Tutorial

Stepper Motor Control With a PC-Based Data Acquisition Toolkit

by *N. Shinjo, L. Buist and C. S. Subramanian*
Florida Institute of Technology, Melbourne, Florida

The use of LabVIEW is on the increase in universities and industries especially for data acquisition and process control. Frequently, there is a need to develop LabVIEW application software to work with foreign hardware, like a Slo-Syn stepper motor. Unfortunately, simple and clear procedures to develop motion control routines are not easily available. This paper describes a method developed to control a stepper motor using the LabVIEW 5.0 software and the National Instrument SCXI data acquisition system.

LabVIEW est de plus en plus utilisé dans les mondes universitaire et industriel, particulièrement pour la saisie de données et le contrôle de procédés. Régulièrement, des applications LabVIEW doivent être développées pour travailler avec du matériel étranger tel un moteur pas-à-pas Slo-Syn. Malheureusement, il n'existe pas de procédures simples et claires pour développer ces routines de gestion de mouvement. Cet article décrit une méthode pour contrôler un moteur pas-à-pas en utilisant le logiciel LabVIEW version 5.0 et le système de saisie de données SCXI de National Instrument.

1.0 Introduction

Frequently, data acquisition processes involve different types of device controls. However, most commonly used data acquisition software does not include general-purpose device control drivers or optional add-ons. So, the driver interface has to be developed by the user depending on the application. This paper describes two methods developed at the Florida Institute of Technology to control a stepper motor utilizing the commonly used PC data acquisition toolkit, LabVIEW 5.0, and parallel port or general-purpose digital I/O port of a data acquisition board. The first method uses digital I/O port of SCXI-1200 data acquisition unit (from National Instrument). The second method uses the same software but compiling the external code (C++) and controls the motor though a parallel port. The source code used for the second method can be used as a stand-alone motor control, without using any data acquisition card or the data acquisition software. With use of a translation module presented in this paper, most of the general-purpose stepper motor can be controlled using only two digital signal lines and a digital ground.

Any programming language such as BASIC or C or data acquisition software with digital I/O can be used for the motor control.

2.0 Motion Control System

Figure 1 shows the general schematic of motion control system. It consists of an 8-bit digital communication port, a driver or translation module, and a stepper motor. A parallel port (or printer port) or digital I/O ports on the data acquisition board can be used to control the stepper motor via translation module. Two communication methods are provided since typical external data acquisition boards are interfaced to the computer via parallel port.

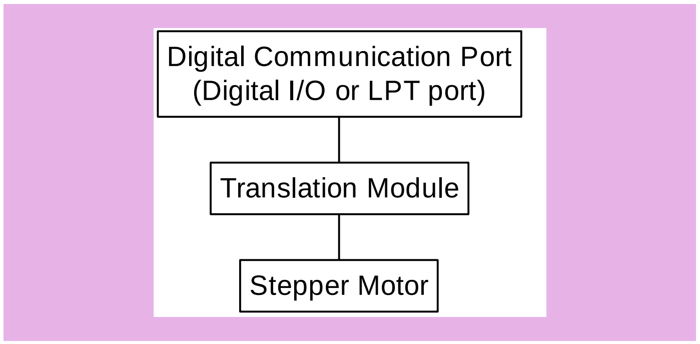


Figure 1: Schematic of Motion Control Setup

Two digital lines and a digital ground are connected to the translation module. One digital line controls the step and other controls the direction. The translation module converts the signal from the communication port to a specific winding energizing sequence to step the motor. The output signal from the translation module is sent to each winding (typically 2 sets of winding on the stepper motor). Details of the wiring, signals, and program code are described in the following sections.

2.1 Parallel Port Communication

An 8-bit data signal is sent to the translator module to move the stepper motor. To send 8-bit data to a specified port, port address and 8-bit hexadecimal code are required. Table 1 shows the data status addresses for parallel port (or LPT ports) of a PC.

Pin #2 (data 1), pin #9 (data 8), and pin #20 (ground) of LPT1 (378 hex) port was used to send the signals to the translation module. The pin #2 controls the step and the pin #9 sets the rotational direction. It should be noted that the any combination of 2 pins between # 2 to #9 can be used to control the motor. Table 2 shows an example of parallel port pin-outs. In the present case the following sequence is employed.

Table 1: Data Status Addresses

Port Number	Data (hex)	Status (hex)	Control (hex)
LPT 1	378	379	37A
LPT 2	278	279	27A

- The pin states 1 and 0 (pin # 2 high state and pin # 9 low state) initialize the forward (or clockwise) step.
- The pin states 1 and 1 (both pin # 2 and # 9 are high) initialize the backward (or counterclockwise) step.
- The pin states 0 and 0 (both pin # 2 and # 9 are low) followed by the state 1-1 or 1-0 executes the step.

Thus, the following 2 statements send the signals to LPT1 and move the motor one step (typically 1.8°). Note that the command '_outp()' only works for Microsoft Visual C++. The other compiler may use different command.

```
_outp(0x378, 1);    binary representation of 1 = 00000001
_outp(0x378, 0);    binary representation of 0 = 00000000
```

The first statement sets the 1st bit to 'high' state, rest of the bits are at 'low' state (2nd to 8th bits). The second statement sets all the bits to zero.

Table 2: Pin Connection and Description for Parallel interface to IBM PC (Hall, 1991)

Pin #	Signal	Description	Signal Direction
1	STROBE	STROBE pulse to read data in	IN/OUT
2	DATA 1	These signals represent information of the 1st to 8th bits of parallel data respectively. Each signal is at 'high' level when data is logical '1' and 'low' when logical '0'.	IN/OUT
3	DATA 2		IN/OUT
4	DATA 3		IN/OUT
5	DATA 4		IN/OUT
6	DATA 5		IN/OUT
7	DATA 6		IN/OUT
8	DATA 7		IN/OUT
9	DATA 8		IN/OUT
10	ACKNLG	A 'low' indicates that data has been received & printer is ready to accept other data.	OUT
11	BUSY	A 'high'signal indicates that the printer cannot receive data.	OUT
12	PE	A 'high' signal indicates that the printer is out of paper.	OUT
13	SLCT	This signal indicates that the printer is in the selected state.	OUT
14	AUTO FEED XT	This signal being at 'low' level, the paper is automatically fed one line after printing.	IN
15	NC	Not used	-
16	OV	Logic GND level	-
17	CHASIS GND	Printer chassis GND	-
18	NC	Not used	-
19	GND	Ground	-
20	GND		-
21	GND		-
22	GND		-
23	GND		-
24	GND		-
25	GND		-

The sequence of these two statements activate the flip-flop gate to flip. This causes the motor to move one step.

Similarly, the sequence of following two statements move the motor in reverse direction (counterclockwise) one step. The first statement reverses the direction of the rotation. The second statement activates the flip-flop circuit and sends a signal to the stepper motor.

```
_outp(0x378, 255);    binary representation of 255 = 11111111
_outp(0x378, 0);    binary representation of 0 = 00000000
```

2.2 Translation Module

The translation module generates pulse signals as and when it receives the control data from the computer. The pulse in (from high to low state) activates the circuit and sends signals to two sets of windings on the stepper motor. The first set of signals changes the 1st winding polarity and the 2nd set of signals changes the 2nd winding polarity, causing the motor to rotate one step (1.8°). When the circuit is powered, it supplies the power to the windings even if it is not turning. The translation module circuit diagram is shown in Figure 2.

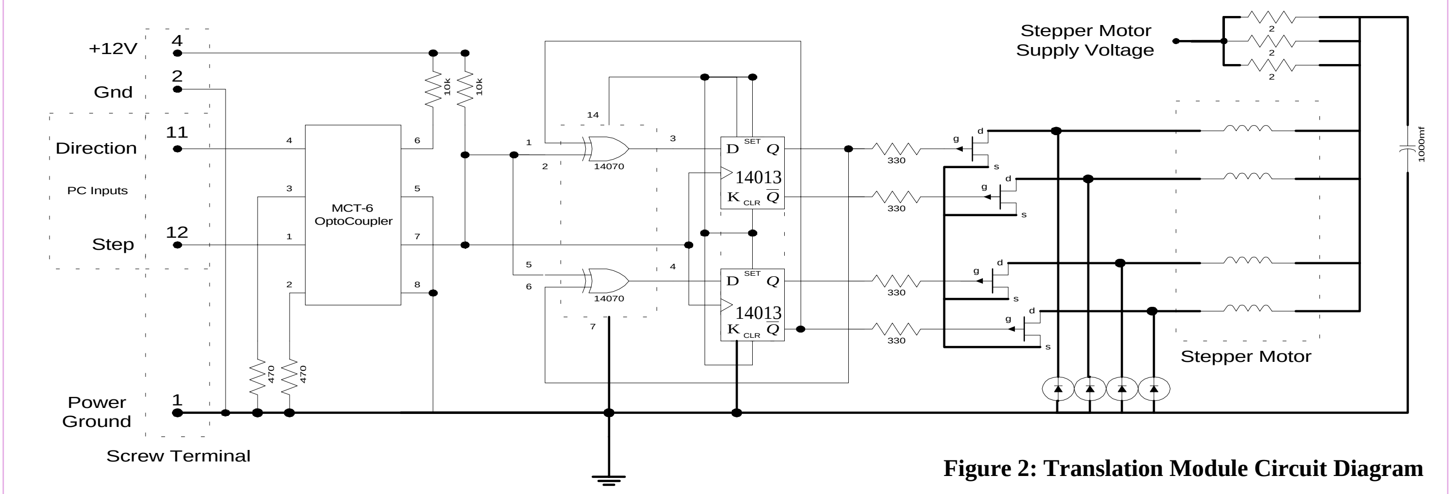


Figure 2: Translation Module Circuit Diagram

2.3 Stepper Motor

A stepper motor is a device that positions loads by operating in discrete increments (or steps). The stepping is accomplished by switching the power to the motor windings so that the motor phases are energized in a specific sequence. The Figure 3 shows the typical wiring diagram of the stepper motor (2 winding sets).

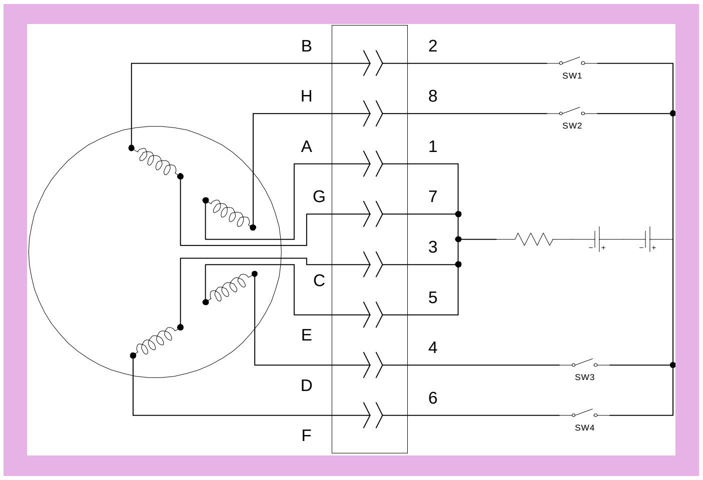


Figure 3: Schematic of Stepper Motor Windings

Table 3 shows the switching sequence required to turn the motor through one step (typical step size is 1.8°), and 200 steps are used for one complete revolution, i.e. 360° turn. To step in the reverse direction, the switching sequence is reversed (i.e. 4, 3, 2, 1).

.

Table 3: Winding Sequence

Steps	SW1	SW2	SW3	SW4
1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	1	0

1: high state, 0: low state

The stepper motor is capable of operating in a half step (0.9°) mode also. However, that would require additional switching sequence and a different translation module circuitry. Further, the half-step reduces the holding torque of the motor. For present application only the full-step increment (1.8°) is used.

3.0 Motion Control Using Computer Parallel Port

The schematic setup of the parallel port motor control is described in Section 2.1. With use of the translation module described in Section 2.2, the following lines of code executes one step in a specified direction. The following code is written for Microsoft Visual C++, a command for parallel port communication varies with the language, for example Borland C++ uses 'outp' (port address in hex, binary code) and the Visual Basic uses 'out' (port address in dec, binary code). By controlling the loop execution time, the speed of the stepping can be controlled. For more accurate stepping time control, a timer command (wait statement) can be used instead of a loop command.

```
do loop
_outp(0x378, 0 or 255);
(an empty loop to adjust the timing between the command)
_outp(0x378, 0)
(an empty loop to adjust the timing between the command)
(an empty loop or timer to adjust loop execution time)
end loop
```

The first '_outp' command sets the direction of the motor rotation and the second executes the step. It should be noted that for a faster CPU, it may require empty loops (or timing command) to create a delay between each command execution. Rapid command execution can cause erratic stepper motor motion. For the motor used in the present application, the ideal timing between commands was 100 ms.

3.1 Motion Control Using LabVIEW

The basic setup for the parallel port motion control using LabVIEW (Figure 4) is the same as the conventional setup described in Section 2.1. The exact same code described in the previous section can be used for the control. The only difference is that instead of compiling the code as a stand-alone executable program it is compiled as a LabVIEW executable program. In this way, the motion control routine can be used along with data acquisition routines developed with the LabVIEW. For this, the LabVIEW function called the Code Interface Node (CIN) is used to link the external code written in a conventional programming language to LabVIEW [3,4]. LabVIEW calls the executable code when the node executes, passing input data from the block diagram to the executable code, and return the data from the executable code to the block diagram. The code can be compiled as either single-threaded or multi-thread operations. Only the following compilers can be used to create CIN.

- The Microsoft Visual C++ compiler
- Symantec C compiler
- The Watcom C/386 compiler for Windows 3.1.

The external code (C++) loaded into the LabVIEW's CIN is simplified to minimize lockup of the program while the CIN code is executing. The CIN code executes only one step at a time and returns to the motion control VI. The one step is equivalent to 1.8° rotation. To do multiple steps, the CIN was called multiple times. Inserting the loop delay in the motion control VI, the speed of the subsequent steps can be controlled. The drawback of this control configuration is that the stepping speed varies with the CPU speed. This can be improved by placing the loop structure (for the number of steps you want to execute) in the C code.

3.1.1 Description of 'Make File'

Compilers need instructions on how to build a project or a file. These instructions come in the form of make files. LabVIEW installs a make file in the LabVIEW\cintools directory that has instructions for the compilers to build a generic CIN (ntlvsb.mak); but other compilers will need some additional instructions on how to build a CIN. A specific make file, combined with the LabVIEW generic make file, will instruct the compiler to build a .lsb file (instead of a .exe file). The .lsb file is used

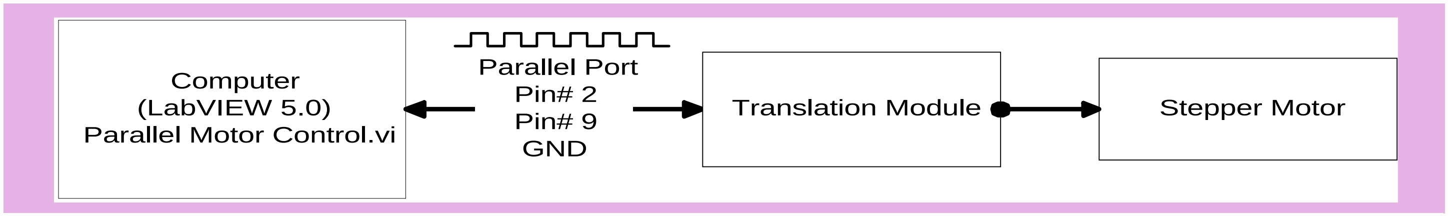


Figure 4: Schematic of Stepper Motor Control Setup Using Computer Parallel Port

to interface the external code such as C to LabVIEW program. The CIN only accepts the external code with .lsb extension (it does create .c extension file but it does not accept .c extension file as a CIN source code). Additional information can be found on “LabVIEW Code Interface Reference Manual” [2].

3.1.2 Steps for Creating a CIN

The procedures for creating a CIN differs with the platform and compiler used. In this section, the procedures for using Microsoft Windows 95 with Microsoft Visual C++ compiler are illustrated. For more detail and other compiler's procedure, refer to “LabVIEW Code Interface Reference Manual”.

1. Prior to the CIN generation, determine the data and the data format (i.e. integer, floating point, strings, etc.) to be passed to and received from the CIN.

2. Write a C code and verify that there are no errors in the code.

3. From the LabVIEW VI diagram, select a code interface node icon (Functions => Advanced) and place on the diagram. It should look like in Figure 5.

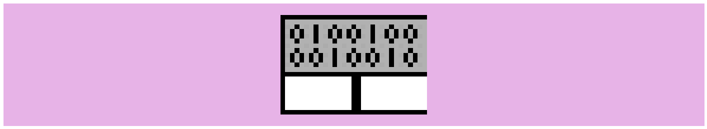


Figure 5: Code Interface Node Icon

4. Initially, the CIN has one set of terminals (a set of input and output terminals). The additional terminals can be added by dragging the bottom left or right corner of the icon or by selecting “Add Parameters” from the CIN terminal pop-up menu (right mouse click on the parameter terminals) (Figure 6).

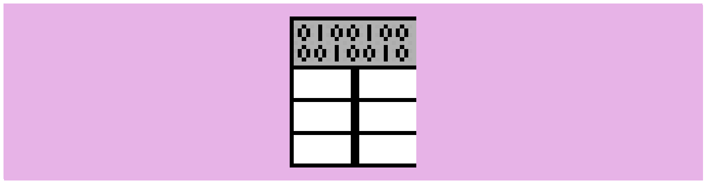


Figure 6: CIN Icon with Multiple Input and Output

5. Connect the terminals to control and indicator panels (Figure 7).

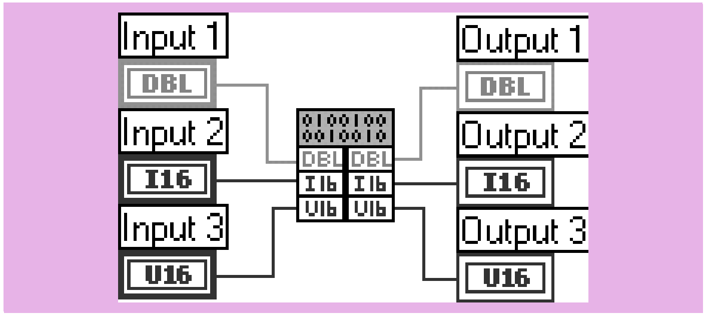


Figure 7: CIN Icons with Terminals

6. Select “Create .C File” from the pop-up menu (right click on the icon). It will ask for the file name (filename.c format). Type C source filename and save it. The saved file contains following C code.

```
/*
 * CIN source file
 */

#include "extcode.h"

CIN MgErr CINRun(float64 *Input_1, int16 *Input_2, uInt16
*Input_3);

CIN MgErr CINRun(float64 *Input_1, int16 *Input_2, uInt16
*Input_3)
{
        /* ENTER YOUR CODE HERE */
        return noErr;
}
```

It should be noted that the labels (i.e. Input 1, Output 1, etc.) and its data format (i.e. double, integer 16, etc.) on the controls and indicators become the variable names and data format of the C code. In addition, #include “extcode.h” should be the first header file. Additional header files such as “math.h” and “stdio.h” should follow “extcode.h” header.

7. Insert C code that you wrote into the LabVIEW generated C code (where it says /* ENTER YOUR CODE HERE */). Make sure the variable names and types match with the declared variables. For example, for the present case:

C code for the stepper motor control

```
/*
 * CIN source file
 */

#include "extcode.h"
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

CIN MgErr CINRun(int16 *var1);

CIN MgErr CINRun(int16 *var1)
{
        {
        _outp(0x378, *var1);
        _outp(0x378, 0);
        }
        return noErr;
}
```

8. Create a 'make' file using text editor (notepad, word, word pad, etc.) and save it with the same file name as C source code (with .mak extension). The make file should contain following 4 lines.

```
Name = name of C file without extension
Type = CIN
CINTOOLSDIR = path to cintools directory
!include<$(CINTOOLSDIR)\ntlvsb.mak>
```

For example, for the present case:

Make File for the stepper motor control

```
name = step
type = CIN
cintoolsdir = c:\labview\cintools
!include <$(cintoolsdir)\ntlvsb.mak>
```

The path to the cintools directory can be found (for the Windows 95 system) in "C:\ProgramFiles\National Instruments\LabVIEW\Cin-tools".

9. Prior to compiling the source code, make sure that the C source code and the make file are in the same directory (or the compiler will not create .lsb extension file, it will create .exe file instead).

10. Open the make file in Microsoft Visual C++. When the file is opened, the program displays two warning dialog boxes. First, it says, "This makefile was not generated by Developer Studio". Answer "Yes" to "Do you want to continue?". Then in the second dialog box shows the type of platform you wish to operate on (default is Win32). Make sure the Win32 is checked and click "OK".

11. Select "Build filename.exe file" from the build pull-down menu in Microsoft Visual C++. The program follows the instruction in the make file and creates filename.lsb instead of filename.exe.

    If .lsb extension file is created correctly, the status window displays the following status report.

```
------------ Configuration: Filename - Win 32 Debug ------------
Microsoft (R) Program Maintenance Utility version 1.62.7022
Copyright (C) Microsoft Corp 1988-1997. All rights reserved.
filename.c
Microsoft (R) 32-Bit Incremental Linker Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.
LabVIEW resource file, type 'CIN', name 'filename.lsb', created properly.

Filename.exe - 0 error(s), 0 warning(s)
```

    Where filename is the name of the C source code or the make file that you compiling. The .lsb extension file should be in the directory where the C source code and make files are in.

12. From the LabVIEW vi diagram, (Step 3), right-click on the CIN icon and choose 'Load Code Resources'. The dialog box indicates the path to the .lsb extension files that you want to load. Select the file and click 'Open'.

13. The source code is now loaded onto the CIN icon and ready to perform the task. Once the code is loaded, it will become a stand-alone, executable icon (does not require c source code or make file). When this vi is saved (including CIN icon), it can be called from other VI's.

A LabVIEW sub VI, called Motor Control.vi, is developed to control the stepper motor through a parallel port. The Front Panel diagram and the Block Diagram for which are shown below in Figure 8.

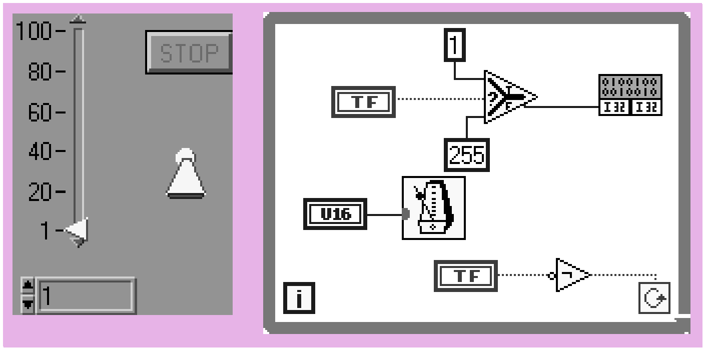


Figure 8: Front Panel and Block Diagram of the LABVIEW sub-VI For the parallel Port Stepper Motor Control

4.0 Motion Control Using SCXI System

National Instrument Signal Conditioning Extension for Instrumentation (SCXI) system with LabVIEW 5.0 was used to develop the program for running a wind tunnel experiment involving a stepper motor. The system consists of SCXI-1000 chassis which has 4 slots for data acquisition, multiplexing, and control modules. Two modules are installed in the present system: SCXI-1200 and SCXI-1121. The SCXI-1200 is the general-purpose data acquisition board (A/D card). It has 4 differential channels (or 8 single-end channels) and three 8-bit digital I/O ports. The SCXI-1121 (with SCXI-1321 accessory) has 4 optically isolated channels (differential) and excitations. The basic setup of the SCXI system is the same as the parallel motion control setup. Since the SCXI system uses parallel port to communicate with a computer, SCXI-1200's digital I/O port was used to control the stepper motor.

4.1 Stepper Motor Connection (SCXI-1200)

SCXI-1200's digital I/O port (PA-0, PA-7, and digital ground) controls the stepper motor (via translator circuit) as shown in Figure 9. The detailed descriptions of the ports are provided in the SCXI-1200 manual [3]. The digital pulse combination (described above) and frequency of PA-0 and PA-7 control the direction and the speed of the motor.

The pin connections between the SCXI-1200 unit (50-pin) and the translation module terminal are described in the Table 4. For the connection from the translation modules to the stepper motor, refer to the translator circuit diagram (Figure 2).

Table 4: Connector Pin-outs (SCXI-1200)

SCXI 1200 Pin Number (50 pin)	Translation Circuit Terminal Number
13 (Digital Ground)	3 (Ground)
14 (PA-0)	11 (Step Signal)
21 (PA-7)	12 (Direction Signal)

A LabVIEW sub VI, called DIO Motor Control (one step).vi, is developed to control the stepper motor. The Front Panel diagram and the Block Diagram of which are shown below in Figure 10.

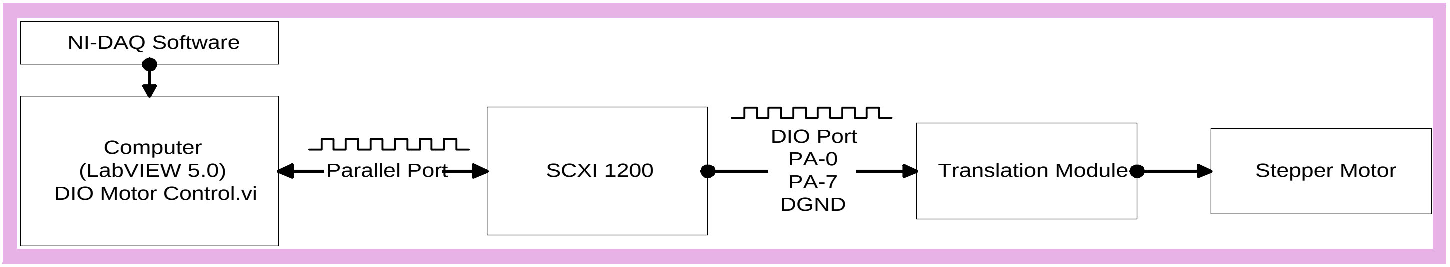


Figure 9: Schematic of Stepper Motor Control Setup Using SCXI-1200

5.0 Summary

The methods to control a stepper motor using a parallel port and National Instrument's SCXI unit are described. The use of the translation module, described in this paper, enables control of a stepper motor using conventional programming languages or data acquisition software. It is an inexpensive way of developing a prototype data acquisition/control system. In addition, data acquisition software, which has a capability of compiling conventional programming codes (such as LabVIEW Code Interface Node (CIN), allows the motion control routine to be embedded as a part of the acquisition function. The procedure is explained in detail here for the driver routine written in C++. The procedure can be applied to any motion control such as, probe traversing, model rotation etc.

6.0 Acknowledgments

The authors are very grateful to Mr. Dan Simpson for his assistance in the lab.

7.0 References

[1]. Hall, Douglas V., “Microprocessors and Interfacing: Programming and Hardware”, McGraw-Hill, 2nd Ed., 1991.

[2]. National Instruments, “LabView Code Interface Reference Manual”, National Instruments, Jan. 1998 Ed., Part No. 320539D-01.

[3]. National Instruments, “LabView User Manual”, National Instruments, January 1998 Edition, Part Number 320999D-01.

[4]. National Instruments, “G Programming Reference Manual”, National Instruments, Jan. 1998 Ed., Part Number 321296B-01.

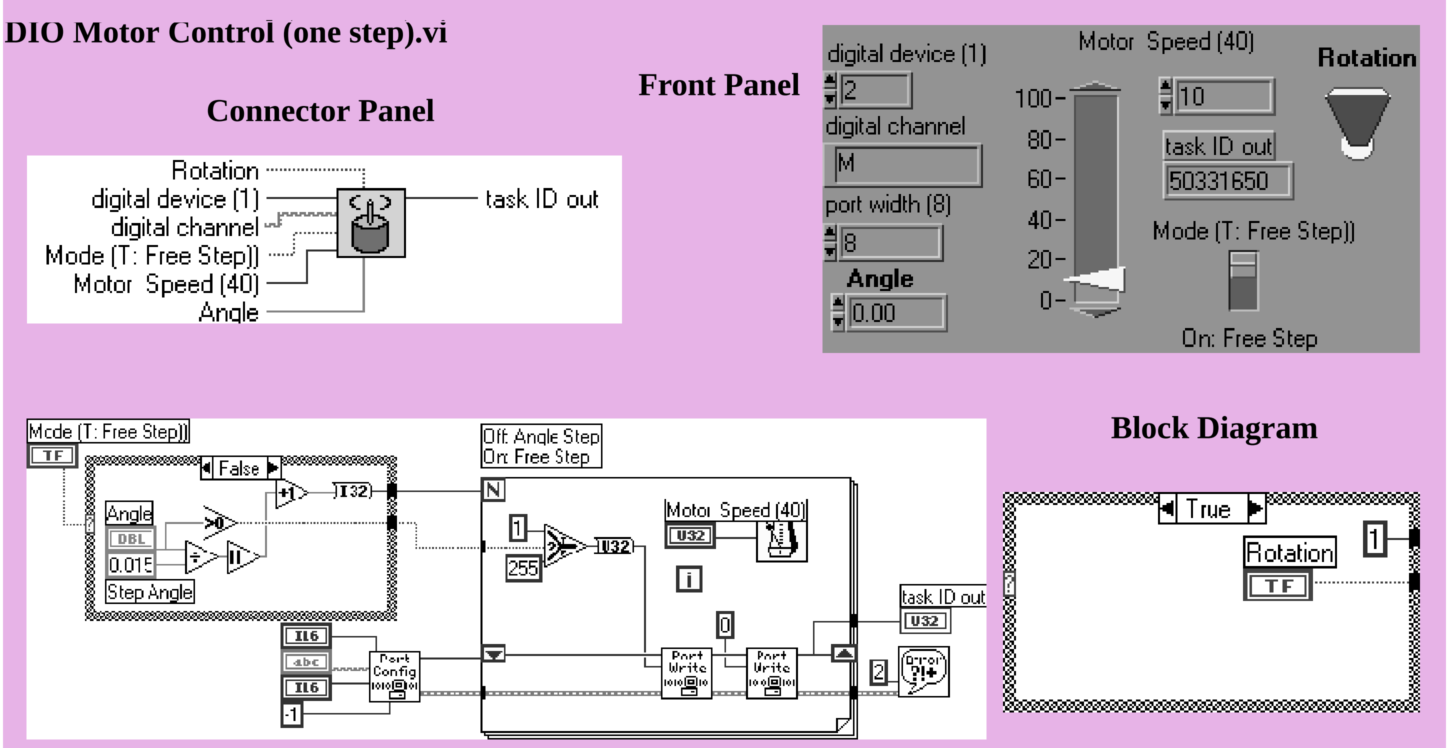
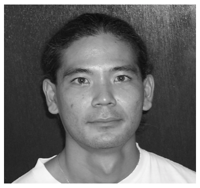


Figure 10: Front Panel and Block Diagram of the LABVIEW sub-VI For the SCXI Stepper Motor Control

About the Authors



Nagahiko Shinjo is a graduate research student pursuing Ph.D. degree in Ocean Engineering at Florida Institute of Technology. He obtained his M.S. also in Ocean Engineering from the same University. His general areas of research include bio-fouling control, autonomous underwater vehicle control system, and underwater instrumentation. Currently he is developing a memory alloy based ballast system for autonomous underwater vehicles and biologically based underwater vehicles.

E-mail: shinjon@winnie.fit.edu

Larry Buist, a 13 year veteran of Florida Tech, has extensive prototyping experience with Harris GISD (Intersil), MicroPac Industries, and a variety of electronics start-up companies in Nevada, Texas and Florida. He holds a patent in the video game industry and is involved in the design and construction of electronic and electro-mechanical circuits on a variety of research projects at Florida Tech.

E-mail: lbuist@fit.edu

Chelakara S. Subramanian is an associate professor of Aerospace Engineering at Florida Institute of Technology. He obtained his Ph.D. in Mechanical Engineering from University of Newcastle, Australia and M.E. in Aerospace Engineering from the Indian Institute of Science, India. His research expertise is in experimental fluid mechanics and new instrumentation. Currently, he is involved in research projects on pressure sensitive/ temperature sensitive paint system, particle imaging velocimetry, low-cost diode powered 2-component laser doppler velocimetry, hurricane flow field measurements and wind engineering studies. He is an associate fellow of AIAA, member of ASME, member of Society of Engineers, U.K., licensed Professional Engineer in U.K. and a member of International Society of Professional Engineers in France. E-mail: subraman@winnie.fit.edu