motor current
N
Move motor at medium velocity or up to max. velocity
Audible Chopper noise?
Y
CHOPCONF decrease *HEND* and increase *HSTART* (max. 7)
Finished or enable coolStep

Figure 22.2 Tuning StealthChop and SpreadCycle

Moving the Motor Using the Motion Controller

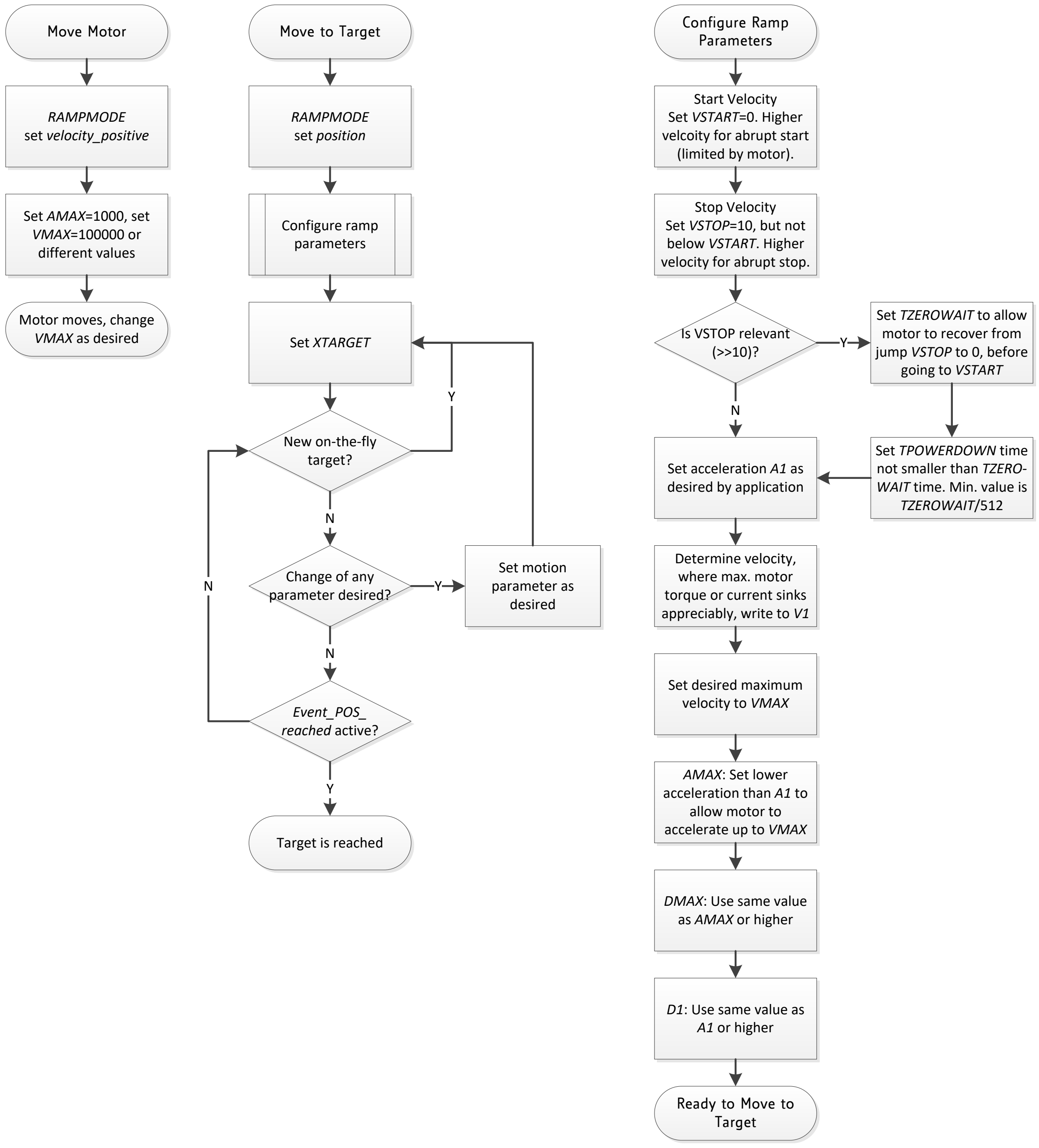
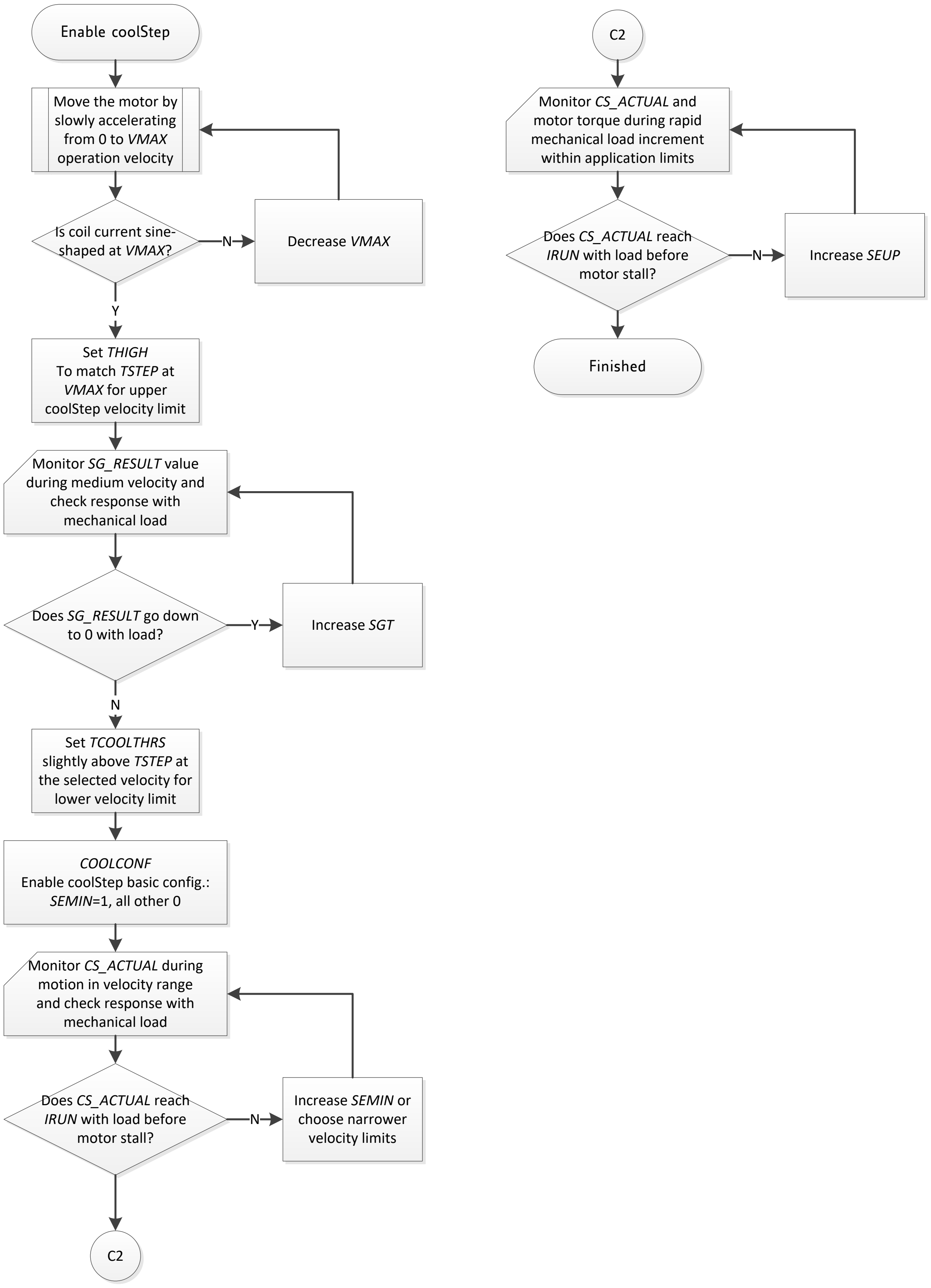

Move Motor
RAMPMODE set *velocity_positive*
Set *AMAX*=1000, set *VMAX*=100000 or different values
Motor moves, change *VMAX* as desired
Move to Target
RAMPMODE set *position*
Configure ramp parameters
Set *XTARGET*
New on-the-fly target?
Y
N
Change of any parameter desired?
Y
Set motion parameter as desired
N
Event_POS_reached active?
N
Y
Target is reached
Configure Ramp Parameters
Start Velocity Set *VSTART*=0. Higher velcoity for abrupt start (limited by motor).
Stop Velocity Set *VSTOP*=10, but not below *VSTART*. Higher velocity for abrupt stop.
Is VSTOP relevant (>>10)?
Y
Set *TZEROWAIT* to allow motor to recover from jump *VSTOP* to 0, before going to *VSTART*
Set *TPOWERDOWN* time not smaller than *TZEROWAIT* time. Min. value is *TZEROWAIT*/512
N
Set acceleration *A1* as desired by application
Determine velocity, where max. motor torque or current sinks appreciably, write to *V1*
Set desired maximum velocity to *VMAX*
AMAX: Set lower acceleration than *A1* to allow motor to accelerate up to *VMAX*
DMAX: Use same value as *AMAX* or higher
D1: Use same value as *A1* or higher
Ready to Move to Target

Figure 22.3 Moving the motor using the motion controller

ENABLING COOLSTEP (ONLY IN COMBINATION WITH SPREADCYCLE)



Enable coolStep
Move the motor by slowly accelerating from 0 to *VMAX* operation velocity
Is coil current sine-shaped at *VMAX*?
N
Decrease *VMAX*
Y
Set *THIGH* To match *TSTEP* at *VMAX* for upper coolStep velocity limit
Monitor *SG_RESULT* value during medium velocity and check response with mechanical load
Does *SG_RESULT* go down to 0 with load?
Y
Increase *SGT*
N
Set *TCOOLTHRS* slightly above *TSTEP* at the selected velocity for lower velocity limit
COOLCONF Enable coolStep basic config.: *SEMIN*=1, all other 0
Monitor *CS_ACTUAL* during motion in velocity range and check response with mechanical load
Does *CS_ACTUAL* reach *IRUN* with load before motor stall?
N
Increase *SEMIN* or choose narrower velocity limits
C2
C2
Monitor *CS_ACTUAL* and motor torque during rapid mechanical load increment within application limits
Does *CS_ACTUAL* reach *IRUN* with load before motor stall?
N
Increase *SEUP*
Finished

Figure 22.4 Enabling CoolStep (only in combination with SpreadCycle)

Setting up DcStep

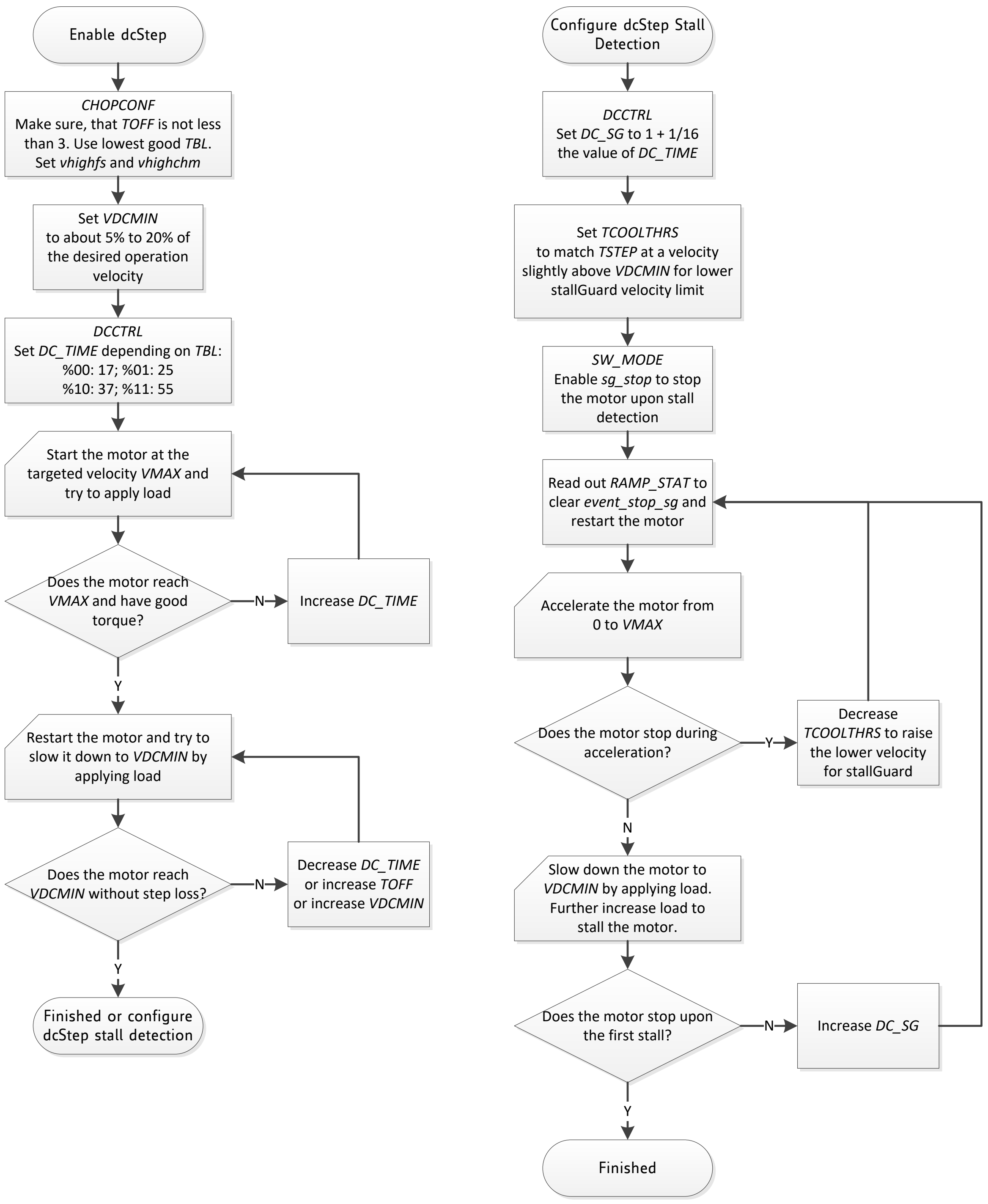

Enable dcStep
CHOPCONF
Make sure, that TOFF is not less than 3. Use lowest good TBL. Set vhighfs and vhighchm
Set VDCMIN to about 5% to 20% of the desired operation velocity
DCCTRL
Set DC_TIME depending on TBL: %00: 17; %01: 25 %10: 37; %11: 55
Start the motor at the targeted velocity VMAX and try to apply load
Does the motor reach VMAX and have good torque?
N
Increase DC_TIME
Y
Restart the motor and try to slow it down to VDCMIN by applying load
Does the motor reach VDCMIN without step loss?
N
Decrease DC_TIME or increase TOFF or increase VDCMIN
Y
Finished or configure dcStep stall detection
Configure dcStep Stall Detection
DCCTRL
Set DC_SG to 1 + 1/16 the value of DC_TIME
Set TCOOLTHRS to match TSTEP at a velocity slightly above VDCMIN for lower stallGuard velocity limit
SW_MODE
Enable sg_stop to stop the motor upon stall detection
Read out RAMP_STAT to clear event_stop_sg and restart the motor
Accelerate the motor from 0 to VMAX
Does the motor stop during acceleration?
Y
Decrease TCOOLTHRS to raise the lower velocity for stallGuard
N
Slow down the motor to VDCMIN by applying load. Further increase load to stall the motor.
Does the motor stop upon the first stall?
N
Increase DC_SG
Y
Finished

Figure 22.5 Setting up DcStep