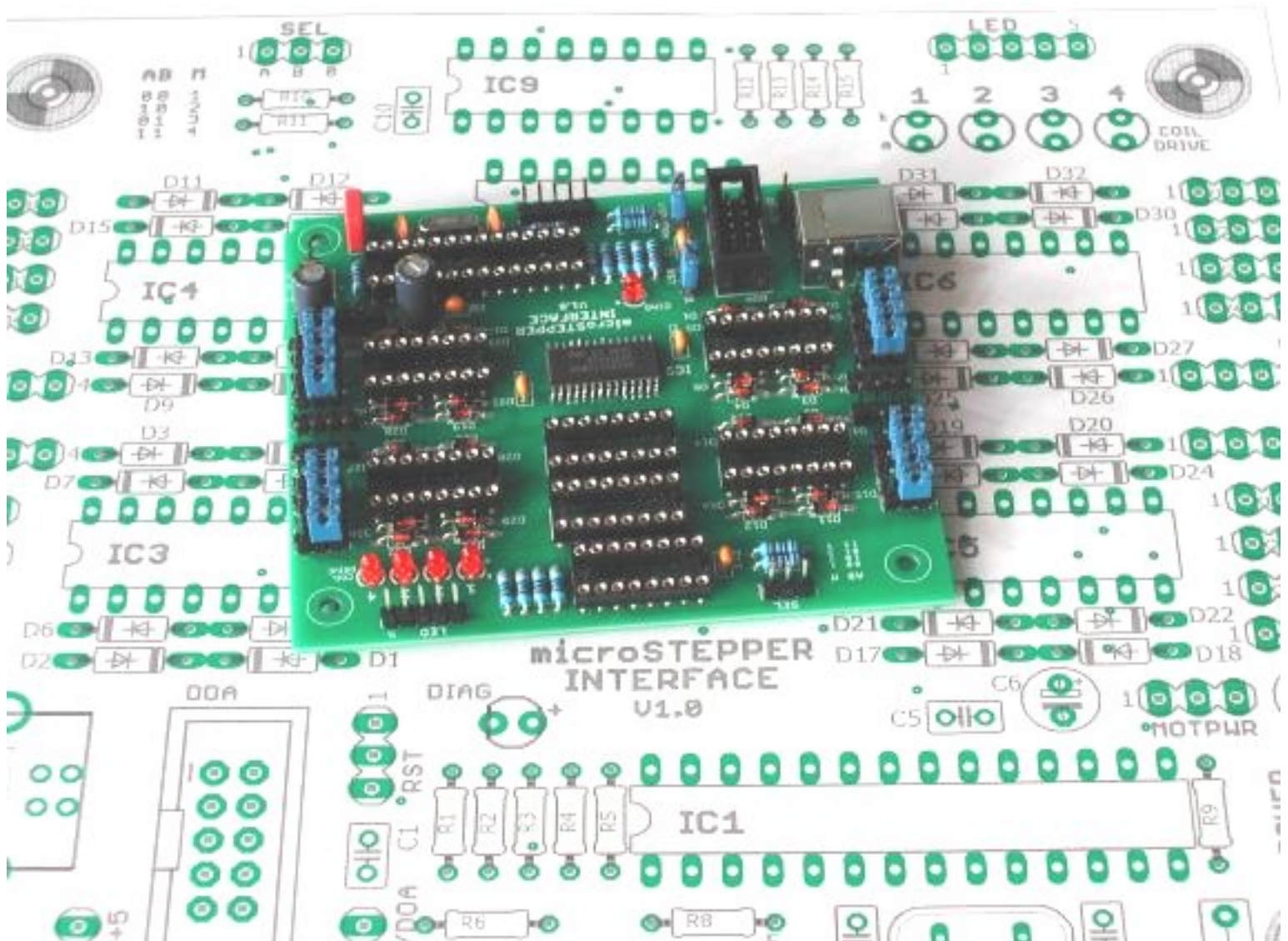


# PHCC

## $\mu$ Stepper controller



© Henk Gooijen – *henkie*

Version: V1.0 (draft) – 28 February 2019

## ***History***

### **07-2018 : Draft design PCB, first 18F2550 PIC firmware version**

Initial work started for the new PHCC daughterboard called  $\mu$ STEPPER. This board can control up to 4 stepper motors, just like the existing PHCC STEPPER. The improvement is in the use of a stepper controller IC to enable micro-stepping which results in a significant more smooth movement.

### **10-2018 : Prototype PCB**

The first prototype PCB order was lost in the mail ☹ New prototype PCBs arrived.

### **01-2019 : First firmware version coded**

Command set defined. First X.27 motor running.

### **02-2019 : Firmware debugging and more commands**

Motors #2 and #3 were not working; well actually they were OK, but the silkscreen text on the PCB for MOTOR2 and MOTOR3 connection pins was swapped. Commands to use a home sensor are expanded to specify the active signal level of the sensors. The absolute range expanded to up to 6399 steps to accommodate steppers with very high resolution. The acceleration/deceleration table is now \*per\* motor defined, but the default table is identical for all motors. However, it is now possible to upload a user-defined acceleration/deceleration table which is also \*per\* motor. The DIAG LED has an additional diagnostic indication mode to show whether a DOA packet is corrupted, causing DOA reception out-of-sync condition (which can be resolved by the watchdog functionality).

## **1. Introduction**

## **2. Overview of the PHCC $\mu$ STEPPER board**

1. Block diagram
2. Power supply connections
3. Motor connections
4. Interface connections

## **3. Connections & jumpers on the $\mu$ STEPPER**

1. POWER connector
2. M1, M2, M3, M4 connector
3. SENS connector
4. SEL connector
5. LED connector
6. DOA connector
7. USB connector
8. USB/DOA jumper
9. RST jumper
10. DOA+5 jumper
11. MOTPWR jumper
12. MOTOR OUTPUT SELECT jumpers

## **4. Assembly of the $\mu$ STEPPER**

## **5. Connection wiring**

1. Introduction
2. Power supply wiring
3. Motor(s) wiring
4. Interface wiring

## **6. $\mu$ STEPPER commands**

1. How to control a  $\mu$ STEPPER motor
2. Power control
3. IDENTIFY command
4. USB debug command
5. Watchdog functionality

## **7. Parts list $\mu$ STEPPER circuit board**

Appendix A – Component locator  $\mu$ STEPPER circuit board

Appendix B –  $\mu$ STEPPER Demonstrator program

# 1. Introduction

The PHCC system already has a stepper motor interface board, called PHCC STEPPER. The stepper motor signals are generated by the firmware in the PIC processor, and driver ICs provide the current to drive the motor coils, because the PIC cannot supply the required motor current.

This new PHCC  $\mu$ STEPPER interface also uses the same motor driver ICs, but this design uses an advanced stepper motor controller IC which is controlled by a PIC. The firmware now only generates a movement direction signal and pulses equal to the number of steps to move. The major improvement is the possibility to do micro-stepping with the motor controller IC. With “normal” stepping you can clearly see the step-by-step movement of the motor axis. When micro-stepping is used, the “normal” step is divided in a number of smaller steps. This results in a significant more smooth movement of the axis.

The power supply for the  $\mu$ STEPPER board is also identical to the PHCC STEPPER board. You can power the logic (+5V) via a separate POWER connector or via the DOA connection cable. Note that the stepper motors also use this power supply, unless you connect a separate power supply and set the MOTPWR jumper accordingly.

You can connect small motors (such as X27 steppers) directly to the motor controller IC outputs. These outputs have diodes to handle back EMF to protect the IC. If you use stepper motors that require more coil current, you must use the outputs of the driver ICs. For each motor there are 4 jumpers to select whether the motor coils are connected to the motor controller or the driver ICs.

The  $\mu$ STEPPER V1.0 circuit board has the following features.

- Small PCB (3.8” x 3.95”), easy to build, but the motor controller is an SMD component
- Generation of micro-step signals for 4 stepper motors
- Software-selectable “home” input for each stepper motor axis
- LED indicators for firmware and motor output (signal monitoring selectable per motor)
- DOA connection or USB connection, jumper selectable

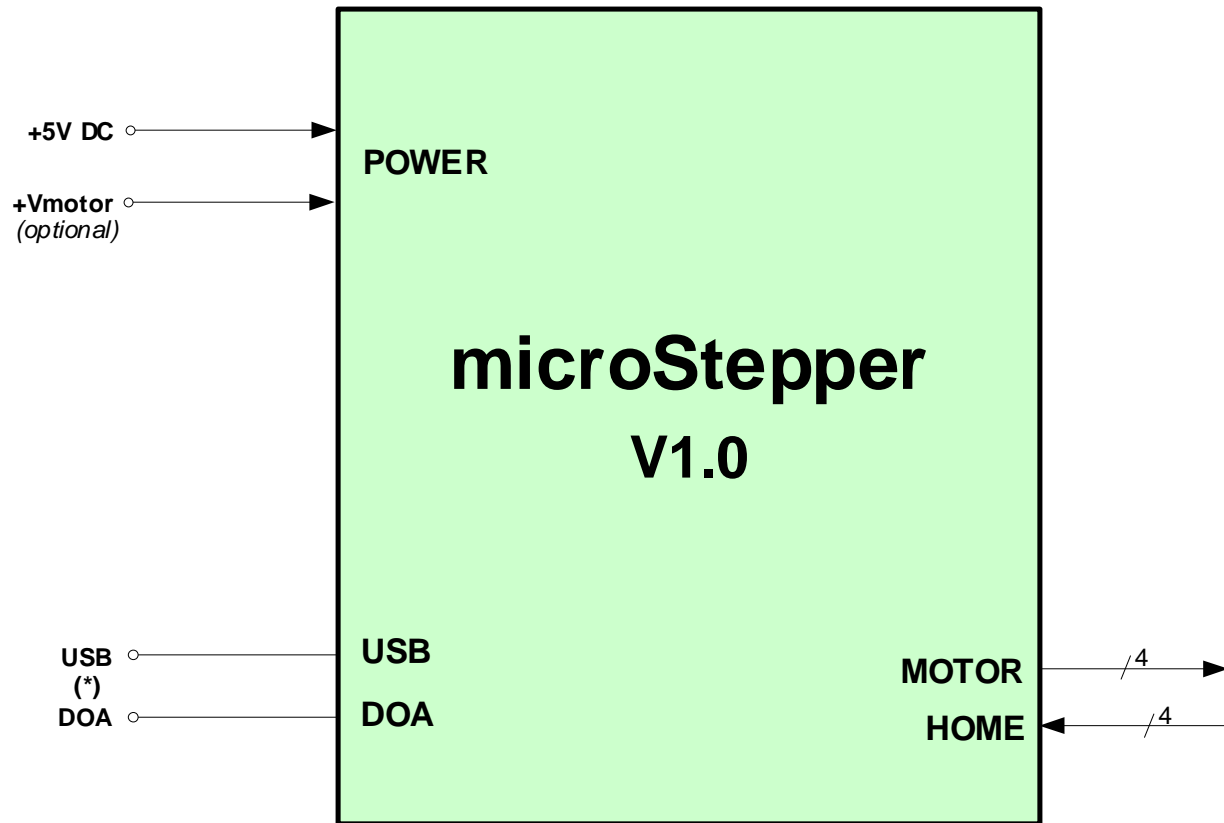
I spent quite some time developing and testing the hardware. Therefore, I put some “copyright” on my work. You are allowed to build and use the  $\mu$ STEPPER design for your own purpose, but you are not allowed to make commercial profit, building and/or selling the work. This manual gives detailed descriptions, because I feel that this information is useful when another device is used. However, it is not the intention to enable easy copying of my work and try to make money. *The  $\mu$ STEPPER is developed by a hobbyist for hobbyists.*

**Acknowledgement** The stepper motor control firmware part is based on the Arduino SwitecX12 library developed by Guy Carpenter.

**Disclaimer** All use of the  $\mu$ STEPPER and all mentioned hardware are solely *your* responsibility. Any damage to the  $\mu$ STEPPER, other hardware or connected instrument(s) is *not* my responsibility. I have tried to remove any typo or error, but you cannot hold me responsible for errors if any error causes a defect.

## 2. Overview of the PHCC $\mu$ STEPPER board

### 2.1 Block diagram



### 2.2 Power supply connections

The “POWER” connector connects the +5V DC power supply to the  $\mu$ STEPPER, and optionally a separate power supply for all connected stepper motors. An on-board jumper selects whether all stepper motors are powered from the +5V logic or the separate motor power supply connection. Further, the +5V for the logic is jumper-selectable taken from the “POWER” connector or from the DOA bus. Note that power via the USB connection is never used.

### 2.3 Motor connections

The  $\mu$ STEPPER board has four 4-pin headers to connect 4 stepper motors. On-board jumpers set whether a stepper motor is connected directly to the controller or via the driver IC outputs.

### 2.4 Interface connections

The  $\mu$ STEPPER has two interface connections, mutual exclusive, either USB or DOA (PHCC). An on-board jumper defines which communication connection is selected at power-up.

### 3. Connections & jumpers on the $\mu$ STEPPER

The  $\mu$ STEPPER board has 7 connectors and 20 jumpers. Pin #1 is indicated on the PCB.

#### 3.1 POWER connector

The POWER connector connects +5V power for the logic and (optionally) a separate power supply for all stepper motors.

pin number	Signal	Usage
1	+5V	+5V (logic) – see 3.10 DOA+5 jumper
2	GND	Ground (0V)
3	+MT	+VM (motor) – see MOTPWR jumper

*→ The  $\mu$ STEPPER has no provisions against wrong power supply connections !*

#### 3.2 M1, M2, M3, M4 connector

The M1, M2, M3, M4 connectors connect 4 stepper motors.

pin number	Signal	Usage
1	Motor1	Motor coil #1 connection 1
2	Motor2	Motor coil #1 connection 2
3	Motor3	Motor coil #2 connection 1
4	Motor4	Motor coil #2 connection 2

#### 3.3 SENS connector

The SENS connector is used as (optional) “HOME” sensor input for each stepper motor axis.

pin number	Signal	Usage
1	Home1	Home sensor for stepper motor axis #1
2	Home2	Home sensor for stepper motor axis #2
3	Home3	Home sensor for stepper motor axis #3
4	Home4	Home sensor for stepper motor axis #4
5	GND	Ground (0V)

#### 3.4 SEL connector

The SEL connector specifies which motor signals are monitored on the diagnostic LEDs.

pin number	Signal	Usage
1	SEL-A	Selection line A
2	SEL-B	Selection line B
5	GND	Ground (0V)

### 3.5 LED connector

The LED connector connects (optionally) the 4 motor signals (for external LEDs).

pin number	Signal	Usage
1	LED signal 1	Diagnostic motor signal 1 (LED must connect via series resistor)
2	LED signal 2	Diagnostic motor signal 2 (LED must connect via series resistor)
3	LED signal 3	Diagnostic motor signal 3 (LED must connect via series resistor)
4	LED signal 4	Diagnostic motor signal 4 (LED must connect via series resistor)
5	+5V	Common power supply for external LEDs

### 3.6 DOA connector

The DOA connector is the standard PHCC DOA 10-pin connector. See also “3.8 RST jumper”.

### 3.7 USB connector

The USB connector is a standard USB Type B connector.

### 3.8 USB/DOA jumper

The USB/DOA jumper selects which communication interface is used.

pin number	Signal	Usage
1 – 2	USE_USB	The USB connector is used for communication
no jumper	USE_DOA	The DOA connector is used for communication

### 3.9 RST jumper

The RST jumper selects what the function is of pin #1 of the DOA connector.

pin number	Signal	Usage
1 – 2	DOA_GND	Pin #1 of DOA connector tied to GND (original DOA specification)
2 – 3	DOA_RST	Pin #1 of DOA connector tied to RST* of PIC Pin 1 of the DOA connector can be used as external reset signal for the PIC. <b>Note that the original ribbon cable from the PHCC Motherboard for DOA cannot be used, because it will keep the PIC in reset.</b>

### 3.10 DOA+5 jumper

The DOA+5 jumper selects the power supply source for the +5V logic.

pin number	Signal	Usage
1 – 2		+5V LOGIC is taken from the DOA bus NOTE: this connection is also wired to the POWER connector
no jumper		+5V LOGIC is taken from the POWER connector

```
graph LR
    POWER[POWER +5V DC] --- Jumper[DOA+5]
    DOA[DOA +5V DC] --- Jumper
    Jumper --- LOGIC[+5V LOGIC]
```

### 3.11 MOTPWR jumper

The MOTPWR jumper selects the power supply source for the motors.+5V logic.

pin number	Signal	Usage
1 – 2		Motors powered from +5V LOGIC pin of the POWER connector
2 – 3		Motors powered from +MT pin of the POWER connector

### 3.12 MOTOR OUTPUT SELECT jumpers

For each motor there are four 3-pin jumpers to select whether the motor coils are connected to the controller IC or the driver IC outputs. Only small motors such as the X27 can be connected to the controller IC (current less than 20 mA). If in doubt what the current can be, drawn by the motor, always select the driver IC outputs. **All 4 jumpers for one motor must be set identical.**

pin number	Signal	Usage
1 – 2	Controller IC	Maximum motor current < 20 mA (specifically for X27 stepper motors)
2 – 3	Driver IC	Maximum motor current < 500 mA



## 4. Assembly of the $\mu$ STEPPER

Only a fine low-wattage soldering iron is required. Read through the steps below to have an idea of the work you are about to do. Take your time to solder the components on the PCB. Better spend a few more minutes working accurately now, than searching for that little solder excess that causes a short circuit or wrong component placement (especially resistors).

Soldering the components in order from smallest height to higher has the advantage that the board lays stable on your desk while soldering, and keeps the component against the PCB. Therefore the following soldering order is proposed. All components are placed on the component side of the PCB. The component side has the white text painted on the PCB (the so-called silkscreen). See the “Appendix A – Component locator  $\mu$ STEPPER circuit board” for reference.

**Observe ESD safety measures to prevent static discharge damage.**  
(This applies to the LEDs, diodes and ICs)

1. Solder diodes D1 thru D32. Observe the orientation. The “bar” is the cathode.
2. Solder all resistors. Make sure that the resistors with different values are in their correct position. Check chapter 7 and Appendix A for reference. If you are not sure that you read the color code correctly, use an Ohm meter.
3. Solder the crystal and the capacitors, except the two polarized capacitors.
4. Solder the LEDs. The anode (longer wire) of the LED is indicated by “+” or “a”.
5. Solder the IC sockets, if you want to use sockets. I always use sockets, not to protect the ICs, but the PCB! If an IC is defective, it can easily be swapped. A soldered IC is difficult to remove and you likely damage the PCB traces or the through-hole plating. The PCB is more valuable than any of the ICs. Make sure that the notch, which indicates pin #1 location, is at the correct side. The silkscreen shows the notch.
6. Solder all pin headers, then the DOA connector (observe pin #1), and the USB connector.
7. Solder the two polarized capacitors. Observe polarity!
8. Solder the SOP28 motor controller package. You really need a fine-tip soldering iron! Position the IC exactly on the soldering pads, observe pin #1 location! Solder the pin at one corner by just heating the pin. Check the position of the IC, are all other pins aligned on their soldering pads? If needed, manually reposition the IC (without soldering the soldered pin). If alignment on the pads looks OK, solder the opposite corner pin (by just heating the pin). Now you can solder the other pins.

Before you proceed, do a visual inspection of the board with a bright light and magnifying glass.

- ✓ Are all soldering joints clean and shiny? A dull soldering joint may be a bad soldered joint.
- ✓ No small droplets of solder near the soldering joints?
- ✓ No short circuit between pins (especially the motor controller IC)?

**TIP** *You can use an old tooth brush to brush off solder residue and tiny solder droplets.*

## 5. Connection wiring

### 5.1 Introduction

This chapter describes the interconnections per identified functionality.

The required connections are the following.

- power supply
- motor(s)
- USB (to PC) or DOA (to PHCC Motherboard)

### 5.2 Power supply wiring

The power supply wiring for the  $\mu$ STEPPER board is simple, just +5V for the logic and optionally a separate power supply for the motors. Check the DOA+5 and MOTPWR jumpers, see chapter 3.10 and 3.11.

Connection	From	To
+5V DC	+5V DC power supply	POWER header
+MV DC (optional)	+Vm motor power supply	POWER header Set MOTPWR jumper accordingly!
GND	power supplies GND	POWER header

⇒ *Always double-check the connections before you switch on power!*

### 5.3 Motor(s) wiring

Each motor has its own connector, M1 for motor #1, M2 for motor #2, etc. Each connector has 4 pins to connect the motor. Check the motor output jumpers, see chapter 3.12.

Connection	From	To
M1	M1 header	Motor #1
M2	M2 header	Motor #2
M3	M3 header	Motor #3
M4	M4 header	Motor #4

### 5.4 Interface wiring

The  $\mu$ STEPPER must be connected to the PC using the USB connection, or must be connected to the PHCC Motherboard using the DOA connection. Set the DOA/USB jumper accordingly, see chapter 3.8.

If you use the PHCC connection you must connect the DOA bus, and optionally the RESET pin (not standard PHCC, see chapter 3.9 for details) from the  $\mu$ STEPPER to the PHCC Motherboard. Optionally,  $\mu$ STEPPER uses +5V from the DOA bus, check DOA+5 jumper, see chapter 3.10.

If you use the USB connection, you need a USB Type B to USB Type A cable to connect the PC. The  $\mu$ STEPPER does *not* use +5V from the USB connection.

## 6. $\mu$ STEPPER commands

If the  $\mu$ STEPPER is used as a PHCC daughter (DOA communication) you must send the DOA address byte, sub-address byte and data byte. The DOA address byte identifies that the data packet sent is for the  $\mu$ STEPPER firmware. The value of the DOA address is hard-coded in the PIC firmware, and can be any value as long as the value is unique on the entire DOA bus. The sub-address and data byte define for which functionality the command and data is intended.

If you use the USB connection it is clear what the destination is, and the address byte is not needed. The USB data packet thus consists of the sub-address and the data byte (in that order).

This chapter describes the implemented commands with their possible data (byte) values. Remember that the  $\mu$ STEPPER is a “listen-only” device, thus if you want to know what data was sent, the sending program must keep a local copy.

### microSTEPPER Move to absolute position / define max number of steps range / acc-dec table values

sub-address	data byte	function / description
0	0x00 ... 0xFF	ABSPOS_0 – set axis position in “range” 0000 – 0255
1	0x00 ... 0xFF	ABSPOS_1 – set axis position in “range” 0256 – 0511
2	0x00 ... 0xFF	ABSPOS_2 – set axis position in “range” 0512 – 0767
3	0x00 ... 0xFF	ABSPOS_3 – set axis position in “range” 0768 – 1023
4	0x00 ... 0xFF	ABSPOS_4 – set axis position in “range” 1024 – 1279
5	0x00 ... 0xFF	ABSPOS_5 – set axis position in “range” 1280 – 1535
6	0x00 ... 0xFF	ABSPOS_6 – set axis position in “range” 1536 – 1791
7	0x00 ... 0xFF	ABSPOS_7 – set axis position in “range” 1792 – 2047
8	0x00 ... 0xFF	ABSPOS_8 – set axis position in “range” 2048 – 2303
9	0x00 ... 0xFF	ABSPOS_9 – set axis position in “range” 2304 – 2559
10	0x00 ... 0xFF	ABSPOS_A – set axis position in “range” 2560 – 2815
11	0x00 ... 0xFF	ABSPOS_B – set axis position in “range” 2816 – 3071
12	0x00 ... 0xFF	ABSPOS_C – set axis position in “range” 3072 – 3327
13	0x00 ... 0xFF	ABSPOS_D – set axis position in “range” 3328 – 3583
14	0x00 ... 0xFF	ABSPOS_E – set axis position in “range” 3584 – 3839
15	0x00 ... 0xFF	ABSPOS_F – set axis position in “range” 3840 – 4095
16	0x00 ... 0xFF	ABSPOS_G – set axis position in “range” 4096 – 4351
17	0x00 ... 0xFF	ABSPOS_H – set axis position in “range” 4352 – 4607
18	0x00 ... 0xFF	ABSPOS_I – set axis position in “range” 4608 – 4863
19	0x00 ... 0xFF	ABSPOS_J – set axis position in “range” 4864 – 5119
20	0x00 ... 0xFF	ABSPOS_K – set axis position in “range” 5120 – 5375
21	0x00 ... 0xFF	ABSPOS_L – set axis position in “range” 5376 – 5631
22	0x00 ... 0xFF	ABSPOS_M – set axis position in “range” 5632 – 5887
23	0x00 ... 0xFF	ABSPOS_N – set axis position in “range” 5888 – 6143
24	0x00 ... 0xFF	ABSPOS_O – set axis position in “range” 6144 – 6399

**microSTEPPER Define use for absolute values (commands 0 – 24)**

sub-address	data byte	function / description
25	0 – 1 – 2 – 3	Define motor for which next absolute position command applies
26	0 – 1 – 2 – 3	Define motor for which next max number of steps command applies
27	special format	Define motor and entry position for user-defined acc/dec table

**microSTEPPER Move relative number of steps (0x00 – 0x7F ≡ cw // 0x80 – 0xFF ≡ ccw)**

sub-address	data byte	function / description
28	0x00 ... 0xFF	STEPREL_M1 – motor #1 step relative cw/ccw x positions
29	0x00 ... 0xFF	STEPREL_M2 – motor #2 step relative cw/ccw x positions
30	0x00 ... 0xFF	STEPREL_M3 – motor #3 step relative cw/ccw x positions
31	0x00 ... 0xFF	STEPREL_M4 – motor #4 step relative cw/ccw x positions

**microSTEPPER Move to home position / use home sensor input**

sub-address	data byte	function / description
32	don't care	HOME_M1 – motor #1 to home position
33	don't care	HOME_M2 – motor #2 to home position
34	don't care	HOME_M3 – motor #3 to home position
35	don't care	HOME_M4 – motor #4 to home position
36	don't care	HOME_ALL – all motors to home position
37	special format	SENS_M1 – motor #1 use home sensor & define active level
38	special format	SENS_M2 – motor #2 use home sensor & define active level
39	special format	SENS_M3 – motor #3 use home sensor & define active level
40	special format	SENS_M4 – motor #4 use home sensor & define active level

**microSTEPPER Motion profile**

sub-address	data byte	function / description
41	special format	VEL_TABLE – default or user-defined acc/dec table per motor

**microSTEPPER Power control**

sub-address	data byte	function / description
42	0 – 1	PWR_L293 – disable (0) / enable (1) L293 power outputs
43	don't care	CTRL_RST – reset microcontroller IC

**WATCHDOG functionality**

sub-address	data byte	function / description
44	(*)	Disable watchdog functionality -- See text for description.
45	0x00 ... 0xFF	<b>Watchdog control</b> -- See text for description.

## MISCELLEANOUS

sub-address	data byte	function / description
46	0 – 1 – 2 – 3 – 4	<b>DIAG</b> LED operation mode 0 – LED always OFF 1 – LED always ON 2 – LED flashes at heart beat rate (power-up default) 3 – LED toggles ON/OFF state per <i>accepted</i> command 4 – LED is ON during DOA packet reception
47	don't care	<b>IDENTIFY</b> USB only: send identification “uSTEP vA.B \$xy”
48	‘N’ or ‘Y’	<b>USB</b> debug command (USB only ☺)

## 6.1 How to control a µSTEPPER motor

### 6.1.1 Homing a stepper motor

When power is applied, the position of a stepper motor is not known (if no absolute encoder is attached to the axis). Thus, before you can control a stepper motor, you must determine somehow the position of the axis. This is called “homing”. If the motor axis does not have an absolute encoder attached, you can home a stepper motor by stepping in one direction until the axis runs into a mechanical end stop, or some simple sensor detects the “zero” position. Both methods are supported by the µStepper. Note that for high-torque stepper motors stepping against an end stop may require a strong construction.

With the commands #32 - #36 a stepper motor can be homed. If no home sensor is defined using commands #37 – #40, the stepper motor steps the maximum number of steps specified in the counter-clockwise direction. This works fine if the maximum number of steps represent the full scale movement of the axis. No matter what the position of the axis is, after stepping the maximum number of steps the axis is guaranteed to be at “zero” position. With command #26 followed by a command #0 – #24 you can define the maximum number of steps for a motor. The data byte of command #26 specifies the motor. Possible values are 0, 1, 2, or 3, where 0 is motor #1, etc. The default maximum value is 3780, which is correct for the X27 stepper motor. Note that when the maximum number of steps is changed, the motor must be homed again.

With the commands #37 – #40 you can specify that a stepper motor has a reference position sensor. The data byte of these commands specify two properties, a reference sensor is used or not used, and if a sensor is used its active level. The low nibble (bits 3-0) can be either “0000” meaning “no reference sensor used for homing”, or “0001” meaning “reference sensor used for homing”. The high nibble (bits 7-4) can be either “0000” meaning “reference sensor active level is logic 0”, or “0001” meaning “reference sensor active level is logic 1”. When a home command is issued (commands #32 - #36), the stepper motor steps in the counter-clockwise direction until the specified active level of the reference sensor is detected.

After the stepper motor is homed, its axis position is set to 0. This implies that when you use a reference sensor, you can never move (counter-clockwise) beyond the “zero” position.

### **6.1.2 Moving a stepper motor to an absolute position**

With commands #0 – #24 you can move a stepper to an absolute position. Note that any axis movement can only be done *after* the stepper motor is homed. If the stepper motor has not been homed, all movement commands will be ignored. With command #25 you can specify for which stepper motor the next absolute position command applies. The data byte of command #25 can be 0, 1, 2, or 3, where 0 is motor #1, etc. As long as you do not change the “selected” motor (with command #25), you do not need to send command #25 every time a new absolute position for the same motor must be set. After any motor is homed, the “selected” motor is set to “none”.

### **6.1.3 Moving a stepper motor to a relative position**

With commands #28 – #31 you can move a stepper motor to a relative position (from its current position). Note that any axis movement can only be done *after* the stepper motor is homed. If the stepper motor has not been homed, all movement commands will be ignored. The movement can be in positive (clockwise) or negative (counter-clockwise) direction. The data byte of the “move relative” command uses the low 7 bits to specify the number of steps (thus a maximum of 127), and bit 8, the most significant bit, specifies the movement direction, where “0” is clockwise, and “1” is counter-clockwise. If the motor is already moving, the specified number of steps is applied to the position the motor was moving to. Depending on the current movement direction and the specified relative movement, the new end position maybe reached sooner (movement shortened) or later (movement lengthened).

Note that for relative movements the specified motor is directly “addressed” and the maximum count is positive or negative 127 steps. Another very important aspect is that with relative position stepping the risk losing the actual axis position is higher. Assuming that the motor does not loose steps, if you move to an absolute position, the motor will move to that position. However, if for some reason a relative movement command is lost, the application will no longer know the axis position needed for a subsequent “move relative” command.

### **6.1.4 X27 stepper motor specifics**

Default, an X27 motor needs 3 steps to rotate 1 degree. Without “custom” modification the motor can rotate 315 degrees. Thus  $3 * 315 = 945$  steps are needed for the full rotation. However, the micro-stepping of the motor controller adds 4 micro-steps per “normal” step, thus in total  $4 * 945 = 3780$  micro-steps are needed for one full rotation. As other motors may need other number of steps, you can specify the total number of micro-steps needed for one full rotation. This maximum number of steps can be defined for each stepper motor separately. To change the maximum number of steps of a motor, you use command #26 specifying the motor, followed by a command #0 – #24. Note that the motor must be homed again.

### 6.1.5 Stepper motor movement profiles

You can move the axis of a stepper motor by simply sending step pulses to the controller. In certain applications that is exactly what you want: a linear movement at a constant speed. When a stepper motor is used to drive the needle of an indicator that constant movement is not “normal”. Especially when the indication change is “large”, a needle would start a fast movement and slow down as it reaches the specified “setpoint”. However, you cannot step a stepper motor at a high rate from stand-still. The axis must have some velocity before you can increase the stepping speed. Likewise, you cannot abruptly stop a stepper motor when stepping at high speed. You must slow down before stopping the steps. If you move too fast or stop too fast, the stepper motor will lose steps, or will not move at all. In either case, you no longer know the axis position.

The  $\mu$ Stepper uses an acceleration/deceleration table to start and stop a movement, and provides a motion profile so that, depending on the number of steps to move, a part of the movement is executed at “high speed”. The default acceleration/deceleration table in the firmware works fine for the X27 motor, but for fine-tuning and for other stepper motors (different inertia, thus different acceleration/deceleration motion profiles), you can upload a user-defined acc/dec table, and subsequently specify that the user-defined acc/dec table must be used for motion profiles.

With command #27 you can upload a user-defined acc/dec table. Note that you can only have one user-defined acc/dec table for each motor. The data byte of command #27 specifies for which motor the acc/dec table the definition applies, and the specific table entry. The low nibble (bits 3-0) specifies the motor (“0000”–“0001”–“0010”–“0011” for motor #0–#1–#2–#3 respectively). The high nibble (bits 6-4) specifies the entry number in the acc/dec table, and bit 7 (the most-significant bit) specifies whether the data is a step value or a delay value for the table entry.

The acc/dec table has 5 entries. Each entry contains two values, the stepcount (velocity) and the time delay between two steps. This is the default table:

20	750
50	325
100	150
150	120
300	90

The first entry is the step speed, the second entry is the delay. The consecutive first entry of each row must be higher than the previous row. To upload a table entry, for example, to change the “50” entry of the table the high nibble should be “0001”, and to change the “325” entry of the table the high nibble should be “1001”. The subsequent command #0 – #24 stores the value of this command in the specified table cell.

With command #41 you specify whether the default acc/dec table is used or the user-defined acc/dec table is used. Note that there is no check when you use the user-defined acc/dec table whether an acc/dec table is actually uploaded! The data byte of command #41 specifies for which motor the acc/dec table the definition applies, and whether the default acc/dec table or the user-defined acc/dec table is used. The low nibble (bits 3-0) specifies the motor (“0000”–“0001”–

“0010”–“0011” for motor #0–#1–#2–#3 respectively). The high nibble (bits 7-4) can be either “0000” meaning “use the default acc/dec table, or “0001” meaning “use the user-defined acc/dec table”.

## **6.2 Power control**

Command #42 enables or disables the power outputs of the L293 drivers. If you use X27 stepper motors directly connected to the stepper controller IC (see chapter 3.12), the L293 drivers are not used, so they can be disabled (default state after power-up or reset). If you use larger stepper motors and must use the driver outputs, you must issue a #42 enable command otherwise the stepper motor(s) will not be driven.

Command #43 will generate a reset signal only to the stepper motor controller IC. Normally you never need to reset the controller IC. Note that if you reset the motor controller IC, you must home all connected stepper motors again, because their position is no longer valid (unknown).

## **6.3 IDENTIFY command**

The IDENTIFY command (#47) is only available when the USB communication connection is used. When the IDENTIFY command with an “any value” data byte is sent, the  $\mu$ STEPPER returns a message in the form of “ $\mu$ STEP vA.B \$xx” followed by a CR (0x13) and LF (0x0A). The “A” and “B” represent the major and minor version number of the firmware. “\$xx” is the hexadecimal device code of the  $\mu$ STEPPER firmware, and used for recognition of DOA PHCC commands. Although the DOA device code is irrelevant for the USB communication, it can still be useful. Suppose you have more than one USB-connected PHCC device (for example the SDI boards of the ADI or the altimeter). Using the IDENTIFY command you can find out which specific instrument is connected to a virtual COM port, because the device code (should be) unique. Note that because of this, the firmware for each PHCC device is unique, but that is always the case with PHCC DOA.

## **6.4 USB debug command**

The USB debug command (#48) enables or disables debug output sent to the USB connection. Obviously, this command only operates when you use a USB connection. Default USB debug output is disabled. To enable USB debugging output you must send the command with the data byte ASCII character ‘Y’. Likewise, to disable the USB debugging output send the command with the data byte ASCII ‘N’. When USB debug is enabled a specific character is returned every time a byte is received via the USB connection, followed by the data byte itself (in hexadecimal format). Depending on the interpretation (by the firmware) of the data byte one of the following specific characters is returned leading the echoed (hexadecimal) data byte.



returned character	SDI interpretation
–	following 2 characters are hexadecimal command (first byte)
=	following 2 characters are hexadecimal data (second byte)
X	"disable watchdog" command received
#	invalid sub-address in command
!	USB receive data state machine in illegal state (firmware error ☹)

## 6.5 Watchdog functionality

The watchdog functionality makes sure that the data communication stays “synchronized”. Commands sent via the PHCC DOA channel are 3 “bytes” (device address, sub-address and data byte, where the sub-address is 6 bits!). Commands sent via the USB channel are 2 bytes. The firmware uses a state machine to keep track of the received data bits and bytes. If, due to some external disturbance, one byte is not correctly received the receive routine may treat the next byte as the first, because the state machine is in the wrong state. If the firmware gets in this state, all subsequent commands received will be wrong. This condition is not recognized by the sender (PHCC Motherboard or the PC), but you will notice that all functionality of the  $\mu$ STEPPER no longer seems to work (because the “commands” are wrong). You could say that the firmware has become “deaf”.

To solve this problem a so-called watchdog timer is implemented in the firmware. In a normal condition the bytes sent via USB that belong together (they form the command) are sent one after the other without much delay. Every time a byte is received the watchdog timer is reloaded to some value. Every millisecond the value is decreased by one. When the watchdog timer reaches 0, the firmware state variables that control the data reception are reset to the initial state. Thus, as long as a command transmission is active the watchdog will not expire. If there is some (predefined) time no communication activity the watchdog will expire and by resetting the state variables that control the data reception, the communication channel is forced to be “in sync”.

For DOA the story is similar, but here the data *bits* are sent one after the other, and the watchdog timer is reloaded at every received *bit*. If some disturbance causes the communication to go out of sync, it is corrected as soon as a short pause between commands occurs. That the mechanism works is proved by “deaf” PHCC daughter boards after power-up. Without the watchdog functionality, they remain “deaf”, but with the watchdog functionality they work OK. (Power-up can cause spurious pulses which can mistakenly be detected as data bits).

However, experiments have shown that this watchdog solution is not perfect either ☹ To solve the most common DOA communication problem at power-up start, a one second delay is implemented if DOA communication is selected. After the one second delay it is assumed that all noise spikes on the DOA communication lines (specifically clock) are gone, and the state machine is initialized. During the one second delay the diagnostic LED flashes at a high rate.

The watchdog functionality is by default disabled. If you are *manually* testing commands (for example using PuTTY), data bytes that belong to each other to form a complete command using the USB connection are transmitted with delay between them. If the watchdog mechanism is enabled you will never succeed sending a complete command, because the time delay between the first and second data byte is sufficient to let the watchdog do its work. Thus, your second data byte will be treated as if it is the first data byte of a next command. For this reason, it must be possible to disable the watchdog. But here you get into a “chicken – egg” problem.

The watchdog command itself consists of 2 data bytes. As long as the watchdog is active, a manually sent command to disable the watchdog will fail, because of the described watchdog action. Therefore, the “disable watchdog” command is “special command”, because it does not require the data byte. This solution only works for USB as it will make the command a single byte. Sending commands via the USB connection with a PC terminal communication program (for example PuTTY) will work fine. For DOA the solution will not work. However, if you use the PHCC TestTool for testing via the DOA connection, the watchdog activity will not be a problem. (Note that with the PHCC TestTool a message always consists of 3 data bytes).


A separate command is available to define the watchdog timer count-down value, and optionally, enable/disable the watchdog. The data format is as follows.

7	6	5	4	3	2	1	0
<b>ENA</b>	<b>0</b>	6 data bits representing the watchdog count-down value					

Bit 7 (“ENA”) controls whether the watchdog is enabled or disabled. When bit 7 is ‘1’ the watchdog is enabled. When bit 7 is ‘0’ the watchdog is disabled. So, you can disable the watchdog with this command and you can disable the watchdog with the special “disable watchdog” command. The 6 bits count-down value allows a less strict setting than default set by the firmware (8). If the 6 bits are all zero (‘000000’), the firmware default value is set. Note that you can set the count-down value also when the watchdog is disabled (“ENA” bit is ‘0’).

## 7. Parts list $\mu$ STEPPER circuit board

Quantity	Component	Description
1	IC1	PIC 18F2550
1	IC2	AX1201728SG
4	IC3, IC4, IC5, IC6	L293D
2	IC7, IC8	CD4052
1	IC9	CD4049
7	IC socket DIL, 16-pin	high-quality machined pin socket
1	IC socket DIL, 28-pin	high-quality machined pin socket
4	LED	3 mm, any color (motor monitor)
1	LED	3 mm, any color (diagnostic)
32	D1 ~ D32	1N4148
1	Q1	20Mhz crystal
5	C1, C5, C8, C9, C10	100 nF
2	C2, C3	18 pF
1	C4	220 nF
2	C6, C7	47 $\mu$ F (polarized) 25V
8	R1, R3, R4, R5, R8, R9, R10, R11	10k
1	R6	1k
1	R7	100k
5	R2, R12, R13, R14, R15	390 $\Omega$
2	USB/DOA, DOA+5	2-pin male header
4	RST, SEL, POWER, MOTPWR	3-pin male header
4	M1, M2, M3, M4	4-pin male header
2	LED, SENS	5-pin male header
8	A1, A2, A3, A4, B1, B2, B3, B4	3-pin male header
8	X1, X2, X3, X4, Y1, Y2, Y3, Y4	3-pin male header
1	DOA	2x5pin male header with shroud
1	USB	USB Type B
1	PCB	microSTEPPER V1.0

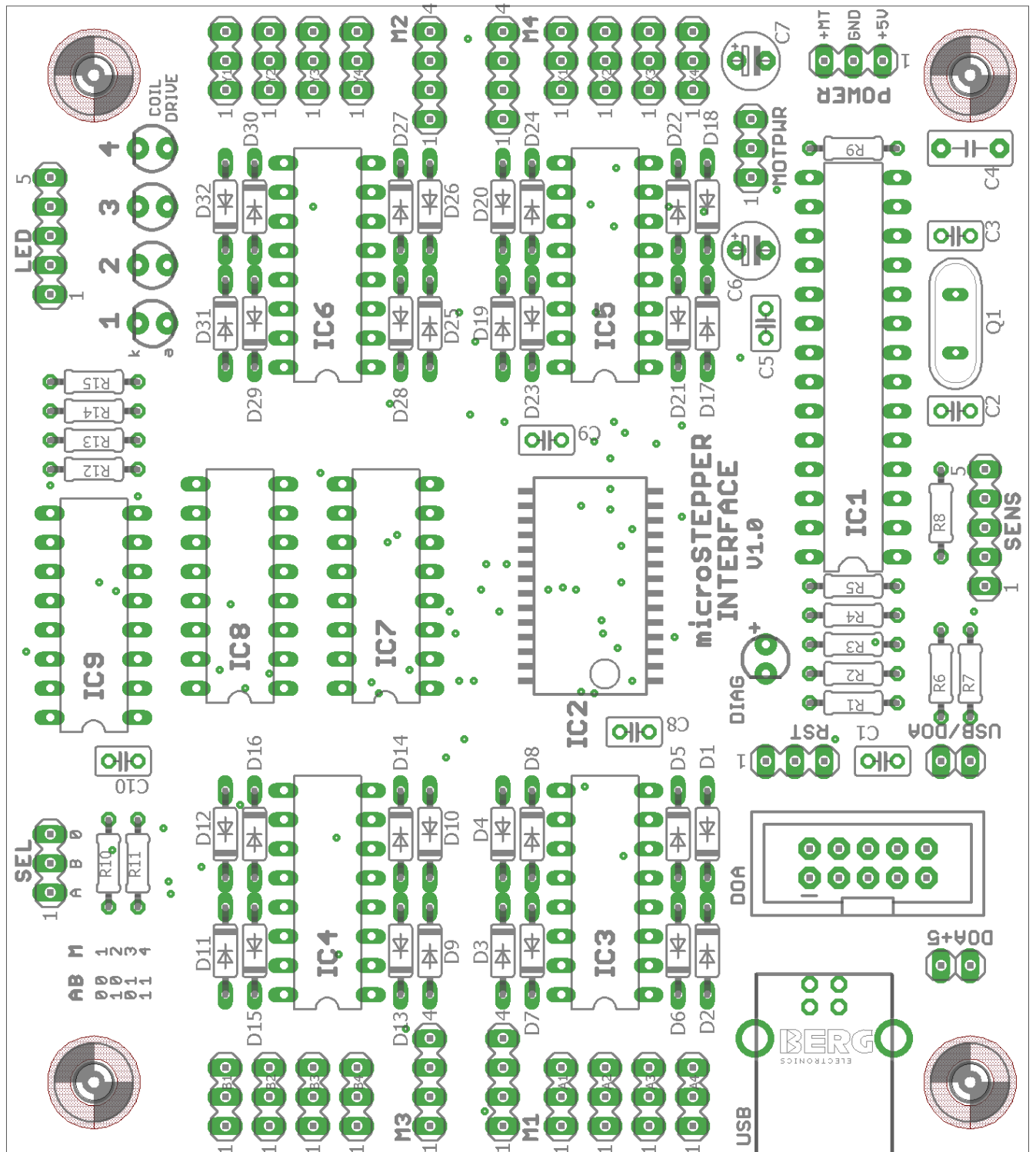
 Component values may change without notice.

❶ The following components are only for “monitoring” purposes and can be omitted.  
R10, R11, R12, R13, R14, R15, IC7, IC8, IC9, SEL header, LED header, 4 monitor LEDs.

❷ If only X27 stepper motors are used, IC3, IC4, IC5, IC6 can be omitted.

## Appendix A. Component locator $\mu$ STEPPER circuit board

This image is retrieved from the actual Eagle .BRD file. For clarity all PCB traces are hidden.



## Appendix B – $\mu$ STEPPER Demonstrator program

I wrote a test application in Python. The language allows for fast program development and with available libraries you can create programs with a user-friendly graphical interface. I do not pretend my Python program is well-written – it gets the job done ☺

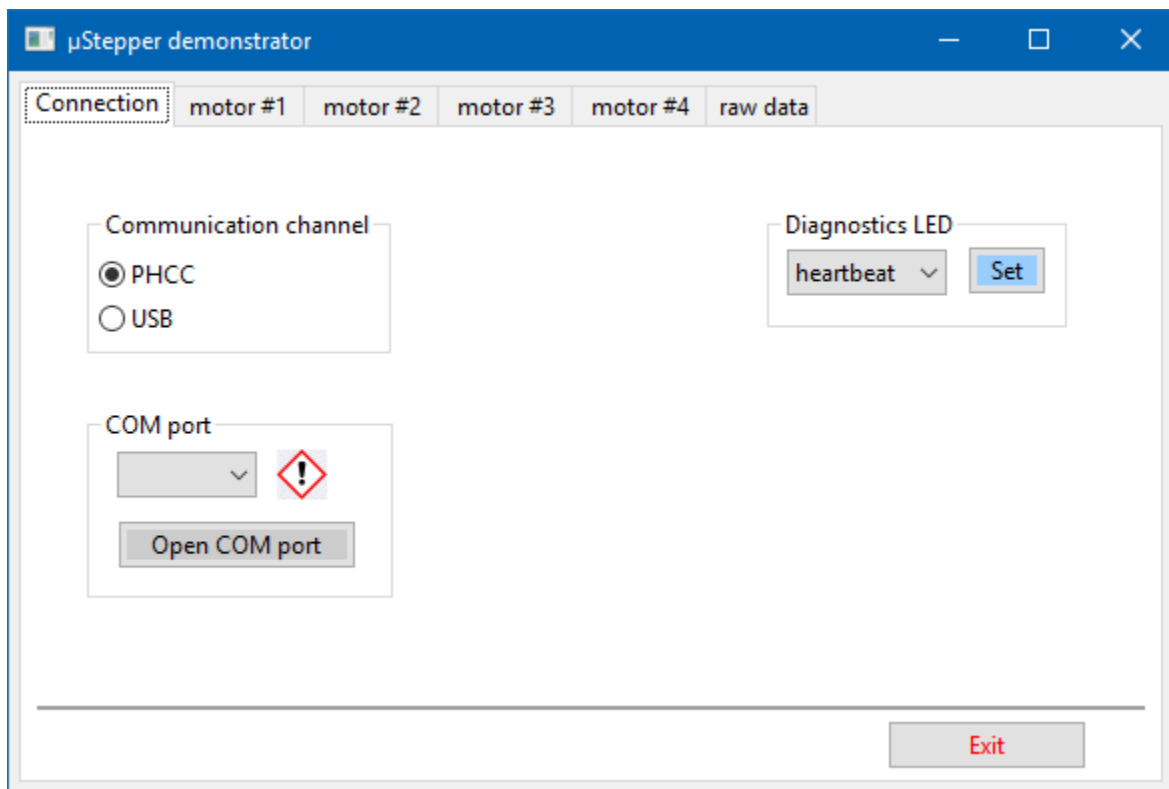


ustep-testappl.zip

The “Package” is a zip file that contains the Python application “ustep-testappl.py” and a small “error.gif” icon used by the application. Extract and put the Python and GIF file together in a folder. Open the file “ustepper.py” with a text editor (notepad is fine). In the first lines you can read a description of the steps you must do to install the Python environment, wxPython and the Python serial package, needed to run the  $\mu$ STEPPER test application. After the installations you can double-click the .py file to start the application, if all went well ☺.

- Before you start the application program, you must have the interface connected to the PC. That is either a connection to the PHCC Motherboard or a USB cable.

Two windows appear when the program starts. The CMD window, that shows diagnostic output, and the application window with 6 tabs.



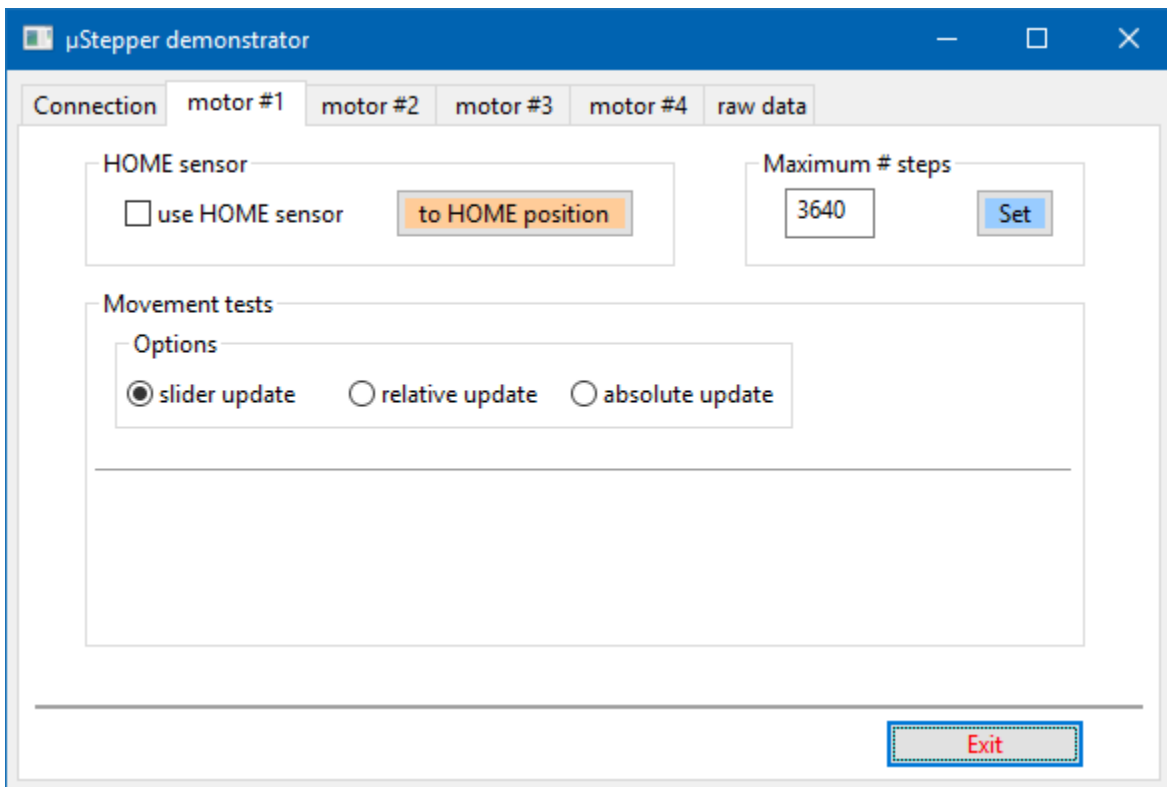
## Connection tab

Default, the program starts with the “Connection” tab. The first action you must do is tell the program how you connect to the  $\mu$ STEPPER interface. This can be using the PHCC Motherboard (DOA cable) or a USB connection cable. This is selected with the radio buttons in the section “Communication channel”. Below that section you can select the COM port which is used to connect (either DOA or USB). You can see in the diagnostic window which COM ports the program detected, but you can also check the Windows Device Manager under “Ports (COM & LPT)”. As long as you do not select a COM port the error sign will stay up blinking as a warning that you must select the COM port. When a COM port is selected, the error sign disappears and the color of the “Open COM port” button changes from grey to green. Click the “Open COM port” button to open the communication channel. The “Open COM port” button legend changes to “Close COM port” button. If you click the button again, the communication port will be closed and the connection is thus terminated.

In the section “Diagnostic LED” you can set the functionality of the DIAG LED on the microSTEPPER board. You can select the options “OFF”, “ON”, “DIAG” and “ACK”. “OFF” means that the DIAG LED is not used and it is turned off. Likewise, “ON” means that the DIAG LED is not used and it is turned on. “DIAG” means that the DIAG LED blinks at a certain rate to indicate that the firmware is running. “ACK” toggles the state (on or off) of the DIAG LED per received command.

## Motor #1, Motor #2, Motor #3, Motor #4 tab

Each tab has the same functionality for a specific motor, so only the Motor #1 tab is described.

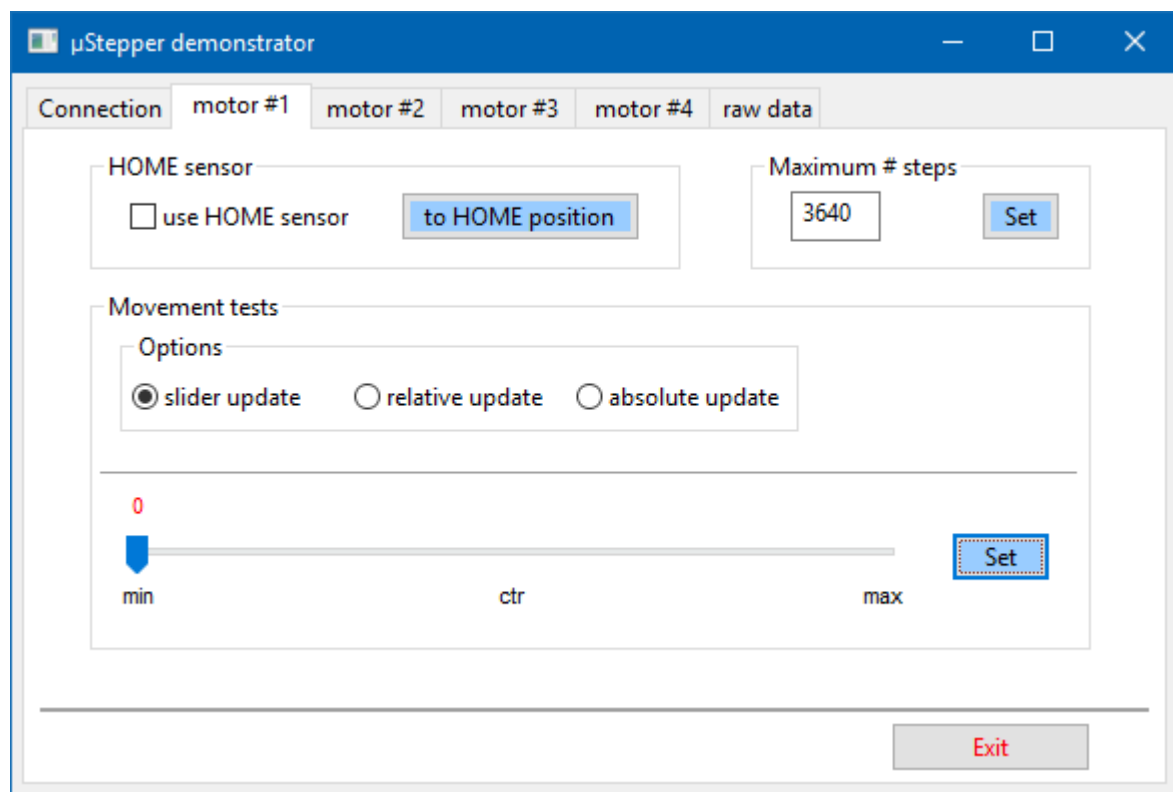


Initially, the “to HOME position” button blinks, changing color between orange and blue. This is an indication that the stepper motor is not yet homed (meaning that the axis position is unknown). When you click the “to HOME position” the motor is homed to its “zero” position. The home position is reached by slowly stepping the motor counter-clockwise. *How* the home position is detected can be selected with the “use HOME sensor” checkbox. If the checkbox contains a checkmark, the firmware checks the state of the sensor input for the specific motor. You need a sensor that will be activated when the axis position of the stepper motor is at “zero” position. If the checkbox does not contain a checkmark, the sensor input is not used, but instead the motor is stepped for the number of steps that is set in the “Maximum # steps” section.

When a motor is homed, but you change the setting of the “Maximum # steps”, the result will be that the motor is no longer homed, and the “to HOME position” button starts blinking again.

After the stepper motor is homed, you can define how you change the axis position of the stepper motor in the “Movement tests” section with the radio buttons “slider update”, “relative update”, and “absolute update”.

### ***“slider update”***



When the “slider update” radio button is selected, a slider appears in the lower part of the “Movement test” section. You can drag the slider from “min” to “max” to any position. The red number above the slider shows the position the axis will move to when you click the “Set” button.

*“relative update”*

The screenshot shows the µStepper demonstrator software window. It has a blue title bar and a tabbed interface with tabs for 'Connection', 'motor #1', 'motor #2', 'motor #3', 'motor #4', and 'raw data'. The 'motor #1' tab is active. Inside the tab, there are several control sections: a 'HOME sensor' section with a checkbox for 'use HOME sensor' and a 'to HOME position' button; a 'Maximum # steps' section with a text box containing '3640' and a 'Set' button; a 'Movement tests' section containing an 'Options' box with three radio buttons: 'slider update', 'relative update' (which is selected), and 'absolute update'; a 'Direction' box with 'cw' (selected) and 'ccw' radio buttons; a 'Relative position update' section with a text box containing '0' and a 'Set' button; and a 'position' indicator showing a red '0'. At the bottom right, there is an 'Exit' button.

When the “relative update” radio button is selected, two boxes and the axis position indicator appears. With this position update option you specify the movement direction in the “Direction” box. The direction can be “cw” (clockwise) or “ccw” (counter-clockwise). The number of steps that the axis must move is specified in the “Relative position update” box. When you click the “Set” button, the motor will move the specified number of steps from its current position. The red number below “position” shows the actual motor position.

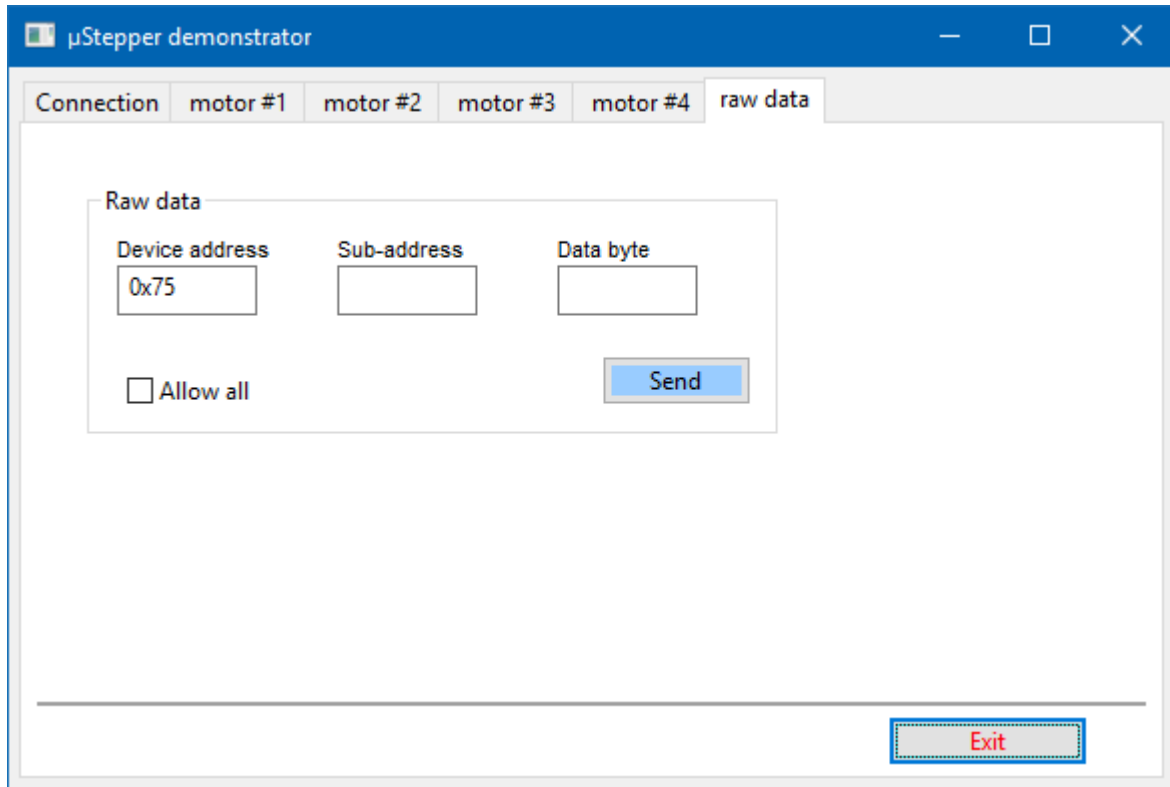


*“absolute update”*

The screenshot shows the 'μStepper demonstrator' application window. It has a blue title bar and a tabbed interface with tabs for 'Connection', 'motor #1', 'motor #2', 'motor #3', 'motor #4', and 'raw data'. The 'motor #1' tab is active. Inside the tab, there are two main sections. The first section, 'HOME sensor', contains a checkbox labeled 'use HOME sensor' which is unchecked, and a button labeled 'to HOME position'. The second section, 'Maximum # steps', contains a text input field with the value '3640' and a 'Set' button. Below these is a 'Movement tests' section with a sub-section 'Options' containing three radio buttons: 'slider update', 'relative update', and 'absolute update'. The 'absolute update' radio button is selected. Below the 'Options' section is a horizontal line. Under this line is the 'Absolute position update' section, which includes a text input field with the value '0' and a 'Set' button. To the right of this section, the word 'position' is displayed in blue text, and below it, the number '0' is displayed in red text. At the bottom right of the window is an 'Exit' button.

When the “absolute update” radio button is selected, the “Absolute position update” box and the axis position indicator appears. With this position update option you specify the end position to which the motor must move. When you click the “Set” button, the motor will move the specified position. The red number below “position” shows the actual motor position.

## Raw data tab



The screenshot shows a window titled "µStepper demonstrator" with a blue header bar. Below the header is a tabbed interface with five tabs: "Connection", "motor #1", "motor #2", "motor #3", and "raw data". The "raw data" tab is selected and active. Inside this tab, there is a sub-section titled "Raw data" which contains three input fields: "Device address" (pre-filled with "0x75"), "Sub-address", and "Data byte". Below these fields is a checkbox labeled "Allow all" and a blue "Send" button. At the bottom right of the window, there is a red "Exit" button.

On the “Raw data” tab you can specify any command in the “Raw data” box. The edit field “Device address” defines the device address for which the command is intended. If you use a USB connection, this byte is not sent; it is only relevant if you use PHCC. The device address is the first byte of a DOA command, and effectively defines which PHCC daughterboard must react on the received message. The “Sub-address” and “Date byte” edit fields are used for USB and for DOA. These two bytes define the command and, if applicable, the data required by the command. As this is the test application for the µSTEPPER, the “Device address” edit field is preloaded with the device address of the microSTEPPER interface (ASCII lowercase letter “u”, 75 hexadecimal). When you click the “SEND” button the data is sent to the microSTEPPER interface.

If data entered in an edit field is not correct an error sign appears. Specify a correct value and click the “Send” button. If the data is correct, the error sign disappears and the command is sent.

If you want to send a command to another device you must change the “Device address” value. Default, the test program only accepts the address of the microSTEPPER interface, and if you change that value an error sign appears. To allow a different device address you must put a checkmark in the “Allow all” checkbox.

With the “Exit” button you terminate the application.