Peter Norberg Consulting, Inc.

Professional Solutions to Professional Problems

P.O. Box 10987 Ferguson, MO 63135-0987 (314) 521-8808

PRELIMINARY
Relay Control when using GenStepper Firmware

Version 0.02

By

Peter Norberg Consulting, Inc.

Table of Contents

Overview

This document describes a technique that allows GenStepper based boards to be used to control one or two relays at the same time as they control one stepper motor. This technique was suggested by Sass Attila from Hungary, and is being made available to all as a general application note.

As a warning, although this technique works for inductance-based loads (i.e., relays), it does not work for driving fast-response circuits. Due to a timing detail in the firmware, when a winding is "on", there is an 8 microsecond "off" period introduced once every 2048 microseconds. Relays will ignore this "off" period, due to the inductance of the windings and due to the mechanical delays involved in release of a magnetic connection; electronic switches, however, will "feel" this off period, and will trigger.

The revision history of this manual is as follows:

Version	Date	Description
0.00	August 3, 2003	First preliminary copy
0.01	August 9, 2003	Added notes about pulses on relay lines
0.02	August 20, 2003	Corrected minor spelling errors

If you have any problems which this manual does not appear to be able to resolve, please do not hesitate to contact us at our customer support email address or via the telephone.

Our customer support email address is:

(support@stepperboard.com)

Our phone numbers are:

U.S. Toll Free: 877-230-5270

International: 314-521-8808

Theory

If only one motor needs to be controlled, and double-current mode is not required, then the drivers for the "other" motor become available for use in operating one or two relays. The general gist of the strategy is to issue motor motion commands on that "other" motor (which is actually wired as the relays) which result in the desired relay behaviors. As an added bonus, from 2 to 4 TTL input lines may become available for generic input events monitoring.

As a warning, although this technique works for inductance-based loads (i.e., relays), it does not work for driving fast-response circuits. Due to a timing detail in the firmware, when a winding is "on", there is an 8 microsecond "off" period introduced once every 2048 microseconds. Relays will ignore this "off" period, due to the inductance of the windings and due to the mechanical delays involved in release of a magnetic connection. Electronic switches, however, will "feel" this off period, and will trigger.

When viewed at the ½ step level (which is the level at which the code performs major state changes), the order of enabling current to the windings on the motor drivers is as follows:

Step	WB2	WB1	WA2	WA1
0	0	0	0	1
1	0	1	0	1
2	0	1	0	0
3	0	1	1	0
4	0	0	1	0
5	1	0	1	0
6	1	0	0	0
7	1	0	0	1

If we wire the relays to the WB1 and WA2 lines (one side to WB1 or WA2, the other to GND for any of the BiStep boards, or to +Vm for the SimStepA04 or SS0705 boards), then we get the sequence:

Step	WB1	WA2
0	0	0
1	1	0
2	1	0
3	1	1
4	0	1
5	0	1
6	0	0
7	0	0

From this, we can determine that there are 4 key locations, vis:

- 0, 6, or 7: Both relays off
- 1 or 2: Relay B on, relay A off
- 3: Both relays on
- 4 or 5: relay B off, relay A on

Superficially (ignoring issues of microstepping), we could therefore just issue commands as "0G" to set both relays off, or "3G" to set both relays on. However, we get into issues related to undesired setting of relays, if we were to go from location 5 (A on, B off) to 0 (both off).

The fix is to recognize that the locations are all modulus 8: that is to say, you simply go in the correct direction to get to the desired state from the current state. We shall initialize by going first to location –1 (which is modulo-equivalent to location 7), and then using the modulo locations as follows:

- 7: b off, a off
- 1: B on, a off
- 3: B on, A on
- 5: b off, A on

Observe that this pattern has each key location offset from its neighbor by 2, thus giving us consistent deltas. From here on, we use the "S" (seek) command for all requests to change relay values, and we maintain the current logical location strictly via the modulus number:

Loc	***7***	***1***	***3***	***5***
7	0	+2	-4	-2
1	-2	0	+2	-4
3	-4	-2	0	+2
5	+2	-4	-2	0

When maintaining this in an array, it can be confusing. Therefore, it is often easier to use a table whose labels are bit-related to relays being on or off. This can be done by assigning bit 0 (a decimal value of 1) to "Relay B is ON", and bit 1 (a decimal value of 2) to "Relay A is ON". This generates a table of relay on/off values of:

Value	***Loc***	**A**	**B**
0	*7*	Off	Off
1	*1*	Off	On
2	*5*	On	Off
3	*3*	On	On

Merging the above two tables (and reorganizing for easy access) gives us the delta values of:

Value	**0**	**1**	**2**	**3**
0	0	+2	-2	-4
1	-2	0	-4	+2
2	+2	-4	0	-2
3	-4	-2	+2	0

This means, for example, that to go from both relays off (a logical value of 0) to both relays on (a logical value of 3), we need to step –4 half-steps.

Note that we need to disable the limit switch detection for the motor driver (X or Y) which is being used to operate the relays. Otherwise, we could end up with new relay commands being ignored, due to the status of the + or – limit switches associated with the driver. As a side effect, those limit switches (for example +LY and –LY) become available for generic low speed input monitoring.

Relay Control Implementation Details

In order to actually implement the relay technique, we need to set up the motor parameters for the relay-driving "motor" system to match the requirements for operating a relay.

1. Select the correct "motor" which is really driving the relays ("X" or "Y", as needed)
2. Set ½ step mode for winding updates ("1o"), so no attempts at microstepping are done
3. Set the step rate to the maximum value ("62500r")
4. Set the "stop ok" rate to the maximum value ("62500k")
5. Force the windings to always be on at full power ("1w")
6. Calculate the size of a ½ step, in current microstep units. By default, at power on to the system, this value is 8 (8 microsteps per half-step, or 16 microsteps per full step)
7. Tell the motor to go back ½ step by issuing a minus seek of ½ step. For example, if the microstep size is $1/16^{th}$ of a full step, then this would be "-8s".
8. Set your current "logical" location to 0 (both windings are off).
9. At the extreme least, disable limit switch detection for the motor driver which is being used as the relay control. To disable detection on the X driver, issue the command "12t". To disable detection on the Y driver, issue the command "3t". To disable ALL limit switch detection (both X and Y drivers), issue the command "15t".

From now on, whenever you need to change relay A or relay B's state, you need to:

1. Save the current state (0 to 3), as your "from state" value.
2. Set or clear the appropriate bit(s) to decide your new state (i.e., clear bit 0 in your state to turn off relay B)
3. Use the above table to look up the scale factor for the "distance" to travel to get to the new state
4. Multiple the result by the ½ step size. For example, if the table reports that you need to go –4 ½ steps, and it takes 8 microsteps per ½ step, then you will need to go (-4 * 8) or -32 steps to go to the new state.
5. Issue the "seek relative" command on the appropriate motor to generate the correct relay values. In the above example (-4 ½ steps) implemented on motor Y as the relay driving motor, you would issue the command "y-32s".
6. Your new state value is (the 0-3) is the value to be used for your next relay action start information.

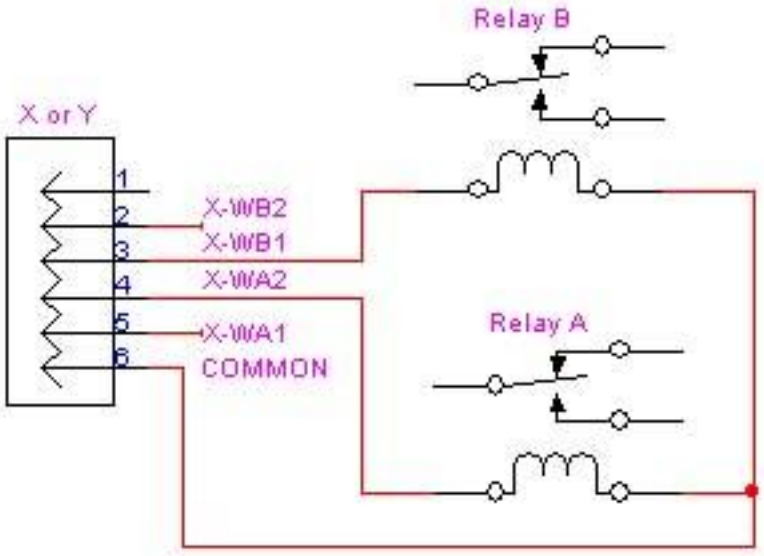
The next section shows how to wire the relays.

Relay Wiring via the X and Y connectors

These two identical connectors are used to operate either one motor or a pair of relays (each). When they are used to wire relays with the GenStepper firmware, they are wired as follows for all products except the BiStepA04. Note that the pins are labeled counting from the top to the bottom, as they appear on the board.

Pin	Name
1	GND or +V
2	WB2
3	WB1
4	WA2
5	WA1
6	GND or +V

Schematically, these may be wired as follows:



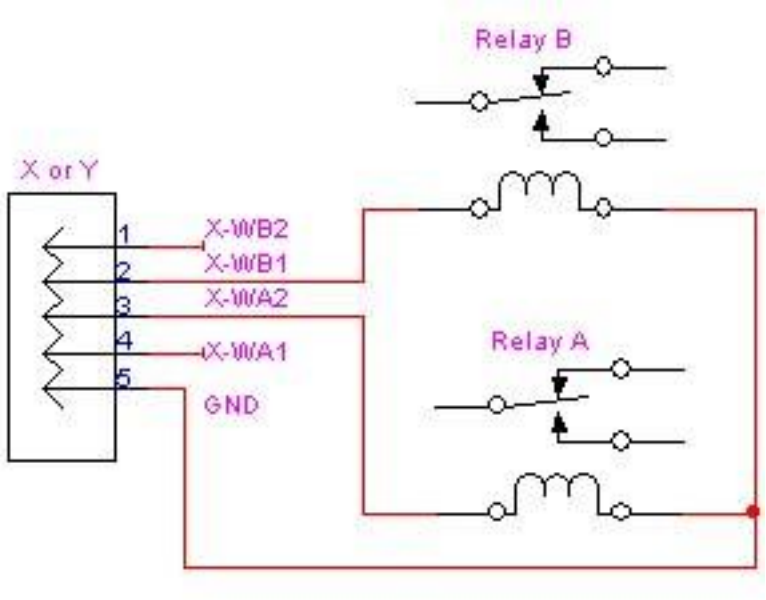
Wiring a Typical Set of 2 Relays
To a BiStep or SimStep unit
with the GenStepper Firmware

NOTE: "COMMON" is GND on BiStep boards,
and is +Vm on Unipolar boards
(SimStepA04 or SS0705)

The BiStep A04 board has one less pin one each X and Y connector. The "top" ground pin is deleted (the connectors are 5 pins, not 6), generating the pinout of:

Pin	Name
1	WB2
2	WB1
3	WA2
4	WA1
5	GND

Schematically, these connectors may be wired to relays as follows:



Wiring a Typical Set of 2 Relays
To a BiStepA04 with the GenStepper Firmware

Generic TTL Input

As noted in step 9 of the initialization phase, we are disabling response to the TTL limit switches as part of operating relays for the associated motor driver (though use of the 'T' command). This means that the limit switch inputs become available for several types of behaviors, which may be quite useful.

In the GenStepper firmware, you can directly read the current state of the limit switches via reading their data register. This is done through the command "X5?", which returns a bit-encoded value containing the current state of ALL of the limit switches. Note, however, that there is no "edge detection" or trigger capability: it is quite possible to completely miss a pair of state changes (0->1->0) via a polling loop, since your poll cycle is going to be separated by large amounts of non-polling time.

The format of the data returned by the "X5?" command is as follows:

X,5,##

The "##" is the bit-encoded contents of the register which contains the TTL input signals. They are reported as follows:

Bit	Value	Description
0	+1	Y- limit
1	+2	Y+ limit
2	+4	X- limit
3	+8	X+ limit

Any input line which is not connected will be reported as being high, since the board ties the inputs high through a very weak pull-up resistor. Therefore, if there are no wires connected to the limit switch inputs, GenStepper will generate the response:

X,5,15

to the "X5?" command sequence.

The next section ("Labeling of Board Signals") will help you to identify the board locations of the limit switch input signals.

Labeling Of Board Signals

There are currently a total of 6 significant versions of the SimStep and BiStep series of boards available. These versions can be roughly grouped into two major releases: ***Release 1***, which has a large (19 pin) SIP header in the middle of the board which contains all of the standard TTL I/O signals, and ***Release 2***, which uses clearly labeled connectors at the edge of the board, containing the same signals.

Release 1 Pinout for J1– SimStepA04, BiStepA04, and BiStepA05

The pinout for the J1 connector on the Release 1 set of boards (the SimStepA04, BiStepA04, and BiStepA05) is as follows, counting from the "top" part of the connector (nearest the DB9 serial connector) and proceeding downward. This connector is the 19 pin SIP header mounted in the middle of the board.

Pin	Board Label	Signal as used in this manual
1	RTC	
2	+5	
3	RST	
4	GND	GND
5	GND	GND
6	A0	LY-
7	A1	LY+
8	A2	LX-
9	A3	LX+
10	B0	Y-
11	B1	Y+
12	B2	X-
13	B3	X+
14	B4	NXT
15	B5/READY	RDY
16	B6/SERIN	SI
17	B7/SEROUT	SO
18	+5	
19	GND	GND

Release 2 – BiStepA06, BiStep2A, and SS0705

The Release 2 boards' serial connectors are fully labeled directly on the artwork. The signal names can be found by looking on the board near each connector. There are 3 input connectors, containing equivalent signals to those from the J1 connector on the earlier release.

- LIM contains RST, LY-, LY+, LX-, and LX+
- SLEW contains Y-, Y+, X-, and X+
- IO contains NXT, RDY, SI, and SO

Example VBS Script Program

The following is the complete text of an example VBScript program which can be used to demonstrate control of 2 relays, along with monitoring of all of the limit switch input lines.

```
'************************************************************
' GenStepperWithRelay.vbs
` NOTE: For use with the GenStepper Firmware version 1.70 or later
' Sample Script to demonstrate use of relays on
' GenStepper-firmware based boards, as well as to show
' generic TTL input from the same board.
'
' In order to actually implement the relay technique,
' we need to set up the motor parameters
' for the relay-driving "motor" system to match
' the requirements for operating a relay.
'   1.      Select the correct "motor" which is really
'       driving the relays ("X" or "Y", as needed)
'   2.      Set ½ step mode for winding updates ("1o"),
'       so no attempts at microstepping are done
'   3.      Set the step rate to the maximum value ("62500r")
'   4.      Set the "stop ok" rate to the maximum value ("62500k")
'   5.      Force the windings to always be on at full power ("1w")
'   6.      Calculate the size of a ½ step, in current microstep units.
'       By default, at power on to the system, this value is 8
'       (8 microsteps per half-step, or 16 microsteps per full step)
'   7.      Tell the motor to go back ½ step by issuing a minus seek of ½ step size.
'       For example, if the microstep size is 1/16th of a full step,
'       then this would be "-8s".
'   8.      Set your current "logical" location to 0 (both windings are off).
'   9.      At the extreme least, disable limit switch detection for the motor driver
'       which is being used as the relay control.
'       To disable detection on the X driver, issue the command "12t".
'       To disable detection on the Y driver, issue the command "3t".
'       To disable ALL limit switch detection (both X and Y drivers),
'       issue the command "15t".
'
' From now on, whenever you need to change relay A or relay B's state,
' you need to:
'   1.      Save the current state (0 to 3), as your "from state" value.
'   2.      Set or clear the appropriate bit(s) to decide your new state
'       (i.e., clear bit 0 in your state to turn off relay B)
'   3.      Use the above table to look up the scale factor for the
'       "distance" to travel to get to the new state
'   4.      Multiple the result by the ½ step size.
'       For example, if the table reports that you need to go -4 ½ steps,
'       and it takes 8 microsteps per ½ step,
'       then you will need to go (-4 * 8) or -32 steps to go to the new state.
'   5.      Issue the "seek relative" command on the appropriate motor
'       to generate the correct relay values.
'       In the above example (-4 ½ steps) implemented on motor Y as the
'       relay driving motor, you would issue the command "y-32s".
'   6.      Your new state value is (the 0-3) is the value to be used for
'       your next relay action start information.
'
' For this sample implementation, we shall make the following assumptions:
'
'   1.  We are assuming that the 'X' motor is connected as a stepper motor,
'       and the 'Y' motor is connected as relay control.  We are therefore not
'       issuing any X motor commands.
'   2.  The board is connected to port COM1 on the host computer
'   3.  All we are trying to do is cycle all possible relay combinations
'   4.  Power-on board defaults (such as 1/16th of a full step) are assumed
'
' NOTE: In a real implementation, the aDistances table would normally
'       be entered scaled to the microstep size (i.e., all of the values
'       multiplied by an appropriate constant), and the subsequent scaling
```

```
'       of distances to slew would be removed.  For this implementation, we
'       are leaving the table as is, for clarity of code.
'************************************************************

Option Explicit


    Dim strResponse
    Dim strSep
    Dim hBiStep                          ' Our local hBiStep
    Dim lLatchedData                    ' Gets the latched data
    Dim idWhich                         ' Which cycle; also gets the desired relay
pattern
    Dim aDistances(3,3)                 ' Gets our 4x4 distance table
    Dim idCurrentRelayState             ' Gets the current relay state (0-3)
    Dim cDistanceScale                  ' Gets the distance scale factor: Set to 8,
since we are operating in 1/16th full step mode

    cDistanceScale = 8                  ' 8 microsteps per half-step
    idCurrentRelayState = 0             ' Both A and B relays are OFF

    InitializeDistanceArray             ' Initialize the distance array

    Set hBiStep = CreateObject("BiStepComCtl.BiStepCom")        ' Create our object

    strSep = vbCrLf & String(20, "*") & vbCrLf

    hBiStep.FlushInput                  ' Attempt to flush the input
    hBiStep.ActionLogClear
    hBiStep.ErrorLogClear
    WScript.Echo "Initializing communications"
    hBiStep.StringWriteCommandSequence ("/pcom1")
           ' Select the com port: EDIT HERE TO CHANGE TO YOUR PORT NUMBER!
    hBiStep.StringWriteCommandSequence ("/i60000")
           ' Set the idle time to 1 minute (60 seconds)

    WScript.Echo "Initializing the board"
    hBiStep.StringWriteCommandSequence ("4!i")
           ' 1. Reset controller, wait for reset to complete

    WScript.Echo "Setting up correct relay actions"

    hBiStep.StringWriteCommandSequence ("0V")
           ' 2. Turn off verbose responses

    hBiStep.StringWriteCommandSequence ("Y")
           ' 3. All of our configurations are directed to the Y motor
    hBiStep.StringWriteCommandSequence ("1o")
           ' 4. Update only in 1/2 step mode
    hBiStep.StringWriteCommandSequence ("62500kr")
           ' 5. Set the initial and target slew rates to their maximum
    hBiStep.StringWriteCommandSequence ("1w")
           ' 6. Windings remain enabled at full power
    hBiStep.StringWriteCommandSequence ("-8s")
           ' 7. Back up 1/2 step, so we are correctly positioned
           `       for the distance table

    ' Note: at this point, both relays are off

    WScript.Echo "Now we shall cycle the relays 10 times"

    For idWhich=0 to 40
           ' Cycle 10 full sequences, 4 relay settings per sequence
        SetNewRelayState(idWhich AND 3)
           ' Set the new state to just the low 2 bits of our cycle counter
        ReportTTLInputValues
           ' Report TTL input values
        WScript.sleep 1000
           ' Sleep for 1 second
    Next

'    strResponse = "Current Action log is" & strSep & hBiStep.ActionLog & strSep

'    WScript.StdOut.Write strResponse
```

```
'    WScript.Echo hBiStep.Copyright

    hBiStep.ActionLogClear
    hBiStep.ErrorLogClear

    Set hBiStep = Nothing


'*****************************************************************************
' Sub InitializeDistanceArray
' Initialize the array
'          0      1      2      3
'    0     0      +2     -2     -4
'    1     -2     0      -4     +2
'    2     +2     -4     0      -2
'    3     -4     -2     +2     0
'
'*****************************************************************************
Sub InitializeDistanceArray
    aDistances(0,0) = 0
    aDistances(0,1) = 2
    aDistances(0,2) = -2
    aDistances(0,3) = -4

    aDistances(1,0) = -2
    aDistances(1,1) = 0
    aDistances(1,2) = -4
    aDistances(1,3) = +2

    aDistances(2,0) = +2
    aDistances(2,1) = -4
    aDistances(2,2) = 0
    aDistances(2,3) = -2

    aDistances(3,0) = -4
    aDistances(3,1) = -2
    aDistances(3,2) = +2
    aDistances(3,3) = 0
End Sub

'*****************************************************************************
' Sub SetNewRelayState(idNewState)
' Select a new relay state from the current state
'*****************************************************************************
Sub SetNewRelayState(ByVal idNewState)
    Dim lDistance               ' Gets the new distance
    idNewState = idNewState AND 3   ' Force valid new state
    lDistance = aDistances(idCurrentRelayState, idNewState AND 3)
           ' Get the new 1/2 step distance
    lDistance = cDistanceScale * lDistance
           ' Scale it correctly
    WScript.Echo "Old state=" & idCurrentRelayState & ", new state=" & idNewState & ",
distance=" & lDistance
    hBiStep.StringWriteCommandSequence ("y" & lDistance & "s")
           ' Select the Y relay motor, and tell it to slew
    idCurrentRelayState = idNewState
           ' Claim we are at the new location
End Sub

'*****************************************************************************
' Sub ReportTTLInputValues
' Input and report the TTL input value
'*****************************************************************************
Sub ReportTTLInputValues
    hBiStep.StringWriteCommandSequence ("y5?")
           ' Select the Y relay motor, and do the report of the TTL input lines
    WScript.Echo "TTL Input Lines=" & hBiStep.ResultGet("Y","5")
End Sub
```