AMIS-30621 LIN Microstepping Motordriver

Data Sheet

General Description

The AMIS-30621 is a member of a stepper motordriver family with position controller and control/diagnostics interface integrated in one single chip.
The family consists of two products:
• AMIS-30621 with LIN interface, ready to build dedicated mechatronics solutions connected remotely with a LIN master.
• AMIS-30622 with SERIAL interface, ready to act as peripheral device next to a microcontroller.

The chip receives high-level positioning instructions through the interface and subsequently drives the motor coils until the desired position is reached. The on-chip position controller is configurable (OTP and Interface) for different motor types, positioning ranges and parameters for speed, acceleration, and deceleration.

The AMIS-30621 acts as a slave on the bus and the master can fetch specific status information like actual position, error flags, etc. from each individual slave node.

Features

Motordriver
- Microstepping (1/2, 1/4, 1/8, 1/16)
- Low resonance & noise
- High resolution
- Programmable peak current up to 800mA
- 20kHz PWM current-control
- Automatic selection of fast & slow decay mode
- Internal fly-back FETs
- Fully integrated current sense
- 8V-29V supply voltage
- Automotive compliant
- Full diagnostics and status information

Controller with RAM and OTP Memory
- Position controller
- Configurable speeds, acceleration and deceleration
- Flexible hold-current
- Movement/position sensor-input
- Optional stall detection

LIN Interface
- Physical - and data-link-layer (conform to LIN rev. 1.3)
- Dynamically allocated indentifiers
- Up to 128 node addresses

Protection
- Over-current protection
- Under-voltage management
- Over-voltage protection
- High-temp warning and shutdown
- Low-temp warning
- LIN bus short-circuit protection to supply & ground

Power Saving
- Power-down supply current <50µA
- 5V regulator with wake-up on LIN activity

EMI compatibility
- Power drivers with slope control

Applications and Benefits

The AMIS-30621 is ideally suited for small positioning applications. Target markets include: automotive (headlamp alignment, HVAC, idle control, cruise), industrial equipment (lighting, fluid control, labeling, process, XYZ tables) and building automation (HVAC, surveillance, satellite dish positioning). Suitable applications typically have multiple axes or require mechatronic solutions with the driver chip mounted directly on the motor.

The high abstraction level of the products' command set reduces the load of the processor on the master side. Scaling of the application towards number of axes is straight-forward: hardware - and software designs are extended in a modular way, without severely effecting the demands on the master microcontroller. The bus structure simplifies PCB track-layout and/or wiring architectures.

Microstepping operation removes the design trade-off between minimal operation speed and avoiding the risk of noise and step-loss due to resonance phenomena. The stall-detection feature (optional) offers silent, yet accurate position-calibrations during the referencing run and allows semi-closed loop operation when approaching the mechanical end-stops.

All these benefits result in reduced system-cost and time-to-market and improved technical performance.

Ordering Information

Part N°	AMIS-30621
Package	SOIC-20
Peak Current	800mA
Temp. Range	-40°C...125°C

Table of Contents

Document History

Version	Date of Version
1.0	July 16th, 2002
1.1	October 18th, 2002
1.2	January 27th, 2003
1.3	February 19th, 2003
1.4	March 3rd, 2003

1.0 Quick Reference Data

1.1 Absolute Maximum Ratings

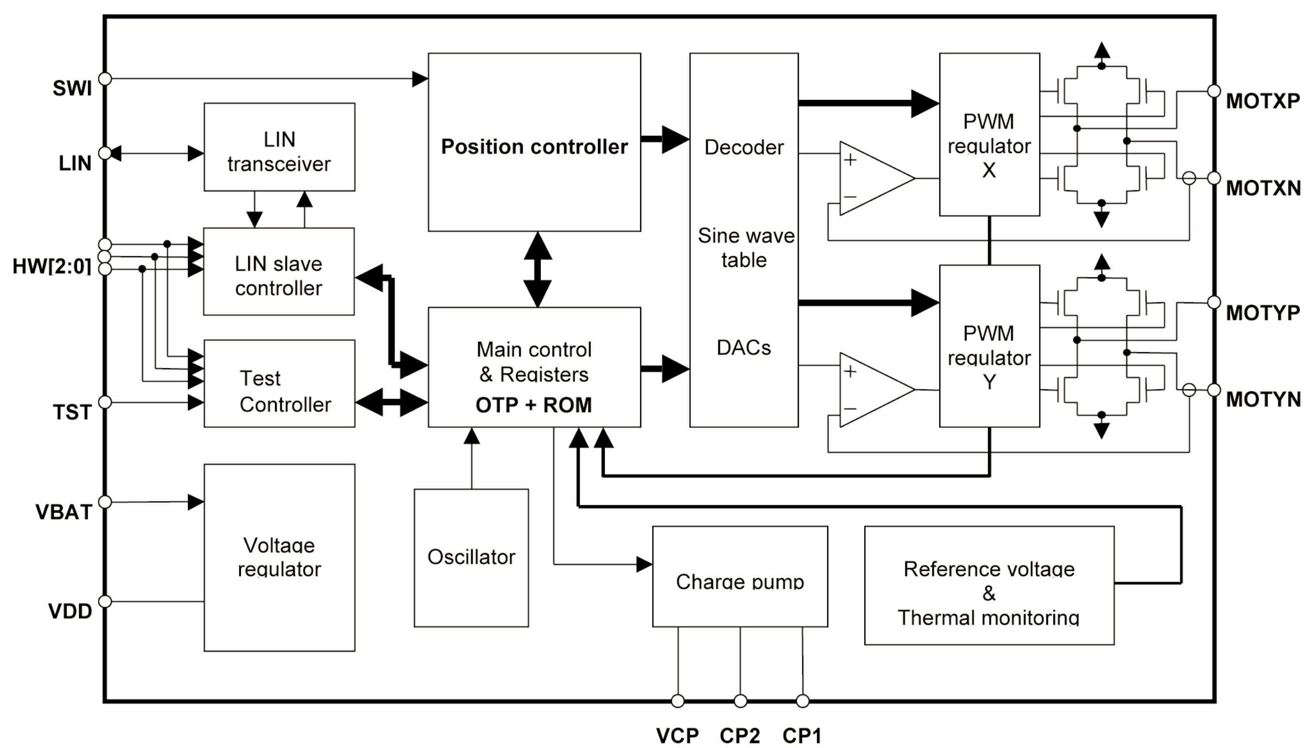
Parameter		Min	Max	Unit	
Vbb	Supply voltage	-0.3	+40[1]	V	
Vlin	Bus input voltage	-80	+80	V	
Tamb	Ambient temperature under bias[2]	-50	+150	°C	
Tst	Storage temperature	-55	+160	°C	
Vesd[3]	Electrostatic discharge voltage on LIN pin	-4	+4	kV	
	Electrostatic discharge voltage on other pins	-2	+2	kV	

Notes
(1) For limited time: < 0.5 s.
(2) The circuit functionality is not guaranteed.
(3) Human body model (100 pF via 1.5 kΩ, according to MIL std. 883E, method 3015.7).

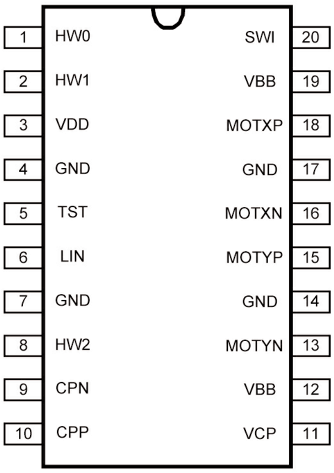
1.2 Operating Ranges

Parameter			Min	Max	Unit
Vbb	Supply voltage		+8	+29	V
Top	Operating temperature range	Vbb ≤ 18V	-40	+125	°C
		Vbb ≤ 29V	-40	+85	°C

2.0 Block Diagram



3.0 Pin-out



SOIC-20

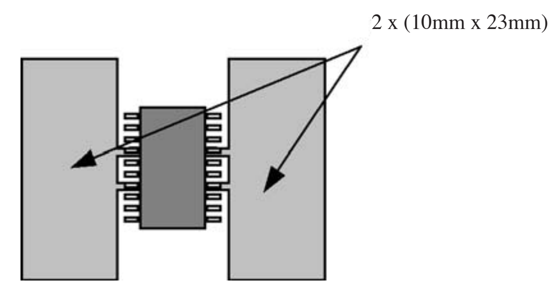
Pin Name	Pin Description		SOIC-20
HW0	Bit 0 of LIN-ADD	to be tied to GND	1
HW1	Bit 1 of LIN-ADD	or VDD if not used	2
VDD	Internal supply (needs external decoupling capacitor)	3	
GND	Ground, heat sink		4,7,14,17
TST	Test pin (to be tied to ground in normal operation)	5	
LIN	LIN-bus connection		6
HW2	Bit 2 of LIN-ADD		8
CPN	Negative connection of pump-capacitor (charge pump)		9
CPP	Positive connection of pump-capacitor (charge pump)		10
VCP	Charge-pump filter-capacitor		11
VBB	Battery voltage supply		12, 19
MOTYN	Negative end of phase Y coil		13
MOTYP	Positive end of phase Y coil		15
MOTXN	Negative end of phase X coil		16
MOTXP	Positive end of phase X coil		18
SWI	Switch input		20

4.0 Package Thermal Resistance

4.1 SO20

The junction-case thermal resistance is 28°C/W, leading to a junction-ambient thermal resistance of 63°C/W, with the PCB ground plane layout condition given on the figure beside, and with:

- PCB thickness = 1.6mm
- 1 layer
- Copper thickness = 35µm



5.0 DC Parameters

The DC parameters are given for Vbb and temperature in their operating ranges.
Convention: currents flowing in the circuit are defined as positive.

Symbol	Pin(s)	Parameter	Test Conditions	Min	Typ	Max	Unit
Motordriver							
IMSmax Peak	MOTXP MOTXN MOTYP MOTYN	Max current trough motor coil in normal operation			800		mA
IMSmax RMS		Max RMS current trough coil in normal operation			570		mA
IMSabs		Absolute error on coil current		-10		10	%
IMSrel		Error on current ratio Icoilx / Icoily		-7		7	%
RDSon		On resistance for each pin (including bond wire)	To be confirmed by characterization			1	Ω
IMSL		Pull down current	HZ mode		1		mA
LIN Transmitter							
Ibus_on	LIN	Dominant state, driver on	Vbus = 1.4V	40			mA
Ibus_off		Dominant state, driver off	Vbus = 0V	-1			mA
Ibus_off		Recessive state, driver off	Vbus = Vbat			20	μA
Ibus_lim		Current limitation		50		200	mA
Rslave		Pull-up resistance		20	30	47	kΩ
LIN Receiver							
Vbus_dom	LIN	Receiver dominant state		0		0.4	Vbb
Vbus_rec		Receiver recessive state		0.6		1	Vbb
Vbus_hys		Receiver hysteresis		0.05		0.2	Vbb
Thermal Warning & Shutdown							
Ttw		Thermal Warning		138	145	152	°C
Ttsd (1)		Thermal shutdown			Ttw+10		°C
Tlow		Low temperature warning			Ttw-155		°C
Supply & Voltage Regulator							
Vbb	VBB	Nominal operating supply range		6.5		18	V
VbbOTP		Supply voltage for OTP zapping		9.0		10.0	V
UV1		Stop voltage high threshold		8.8	9.4	9.8	V
UV2		Stop voltage low threshold		8.1	8.5	8.9	V
Ibat		Total current consumption	Unloaded outputs		10		mA
Ibat_s (2)		Sleep mode current consumption				50	μA
Vdd	VDD	Internal regulated output (3)	8V < Vbb < 18V Cload = 1μF (+100nF cer.)	4.75	5	5.25	V
IddStop		Digital current consumption	Vbb < UV2		2		mA
VddReset		Digital supply reset level (4)				4.4	V
IddLim		Current limitation	Pin shorted to ground			40	mA
Switch Input and Hardwire Address Input							
Rt_OFF	SWI HW2	Switch OFF resistance (5)	Switch to Gnd or Vbat,	10			kΩ
Rt_ON		Switch ON resistance (5)				2	kΩ
Vbb_sw		Vbb range for guaranteed operation of SWI and HW2		6		29	V
Vmax_sw		Maximum voltage	T < 1s			40V	V
Ilim_sw		Current limitation	Short to Gnd or Vbat		30		mA
Hardwired Address Inputs and Test Pin							
Vlow	HW0	Input level high		0.7		.	Vdd
Vhigh	HW1	Input level low				0.3	Vdd
HWhyst	TST	Hysteresis		0.075			Vdd
Charge Pump							
Vcp	VCP	Output voltage	Vbb > 15V Vbb > 8V	Vbb+10 Vbb+5.8V	Vbb+12.5	Vbb+15	V V
Cbuffer		External buffer capacitor		220		470	nF
Cpump	CPP CPN	External pump capacitor		220		470	nF

Notes
(1) No more than 100 cumulated hours in life time above Ttsd.
(2) To be confirmed by measurements.
(3) Pin VDD must not be used for any external supply.
(4) The RAM content will not be altered above this voltage.
(5) External resistance value seen from pin SWI or HW2, including 1kW series resistor.

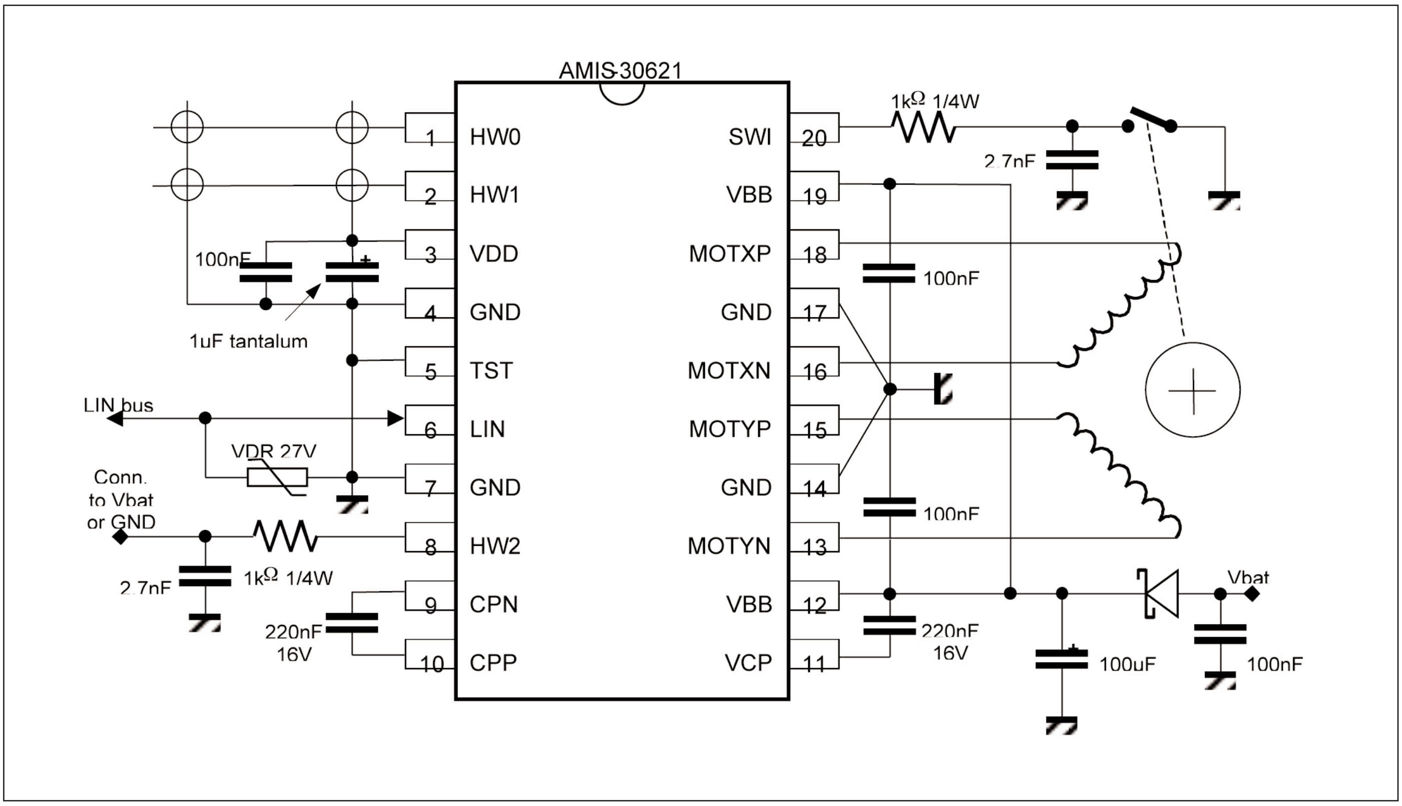
6.0 AC Parameters

The AC parameters are given for Vbb and temperature in their operating ranges.

Symbol	Pin(s)	Parameter	Test Conditions	Min	Typ	Max	Unit
Power-up							
Tpu		Power-up time				10	ms
Internal Oscillator							
Fosc		Frequency of internal oscillator		3.6	4.0	4.4	MHz
LIN Transmitter							
Slope_F/R	LIN	Slope falling (or rising) edge	Between 40% and 60%	0.1		3	V/µs
t_slope_F/R	LIN	Slope time falling (or rising) edge	Extrapolated	2.6		22.5	µs
T_tr_F	LIN	Propagation delay TxD low to bus		0.1	1	4	µs
T_tr_R	LIN	Propagation delay TxD high to bus		0.1	1	4	µs
t_slope_Sym		Slope time symmetry	t_slope_F – t_slope_R	-4		4	µs
Tsym_tr		Transmitter delay symmetry	T_tr_F – T_tr_R	-2		2	µs
LIN Receiver							
T_rec_F	LIN	Propagation delay bus dominant to TxD low		0.1	4	6	µs
T_rec_R	LIN	Propagation delay bus recessive to TxD high		0.1	4	6	µs
Tsym,Rec	LIN	Receiver delay symmetry		-2		2	µs
Twake	LIN	Wake-up delay time		50	100	200	µs
Switch Input and Hardwire Address Input							
Tsw	SWI	Scan pulse period (1)			1024		µs
Tsw_on	HW2	Scan pulse duration			1/16		Tsw
Motordriver							
Fpwm	MOTxx	PWM frequency (1)		18	20	22	kHz
Tbrise	MOTxx	Turn-on transient time	Between 10% and 90%		350		ns
Tbfall	MOTxx	Turn-off transient time			250		ns
Charge Pump							
FCP	CPN CPP	Charge pump frequency (1)			250		kHz

Note
(1) Derived from the internal oscillator.

7.0 Typical Application



Notes
(1) The switch can be connected to battery instead of ground.
(2) Resistors tolerance: ± 5%.
(3) 2.7nF capacitors: 2.7nF is the minimum value, maximum value is 10nF.
(4) Depending on the application the ESR value of the 1µF and 100µF capacitors must be carefully chosen.
(5) 100nF capacitors must be close to pins VBB and VDD.
(6) 220nF capacitors must be as close as possible to pins CPN, CPP, VCP and VBB to reduce EMC radiation.

8.0 Positioning Data

8.1 Stepping Modes

One of four possible stepping modes can be programmed:

- Half-stepping
- 1/4 micro-stepping
- 1/8 micro-stepping
- 1/16 micro-stepping

8.2 Maximum Velocity

For each stepping mode, Vmax can be programmed to 16 possible values given in the table below.

The accuracy of Vmax is derived from the internal oscillator. Under special circumstances it is possible to change the Vmax parameter while a motion is ongoing. All 16 entries for the Vmax parameter are divided into four groups. When changing Vmax during a motion the application must take care that the new Vmax parameter stays within the same group.

Vmax Index	Vmax (Full Step/s)	Group	Stepping Mode: Half-stepping (half-step/s)	1/4th Micro-stepping (micro-step/s)	1/8th Micro-stepping (micro-step/s)	1/16th Micro-stepping (micro-step/s)
0	99	A	197	395	790	1579
1	136		273	546	1091	2182
2	167		334	668	1335	2670
3	197	B	395	790	1579	3159
4	213		425	851	1701	3403
5	228		456	912	1823	3647
6	243		486	973	1945	3891
7	273		546	1091	2182	4364
8	303		607	1213	2426	4852
9	334	C	668	1335	2670	5341
10	364		729	1457	2914	5829
11	395		790	1579	3159	6317
12	456		912	1823	3647	7294
13	546		1091	2182	4364	8728
14	729	D	1457	2914	5829	11658
15	973		1945	3891	7782	15564

8.3 Minimum Velocity

Once Vmax is chosen, 16 possible values can be programmed for Vmin. The table below provides the obtainable values in Full-step/s. The accuracy of Vmin is derived from the internal oscillator.

Vmax Index	Vmax Factor	Vimax (Full-step/s): 99	136	167	197	213	228	243	273	303	334	364	395	456	546	729	973
0	1	99	136	167	197	213	228	243	273	303	334	364	395	456	546	729	973
1	1/32	3	4	5	6	6	7	7	8	8	10	10	11	13	15	19	27
2	2/32	6	8	10	11	12	13	14	15	17	19	21	23	27	31	42	57
3	3/32	9	12	15	18	19	21	22	25	27	31	32	36	42	50	65	88
4	4/32	12	16	20	24	26	28	30	32	36	40	44	48	55	65	88	118
5	5/32	15	21	26	31	32	35	37	42	46	51	55	61	71	84	111	149
6	6/32	18	25	31	36	39	42	45	50	55	61	67	72	84	99	134	179
7	7/32	21	30	36	43	46	50	52	59	65	72	78	86	99	118	156	210
8	8/32	24	33	41	49	52	56	60	67	74	82	90	97	113	134	179	240
9	9/32	28	38	47	55	59	64	68	76	84	93	101	111	128	153	202	271
10	10/32	31	42	51	61	66	71	75	84	93	103	113	122	141	168	225	301
11	11/32	34	47	57	68	72	78	83	93	103	114	124	135	156	187	248	332
12	12/32	37	51	62	73	79	85	91	101	113	124	135	147	170	202	271	362
13	13/32	40	55	68	80	86	93	98	111	122	135	147	160	185	221	294	393
14	14/32	43	59	72	86	93	99	106	118	132	145	158	172	198	237	317	423
15	15/32	46	64	78	93	99	107	113	128	141	156	170	185	214	256	340	454

Notes

(1) The Vmax factor is an approximation.

(2) In case of motion without acceleration (**AccShape** = 1) the length of the steps = 1/**Vmin**. In case of accelerated motion (**AccShape** = 0) the length of the first step is shorter than 1/**Vmin** depending of **Vmin**, **Vmax** and **Acc**.

8.4 Acceleration and Deceleration

Sixteen possible values can be programmed for Acc (acceleration and deceleration between Vmin and Vmax). The table below provides the obtainable values in Full-step/s². One observes restrictions for some combination of acceleration index and maximum speed (gray cells).
The accuracy of **Acc** is derived from the internal oscillator.

<table>
<tr><th>Vmax (FS/s) ′</th><th>99</th><th>136</th><th>167</th><th>197</th><th>213</th><th>228</th><th>243</th><th>273</th><th>303</th><th>334</th><th>364</th><th>395</th><th>456</th><th>546</th><th>729</th><th>973</th></tr>
<tr><th>ACC Index ↓</th><th colspan="16">Acceleration (Full-step/s²)</th></tr>
<tr><td>0</td><td colspan="7">49</td><td colspan="6">106</td><td colspan="3">473</td></tr>
<tr><td>1</td><td colspan="13">218</td><td colspan="3">735</td></tr>
<tr><td>2</td><td colspan="16">1004</td></tr>
<tr><td>3</td><td colspan="16">3609</td></tr>
<tr><td>4</td><td colspan="16">6228</td></tr>
<tr><td>5</td><td colspan="16">8848</td></tr>
<tr><td>6</td><td colspan="16">11409</td></tr>
<tr><td>7</td><td colspan="16">13970</td></tr>
<tr><td>8</td><td colspan="16">16531</td></tr>
<tr><td>9</td><td rowspan="5">14785</td><td colspan="15">19092</td></tr>
<tr><td>10</td><td colspan="15">21886</td></tr>
<tr><td>11</td><td colspan="15">24447</td></tr>
<tr><td>12</td><td colspan="15">27008</td></tr>
<tr><td>13</td><td colspan="15">29570</td></tr>
<tr><td>14</td><td colspan="7" rowspan="2">29570</td><td colspan="9">34925</td></tr>
<tr><td>15</td><td colspan="9">40047</td></tr>
</table>

The formula to compute the number of equivalent Full-step during acceleration phase is:

$$\text{Nstep} = \frac{\text{Vmax}^2 - \text{Vmin}^2}{2 \times \text{Acc}}$$

8.5 Positioning

The position programmed in commands **SetPosition** and **SetPositionShort** is given as a number of (micro)steps. According to the chosen stepping mode, the position words must be aligned as described in the table below. When using command **SetPositionShort** or **GotoSecurePosition**, data is automatically aligned.

Stepping Mode	Position Word: Pos [15 : 0]																Shift
$1/16^{th}$	S	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB	No shift
$1/8^{th}$	S	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB	0	1-bit left ↔ x2
$1/4^{th}$	S	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB	0	0	2-bit left ↔ x4
Half-stepping	S	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB	0	0	0	3-bit left ↔ x8
PositionShort	S	S	S	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB	0	0	0	No shift
SecurePosition	S	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB	0	0	0	0	0	No shift

8.5.1 Position Ranges

A position is coded by using the binary two's complement format. According to the positioning commands which are used (see § 9.2.2.11) and to the chosen stepping mode, the position range will be as shown in the table below.

Command	Stepping Mode	Position Range	Full Range	Number
`SetPosition`	Half-stepping	-4096 to +4095	8192 half-steps	13
	$1/4^{th}$ micro-stepping	-8192 to +8191	16384 micro-steps	14
	$1/8^{th}$ micro-stepping	-16384 to +16383	32768 micro-steps	15
	$1/16^{th}$ micro-stepping	-32768 to +32767	65536 micro-steps	16
`SetPositionShort`	Half-stepping	-1024 to +1023	2048 half-steps	11

When using the command **`SetPosition`**, although coded on 16 bits, the position word will have to be shifted on the left by a certain number of bits, according to the chosen stepping mode.

8.5.2 Secure Position

A secure position can be programmed. It is coded on 11-bit, thus having a lower resolution than normal positions, as shown in the table below. See command **GotoSecurePosition**.

Stepping Mode	Secure Position Resoultion
Half-stepping	4 half-steps
1/4th micro-stepping	8 micro-steps (1/4th)
1/8th micro-stepping	16 micro-steps (1/8th)
1/16th micro-stepping	32 micro-steps (1/16th)

Important note

The secure position is disabled in case the programmed value is the reserved code "10000000000"
(most negative position).

8.5.3 Shaft

A shaft bit can be programmed to define whether a positive motion is an outer or an inner motion:
• Shaft = 0 → MOTXP is used as positive pin of the X coil, while MOTXN is the negative one.
• Shaft = 1 → opposite situation.

9.0 Functional Description

9.1 Structure Description

9.1.1 Stepper Motordriver

The Motordriver receives the control signals from the control logic. It mainly features:

- Two H-bridges designed to drive a two separated coils stepper motor. Each coil (X and Y) is driven by one H-bridge, and the driver controls the currents flowing through the coils.

The rotational position of the rotor, in unloaded condition, is defined by the ratio of current flowing in X and Y. The torque of the stepper motor when unloaded is controlled by the magnitude of the currents in X and Y.

- The control block for the H-bridges including the PWM control, the synchronous rectification and the internal current sensing circuitry
- The charge pump to allow driving of the H-bridges' high side transistors.
- Two pre-scale 4-bit DACs to set the maximum magnitude of the current through X and Y.
- Two DACs to set the correct current ratio through X and Y.

Battery voltage monitoring is also performed by this block which provides needed information to the control logic part. The same applies for detection and reporting of an electrical problem that could occur on the coils or the charge pump.

9.1.2 Control Logic (Position Controller and Main Control)

The control logic block stores the information provided by the LIN interface (in a RAM or an OTP memory) and digitally controls the positioning of the stepper motor in terms of speed and acceleration, by feeding the right signals to the motordriver state machine.

It will take into account the successive positioning commands to initiate or stop properly the stepper motor in order to reach the set point in a minimum time.

It also receives feedback from the motordriver part in order to manage possible problems and decide about internal actions and reporting to the LIN interface.

9.1.3 LIN Interface

The LIN interface implements the physical layer and the MAC and LLC layers according to the OSI Reference Model. It provides and gets information to and from the Control logic block, in order to drive the stepper motor, to configure the way this motor must be driven, or to get information such as actual position or diagnosis (temperature, battery voltage, electrical status...) and pass it to the LIN master node.

9.1.4 Miscellaneous

The AMIS-30621 also implements the followings:

- An internal oscillator, needed for the LIN protocol handler as well as for the Control logic and for the PWM control of the motordriver.
- An internal trimmed voltage source for precise referencing.
- A protection block featuring a Thermal Shutdown and a Power-on-reset circuit.
- A 5V regulator (from the battery supply) to supply the internal logic circuitry.

9.2 Functions Description

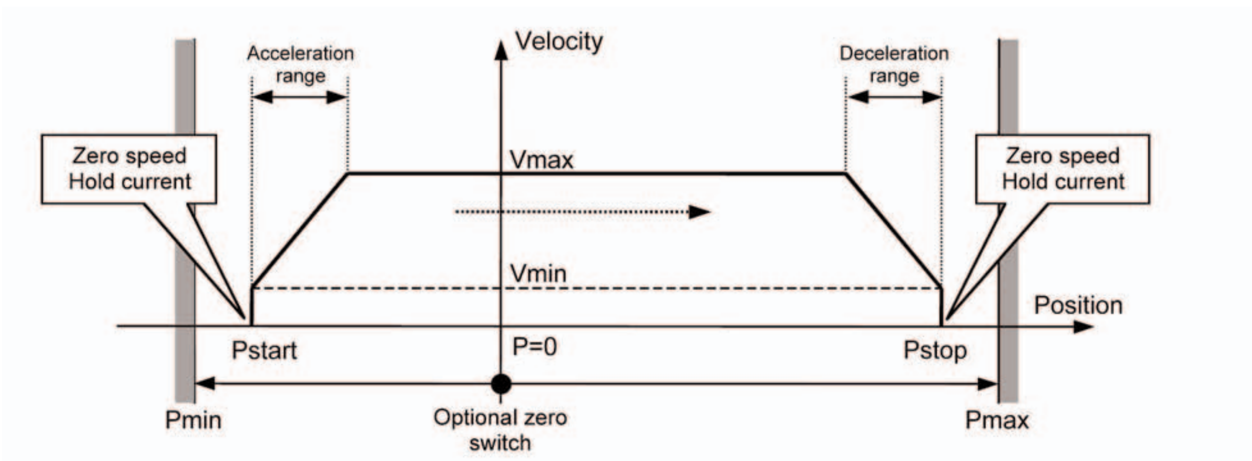
This chapter describes the four most important blocks:

- Position controller
- Main control and register, OTP memory + ROM
- Motordriver
- LIN controller

9.2.1 Position Controller

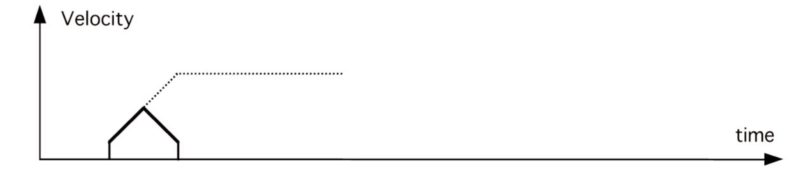
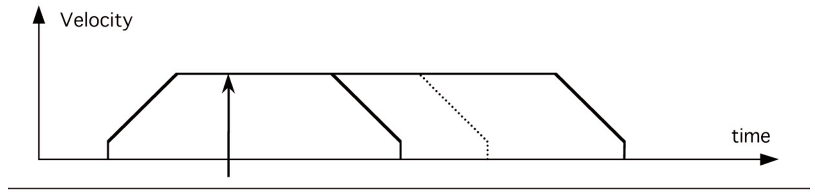
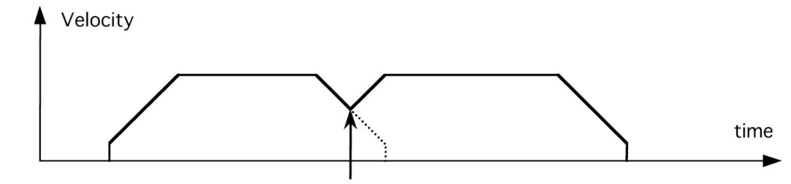
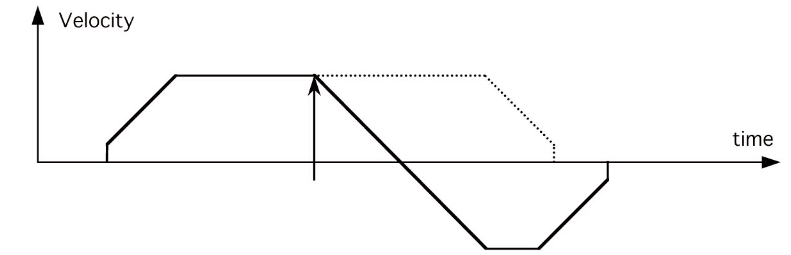
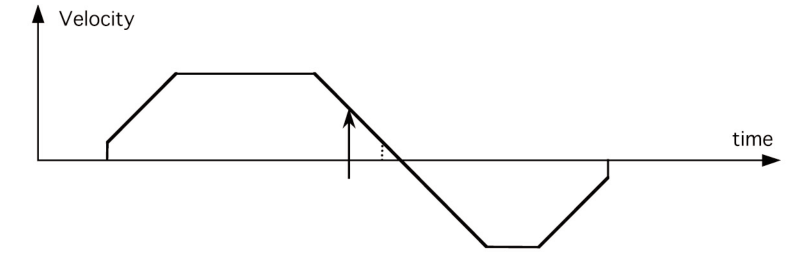
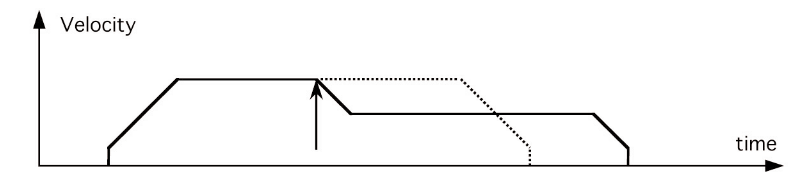
9.2.1.1 Positioning and Motion Control

A positioning command will produce a motion as illustrated below. A motion starts with an acceleration phase from minimum velocity (Vmin) to maximum velocity (Vmax), and ends with a symmetrical deceleration. This is defined by the Control logic according to the position required by the application and to the parameters programmed by the application during configuration phase. The current in the coils is also programmable.



Parameter	Value
Pmax - Pmin	See § 8.5
Zero speed Hold Current	See § 9.2.2.13 (**`Ihold`**)
Maximum current	See § 9.2.2.13 (**`Irun`**)
Acceleration and deceleration	See § 8.4
Vmin	See § 8.3
Vmax	See § 8.2

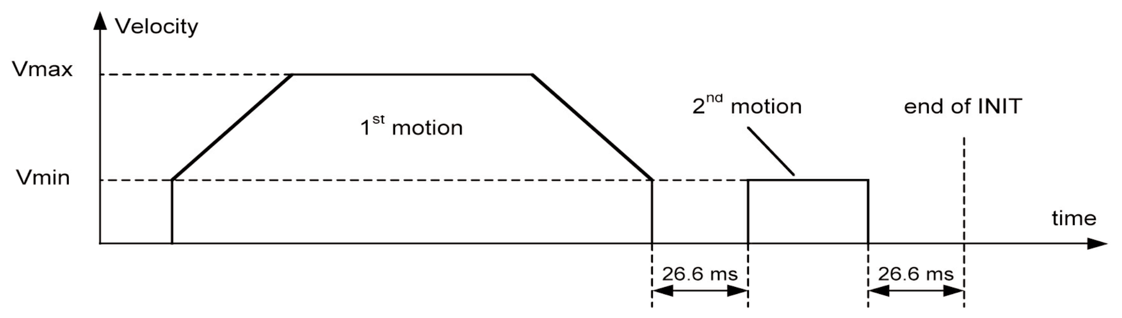
Different positioning examples are shown in the table below.

Short motion	
New positioning command in same direction, shorter or longer, while a motion is running at maximum velocity	
New positioning command in same direction while in deceleration phase Note: there is no wait time between the deceleration phase and the new acceleration phase.	
New positioning command in reverse direction while motion is running at maximum velocity	
New positioning command in reverse direction while in deceleration phase	
New velocity programming while motion is running	

9.2.1.2 Position Initialization

After power-up or when a Vdd reset has been acknowledged to the master, a position initialization of the stepper-motor can be requested by the application, by use of the **RunInit** command (see § 9.2.2.11). The position initialization is performed by the position controller under the control of the Main control block. This operation cannot be interrupted or influenced by any further command. A position initialization can only be interrupted by the occurrence of the conditions driving to a Motor shutdown (see § 9.2.2.8) or by a **HardStop** command. On the other hand, sending a **RunInit** command while a motion is already ongoing is not recommended.

A position initialization consists of 2 successive motions, as illustrated below.



The first motion is done with the specified **Vmin** and **Vmax** velocities in the **RunInit** command, with the acceleration (deceleration) parameter already in RAM, to a position **Pos1[15:0]** also specified in **RunInit**. The goal here is to perform a motion large enough to reach a stall position (considered to be the reference position).

Then a second motion to a position **Pos2[15:0]** is done at the specified **Vmin** velocity in the **RunInit** command (no acceleration). The purpose of this second motion is to confirm with a low velocity the positioning of the motor at the stall position, assuming that the stepper motor may have bounced against the stall position. Therefore, **Pos2** should only be a few half or micro steps further than **Pos1**, in order to perform a displacement of at least one electrical period.

Once the second motion is achieved, the **ActPos** register (see § 0) is reset to zero, to set the reached position as the reference position, whereas **TagPos** register is not changed.

Notes

(0) The priority encoder (see § 9.2.2.12) is describing the management of states and commands. The notes below are to be considered illustrative.

(1) The last **SetPosition(Short)** command issued during an initialization sequence will be kept in memory and executed afterwards. This applies also for the commands **Sleep** and **SetMotorParam** and **GotoSecurePosition**.

(2) Commands such as **GetActualPos** or **GetStatus** will be executed while the position initialization is running. This applies also for a dynamic ID assignment LIN frame (see § 9.2.4.6.4).

(3) An initialization sequence starts by setting **TagPos** register to **SecPos** value, provided secure position is enabled otherwise **TagPos** is reset to zero.

(4) The acceleration/deceleration value applied during an initialization sequence is the one stored in RAM before the **RunInit** command is sent. The same applies for Shaft bit, but not for **Irun**, **Ihold** and **StepMode**, which can be changed during an initialization sequence.

(5) The **Pos1**, **Pos2**, **Vmax**, and **Vmin** values programmed in a **RunInit** command apply only for a this initialization sequence. All further positioning will use the parameters stored in RAM (programmed for instance by a former **SetMotorParam** command).

(6) Commands **ResetPosition**, **RunInit** and **SoftStop** will be ignored while an initialization sequence is ongoing, and will not be executed afterwards.

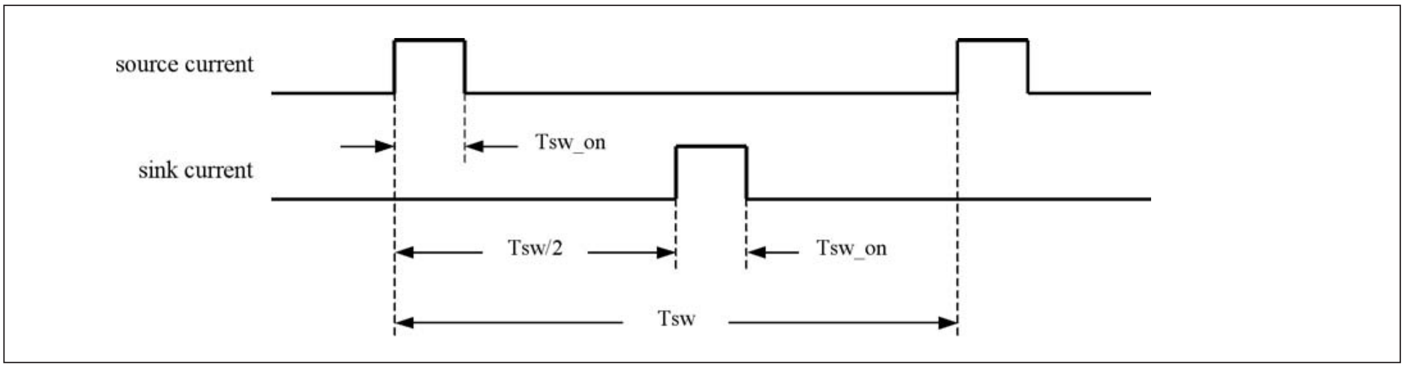
(7) A **SetMotorParam** command should not be sent during an initialization sequence.

(8) If for some reason **ActPos** equals **Pos1[15:0]** at the moment the **RunInit** command is issued, the circuit will enter in deadlock state. Therefore, the application should check the actual position by a **GetPosition** or a **GetFullStatus** command prior to an initialization. Another solution may consist in programming a value out of the stepper motor range for **Pos1[15:0]**.

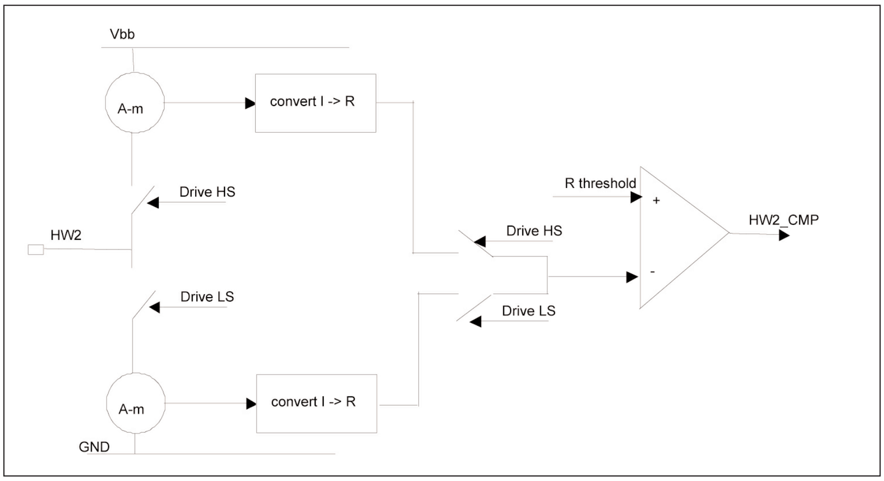
9.2.1.3 External Switch and HW2 Pin

Pin SWI and hardwired address pin HW2 (see § 9.2.4.5) will alternatively attempt to source and sink a current in/from the external switch (see application schematic) to test whether it is ON or OFF. This current is set around 10mA when a 1kW external series resistor is used.

This can be represented by the following time diagram (The timings are given in § 6).



If the switch is detected ON (closed), then the flag <ESW> is raised. The status of this flag can be read by the application via a **`GetActualPos`**, a **`GetStatus`** or a **`GetFullStatus`** reading frame. At the falling edge of every current pulse (at around 1kHz), the stepper-motor actual position is refreshed (register **`ActPos`**, see § 9.2.2.9), so that the master node may get synchronous information about the state of the switch together with the position of the motor. The position is then given with an accuracy of ± 1 half-step (or micro-step, depending of the programmed stepping mode). The block diagram below shows how this function is implemented for HW2.



With the following truth table:

State	Drive LS	Drive HS	HW2_CMP	New State
Float	1	0	0	Float
Float	1	0	1	HW2Hi
Float	0	1	0	Float
Float	0	1	1	HW2Lo
HW2Lo	1	0	0	HW2Lo
HW2Lo	1	0	1	HW2Hi
HW2Lo	0	1	0	Float
HW2Lo	0	1	1	HW2Lo
HW2Hi	1	0	0	Float
HW2Hi	1	0	1	HW2Hi
HW2Hi	0	1	0	HW2Hi
HW2Hi	0	1	1	HW2Lo

HW2Hi	address = "1"
HW2Lo	address = "0"
HW2_CMP	CMP output; active high if low resistance detected by the LS or HS Ohm-meter
Drive LS	request from digital to turn on the low-side part of the Ohm-meter
Drive HS	request from digital to turn on the high-side part of the Ohm-meter

note that e.g. if HW2 is connected to GND, LS-part will report "float" while HS-part will report "low resistance detected"

Note:
If HW2 is detected to be floating, motion to the secure position is performed.

9.2.2 Main Control and Register, OTP Memory + ROM

9.2.2.1 Power-up Phase

Power-up phase of the AMIS-30621 will not exceed 10ms. After this phase, the AMIS-30621 is in Shutdown mode, ready to receive LIN messages and to execute the associated commands. After power-up, the registers and flags are in the Reset state, some of them being loaded with the OTP memory content (see § 9.2.2.2).

9.2.2.2 Reset State

After power-up, or after a reset occurrence (e.g. a micro cut on pin VBB has made Vdd to go below VddReset level), the H-bridges will be in high impedance mode, and the registers and flags will be in a predetermined position. This is documented in § 0 and § 9.2.2.10.

9.2.2.3 Soft Stop

A Soft Stop is an immediate interruption of a motion, but with a deceleration phase. At the end of this action, the register **`TagPos`** is loaded with the value contained in register **`ActPos`** to avoid an attempt of the circuit to achieve the motion (see § 0).

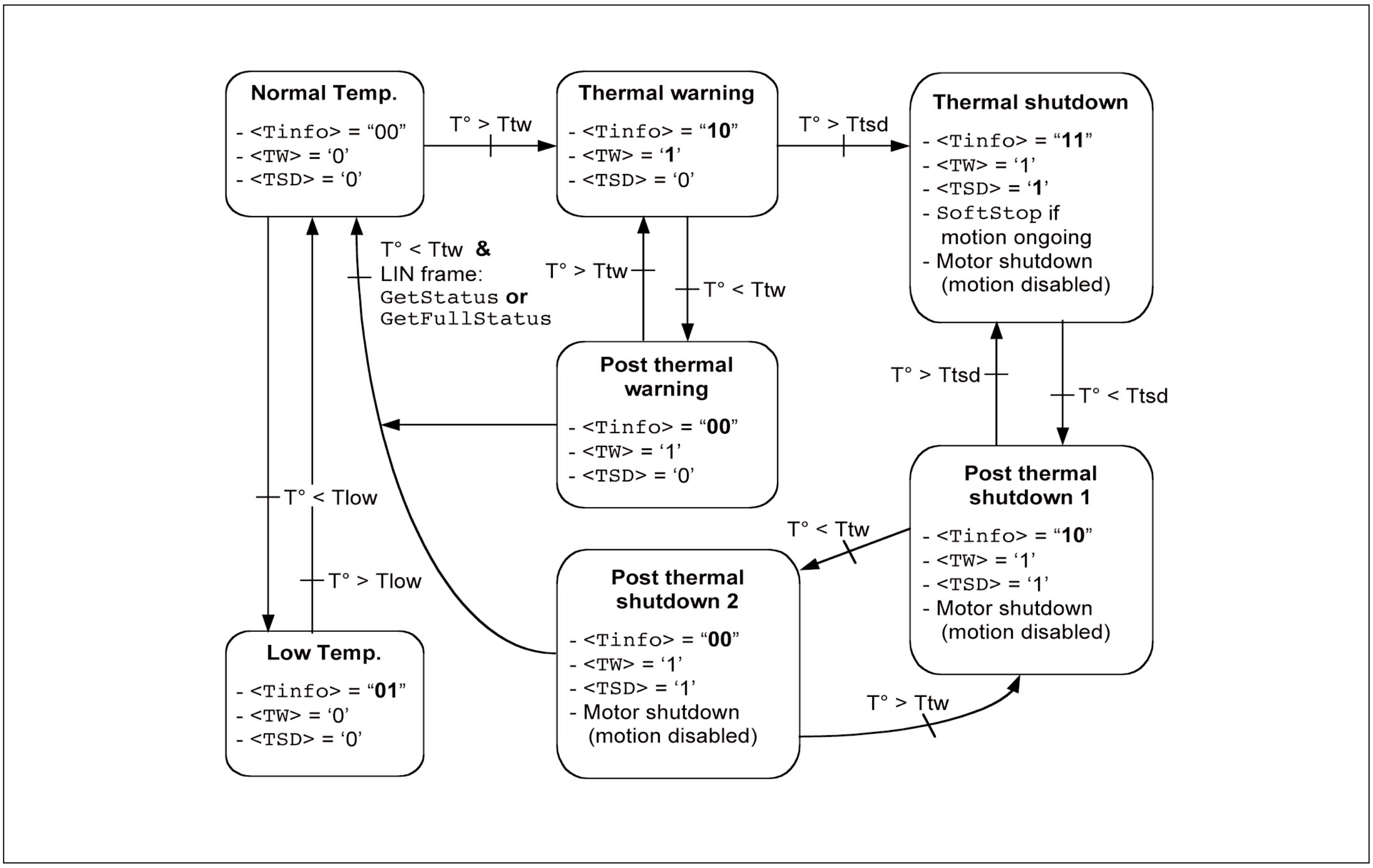
The circuit is then ready to execute a new positioning command, provided thermal and electrical conditions allow for it.

9.2.2.4 Thermal Shutdown Mode

When thermal shutdown occurs, the circuit performs a **`SoftStop`** command and goes to Motor shutdown mode (see below).

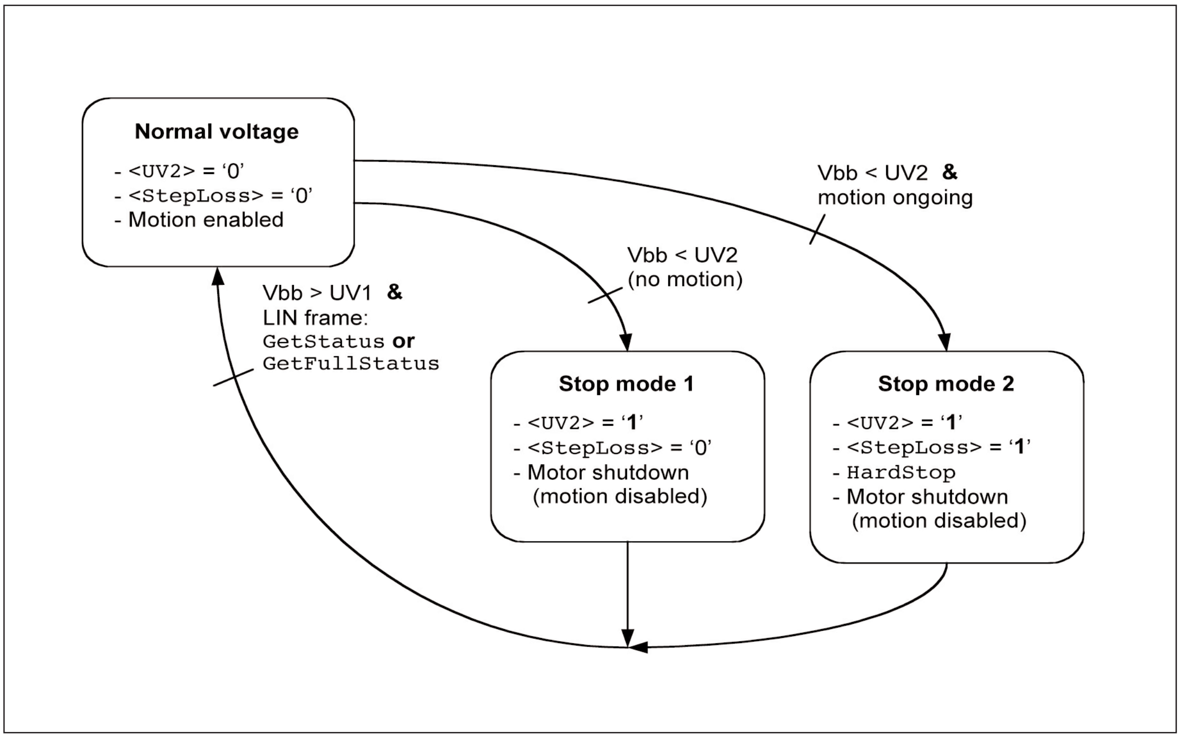
9.2.2.5 Temperature Management

The AMIS-30621 monitors temperature by mean of two thresholds and one shutdown level, as illustrated in the state diagram below. The only condition to reset flags <**`TW`**> and <**`TSD`**> (respectively Thermal Warning and Thermal Shutdown) is to be at a temperature lower than Ttw and to get the occurrence of a **`GetStatus`** or a **`GetFullStatus`** LIN frame.



9.2.2.6 Battery Voltage Management

The AMIS-30621 monitors the battery voltage by means of one threshold and one shutdown level, as illustrated in the state diagram below. The only condition to reset flags <**UV2**> and <**StepLoss**> is to recover a battery voltage higher than UV1 and to receive a **GetStatus** or a **GetFullStatus** command.



9.9.2.7 Sleep Mode

When entering in Sleep mode, the stepper-motor can be driven to its secure position. After which, the circuit is completely powered down, apart from the LIN receiver which remains active to detect dominant state on the bus.
In case sleep mode is entered while a motion is ongoing, a transition will occur towards secure position as described in § 9.2.1. provided **SecPos** is enabled, otherwise **SoftStop** is performed.

Sleep mode can be entered in the following cases:
- The circuit receives a LIN frame with identifier **0x3C** and first Data byte containing **0x00**, as required by LIN specification rev 1.2.
- The LIN bus remains inactive (or is lost) during more than 25000 time slots (1.30 s at 19.2 kbit/s), after which a time-out signal switches the circuit to sleep mode.

The circuit will return to normal mode if a valid LIN frame is received while entering the Sleep mode (this valid frame can be addressed to another slave).

9.2.2.8 Motor Shutdown Mode

A motor shutdown occurs when:

- The chip temperature rises above the thermal shutdown threshold Ttsd (see § 5)
- The battery voltage goes below UV2 (see § 5)
- Flag <**ElDef**> = '1', meaning an electrical problem is detected on one or both coils, e.g. a short circuit
- Flag <**CPFail**> = '1', meaning there is a charge pump failure

A motor shutdown leads to the followings:

- H-bridges in high impedance mode
- The **TagPos** register is loaded with the ActPos (to avoid any motion after leaving the motor shutdown mode)

The LIN interface remains active, being able to receive orders or send status.

The conditions to get out of a motor shutdown mode are:

- Reception of a **GetStatus** or **GetFullStatus** command AND
- The four above causes are no more detected

Which leads to H-bridges in Ihold mode. Hence the circuit is ready to execute any positioning command.

This can be illustrated in the following sequence given as an application tip. The Master can check whether there is a problem or not and decide which application strategy to adopt.

Tj ≥ Tsd or Vbb ≤ UV2 or <ElDef> = '1' or <CpFail> = '1' ↓	SetPosition frame ↑	GetFullStatus or GetStatus frame ↑	GetFullStatus or GetStatus frame ↑ . . .
- The circuit is driven in motor shutdown mode - The application is <u>not</u> aware of this	- The position set-point is updated by the LIN Master - Motor shutdown mode ⇒ no motion - The application is still unaware	- The application is aware of a problem	- Possible confirmation of the problem
		- Reset <TW> or <TSD> or <UV2> or <StepLoss> or <ElDef> or <CPFail> by the application - Possible new detection of over temperature or low voltage or electrical problem ⇒ Circuit sets <TW> or <TSD> or <UV2> or <StepLoss> or <ElDef> or <CPFail> again at '1'	

Important

While in shutdown mode, since there is no hold current in the coils, the mechanical load can cause a step loss, which indeed cannot be flagged by the AMIS-30621.

Warning

The application should limit the number of consecutive **GetStatus** or **GetFullStatus** commands to try to get the AMIS-30621 out of Shutdown mode when this proves to be unsuccessful, e.g. there is a permanent defect. The reliability of the circuit could be altered since **Get(Full)Status** attempts to disable the protection of the H-bridges.

Note

(0) The priority encoder (see § 9.2.2.12) is describing the management of states and commands. The note below is to be considered illustrative.

(1) If the LIN communication is lost while in shutdown mode, the circuit enters the sleep mode immediately.

9.2.2.9 RAM Registers

Register	Mnemonic	Length (bit)	Related Commands	Comment	Reset State

Actual Position	`ActPos`	16	`GetActualPos`
			`GetFullStatus`
			`GotoSecurePos`
			`ResetPosition`
				- 16-bit signed	Note 1

Last Programmed Position	`Pos/`
	`TagPos`
		16/11	`GetFullStatus`
			`GotoSecurePos`
			`ResetPosition`
			`SetPositionShort`
				- 16-bit signed or
				- 11-bit signed for half stepping
				(see § 8.5)	

Acceleration Shape	`AccShape`	1	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorParam`
				'0' → normal acceleration from Vmin to Vmax
				'1' → motion at Vmin without acceleration
					'0'

Coil Peak Current	`Irun`	4	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorParam`
				Operating current (see § 9.2.2.13)	From OTP memory

Coil Hold Current	`Ihold`	4	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorParam`
				Standstill current (see § 9.2.2.13)	

Minimum Velocity	`Vmin`	4	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorParam`
				See § 8.3 and § 9.2.2.13
				(look-up table)	

Maximum Velocity	`Vmax`	4	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorParam`
				See § 8.2 and § 9.2.2.13
				(look-up table)	

Shaft	`Shaft`	1	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorParam`
				Direction of movement for positive velocity	

Acceleration/ Deceleration	`Acc`	4	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorParam`
				See § 8.4 and § 9.2.2.13
				(look-up table)	

Secure Position	`SecPos`	11	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorPara`
				Target position when LIN connection fails; 11 MSBs of 16-bit position (LSBs fixed to '0')	

Stepping Mode	`StepMode`	2	`GetFullStatus`
			`ResetToDefault`[1]
			`SetMotorParam`
				See § 8.1 and § 9.2.2.13	

Note
(1) A **`ResetToDefault`** command will act as a reset of the RAM content, except for **`ActPos`** and **`TagPos`** registers that are not modified. Therefore, the application should not send a **`ResetToDefault`** during a motion, to avoid any unwanted change of parameter.

9.2.2.10 Flags Table

Register	Mnemonic	Length (bit)	Related Commands	Comment	Reset State
Charge Pump Failure	**CPFail**	1	**GetFullStatus**	‘0’ = charge pump OK ‘1’ = charge pump failure reset only after **GetFullStatus**	‘0’
Electrical Defect	**ElDef**	1	**GetActualPos** **GetStatus** **GetFullStatus**	<**OVC1**> or <**OVC2**> or <**open circuit 1**> or <**open circuit 2**> or <**CPFail**> resets only after **Get(Full)Status**	‘0’
External Switch Status	**ESW**	1	**GetActualPos** **GetStatus** **GetFullStatus**	‘0’ = open ‘1’ = close	‘0’
Electrical Flag	**HS**	1	Internal use	<**CPFail**> or <**UV2**> or <**ElDef**> or <**VDDreset**>	‘0’
Motion Status	**Motion**	3	**GetFullStatus**	“x00” = Stop “001” = inner motion acceleration “010” = inner motion deceleration “011” = inner motion max. speed “101” = outer motion acceleration “110” = outer motion deceleration “111” = outer motion max. speed	“000”
Over Current in Coil X	**OVC1**	1	**GetFullStatus**	‘1’ = over current reset only after **GetFullStatus**	‘0’
Over Current in Coil Y	**OVC2**	1	**GetFullStatus**	‘1’ = over current reset only after **GetFullStatus**	‘0’
Secure Position Enabled	**SecEn**	1	Internal use	‘0’ if **SecPos** = “100 0000 0000” ‘1’ otherwise	n.a.
Circuit Going to Sleep Mode	**Sleep**	1	Internal use	‘1’ = Sleep mode reset by LIN command	‘0’
Step Loss	**StepLoss**	1	**GetActualPos** **GetStatus** **GetFullStatus**	‘1’ = step loss due to under voltage, over current or open circuit	‘1’
Motor Stop	**Stop**	1	Internal use	See § 9.2.2.12	‘0’
Temperature Info	**Tinfo**	2	**GetActualPos** **GetStatus** **GetFullStatus**	“00” = normal temperature range “01” = low temperature warning “10” = high temperature warning “11” = motor shutdown	“00”
Thermal Shutdown	**TSD**	1	**GetActualPos** **GetStatus** **GetFullStatus**	‘1’ = shutdown (> 155°C typ.) reset only after **Get(Full)Status** and if <**Tinfo**> = “00”	‘0’
Thermal Warning	**TW**	1	**GetActualPos** **GetStatus** **GetFullStatus**	‘1’ = over temp. (> 145°C) reset only after **Get(Full)Status** and if <**Tinfo**> = “00”	‘0’
Battery Stop Voltage	**UV2**	1	**GetActualPos** **GetStatus** **GetFullStatus**	‘0’ = Vbb > UV2 ‘1’ = Vbb ≤ UV2 reset only after **Get(Full)Status**	‘0’
Digital Supply Reset	**VddReset**	1	**GetActualPos** **GetStatus** **GetFullStatus**	Set at ‘1’ after power-up of the circuit. If this was due to a supply micro-cut, it warns that the RAM contents may have been lost; can be reset to ‘0’ with a **GetStatus** or a **GetFullStatus** command.	‘1’

9.2.2.11 Application Commands

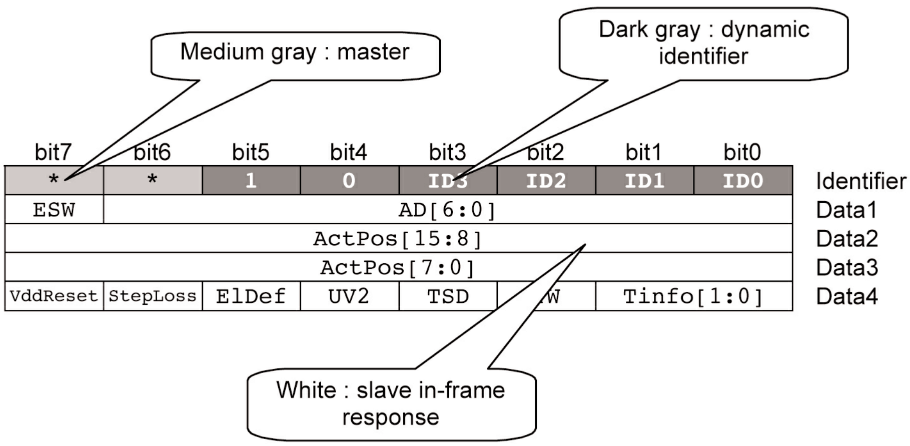
The LIN Master will have to use commands to manage the different application tasks the AMIS-30621 can feature. The commands summary is given in the table below.

Command Mnemonic	Feature
`GetActualPos`	Returns the actual position of the motor
`GetFullStatus`	Returns a complete status of the circuit
`GetOTPparam`	Returns the OTP memory content
`GetStatus`	Returns a short status of the circuit
`GotoSecurePosition`	Drives the motor to its secure position
`HardStop`	Immediate motor stop
`ResetPosition`	Actual position becomes the zero position
`ResetToDefault`	RAM content reset
`RunInit`	Initialization sequence
`SetMotorParam`	Programs the motion parameters and values for the current in the motor's coils
`SetOTPparam`	Programs (and zaps) a selected byte of the OTP memory
`SetPosition`	Drives the motor to a given position
`SetPositionShort (1 motor)`	Drives the motor to a given position (half stepping mode only)
`SetPositionShort (2 motors)`	Drives 2 motors to 2 given positions (half stepping mode only)
`SetPositionShort (4 motors)`	Drives 4 motors to 4 given positions (half stepping mode only)
`Sleep`	Drives circuit into sleep mode
`SoftStop`	Motor stopping with a deceleration phase

These commands are described hereafter, with their corresponding LIN frames. One should also refer to § 9.2.4.6 for more details on LIN frames, particularly for what concerns dynamic assignment of identifiers. A gray scale coding is used to distinguish between master and slave parts within the frames and to highlight dynamic identifiers. An example is shown below.

Usually, the AMIS-30621 makes use of dynamic identifiers for general-purpose 2, 4 or 8 bytes writing frames. If dynamic identifiers are used for other purpose, this is acknowledged.

Some frames implement a **`Broad`** bit that allows to address a command to all the AMIS-30621 circuits connected to the same LIN bus. **`Broad`** is active when at '0', in which case the physical address provided in the frame is thus not taken into account by the slave nodes.



GetActualPos

This command is provided to the circuit by the LIN Master to get the actual position of the Stepper-motor. This position (**ActPos[15:0]**) is returned in signed two's complement 16-bit format. One should note that according to the programmed stepping mode, the LSBs of **ActPos[15:0]** may have no meaning and should be assumed to be at '0', as described in § 8.5.

GetActualPos provides also a quick status of the circuit and of the Stepper-motor, identical to that obtained by command **GetStatus** (see further).

Note

A **GetActualPosition** command will not attempt to reset any flag.

GetActualPos corresponds to the following LIN reading frames.

1) 4 data bytes in-frame response with direct ID (type #5).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	1	0	ID3	ID2	ID1	ID0	Identifier
ESW	AD[6:0]							Data1
ActPos[15:8]								Data2
ActPos[7:0]								Data3
VddReset	StepLoss	ElDef	UV2	TSD	TW	Tinfo[1:0]		Data4

*) according to parity computation

ID[5:0] : Dynamically allocated direct identifier. There should be as many dedicated identifiers to the **GetActualPos** command as there are stepper motors connected to the LIN bus.

2) One preparing frame prior 4 data bytes in-frame response with **0x3D** indirect ID.

Preparing Frame (type #7 or #8)								
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0] = 0x00							Data1
1	AD[6:0]							Data2
In-frame Response (type #6)								
0	1	1	1	1	1	0	1	Identifier
ESW	AD[6:0]							Data1
ActPos[15:8]								Data2
ActPos[7:0]								Data3
VddReset	StepLoss	ElDef	UV2	TSD	TW	Tinfo[1:0]		Data4
0xFF								Data5
0xFF								Data6
0xFF								Data7
0xFF								Data8

*) according to parity computation

`GetFullStatus`

This command is provided to the circuit by the LIN Master to get a complete status of the circuit and of the Stepper-motor. Refer to § 0 and § 9.2.2.10 to see the meaning of the parameters sent to the LIN Master.

Note

A **`GetFullStatus`** command will attempt to reset flags <**`TW`**>, <**`TSD`**>, <**`UV2`**>, <**`ElDef`**>, <**`StepLoss`**>, <**`CPFail`**>, <**`OVC1`**>, <**`OVC2`**>, and <**`VddReset`**>.

`GetFullStatus` corresponds to two successive LIN in-frame responses with **0x3D** indirect ID.

Preparing Frame (type #7 or #8)								
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0] = 0x01							Data1
1	AD[6:0]							Data2
In-frame Response 1 (type #6)								
0	1	1	1	1	1	0	1	Identifier
1	AD[6:0]							Data1
Irun[3:0]				Ihold[3:0]				Data2
Vmax[3:0]				Vmin[3:0]				Data3
AccShape	StepMode[1:0]		Shaft	Acc[3:0]				Data4
VddReset	StepLoss	ElDef	UV2	TSD	TW	Tinfo[1:0]		Data5
Motion[2:0]			ESW	OVC1	OVC2	1	CPFail	Data6
Lin error status register (see 9.2.4.4)								Data7
0xFF								Data8
In-frame Response 2 (type #6)								
0	1	1	1	1	1	0	1	Identifier
1	AD[6:0]							Data1
ActPos[15:8]								Data2
ActPos[7:0]								Data3
TagPos[15:8]								Data4
TagPos[7:0]								Data5
SecPos[7:0]								Data6
1	1	1	1	1	SecPos[10:8]			Data7
0xFF								Data8

*) according to parity computation

Important

It is not mandatory for the LIN Master to initiate the second in-frame response if **`ActPos`**, **`TagPos`** and **`SecPos`** are not needed by the application.

`GetOTPparam`

This command is provided to the circuit by the LIN Master after a preparation frame (see § 9.2.4.6.3) was issued, to read the content of an OTP Memory segment which address was specified in the preparation frame.

`GetOTPparam` corresponds to a LIN in-frame response with **0x3D** indirect ID.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	Byte
Preparing Frame (type #7 or #8)								
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1			CMD[6:0] = 0x02					Data1
1				AD[6:0]				Data2
In-frame Response (type #6)								
0	1	1	1	1	1	0	1	Identifier
			OTP byte @0x00					Data1
			OTP byte @0x01					Data2
ADM	HW2	HW1	HW0	PA3	PA2	PA1	PA0	Data3
			OTP byte @0x03					Data4
			OTP byte @0x04					Data5
			OTP byte @0x05					Data6
			OTP byte @0x06					Data7
			OTP byte @0x07					Data8

*) according to parity computation

HW[2:0] : Although not stored in the OTP memory, the hardwired address is returned by **`GetOTPparam`** as if stored at address 0x02 of the OTP memory.

`GetStatus`

This command is provided to the circuit by the LIN Master to get a quick status (compared to that of **`GetFullStatus`** command) of the circuit and of the stepper motor. Refer to § 9.2.2.10 to see the meaning of the parameters sent to the LIN Master.

Note

A GetStatus command will attempt to reset flags <**TW**>, <**TSD**>, <**UV2**>, <**ElDef**>, <**StepLoss**>, and <**VddReset**>.

`GetStatus` corresponds to a 2 data bytes LIN in-frame response with a direct ID (type #5).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	Byte
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
ESW					AD[6:0]			Data1
VddReset	StepLoss	ElDef	UV2	TSD	TW	Tinfo[1:0]		Data2

*) according to parity computation

ID[5:0] : Dynamically allocated identifier. There should be as many dedicated identifiers to the **`GetStatus`** command as there are stepper motors connected to the LIN bus.

GotoSecurePosition

This command is provided by the LIN Master to one or all the Stepper-motors to move to the secure position **SecPos[10:0].** It can also be internally triggered if the LIN bus communication is lost, or after an initialization phase or prior to going into sleep mode. See the priority encoder description for more details (§ 9.2.1.2). The priority encoder table also acknowledges the cases where a **GotoSecurePosition** command will be ignored.

GotoSecurePosition corresponds to the following LIN writing frame (type #1).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0] = 0x04							Data1
Broad	AD[6:0]							Data2

*) according to parity computation

If **Broad** = '0' all the Stepper motors connected to the LIN bus will reach their secure position.

HardStop

This command will be internally triggered when an electrical problem is detected in one or both coils, leading to Shutdown mode. If this occurs while the motor is moving, the <**StepLoss**> flag is raised to allow warning of the LIN Master at the next **GetStatus** command that steps may have been lost. Once the motor is stopped, **ActPos** register is copied into **TagPos** register to ensure keeping the stop position.

A **hardstop** command can also be issued by the LIN Master for some safety reasons. It corresponds then to the following 2 Data bytes LIN writing frame (type #1).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0] = 0x05							Data1
Broad	AD[6:0]							Data2

*) according to parity computation

If **Broad** = '0' all the Stepper motors connected to the LIN bus will stop.

ResetPosition

This command is provided to the circuit by the LIN Master to reset **ActPos** and **TagPos** registers to zero. This can be helpful to prepare for instance a relative positioning.

ResetPosition corresponds to the following LIN writing frames (type #1).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0] = 0x06							Data1
Broad	AD[6:0]							Data2

*) according to parity computation

If **Broad** = '0' all the circuits connected to the LIN bus will reset their **ActPos** and **TagPos** registers.

ResetToDefault

This command is provided to the circuit by the LIN Master in order to reset the whole Slave node into the initial state. **ResetToDefault** will for instance overload the RAM with the Reset state of the Registers parameters (see § 0). This is another way for the LIN Master to initialize a slave node in case of emergency, or simply to refresh the RAM content.

Note
ActPos and **TagPos** are not modified by a **ResetToDefault** command.

Important:
Care should be taken not to send a **ResetToDefault** command while a motion is ongoing, since this could modify the motion parameters in a way forbidden by the position controller.

ResetToDefault corresponds to the following LIN writing frames (type #1).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0] = 0x07							Data1
Broad	AD[6:0]							Data2

*) according to parity computation

If **Broad** = '0' all the circuits connected to the LIN bus will reset to default.

RunInit

This command is provided to the circuit by the LIN Master in order to initialize positioning of the motor by seeking the zero (or reference) position. See § 9.2.1.2 for a detailed description or the initialization phase.

Note1:
This sequence cannot be interrupted by another positioning command.

Important:
If for some reason **ActPos** equals **Pos1[15:0]** at the moment the **RunInit** command is issued, the circuit will enter in deadlock state. Therefore, the application should check the actual position by a **GetPosition** or a **GetFullStatus** command prior to an initialization. Another solution may consist in programming a value out of the stepper motor range for **Pos1[15:0]**. For the same reason **Pos2[15:0]** should not be equal to **Pos1[15:0]**.

RunInit corresponds to the following LIN writing frame with 0x3C identifier (type #4).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
0	0	1	1	1	1	0	0	Identifier
AppCMD = 0x80								Data1
1	CMD[6:0] = 0x08							Data2
Broad	AD[6:0]							Data3
Vmax[3:0]				Vmin[3:0]				Data4
Pos1[15:8]								Data5
Pos1[7:0]								Data6
Pos2[15:8]								Data7
Pos2[7:0]								Data8

If **Broad** = '0' all the circuits connected to the LIN bus will run the init sequence.

Vmax[3:0] : Max velocity for first motion of the init run
Vmin[3:0] : Min velocity for first motion of the init run and velocity for the second motion of the init run
Pos1[15:0] : First position to be reached during the init run
Pos2[15:0] : Second position to be reached during the init run

SetMotorParam()

This command is provided to the circuit by the LIN Master to set the values for the Stepper motor parameters (listed below) in RAM. Refer to § 0 to see the meaning of the parameters sent by the LIN Master.

Important: If a **SetMotorParam** occurs while a motion is ongoing, it will modify at once the motion parameters (see § 9.2.1.1). Therefore the application should not change other parameters than **Vmax** and **Vmin** while a motion is running, otherwise correct positioning cannot be guaranteed.

SetMotorParam corresponds to the following LIN writing frame with 0x3C identifier (type #4).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
0	0	1	1	1	1	0	0	Identifier
AppCMD = 0x80								Data1
1	CMD[6:0] = 0x09							Data2
Broad	AD[6:0]							Data3
Irun[3:0]				Ihold[3:0]				Data4
Vmax[3:0]				Vmin[3:0]				Data5
SecPos[10:8]			Shaft	Acc[3:0]				Data6
SecPos[7:0]								Data7
1	1	1	AccShape	StepMode[1:0]		1	1	Data8

If **Broad** = '0' all the circuits connected to the LIN bus will set the parameters in their RAMs as requested.

SetOTPparam()

This command is provided to the circuit by the LIN Master to program the content **D[7:0]** of the OTP memory byte **OTPA[2:0]**, and to zap it.

Important:
This command must be sent under a specific Vbb voltage value. See parameter VbbOTP in § 5. This is a mandatory condition to ensure reliable zapping.

SetMotorParam corresponds to a 0x3C LIN writing frames (type #4).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
0	0	1	1	1	1	0	0	Identifier
AppCMD = 0x80								Data1
1	CMD[6:0] = 0x10							Data2
Broad	AD[6:0]							Data3
1	1	1	1	1	OTPA[2:0]			Data4
D[7:0]								Data5
0xFF								Data6
0xFF								Data7
0xFF								Data8

If **Broad** = '0' all the circuits connected to the LIN bus will set the parameters in their OTP memories as requested.

`SetPosition()`

This command is provided to the circuit by the LIN Master to drive one or two motors to a given absolute position. See § 8.5 for more details.

The priority encoder table (§ 9.2.2.12) acknowledges the cases where a **`SetPosition`** command will be ignored.

`SetPosition` corresponds to the following LIN write frames.

1) 2 Data bytes frame with a direct ID (type #3)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
Pos[15:8]								Data1
Pos[7:0]								Data2

*) according to parity computation

`ID[5:0]` : Dynamically allocated direct identifier. There should be as many dedicated identifiers to this SetPosition command as there are stepper motors connected to the LIN bus.

2) 4 Data bytes frame with a general purpose identifier (type #1)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	1	0	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0] = 0x0B							Data1
Broad	AD[6:0]							Data2
Pos[15:8]								Data3
Pos[7:0]								Data4

*) according to parity computation

If **`Broad`** = '0' all the Stepper motors connected to the LIN will must go to **`Pos[15:0]`**.

3) Two motors positioning frame with **0x3C** identifier (type #4)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
0	0	1	1	1	1	0	0	Identifier
AppCMD = 0x80								Data1
1	CMD[6:0] = 0x0B							Data2
1	AD1[6:0]							Data3
Pos1[15:8]								Data4
Pos1[7:0]								Data5
1	AD2[6:0]							Data6
Pos2[15:8]								Data7
Pos2[7:0]								Data8

`ADn[6:0]` : Motor #n physical address (n ∈[1,2]).

`Posn[15:0]` : Signed 16-bit position set-point for Motor #n.

`SetPositionShort()`

This command is provided to the circuit by the LIN Master to drive one, two or four motors to a given absolute position. It applies only for half stepping mode (**`StepMode[1:0]`** = "00") and is ignored when in other stepping modes. See § 8.5 for more details.

The physical address is coded on 4 bits, hence **`SetPositionShort`** can only be used with a network implementing a maximum of 16 slave nodes. These 4 bits are normally corresponding to the bits **`PA[3:0]`** in OTP memory (address **`0x00`**), while bits **`AD[6:4]`** must be at '1'. Two different cases must in fact be considered, depending on the programmed value of bit ADM in the OTP memory (see § 9.2.4.5):

ADM	AD[3]	pin HW0	pin HW1	pin HW2	bit PA0 in OTP memory
0	X	Tied to Vdd		Tied to Vbb	AD[0]
1	0	Tied to Vdd		Tied to Gnd	1
1	1	Tied to Vdd		Tied to Vbat	1

The priority encoder table (§ 9.2.2.12) acknowledges the cases where a **`SetPositionShort`** command will be ignored.

`SetPositionShort` corresponds to the following LIN writing frames.

1) 2 Data bytes frame for 1 motor, with specific identifier (type #2)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
	Pos[10:8]		Broad	AD[3:0]				Data1
			Pos[7:0]					Data2

*) according to parity computation

`SetPositionShort` command.

If **`Broad`** = '0' all the stepper motors connected to the LIN bus will go to **`Pos[10:0]`**.

`ID[5:0]` : Dynamically allocated identifier to 2 Data bytes

2) 4 Data bytes frame for two motors, with specific identifier (type # 2)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	1	0	ID3	ID2	ID1	ID0	Identifier
	Pos1[10:8]	1		AD1[3:0]				Data1
			Pos1[7:0]					Data2
	Pos2[10:8]	1		AD2[3:0]				Data3
			Pos2[7:0]					Data4

*) according to parity computation

`ID[5:0]` : Dynamically allocated identifier to 4 Data bytes **`SetPositionShort`** command.

`ADn[3:0]` : Motor #n physical address least significant bits (n ∈ [1,2]).

`Posn[10:0]` : Signed 11-bit position set point for Motor #n

3) 8 Data bytes frame for 4 motors, with specific identifier (type #2)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	1	1	ID3	ID2	ID1	ID0	Identifier
Pos1[10:8]			1	AD1[3:0]				Data1
			Pos1[7:0]					Data2
Pos2[10:8]			1	AD2[3:0]				Data3
			Pos2[7:0]					Data4
Pos3[10:8]			1	AD3[3:0]				Data5
			Pos3[7:0]					Data6
Pos4[10:8]			1	AD4[3:0]				Data7
			Pos4[7:0]					Data8

*) according to parity computation

`ID[5:0]` : Dynamically allocated identifier to 8 Data bytes **`SetPositionShort`** command.

`ADn[3:0]` : Motor #n physical address least significant bits (n ∈ [1,4]).

`Posn[10:0]` : Signed 11-bit position set point for Motor #n (see § 0).

Sleep

This command is provided to the circuit by the LIN Master to put all the Slave nodes connected to the LIN bus into sleep mode. If this command occurs during a motion of the motor, **TagPos** is reprogrammed to **SecPos** (provided **SecPos** is different from "100 0000 0000"), or a **SoftStop** is executed before going to sleep mode. See LIN 1.2 specification and § 9.2.2.7.

The corresponding LIN frame is a Master Request Command Frame (identifier 0x3C) with Data byte 1 containing 0x00 while the followings contain 0xFF.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
0	0	1	1	1	1	0	0	Identifier
0x00								Data1
0xFF								Data2

SoftStop

If a **SoftStop** command occurs during a motion of the stepper motor, it provokes an immediate deceleration to **VMIN** (see § 8.3) followed by a stop, regardless of the position reached. Once the motor is stopped, **TagPos** register is overwritten with value in **ActPos** register to ensure keeping the stop position.

Note

A **SoftStop** command occurring during a **RunInit** sequence is not taken into account.

Command **SoftStop** occurs in the following cases:
- The chip temperature rises above the Thermal shutdown threshold (see § 5 and § 9.2.2.5);
- The LIN Master requests a **SoftStop**. Hence **SoftStop** will correspond to the following 2 Data bytes LIN writing frame (type #1).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0] = 0x0F							Data1
Broad	AD[6:0]							Data2

*) according to parity computation

If **Broad** = '0' all the stepper motors connected to the LIN bus will stop with deceleration.

9.2.2.12 Priority Encoder

The table below describes the state management performed by the Main control block.

State → Command ↓	**Stopped** motor stopped, Ihold in coils	**GotoPos** motor motion ongoing	**RunInit** no influence on RAM and `TagPos`	**SoftStop** motor decelerating	**HardStop** motor forced to stop	**ShutDown** motor stopped, H-bridges in Hi-Z	**Sleep** no power (note 1)
`GetActualPos`	LIN in-frame response	LIN in-frame response	LIN in-frame response	LIN in-frame response	LIN in-frame response	LIN in-frame response	
`GetOTPparam`	OTP refresh; LIN in-frame response	OTP refresh; LIN in-frame response	OTP refresh; LIN in-frame response	OTP refresh; LIN in-frame response	OTP refresh; LIN in-frame response	OTP refresh; LIN in-frame response	
`GetFullStatus` or `GetStatus` [attempt to clear `<TSD>` and `<HS>` flags]	LIN in-frame response	LIN in-frame response	LIN in-frame response	LIN in-frame response	LIN in-frame response	LIN in-frame response; if (`<TSD>` or `<HS>`) = '0' then → **Stopped**	
`ResetToDefault` [`ActPos` and `TagPos` are not altered]	OTP refresh; OTP to RAM; `AccShape` reset	OTP refresh; OTP to RAM; `AccShape` reset	OTP refresh; OTP to RAM; `AccShape` reset (note 3)	OTP refresh; OTP to RAM; `AccShape` reset	OTP refresh; OTP to RAM; `AccShape` reset	OTP refresh; OTP to RAM; `AccShape` reset	
`SetMotorParam` [Master takes care about proper update]	RAM update	RAM update	RAM update	RAM update	RAM update	RAM update	
`ResetPosition`	`TagPos` and `ActPos` reset					`TagPos` and `ActPos` reset	
`SetPosition`	`TagPos` updated; → **GotoPos**	`TagPos` updated	`TagPos` updated				
`SetPositionShort` [half-step mode only)]	`TagPos` updated; → **GotoPos**	`TagPos` updated	`TagPos` updated				
`GotoSecPosition`	If `<SecEn>` = '1' then `TagPos` = `SecPos`; → **GotoPos**	If `<SecEn>` = '1' then `TagPos` = `SecPos`	If `<SecEn>` = '1' then `TagPos` = `SecPos`				
`RunInit`	→ **RunInit**						
`HardStop`		→ **HardStop**; `<StepLoss>` = '1'	→ **HardStop**; `<StepLoss>` = '1'	→ **HardStop**; `<StepLoss>` = '1'			
`SoftStop`		→ **SoftStop**					
`Sleep` or LIN timeout [⇒ `<Sleep>` = '1', reset by any LIN command received later]	See note 9	If `<SecEn>` = '1' then `TagPos` = `SecPos` else → **SoftStop**	If `<SecEn>` = '1' then `TagPos` = `SecPos`; will be evaluated after RunInit	No action; `<Sleep>` flag will be evaluated when motor stops	No action; `<Sleep>` flag will be evaluated when motor stops	→ **Sleep**	
`HardStop` [↔ (`<CPFail>` or `<UV2>` or `<ElDef>`) = '1' ⇒ `<HS>` = '1']	→ **Shutdown**	→ **HardStop**	→ **HardStop**	→ **HardStop**			
Thermal shutdown [`<TSD>` = '1']	→ **Shutdown**	→ **SoftStop**	→ **SoftStop**				
Motion finished	n.a.	→ **Stopped**	→ **Stopped**	→ **Stopped**; `TagPos` =`ActPos`	→ **Stopped**; `TagPos` =`ActPos`	n.a.	n.a.

With the following color code:

- (light grey) Command ignored
- (medium grey) Transition to another state
- (dark grey) Master is responsible for proper update (see note 7)

Notes:

1 Leaving Sleep state is equivalent to Power-on-reset.

2 After Power-on reset, the Shutdown state is entered. The Shutdown state can only be left after **GetFullStatus** command (so that the Master could read the <**VddReset**> flag).

3 A RunInit sequence runs with a separate set of RAM registers. The parameters that are not specified in a RunInit command are loaded with the values stored in RAM at the moment the RunInit sequence starts. **AccShape** is forced to ‘1’ during second motion even if a **ResetToDefault** command is issued during a RunInit sequence, in which case **AccShape** at ‘0’ will be taken into account after the RunInit sequence. A **GetFullStatus** command will return the default parameters for **Vmax** and **Vmin** stored in RAM.

4 The <**Sleep**> flag is set to ‘1’ when a LIN timeout or a Sleep command occurs. It is reset by the next LIN command (<**Sleep**> is cancelled if not activated yet).

5 Shutdown state can be left only when <**TSD**> and <**HS**> flags are reset.

6 Flags can be reset only after the master could read them via a **GetStatus** or **GetFullStatus** command, and provided the physical conditions allow for it (normal temperature, correct battery voltage and no electrical or charge pump defect).

7 A **SetMotorParam** command sent while a motion is ongoing (state GotoPos) should not attempt to modify Acc and Vmin values. This can be done during a RunInit sequence since this motion uses its own parameters, the new parameters will be taken into account at the next **SetPosition** or **SetPositionShort** command.

8 Some transitions like GotoPos ➔ Sleep are actually done via several states: GotoPos ➔ SoftStop ➔ Stopped ➔ Sleep (see diagram below).

9 Two transitions are possible from state Stopped when <**Sleep**> = ‘1’:

1) Transition to state Sleep if (<**SecEn**> = ‘0’) or ((<**SecEn**> = ‘1’) and (**ActPos** = **SecPos**)) or <**Stop**> = ‘1’.

2) Otherwise transition to state **GotoPos**, with **TagPos** = **SecPos.**

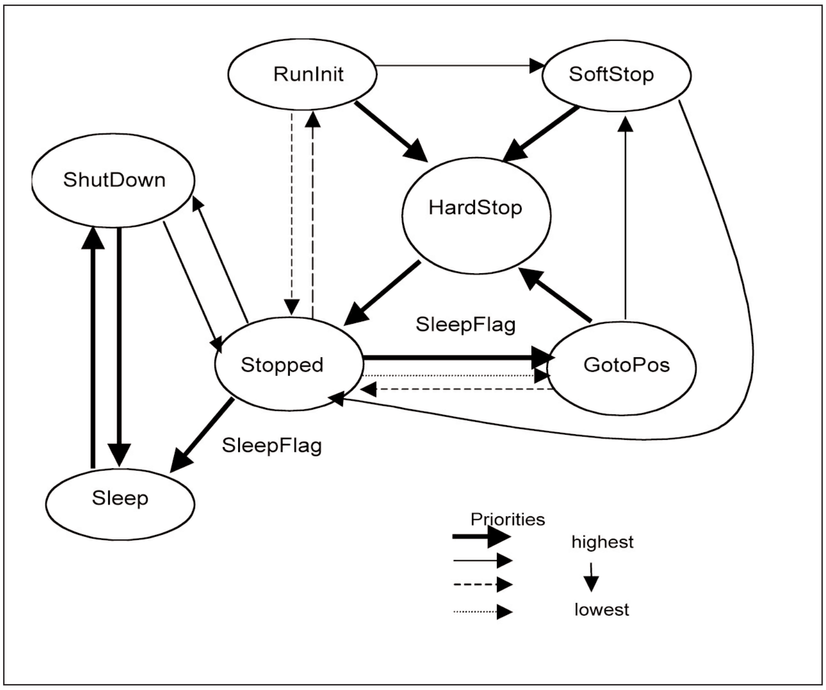
10 <**SecEn**> = ‘1’ when register SecPos is loaded with a value different from the most negative value (i.e. different from 0x400 = “100 0000 0000”).

11 <**Stop**> flag allows to distinguish whether state Stopped was entered after HardStop/SoftStop or not. <**Stop**> is set to ‘1’ when leaving state **HardStop** or **SoftStop** and is reset during first clock edge occurring in state Stopped.

12 Command for dynamic assignment of IDs is decoded in all states except Sleep and has not effect on the current state.

13 While in state Stopped, if **ActPos** ≠ **TagPos** there is a transition to state **GotoPos**. This transition has the lowest priority, meaning that <**Sleep**>, <**Stop**>, <**TSD**>, etc. are first evaluated for possible transitions.

14 If <**StepLoss**> is active, then SetPosition, SetPositionShort and **GotoSecurePosition** commands are ignored (they will not modify **TagPos** register whatever the state), and motion to secure position is forbidden after a **Sleep** command or a LIN timeout (the circuit will go into Sleep state immediately, without positioning to secure position). Other command like **RunInit** or **ResetPosition** will be executed if allowed by current state. <**StepLoss**> can only be cleared by a **GetStatus** or **GetFullStatus** command.



9.2.2.13 Application Parameters Stored in OTP Memory

Except for the physical address **AD[3:0]** and for bit **ADM**, these parameters, although programmed in a non-volatile memory can still be overridden in RAM by a LIN writing operation.

AD[6:0] Physical address of the stepper motor. Up to 128 stepper motors can theoretically be connected to the same LIN bus.

ADM Addressing mode bit (see § 9.2.4.5)

Irun[3:0] Peak current value to be fed to each coil of the stepper-motor. The table to the right provides the 16 possible values for **IRUN**.

Irun				Peak Current (mA)
0	0	0	0	59
0	0	0	1	71
0	0	1	0	84
0	0	1	1	100
0	1	0	0	119
0	1	0	1	141
0	1	1	0	168
0	1	1	1	200
1	0	0	0	238
1	0	0	1	283
1	0	1	0	336
1	0	1	1	400
1	1	0	0	476
1	1	0	1	566
1	1	1	0	673
1	1	1	1	800

Ihold[3:0] Hold current for each coil of the stepper-motor. The table to the right provides the 16 possible values for **IHOLD**.

Ihold				Hold Current (mA)
0	0	0	0	59
0	0	0	1	71
0	0	1	0	84
0	0	1	1	100
0	1	0	0	119
0	1	0	1	141
0	1	1	0	168
0	1	1	1	200
1	0	0	0	238
1	0	0	1	283
1	0	1	0	336
1	0	1	1	400
1	1	0	0	476
1	1	0	1	566
1	1	1	0	673
1	1	1	1	800

StepMode Indicator of stepping mode to be used.

StepMode		Step Mode
0	0	Half stepping
0	1	1/4 micro step
1	0	1/8 micro step
1	1	1/16 micro step

Shaft Indicator of Reference Position. If **Shaft** = '0', the reference position is the maximum inner position, whereas if **Shaft** = '1', the reference position is the maximum outer position.

SecPos[10:0] Secure Position of the stepper motor. This is the position to which the motor is driven in case of a LIN communication loss or when the LIN error counter overflows. If **SecPos[10:0]** = "100 0000 0000", this means that Secure Position is disabled, e.g. the stepper motor will be kept in the position occupied at the moment these events occur.

The Secure Position is coded on 11 bits only, providing actually the most significant bits of the position, the non coded least significant bits being set to '0'.

Vmax[3:0]
Vmin[3:0]
Acc[3:0] Maximum velocity, minimum velocity and acceleration of the stepper motor are programmed by coding the respective Vmax, Vmin and Acc parameters index as defined in § 8 Positioning data.

Code				Parameter Index
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

9.2.2.14 OTP Memory Structure

The table below shows how the parameters to be stored in the OTP memory are located.

OTP address	7	6	5	4	3	2	1	0
0x00	**OSC3**	**OSC2**	**OSC1**	**OSC0**	**IREF3**	**IREF2**	**IREF1**	**IREF0**
0x01	**EnableLIN**	**TSD2**	**TSD1**	**TSD0**	**BG3**	**BG2**	**BG1**	**BG0**
0x02	**ADM**				**PA3**	**PA2**	**PA1**	**PA0**
0x03	**Irun3**	**Irun2**	**Irun1**	**Irun0**	**Ihold3**	**Ihold2**	**Ihold1**	**Ihold0**
0x04	**Vmax3**	**Vmax2**	**Vmax1**	**Vmax0**	**Vmin3**	**Vmin2**	**Vmin1**	**Vmin0**
0x05	**SecPos10**	**SecPos9**	**SecPos8**	**Shaft**	**Acc3**	**Acc2**	**Acc1**	**Acc0**
0x06	**SecPos7**	**SecPos6**	**SecPos5**	**SecPos4**	**SecPos3**	**SecPos2**	**SecPos1**	**SecPos0**
0x07					**StepMode1**	**StepMode0**	**LOCKBT**	**LOCKBG**

Parameters stored at address 0x00 and 0x01 and bit **LOCKBT** are already programmed in the OTP memory at circuit delivery, they correspond to the calibration of the circuit and are just documented here as an indication.

Each OPT bit is at '0' when not zapped. Zapping a bit will set it to '1'. Thus only bits having to be at '1' must be zapped. Zapping of a bit already at '1' is disabled.

Each OTP byte will be programmed separately (see command **SetOTPparam**).

Once OTP programming is completed, bit **LOCKBG** can be zapped, to disable future zapping, otherwise any OTP bit at '0' could still be zapped by using a **SetOTPparam** command.

Lock Bit	Protected Byte
LOCKBT (zapped before delivery)	0x00 to 0x01
LOCKBG	0x00 to 0x07

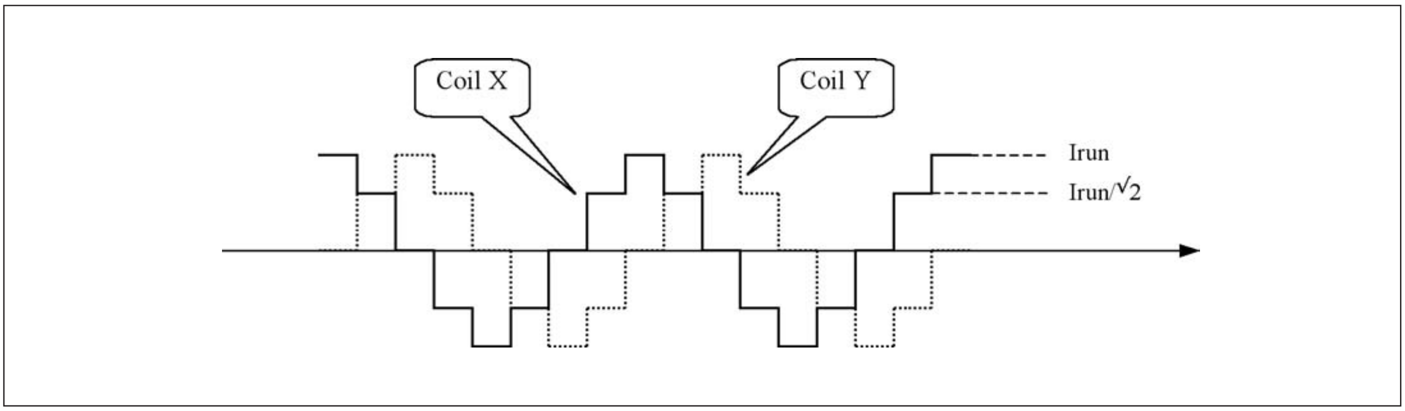
The command used to load the application parameters via the LIN bus in the RAM prior to an OTP Memory programming is **SetMotorParam**. This allows for a functional verification before using a **SetOTPparam** command to program and zap separately one OTP memory byte. A **GetOTPparam** command issued after each **SetOTPparam** command allows to verify the correct byte zapping.

Note
Zapped bits will really be "active" after a **GetOTPparam** or a **ResetToDefault** command or after a power-up.

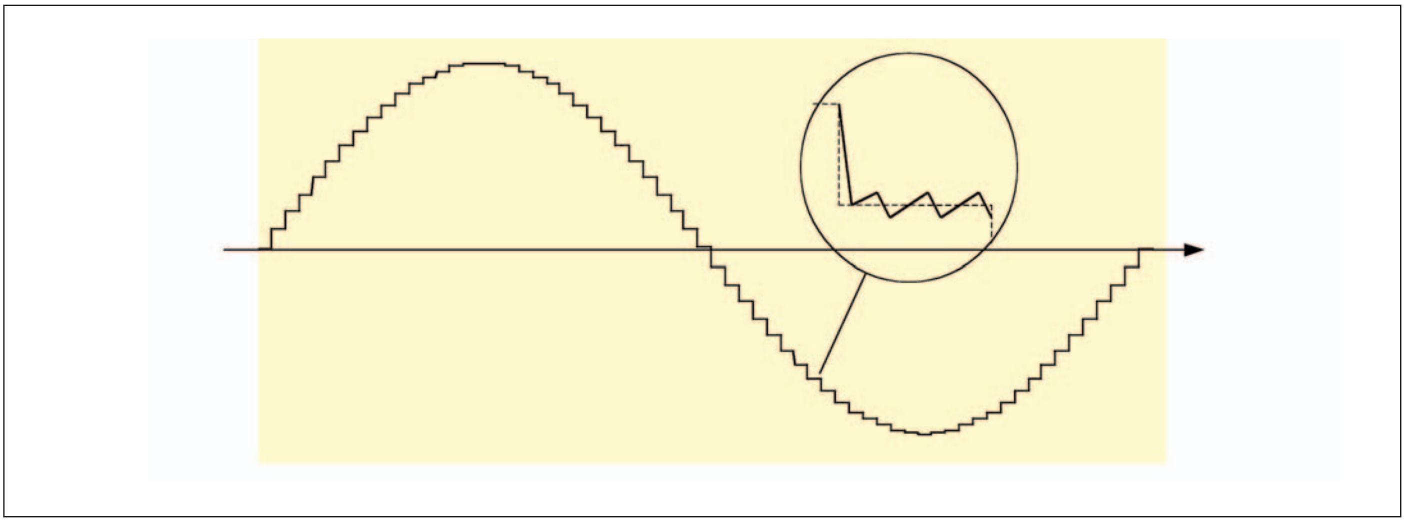
9.2.3 Motordriver

9.2.3.1 Current Waveforms in the Coils

The figure below illustrates the current fed to the motor coils by the motordriver in half-step mode.



Whereas the figure below shows the current fed to one coil in 1/16th microstepping (1 electrical period).



9.2.3.2 PWM Regulation

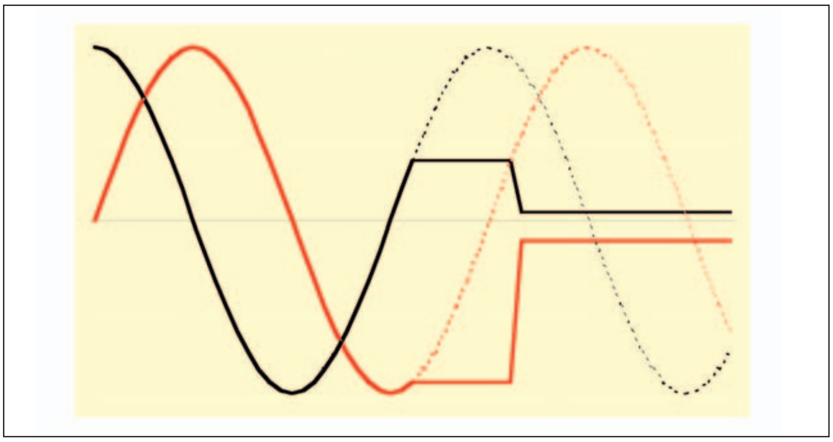
In order to force a given current (determined by Irun or Ihold and the current position of the rotor) through the motor coil while ensuring high energy transfer efficiency, a regulation based on PWM principle is used. The regulation loop performs a comparison of the sensed output current to an internal reference, and features a digital regulation generating the PWM signal that drives the output switches. The zoom over one micro-step in the figure above shows how the PWM circuit performs this regulation.

9.2.3.3 Motor Starting Phase

At motion start, the currents in the coils are directly switched from **`Ihold`** to **`Irun`** with a new sine/cos ratio corresponding to the first half (or micro) step of the motion.

9.2.3.4 Motor Stopping Phase

At the end of the deceleration phase, the currents are maintained in the coils at their actual DC level (hence keeping the sine/cos ratio between coils) during **1/4**th of an electrical period at minimum velocity (thus 2 half-steps). The currents are then set to the hold values, respectively **Ihold x sin(TagPos)** and **Ihold x cos(TagPos)** as illustrated below. A new positioning order can then be executed.



9.2.3.5 Charge Pump Monitoring

If the charge pump voltage is not sufficient for driving the high side transistors (due to a failure), an internal **`HardStop`** command is issued. This is acknowledged to the master by raising flag <**`CPFail`**> (available with command **`GetFullStatus`**).

In case this failure occurs while a motion is ongoing, the flag <**`StepLoss`**> is also raised.

9.2.3.6 Electrical Defect on Coils, Detection and Confirmation

The principle relies on the detection of a voltage drop on at least one transistor of the H-bridge. Then the decision is taken to open the transistors of the defective bridge.
This allow to detect the following short circuits:

- External coil short circuit.
- Short between one terminal of the coil and Vbat or Gnd.
- One cannot detect internal short in the motor.

Open circuits are detected by 100% PWM duty cycle value during a long time.

Pins	Fault Mode
Yi or Xi	Short circuit to GND
Yi or Xi	Short circuit to Vbat
Yi or Xi	Open
Y1 and Y2	Short circuited
X1 and X2	Short circuited
Xi and Yi	Short circuited

9.2.4 LIN Controller

9.2.4.1 General Description

The LIN (Local Interconnect Network) is a serial communications protocol that efficiently supports the control of mechatronic nodes in distributed automotive applications.

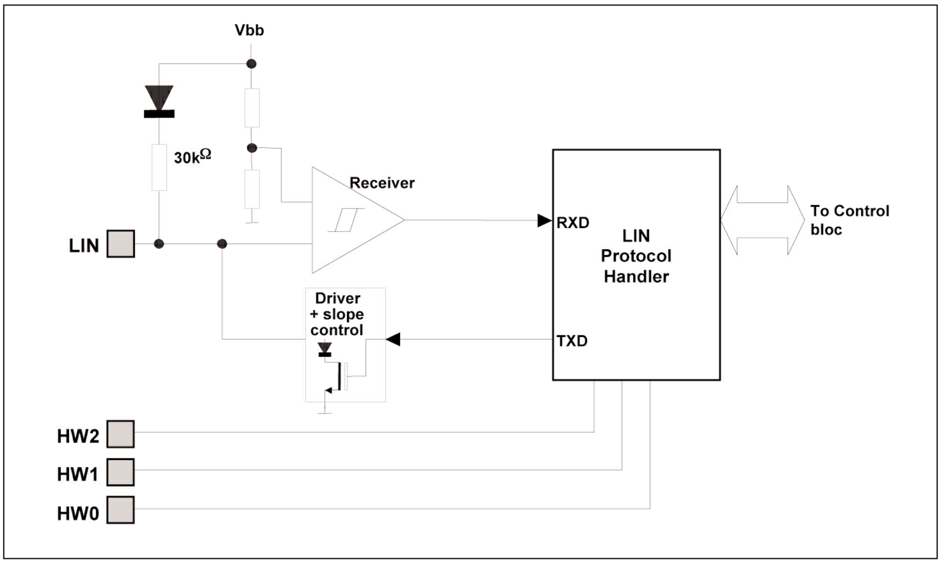
The interface implemented in the AMIS-30621 is compliant with the LIN rev. 1.2 specifications. It features a **slave node**, thus allowing for:

- single-master / multiple-slave communication
- self synchronization without quartz or ceramics resonator in the slave nodes
- guaranteed latency times for signal transmission
- single-wire communication
- transmission speed of 19.2 kbit/s
- selectable length of Message Frame: 2, 4, and 8 bytes
- configuration flexibility
- data checksum security and error detection;
- detection of defective nodes in the network.

It includes the analog physical layer and the digital protocol handler.

The analog circuitry implements a low side driver with a pull-up resistor as a transmitter, and a resistive divider with a comparator as a receiver.

The specification of the line driver/receiver follows the ISO 9141 standard with some enhancements regarding the EMI behavior.



9.4.2.4 Slave Operational Range for Proper Self Synchronization

The LIN interface will synchronize properly in the following conditions:

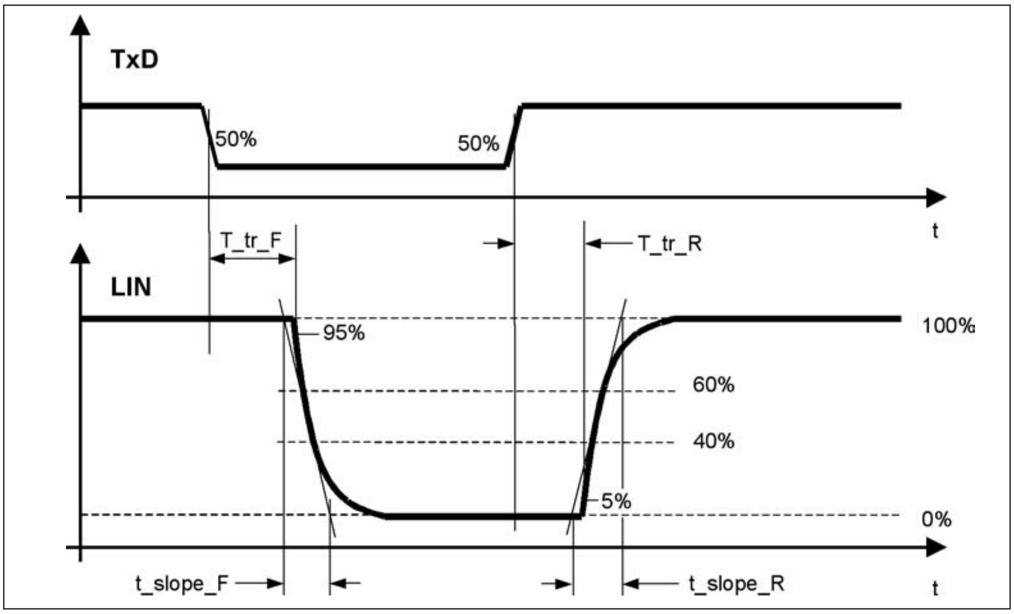
- Vbb ≥ 8 V
- Ground shift between Master node and Slave node < ±1V

It is highly recommended to use the same type of reverse battery voltage protection diode for the Master and the Slave nodes.

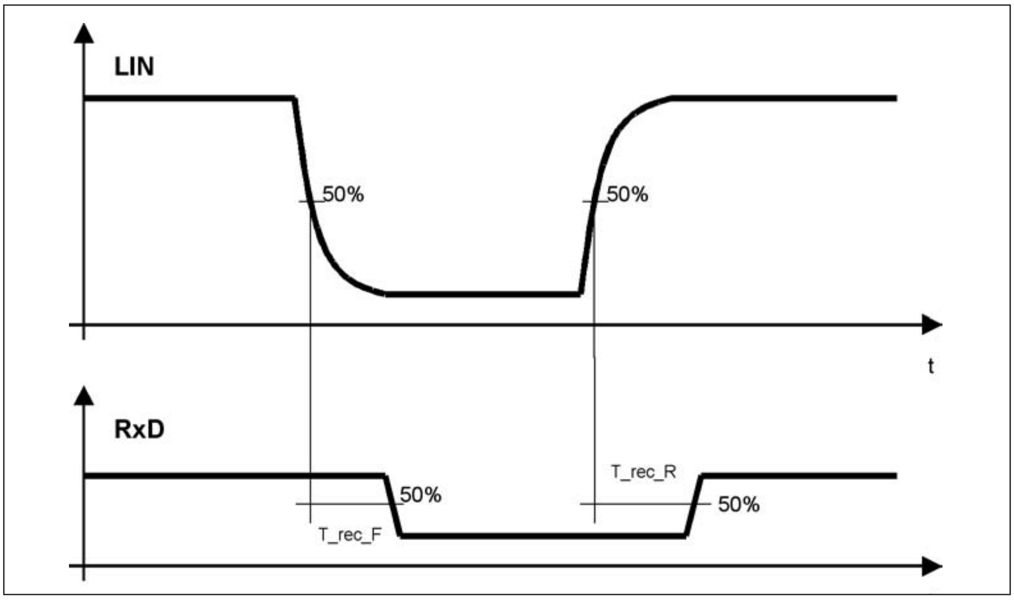
9.2.4.3 Functional Description

9.2.4.3.1 Analog Part

The transmitter is a low-side driver with a pull-up resistor and slope control. The figure below shows the characteristics of the transmitted signal, including the delay between internal TxD signal and LIN signal. See § 6 for timing values.



The receiver mainly consists of a comparator which threshold is equal to Vbb/2. The figure below shows the delay between the received signal and the internal RxD signal. See § 6 for timing values.



9.2.4.3.2 Protocol Handler

This block implements:
- bit synchronization
- bit timing
- the MAC layer
- the LLC layer
- the supervisor

9.2.4.3.3 Electro Magnetic Compatibility

EMC behavior fulfills requirements defined by LIN specification, rev. 1.2.

9.2.4.4 Error Status Register

The LIN interface implements a register containing an error status of the LIN communication. This register is as follows:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
not used	not used	not used	not used	Time out error	Data error flag	Header error flag	Bit error flag

With:
- Data error flag = Checksum error + StopBit error + Length error
- Header error flag = Parity + SynchField error

A **`GetFullStatus`** frame will reset the error status register.

9.2.4.5 Physical Address of the Circuit

The circuit must be provided with a physical address in order to discriminate this circuit from other ones on the LIN bus. This address is coded on 7 bits, yielding the theoretical possibility of 128 different circuits on the same bus. It is a combination of 4 OTP memory bits (see § 9.2.2.14) and of the 3 hardwired address bits (pins HW[2:0]).

	AD6	**AD5**	**AD4**	**AD3**	**AD2**	**AD1**	**AD0**	Physical address
ADM	↑	↑	↑	PA3	PA2	PA1	PA0	OTP memory
	HW0	HW1	HW2					Hardwired bits

The OTP memory contains also an extra bit, ADM, which allows for the following combinations:

ADM	AD6	AD5	AD4	AD3	AD2	AD1	AD0
0	HW0	HW1	HW2	PA3	PA2	PA1	PA0
1	PA0	HW0	HW1	HW2	PA3	PA2	PA1

Note
Pins HW0 and HW1 are 5V digital inputs, whereas pin HW2 is compliant with a 12V level, e.g. it can be connected to Vbat or Gnd via a terminal of the PCB. To provide cleaning current for this terminal, the system used for pin SWI is also implemented for pin HW2 (see § 9.2.1.3).

9.2.4.6 LIN Frames

The LIN frames can be divided in writing and reading frames. A frame is composed of an 8-bit Identifier followed by 2, 4 or 8 Data-bytes. Writing frames will be used to:
- Program the OTP Memory;
- Configure the component with the stepper motor parameters (current, speed, stepping mode, etc.);
- Provide set-point position for the stepper motor.

Whereas Reading frames will be used to:
- Get the actual position of the stepper motor;
- Get status information such as error flags;
- Verify the right programming and configuration of the component.

9.2.4.6.1 Writing Frames

A writing frame is sent by the LIN Master to send commands and/or information to the Slave nodes.

According to the LIN specification, identifiers are to be used to determine a specific action. If a physical addressing is needed, then some bits of the Data field can be dedicated to this, as illustrated in the example below.

Identifier byte								Data byte 1								Data byte 2							
ID0	ID1	ID2	ID3	ID4	ID5	ID6	ID7																
								phys. address				command parameters (e.g. position)											

Another possibility is to determine the specific action within the Data field in order to use less identifiers. One can for example use the reserved identifier **0x3C** and take advantage of the 8-byte Data field to provide a physical address, a command and the needed parameters for the action, as illustrated in the example below.

Note
Bit 7 of byte Data1 must be at '1' since the LIN specification requires that contents from 0x00 to 0x7F must be reserved for broadcast messages (0x00 being for the "Sleep" message).

ID	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Data8
0x3C 00	1							
	AppCmd	command	physical address	parameters				

The writing frames used with the AMIS-30621 are the followings:

- **Type #1**: General purpose 2 or 4 Data bytes writing frame with a dynamically assigned identifier

This type is dedicated to short writing actions when the bus load can be an issue. They are used to provide direct command to one (**Broad** = '1') or all the slave nodes (Broad = '0'). If **Broad** = '1', the physical address of the slave node is provided by the 7 remaining bits of DATA2. DATA1 will contain the command code (see § 9.2.4.7), while, if present, DATA3 to DATA4 will contain the command parameters, as shown below.

ID								Data1	Data2		Data3…
ID0	ID1	ID2	ID3	ID4	ID5	ID6	ID7	command	Physical address	Broad	parameters…

- **Type #2**: 2, 4 or 8 Data bytes writing frame with an identifier dynamically assigned to an application command, regardless of the physical address of the circuit.
- **Type #3**: 2 Data bytes writing frame with an identifier dynamically assigned to a particular slave node together with an application command. This type of frame requires that there are as many dynamically assigned identifiers as there are AMIS-30621 circuits using this command connected to the LIN bus.
- **Type #4**: 8 Data bytes writing frame with **0x3C** identifier.

The structure is similar to Type #1 but uses the reserved ID 0x3C. Type #1 has the advantage to be shorter than Type #4.

ID	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Data8
0x3C 00	0x80	CMD[6:0] 1	AD[6:0] B					
	AppCmd	command	physical address	parameters				

Broad bit

9.2.4.6.2 Reading frames

A reading frame uses an in-frame response mechanism. That is: the master initiates the frame (synchronization field + identifier field), and one slave sends back the data field together with the check field. Hence, two types of identifiers can be used for a reading frame:

- Direct ID, which points at a particular slave node, indicating at the same time which kind of information is awaited from this slave node, thus triggering a specific command. This ID provides the fastest access to a read command but is forbidden for any other action.

- Indirect ID, which only specifies a reading command, the physical address of the slave node that must answer having been passed in a previous writing frame, called a preparing frame. Indirect ID gives more flexibility than a direct one, but provides a slower access to a read command.

Notes

(1) A reading frame with indirect ID must always be consecutive to a preparing frame. It will otherwise not be taken into account.

(2) A reading frame will always return the physical address of the answering slave node in order to ensure robustness in the communication.

The reading frames used with the AMIS-30621 are the followings:

- **Type #5**: 2, 4 or 8 Data bytes reading frame with a direct IDs dynamically assigned to a particular slave node together with an application command. A preparing frame is not needed.
- **Type #6**: 8 Data bytes reading frame with 0x3D identifier. This is intrinsically an indirect type, needing therefore a preparation frame. It has the advantage to use a reserved identifier.

9.2.4.6.3 Preparing Frames

A preparing frame is a writing frame that warns a particular slave node that it will have to answer in the next frame (hence a reading frame). A preparing frame is needed when a reading frame does not use a dynamically assigned direct ID. Preparing and reading frames must be consecutive. A preparing frame will contain the physical address of the LIN slave node that must answer in the reading frame, and will also contain a command indicating which kind of information is awaited from the slave.

The preparing frames used with the AMIS-30621 can be of Type #7 or Type #8 described below.

- **Type #7**: 2 Data bytes writing frame with dynamically assigned identifier.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
*	*	0	ID4	ID3	ID2	ID1	ID0	Identifier
1	CMD[6:0]							Data1
1	AD[6:0]							Data2

*) according to parity computation

- **Type #8**: 8 Data bytes writing frame with 0x3C identifier.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
0	0	1	1	1	1	0	0	Identifier
AppCMD = 0x80								Data1
1	CMD[6:0]							Data2
1	AD[6:0]							Data3
0xFF								Data4-8

AppCMD : 0x80 indicates that Data2 contains an application command byte

CMD[6:0] : Command byte

AD[6:0] : Slave node's physical address

9.2.4.6.4 Dynamic Assignment of Identifiers

Apart from identifiers 0x3C to 0x3F, the LIN rev 1.2 specification does not indicate how identifiers can be allocated. Therefore, slave nodes need to be flexible enough to adapt themselves to a given LIN network.

One solution proposed by BMW is to implement a dynamic assignment of the identifiers by the LIN Master. This is done at start-up of the system by writing identifiers in the slave's RAM to make them correspond to commands pointers located in the slave's ROM. This is the strategy adopted for the AMIS-30621.

Dynamic assignment must be done by a writing frame with identifier 0x3C.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
`0`	`0`	`1`	`1`	`1`	`1`	`0`	`0`	Identifier
`AppCmd = 0x80`								Data1
`1`	`CMD[6:0] = 0x11`							Data2
`Broad`	`AD6`	`AD5`	`AD4`	`AD3`	`AD2`	`AD1`	`AD0`	Data3
`ID1[3:0]`				`ROMp1[3:0]`				Data4
`ID2[1:0]`		`ROMp2[3:0]`				`ID1[5:4]`		Data5
`ROMp3[3:0]`				`ID2[5:2]`				Data6
`ROMp4[1:0]`		`ID3[5:0]`						Data7
`ID4[5:0]`						`ROMp4[3:2]`		Data8

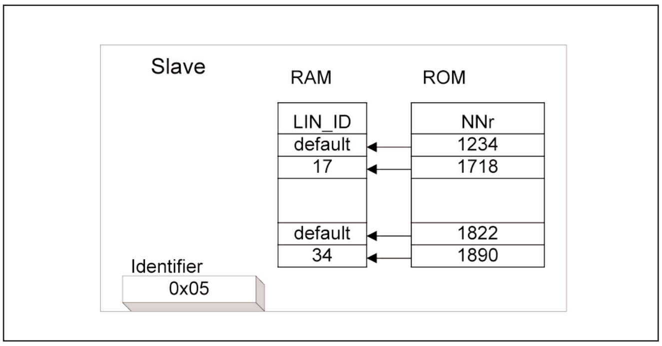
With:

`CMD[6:0]` : 0x11, corresponding to dynamic assignment of 4 LIN identifiers.

`Broad` If **`Broad`** = '0' all the circuits connected to the LIN bus will share the same dynamically assigned identifiers.

`IDn[5:0]` : Dynamically assigned LIN identifier to the application command which ROM pointer is **`ROMpn[3:0]`** (see § 9.2.4.7).

One frame allows only to assign 4 identifiers. Therefore, additional frames could be needed in order to assign more identifiers (maximum 3 for the AMIS-30621).



9.2.4.7 Commands Table

Command Mnemonic	Command Byte (CMD)		Dynamic ID (example)	ROM Pointer
`GetActualPos`	000000	**`0x00`**	100xxx	0010
`GetFullStatus`	000001	**`0x01`**	n.a.	
`GetOTPparam`	000010	**`0x02`**	n.a.	
`GetStatus`	000011	**`0x03`**	000xxx	0011
`GotoSecurePosition`	000100	**`0x04`**	n.a.	
`HardStop`	000101	**`0x05`**	n.a.	
`ResetPosition`	000110	**`0x06`**	n.a.	
`ResetToDefault`	000111	**`0x07`**	n.a.	
`RunInit`	001000	**`0x08`**	n.a.	
`SetMotorParam`	001001	**`0x09`**	n.a.	
`SetOTPparam`	010000	**`0x10`**	n.a.	
`SetPosition (16-bit)`	001011	**`0x0B`**	010xxx	0100
`SetPositionShort (1 motor)`	001100	**`0x0C`**	001001	0101
`SetPositionShort (2 motors)`	001101	**`0x0D`**	101001	0110
`SetPositionShort (4 motors)`	001110	**`0x0E`**	111001	0111
`Sleep`	n.a.	n.a.		
`SoftStop`	001111	**`0x0F`**	n.a.	
Dynamic ID assignment	010001	**`0x11`**	n.a.	
General purpose 2 Data bytes			011000	0000
General purpose 4 Data bytes			101000	0001
Preparation frame			011010	1000

`xxx` allows to address physically a slave node. Therefore, these dynamic IDs cannot be used for more than 8 stepper motors.

Only 9 ROM pointers are needed for the AMIS-30621.

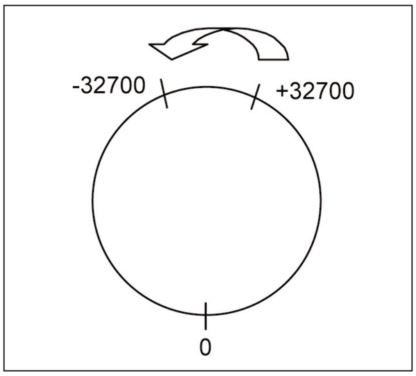
10.0 Features

10.1 Position Periodicity

Depending on the stepping mode the position can range between –4096 to +4095 in half-step mode to –32768 to +32767 in 1/16th microstepping mode (see 8.5.1) one can project all these positions lying on a circle. When executing the command **`SetPosition`** the position controller will set the movement direction in such a way that the traveled distance is minimum.

As an example in the figure below is illustrated the moving direction going from **`ActPos`** = +32700 to **`SetPos`** = –32700 is counter clockwise.

If a clockwise motion is required in this example, several consecutive **`SetPosition`** commands can be used.



11.0 Resistance to Electrical and Electromagnetic Disturbances

11.1 Electrostatic Discharges

See. § 1.1 Absolute Maximum Ratings

11.2 Schäffner Pulses

Shäffner Pulses are applied to the power supply wires of the equipment implementing the AMIS-30621 (see application schematic), according to Renault 36-00-808/--E document.

Pulse	Amplitude	Rise Time	Pulse Duration	Rs	Operating Class
#1	-100V	≤ 1μs	2ms	10Ω	C
#2a	+100V	≤ 1μs	50μs	2Ω	B
#3a	-150V (from +13.5V)	5ns	100ns (burst)	50Ω	A
#3b	+100V (from +13.5V)	5ns	100ns (burst)	50Ω	A
#5b (load dump)	+21.5V (from +13.5V)	≤ 10ms	400ms	≤ 1Ω	C

11.3 EMC

Bulk current injection (BCI), according to Renault 36-00-808/--E document (p61).

Current	Opertaing Class
60mA	A
100mA	B
200mA	C

11.4 EMI

EMI requirement is given here as a target, since it is also PCB dependent. Any EMI issue will have to be solved on common basis with the customer.

Radiated disturbance electromagnetic quietness test, according to Renault 36-00-808/--E document:

- Permanent broadband limit (Renault 36-00-808/--E document diagram p98)
- Narrow band limit (Renault 36-00-808/--E document diagram p99)

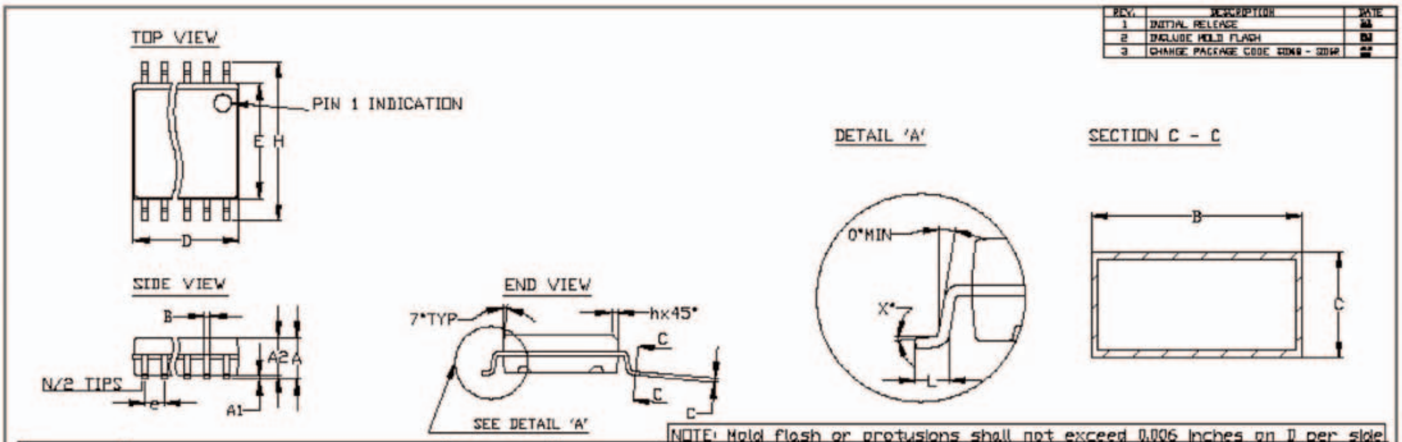
11.5. Power Supply Micro-Interruptions

According to Renault 36-00-808/--E (p47 and followings).

Test	Operating Class
10µs micro-interruptions (1)	A
100µs micro-interruptions	B
5ms micro-interruptions	B
50ms micro-interruptions	C
300ms micro-interruptions	C

Note 1: To achieve Class A a 100nF capacitor between Vbat and ground is needed in case HW is connected to Vbat. (see § 7 typical application)

12.0 Package Outlines



SYMBOL	INCHES MIN	INCHES NOM	INCHES MAX	NOTE	NOTE VARIATIONS	1 D MIN	1 D NOM	1 D MAX	2 N	ALCATEL PACKAGE NR.	ALCATEL PACKAGE CODE
A	0,093	0,099	0,104		AA	0,398	0,405	0,412	16	87_0034	SO16B,SO16R
A1	0,004	0,008	0,012		AB	0,451	0,456	0,461	18	87_0035	SO18,SO18Q
A2	0,088	0,094	0,100		AC	0,496	0,503	0,510	20	87_0036	SO20,SO20Q
B	0,013	0,016	0,020		AD	0,599	0,606	0,613	24	87_0037	SO24,SO24Q
C	0,0090	0,0100	0,0125		AE	0,697	0,704	0,711	28	87_0038	SO28,SO28Q
D	SEE VARIATIONS			1	AF	0,810	0,818	0,822	32	87_0040	SO32,SO32Q
E	0,292	0,296	0,299								
e	0,050 BSC										
H	0,394	0,402	0,419								
h	0,010	0,015	0,019								
L	0,016	0,033	0,050								
N	SEE VARIATIONS			2							
X	0	5	8								

Note: See variations AC for dimensions D and N.

13.0 Conditioning

To be documented.