

# ON Semiconductor

## Is Now



To learn more about onsemi™, please visit our website at  
[www.onsemi.com](http://www.onsemi.com)

onsemi and onsemi. and other names, marks, and brands are registered and/or common law trademarks of Semiconductor Components Industries, LLC dba "onsemi" or its affiliates and/or subsidiaries in the United States and/or other countries. onsemi owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of onsemi product/patent coverage may be accessed at [www.onsemi.com/site/pdf/Patent-Marking.pdf](http://www.onsemi.com/site/pdf/Patent-Marking.pdf). onsemi reserves the right to make changes at any time to any products or information herein, without notice. The information herein is provided "as-is" and onsemi makes no warranty, representation or guarantee regarding the accuracy of the information, product features, availability, functionality, or suitability of its products for any particular purpose, nor does onsemi assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using onsemi products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by onsemi. "Typical" parameters which may be provided in onsemi data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. onsemi does not convey any license under any of its intellectual property rights nor the rights of others. onsemi products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use onsemi products for any such unintended or unauthorized application, Buyer shall indemnify and hold onsemi and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that onsemi was negligent regarding the design or manufacture of the part. onsemi is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner. Other names and brands may be claimed as the property of others.

# LV8702VSLDGEVK

## Stepper Motor Module Solution Kit

This Application Note provides supporting information for the LV8702VSLDGEVK Motor Driver Solution Kit.

### KIT OVERVIEW

When developing a motor control application system using any motor driver product provided by ON Semiconductor, it is first necessary to design the hardware after understanding the specifications of the motor driver product. And then proceed to generating operation control signals (for the rotating direction, speed, angle, etc.) that will be inputted into the driver IC. Given that operation control signals like these are generally generated with the use of a microcomputer, it is necessary to develop software for the microcomputer in addition to the hardware design mentioned above.

This kit provides an API function library for motor driver control designed to control the ON Semiconductor motor driver product (LV8702V) via the Arduino Micro microcomputer. It also provides a dedicated GUI for controlling motors connected to the Arduino Micro microcomputer that embedded with the API library from a



ON Semiconductor®

[www.onsemi.com](http://www.onsemi.com)

### APPLICATION NOTE

computer via USB communication. This means that it is possible to easily tune and otherwise debug motor control sequences and operation parameters without developing motor control software for the Arduino Micro.

In addition, this GUI also has an automatic code generation feature. It outputs control software (source code) to achieve debug control sequences and operation parameters on the Arduino Micro microcomputer in a format (sketch) in which it can be compiled with the Arduino IDE.

As a result, the use of this kit allows users to easily develop a prototype of motor control application system without special knowledge of the specifications of the motor driver on the software development using the API functions. It also reduces the development period and significantly lowers costs.

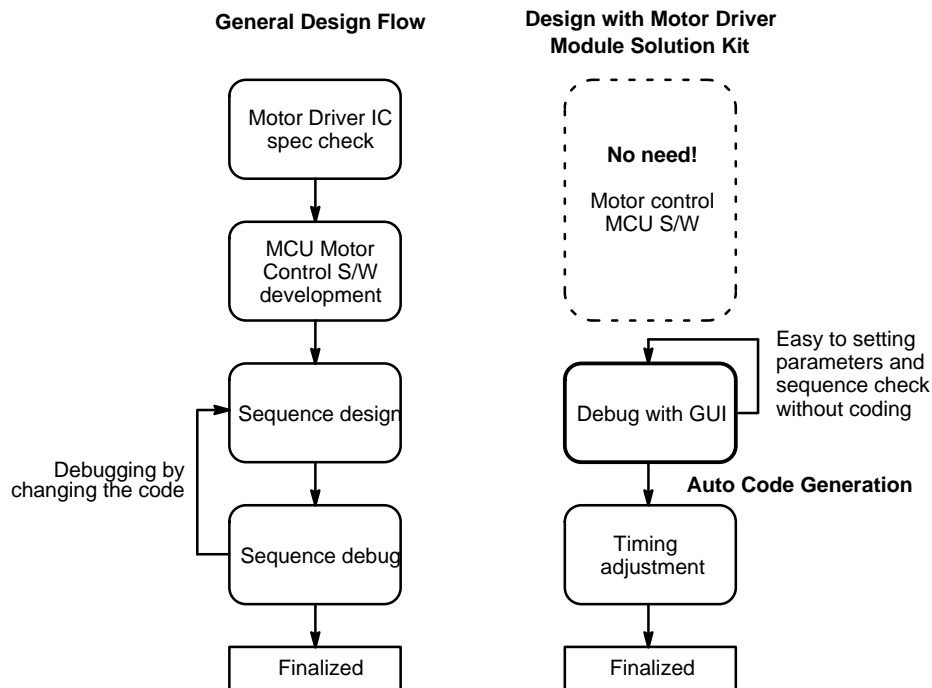


Figure 1. General Development Process Flow and Development Process Flow Using This Kit

## GUI DEBUG MODE AND STANDALONE DEVELOPMENT MODES

This kit is presumed to be used in a total of three different debug and development modes: the GUI debug mode, the standalone development mode using the automatic code generation feature, and the standalone development mode using an original sketch.

### GUI Debug Mode

In the GUI debug mode, the user will operate the dedicated GUI installed on the computer to change motor control sequences and operation parameters for the LV8702V. The user can tune the parameters while actually operating the motor.

It also has an automatic code generation feature, which outputs a sketch (an .ino file) that can be compiled and written into the Arduino Micro that reflects the motor control sequences and operation parameters set by the user by operating the GUI.

To use this mode, it is necessary to write firmware generated by using Arduino IDE into the Arduino MICRO. The user must compile the sketch for the GUI (LV8702\_Program.ino) with motor driver API Functions library (LV8702\_Lib.cpp/h) and TimerOne library which is separately downloaded off the Internet into the Arduino MICRO to generate the firmware by Arduino IDE.

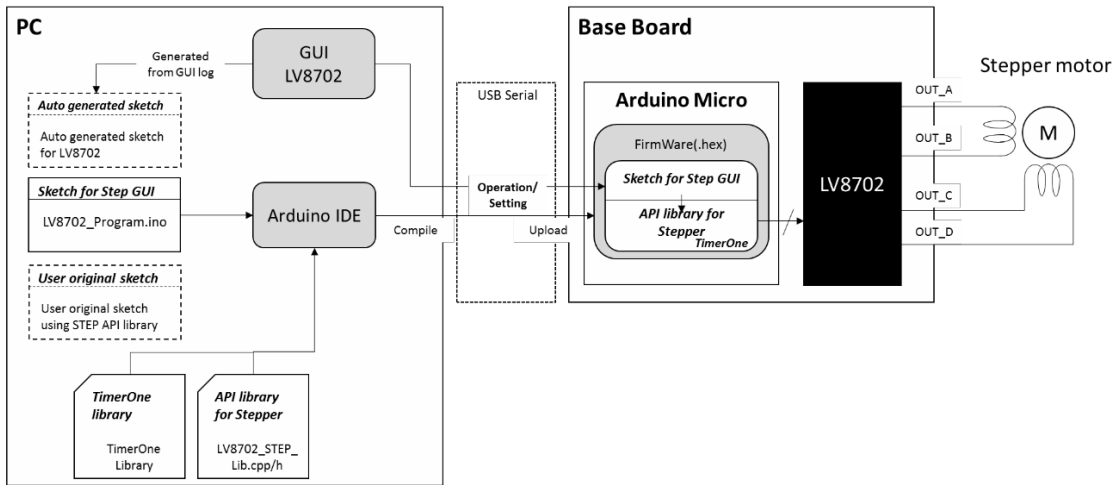


Figure 2. Outline of the GUI Debug Mode

### Standalone Development Modes

In the standalone development modes, the user adjusts the timing of the motor drive and adds the user's original source code based on the automatically generated sketch in the GUI

debug mode and writes them into the Arduino Micro instead of the sketch that is exclusive for the GUI to facilitate standalone Stepper motor driving using this kit.

Figure 3 provides a graphical representation of this mode.

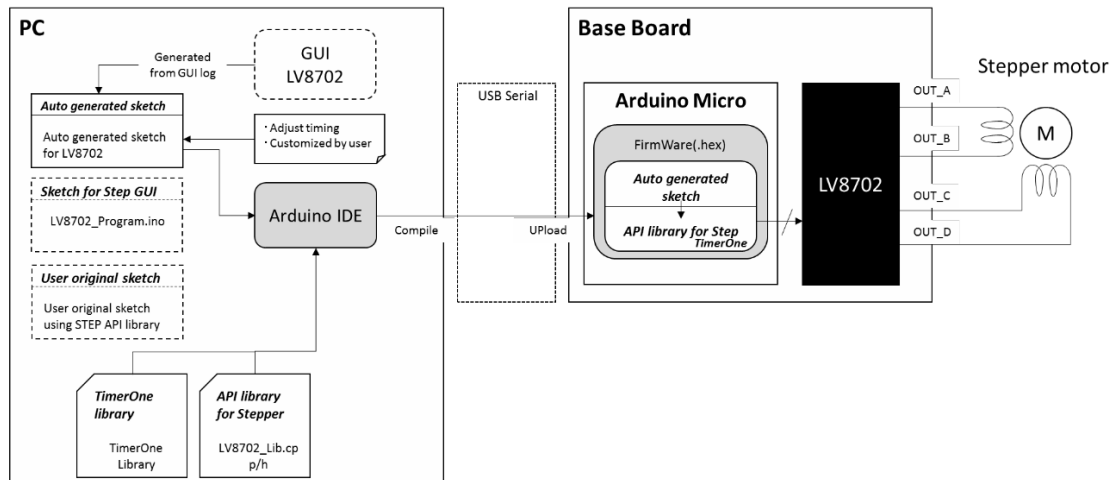


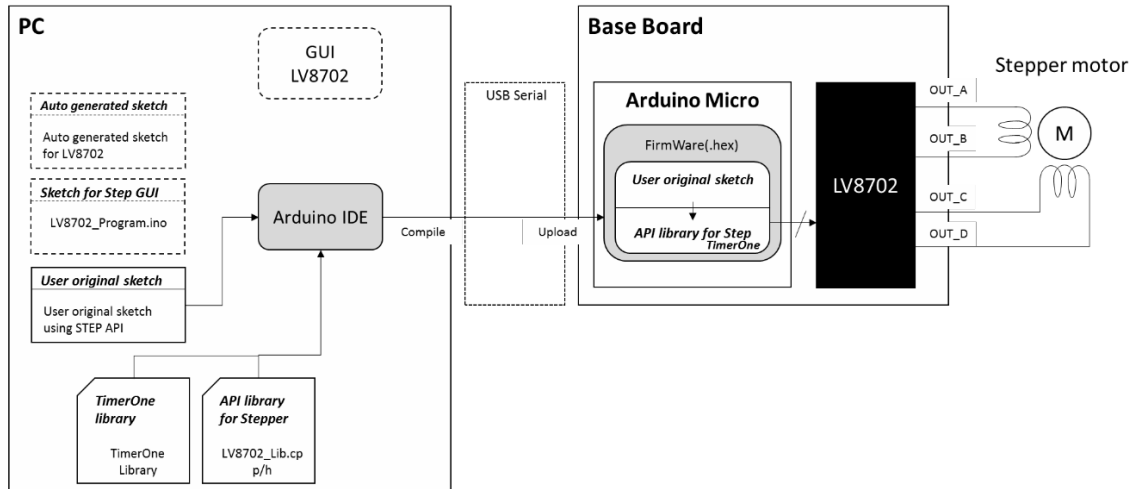
Figure 3. Standalone Development Mode Using Automatic Generation Feature

## LV8702VSLDGEVK

It is also possible to develop sketches from scratch with the API function library instead of using the automatic code generation feature. With advanced knowledge of

programming, it is possible to develop a more complicated and sophisticated motor control application.

Refer to Figure 4.



**Figure 4. Standalone Development Mode Using Original Sketch**

## PROGRAMMING GUIDE

### ARDUINO SKETCH

#### Sketch Overview

A sketch is a program written in Arduino language used for the Arduino. A sketch is a program, or the unit of code that is compiled, uploaded to and run on an Arduino board. The Arduino language is based on the C/C++ languages, and it supports the full structure of the C language and some features of the C++ language.

For details and help regarding Arduino commands, please refer to: <https://www.arduino.cc/en/Tutorial/Sketch>

*setup() and loop()*

If you create a new sketch on the Arduino IDE, *setup()* and *loop()* will be automatically inserted as below.



*setup()* is a function that is called only once after the Arduino Board is powered on or reset. This function mainly conducts initial settings including preparation of the libraries to be used, initialization of variables, and initialization of pin modes.

*loop()* is, as its name suggests, a function that is executed repeatedly after *setup()* function executed. It contains programs which should be run many times

#### Overview of Motor Driver Library

The motor driver library (LV8702\_APILibrary) provides a library for controlling a Stepper motor with the use of ON Semiconductor's LV8702V motor driver from the Arduino Micro. Stepper motor control using the LV8702V is easily achieved by including this API library via the Arduino IDE and calling the API functions suited to the purposes in the sketch

**Table 1. LIST OF MOTOR DRIVER LIBRARY FILES**

#	File Name	Description
1	keywords.txt	Keyword file (sets highlighted words in sketches)
2	LV8702_Lib.cpp	Source file
3	LV8702_Lib.h	Header file

#### Using the Stepper Motor Driver API Library

For instructions for the inclusion of the library, refer to the Quick Start Guide.

To use the stepper motor control API library in the Arduino, include the header file of the stepper motor control API library at the beginning of the sketch, as explained below, and instantiate the class to be used.

To use the GUI tool, it is necessary to separately call the serial communication API. For details, refer to [API Function Specifications](#). The following shows a sketch that includes the stepper motor control API library.

#### Inclusion of the motor driver library

```
#include <LV8702_Lib.h>
// Importing a header file for using the API function for LV8702V
#include <TimerOne.h>
// Importing a header file for using the TimerOne library
Lib_LV8702V Lib // Instantiating-LV8702V class (Note 1)
void setup(){ // Function called at the start (Refer to setup\(\) and loop\(\))
}
void loop(){ // Function executed repeatedly (Refer to setup\(\) and loop\(\))
}
```

1. In this example, the instantiation was conducted with the name of Lib. The addition of 'Lib.' as prefix makes it possible to call the API functions in the Lib\_LV8702V class.  
e.g. Lib.initLib();

#### Compiling and Writing a Sketch into Arduino

For the steps for compiling and writing a sketch (Arduino program), refer to *Compiling an Arduino Program and Writing into Arduino* in Quick Start Guide.

## CODING A PROGRAM (SKETCH)

### Automatic Code Generation

The following explains the functions which are written by the automatic code generation at the timing of sketch output.

An generated sample sketch is used as an example for explanation.

For details of the API functions of the motor driver library called by the GUI debug operation, refer to section [API Function Specifications](#).

A sample of automatically generated code

```
#include <LV8702_Lib.h> // Refer to Using the Stepper Motor Driver API Library
#include <TimerOne.h> // Refer to Using the Stepper Motor Driver API Library
Lib_LV8702V Lib; // Refer to Using the Stepper Motor Driver API Library
void setup(){ // Refer to setup\(\) and loop\(\)
  Serial.begin(19200); // Set the baud rate at 19200 and open the port (Note 2)

  Lib.initLib(); // Initialize the Arduino parameters and registers

  Timer1.initialize(50); // Timer1 is initialized with 50us interrupt period
  Timer1.attachInterrupt(interrupt); // Attached interrupt handler routine for timer
  delay(5000); // Interval time [ms] after start-up of the Arduino (Note 3)

  Lib.setChipEnable(1);
  delay(0); // Interval time [ms] after execution of an API function
  Lib.setStepAngle(7.5);
  delay(0); // Interval time [ms] after execution of an API function
  Lib.motorRotationStep(100, 100.0, 1, 1);
  delay(0); // Motor driving time [ms] (Note 4)
  Lib.motorRotationStop();
  delay(0); // Interval time [ms] after execution of an API function
  Lib.motorRotationFree();
  delay(0); // Interval time [ms] after execution of an API function
}
void interrupt(){ // Interrupt handler routine that is called when timer is fired
  Timer1.initialize(Lib.timerFire());
}
void loop(){ // Refer to Sketch Overview
}
```

2. This function is required in the case of using serial communication.  
If not using serial communication, you can remove this without affection for the motor operation.
3. The default value of 5000 ms (5 sec) is set.
4. delay() after the start of the motor means the motor driving time.  
The default value of 0 is set. Change the value as required.

## Using an Automatically Generated Sketch

An automatically generated sketch is so simply structured that it is easy for programming beginners to use. Customization will turn it into a more practical program. This sections sketch is representative of the functionality of

the Arduino setup part and motor driver part which can be called using setup() and loop(). An example of this customization is shown in the sketch generated in [Automatic Code Generation](#)

```
#include <LV8702_Lib.h>
#include <TimerOne.h>
Lib_LV8702V Lib;
void setup(){
  motorSetup();// The functionalized initial settings of the Arduino are called with setup()
}
void interrupt(){
  Timer1.initialize(Lib.timerFire());
}
void loop(){
  motorControl();// The functionalized motor drive part is called with loop().
}
// The initial settings of the Arduino are functionalized. //
void motorSetup(){
  Serial.begin(19200);
  Lib.initLib();
  Timer1.initialize(50);
  Timer1.attachInterrupt(interrupt);
  delay(5000);
}
// The motor drive part is functionalized //
void motorControl(){
  Lib.setChipEnable(1);
  delay(0);
  Lib.setStepAngle(7.5);
  delay(0);
  Lib.motorRotationStep(100, 100.0, 1, 1);
  delay(5000); // The delay value is changed to 5000 and the motor 1 drive time is set to 5000 [ms].
}
```

## Example of Application

### Overview

In this example, we will develop an application to use the stepper motor (MDP-35A) provided with the kit as a winch (hoist) of a crane.

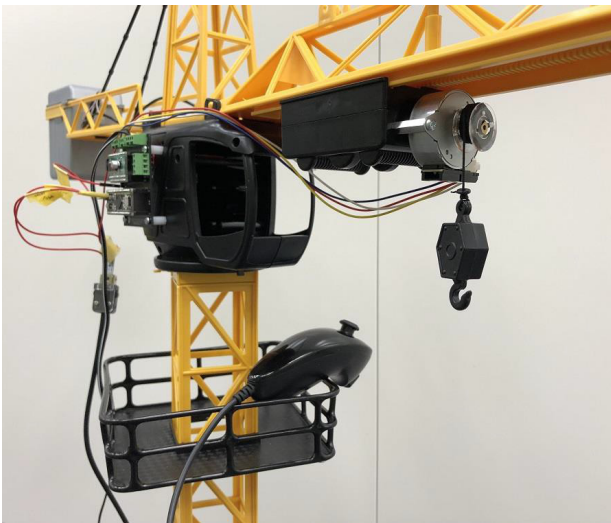


Figure 5. FA Crane and a Winch (Hoist)

### Specifications

- Joystick controls hoisting, lowering, and stopping of the winch.
- Button C            hoisting    100 rpm keep hoisting while button being pressed
- Button Z            lowering    100 rpm keep lowering while button being pressed
- Button C + Z       Stop
- The joystick does polling with the 200 ms monitoring period by I<sup>2</sup>C polling.

# LV8702VSLDGEVK

## Items required

- |                              |   |                    |   |
|------------------------------|---|--------------------|---|
| • Module Solution Kit        | MDP-35A in this kit is also used                  | • Toy crane        | Not essential to program the kit, but more fun to play with!  |
| • Bobbin for sewing machines | Used as a part of the winch (plastic recommended) | • Joystick         | Control the winch<br>(Example: Wii Nunchuk<br><a href="https://www.nintendo.co.jp/wii/controllers/index.html#nunchuk">https://www.nintendo.co.jp/wii/controllers/index.html#nunchuk</a> ) |
| • Screw to fix the bobbin    | Around the size of 2Φ 2 mm                        | • Joystick adaptor | Support joystick connection to Base Board<br>(Example: WiiChuck adaptor<br><a href="https://www.sparkfun.com/products/retired/9281">https://www.sparkfun.com/products/retired/9281</a> )  |
| • String for the winch       | String length is arbitrary.                       |                    |   |
| • Hook for lifting           | Recommended to use adequate weight stability      |                    |   |

## Block Diagram

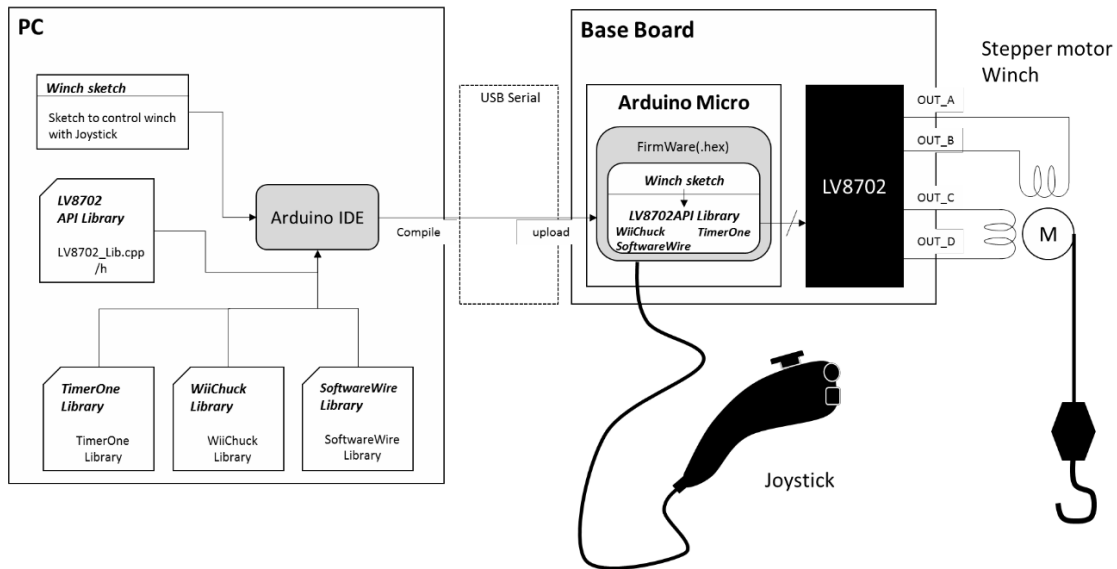


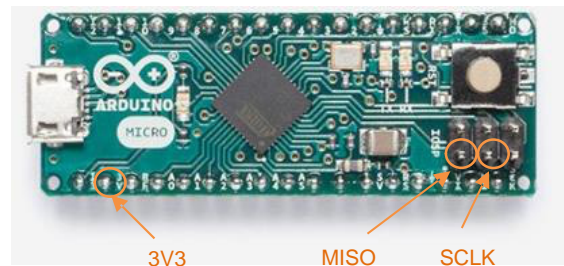
Figure 6. Block Diagram

## Connection

- Motor  
Connect to Base Board as explained in the Quick Start Guide.
- Joystick  
Connector side



If a joystick adaptor is not available, disassemble the joystick connector and wire directly, if desired.  
Arduino Micro side



Positions of 3V3, MISO, SCLK connections  
Though wiring needs to be drawn for 3.3 V (3V3) from Base Board, GND can be taken from J2.

Connect the joystick adaptor to Arduino Micro

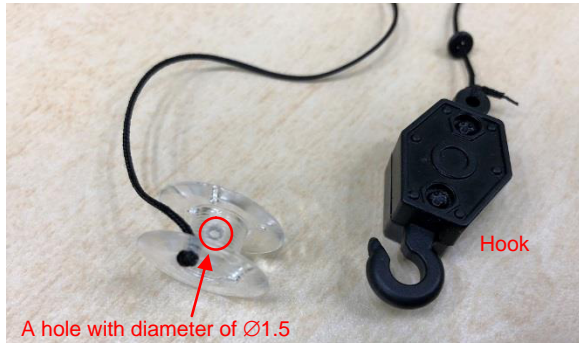
- ♦ Adaptor (+) → Arduino Micro (3V3)
- ♦ Adaptor (-) → Arduino Micro (GND)
- ♦ Adaptor (d) → Arduino Micro (MISO)
- ♦ Adaptor (c) → Arduino Micro (SCLK)



## LV8702VSLDGEVK

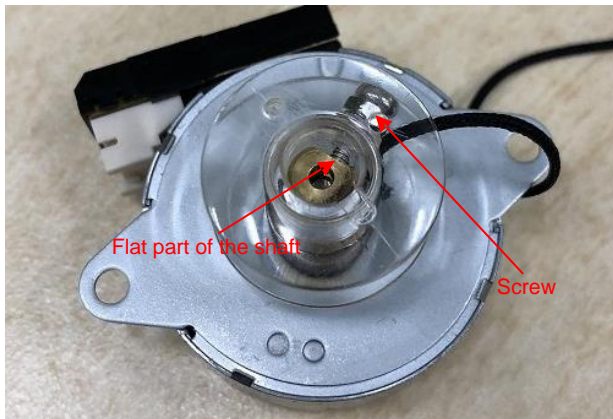
- Bobbin

Fix the bobbin to the shaft of the stepper motor, then connect string to it.



Customizing a bobbin, string and hook

The motor shaft is 6Ø in diameter, the hole of a bobbin is 6 to 6.2. Drill a hole with the diameter of Ø1.5 on the bobbin run the string through the hole and tie a knot in the string to prevent the string from pulling back through the bobbin. See figure above



Fixing the bobbin

Align the hole drilled in the bobbin with the flat part of the motor shaft. Insert a 2Ø screw and tighten it to fix the bobbin securely to the motor shaft.

### *Make the Base Code Using the GUI*

Before coding, here we use the GUI to look up the API that corresponds to the operation of the motor.

#### **Step 1: Write the Firmware:**

Write the firmware to the Arduino following the steps explained in Quick Start Guide. Double-click the sketch LV8702\_Program.ino to launch the Arduino IDE and push “→” Button or execute [Sketch] → [Upload]. Make sure that the COM port to which the Arduino is connected is selected from [Tools] > [Port].

#### **Step 2: Debug by the GUI:**

Launch the GUI, select the COM port to which the Arduino is connected from the Serial Port Settings, and click Connect. The LV8702 tab opens automatically.

Then set the GUI parameters as follows:

- Excitation Half Step (Full-torque)
- Direction CW
- Step Angle 7.5 (Optimize for the motor used, press Set button to enable the set value)
- Transfer Unit Seconds
- Transfer Step 0 (Infinity)
- Chip Enable/Disable Operation
- Power Supply/Motor Spec Optimize for the power supply and the motor

After setting these parameters, change Motor Speed to 300 step/s and press Start.

Find the right values for Motor Speed and check the parameters when the rotational directions (CW, CCW) is switched through this debugging process. Here we tried 300 step/sec at first and found that's too fast. Therefore, we kept trying motor rotations while decreasing the speed (you can see this process in the sample sketch in the next step).

## Step 3: Automatic Code Generation by the GUI:

Use code generation function to examine the API functions which were executed during the operation with the GUI.

A sample sketch from automatic code generation

```
#include <LV8702_Lib.h>
#include
<TimerOne.h>
Lib_LV8702V Lib;
void setup()
{
    Serial.begin(19200);
    Lib.initLib();
    Timer1.initialize(50);
    Timer1.attachInterrupt(interrupt);
    delay(1000);
    Lib.setChipEnable(1);
    delay(0);//0msec
    Lib.setStepAngle(7.5);
    delay(0);//0msec
    Lib.motorRotationTime(300, 0, 0, 1); // CW
    delay(0);//0msec
    Lib.motorRotationStop();
    delay(0);//0msec
    Lib.motorRotationTime(200, 0, 1, 1); // CCW
    delay(0);//0msec
    Lib.motorRotationStop();
    delay(0);//0msec
    Lib.motorRotationTime(100, 0, 1, 1); // CCW
    delay(0);//0msec
    Lib.motorRotationStop();
    delay(0);//0msec
    Lib.motorRotationTime(100, 0, 0, 1); // CW
    delay(0);//0msec
    Lib.motorRotationStop();
    delay(0);//0msec
}
void interrupt()
{
    Timer1.initialize(Lib.timerFire());
}
void loop()
}
```

Based on this code, we can start to develop the specific application code.

### Application Development

Here we begin to write the actual code to control a stepper motor with a joystick.

In this section, we use WiiChuck (<https://www.arduino-libraries.info/libraries/wii-chuck>) library to retrieve data from a joystick (Wii Nunchuk).

Also, because LV8702V module use many terminals of the Arduino, the I<sup>2</sup>C terminals of the Arduino are already occupied. Use SoftwareWire

(<https://github.com/Testato/SoftwareWire>) library that allows I<sup>2</sup>C slaves to be accessed by controlling general-purpose GPIO terminals from software.

NOTE: You can use the WiiChuck and SoftwareWire libraries by copying them under Document\Arduino\libraries.

### Step 1 Modification of WiiChuck Library:

The WiiChuck library is coded to use I<sup>2</sup>C hardware of the Arduino. Modify this to use SoftwareWire by Replacing #include <Wire.h> in the beginning of Accessory.cpp in WiiChuck library with the red portion shown below.

Accessory.cpp

```
#include "Accessory.h"

#include <SoftwareWire.h>
#define MISO 14 //!< Arduino micro MISO pin number define
#define SCLK 15 //!< Arduino micro SCK pin number define
SoftwareWire _i2c(MISO, SCLK, false, true);

Accessory::Accessory() {
```

Next replace all instances of “Wire.” in Accessory.cpp to “\_i2c.” which is the name of SoftwareWire instance which was instantiated in the red portion above. The WiiChuck library modification is now complete.

## STEP 2 Code Example for the Application:

Now we actually start coding. WiiChuck significantly simplifies this effort.

```
#include <LV8702_Lib.h>
#include <TimerOne.h>
#include <WiiChuck.h> // WiiChuck library header
Accessory nunchuck1; // WiiChuck instantiation Lib_LV8702V Lib;
void setup()
{
  Serial.begin(19200);
  Lib.initLib(); Timer1.initialize(50);
  Timer1.attachInterrupt(interrupt);
  nunchuck1.begin(); // WiiChuck Initialization
  if (nunchuck1.type == Unknown) {
    nunchuck1.type = NUNCHUCK;
  }
  delay(1000);
  // Leave basic motor setting in setup()
  Lib.setChipEnable(1); // Chip Enable (Operation)
  Lib.setStepAngle(7.5); // Set Step angle to 7.5°
}
void interrupt()
{
  Timer1.initialize(Lib.timerFire());
}
void loop()
{
  nunchuck1.readData(); // Read inputs and update
  maps int joy_y = nunchuck1.getJoyY();
  Serial.println(joy_y); // For debugging purpose
  if (nunchuck1.getButtonC() == true && nunchuck1.getButtonZ() == true) {
    // Stop if C button and Z button are pressed
    Lib.motorRotationStop();
  } else if (nunchuck1.getButtonC() == true) {
    // Rotate clockwise if C button is pressed.
    Lib.motorRotationTime(100, 0, 0, 2);
  } else if (nunchuck1.getButtonZ() == true) {
    // Rotate counterclockwise if Z button is pressed.
    Lib.motorRotationTime(100, 0, 1, 2);
  }
  delay(200); // Add 200 ms delay to the loop.
}
```

## Next Step

Using this example as a reference, try more advanced application development like the followings:

1. Examine the difference between excitation methods  
Use different excitation methods to see how the operation or sound differs.
2. Try the high efficiency drive mode  
Enable the high efficiency drive mode (by setting it High) to examine the difference of the current consumption (the sound will also be different).

3. Make full use of joysticks.

The inclination of the joystick can be obtained numerically. You can use `int joy_y = nunchuck1.getJoyY();` to get the inclination for Y direction in values between 0 to 255. Use them to change the motor speed or the number of steps. In addition, the value of the acceleration sensor can also be obtained. See WiiChuck's examples directories for more information.

# LV8702VSLDGEVK

## API SPECIFICATIONS

### Overview of the Stepper Motor Control API

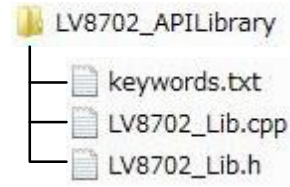
This section outlines the API for control of the Arduino Micro Stepper motor for LV8702V motor driver.

#### Outline

This API provides a library for controlling the stepper motor from the Arduino Micro with the use of ON Semiconductor's LV8702V motor driver. It allows the user to easily control the stepper motor using LV8702V by including the API library in the Arduino IDE and by writing the API functions that match with the user's desired purpose(s) of use in the sketch.

When using this API library, please note that it is also necessary to include the separate TimerOne library. For details of how to include the TimerOne library, refer to Quick Start Guide "Including the TimerOnce library."

## Library File Structure





**Table 2. LIST OF LIBRARY FILES**

#	File Name	Description
1	keywords.txt	Keyword file (sets highlighted words in sketches)
2	LV8702_Lib.cpp	Source file
3	LV8702_Lib.h	Header file

### Pin Assignment of the Arduino Micro/LV8702V Base Board

The following portrays the pin assignment of the Arduino Micro/LV8702V Base Board.

The white background color for the Arduino Micro input and the output pins represents the resources available to users.

Description (*1)	BaseBoard Pin#	LV8702V Pin name	Arduino Micro		LV8702V Pin name	BaseBoard Pin#	Description (*1)
Boost-up adjuster output pin	CN3A-2	GST1	MO		SCK	CN3A-3	
		NC	SS		MI	CN3A-4	
Serial1 Transmitter output pin	TX	NC	Tx		VI	NC	
Serial1 Receiver input pin	RX	NC	Rx			SGND	CN3A-1
		NC	RST		RST	NC	Signal GND
Signal GND	CN3A-1	SGND			5V (Pull-up)	CN3A-5	
Driving capability margin adjuster output pin	CN3B-1	GMG1	2		NC	NC	
Driving capability margin adjuster output pin	CN3B-2	GMG2	3		NC	NC	
High-efficient drive switching output pin	CN3B-3	GAD	4		A5	SST	CN3A-6
Forward/ reverse signal output pin	CN3B-4	FR	5		A4	MONI	CN3A-7
Constant current control reference voltage control output pin	CN3B-5	(PWM_VREF)	6		A3	DST1	CN3A-8
RESET signal output pin	CN3B-6	RST	7		A2	DST2	CN3A-9
Excitation mode switching output pin	CN3B-7	MD2	8		A1	VREF	CN3A-10
Excitation mode switching output pin	CN3B-8	MD1	9		A0	P/N	CN3A-11
Output enable signal output pin	CN3B-9	OE	10		RF	NC	
STEP signal output pin	CN3B-10	STEP	11		3V	NC	CN3A-12
Boost-up adjuster output pin	CN3B-11	GST2	12		13	ST	CN3B-12
				USB			Chip enable output pin

\*1 Inputs and outputs in this column refer to those of the Arduino Micro.

**Figure 7. Pin Assignment of Arduino Micro/LV8702V Base Board**

## LV8702VSLDGEVK

### INITIAL SETTINGS FOR THE STEPPER MOTOR CONTROL API

This section describes the initial settings for the Stepper motor control API.

### Resources Used by the API

This API uses the Arduino pins shown in Table 3 and the corresponding ATmega32U4 timer register, which are not available to users. See ATmega16U4–32U4\_Datasheet.pdf for more information about the register.

**Table 3. ARDUINO MICRO PIN AND CORRESPONDING ATmega32U4 TIMER REGISTER**

#	Arduino Pin	Timer Register	Description	Explained in
1	D6, D13	TCCR4B	Timer/Counter4 Control Register B	<a href="#">Details of Timer Resistor Settings</a>

### Initial settings

Arduino Micro's output pins are initialized with the initLib function. At the time of using this API library, be sure to call the initLib function in setup() in a sketch to initialize the parameters, timer registers, and output pins.

For details of how to use the initLib function, refer to [initLib](#).

**Table 4. INITIALIZED PIN SETTINGS**

#	Initialized Items	Initial Setting Value
1	Timer registers	Refer to <a href="#">Details of Timer Resistor Settings</a>
2	Input and output pins	Refer to table 5
3	Parameters	Refer to <a href="#">Internal Parameters List</a>

**Table 5. INITIALIZED PIN SETTINGS**

#	Arduino Micro Output Pin	Initial Setting Value	Relationship
1	D2	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
2	D3	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
3	D5	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
4	D6	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
5	D7	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
6	D8	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
7	IO8	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
8	IO9	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
9	IO10	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
10	IO11	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
11	IO12	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
12	IO13	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>
13	MOSI	OUTPUT	<a href="#">Pin Assignment of the Arduino Micro/LV8702V Base Board</a>

## API FUNCTION SPECIFICATIONS

## Overview of API Functions

#	Function	Description	Chapter
1	initLib	<ul style="list-style-type: none"> <li>• Register settings</li> <li>• Input/output pin settings</li> <li>• Parameters settings</li> </ul>	<a href="#">initLib</a>
2	setChipEnable	<ul style="list-style-type: none"> <li>• Switch the mode of the IC between standby mode and operation mode</li> </ul>	<a href="#">setChipEnable</a>
3	setReset	<ul style="list-style-type: none"> <li>• Reset the excitation position</li> </ul>	<a href="#">setReset</a>
4	setMaxCurrent	<ul style="list-style-type: none"> <li>• Maximum output motor current setting</li> </ul>	<a href="#">setMaxCurrent</a>
5	setRefVoltage	<ul style="list-style-type: none"> <li>• Output voltage setting</li> </ul>	<a href="#">setRefVoltage</a>
6	setStepAngle	<ul style="list-style-type: none"> <li>• Step angle setting</li> </ul>	<a href="#">setStepAngle</a>
7	motorRotationDeg	<ul style="list-style-type: none"> <li>• Rotates the motor by the set number of degrees</li> </ul>	<a href="#">motorRotationDeg</a>
8	motorRotationTime	<ul style="list-style-type: none"> <li>• Rotates the motor by the set rotation time</li> </ul>	<a href="#">motorRotationTime</a>
9	motorRotationStep	<ul style="list-style-type: none"> <li>• Rotates the motor for the set number of steps</li> </ul>	<a href="#">motorRotationStep</a>
10	motorRotationStop	<ul style="list-style-type: none"> <li>• Stops the motor (but maintains the excitation state)</li> </ul>	<a href="#">motorRotationStop</a>
11	motorRotationFree	<ul style="list-style-type: none"> <li>• Stops the motor (turns all outputs off)</li> </ul>	<a href="#">motorRotationFree</a>
12	setEfficiency	<ul style="list-style-type: none"> <li>• High efficiency drive setting</li> </ul>	<a href="#">setEfficiency</a>
13	readAdc	<ul style="list-style-type: none"> <li>• Analog voltage measurement</li> </ul>	<a href="#">readAdc</a>
14	readDriveStatus	<ul style="list-style-type: none"> <li>• Reads the status of A2(DST2), A3(DST1), A4(MONI) and returns a single value which contains those status.</li> </ul>	<a href="#">readDriveStatus</a>
15	clrDstCount	<ul style="list-style-type: none"> <li>• Initialize DST count error and MONI error</li> </ul>	<a href="#">clrDstCount</a>
16	guiSerialRead	<ul style="list-style-type: none"> <li>• Analyzes Bytestream made of the data sent by serial communication and calls the API</li> </ul>	<a href="#">guiSerialRead</a>
17	guiSerialParse	<ul style="list-style-type: none"> <li>• Analyzes the data sent by serial communication from the GUI</li> </ul>	<a href="#">guiSerialParse</a>

## Details of API Functions

### *initLib*

**Table 6. initLib**

API function	initLib()		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>void</i>	None	None
Return values	Type		Description
	<i>int</i>		0: "Success" 1: "Failure"
Processing outline	1. Set timer resistors 2. Set output pins. • D2~D7, IO8~IO13, MOSI pin 3. Initialize functions • setChipEnable(), setReset(), setRefVoltage(), setStepAngle(), setEfficiency(), _setOutputEnable(), _setDirection(), _setExcitation()		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() { }</pre>		

### *setChipEnable*

**Table 7. setChipEnable**

API function	setChipEnable(byte select)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>byte</i>	select	0 (standby mode) / 1(operation mode)
Return values	Type		Description
	<i>int</i>		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Change chip enable (ST pin) setting</li> </ul> * Refer to <a href="#">Pin Assignment for Arduino Micro/LV8702V Base Board</a> for more details select = 1 sets the IC in operation mode select = 0 sets the IC in standby mode (the IC becomes inoperational)		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization   Lib.setChipEnable(1); // Operation mode   Lib.setChipEnable(0); // Standby mode } void loop() { }</pre>		

*setReset***Table 8. setReset**

API function	setReset( <i>byte</i> select)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>byte</i>	Select	0 (Reset OFF) / 1 (Reset ON)
Return values	Type		Description
	<i>Int</i>		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Change the Reset (RST Pin) setting</li> </ul> <p>* Refer to <a href="#">Pin Assignment for Arduino Micro/LV8702V Base Board</a> for more details  reset = 0 sets Reset OFF (Normal state)  reset = 1 sets Reset ON (excitation position is fixed to the initial position and the IC becomes inoperational)</p>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization   Lib.setChipEnable(1); // Operation mode   Lib.setOutputEnable (1); // Output ON } void loop() {   Lib.motorRotationDeg(1000,180,0,0); // Rotate the motor in Fullstep, frequency 1 kHz, forward, 180 degrees   condition   delay(1000);   <b>Lib.setReset(1);</b> //RESET ON   <b>Lib.setReset(0);</b> //RESET OFF }</pre>		



*setMaxCurrent***Table 9. setMaxCurrent**

API function	setMaxCurrent(float adpVolt, float adpCrt, float mtrCrt, float mtrRst)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	float	adpVoltage	Supply voltage: 9~32 [V]
	float	adpCurrent	Supply current: 0~10 [A]
	float	mtrCurrent	Motor rated current: 0.1~2.5 [A]
	float	mtrResistance	Motor winding resistance: 0.1~500 [ $\Omega$ ]
Return values	Type		Description
	Int		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Derive the limit of the output current and set the value (Use the smallest value among adpVoltage / mtrResistance, adpCurrent / 2, and mtrCurrent as the limit of the current)</li> <li>• Store the result into "_current max" (a variable for output motor current limit value, this variable is used in the GUI and doesn't affect other API function operations. )</li> </ul>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization   Lib.setMaxCurrent(9.0, 2.0, 2.5, 36.0); // Supply voltage 9.0 [V], Supply current 2.0 [A], Motor rated current 2.5 [A],   Motor winding resistance 36.0 [<math>\Omega</math>] } void loop() { }</pre>		

*setRefVoltage***Table 10. setRefVoltage**

API function	setRefVoltage(float vref)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	float	Vref	Output current: 0~3 [V]
Return values	Type		Description
	Int		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Translate vref(output voltage) → dutyVal(duty cycle)</li> <li>• Set PWM duty cycle in PWM_VREF output D6 pin</li> </ul>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization   Lib.setRefVoltage(0.2); // Reference voltage 0.2 [V] } void loop() { }</pre>		

*setStepAngle***Table 11. setStepAngle**

API function	setStepAngle( <i>float</i> angle)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>float</i>	Angle	Step angle: 0.01~360 [degree/step]
Return values	Type		Description
	<i>Int</i>		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Set the step angle</li> </ul>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization   Lib.setStepAngle(1.8); // Step angle 1.8 degrees. } void loop() { }</pre>		

*motorRotationDeg***Table 12. motorRotationDeg**

API function	motorRotationDeg( <i>float</i> freq, <i>float</i> deg, <i>byte</i> direction, <i>byte</i> excitation)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>float</i>	Freq	Frequency [Hz] 1 – 4800
	<i>float</i>	Deg	Drive time [sec] 0.01~16777215 0: infinity (perpetual drive)
	<i>Byte</i>	direction	Direction of rotation (0: clockwise, 1: counter clockwise)
	<i>Byte</i>	excitation	Method of excitation 0: Full step, 1: Half step (Full-torque), 2: Half step (Smooth), 3: Quarter step
Return values	Type		Description
	<i>Int</i>		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Enable output (OE pin)</li> <li>• Rotates the motor by the number of degrees specified by the parameter deg, while controlling the frequency, direction of rotation and method of excitation as specified in the parameters freq, direction and excitation. (Refer to <a href="#">Internal List Parameters</a> for initial values for each parameter.)</li> <li>• If motorRotationFree() (turn off all outputs) conducted just before, this function enables the output and then rotate the motor)</li> </ul>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   Lib.motorRotationDeg(1000,180,0,0); // Rotate the motor in Fullstep, frequency 1 kHz, forward, 180 degrees   condition   delay(5000);   Lib.motorRotationFree(); // All outputs OFF }</pre>		

*motorRotationTime***Table 13. motorRotationTime**

API function	MotorRotationTime(float freq, float time, byte direction, byte excitation)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	float	freq	Frequency [Hz] 1 – 4800
	float	time	Drive time [sec] 1 – 65535 0: infinity (perpetual drive)
	byte	direction	Direction of rotation (0: clockwise, 1: counter clockwise)
	byte	excitation	Method of excitation 0: Full step, 1: Half step (Full-torque), 2: Half step (Smooth), 3: Quarter step
Return values	Type		Description
	Int		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Enable output (OE pin)</li> <li>• Rotates the motor for the amount of time specified by the parameter time while controlling the frequency, direction of rotation and method of excitation as specified in the parameters freq, direction and excitation. (Refer to <a href="#">Internal List Parameters</a> for initial values for each parameter.)</li> <li>• If motorRotationFree() (turn off all outputs) conducted just before, this function enables the output and then rotate the motor)</li> </ul>		
Example usage (sketch)	<pre> Lib_LV8702V Lib; // Lib_LV8702V class class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   <b>Lib.motorRotationTime(1000,5,0,0);</b> Rotate the motor in Fullstep, frequency 1 kHz, forward, 180 degrees condition   delay(5000);   Lib.motorRotationFree(); // All outputs off } </pre>		

*motorRotationStep***Table 14. motorRotationStep**

API function	motorRotationStep( <i>float</i> freq, <i>float</i> step, <i>byte</i> direction, <i>byte</i> excitation)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>float</i>	freq	Frequency [Hz] 1 – 4800
	<i>float</i>	step	Number of steps 1~16777215 0: infinity (perpetual drive)
	<i>byte</i>	direction	Direction of rotation (0: clockwise, 1: counter clockwise)
	<i>byte</i>	Excitation	Method of excitation 0: Full step, 1: Half step (Full-torque), 2: Half step (Smooth), 3: Quarter step
Return values	Type		Description
	<i>Int</i>		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Enable output (OE pin)</li> <li>• Rotates the motor by the number of steps specified by the parameter step, while controlling the frequency, direction of rotation and method of excitation as specified in the parameters freq, cwccw and exc. (Refer to <a href="#">Internal List Parameters</a> for initial values for each parameter.)</li> <li>• Stops the motor after rotating it by the specified number of steps (but maintains a state of excitation).</li> <li>• If the values for the parameters freq, step, cwccw or exc are outside the value range, the function returns Failure (1).</li> </ul>		
Example usage (sketch)	<pre> Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   <b>Lib.motorRotationStep(1000,100,0,0);</b> // Rotate the motor in Fullstep, frequency 1 kHz, forward, 180 degrees   condition   delay(5000);   Lib.motorRotationFree(); // All outputs off } </pre>		

*motorRotationStop***Table 15. motorRotationStop**

API function	MotorRotationStop()		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>void</i>	None	None
Return values	Type		Description
	<i>Void</i>		None
Processing outline	Stops the motor (but maintains torque in a state of excitation.)		
Example usage (sketch)	<pre> Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   Lib.motorRotationStep(1000,100,0,0); // Rotate the motor in Fullstep, frequency 1 kHz, forward, 180 degrees   condition   delay(5000);   <b>Lib.motorRotationStop();</b> // Stop the motor } </pre>		

*motorRotationFree***Table 16. motorRotationFree**

API function	motorRotationFree()		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>void</i>	None	None
Return values	Type		Description
	<i>Void</i>		None
Processing outline	Stops the motor (all outputs are switched off, and torque is lost.)		
Example usage (sketch)	<pre> Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   Lib.motorRotationStep(1000,100,0,0); // Rotate the motor in Fullstep, frequency 1 kHz, forward, 100 steps   condition   delay(5000);   Lib.motorRotationFree(); // All outputs off } </pre>		

*setEfficiency***Table 17. setEfficiency**

API function	setEfficiency( <i>byte</i> efficiency, <i>byte</i> driveMargin, <i>byte</i> boostup)		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	byte	Efficiency	High efficiency drive function 0: Normal, 1: High efficiency
	byte	driveMargin	Margin adjustment function 0: Small, 1: Middle, 2: High
	byte	boostup	Boost up adjustment function 0: Min, 1: Low, 2: High, 3: Max
Return values	Type		Description
	<i>Int</i>		0: "Success" 1: "Failure" (if the parameter values set are outside the value range)
Processing outline	<ul style="list-style-type: none"> <li>• Check Index value validity</li> <li>• Set the following functions (refer to Quick Start guide for more details)               <ol style="list-style-type: none"> <li>1. efficiency: High efficiency drive function (GAD pin) setting</li> <li>2. driveMargin: Margin adjustment function (GMG1, 2 pins) setting</li> <li>3. boostup: Boost up adjustment function (GST1, 2 pins)</li> </ol> </li> </ul>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization   Lib.setEfficiency(0,0,0) // high efficiency drive off, drive merge small, boost up min } void loop() { }</pre>		

*readAdc***Table 18. readAdc**

API function	readAdc()		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>void</i>	None	None
Return values	Type		Description
	<i>Int</i>		Analog voltage of the pin in integer (0x0~0x3FF), Return 0xFFFF if the input voltage is outside the value range.
Processing outline	<ul style="list-style-type: none"> <li>• Call analogRead for PWM_VREF pin and return the read value.</li> </ul>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   int value;   value = Lib.readAdc(); // Read VREF voltage }</pre>		

*readDriveStatus***Table 19. readDriveStatus**

API function	readDriveStatus()		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>void</i>	None	None
Return values	Type		Description
	<i>unsigned long</i>		bit31 – 20: DST1 error count[0 – 255] bit19 – 8: DST2 error count [0 – 255] bit7 – 5: Unused bit4: Step count error using MONI pin [0 – 1] 0: Normal 1: Error bit3 – 2: Unused bit1: DST1 value (real time) [0 – 1] bit0: DST2 value (real time) [0 – 1]
Processing outline	<ul style="list-style-type: none"> <li>Assign the drive status stored in each variable to each bit of the unsigned long variable (Values of DST1, DST2 and MONI should be read in certain timing. Use this function to know the true status instead of reading the values of these pins directly.).</li> <li>Return unsigned longvariable.</li> </ul>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_ LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   unsigned long value;   value = <b>Lib.readDriveStatus ();</b> // Read drive status }</pre>		

*clrDstCount***Table 20. clrDstCount**

API function	clrDstCount()		
Class	Lib_LV8702V		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>void</i>	None	None
Return values	Type		Description
	<i>Void</i>		None
Processing outline	<ul style="list-style-type: none"> <li>Clear DST1 error count, DST2 error count, and Stepcount error flag.</li> </ul>		
Example usage (sketch)	<pre>Lib_LV8702V Lib; // Lib_ LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   int value;   value = <b>Lib.clrDstCount ();</b> // Initialize the error count }</pre>		

*guiSerialRead***Table 21. guiSerialRead**

API function	guiSerialRead()		
Class	GuiSerialInterface		
Attribute	Public		
Parameters	Type	Variable	Description
	<i>void</i>	None	None
Return values	Type		Description
	<i>Void</i>		None
Processing outline	<ul style="list-style-type: none"> <li>• Gather the byte data sent by serial communication to make a Bytestream</li> <li>• Analyze the Bytestream to call the corresponding API function</li> </ul> <p>* Assumed to be called in a loop function of a sketch, can't be called by using serial communication.</p>		
Example usage (sketch)	<pre> Lib_LV8702V Lib; // Lib_ LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   <b>Lib. guiSerialRead ();</b> // Receive serial messages. } </pre>		



Table 22. guiSerialParse

API function	guiSerialParse(char *serialRecvStr)		
Class	GuiSerialInterface		
Attribute	Public		
Parameters	Type	Variable	Description
	char*	serialRecvStr	Pointer to the data recived from serial communication
Return values	Type		Description
	Int		0: "Success" 1: "Failure"
Processing outline	<ul style="list-style-type: none"> <li>Return Failure (1) if parameter is out of the valid range</li> <li>Analyse theserial comucition with the GUI</li> </ul>		
Example usage 1 (sketch)	<pre> Lib_LV8702V Lib; // Lib_LV8702V class declaration void setup() {   Lib.initLib(); // Initialization } void loop() {   Lib.guiSerialRead(); // Call guiSerialParse function in the guiSerialRead function } </pre>		
Example usage 2 (sketch)	<pre> // The user can use a customized serial interface by overriding guiSerialParse() //  //Create a derived class that inherits from the Lib_LV8702 class. // class Lib_LV8702_custom : public Lib_LV8702{ public:   virtual ~Lib_LV8702_custom() {}   int guiSerialParse(char *type) override; }; Lib_LV8702_custom Ex; // Instantiate inherited class  //Override guiSerialParse function. // int Lib_LV8702_custom::guiSerialParse(char *serialRecvStr){   // Implement for serial code execution.   switch (serialRecvStr[0]){     case 'a':     {       Serial.println("Command");       return SUCCESS; // Return Success (0).     }     default:     {       Serial.println("unknown command");     }   }   return FAILURE; // Return Failure (1). }  void setup() {   Serial.begin(19200); // Open a port at Baud rate 19200.   Ex.initLib(); // Initialization }  void loop() {   Ex.guiSerialRead(); // Call guiSerialParse function in the guiSerialRead function process } </pre>		

**SERIAL INTERFACE SPECIFICATIONS**

This section describes the serial interface for the USB connection between the computer and the Arduino Micro.

The stepper motor may be controlled using LV8702V by implementing the serial communication settings and the `guiSerialRead` function in a program on the Arduino side

(sketch) and by sending messages matched with individual APIs from the computer through serial communication.

For the method of implementing the `guiSerialRead` function, refer to [guiSerialRead](#).

**Table 23. LIST OF MESSAGES**

#	Command Value	Command Name	Command Description	Length	Chapter
1	0x03	getId	For acquiring API library identification ID.	1 byte	<a href="#">getId</a>
2	0x04	timeoutPol	Sent at uniform intervals for monitoring the serial connection.	1 byte	<a href="#">timeoutPol</a>
3	0x41	setChipEnable	For calling same name API as command to set the power setting.	2 byte	<a href="#">setChipEnable</a>
4	0x43	setReset	For calling to set the RESET setting.	2 byte	<a href="#">setReset</a>
5	0x44	setMaxCurrent	For calling same name API as command to set the output motor current limit.	9 byte	<a href="#">setMaxCurrent</a>
6	0x45	setRefVoltage	For calling same name API as command to set the output motor voltage.	3 byte	<a href="#">setRefVoltage</a>
7	0x46	setStepAngle	For calling same name API as command to set the step angle.	3 byte	<a href="#">setStepAngle</a>
8	0x51	motorRotationDeg	For calling same name API as command to control motor rotation by specifying the angle of rotation.	9 byte	<a href="#">motorRotationDeg</a>
9	0x52	motorRotationTime	For calling same name API as command to control motor rotation by specifying the rotation time.	7 byte	<a href="#">motorRotationTime</a>
10	0x53	motorRotationStep	For calling same name API as command to control motor rotation by specifying the step.	9 byte	<a href="#">motorRotationStep</a>
11	0x54	motorRotationStop	For calling same name API as command to stop the motor (while maintaining a state of excitation.)	1 byte	<a href="#">motorRotationStop</a>
12	0x55	motorRotationFree	For calling same name API as command to stop the motor (a turn off all outputs.)	1 byte	<a href="#">motorRotationFree</a>
13	0x61	setEfficiency	For calling same name API as command to set high efficiency drive settings	4 byte	<a href="#">setEfficiency</a>
14	0x64	readAdc	For calling same name API as command to read the ADC converted value of the VREF(A1) voltage.	2 byte	<a href="#">readAdc</a>
15	0x65	readDriveStatus	For calling same name API as command to read drive status.	1 byte	<a href="#">readDriveStatus</a>
16	0x68	clrDstCount	For calling same name API as command to clear error counts of DST1 and 2 and Step count error.	1 byte	<a href="#">clrDstCount</a>

**Details of Message Composition***getId*

- Command description

This is a command for acquiring library identification data.

Upon the receipt of this command, the API returns identification data that specifies the corresponding motor driver name, the API version and other details.

**Command from GUI to Motor Driver Kit**

Byte	0
Field	CMD
Value	0x03

## *timeoutPol*

- Command description

This command is for monitoring the state of the serial connection with the Arduino Micro.

The GUI sends this command every second to the Arduino Micro. If three seconds elapse without receiving data from serial communication (including this command), the API will call the timeoutPole function to automatically stop the motor for failsafe purposes.

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x04

## *setChipEnable*

- Command description

This command is for calling the setChipEnable function to set the operation status of the IC

The guiSerialParse function converts received parameters into function parameter (argument) format and calls the setChipEnable function.

For details of the setChipEnable function, refer to [setChipEnable](#).

Command from GUI to Motor Driver Kit

Byte	0	1
Field	CMD	SELECT
Value	0x41	0x00 – 0x01

Field      SELECT: Switch ChipEnable between standby mode and operation mode

## *setReset*

- Command description

This command is for calling the setChiopEnable function to set the RESET status.

The guiSerialParse function converts received parameters into function parameter (argument) format and calls the setReset function.

For details of the setReset function, refer to [setReset](#).

Command from GUI to Motor Driver Kit

Byte	0	1
Field	CMD	SELECT
Value	0x43	0x00 – 0x01

Field      SELECT: Switch setReset between ON and OFF

## LV8702VSLDGEVK

### *setMaxCurrent*

- Command description

This command is for calling the setMaxCurrent function to set the limit of output motor current.

The guiSerialParse function converts received parameters into function parameter (argument) format and calls the setMaxCurrent function.

For details of the setMaxCurrent function, refer to [setMaxCurrent](#).

#### Command from GUI to Motor Driver Kit

Byte	0	1 (low)	2 (high)	3 (low)	4 (high)	5 (low)	6 (high)	7 (low)	8 (high)
Field	CMD	ADPVOLTAGE		ADPCURRENT		MTRCURRENT		MTRRESISTANCE	
Value	0x44	0x005A – 0x0140		0x0000 – 0x0064		0x0001 – 0x0019		0x0001 – 0x1388	

Field      ADPVOLTAGE: Power Supply Voltage [V \* 10] (90 – 320)

The GUI multiplies the input power supply voltage (minimum unit 0.1) by 10 and transmits the resulting value as an integer to the Arduino Micro.

The guiSerialParse function converts the received ADPVOLT to power supply voltages (9 – 32 [V]) that can be treated as arguments to the setMaxCurrent function.

Field      ADPCURRENT: Max. power supply current [A \* 10] (0 – 100)

The GUI multiplies the input maximum power supply current (minimum unit 0.1) by 10 and transmits the resulting value as an integer to the Arduino Micro.

The guiSerialParse function converts the received ADPCRT to the maximum power supply current (0 – 10 [A]) that can be treated as arguments to the setMaxCurrent function.

Field      MTRCURRENT: Motor rated current [V \* 10] (1 – 25)

The GUI multiplies the input motor rated current (minimum unit 0.1) by 10 and transmits the resulting value as an integer to the Arduino Micro. The guiSerialParse function converts the received MTRCRT to a motor rated current (0.1 – 2.5 [A]) that can be treated as arguments to the setMaxCurrent function.

Field      MTRRESISTANCE: Motor winding resistance [ $\Omega$  \* 10] (1 – 5000)

The GUI multiplies the inputted motor winding resistance (minimum unit 0.1) by 10 and transmits the resulting value as an integer to the Arduino Micro. The guiSerialParse function converts the received MTRRST into motor-coil resistances (0.1 – 500 [ $\Omega$ ]) that can be treated as arguments of the setMaxCurrent function.

### *setRefVoltage*

- Command description

This command is for calling the setRefVoltage function to set the output motor voltage.

The guiSerialParse function converts received parameters into function parameter (argument) format and calls the setRefVoltage function.

For details of the setRefVoltage function, refer to [setRefVoltage](#).

#### Command from GUI to Motor Driver Kit

Byte	0	1 (low)	2 (high)
Field	CMD	VREF	
Value	0x45	0x0000 – 0x0110	

Field      VREF: Output voltage [V \* 100] (0 – 300)

The GUI multiplies the output voltage input as the argument (minimum unit 0.01) by 100 and transmits the output voltage as an integer to the Arduino Micro.

The guiSerialParse function reconverts the received VREF to output voltages (0.00 – 3.00 [V]) that can be treated as arguments to the setRefVoltage function.

*setStepAngle*

- Command description

This command is for calling the setStepAngle function to set the step angle.

The guiSerialParse function converts received parameters into function parameter (argument) format and calls the setStepAngle function.

For details of the setStepAngle function, refer to [setStepAngle](#).

Command from GUI to Motor Driver Kit

Byte	0	1 (low)	2 (high)
Field	CMD	STEPANGLE	
Value	0x46	0x0001 – 0x8CA0	

Field STEP ANGLE: step angle [degrees \* 100] (0 – 36000)

The GUI multiplies the input step angle (smallest unit 0.01) by 100, and sends it to the Arduino Micro as an integer value. The guiSerialParse function then reconverts the received STEP ANGLE into a step angle parameter (0.00 – 360.00 [Degrees]) that can be accepted by the setStepAngle function.

*motorRotationDeg*

- Command description

This command is for calling the motorRotationDeg function to rotate/drive the motor by the specified angle.

The guiSerialParse function converts received parameters into function parameter (argument) format and calls the motorRotationDeg function.

For details of the motorRotationDeg function, refer to [motorRotationDeg](#).

Command from GUI to Motor Driver Kit

Byte	0	1 (low)	2 (high)	3 (lowest)	4 (low)	5 (high)	6 (highest)	7	8
Field	CMD	FREQ		ANGLE				DIRECTION	EXCITATION
Value	0x51	0x0000 – 0x0960		0x00000000 – 0x63FFFF9C				0x00 – 0x1	0x00 – 0x03

Field FREQ: excitation signal frequency [Hz \* 10] (1 – 48000)

The GUI multiplies the input excitation signal frequency (smallest unit 0.1) by 10, and sends it to the Arduino Micro as an integer value. The guiSerialParse function then reconverts the received FREQ into an excitation signal frequency parameter (1 – 4800 [Hz]) that can be accepted by the motorRotationDeg function.

Field ANGLE: rotation/drive angle [Degrees \* 100] (0 – 1677721500)

The GUI multiplies the input rotation/drive angle (smallest unit 0) by 100, and sends it to the Arduino Micro as an integer value. The guiSerialParse function then reconverts the received ANGLE into an excitation signal frequency parameter (0 – 16777215 [degrees]) that can be accepted by the motorRotationDeg function.

Field DIRECTION: direction of rotation

Field EXCITATION: method of excitation

## LV8702VSLDGEVK

### *motorRotationTime*

- Command description

This command is for calling the motorRotationTime function to rotate/drive the motor for the specified period of time.

The guiSerialParse function converts received parameters into function parameter (argument) format and calls the motorRotationTime function.

For details of the motorRotationTime function, refer to [motorRotationTime](#).

#### Command from GUI to Motor Driver Kit

Byte	0	1 (low)	2 (high)	3 (low)	4 (high)	5	6
Field	CMD	FREQ		TIME		DIRECTION	EXCITATION
Value	0x52	0x0000 – 0x0960		0x0000 – 0x0FFFF		0x00 – 0x1	0x00 – 0x03

Field      FREQ: excitation signal frequency [Hz \* 10] (1 – 48000)

The GUI multiplies the input excitation signal frequency (smallest unit 0.1) by 10, and sends it to the Arduino Micro as an integer value. The guiSerialParse function then reconverts the received FREQ into an excitation signal frequency parameter (1 – 4800 [Hz]) that can be accepted by the motorRotationTime function.

Field      TIME: rotation/drive time [sec]

Field      DIRECTION: direction of rotation

Field      EXCITATION: method of excitation

### *motorRotationStep*

- Command description

This command is for calling the motorRotationStep function to rotate/drive the motor for the specified number of steps.

The guiSerialParse function converts received parameters into function parameter (argument) format and calls the motorRotationStep function.

For details of the motorRotationStep function, refer to [motorRotationStep](#).

#### Command from GUI to Motor Driver Kit

Byte	0	1 (low)	2 (high)	3 (lowest)	4 (low)	5 (high)	6 (highest)	7	8
Field	CMD	FREQ		STEP				DIRECTION	EXCITATION
Value	0x53	0x0000 – 0x0960		0x00000000 – 0x00FFFFFF				0x00 – 0x1	0x00 – 0x03

Field      FREQ: excitation signal frequency [Hz \* 10] (1 – 48000)

The GUI multiplies the input excitation signal frequency (smallest unit 0.1) by 10, and sends it to the Arduino Micro as an integer value. The guiSerialParse function then reconverts the received FREQ into an excitation signal frequency parameter (1 – 4800 [Hz]) that can be accepted by the motorRotationStep function.

Field      STEP: number of rotation/drive steps [step]

Field      DIRECTION: direction of rotation

Field      EXCITATION: method of excitation

### *motorRotationStop*

- Command description

This command is for calling the motorRotationStop function to stop the motor (while maintaining torque in a state of excitation.)

For details of the motorRotationStop function, refer to [motorRotationStop](#).

#### Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x54

*motorRotationFree*

- Command description

This command is for calling the motorRotationFree function to stop the motor (and turn of all outputs.)

For details of the motorRotationFree function, refer to [motorRotationFree](#).

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x55

*setEfficiency*

- Command description

This command calls setEfficiency function to set ON/OFF of the high efficiency drive function.

The guiSerialParse function converts the received parameters into arguments and call the setEfficiency function, and returns the calculated limit of the motor current to the transmitter along with the CMD 0x31.

See [setEfficiency](#) for more information on setEfficiency.

Command from GUI to Motor Driver Kit

Byte	0	1	2	3
Field	CMD	EFFICIENCY	DRIVEMARGIN	BOOSTUP
Value	0x61	0x00 – 0x01	0x00 – 0x02	0x00 – 0x03

Field      EFFICIENCY : High efficiency drive function

Field      DRIVEMARGIN: Margin-adjusting function

Field      BOOSTUP: Boost Up adjusting function

Command from Motor Driver Kit to GUI

Byte	0	1 (low)	2 (high)
Field	CMD	MAXCURRENT	
Value	0x31	0x0000 – 0x0110	

Field      MAXCURRENT: Upper limit of the output motor current [A \* 100]

*readAdc*

- Command description

This command is used to call readAdc function and measure the VREF (A1) voltage. guiSerialParse function call readAdc function and returns the read analog-voltage values to the transmitter along with the CMD 0x32.

See [readAdc](#) for more information on readAdc function.

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x64

Field      PIN: Read pin number

## LV8702VSLDGEVK

Command from Motor Driver Kit to GUI

Byte	0	1 (low)	2 (high)
Field	CMD	RECVADC	
Value	0x32	0x0000 – 0x03FF	

Field RECVADC: Analog Voltage Values (0 to 1023)

*readDriveStatus*

- Command description

This command is used to call readDriveStatus function and notify the initial excitation position.

See [readDriveStatus](#) for more information on readDriveStatus function.

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x66

*clrDstCount*

- Command description

This command is used to call clrDstCount function and notify the judged result of low-speed rotation.

See [clrDstCount](#) for more information on clrDstCount functions.

Command from GUI to Motor Driver Kit

Byte	0
Field	CMD
Value	0x67

### DETAILS OF TIMER RESISTOR SETTINGS

#### TCCR4B Setting

The TCCR4B is a 10-bit high speed counter/timer 4 register. Each bit contains the data shown in the table below.

bit	7 (MSB)	6	5	4	3	1	1	0 (LSB)
	PWM4X	PSR4	DTPS41	DTPS40	CS43	CS42	CS41	CS40
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value Arduino Micoro	0	0	0	0	0	1	1	1

For LV8702V, the PWM frequency used for the D6 pins is fixed at 31.373 [kHz]. Therefore, according to the following formula, it is necessary to set the frequency division ratio to “1” for the register.

[Phase Correct PWM frequency derivation]

$$(\text{Clock frequency} / \text{Division ratio}) \times (1 / (\text{TOP} \times 2)) [\text{Hz}]$$

$$= \{(16.0 \times 10^6 / 1) \times (1 / 510)\} / 10^3 = 31.373 [\text{kHz}]$$

Division ratio is determined by the combination of the 4 bits in CS(Clock Selector) of TCCR4B as shown in the table below. Set (CS43, CS42, CS41, CS40) = (0, 0, 0, 1) for LV8702V .



## LV8702VSLDGEVK

CS43	CS42	CS41	CS40	Division (Ratio)
0	0	0	0	Timer / Counter stop operation
0	0	0	1	1/1
0	0	1	0	1/2
0	0	1	1	1/4
0	1	0	0	1/8
0	1	0	1	1/16
0	1	1	0	1/32
0	1	1	1	1/64
1	0	0	0	1/128
1	0	0	1	1/256
1	0	1	0	1/512
1	0	1	1	1/1024
1	1	0	0	1/2048
1	1	0	1	1/4096
1	1	1	0	1/8192
1	1	1	1	1/16384

### OVERVIEW OF INTERNAL PARAMETERS, TABLES AND FUNCTIONS

#### Internal Parameters List

The table below provides a list of internal parameters.

**Table 24. LIST OF INTERNAL PARAMETERS**

#	Parameter	Initial Value	Description	Timing of Update
<b>PROTECTED</b>				
1	_isRotation	false	Motor rotation flag	Change using motorRotationDeg motorRotationTime motorRotationStep motorRotationStop motorRotationFree
2	_currentMax	2.72	Limit of output motor current	Change using setMaxCurrent
3	_stepAngle	STEP_ANGLE_MIN	Motor step angle	Change using setStepAngle
4	_reset	RESET_OFF	RESET status	Change using setReset
5	_direction	DIRECTION_CW	Direction of rotation	Change using motorRotationDeg motorRotationTime motorRotationStep
6	_excitation	EXCITATION_FULL	Method of excitation	Change using motorRotationDeg motorRotationTime motorRotationStep
7	_stepFreq	0	Step frequency	Change using motorRotationDeg motorRotationTime motorRotationStep

# LV8702VSLDGEVK

**Table 24. LIST OF INTERNAL PARAMETERS** (continued)

#	Parameter	Initial Value	Description	Timing of Update
<b>PROTECTED</b>				
8	_nowFreq	0	Current step frequency	Change using motorRotationDeg motorRotationTime motorRotationStep
9	_phaseFlag	0	Phase flag	Change using motorRotationFree
10	_targetStep	0	Target step angle	Change using motorRotationDeg motorRotationTime motorRotationStep
11	_nowStep	0	Current step number	Change using motorRotationDeg motorRotationTime motorRotationStep motorRotationFree timerFire
12	_dstVal1	HIGH	DST1 value (read in realtime)	Change using timerFire
13	_dstVal2	HIGH	DST2 value (read in realtime)	Change using timerFire
14	_dstCnt1	DST_COUNT_MIN	DST1 error count	Change using ClrDstCount timerFire
15	_dstCnt2	DST_COUNT_MIN	DST2 error count	Change using clrDstCount timerFire
16	_stepCnt	STEP_COUNT_INITIAL	Counter for generated step pulse	Change using timerFire
17	_stepCntLimit	STEP_COUNT_LIMIT_FULL	Maximum value for step pulse generation	Change using timerFire
18	_stepCntErr	STEP_COUNT_NON_ERROR	Step count error status	Change using clrDstCount timerFire
19	_stepCntErrMask	STEP_COUNT_ERROR_MASK	Count error state mask	Change using timerFire
20	_fallingEdge	true	Pulse falling edge flag	Change using motorRotationStep motorRotationStop motorRotationFree timerFire
<b>PRIVATE</b>				
21	CHIP_ENABLE_OFF	0	Power OFF	Constant value
22	CHIP_ENABLE_ON	1	Power ON	Constant value
23	OUTPUT_DISABLE	0	Output OFF	Constant value
24	OUTPUT_ENABLE	1	Output ON	Constant value
25	RESET_OFF	0	Reset OFF status	Constant value
26	RESET_ON	1	Reset ON status	Constant value
27	ADAPTER_VOLTAGE_MIN	9	Minimum value of supply voltage	Constant value
28	ADAPTER_VOLTAGE_MAX	32	Maximum value of supply voltage	Constant value

Table 24. LIST OF INTERNAL PARAMETERS (continued)

#	Parameter	Initial Value	Description	Timing of Update
<b>PRIVATE</b>				
29	ADAPTER_CURRET_MIN	0	Minimum value of supply current	Constant value
30	ADAPTER_CURRET_MAX	10	Maximum value of supply current	Constant value
31	MOTOR_CURRET_MIN	0.1	Minimum value of Motor rated current	Constant value
32	MOTOR_CURRET_MAX	2.5	Maximum value of Motor rated current	Constant value
33	MOTOR_RESISTANCE_MIN	0.1	Minimum value of Motor winding resistance	Constant value
34	MOTOR_RESISTANCE_MAX	500	Maximum value of Motor winding resistance	Constant value
35	VREF_MIN	0	Minimum value of output voltage	Constant value
36	VREF_MAX	3	Maximum value of output voltage	Constant value
37	REFERENCE_VREF	0.33	Output motor voltage for the reference motor	Constant value
38	STEP_ANGLE_MIN	0.01	Minimum value for number of steps	Constant value
39	STEP_ANGLE_MAX	360	Maximum value for number of steps	Constant value
40	FREQ_MIN	1	Minimum value of PWM frequency	Constant value
41	FREQ_MAX	4800	Maximum value of PWM frequency	Constant value
42	ANGLE_MIN	0	Minimum value of rotation angle	Constant value
43	ANGLE_MAX	16777215	Maximum value of rotation angle	Constant value
44	TIME_MIN	0	Minimum value of rotation time	Constant value
45	TIME_MAX	65535	Maximum value of rotation time	Constant value
46	STEP_MIN	0	Minimum value for number of steps	Constant value
47	STEP_MAX	16777215	Maximum value for number of steps	Constant value
48	DIRECTION_CW	0	Clockwise rotation	Constant value
49	DIRECTION_CCW	1	Counterclockwise rotation	Constant value
50	EXCITATION_FULL	0	Full step	Constant value
51	EXCITATION_HALF100	1	Half 100% step	Constant value
52	EXCITATION_HALF70	2	Half 70% step	Constant value
53	EXCITATION_QUARTER	3	1/4 step	Constant value
54	EFFICIENCY_NORMAL	0	Normal drive	Constant value
55	EFFICIENCY_HIGH	1	High efficiency drive	Constant value
56	DRIVE_MARGIN_S	0	DriveMargin SMALL	Constant value
57	DRIVE_MARGIN_M	1	DriveMargin MIDDLE	Constant value
58	DRIVE_MARGIN_L	2	DriveMargin LARGE	Constant value
59	BOOST_UP_MIN	0	BoostUp MIN	Constant value

Table 24. LIST OF INTERNAL PARAMETERS (continued)

#	Parameter	Initial Value	Description	Timing of Update
<b>PRIVATE</b>				
60	BOOST_UP_LOW	1	BoostUp LOW	Constant value
61	BOOST_UP_HIGH	2	BoostUp HIGH	Constant value
62	BOOST_UP_MAX	3	BoostUp MAX	Constant value
63	DST_COUNT_MIN	0	Minimum value for DST Count	Constant value
64	DST_COUNT_MAX	255	Maximum value of DST Count	Constant value
65	MONI_INITIAL	0	The Initial position for Excitation position	Constant value
66	MONI_NON_INITIAL	1	Excitation position other than the Initial position	Constant value
67	STEP_COUNT_INITIAL	0	Initial position for step count	Constant value
68	STEP_COUNT_LIMIT_FULL	4	Maximum value of step count in Full Step mode	Constant value
69	STEP_COUNT_LIMIT_HALF	8	Maximum value of step count in Half Step mode	Constant value
70	STEP_COUNT_LIMIT_QUARTER	16	Maximum value of step count in Quarter Step mode	Constant value
71	STEP_COUNT_NON_ERROR	0	No error in step count	Constant value
72	STEP_COUNT_ERROR	1	Error in step count	Constant value
73	STEP_COUNT_ERROR_UNMASK	0	Unmask flag for step count error	Constant value
63	STEP_COUNT_ERROR_MASK	2	Mask flag for step count error	Constant value
64	SRMES_GET_ID	0x03	getId API response serial message identifier	Constant value
65	SRMES_POLLING_ID	0x04	timeoutPol API serial message identifier	Constant value
66	SRMES_SET_CHIP_ENABLE	0x41	setChipEnable API serial message identifier	Constant value
67	SRMES_SET_RESET	0x43	setReset API serial message identifier	Constant value
68	SRMES_SET_MAX_CURRENT	0x44	setMaxCurrent API serial message identifier	Constant value
69	SRMES_SET_REF_VOLTAGE	0x45	setRefVoltage API serial message identifier	Constant value
70	SRMES_STEP_ANGLE	0x46	setStepAngle API serial message identifier	Constant value
71	SRMES_ROTATION_ANGLE	0x51	motorRotationDeg API serial message identifier	Constant value
72	SRMES_ROTATION_TIME	0x52	motorRotationTime API serial message identifier	Constant value
73	SRMES_ROTATION_STEP	0x53	motorRotationStep API serial message identifier	Constant value
74	SRMES_ROTATION_STOP	0x54	motorRotationStop API serial message identifier	Constant value
75	SRMES_ROTATION_FREE	0x55	motorRotationFree API serial message identifier	Constant value
76	SRMES_SET EFFICIENCY	0x61	setEfficiency API serial message identifier	Constant value

**Table 24. LIST OF INTERNAL PARAMETERS** (continued)

#	Parameter	Initial Value	Description	Timing of Update
<b>PRIVATE</b>				
77	SRMES_READ_ADC	0x64	readAdc API serial message identifier	Constant value
78	READ_DRIVESTATUS	0x65	readDriveStatus API serial message identifier	Constant value
79	SRMES_CLR_DST_COUNT	0x68	readSst API serial message identifier	Constant value
80	SRMES_RES_MAX_CURRENT	0x31	setMaxCurrent (for transmission) API serial message identifier	Constant value
81	SRMES_RES_READ_ADC	0x32	readAdc (for transmission) API serial message identifier	Constant value
82	SRMES_RES_READ_DRIVESTATUS	0x33	readMoni (for transmission) API serial message identifier	Constant value

**List of Internal Structures**

The table below provides a list of internal structures.

**Table 25. LIST OF INTERNAL STRUCTURES**

#	Structure	Description
1	SrMesDivSetChipEnable	Structure containing serial communication parameters for setChipEnable
2	SrMesDivSetReset	Structure containing serial communication parameters for setReset
3	SrMesDivSetMaxCurrent	Structure containing serial communication parameters for setMaxCurrent
4	SrMesDivSetRefVoltage	Structure containing serial communication parameters for setRefVoltage
5	SrMesDivSetStepAngle	Structure containing serial communication parameters for setStepAngle
6	SrMesDivRotationDeg	Structure containing serial communication parameters for motorRotationDeg
7	SrMesDivRotationTime	Structure containing serial communication parameters for motorRotationTime
8	SrMesDivRotationStep	Structure containing serial communication parameters for motorRotationStep
9	SrMesDivSetEfficiency	Structure containing serial communication parameters for setEfficiency
10	SrMesDivRecvMaxCurrent	Structure containing serial communication parameters for setMaxCurrent (transmission)
11	SrMesDivRecvReadAdc	Structure containing serial communication parameters for readAdc (transmission)
12	SrMesDivRecvReadDriveStatus	Structure containing serial communication parameters for readDriveStatus (transmission)

**List of Internal Functions**

The table below provides a list of internal functions.

**Table 26. LIST OF INTERNAL FUNCTIONS**

#	Function	Description	Calling Function
1	_setOutputEnable	Reflect Output Enable setting in IO10 pin.	motorRotationDeg motorRotationTime motorRotationStep motorRotationFree
2	_setExcitation	Reflect Excitation setting in IO9/IO8 pin	motorRotationDeg motorRotationTime motorRotationStep
3	_setDirection	Reflect Direction setting in D5 pin	motorRotationDeg motorRotationTime motorRotationStep

## HARDWARE SPECIFICATIONS

## OPERATING CONDITIONS

*Power Supply Voltage*

The LV8702V data sheet specifies the power voltage range of 9.0 V to 32 V as a part of the recommended operating conditions. This means that the IC will operate stably when a voltage within this range is applied to it. Check the specifications of the motor used before determining the voltage.

*If any motor with low winding resistance is used, an overcurrent may occur that may cause damage to the IC.*

In the case of using batteries as a power supply, it is presumed that the voltage will fall to 4 volts or below. In this event, the circuit inside the IC may become unstable and, with its low voltage protection feature, the IC will automatically stop operation. (Low Voltage Protection Function)

When the motor is in operation, the power supply voltage may be raised. The LV8702V has a rated maximum voltage of 36 V. *Even an instantaneous excess over this level can cause damage to the IC.*

(Refer to [Regenerative Current](#) in the next section)

*Maximum Current*

The electric current level that may flow through the metal lines between the output transistors in a motor driver IC and between the IC chip and IC pins is limited. Current beyond those limits may cause damage to the ICs. For this reason, the maximum output current (hereinafter referred to as “I<sub>max</sub>”) for motor driver ICs is specified on a model-by-model basis.

*For LV8702V, I<sub>max</sub> is specified at 2.5 A. It is necessary to pay attention to avoid any excess over this level.*

In addition, please note that I<sub>max</sub> does not guarantee the level of current that can always flow. IC operation may be stopped even if the output current is lower than the I<sub>max</sub> due to the temperature rise of the IC. (Refer to the next paragraph).

*Operating Temperature*

In semiconductor products, an excess over the rated maximum junction temperature (around 150°C) causes damage to ICs.

In other words, normal operation is anticipated at any temperature lower than the rated maximum junction

temperature. ICs, and especially motor driver ICs, have a high power consumption and generate heat on their own. *With their thermal shutdown feature, ICs automatically stop functioning if their internal temperature surpasses the rated maximum junction temperature due to the ambient temperature of the operating environment and the ICs' self-heating.*

Please also note that *the rated maximum levels, recommended operating conditions and electrical characteristics of ICs are specified under conditions with an ambient temperature (T<sub>a</sub>) of 25 °C.* (Refer to the datasheet of LV8702V)

## GLOSSARY

*Back Electromotive Voltage*

Motors not only convert electrical energy into mechanical energy, but also generate electricity from mechanical energy. The voltage thus generated is called *back electromotive voltage, back electromotive force or induced voltage.*

At the time of the startup of a motor, no back electromotive voltage is generated. An electric current called a rush current flows, induced by the voltage that applies to the motor and the motor winding resistance.

The rush current lasts only a short time in the event of motor startup, but it is the peak current for the motor drive. The strength of back electromotive force increases as the rotation speed of the motor increases, and the back electromotive force acts to cancel the applied voltage, which makes it difficult for the motor drive current to flow.

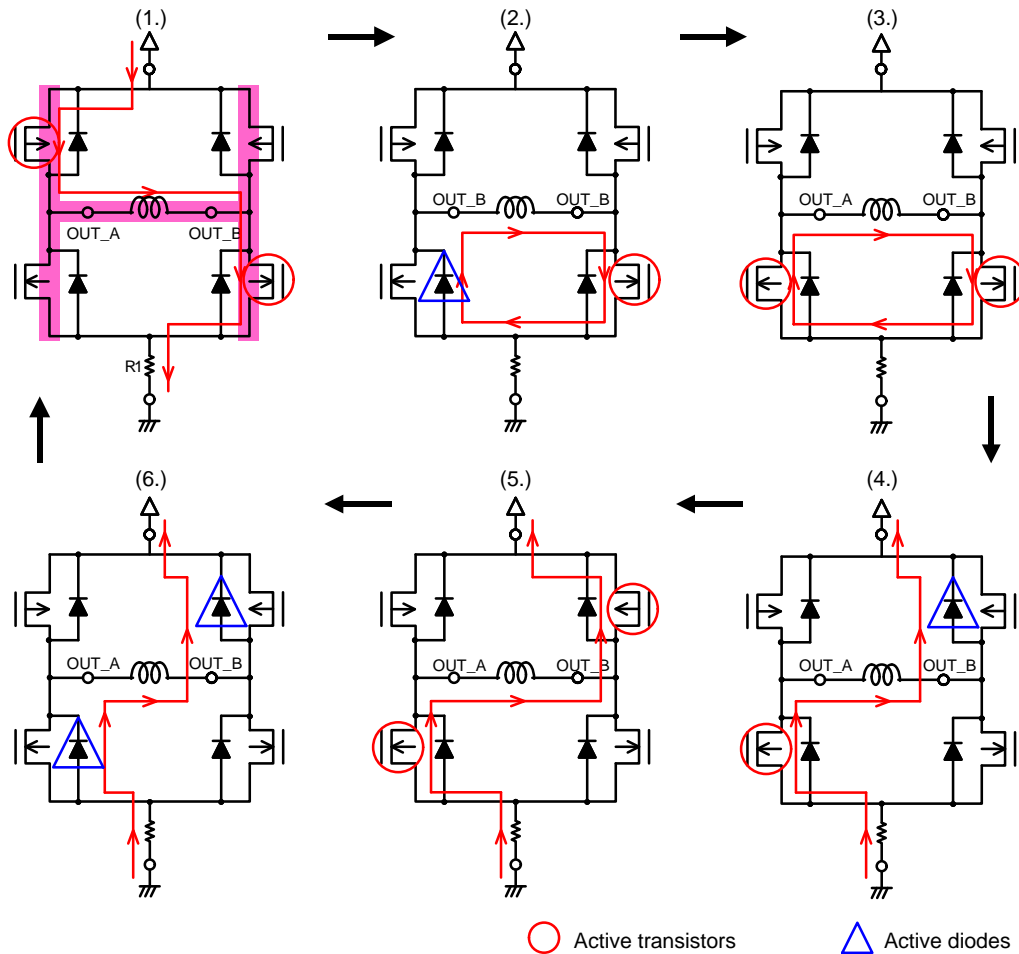
*H-bridge*

As shown in Fig. 8, transistors are connected inside the IC terminals connected to the motor connection terminals, namely OUT\_A, OUT\_B, OUT\_C, and OUT\_D, on the baseboard. This circuit is called H-bridge after the shape of the alphabetic character constituted by the transistor and motor coils.

*LV8702V incorporates two H-bridges.*

*Though some ICs (e.g. LV8548MC) can drive two DC motors with this circuit, LV8702V specializes to drive one bipolar type step motor.*

## LV8702VSLDGEVK



**Figure 8. Electric Current of Each H Bridge States of LV8702V**

### PWM Chopping

LV8702V controls motor current by PWM–current chopping. PWM stands for *Pulse Width Modulation*. PWM–current chopping is widely used to apply a pulsed voltage, control the rotational speed and torque of the motor, and drive the motor efficiently, instead of applying a constant voltage as when the battery is connected directly to the motor.

LV8702V switches ON/OFF of the H–bridge transistors to control the motor current by repeating transitions to 6 states (See Figure 8).

The duration of the cycle of (1.) to (6.) depends on the capacitance of the capacitor between the CHOP terminal and the ground, and is set to 15  $\mu$ S at the time of shipping.

1. The two circle–marked transistors turn on and current flows from the power supply in the direction  $\text{OUT\_A} \rightarrow \text{OUT\_B}$ .

(For reverse conduction, the lower transistor turns on on the OUT\_A side and the upper transistor turns on on the OUT\_B side.)

Since the motor coils store energy by this current, the condition (1.) is called charge. At this time, the motor current flows through R1 (hereinafter referred

to as “current detection resistor”) to GND, and LV8702V monitors the voltages generated by the motor current and the current detection resistor.

2. When the voltage applied to the current detecting resistor exceeds  $1/5$  of the VREF terminal input voltage, the upper transistor of OUT\_A is turned off to stop supplying current to the motor. Since the coil attempts to flow current in the same direction as (1.) using the stored energy, the current flows through the triangle–marked diode in the built–in IC in the path indicated by the arrow.  
If the state (2.) does not exist and the transition from (1.) to (3.) is made, if the upper transistor of OUT\_A is delayed to become off faster than the lower transistor becoming on, the power supply and GND will be short–circuited, resulting in a high current flow (through current). In order to prevent this, LV8702V has (2.) in a very short period.
3. The lower transistor of OUT\_A is turned on and the current loop of (2.) is maintained. At this time, the motor current gradually decreases because the current source is cut off. Because the rate of decline is slow, the condition (3.) is called Slow Decay.

4. The lower transistor of OUT\_B turns OFF about 1  $\mu$ S before charging again. Since the motor current continues to flow from OUT\_A to OUT\_B, the current flows through the triangle-marked diode in the IC in the loop indicated with the arrows (regenerative current). (4.) is provided to prevent the generation of a through current during the transition from (3.) to (5.).
5. Continued application of current to the diode may cause heat generation. To avoid this, two transistors are turned on to allow regenerative current to flow. (Synchronous rectification)  
The decay speed of the current at this time is faster than that in (3.) and is therefore referred to as Fast decay. If the motor current decreases and the energy

is lost, current may flow from the power supply through the two transistors in the opposite direction to the motor, but LV8702V also prevents such phenomenon.

6. During the transition from (5.) to (1.), in order to prevent the generation of the through current and to maintain the direction of the motor current flow, all transistors are turned off, and the regenerative current flows through the two triangle-marked diodes.

This method of controlling the motor current to a constant level by repeatedly increasing (charging) and decreasing (slow/fast decay) the motor current is called *PWM current chopping*.

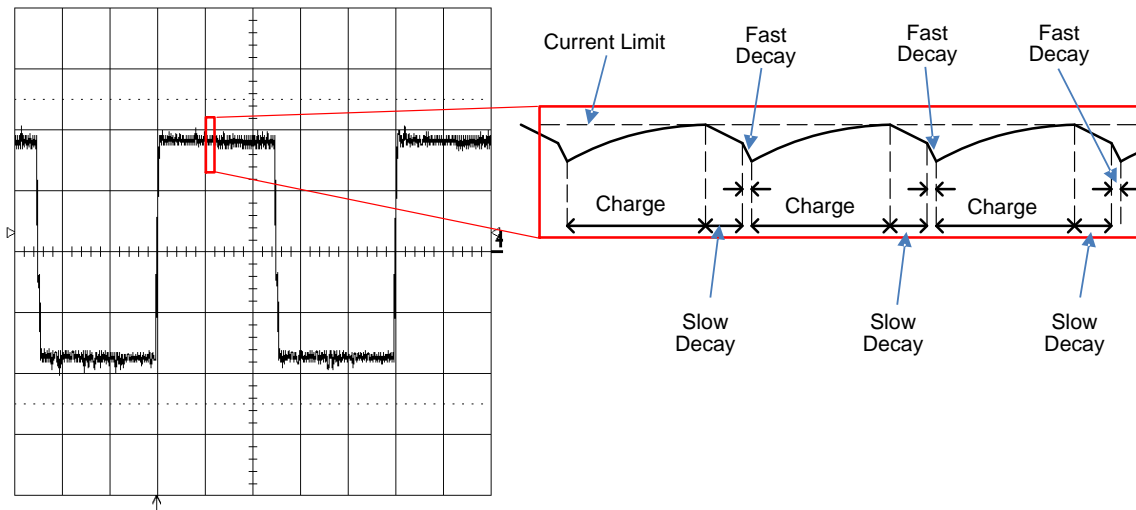


Figure 9. Output Current Waveform of PWM Current Chopping



### Regenerative Current

As described above, the energy stored in the motor generates regenerative current which flows into the power supply. If a stabilized power supply is used, this current can be absorbed to maintain the applied voltage. However, in the case of power supplies such as AC adapters and batteries, they cannot absorb the regenerative current, which causes a large increase in voltage.

This results in a sudden increase in the voltage applied to the IC, in some cases damaging or destroying the IC itself.

In the case of the LV8702V, the maximum rated power supply voltage is 36 V. Care is required to ensure that the applied voltage does not exceed this rated value. Mounted on the baseboard, a 100  $\mu$ F electrolytic capacitor works to suppress the voltage surge in this situation (refer to Fig. 10).

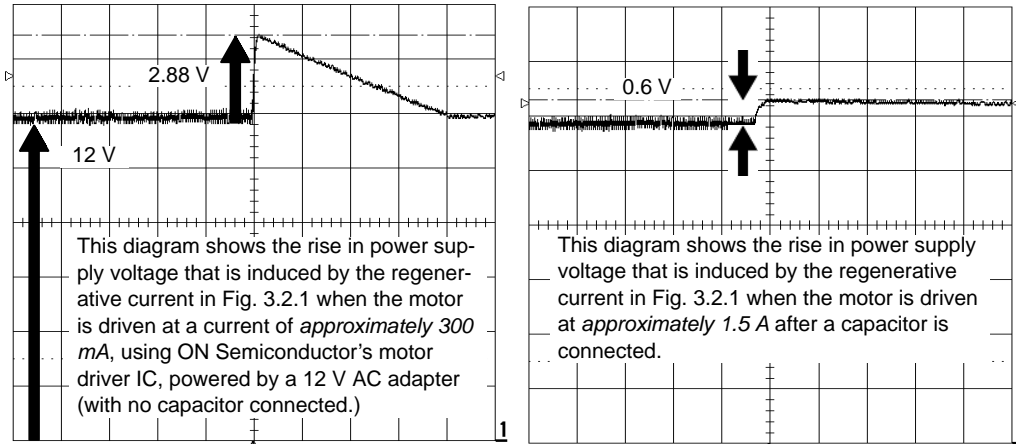


Figure 10. Rise in Power Supply Voltage by Regenerative Current and Effect of Electrolytic Capacitor

As a method of suppressing this voltage rise, there is a method of introducing a voltage clamp circuit as shown in Fig. 11. When Tr: FQP20N06, Di: 1N5240B (both ON semiconductor) and R1: 1.5 k $\Omega$  are used, the VCC voltage can be limited to less than 36 V by adjusting R2 to 270 to 470  $\Omega$ . When the VCC voltage exceeds the set value, Tr is turned on, and the current causing the voltage rise is pulled to GND, thereby suppressing the voltage rise further.

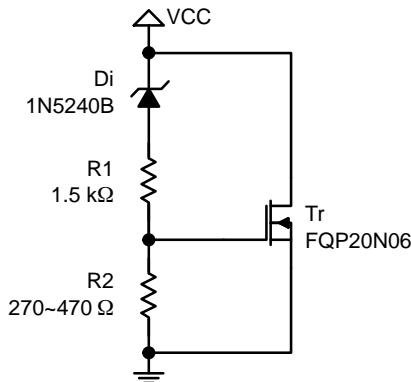


Figure 11. An Example of Power Supply Voltage Clamp Circuit

### Unipolar Type and Bipolar Type Stepper Motors

As shown below in Fig. 12 and 13, stepper motors are classified into two types according to their internal structures and corresponding differences in drive mechanism.

	Direction of Current Flow	Number of Wires
Unipolar	Unidirectional	6 or 5
Bipolar	Bidirectional	4

ON Semiconductor's monolithic LSI stepper motor driver ICs (including LV8702V) are designed for bipolar types (two-phase) motors.

While it is also possible to use them to drive unipolar motors, there are some cases in which differences in motor performance may arise in comparison with dedicated driver ICs. Verification and evaluation are therefore required. (Refer to Fig. 14).

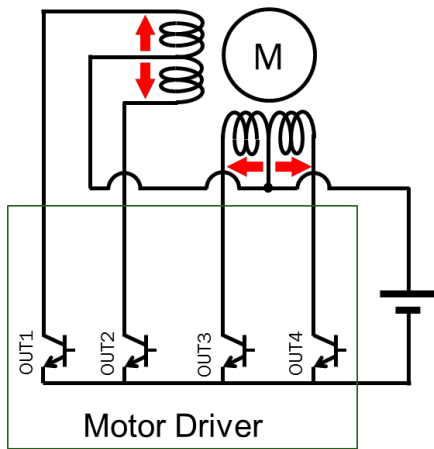


Figure 12. Structure & Drive Method for a Unipolar Motor

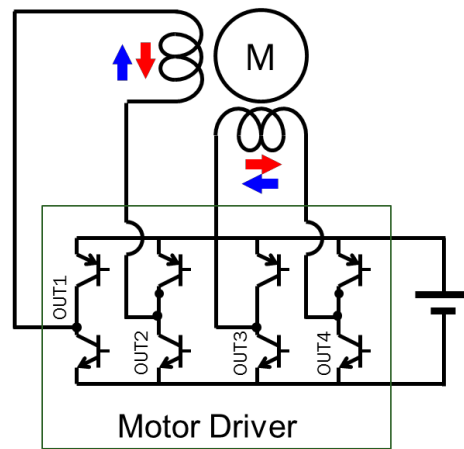
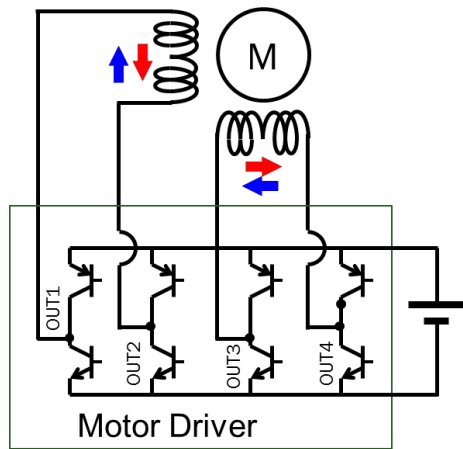
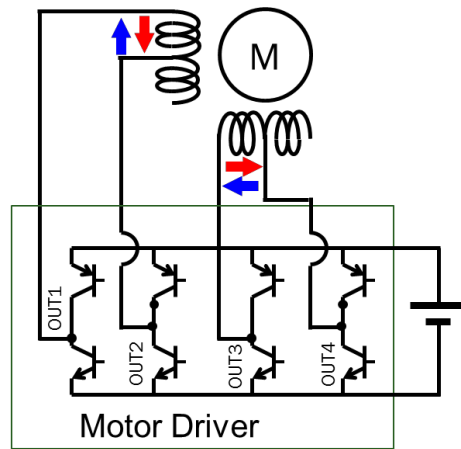


Figure 13. Structure & Drive Method for a Bipolar Motor



Connection example 1



Connection example 2

Figure 14. Connection Methods for Bipolar Drive of a Unipolar Motor

### Step Angle

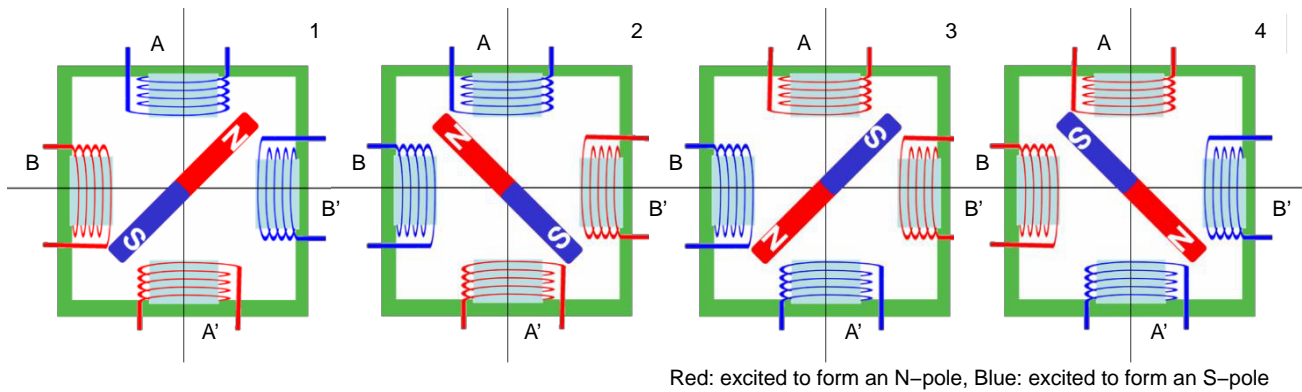
Unlike DC brush motors, it is possible to control the rotor position of a stepper motor even without sensors. This is because motors have characteristic step angles.

This step angle can be understood as “the angle by which the motor (or rotor) rotates per step when driven in Full step mode.” It is therefore possible to ascertain the rotation angle by managing the number of steps that corresponds to the signal entered into a motor driver IC.

The structure of a stepper motor can be explained simply as consisting of orthogonally arranged two sets (two phases)

of coils positioned facing one another, and a bar magnet in the middle that represents the rotor. (Fig. 15, and 16).

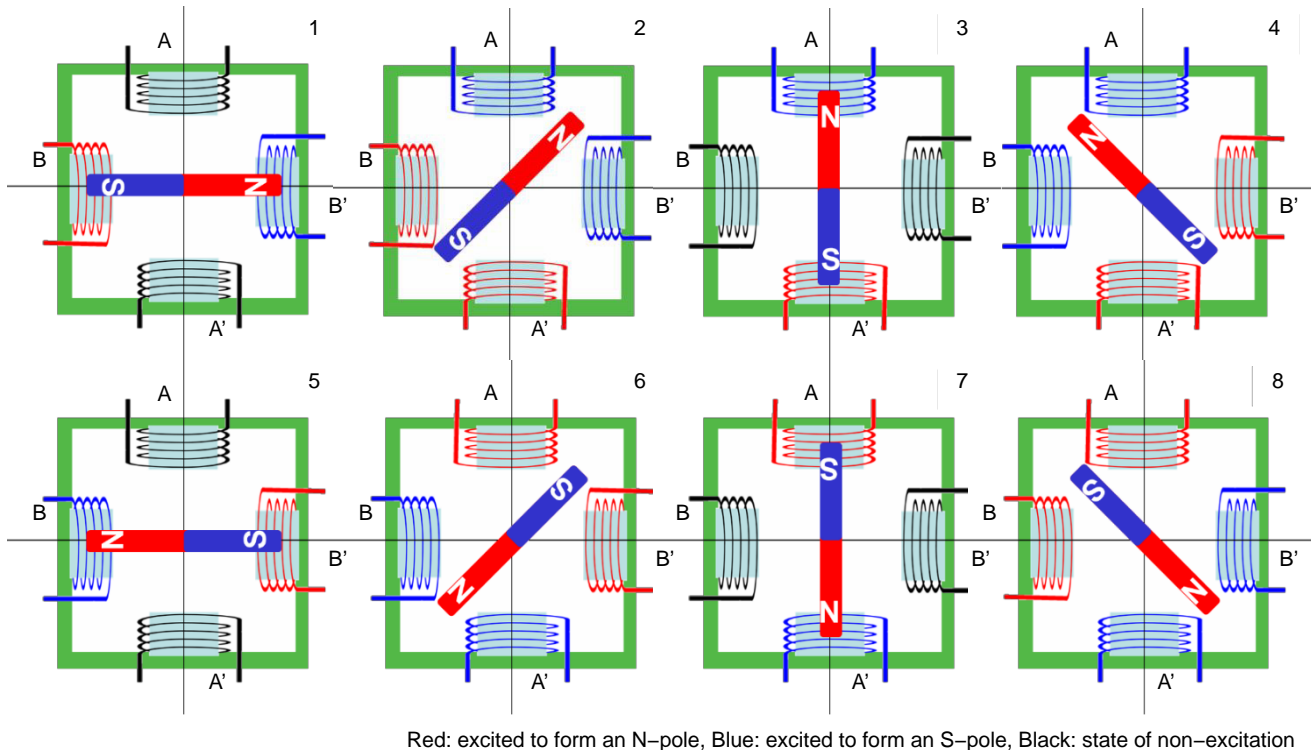
In Full step mode, both sets of coils are constantly excited (have electric current flowing through them). The bar magnet therefore stabilizes at positions where the neighboring coils attract the relevant poles. The direction of current flowing in the two pairs of coils is switched continuously such that the bar magnet rotates. For this reason, Full step method is also called 2-phase excitation method (Refer to Fig. 15).



**Figure 15. Full Step Change in Electrical Angle in Full Step**

For motors running in Half step, the amount of movement per step is half that of the one for Full step. This is because the timing for passing current through one set of coils

(1-phase excitation) and 2-phase excitation are alternated in the case of Half step. For this reason, Half step motors are also called 1 – 2 phase excitation motors. (refer to Fig. 15).



**Figure 16. Half Step Change in Electrical Angle in Full Step**

While Full step operation takes four positions, Half step takes eight.

However, the angle represented in these four or eight positions is electrical angle, which differs from step angle.

As mentioned earlier, step angle is “the angle by which the motor (or rotor) rotates per step when driven in Full step mode.” One step when driven in Full step mode corresponds to an electrical angle of 90 degrees.

Therefore, please understand that step angle = electrical angle of 90 degrees.

### Micro-Step

The Full step and Half step methods controls the position of the rotor according to the excited coils and the direction of excitations. In addition, by controlling the energizing current ratio, differences are generated between the forces of the two coils to attract the rotors, enabling fine position control in units of 1/4, 1/8, 1/16, and smaller portions of the step angle. This is a control method called micro-step.

In Fig. 17,  $\theta_2'$  indicates the two-phase excitation position in Full step and normal Half step, and  $\theta_0$  and  $\theta_4$  indicate the position of the one-phase excitation of Coil A and Coil B.

When the current ratio is changed so that the combined vector of the two coil currents follows the arc, the motor rotates smoothly.

In addition to Full step and Half step (Full-torque), LV8702V can be controlled by the Half step (Smooth) where the combined vector during two-phase excitation is placed on the arc and the Quarter step.

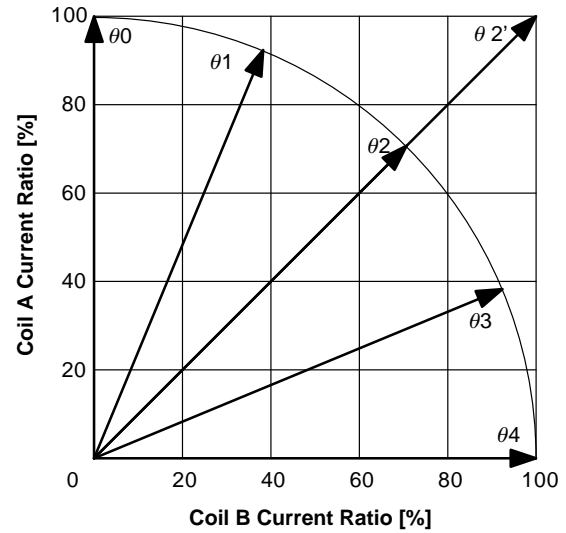


Figure 17. The Arc of the Output Current Vector (Step Angle is Normalized to 90 Degrees)

Table 27. CURRENT RATIO FOR EACH EXCITATION METHODS

	Full Step		Half Step (Full-torque)		Half Step (Smooth)		Quarter Step	
	Coil A	Coil B	Coil A	Coil B	Coil A	Coil B	Coil A	Coil B
$\theta_0$	—	—	100%	0%	100%	0%	100%	0%
$\theta_1$	—	—	—	—	—	—	92%	38%
$\theta_2$	100%	100%	100%	100%	70%	70%	70%	70%
$\theta_3$	—	—	—	—	—	—	38%	92%
$\theta_4$	—	—	0%	100%	0%	100%	0%	100%

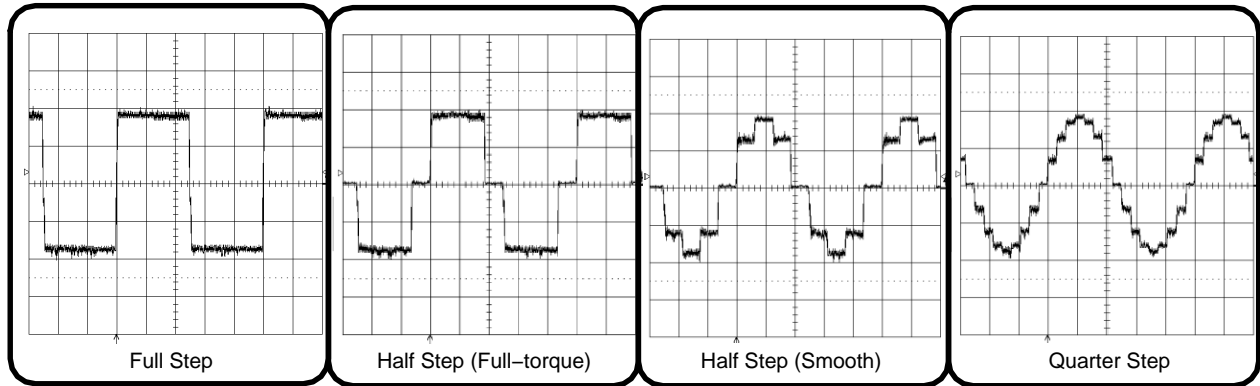


Figure 18. Output Current Waveform (OUT\_A)

The formula for calculating the time taken for a rotation using a step angles, an excitation method, a motor speed and a transfer unit is expressed as follows.

These can be used as parameters (arguments) for the delay function.

Full step	1
Half step (Full torque)	1/2
Half step (Smooth)	1/2
Quarter step	1/4

The algebraic arithmetic expression is an example for a rotation time of 10000 ms (i.e. 10 seconds).

Condition 1)

- Motor Speed Unit: [step/s]
- Transfer Unit: [Degrees]

Rotation Time [ms]

= Transfer Angle [Degrees] ÷ (Step Angle [Degrees] ÷ Excitation Method x Motor Speed [step/s]) x 1000

750 [Degrees] ÷ (7.5 [Degrees] ÷ 1 x 10 [step/s]) x 1000

Condition 2)

- Motor Speed Unit: [rpm]
- Transfer Unit: [Degrees]

Rotation Time [ms]

= Transfer Angle [Degree] ÷ (Motor Speed [rpm] x 360 [°]) x 60 [sec] x 1000

1200 [Degree] ÷ (20 [rpm] x 360 [°]) x 60 [sec] x 1000

Condition 3)

- Motor Speed Unit: [step/s]
- Transfer Unit: [Steps]

Rotation Time [ms]

= Transfer Step [Step] ÷ Motor Speed [step/s] x 1000

500 [Step] ÷ 50 [step/s] x 1000

Condition 4)

- Motor Speed Unit: [rpm]
- Transfer Unit: [Steps]

Rotation Time [ms]

= Transfer Step [Step] x Step Angle [Degrees] ÷ Excitation Method ÷ (Motor Speed [rpm] x 360 [°]) x 60 [sec] x 1000

800 [Step] x 7.5 [Degrees] ÷ 1 ÷ (100 [rpm] x 360 [°]) x 60 [sec] x 1000

### Stall

As mentioned earlier, a stepper motor rotates by transitioning its rotor in synchronization with the state of excitations of its coils. If the transition of the state of excitations operated by control signals is too fast, and the load placed on a motor is too great, the rotor becomes unable to keep up with the signal, and the motor may vibrate or stop moving. This phenomenon is called “stall”.

The maximum speed of rotation and maximum load that can be placed on the motor differs depending on the motor used.

Since these attributes are also dependent on the drive current applied to the motor, in the case of LV8702V, these

attributes may change due to adjustments in power supply voltage. In addition, motor current can cause stall due to the torque shortage of a motor with respect to the load.

LV8702V uses the high efficiency drive function to optimize the motor current by obtaining data from the rotational status of the motor. When no torque is required, such as when no load is applied, the current is suppressed, and when torque is required, a current necessary for preventing step-out is supplied.

However, this does not mean that this function can prevent stall in any case, since no more current than the set value or the maximum value of current supply capability of the power supply is allowed to flow.

## POINTS TO NOTE IN CIRCUIT BOARD LAYOUTS

A circuit board with motor driver ICs must have its layout devised in consideration of the electric current, heat generation due to power consumption, and noise caused by motor operation. Without a properly constructed layout, the expected performance may not be obtained.

While LV8702V is capable of sensitive motor control, it itself can be a source of noise due to PWM-driven operation.

1. *Be sure to put a capacitor between VCC (power supply) and the ground (GND).*

For the purpose of driving a motor, an electric current is supplied from the power supply to the motor. This will cause an instantaneous drop in the power supply voltage. If the PWM control used, this phenomenon occurs often and possibly destabilize the operation of the internal circuit in the IC. A capacitor is placed for the purpose of suppressing this voltage drop.

It is desirable to place the capacitor as close as possible to the IC.

2. *Use thick, short lines for VCC, GND and OUT connection.*

On these three different kinds of lines, high electric currents flow. A fine or long line has resistance that causes heat generation and a voltage drop.

LV8702V motor driver module is designed with a four-layer circuit board, the 2<sup>nd</sup> layer is for GND pattern and the 3<sup>rd</sup> layer is for the power supply pattern and a part of the signal lines. A large number of through holes are provided to connect the same lines in different layers, thereby reducing electrical resistance.

3. *Draw ground wiring for the control circuit system and ground wiring for the power system separately.*

LV8702V has two different types of GND pins. The GND pins of the control circuit inside the IC are SGND, and GND pins through which large currents flow are PGND1 and PGND2. Contrary to the power voltage drop discussed in 1, when the motor drive current flows through the GND, the GND level rises due to the relationship between the GND's interconnection resistance and the electric current. The exposure of the control circuit to this impact leads to a change in the GND level as a reference for control and causes malfunction. To avoid this, it is

ideal to draw the wiring for SGND and the wiring for PGND separately if possible and to make them interconnect solely at the GND position with the capacitor between the VCC and the GND.

The 2<sup>nd</sup> layer of LV8702V motor driver module is the GND layer, where the PGND area (upper part) and the SGND areas (lower part) are completely separated. The connecting point between the SGND and the PGND is indicated by a red circle on the IC-mounting surface in Fig 19. Arduino GNDs are connected to the SGND area on the second layer.

4. *Keep the lines which can be noise sources away from other line.*

Since LV8702V drives motors by PWM-current chopping, the four output lines can be the source of noise. For control signal lines, attention should be paid to the STEP lines through which the pulsed signal continues to be input during motor operation. If these lines run in parallel with other signal lines, noise may jump into the lines and cause malfunction. Care must be taken if lines run across the area which is right under the noise source on the layer which is just above the layer the lines belong to. For example, because four output lines are on the IC mounting surface, signal lines must not run across the area on the 2<sup>nd</sup> layer which is right under the output lines. Conversely, the lines to which the current detection resistors is connected needs to avoid the effect of noise. If noise is applied to this line, the current cannot be controlled correctly and malfunction of the motor or abnormal driving noise may occur.

5. *Soldering the metal surface of the IC to the board for heat dissipation.*

The back surface of LV8702V is partially plastic-free, and the metal surface is exposed. A chip is mounted on this metal inside the IC, and heat generated by the chip is transferred to this metal while driving a motor. By mounting this metal surface on the board, heat dissipation capability of the IC and the chip is improved. This mounting area must be connected to GND, and any line other than GND must not be laid out on any layer under this area.

# LV8702VSLDGEVK

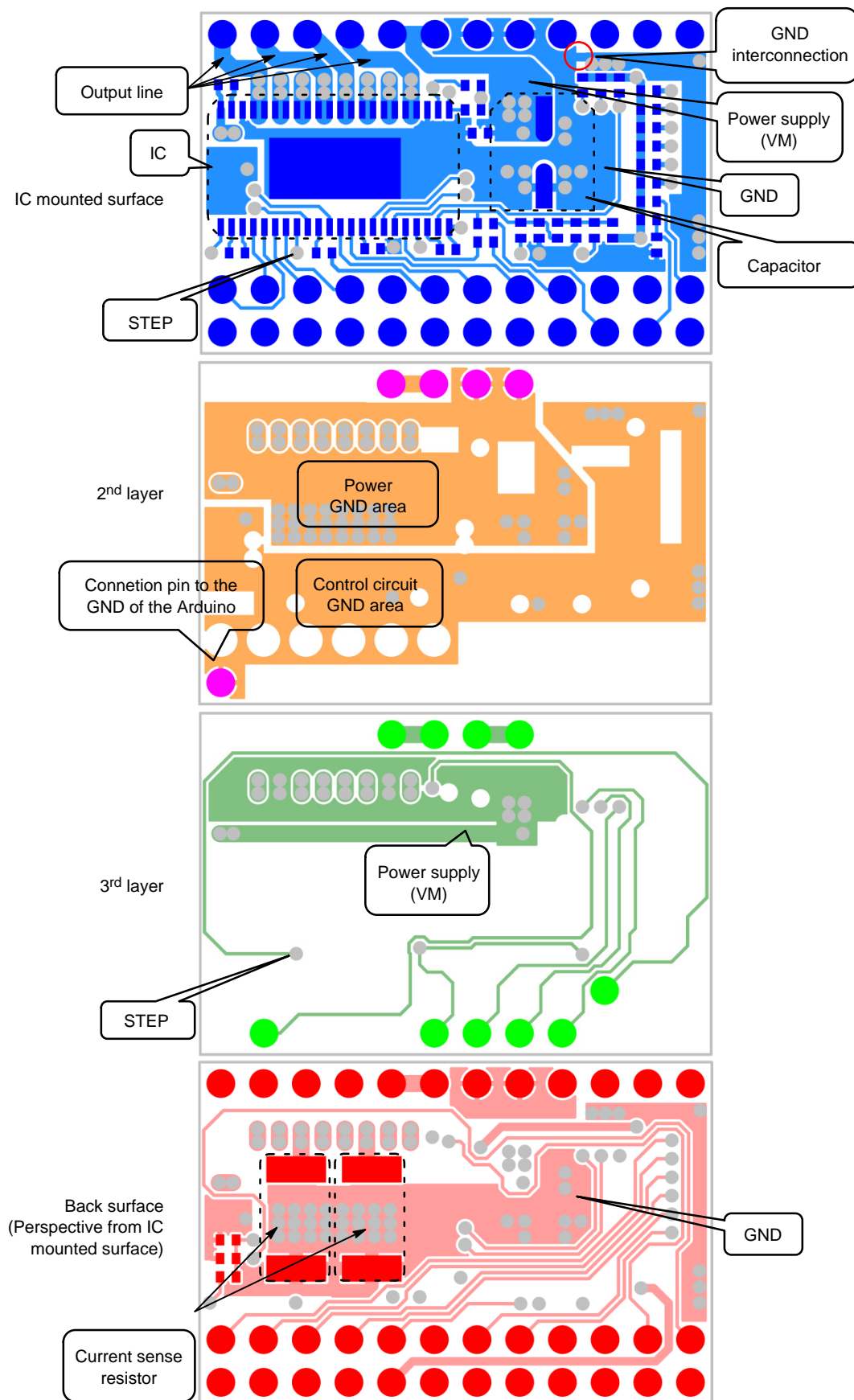


Figure 19. Points to Note in Circuit Board Layouts



# LV8702VSLDGEVK

## BOARD SCHEMATICS

### Motor Driver Module (LV8702VSLDGEVB)

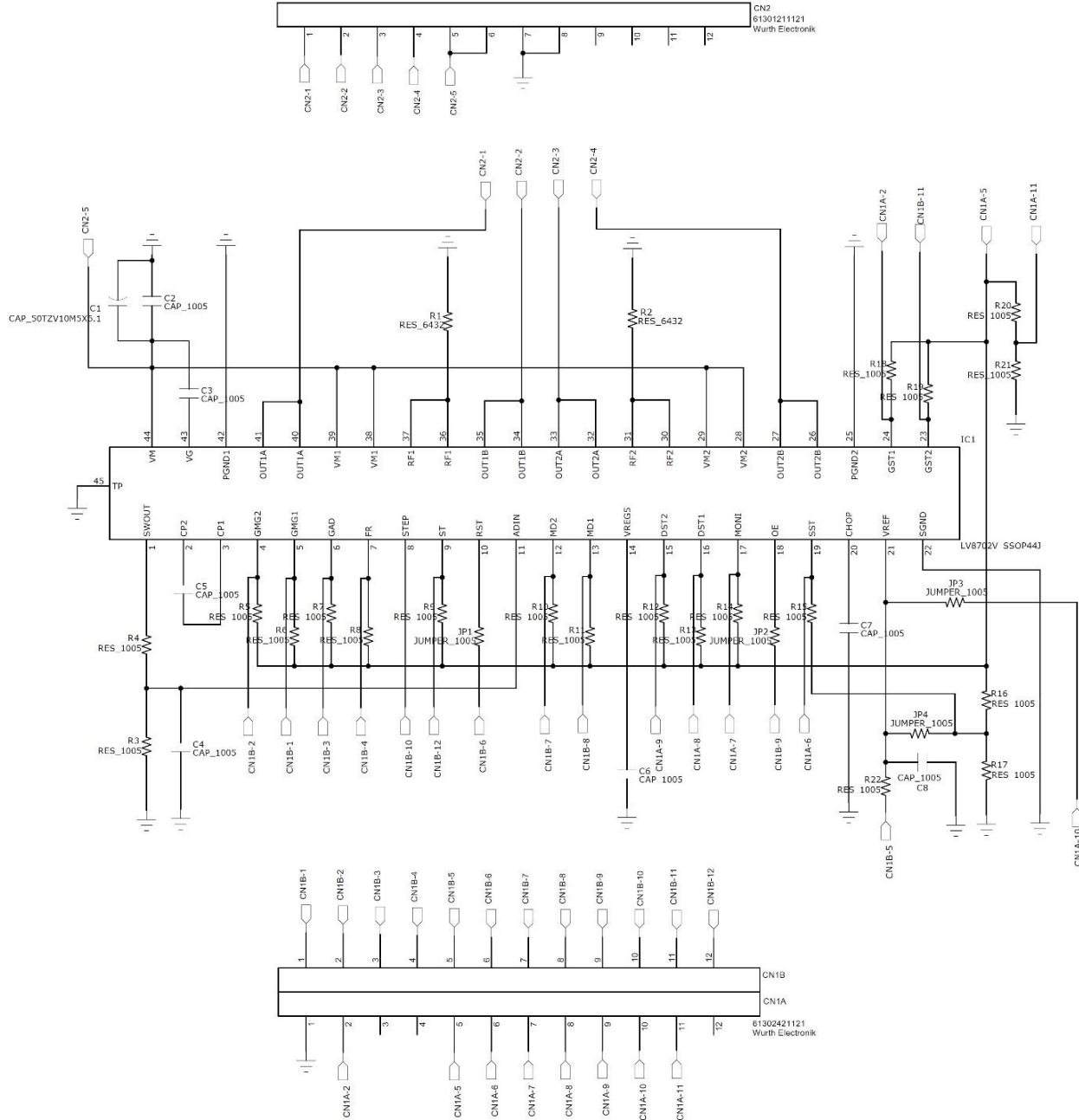


Figure 20. LV8702VSLDGEVB Schematics

- R5 – 11, R18, and R19 are lands for pull-up resistors for input terminals for setting functions.

Connect resistors when you don't need to change function settings and want to keep inputs High. Refer to the bill of materials for the recommended specifications of the resistors. These input terminals are connected to pull-down resistors inside the IC. Therefore, if you want

to keep inputs Low, you do not need to customize the PCB.

- The output motor current is determined by the DC input voltage to the VREF terminal. Since Arduino Micro does not have variable voltage DC output pins, the D6 pin is used to output a pulse signal of 0 – 5 V, and the pulse signal is converted into a variable DC voltage by R22 and C8.



## LV8702VSLDGEVK

- To set the output motor current to a fixed value, use resistors for R16 and R17 and a jumper (0  $\Omega$  resistor) for JP4 to determine the VREF voltage. The VREF voltage and the output motor current are calculated by the following equation.

The resistance values of R16 and R17 are recommended to be several tens of k $\Omega$ .

$$V_{REF} = \frac{R_{17}}{R_{15} + R_{17}} \times 5 \text{ V} \quad (\text{eq. 1})$$

$$\text{Output Current} = V_{REF} \div 5 \div 0.22 \Omega \quad (\text{eq. 2})$$

Note that the VREF voltage input range for LV8702V is specified to 0 to 3 V.

- When the motor is stopped, the SST terminal automatically operates to reduce the motor current. If the STEP signal is not switched for a certain period (13 to 23 ms), LV8702V judges that the motor has stopped operation and the SST terminal becomes Low level. If R15 – 17 and JP4 are connected in this situation, the

VREF voltage will be Low and the motor current will be reduced.

This function is undesirable for rotation at low speeds. To disable this function remove R15. Though R15 is installed at the time of shipment, this function is disabled because R16, R17, and JP4 are not connected.

SST terminal of LV8702V is connected to the A5 pin so that it can be monitored from the Arduino, but the API and the GUI do not support monitoring of SST signals.

If you do not need to monitor the SST from the Arduino, you can use the connector CN–8 on the baseboard to arbitrarily use the A5 terminal after disconnecting the SST terminal from A5 pins.

You can simply cut the pin (CN1A – 6: Refer to Fig. 20) of the pinheader on the motor driver module to achieve this disconnection.

- If you use the GUI, do not connect anything to R21. Any installed resistor prevents the GUI from identifying the type of the connected module.

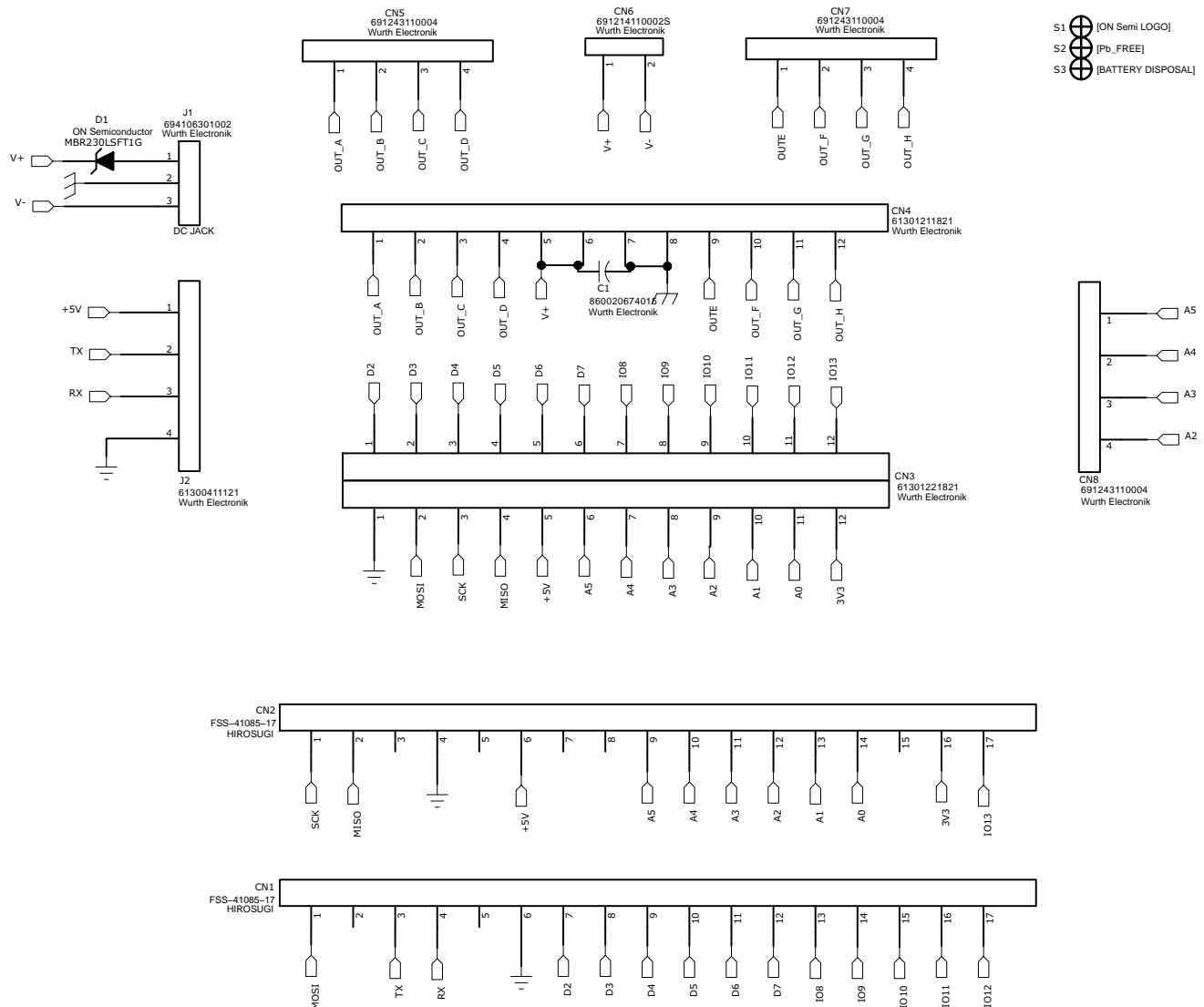
**Table 28. BILL OF MATERIALS FOR LV8702V MOTOR DRIVER MODULE (LV8702VSLDGEVB)**

Comp. Code	Qty	Name	Value	Tolerance	Size	Manufacturer	Product
IC1	1	Motor driver	–	–	SSOP44J	ON Semiconductor	LV8702V
R1–2	2	Chip resistor	0.22 $\Omega$ , 1 W	$\pm 5\%$	6432 (2512 Inch)	KOA	SR73W3AT**R22J
R3	1	Chip resistor	15 k $\Omega$ , 0.1 W	$\pm 5\%$	1005 (0402 Inch)	KOA	RK73B1ET**153J
R4	1	Chip resistor	100 k $\Omega$ , 0.1 W	$\pm 5\%$	1005 (0402 Inch)	KOA	RK73B1ET**104J
R5–11, R18–19	9	Chip resistor	47 k $\Omega$ , 0.1 W	$\pm 5\%$	1005 (0402 Inch)	KOA	RK73B1ET**473J
R12–14, R16	4	Chip resistor	47 k $\Omega$ , 0.1 W	$\pm 5\%$	1005 (0402 Inch)	KOA	RK73B1ET**473J
R15	1	Chip resistor	12 k $\Omega$ , 0.1 W	$\pm 5\%$	1005 (0402 Inch)	KOA	RK73B1ET**123J
R17	1	Chip resistor	15 k $\Omega$ , 0.1 W	$\pm 5\%$	1005 (0402 Inch)	KOA	RK73B1ET**153J
R20	1	Chip resistor	100 k $\Omega$ , 0.1 W	$\pm 5\%$	1005 (0402 Inch)	KOA	RK73B1ET**104J
(R21)							
R22	1	Chip resistor	27 k $\Omega$ , 0.1 W	$\pm 5\%$	1005 (0402 Inch)	KOA	RK73B1ET**273J
JP1–3	3	Jumper	0 $\Omega$ , 1 W	$\pm 20\%$	1005 (0402 Inch)	KOA	RK73Z1ET**
JP4	1	Jumper	0 $\Omega$ , 1 W	$\pm 20\%$	1005 (0402 Inch)	KOA	RK73Z1ET**
C1	1	Electrolytic capacitor	10 $\mu$ F, 50 V	$\pm 20\%$	5 x 5.5	Würth Elektronik	865080642006
C2, C3, C5, C6, C8	5	Chip capacitor	0.1 $\mu$ F, 100 V	$\pm 10\%$	1005 (0402 Inch)	Murata Manufacturing	GRM155R62A104KE14D
C4	1	Chip capacitor	1000 pF, 50 V	$\pm 5\%$	1005 (0402 Inch)	Murata Manufacturing	GRM1555C1H102JA01J
C7	1	Chip capacitor	150 pF, 50 V	$\pm 10\%$	1005 (0402 Inch)	Murata Manufacturing	GRM1555C1H151JA01J
CN1A, 1B	1	Pin header	12 pins x 2	–	30.48 x 5.08	Würth Elektronik	61302421121
CN2	1	Pin header	12 pins	–	30.48 x 2.54	Würth Elektronik	61301211121
PCB	1	PCB	–		30.48 x 20.32		

NOTE: Parts highlighted in yellow are not mounted at the time of product shipment.

**LV8702VSLDGEVK**

## Base Board (ONBB4AMGEVB)



### Figure 21. ONBB4AMGEVB Schematics

- The pins with the same names in the schematics are connected by wiring.
  - When the connector is inserted into the DC jack, the V- of CN6 becomes open.  
Therefore, even if another power supply is connected to the CN6, it does not cause a failure.  
At the same time, the voltage input from CN6 is disabled.  
You cannot access the GND level from V-.  
Please refer to the manufacturer's site for details.
- [https://katalog.we-online.de/en/em/DC\\_RIGHT\\_ANG\\_LED\\_6\\_4\\_69410X301002?sid=c33324d235](https://katalog.we-online.de/en/em/DC_RIGHT_ANG_LED_6_4_69410X301002?sid=c33324d235)
- The power for the Arduino is supplied from the USB-connector.  
Power from the the USB-connector can also be supplied from a smartphone charger during stand-alone operation.
  - Arduino GND, J2 GND and GND of DC jack or V- of CN6 are connected by inserting the motor driver module.

# LV8702VSLDGEVK


**Table 29. BILL OF MATERIALS FOR THE BASE BOARD ONBB4AMGEVB**

Comp. Code	Qty.	Name	Value	Tolerance	Size	Manufacturer	Product
D1	1	Diode	–	–	SOD123	ON Semiconductor	MBR230LSFT1G
CN1, 2	2	Connectors for the Arduino Micro	–	–	Φ1.02 x 17 – 2.54 pitch	HIROSUGI	FSS–41085–17
CN3	1	Connectors for the module	–	–	Φ1.02 x 12 x 2 lines – 2.54 pitch	Würth Elektronik	61302421821
CN4	1	Connectors for the module	–	–	Φ1.02 x 12 – 2.54 pitch	Würth Elektronik	61301211821
CN5, 7, 8	3	Connectors for the module	–	–	Φ1.1 x 4 – 3.5 pitch	Würth Elektronik	691243110004
CN6	1	Connectors for connection with the power supply	–	–	Φ1.1 x 2 – 3.5 pitch	Würth Elektronik	691214110002S
J1	1	DC jack	–	–	9.0 x 14.5	Würth Elektronik	694106301002
J2	1	UART pin header	–	–	Φ1.1 x 4 – 2.54 pitch	Würth Elektronik	61300411121
C1	1	Electrolytic capacitor	100 µF, 50 V	±10%	–		860020674015
PCB	1	PCB	–	–	80 x 60		

When using a circuit board prepared by the user in place of the base board, be sure to install an electrolytic capacitor equivalent to C1 between the VCC and GND terminals. The

non-installation of the capacitor will cause damage to or failure of the module.

ON Semiconductor is licensed by the Philips Corporation to carry the I<sup>2</sup>C bus protocol.

ON Semiconductor and  are trademarks of Semiconductor Components Industries, LLC dba ON Semiconductor or its subsidiaries in the United States and/or other countries. ON Semiconductor owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of ON Semiconductor's product/patent coverage may be accessed at [www.onsemi.com/site/pdf/Patent-Marketing.pdf](http://www.onsemi.com/site/pdf/Patent-Marketing.pdf). ON Semiconductor reserves the right to make changes without further notice to any products herein. ON Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does ON Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using ON Semiconductor products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by ON Semiconductor. "Typical" parameters which may be provided in ON Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. ON Semiconductor does not convey any license under its patent rights nor the rights of others. ON Semiconductor products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use ON Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold ON Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that ON Semiconductor was negligent regarding the design or manufacture of the part. ON Semiconductor is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

## PUBLICATION ORDERING INFORMATION

### LITERATURE FULFILLMENT:

Literature Distribution Center for ON Semiconductor  
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA  
Phone: 303-675-2175 or 800-344-3860 Toll Free USA/Canada  
Fax: 303-675-2176 or 800-344-3867 Toll Free USA/Canada  
Email: [orderlit@onsemi.com](mailto:orderlit@onsemi.com)

**N. American Technical Support:** 800-282-9855 Toll Free  
USA/Canada  
**Europe, Middle East and Africa Technical Support:**  
Phone: 421 33 790 2910

**ON Semiconductor Website:** [www.onsemi.com](http://www.onsemi.com)

**Order Literature:** <http://www.onsemi.com/orderlit>

For additional information, please contact your local Sales Representative