

SCE2

SMALL 4 AXIS STEPPER MOTOR CONTROLLER MODULE
WITH G-CODE INTERPRETATOR

DATASHEET

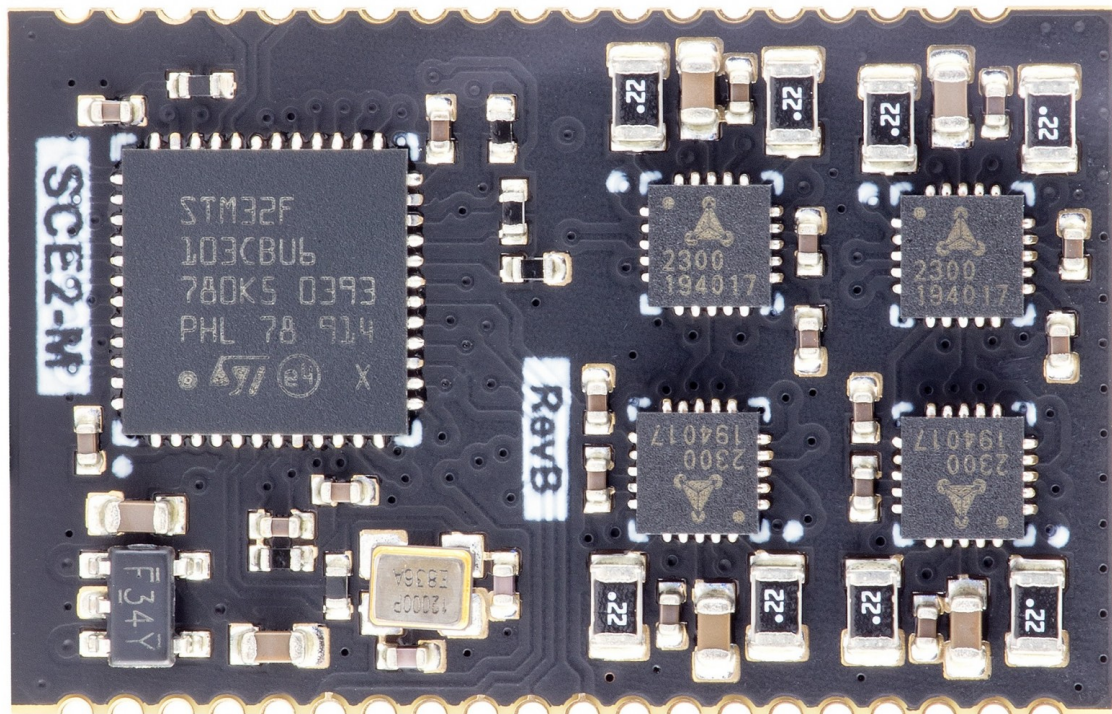


KUROKESU

2021-09-07, Rev. #73

Overview

SCE2-M is a fully integrated stepper motor controller module for digital control for limited space applications. Designed for small 3D printers, laser cutters/engravers, CNC mills, pick and place machines, robots, test fixtures, and other motorized devices. SCE2-M module is the smallest motor controller that requires no external components and runs industry-standard g-code processor with linear interpolation control open-source firmware



Features

- System On Module (SOM) design
- Fully integrated, ready to use
- 4 stepper motors drivers, up to 1/256 microstepping
- 4 Limit switch inputs
- Direct connect USB port
- Optional TTL USART/I2C ports 8 IO ports (some with hardware PWM output capability)
- 32bits ARM-Cortex M3 microcontroller, running at 72Mhz
- Advanced motor control with Trinamic drivers
- Digital current control of stepper motor drivers directly from g-code

Pinout

SCE2-M module has some STM32 pins routed to external pads. Pads grouped by functionality provided in tables below.

GPIO pin mapping table

IO pin	CPU port	Function	CPU features
1	PB1	SPINDLE ENABLE	ADC
2	PB9	PROBE	I2C SDA
3	PA8	PWM OUTPUT	
4	PB4	FLOOD	
5	PB3	MIST	
6	PB0	SPINDLE DIR	ADC
7	PB5	STOP	
8	PB8		I2C SCL

i PA8 (PWM OUTPUT) is set to 10kHz frequency, but can be adjusted in firmware.

Limit switch mapping table

Port	Net name
PB12	LIMIT1
PB13	LIMIT2
PB14	LIMIT3
PB15	LIMIT4
PA9	LIMIT_EN

UART pin mapping table

CPU port	Net name
PB7	USART1_RX
PB6	USART1_TX

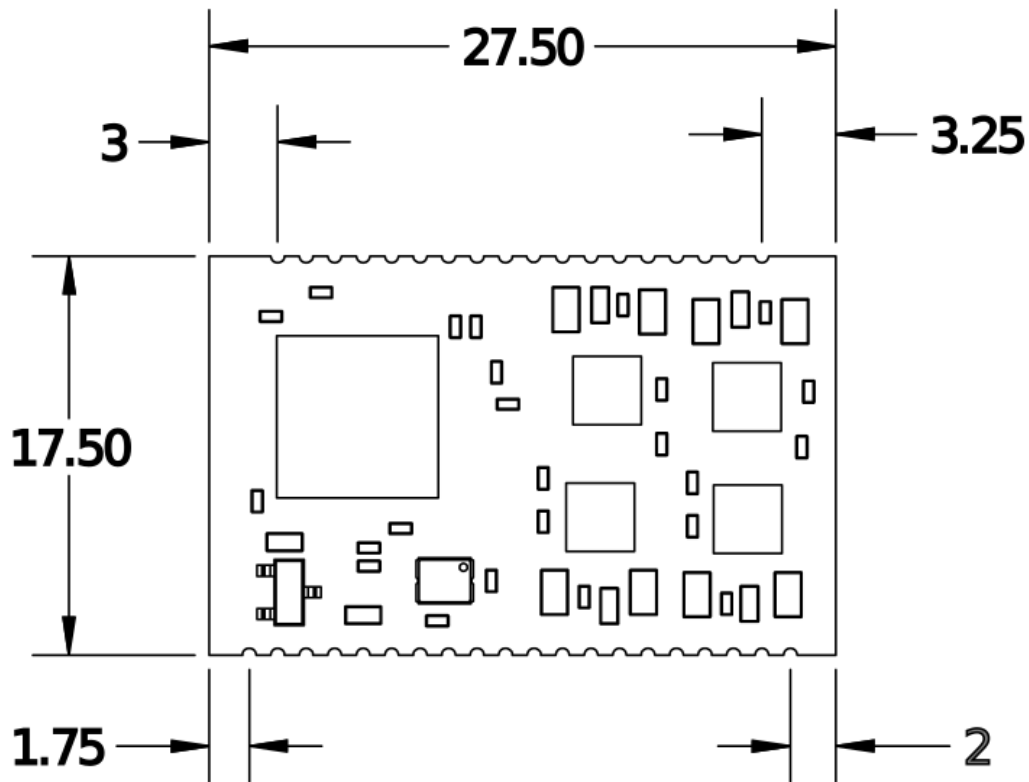
Schematic symbol

M1			
25	IO1	1A	7
26	IO2	1B	8
27	IO3	2A	6
28	IO4	2B	5
29	IO5		
30	IO6	3A	34
31	IO7	3B	35
9	IO8	4A	33
		4B	32
		5A	3
		5B	4
10	SWDCLK	6A	2
11	SWDIO	6B	1
12	RESET		
		7A	38
14	RX	7B	39
13	TX	8A	37
		8B	36
15	USB_N		
16	USB_P	LIMIT1	24
		LIMIT2	23
18	VM	LIMIT3	22
19	5V	LIMIT4	21
17	GND	LIMIT_EN	20
SCE2-M			

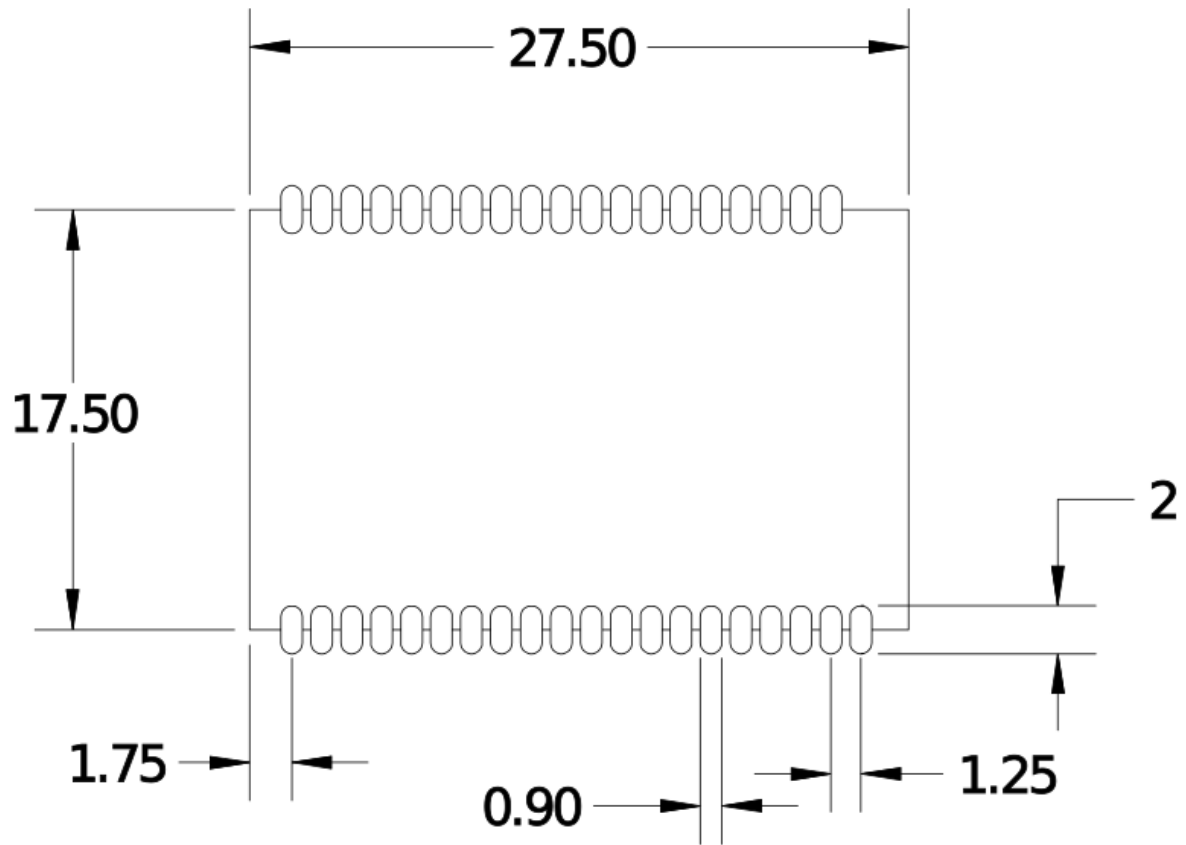
i 5V is used for motor drivers. Logic level of all GPIO pins is 3.3V

Dimensions and footprint

Mechanical dimensions




Component footprint



i 3D STEP file is maintained on [Github](#)

Firmware

 Open source STM32F103 GRBL firmware is maintained on [Github](#)

 By default firmware can be uploaded with STLINK over exposed SWD pins.

 STM32F103 supports flashing over USB port, but in latest release it is still not supported

Communication over USB

Drivers

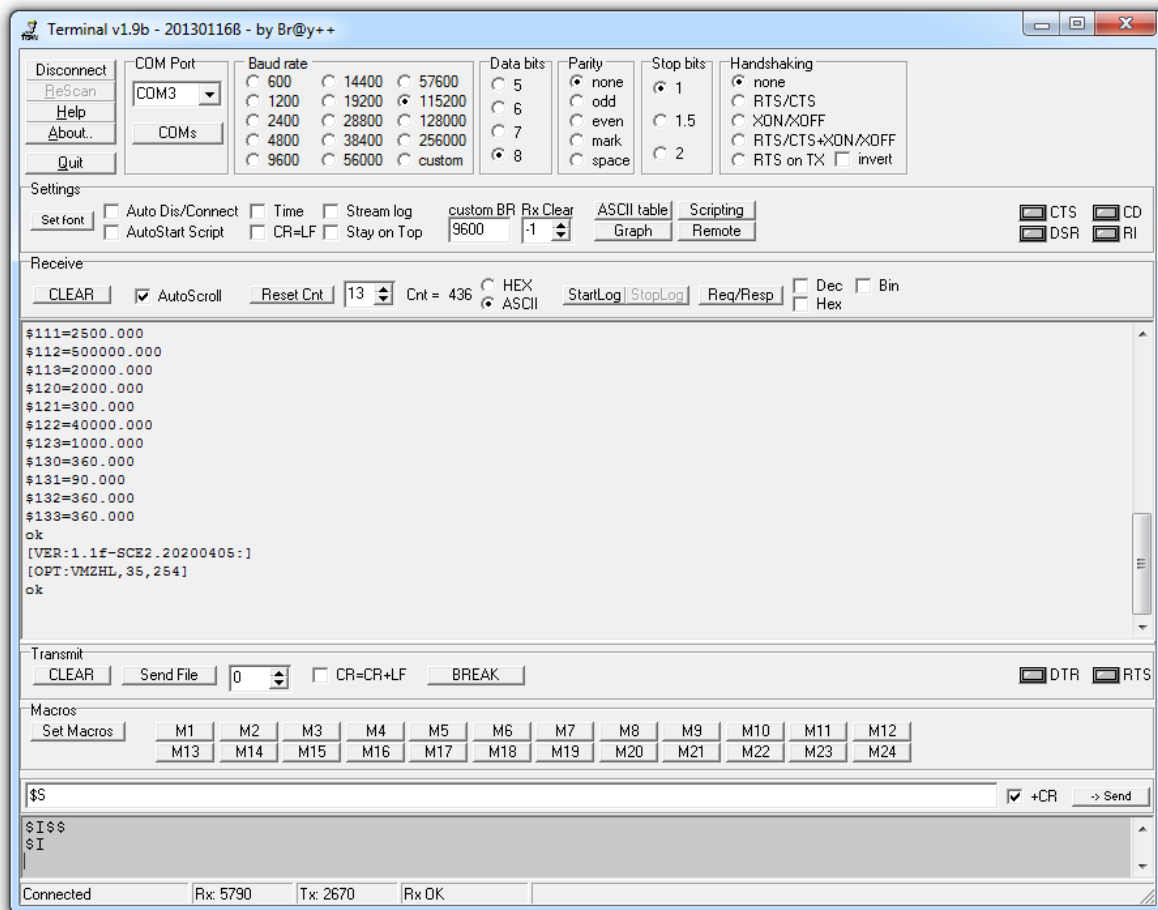
Stock firmware registers as "STM32 Virtual COM". Most modern operating systems should work out of the box, but original drivers can be downloaded from [STMicroelectronics web page](#)

COM port

Firmware is configured to work at **115200 8N1** baud rate

Terminal program

Feel free to use favorite terminal program. For further demonstration will use [Terminal App](#)



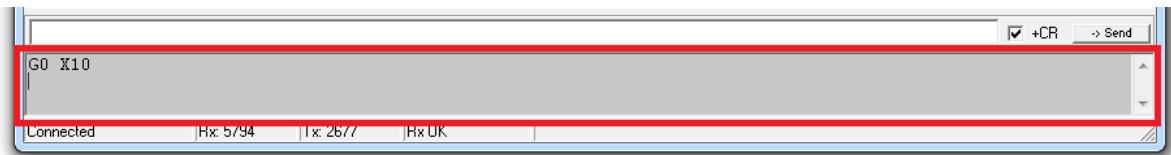
In some Windows cases Prolific USB to serial controller drivers can claim detected COM port, in this case uninstall driver and install downloaded from ST web page.

Quick start guide

First motor moves

Assume axis has proper configuration sent earlier, making moves is as easy as entering command

In terminal bottom field type `G0 X10` and press `Enter`



If linear actuator is connected it should move at **default max speed** by 10mm (rotary actuator should turn 10deg).


In order to move at different speeds G1 command should be used. For example

`G1 Y10 F100`



List of supported G-code commands:

- **G0, G1:** Linear Motions
- **G2, G3:** Arc and Helical Motions
- **G4:** Dwell
- **G10 L2, G10 L20:** Set Work Coordinate Offsets
- **G17, G18, G19:** Plane Selection
- **G20, G21:** Units
- **G28, G30:** Go to Pre-Defined Position
- **G28.1, G30.1:** Set Pre-Defined Position
- **G38.2:** Probing
- **G38.3, G38.4, G38.5:** Probing
- **G40:** Cutter Radius Compensation Modes OFF (Only)
- **G43.1, G49:** Dynamic Tool Length Offsets
- **G53:** Move in Absolute Coordinates
- **G54, G55, G56, G57, G58, G59:** Work Coordinate Systems
- **G61:** Path Control Modes
- **G80:** Motion Mode Cancel
- **G90, G91:** Distance Modes
- **G91.1:** Arc IJK Distance Modes
- **G92:** Coordinate Offset
- **G92.1:** Clear Coordinate System Offsets
- **G93, G94:** Feedrate Modes

- **G100, G101:** TMC2300 register read/write
- **M0, M2, M30:** Program Pause and End
- **M3, M4, M5:** Spindle Control
- **M7*, M8, M9:** Coolant Control
- **M56*** : Parking Motion Override Control

 Controller is shipped with X axis configured for RSB1 and X axis configured for LSA1 (0.635mm version)

Configuration

In order for controller to function correctly pulse count per revolution, acceleration and optionally motion limits has to be specified. In order to read configuration type  and press .

 Recommended values for LSA1 actuator and RSB1 actuator

Read configuration parameters

All internal configuration parameters will be printed out. For example:

```
$0=6
$1=255
$2=0
$3=31
$4=1
$5=0
$6=0
$10=19
$11=0.010
$12=0.002
$13=0
$20=0
$21=0
$22=1
$23=15
$24=50.000
$25=200.000
$26=250
$27=5.000
$30=1000
$31=0
$32=0
$100=111.110
```

```
$101=1007.874  
$102=17.778  
$103=1536.000  
$110=20000.000  
$111=2500.000  
$112=500000.000  
$113=20000.000  
$120=2000.000  
$121=300.000  
$122=40000.000  
$123=1000.000  
$130=360.000  
$131=90.000  
$132=360.000  
$133=360.000
```

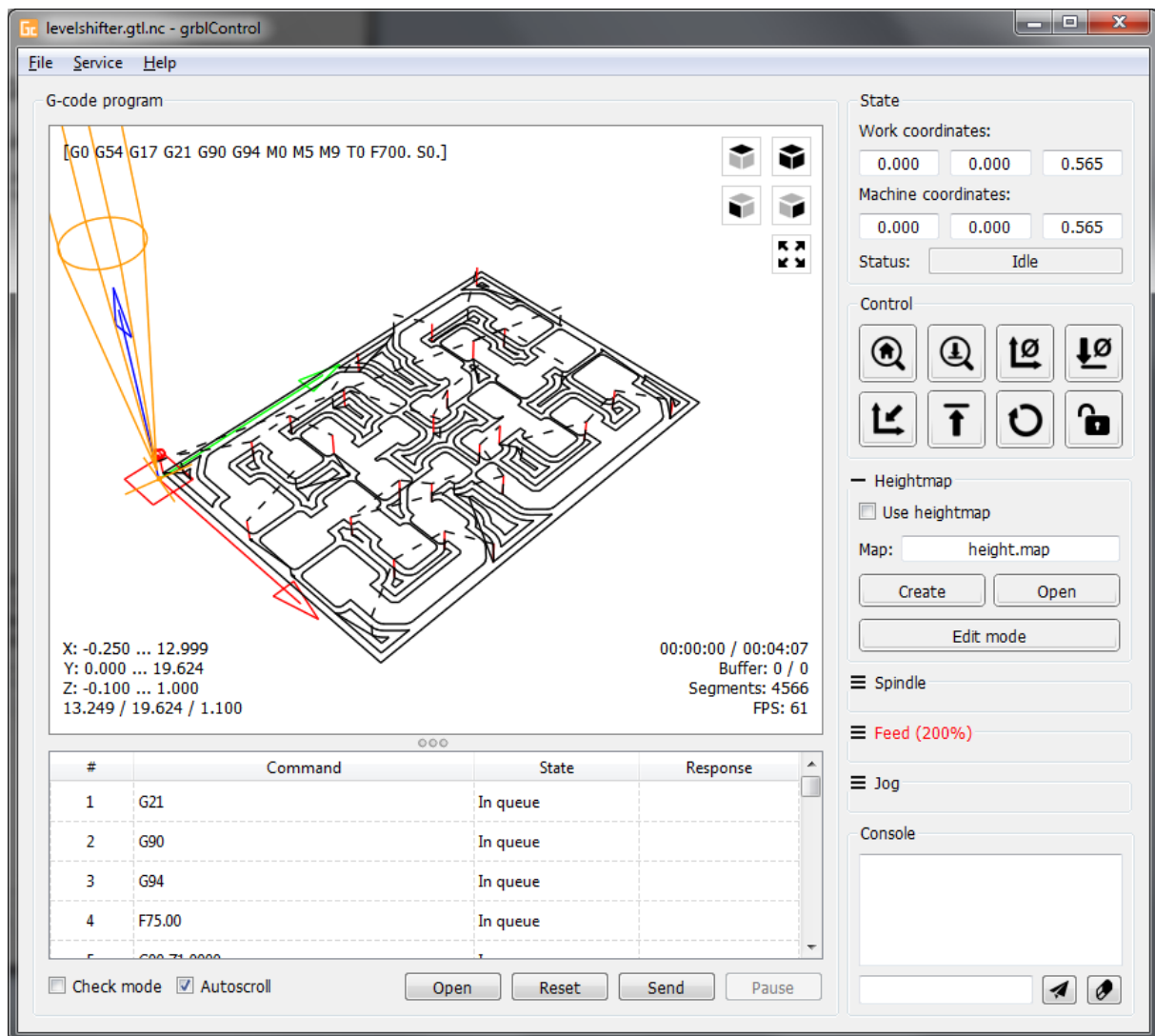
Detailed explanation can be found on [GRBL documentation](#)

Write configuration parameters

Writing specific value is as easy as typing `$100=1000` and pressing `Enter`. If number is valid, this command will overwrite existing value and save to EEPROM.

Control software

There are number of software packages suited to control CNC machines. Comprehensive list is maintained by [GRBL developers on GitHub](#). Our favorite is **Candle** by Denvi



Some g-code examples and recipes

G-code while being universal machine control language has various flavors and capabilities. Below section is dedicated to speed up manual machine control with simple commands.

Move motors

There are two basic linear motion commands G0 and G1. Both are generally the same just G0 uses default max speed constant from configuration. Examples (assume linear actuator is connected):

Command	Explanation
G0 X100	Move X axis to absolute position 100mm
G1 X0 F100	Move X axis to 0 position at speed 100mm/min

Control GPIO pins

GRBL is designed for CNC machines and IO pins are used to make sense. Pin description is provided [here](#), but for actual functionality source code should be inspected.

Command	Explanation
M7	MIST = ON
M8	FLOOD = ON
M9	MIST = OFF, FLOOD = OFF

Command	Explanation
M3	Set clockwise spindle rotation
S300	Set spindle control pin PWM duty cycle at 30% (max S = 1000)
M4	Set counter clockwise spindle rotation
M5	Turn spindle off

Motor homing

After powering controller on it does not know motor position, thus it should be driven to home reference position

Command	Explanation
\$HX	Initialize X axis

Read controller status

Almost any time controller status can be read with command `?`. It will report motor status (Idle/Run), actual positions and some other info.

Command	Output
<code>?</code>	<code><Idle MPos:0.000,90.000,0.000,0.000 Bf:35,254 FS:0,0></code>

Read firmware version string

Functionality between versions will vary, some firmware versions or branches can have special functionality.

Command	Output
<code>\$I</code>	<code>[VER:1.1f-SCE2.20200405:]</code> <code>[OPT:VMZHL,35,254]</code> <code>ok</code>

Probing

Firmware allows use of touch trigger probes. Moves down Z axis and stops on when probe pin is triggered, then reports collision point.

Command	Explanation and Output
<code>G38.2 F100</code> <code>Z-100</code>	Move down Z axis until PROBE pin is triggered <ul style="list-style-type: none"> • F - Speed • Z - Target depth Replies with <code>[PRB:1.503,0.000,-22.860,0.000:1]</code>

Idle / wait

Idle command is useful for certain operations where controller needs to wait and do nothing (for example wait till spindle speeds up).


Command	Explanation
<code>G4 P60000</code>	Wait 1 minute, reply with <code>ok</code> when done

Reading and writing TMC2300 registers

Firmware has implemented functionality to read and write individual Trinamic TMC2300 registers over internal UART single wire interface. It shares transmit and receive line like an RS485 based interface. Data transmission is secured using a cyclic redundancy check and each TMC2300 driver is addressed individually as follows:


Axis	Address
X	0
Y	1
Z	2
A	3
Broadcast to all axes	9

 [Official TMC2300 documentation](#)

 All numbers in g-code arguments are decimal base numbers

Write registers

Write register command forwards g-command arguments to TMC2300 configuration interface. While it is not recommended because of stability issues, write command can be sent even if motors are turning (G0, G1, ... commands)

 7'th bit of TMC2300 register has to be set to 1, thus for example **0x10** IHOLD_IRUN becomes **0x90**

G100	P<xx>	L<xx>	N<xx>	S<xx>	F<xx>	R<xx>
G-command	TMC2300 address	TMC2300 register (mask with 0x80)	DATA3	DATA2	DATA1	DATA0

UART WRITE ACCESS DATAGRAM STRUCTURE																			
each byte is LSB...MSB, highest byte transmitted first																			
0 ... 63																			
sync + reserved								8 bit slave address			RW + 7 bit register addr.			32 bit data				CRC	
0...7								8...15			16...23			24...55				56...63	
1	0	1	0	Reserved (don't cares but included in CRC)				SLAVEADDR=(MS2, MS1)			register address		1	data bytes 3, 2, 1, 0 (high to low byte)				CRC	
0	1	2	3	4	5	6	7	8	..	15	16	..	23	24	..	55	56	..	63

i Required checksum is calculated in firmware

Read registers

Read registers command has two arguments TMC2300 address and register address. It responds to terminal immediately.

G101	P<xx>	R<xx>
G-command	TMC2300 address	TMC2300 register

Response is decimal coded 12 bytes. Sample reply looks like this: [TMC: 5,0,16,112,5,255,16,0,0,0,0,157]

Reply byte	Meaning
0	Echo transmit byte0
1	Echo transmit byte0
2	Echo transmit byte0
3	Echo transmit byte0
4	Reply from TMC2300 sync (0x05)
5	Address
6	Register
7	Data byte3
8	Data byte2
9	Data byte1
10	Data byte0
11	Checksum

UART READ ACCESS REQUEST DATAGRAM STRUCTURE																
each byte is LSB...MSB, highest byte transmitted first																
sync + reserved								8 bit slave address			RW + 7 bit register address			CRC		
0...7								8...15			16...23			24...31		
1	0	1	0	Reserved (don't cares but included in CRC)				SLAVEADDR=(MS2,MS1)			register address		0	CRC		
0	1	2	3	4	5	6	7	8	..	15	16	..	23	24	..	31

UART READ ACCESS REPLY DATAGRAM STRUCTURE																			
each byte is LSB...MSB, highest byte transmitted first																			
0 63																			
sync + reserved								8 bit master address			RW + 7 bit register addr.			32 bit data			CRC		
0...7								8...15			16...23			24...55			56...63		
1	0	1	0	reserved (0)				0xFF			register address	0	data bytes 3, 2, 1, 0 (high to low byte)			CRC			
0	1	2	3	4	5	6	7	8	..	15	16	..	23	24	..	55	56	..	63

Examples

Set motor X current

Each motor driver can drive different motor. Power and other settings can be set individually. For example IHOLD_IRUN (0x10) register controls motor idle, run and timing parameters.

G100 P0 L144 N0 S0 F10 R5

G100	Write to TMC2300 g-command
P0	Driver address = 0 (X axis)
L144	TMC2300 register IHOLD_IRUN (0x10) 0x10 ->(set 7'th bit to 1)-> 0x90 = 144 (dec)
N0	Data3 = 0 (not used)
S0	Data2 = 0 (see IHOLDDELAY in table below)
F10	Data1 = 10 (see IRUN in table below)
R5	Data0 = 5 (see IHOLD in table below)

VELOCITY DEPENDENT DRIVER FEATURE CONTROL REGISTER SET (0x10...0x1F)					
R/W	Addr	n	Register	Description / bit names	
W	0x10	5 + 5 + 4	IHOLD_IRUN	Bit	IHOLD_IRUN – Driver current control
				4..0	IHOLD (Reset default=8) Standstill current (0=1/32 ... 31=32/32) Setting <i>IHOLD</i> =0 allows to choose freewheeling or coil short circuit (passive braking) for motor stand still.
				12..8	IRUN (Reset default=31) Motor run current (8=9/32 ... 31=32/32) Working with values below 8 is not recommended. <i>Hint:</i> Choose sense resistors in a way, that normal <i>IRUN</i> is 16 to 31 for best performance.
				19..16	IHOLDDelay (Reset default=1) Controls the number of clock cycles for motor power down after standstill is detected (<i>stst</i> =1) and <i>TPOWERDOWN</i> has expired. The smooth transition avoids a motor jerk upon power down. 0: instant power down 1..15: Delay per current reduction step in multiple of 2 ¹⁸ clocks

Read input register

Each TMC2300 driver has input status registers GSTAT (0x01) indicating over temperature, short circuit, and more.

G101 P0 R1

Response looks like:

[TMC:5,0,1,193,5,255,1,0,0,0,1,154]

Response contains only 3 bits of useful information. See explanation in table below.

R+ WC	0x01	3	GSTAT	Bit	GSTAT – Global status flags (Re-Write with '1' bit to clear respective flags)
				0	<i>reset</i> 1: Indicates that the IC has been reset since the last read access to GSTAT. All registers have been cleared to reset values.
				1	<i>drv_err</i> 1: Indicates, that the driver has been shut down due to overtemperature or short circuit detection since the last read access. Read DRV_STATUS for details. The flag can only be cleared when all error conditions are cleared.
				2	<i>u3v5</i> 1: Actual state of the supply voltage comparator. A high value means that the voltage sinks below 3.5V. This flag is not latched and thus does not need to be cleared.

Automating tasks

Python g-code sender

Automating tasks can be done with Python script. Below is script example to send commands provided in text file:

```
import time
import os
from tqdm import tqdm
import serial
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('-p', '--port', help='COM port', required=True)
parser.add_argument('-f', '--file', help='File name', required=True)
args = parser.parse_args()

ser = serial.Serial()
ser.port = str(args.port)
ser.baudrate = 115200
ser.timeout = 10
ser.open()
ser.flushInput()
ser.flushOutput()

print("Reading file...")
lines = []
with open(args.file) as fp:
    for cnt, line in enumerate(fp):
        lines.append(line)

print("Sleeping 1s...")
time.sleep(1)

for line in tqdm(lines):
    l = bytes(line.strip()+ '\n', 'utf8')
    ser.write(l)
    grbl_out = ser.readline()
```

Text file example:

G91

M7

M8
G1 X45 F20000
M9
G1 X45 F20000

M8
G1 X45 F20000
M9
G1 X45 F20000

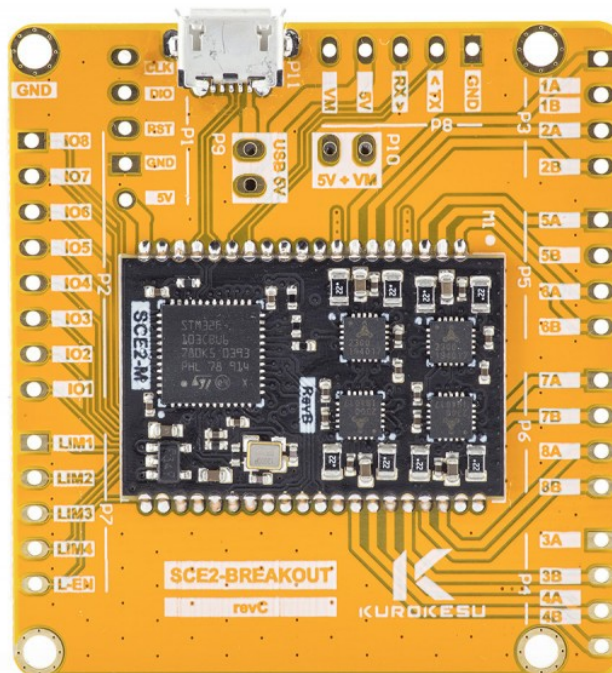
M8
G1 X45 F20000
M9
G1 X45 F20000

M8
G1 X45 F20000
M9
G1 X45 F20000

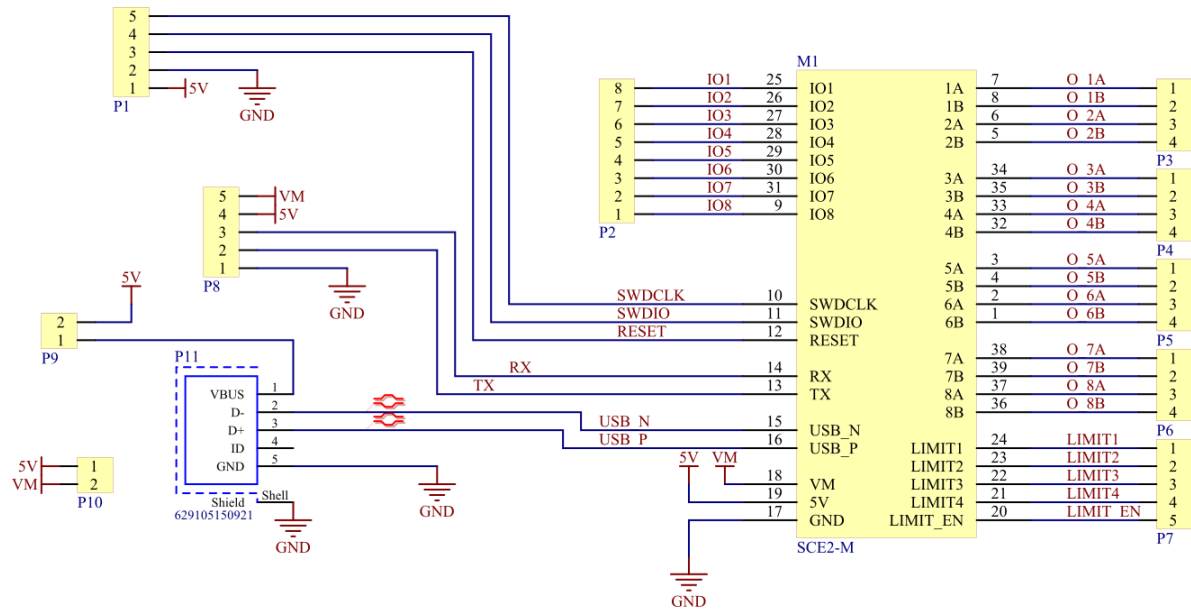
G90

SCE2-BREAKOUT demo board

SCE2-BREAKOUT is a fully integrated stepper motor controller module carrier board for quick SCE2-M module evaluation. Designed for small 3D printers, laser cutters/engravers, CNC mills, pick and place machines, robots, test fixtures, and other motorized devices. SCE2-M module is the smallest motor controller that requires no external components and runs industry-standard g-code processor with linear interpolation control open-source firmware.



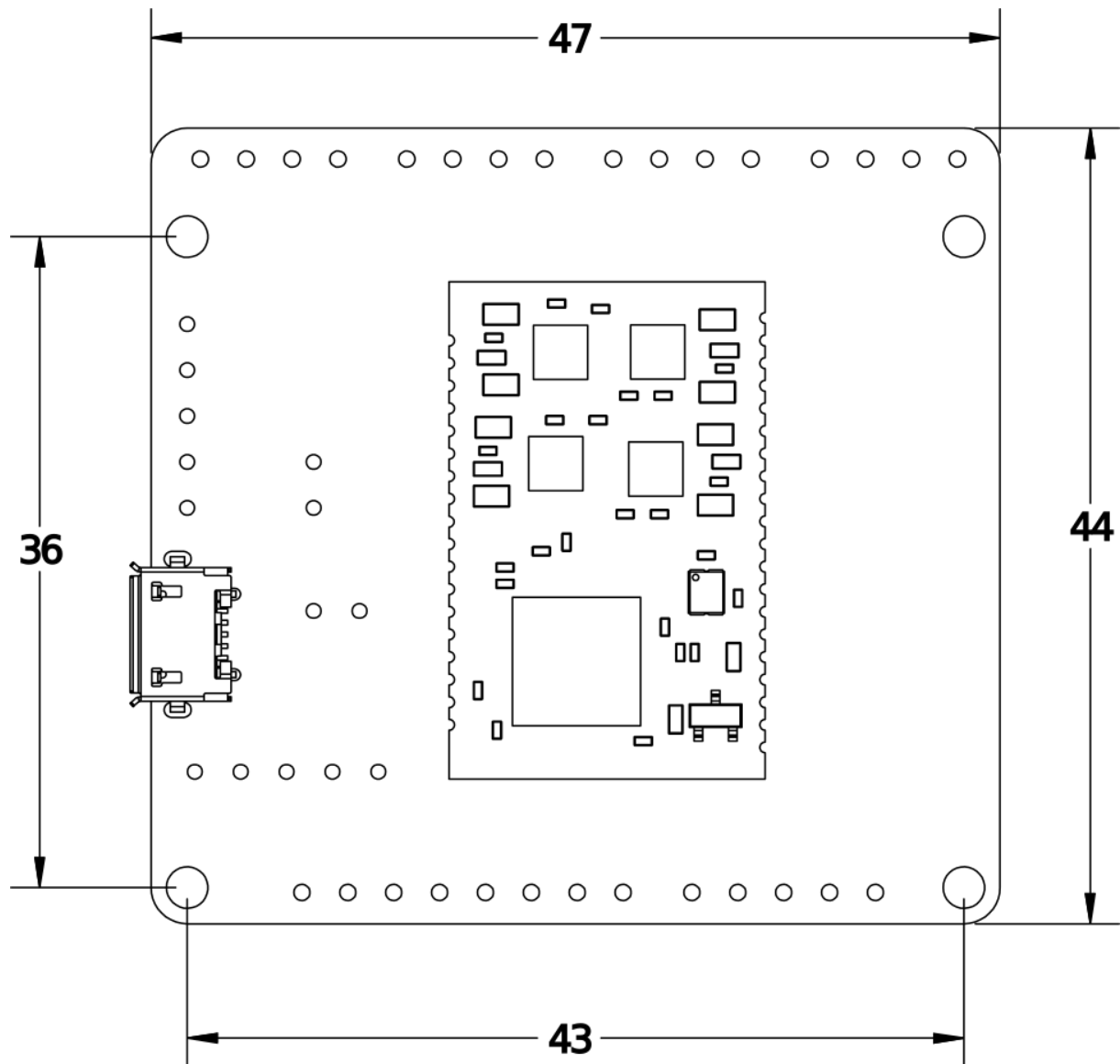
Schematics



⚠ P9 is breaks power supply from USB port. Should be disconnected if power is applied via P8 header

❗ P10 bridges 5V and VM. Should be disconnected if VM is applied via P8

Dimensions



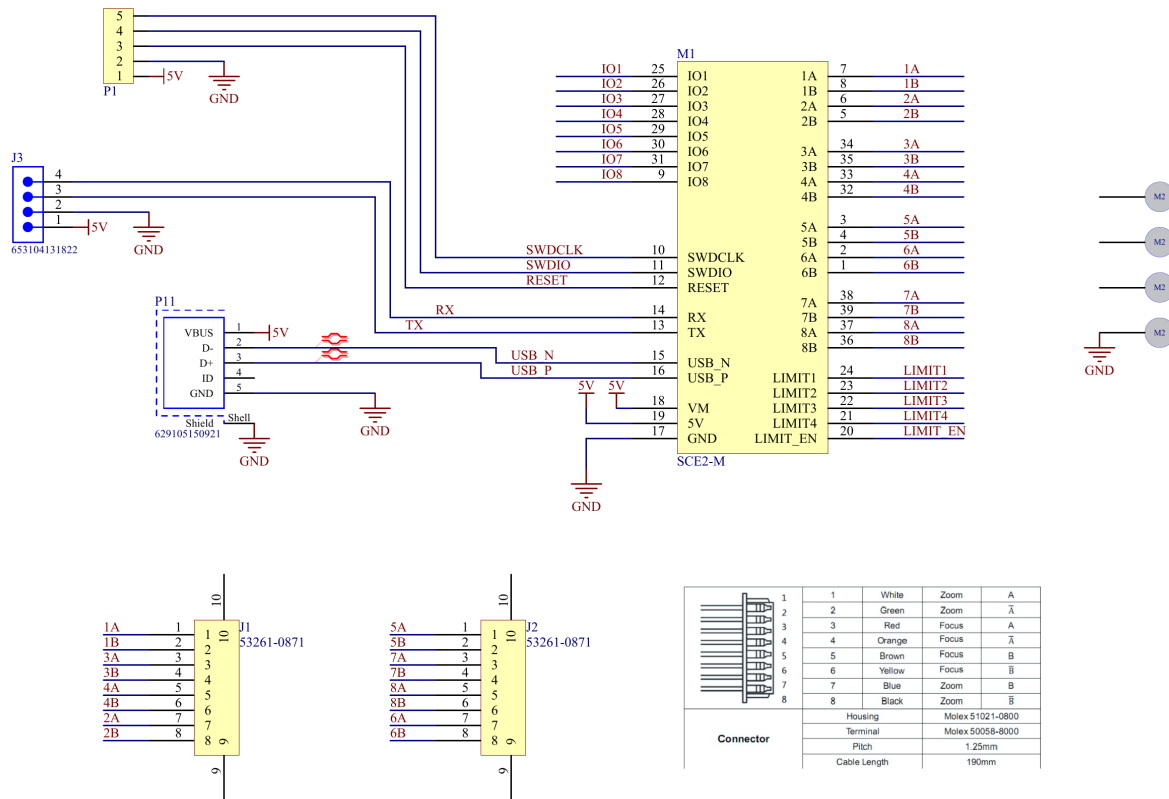
i 3D STEP file is maintained on [Github](#)

SCE2_dual_L012 dual lens controller board

SCE2_dual_L012 is a fully integrated stepper motor controller module carrier board for controlling two motorized lens zoom/focus parameters. Controller requires only 5V power supply from the USB port and controlled with text-based G-code commands over USB or optionally TTL UART.



Schematics



Motor wiring vs axis definition

Axis	Motor function	Connector
X	Focus	J1
Y	Zoom	J1
Z	Focus	J2
A	Zoom	J2

Controlling the lenses

Recommended GRBL settings

Controller is shipped with default settings as listed below.

```
$0=6
$1=255
$2=0
$3=31
$4=1
$5=0
```

```

$6=0
$10=19
$11=0.010
$12=0.002
$13=0
$20=0
$21=0
$22=1
$23=15
$24=50.000
$25=200.000
$26=250
$27=5.000
$30=1000
$31=0
$32=0
$100=111.110
$101=111.110
$102=111.110
$103=111.110
$110=50000.000
$111=50000.000
$112=50000.000
$113=50000.000
$120=5000.000
$121=5000.000
$122=5000.000
$123=5000.000
$130=360.000
$131=360.000
$132=360.000
$133=360.000
$N0=G100P9L144N0S0F1R1

```

Startup sequence

Default TMC2300 power is too high for small stepper motors and will overheat and damage them. Motor drive/sleep current has to be decreased with commands listed below:

```

G100 P0 L144 N0 S0 F1 R1
G100 P1 L144 N0 S0 F1 R1
G100 P2 L144 N0 S0 F1 R1
G100 P3 L144 N0 S0 F1 R1

```

Or in a single command (valid since 20200103 firmware update)

```
G100 P9 L144 N0 S0 F1 R1
```

Which then can be added in GRBL startup sequence `$N0=G100P9L144N0S0F1R1`

Homing sequence

Lenses have no limit switches, thus precision homing is not possible. Instead each motor is driven to max position until hard stop and it is assumed that home position is achieved.

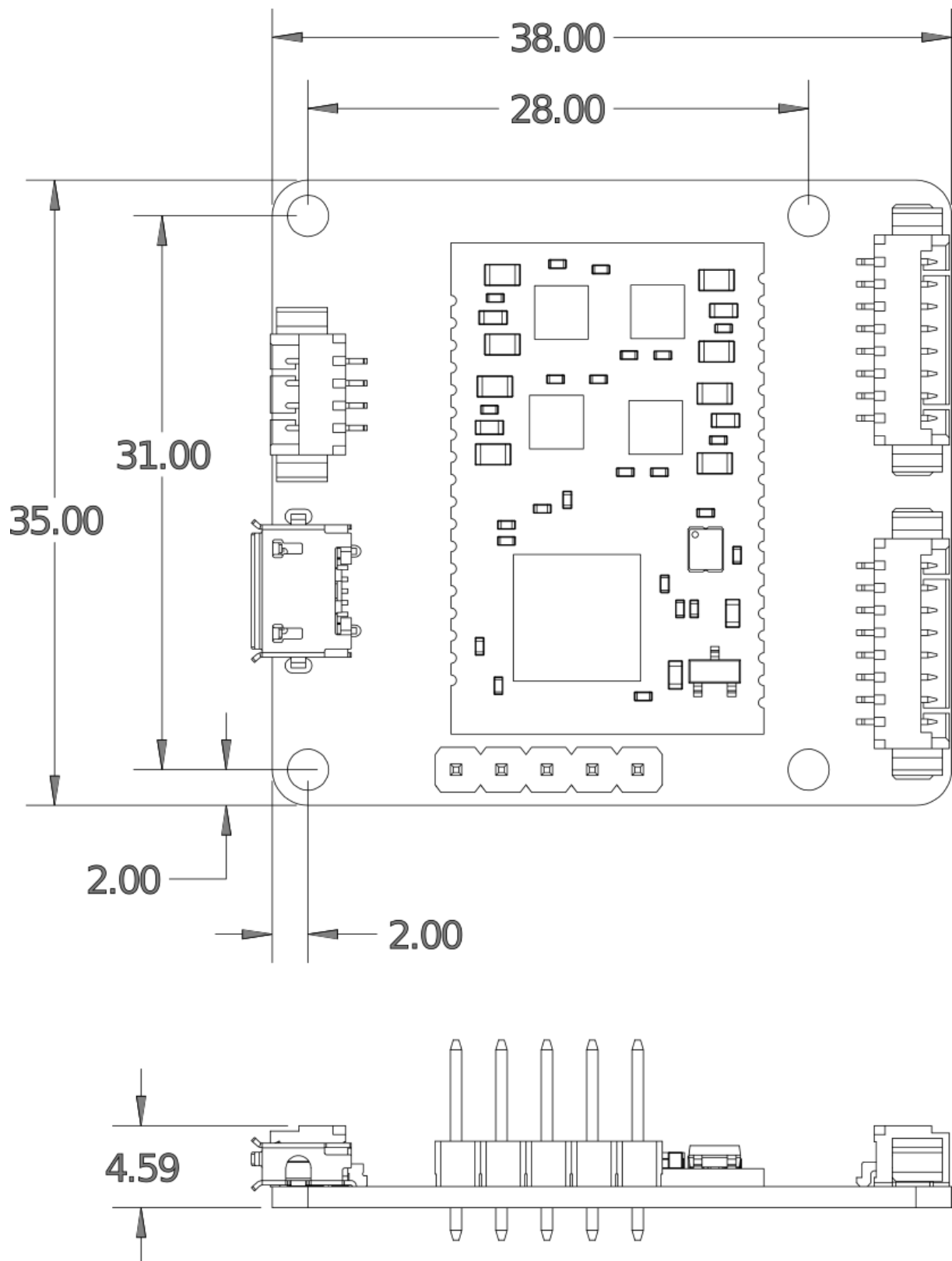
```
G91 # set motion to relative coordinate system
F30000 # set default motion speed
G1 X-500
G1 Y-1200
G1 Z-500
G1 A-1200
G90 # set motion to absolute coordinate system
G92 X0 Y0 Z0 A0 # set current position as 0
```

Control sequence


If power settings are reduced in startup stage, lens motors can be operated right away. Standard G-code motion commands can be used to drive motors. For example:

```
G1 Y100 A100 F40000
G1 Y-100 A100 F40000
G1 Y100 A-100 F40000
G1 Y-100 A-100 F10000
```

Dimensions



❗ Mounting hole diameter is (not indicated) in drawing is 2.3mm

 3D STEP file is maintained on [Github](#)

Precautions and disclaimers

General disclaimer

- ALL PRODUCTS, PRODUCT SPECIFICATIONS, AND DATA ARE SUBJECT TO CHANGE WITHOUT NOTICE TO IMPROVE RELIABILITY, FUNCTION OR DESIGN OR OTHERWISE
- Kurokesu UAB, its affiliates, agents, and employees, and all persons acting on its or their behalf (collectively, "Kurokesu"), disclaim any and all liability for any errors, inaccuracies or incompleteness contained in any datasheet or any other disclosure relating to any product.
- The Information given herein is believed to be accurate and reliable. However, users should independently evaluate the suitability of and test each product selected for their applications
- See Kurokesu standard terms and conditions for warranty and other information

Precautions

- Do not short circuit any part of the module
- Do not exceed nominal input
- Do not overload outputs
- Keep the module dry
- Observe the electrostatic discharge (ESD) precautions when handling the product. Damage caused by non-observance of the above instructions is not covered by the warranty
- Power electronics equipped with thermal shutdown circuitry, but if controller becomes too hot disconnect it
- Module is designed exclusively for installation into a device or housing to prevent external influences such as humidity/water or dirt
- Add active and/or passive power filtering circuitry in an electromagnetically noisy environment if a module becomes unstable
- Add active and/or passive power filtering circuitry if module exceeds allowed EMC emissions