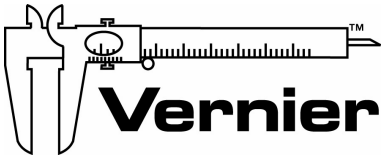
Digital Control Unit (DCU) Manual



Vernier Software & Technology
13979 SW Millikan Way
Beaverton, Oregon 97005-2886
(503) 277-2299
888-837-6437
FAX (503) 277-2440
www.vernier.com
info@vernier.com

Monday, April 25, 2005

Order Code DCU-BTD

Contents

Published by
Vernier Software & Technology
13979 SW Millikan Way
Beaverton, Oregon 97005-2886
(503) 277-2299
(888) 837-6437
FAX (503) 277-2440
www.vernier.com
info@vernier.com

Fifth Edition 2005
Second Printing
Printed in the United States of America

About This Manual

This manual describes the Digital Control Unit (DCU) and how it can be used with the Vernier LabPro™ and the CBL 2™ from Texas Instruments. It also goes into details of how to control LabPro's analog output line. Details on how the DCU and analog output work, how to control the outputs, and how to connect devices to them are all covered in the manual.

Use this manual to learn the programming techniques that allow you to control the DCU and the analog output line with TI graphing calculators, LabVIEW, and REALbasic. In addition, the CD that comes with this manual contains many sample programs for performing digital and analog output with any of these programming languages, using an array of electronic devices. Some of these programs show how to combine output with reading data from a sensor.

This manual focuses on programming the LabPro to control the DCU and analog output line. It does not cover all of the commands that control the LabPro. For more information on this, refer to the *LabPro Technical Reference Manual*. The LabPro manual is available free on our web site, www.vernier.com/diy (for "Do-It-Yourself"). Also, you will need to refer to the manuals that came with your calculator, or your LabVIEW or REALbasic documentation for programming information. If you are using a CBL 2, you may want to refer to *Getting Started with the CBL 2* which is included with every CBL 2. If you are using a calculator for DCU programs, this manual also assumes that you are familiar with the use of TI-GRAPH LINK™ or TI Connect for transferring programs from a computer to the calculator. If you are not, we encourage you to study other manuals and web site information at education.ti.com or www.vernier.com for information on this process.

Organization of This Manual

We have tried to organize the manual to allow you to jump around and find information without having to read the entire manual. However, we do encourage you to read the entire manual (except the programming sections that do not relate) before jumping into DCU projects. The progression of the manual, the information, and the sections is as follows:

Learn about how to use the DCU and analog output as a teaching tool, view some project ideas, and get an overview of the DCU and the analog output driver in the first three sections: *Introduction*, *DCU Overview*, and *Analog Output Overview*.

The next section, *The DCU Sample Programs CD*, provides a description of all of the sample programs included on the CD that come with this manual, as well as information about the platforms and programming languages that can be used. Use the sample programs for testing your electrical devices, for learning how to control the DCU and analog output driver, and as the building blocks of your own custom programs.

Programming is a big part of creating projects with the DCU. Regardless of which programming language you use, the next four sections are important reference sections. While these sections do not cover the specifics of the different programming languages (that can be found in later sections), they do cover the specifics for sending commands to control the LabPro or CBL 2. The first of these programming sections, *Programming Overview*, discusses what a command is, how to send it, and how to initialize the LabPro/CBL 2 at the beginning of every program. The next programming section, *Programming for Data Collection*, goes into the different techniques used for collecting data from analog sensors. Examples of how to perform these techniques, and what commands to use, are given. The third programming section, *Programming for the DCU* describe the two techniques for performing digital output: direct control and sequence control. This section explains the two techniques and provides an example for each. The last programming section, *Programming for Analog Output Control*, goes into the details of how to control the LabPro analog output driver. At the end of each section, except the *Programming Overview*, you will find programming challenges.

Programming specifics are the topic in the next six sections. However, unlike the previous programming sections, these sections are specific to the three programming environments. Each programming language has two sections that are devoted just to them. Study the sections that pertain to your programming environment (calculators, REALbasic, or LabVIEW).

In the next two sections, *Connecting Devices to the DCU* and *Connecting Devices to the LabPro Analog Out Line*, the manual switches gears to focus on the hardware. Here you will find information for connecting all sorts of

devices, such as speakers, polarized devices, non-polarized devices, LEDs, DC motors, stepper motors, and servo motors.

The final section, *DCU Project Ideas*, provides tips and suggestions for some exciting projects. Choose one of these projects, or use the list to help brainstorm your own project idea. These projects combine use of the DCU and analog output driver with programming, electronics, design, and engineering.

Acknowledgements

The Digital Control Unit was designed by John Wheeler . Dr. Fred J. Thomas of Sinclair Community College, Dayton, Ohio, and mathmachines.com, pioneered the use of the digital outputs of the CBL 2 and provided valuable suggestions for this project. Matthew Denton, Ian Honohan , Jeffery Hellman, Laura Owen, and Adam Higley created and tested most of the sample DCU calculator programs and projects we built. Jeffery Hellman, David Stroud, Sam Swartley, Cole Wardell and David Vernier wrote most of the sample DCU computer programs. Thanks to Michele Perin, Sam Swartley, John Gastineau, Kelly Redding, Christine Vernier, Erik Schmidt, Jan White, Dan Holmquist, Gretchen Stahmer DeMoss, and Ian Honohan of Vernier Software & Technology for their editing and testing. Thanks to Adrian Oldknow for encouraging me on this project. Thanks also to Scott Webb and Eren Koont of Texas Instruments for their ideas and encouragement on this project.

David Vernier

Vernier Software & Technology

Introduction

The Digital Control Unit (DCU) is designed to make it easy for you to use the LabPro or Calculator Based Laboratory 2 (CBL 2) digital output lines. Using the DCU and simple programs, you can turn on and off DC motors, lamps, LEDs, buzzers, stepper motors, and other DC electrical devices. You can even develop more elaborate projects, such as robots that move around the room, or automated scientific apparatus. The most exciting projects involve combining the use of sensors connected to the LabPro/CBL 2 with output from the DCU. Examples include alarm systems, temperature-controlled environments, and smart robots.

In this new version of the DCU Manual we are also including information on how to control the analog output lines of LabPro.

The DCU as a Teaching Tool

There are several ways you can use your DCU.

- DCU projects are a great way to teach electronics, sensors, feedback and control, and other engineering concepts. The hardware sections of this manual explain how to connect various electrical devices.
- DCU projects are a great way to teach programming. Many programming project ideas are given in this book. This manual and the sample programs can provide a good place to start. We hope you will take our programs, examine them, add to them, and make your own programs.
- The DCU can provide a useful tool for student (or teacher) projects. Use one of the many projects listed here or create your own. The sample programs we provide may do what you need or most of what you need. If so, start with that program and modify it as necessary. Use the subroutines we provide as building blocks for your own programs.
- Teaching the basics of digital and analog output and control can be accomplished without introducing programming in several different ways:

  - Computer users can use the Digital Control Unit with Logger *Pro* 3.3 and newer versions. Logger *Pro* is a general-purpose data collection program for LabPro and Vernier Sensors, but it also supports setting up the DCU to turn on or off digital devices when a sensor reading reaches a specified limit. You can also set the digital lines to come on at specified times. Logger *Pro* also allows you to control the LabPro analog output line.

  - Computer users can also use the simple applications that we provide on the CD to control the DCU lines and the analog output lines. There are applications for Windows®, Macintosh® OS 9, Macintosh® OS X, and Linux®

  - Calculator users can use the calculator programs as they are on the CD with no programming required.

DCU Project Ideas

Over the years, we have used the Digital Control Unit for dozens of projects, some whimsical and some practical. Here are some of the projects we have built at Vernier. You probably have some better ideas of your own.

- Turn on and off DC motors and operate them in either of two directions
- Control stepper motors or servo motors
- Create temperature-controlled environments
- Add a buzzer to the LabPro/CBL 2 to warn you when an event occurs, such as a person walking near a motion detector
- Make live traps for small animals or bugs, activated by a sensor detecting the presence of the animal
- Create an automatic tea brewer
- Make automated scientific instruments and demonstration equipment
- Control flashing lamps and LEDs and moving objects for marketing displays

- Build kinetic sculptures
- Build an automated battery tester
- Build a roving robot with sensors, controlled by a calculator or a laptop computer

Vernier Robotics and Control Kit

Vernier Software & Technology also sells a Robotics and Control Kit (order code RCK-DCU). It contains connection hardware and assorted components, such as: motors (DC, servo, and stepper), a fan, a speaker, lamps and holders, LEDs, and a buzzer for use with the DCU and LabPro analog output lines. The parts in this kit were selected to be easy to use with the DCU and LabPro analog output. It includes a manual describing 14 projects for student construction, programming and experimentation. Each project includes feedback and control extensions and other challenges.

Using the DCU and the LabPro Analog Output Line with Logger *Pro*

Even though most of this manual discusses using special programs or home-made programs for controlling the Digital Control Unit, you can also use the DCU with the Vernier Logger *Pro* program. Versions of Logger *Pro* from 3.3 and newer can control the DCU. Connect the DCU and an analog sensor to your LabPro and start up the Logger *Pro* program. Choose Set Up Sensors from the Experiment menu. Click on the DCU icon (showing it connected to the LabPro). A pull-down menu should appear. One item on that menu is Digital Output. Choose it. A dialog box will appear which lets you select a sensor (or any other column of data) and set a limit. You can choose to have the DCU turn on any lines you specify when the value goes above (or below) the limit.

Similarly, you can control the analog output line of LabPro in Logger *Pro* 3. Connect the LabPro voltage probe to CH4. Start up Logger *Pro* and choose Set Up Sensors from the Experiment menu. Click on the voltage probe icon (showing it connected to the LabPro). A pull-down menu should appear. One item on that menu is Analog Output. Choose it. A dialog box will appear, which lets you set the waveform, amplitude, and frequency of the analog output. This output will continue even after you close the dialog box. You can take data as the analog output continues; in fact, it is easy to plot the voltage output from CH4 as it is changes.

DCU Overview

The Digital Control Unit (DCU) is a small box with a short cable and clear plastic top. On one side of this box is a socket to plug in a DC power supply. On the same side is the DCU's short cable, which plugs into the Dig/Sonic connector on a LabPro or CBL 2. On the opposite side of the box is a 9-pin D-sub socket for connecting electronic devices that you build. There are connections for all six digital lines, plus a power connection and two ground connections. We supply a cable, with bare wires on one end, for you to use in building your first projects. You can use the DCU to control as many as six electrical devices.

The top of the DCU is transparent. There are six red LEDs and one green LED visible inside the unit. The green LED comes on when the DCU is properly connected and running a DCU program. The red LEDs indicate the status of the six output lines of the DCU.

Power Source

Four different power sources can be used:

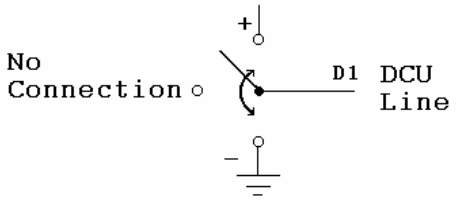
- LabPro power supply (6V DC, regulated, 600 mA) [Vernier order code IPS in North America, different versions are available for use in other parts of the world)]
- CBL 2 power supply (6V, regulated, 300 mA) [TI Model AC 9201]
- ULI power supply (9V, unregulated, 1000mA) [Vernier order code ULI-PS]
- Battery power supply (one lantern battery, or a collection of four to eight 1.5-volt cells in series). To make the cable from the batteries to the DCU, you need to use a 5.5 mm × 2.1 mm power connector (Radio Shack part number 274-1569). Connect the leads so that the center of the connector is negative. The voltage supplied can be anything between 5 volts and 12 volts. One easy way to build a battery power supply is to use a holder for four C or D batteries in series, which will provide about 6 volts.

Never apply more than 12 volts DC to the DCU. Never use AC power supplies with the DCU. Note that the center connector on the DCU is negative. The total current drawn by everything plugged into DCU should not exceed 600 mA.

Digital Output Lines

The transparent top of the DCU reveals six red LEDs and a green LED. The green LED should be on when a program is running and the DCU is properly connected and powered. Learn to check the green LED. It can warn you if things are not set up correctly, and it will keep you from wasting time when they aren't.

The red LEDs indicate the status of the six digital output lines. We refer to the six digital output lines as D1, D2, D3, D4, D5, and D6. You can think of the DCU as a set of six remote-controlled switches. Each of the six lines from the DCU is connected to a switch that can have any one of three positions:



The line can be connected to the positive side of the DCU power supply, to the negative side of the power supply, or left unconnected. There are six switches of this type inside the DCU. Actually they are not mechanical switches, but rather electronic switches using transistors that function like the mechanical switch illustrated above. If you connect an electrical device (such as a motor or lamp) between the DCU line and a ground connection, you can control whether it is on or off using this switch. If the switch is in the + position, current will flow, and the device will be on. Either of the other two positions will turn it off.

If you have read the specifications in the documentation which came with the LabPro or CBL 2, you may be

surprised to see that there are six digital output lines on the DCU. Each digital port on the LabPro or CBL 2 has only four digital output lines. However, we do some digital logic tricks which allow us to control six, instead of four lines. Of course, we had to pay a price for this trickery. We do not have totally independent control of all six lines. We compromised on a pattern that allows us independent control of the first three LEDs and then allows us to use the other three with restrictions. The easiest way to see the restrictions is to examine the 16 possible output patterns from the DCU:

Output	Binary	D1	D2	D3	D4	D5	D6
0	0000	—	—	—	—	X	X
1	0001	+	—	—	—	X	X
2	0010	—	+	—	—	X	X
3	0011	+	+	—	—	X	X
4	0100	—	—	+	—	X	X
5	0101	+	—	+	—	X	X
6	0110	—	+	+	—	X	X
7	0111	+	+	+	—	X	X
8	1000	—	—	—	+	X	X
9	1001	+	—	—	+	X	X
10	1010	—	+	—	+	X	X
11	1011	+	+	—	+	X	X
12	1100	X	X	X	X	—	—
13	1101	X	X	X	X	+	—
14	1110	X	X	X	X	—	+
15	1111	X	X	X	X	+	+

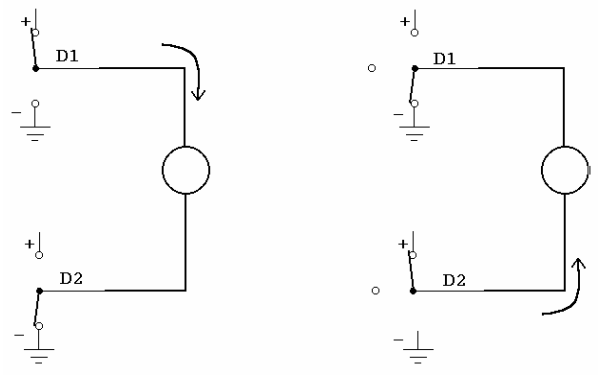
+ indicates the line is connected to the positive voltage of the DCU power supply, **—** indicates the line is connected to ground (negative lead of the DCU power supply), and X means the line is disconnected.

The outputs 0 through 11 can be considered as the binary equivalent of the number, with D1 used for the least significant bit, D2 used for the second digit, D3 for the third, and D4 as the most significant bit.

Outputs 0 through 7 give totally independent control of the first three digital lines.

Outputs 12 through 15 are designed for controlling just D5 and D6, but do not allow any use of the first four lines. One reason for this choice is to allow for building robot cars. With such a car, you might want one wheel to be controlled by D1 and D2, and another to be controlled by D3 and D4. It would still be useful to have some other lines that could be used for other operations. Lines D5 and D6 do this, but these lines can only be used when lines D1 through D4 are off. With this setup you could build a robot that can move around using lines D1 thru D4 and then when it reaches a destination it can do a separate action. For example, you could turn on a buzzer, shoot something, or run a third motor.

Pairs of DCU lines can be used together to allow you to switch the polarity, or direction of current flow in an electrical device. Consider the circuits on the next page. Both circuits show an electrical device wired between the D1 and D2 lines of the DCU. The circuit at the left has the D1 set for **+** and D2 set for **—**, so positive current would flow from D1 through the device and to D2. The circuit on the right has D1 set for — and D2 set for **+**, so positive current would flow from D2 to D1. This idea allows you to have motors run in either of two directions.



Two Digital Output Lines Used to Run a Motor in Two Directions

For connecting electrical devices, we provide a cable which connects to the 9-pin, sub-D socket on the side of the DCU. There are connections for all six digital lines, plus power and ground. The color code of the six wires are labeled on the cable. There are wires for each of the six digital output lines, two ground wires and a wire labeled "+" connected to the power from line. Details on the DCU connector, and power and current limitations, can be found in the "Connecting Devices to the DCU" section of this manual.

Analog Output Overview

This version of the DCU manual includes instructions on how to use the analog output line of the Vernier LabPro interface. You can connect to this line by using the Voltage Probe which comes with every LabPro (VP-BTA). Connect it to the CH4 connector. Note that the analog output line is not available on the CBL 2 interface. Unlike the digital output lines of the DCU, the analog output line can be set at any voltage from –4 to +4 volts, and it can even be used as a function generator. You can set a voltage output waveform to sine, triangle, ramp up, ramp down, or square wave. One command sets the output level or pattern. Sample analog output programs are provided on the DCU CD.

The DCU Sample Programs CD

Sample programs for controlling the DCU are provided on the CD that comes with the package. There are several versions of the programs that allow you to control the DCU using either computers or calculators. The CD can be used on Windows, Macintosh, or Linux computers.

On the Windows/Linux segment of the CD we include:

- LabVIEW 6.1 or 7 programs (VIs) for Windows or Linux operating systems.
- DCU programs for all of the following Texas Instruments graphing calculators: TI-73, TI-83/83+/84+, TI-86, TI-89, TI-92, and TI-92+. You transfer these programs from your Windows computer to your calculator by using the TI-GRAPH LINK cable and TI software.
- Windows or Linux applications that are ready to run (no programming environment required) DCUTOGGL, DCUSTEP3, DCUSERVO, and FUNCTGEN. These applications will let you try out the DCU and analog output lines even if you are not a programmer.

On the Macintosh segment of the CD we include:

- LabVIEW 7 programs (VIs) which may be run on Macintosh OS X.
- LabVIEW 6.1 programs (VIs) which may be run on Macintosh OS 9.
- REALbasic 5.5 versions of the programs to run on Macintosh OS X.
- REALbasic 3.5 versions of the programs to run on Macintosh OS 9.

- DCU programs for all of the following Texas Instruments graphing calculators: TI-73, TI-83/83+/84+, TI-86, TI-89, TI-92, and TI-92+. You transfer these programs from your Macintosh computer to your calculator by using the TI-GRAPH LINK cable and TI software.
- Macintosh OS X or Macintosh OS 9 applications that are ready to run (no programming environment required) DCUTOGGL, DCUSTEP3, DCUSERVO, and FUNCTGEN. These programs will let you try out the DCU and analog output lines even if you are not a programmer.

How to Use the Sample Programs

The programs on the CD can be used in several different ways:

- Use them unchanged for controlling electrical devices you build. The sample program DCUTOGGL is especially good for trying out hardware connected to the DCU. It lets you turn on or off any of the six lines to test your hardware. The sample program FUNCTGEN is good for trying out hardware connected to the analog output line of LabPro. It lets you set a voltage level or control the function generator built into LabPro.
- Study them as examples to learn how to write your own programs. The DCUCOUNT program is a very simple program showing how to produce a pattern of outputs from the DCU. The program DCUTEMPC is a good example of reading a sensor and then taking action using the DCU based on the sensor reading. The program ANOUT is a simple program to show how the analog output line of LabPro is controlled.
- If none of the DCU programs will do what you need, build your program using our subroutines as "raw material." Start with one of the DCU sample programs. Delete the existing code that is not needed. Study the sample programs and see how the methods or subVIs are used. Carefully study *Appendix B* to learn what each subroutine does. Use these subroutines to save yourself work when you can.

Setting up the Computer Applications (for Non-Programmers)

Included on this CD are four computer applications that can be run without having a programming environment installed on your computer. If you will be running the programs using LabVIEW, REALbasic, or Texas Instruments graphing calculators you can ignore the instructions found in this section. The information for setting up the files for LabVIEW, REALbasic, or Texas Instruments graphing calculators can be found later in this manual, in the sections devoted to the specific programming environment.

The four computer applications, DCUTOGGL, DCUSTEP3, DCUSERVO, and FUNCTGEN can be run on Windows, Linux, Macintosh OS X, and Macintosh OS 9 computers following the setup directions below:

Windows (USB or Serial connection)

- Move the folder called "Win DCU Applications" to your hard disk in a convenient location.
- Each of the four programs has its own installer folder. Open the folder for the program that you wish to install and double-click on the file called "setup.exe". This will start the installation procedure. Follow the directions
- If you will be using a USB connection you must have the LabPro USB driver installed. This driver is automatically installed when you install Logger *Pro* 3.3. If you do not have Logger *Pro* 3.3 installed, then you must run the LabPro USB installer. You can find this installer at www.vernier.com/drivers.
- When all installations are complete the programs can be run one at a time. The programs will have the names DCUSERVO.exe, DCUSTEP3.exe, DCUTOGGL.exe, and FUNCTGEN.exe. The programs will be located on your computer in the location that you chose during the installation. Browse to a program and double-click to run.
- You should note that a folder called "data" was also created during installation. It is important to keep this folder and the program together, in the same directory.

Macintosh OS X (USB connection)

- The programs and support files are found in the folder called Mac OSX DCU Applications. Copy this folder to your hard disk in a convenient location.

- The LabVIEW 7.0 Mac OSX Runtime Engine must be installed on your computer. This installer is included in the Mac OSX DCU Applications folder. The name of the installer is "LabVIEW 7.0 Runtime Engine OSX". Run it to start the installation.
- You must have the USB driver installed. We do not have an installer for the Mac OS X driver. Therefore, you must have Logger *Pro* 3.3 installed. Installing a demo version of Logger *Pro* will also install this driver. A demo version can be found at www.vernier.com/downloads.
- When all installations are complete the programs can be run one at a time. Browse to a program and double-click to run.

Macintosh OS 9 (USB or Serial connection)

- The programs and support files are found in the folder called Mac OS9 DCU Applications. Copy this folder to your hard disk in a convenient location.
- The LabVIEW 6.1 Mac OS9 Runtime Engine must be installed on your computer. This installer is included in the Mac OS9 DCU Applications folder. The name of the installer is "LV6.1 Runtime and Plugin". Run it to start the installation.
- Make sure the extension LabPro USB is installed on your computer. This extension comes with the Vernier Logger *Pro* 2 or Logger *Pro* 3 program. You can find this installer at www.vernier.com/drivers.
- When all installations are complete the programs can be run one at a time. Browse to a program and double-click to run.

Linux (Serial connection)

- The programs and support files are found in the folder called Linux DCU Applications. Copy this folder to your hard disk in a convenient location.
- The LabVIEW 6.1 Linux Runtime Engine must be installed on your computer. This installer is included in the Linux DCU Applications folder. The name of the installer is "labview61-rte-6.1-1.i386.rpm". Install, as root, with the following command: # rpm –Uvh labview61-rte-6.1-1.i386.rpm.
- Change permissions of serial port(s) the LabPro will be connected to. Users need full access. For COM1: # chmod 777 /dev/ttyS0. For COM2: # chmod 777 /dev/ttyS1.
- When all installations are complete the programs can be run one at a time. Browse to a program and double-click to run.

Sample Program Descriptions

The names of all the DCU programs we have provided start with the letters DCU and have eight or fewer letters (so that the names match the calculator program names, which are limited to eight letters). The two analog output programs are named ANOUT and FUNCTGEN. On the following page is a list of all sample programs, with a short description of what they do.

Program Name	Description of Program
	Digital Control Unit Programs:
DCU2	Turns on D1-D6 in succession on DCU connected to DIG/Sonic 1 and then turns on D1-D6 in succession on DIG/Sonic 2
DCUALARM	Waits until Motion Detector detects an object closer than a specified distance. Turns on D1 for a few seconds. This is a typical program reading a sensor and waiting for an event before the digital output lines are used.
DCUCAR	Allows you to control a car, driven by two DC motors, using the four arrow keys. This program assumes you have a car with a DC motor powering each of the two drive wheels.
DCUCARS	Allows you to control a car, driven by two stepper motors (and stepper motor ICs), using the four arrow keys. This program assumes you have two stepper motors controlled by a stepper motor IC driving the two wheels of a car.
DCUCOUNT	This simple program sends each of the 16 possible digital outputs to the DCU lines. Counts 0-15 to show the resulting LED displays. A very simple program used in several examples in this manual.
DCUMASS	Program to turn on D1, D2, D3, and D4 in order to accelerate a magnet through a tube (mass driver). A very simple program much like DCUCOUNT.
DCUSERVO	Controls a servo motor connected to D1. This program works for many servo motors, including the one in the Vernier Robotic & Control kit.
DCUSTEP	Simple program that allows user to specify direction and number of steps for a directly-connected stepper motor (unipolar or bipolar). This is a simple, easy to understand, stepper motor program.
DCUSTEP3	Improved program that directly controls a stepper motor. This version is best to use if you plan to use the stepper motor for several different motions, one after the other. It keeps track of the stepper motor position as it moves. This program is more complicated than DCUSTEP, because it keeps track of the stepper motor positions.
DCUSUN	This program assumes that you have two auto-ID Vernier light sensors connected to CH1 and CH2 and a DC motor connected to the D1 and D2 output lines of the DCU. If the two light sensors are mounted pointing a few degrees apart and so they can be rotated by the motor, the program will turn the motor in the direction of the brightest light, so the light sensors follow a light source.
DCUTEMPC	Program for creating a temperature-controlled environment. It turns on a heater (D1) if temperature is below minimum temperature and turns on a fan (D2) if temperature is above maximum value. This is common feedback and control type program. It will work with any AutoID probe in CH1.
DCUTOGGL	Program that allows the user to toggle the digital lines with the keypad or mouse click. Note that it takes a second or so for this program to respond to a key-press on calculators. Also, not all possible combinations of outputs are possible. Use this program to test hardware that you wire up to the DCU.
DCUTRAP2	Used with a photogate connected to DIG/Sonic 2, this program can be used to make a live trap. It monitors the photogate to detect an animal, turns on D1 to take action, then pulses D2 on and off.
DCUTRAP3	Another live trap program, this one uses stepper motors. Monitors photogate connected DIG/Sonic 2 to detect animal, then activates a directly connected stepper motor to lower a trap door.
DCUWARNV	Monitors the analog sensor on Ch 1 of the LabPro/CBL 2 and turns on the D1 output of the DCU if the level exceeds a limit. This is a good sample program to start with for any project involving monitoring an analog signal and taking action based on the reading.
	Analog Ouput Control Programs:
ANOUT	This is a simple program to show how you can control the analog output line of the LabPro and the function generator. Use this program if you want to examine, modify or customize the code used for controlling analog output.
FUNCTGEN	This program is a fairly complete function generator program. You can use it to set a DC voltage output, or set up an output waveform. Use this program if you want to test hardware connected to the analog output line of LabPro.

Appendix C has several of the TI-83 graphing calculator versions of these programs listed line-by-line, so you can examine them.

Programming Overview

The next three sections of the manual are a look at low-level programming for LabPro/CBL 2. The first section, *Programming for Data Collection*, will concentrate on collecting data from sensors. Reading sensors is very useful in writing DCU and analog output control programs. Many interesting projects involve reading a sensor first and then turning on output lines in response. The second section, *Programming for the DCU*, details writing programs to control the DCU. The last section, *Programming for Analog Output Control*, provides details for controlling the LabPro analog output line and function generator.

These three sections are very important to REALbasic programmers and calculator programmers. LabVIEW programmers are encouraged to study this information to understand how the LabPro is controlled; however, many of the subVIs hide this code, making it possible to write useful programs without a complete understanding of these sections.

Two important points to make before jumping into the next three sections are how to send commands to the LabPro/CBL 2 and how to properly initialize the LabPro/CBL 2. This information is discussed below.

Sending Commands

LabPro and CBL 2 are similar devices and the commands you send to them for controlling data collection or output lines are generally the same. The way the commands are sent to the LabPro/CBL 2 depends on the programming language. The LabPro/CBL 2 is controlled by sending a list of comma-separated numbers, enclosed in curly brackets {}. The first number in this list is what we refer to as the *command*. Here is a list of a few command numbers and what they represent:

Command	description
0	All Clear/Reset
1	Channel Setup
3	Sampling Setup
4	Conversion Equation Setup
6	Stop Collection
9	Single-Point Data Collection

The numbers that follow the command in a list are referred to as the *parameters.* The parameters each have special meanings, depending on what command you are using. There is a default value for each of these parameters. If you leave the parameter off the list, the default value will be used. In many of our examples, we will only use parameters that are important and leave off the others (so they will take their default values). For a detailed list of the commands and parameters, refer to *LabPro Technical Reference Manual.* This manual is available free at www.vernier.com/diy (for "Do-It-Yourself").

Here is sample code for sending out a 3 command, with parameters 1,2 and 0. Notice the differences in how the command is sent in the various programming environments.

Using a TI-83:

```
{3,1,2,0}→L6
Send(L6)
```

Using REALbasic:

```
LabPro.Write("s{3,1,2,0}"+ chr(10))
```

Using LabVIEW: Create a string of text characters, "s{3,1,2,0}" and wire that string to the input node of a VI that sends commands to the serial or USB port of LabPro.

Initialization

An important first step of a DCU program for collecting data, controlling the digital output lines, or controlling the analog output driver is to initialize the LabPro/CBL 2 properly. The common commands used for this step are as follows:

Command 7: Ask for status from LabPro

Command 0: Reset the LabPro/CBL 2

Command 102: Set Power Control

In our initialization routines, we set the power control for the LabPro/CBL 2 to always stay on. If this command were not sent, the LabPro/CBL 2 will power down automatically (trying to save batteries), and you may get very unexpected results.

We recommend the use of an initialization routine at the beginning of all your programs. It is easiest to use the routines that we have already developed. With calculator and REALbasic programs this initialization is usually taken care of with the DCUINIT subprogram or a similar initialization routine. In LabVIEW, the Init&ChannelSettup Express VI does the job.

Programming for Data Collection

This section we will concentrate on the commands sent out and not worry about the details of the way the commands are sent. We will list our examples in pseudocode, which explains what the program is doing, but is not the exact characters (not the exact syntax) the programming language requires. The details that are specific to your programming environment (calculator, REALbasic, LabVIEW) will be covered in later sections of this manual.

Programs for reading analog sensors connected to the LabPro/CBL 2 usually follow a pattern like this:

- Initialize the LabPro/CBL 2. (a Command 0 is used)
- Use a Command 1 to set up the channels to which the sensors are connected.
- Use a Command 3 (or a Command 9) to initiate data collection.
- Get and process the data.

You will see this pattern repeatedly in the next sections which demonstrate different methods of collecting data from the LabPro/CBL 2. The first section demonstrates a non-real time data (NRT) collection program, where the data are stored in LabPro until all data have been collected, at which time they are retrieved by your program. The second section demonstrates real-time data collection, where a single data point is collected and immediately sent back from the LabPro, providing the user with a live look at the data. The third section shows how to take a single data point. This is very useful in feedback and control circuits. The fourth section provides an example program for collecting data from a non-auto ID sensor.

Data Collection in Non-Real Time (NRT) Mode

Here is an example of a non-real time (NRT) data-collection program, which simply means that the data will be stored in LabPro memory until collected by your program. This type of data collection is usually used when you want to take data very quickly, at hundreds or thousands of points per second. This program is configured to collect data from a Vernier auto-ID sensor (automatically recognized and calibrated by the LabPro/CBL 2) connected to Channel 1. This program will take 50 readings, 0.25 seconds apart. The specific commands and parameters are as follows:

Initialize with Command 0	Reset LabPro/CBL 2
Send Out {1,1,1}	Set up channel 1
Send Out {3,.25,50,0}	Start NRT data collection
Get Resulting Data	Get data

Command 0: The first line resets the LabPro/CBL 2. This is a common command for initializing at the beginning of a program. You will find this command, as well as other setup commands in all of our initialization subroutines or subVIs.

Command 1: The second line of our pseudocode is used to set up the channel for reading data. The 1 command specifies a number of things about how data is to be collected. The syntax for this command, when used with sensors is:

{1, *channel, operation, postprocessing, statistics, conversion*}

The first number, the "1", represents the command. The numbers that follow are the parameters. Not all of these parameters are important and you will rarely use most of them all. Here are the important parameters we use in this simple program:

- *Channel:* The input channel is specified with the second number in the list. For analog sensors, you can use 1, 2, 3, or 4. For a motion detector connected to Dig/Sonic 1, you use 11. For a motion detector connected to Dig/Sonic 2, you use 12.
- *Operation:* The third number in the command is usually a 1. This signifies to LabPro that an Auto ID probe is connected.

Command 3: The third line of our sample program controls the actual data collection. Here it specifies taking readings every 0.25 seconds for 50 readings, and no triggering is required. The syntax for this command is:

{3, *samptime, numsamp, trigtype, trigch, trigthres, prestore, extclock, rectime*}

Most of the parameters are not important and you should just leave them off, or use zeros. Here are the important parameters—the ones we have used in our sample program:

- *Samptime:* the time between samples (in seconds). The range is 0.0001 to 16000 seconds..
- *Numsamp:* the number of readings to be made. The number of readings can be any integer up to 12,287. (Numsamp = -1 has a special meaning, which will be explained in the next section.)
- *Trigtype:* this specifies if the program should wait for a triggering event before starting the actual data collection. 0 means no triggering. 1 means wait for the Start/Stop button on LabPro/CBL 2 to be pressed. Other numbers can be used to specify triggering on a certain signal level. The default is 1, which you usually do not want, so you should almost always put a zero here. Leaving this parameter off will often result in a program that does not seem to work properly.

Get Resulting Data: The last line of our sample program is used to retrieve the data from the LabPro/CBL 2. As the first lines of our sample program are executed, the LabPro/CBL 2 will go about its business of collecting the data. Now how do we get the data back to the calculator or computer? For NRT data collection the Get command is the answer. A Get command will retrieve a complete list of data from the LabPro/CBL 2.

When using LabPro connected to a computer, the data will appear in the serial or USB buffer of the computer after the Get command. In computer programs you often need to put a pause in your program code before the Get command to allow time for the data collection to be completed before using a Get command to retrieve the data from the serial or USB port buffer.

Data Collection in Real Time Mode

The discussion above assumes that you want to have a certain number of readings taken at specified intervals for later use by the calculator or computer (e.g., making a graph). There is another variation of data collection that is often used. We call this "real-time" data collection. In this type of program, the LabPro/CBL 2 takes one reading, the calculator or computer retrieves the reading (and usually does something with it, such as put a point on a graph), and then the program loops and repeats. This type of data collection is usually done when the number of points collected per second is low.

Here is a second sample program, similar to the first, but with live data collection:

```
Send Out {0}                initialize LabPro/CBL 2
Send Out {1,1,1}            set up channel
Send Out {3,1,-1,0}         start data collection

Label A                     Label this point in the code
Get Resulting Data          get data
Goto A                      Loop back to A and repeat

(Eventually some event such as a key press or a click, stops this loop, and data collection is stopped:

Send Out {6,0}              stop data collection
```

This program is the same as the previous sample through the first two lines; after that, the data collection portion is different. Here is what is going on in the last portion of the program: Notice that the 3 command has a -1 for the third parameter, as the number of samples. The -1 in the 3 command, as the number of points, causes the LabPro/CBL 2 to continuously take data.. The program then gets the data and loops back to point A in the program to get the next reading. This loop continues until the program is interrupted.

This type of data collection works great for many programs where you just want to monitor some reading from a sensor and display the data continuously (a live sensor display) or plot a live graph of the sensor reading vs. time. Calculator programs using Real Time need to have a Get Command to retrieve the data. Computer programs can simply check the serial/USB buffer to retrieve the data.

Single-Point Data Collection

When doing projects involving sensors and output lines, you often want to take a single analog reading, check the value and have the program make a decision on how to proceed based on the result. A simple example would be a program which turns on a fan connected to the DCU when a miniature greenhouse gets too hot. Single point data collection with a 9 command is perfect for situations like that.

The structure of a program using single point data collection is something like this:

```
Send Out {0}                initialize LabPro/CBL 2
Send Out {1,1,1}            set up channel

Label A                     Label this point in the code
Send Out {9,1}              request single reading

Get Resulting Data          get data
Goto A                      Loop back to A and repeat

(Eventually some event such as a key press or a click, stops this loop, and data collection is stopped:

Send Out {6,0}              stop data collection
```

The channel is set up just as before. The 9 Command has the format {9, channel}. It is repeated each time you want to retrieve one sensor reading from the LabPro. If you want to want to read more than one analog sensor, set each of them up with a 1 command and read each one with a separate 9 command.

Calculator programs using a 9 command need to have a Get Command to retrieve the data. Computer programs can simply check the serial/USB buffer to retrieve the data.

Data Collection with Non-Auto ID Sensors

Most analog Vernier sensors are auto-ID and are automatically calibrated when used with the LabPro/CBL 2. Some older analog Vernier sensors do not have this auto-ID feature. Also, homemade sensors will lack the auto-ID feature.

To read calibrated data from one of these non auto-ID sensors, you need to handle calibration in the program. Here is an example using NRT data collection:

Send Out {0}	initialize LabPro/CBL
Send Out {1,1,1,0,0,1}	set up channel
Send Out {4,1,1, 0.809, 0.077 }	specify calibration
Send Out {3,0.25,50,0}	start data collection
Get Resulting Data	get data, measured values first, then times

This program is similar to the first example in this section but the difference is that this program is for a sensor that does not have the auto-ID feature. The changes are in the second and third line.

Command 1: The second line of our pseudocode is used to set up the channel for reading data. As mentioned earlier, the full syntax for this 1 command, when used with sensors is:

{1, *channel, operation, postprocessing, statistics, conversion*}

In the auto-ID probe example, we only used the channel and operation parameters. This example also uses the conversion parameter. If the conversion parameter is set to 1, it tells the LabPro that it should use the calibration provided in the 4 command that follows.

Command 4: The third line of our sample uses the Conversion Equation Setup Command. Command 4 is only used to set up the calibration equation for a non-auto-ID analog sensor. With the proper Command 4 line, the LabPro/CBL will read voltage and convert to the correct values (atmospheres, newtons, degrees, etc). The syntax for this command is:

{4,*channel, equation type*, k_0, k_1}

There are lots of options for this command, and the details are in the *LabPro Technical Reference Manual*. Fortunately, most probes use linear calibrations (1st order polynomial, equation type = 1) and things are fairly simple. The parameter k_0 represents the y-intercept and the parameter k_1 represents the slope (units/volt).

In our sample program, this means that Channel 1 raw voltage will be converted to proper units using a linear equation with a y-intercept 0.809 atm and a slope of 0.077 atm/volt. This is the proper calibration for a Vernier Barometer, calibrated in atmospheres.

Information on the proper slope and y-intercept values for each Vernier sensor is included in the sensor's documentation.

Programming for Data Collection Summary

This has been a summary of the most important things about writing LabPro/CBL 2 programs to collect data. There is much more you can learn. Even so, this should help you get started. The best ways to learn programming are:

- Study other programs that read sensors, including the sample programs on our CD, such as DCUTEMPC and DCUWARNV or the computer programs RT and NRT. Check out programs that your teachers or friends have written, or download some from the Internet.
- Jump right in and start changing the programs and notice what happens.
- Refer to the *LabPro Technical Reference Manual* for more information.

Data-Collection Programming Challenges

- Write a program to read the temperature with an auto-ID temperature probe every second for 30 seconds and then display all the temperatures.
- Write a program to read and display the temperature read by an auto-ID temperature probe every second until the user stops the program.
- Write a program to read and display the temperature read by a non auto-ID temperature probe called a Direct-Connect Temperature Probe. This type of probe produces a voltage output of 0 volts at -17.8 degrees C and the voltage output increases by one volt for every 55.56 degrees C.

Programming for the DCU

The most important piece of information when programming the DCU is the table that lists the 16 possible output patterns. This table was first introduced in the "DCU Overview" section. Here it is again:

Output	Binary	D1	D2	D3	D4	D5	D6
0	0000	—	—	—	—	X	X
1	0001	+	—	—	—	X	X
2	0010	—	+	—	—	X	X
3	0011	+	+	—	—	X	X
4	0100	—	—	+	—	X	X
5	0101	+	—	+	—	X	X
6	0110	—	+	+	—	X	X
7	0111	+	+	+	—	X	X
8	1000	—	—	—	+	X	X
9	1001	+	—	—	+	X	X
10	1010	—	+	—	+	X	X
11	1011	+	+	—	+	X	X
12	1100	X	X	X	X	—	—
13	1101	X	X	X	X	+	—
14	1110	X	X	X	X	—	+
15	1111	X	X	X	X	+	+

The numbers in the "Output" column of this table are frequently used when sending commands to the LabPro/CBL 2. The number that is sent will determine what line, or lines, are turned on and off.

There are two basic approaches when performing digital output. One is to command the LabPro/CBL 2 to turn on a line, or lines, and hold those lines on until told otherwise. The other is to command the LabPro/CBL 2 to perform a sequence of outputs, so that at a specific interval a new line, or lines, is turned on. This sequence of outputs will continue to loop until the total number of steps are completed.

These two approaches are explained with more detail in the next two sections.

Controlling the Digital Output Lines – Direct Control

After you initialize the LabPro/CBL 2, if all you want to do is to set the status of the lines (turn something on or off), you can use a 2001 command. Send a command with this format:

Send Out {2001,D}

Where D is the output pattern from the table above that you want to set and hold. Remember that in LabVIEW and REALbasic, commands and parameters must be formatted as strings.

Example 1: Turn on digital output line D3, and leave it on until you send another command to turn it off.

Send Out {2001,4}

(If you do not understand why a "4" is used as the output pattern, study the chart just above.)

Example 2: Turn off all the digital lines.

Send Out {2001,0}

Controlling the Digital Output Lines – Sequence Control

Producing a series of digital output patterns is very useful in a number of situations, including: flashing lamps, running stepper or servo motors, or making a speaker vibrate to make sound. On the other hand, this kind of control is a little more complicated. Programming to control the digital out lines in this manner is similar to programming for collecting data. The important steps are:

1. Initialize the LabPro/CBL 2.
2. Use a Command 1 to set up the digital output channel (31) for a sequence of digital output patterns.
3. Use a Command 3 to initiate the sequence of outputs and specify the total number of steps to be taken.

Example 1: Here is a very simple example. This program flashes on and off the first three lines of the DCU.

```
Call the subroutine DCUINIT        intialize
Send Out {1,31,2,7,0}              set up sequence
Send Out {3,0.5,8,0}               start through the sequence
```

Initialize: The first line uses the DCUINIT subprogram to initialize the LabPro/CBL 2.

Command 1: The Command 1 line is used to set up the digital output channel and specify the sequence of output patterns to be used. Command 1 is a little different when used with digital output than it was with sensors. The syntax for Command 1, when used with digital output, is:

{1, *channel, # of output patterns, patterns*}

Here are the important parameters:

- *Channel:* For digital output, you use a 31 for the DIG/Sonic1channel. If you are using a LabPro, you can also use a DCU connected to DIG/Sonic2, and in this case the channel is 32. All of our sample DCU programs, except DCU2 (for LabPro only), use 31 for the digital output channel.
- *# of output patterns:* The third number in the list is used to specify the number of patterns in the sequence to be output. The number of patterns can range from 1 to 32. In our sample program, the number 2 is used. This means there will be two output patterns in the sequence.
- *Patterns:* This is a list of numbers specifying the output patterns to be used. Again, these are the numbers from the table above. In our example, they are 7 and 0. Note that in this example since the # of output patterns set in the command was 2, then there have to be 2 numbers in this list.

Command 3: The 3 command is used much like it is with sensor programs.

{3, *samptime, numsamp, trigtype*}

The important parameters, and what they mean in this example are as follows:

- *Samptime:* This is the time between samples in seconds. The range is 0.0001 to 16000 s.
- *Numsamp:* This is the total number of steps to take through the sequence. This can be any integer from 1 to 12,287 (or –1, for a continuous sequence). Note that this number does not have to match the # of output patterns used in the Command 1. For example, in the sample program above, # of output patterns is two (there are two output patterns in the sequence). The numsamp used in Command 3 is eight. This means that the program will go through eight steps. When it finishes the sequence of patterns specified in Command 1, it will start through the sequence again. The sequence of two elements will be repeated four times, for a total of eight steps. Think of the sequence as a loop that will be worked through as many times as needed until all the steps have been executed.
- *Trigtype:* The default value of this parameter is 1, which is for manual trigger. (You have to press the Start/Stop button on the LabPro/CBL 2 to start the digital output going.) You usually do not want this, so almost always use a zero here. This will have the digital output start as soon as the program executes the step.

So what does the sample program above do? It has the digital output lines go through the following output pattern: 7,0,7,0,7,0,7,0. The sequence 7,0 was specified in the Command 1 line and this is repeated until 8 steps are made (as specified in Command 3). If the DCU were connected when this program runs, the first three red LEDs would flash on and off four times. The status of the LEDs would change every 0.5 seconds.

Controlling a DCU Connected to DIG/Sonic 2 or Two DCUs at Once (LabPro Only):

Most of the discussion in this manual has assumed that the DCU was connected to the DIG/Sonic 1 connector. We use numbers between 0 and 15 to control the status of those lines. If you are using a LabPro, you can connect a DCU to the DIG/Sonic 2 connector. You can use any number from 0 to 255 for D in the 2001 command and control any of the lines on either DCU. Below is a table of how the 2001 command can be used to set the pattern on outputs on a DCU connected to the DIG/Sonic 2 connector.

Output	Binary	D1	D2	D3	D4	D5	D6
0	0000	—	—	—	—	X	X
16	0001	+	—	—	—	X	X
32	0010	—	+	—	—	X	X
48	0011	+	+	—	—	X	X
64	0100	—	—	+	—	X	X
80	0101	+	—	+	—	X	X
96	0110	—	+	+	—	X	X
112	0111	+	+	+	—	X	X
128	1000	—	—	—	+	X	X
144	1001	+	—	—	+	X	X
160	1010	—	+	—	+	X	X
176	1011	+	+	—	+	X	X
192	1100	X	X	X	X	—	—
208	1101	X	X	X	X	+	—
224	1110	X	X	X	X	—	+
240	1111	X	X	X	X	+	+

DIG/Sonic 2 digital Output Control

For example, sending a {2001,16} will turn on line 1 of a DCU connected to DIG/Sonic 2.

If you want to control two DCUs at once, you can send out the sum of the two outputs that you would send separately to set them both at once. For example, sending a {2001,17} will turn on the D1 line of both DCUs. Sending {2001,34} will turn on the D2 lines on both DCUs.

There are two additional commands for directly controlling the digital output lines. These commands control only one of the two DIG/Sonic ports, not interfering with anything going on at the other DIG/Sonic port. Use the 2011 command to control DIG/Sonic 1 without affecting DIG/Sonic 2. The same numbers are used for the output, as with the 2001 command, so

Send Out {2001,1}

and

Send Out {2011,1}

are the same, except that if some of the digital output lines of Dig/Sonic 2 happen to be on, the 2001,1 command would turn them off and the 2011,1 command would not change their status.

Similarly, the 2012 command controls just the DIG/Sonic 2 digital output lines.

Send Out {2012,16}

Turns on the first line of a DCU connected to DIG/Sonic 2 and will not change the status of a DCU on DIG/Sonic 1.

(Note: The 2011 and 2012 commands are only available if you are using a LabPro that has had its operating system updated fairly recently. Specifically, operating systems 6.246 and newer support these commands. Free operating system updates are available at www.vernier.com/downloads.)

You can do sequence control of a DCU connected to Dig/Sonic 2 by setting up channel 32, instead of 31 as we did for sequence control for a DCU connected to Dig/Sonic 1. For example, the program below would flash on and off the first three LEDs of a DCU connected to Dig/Sonic 2.

```
Call the subroutine DCUINIT          intialize
Send Out {1,32,2,7,0}                set up sequence for DCU connected to DIG/Sonic 2
Send Out {3,0.5,8,0}                 start through the sequence
```

DCU Programming Challenges:

- Have the DCU red LEDs count in binary from 0 to 11.
- Turn on the 5th and 6th LEDs for 10 seconds.
- Turn on each of the six red LEDs, one at a time.
- Have the DCU turn on the D1 red LED if the temperature of a liquid drops below 30 degrees C.
- Have the DCU indicate approximate temperature by turning on the appropriate LED to indicate the Celsius temperature to the nearest 10 degrees (D1 indicates in 10 to 19.9 degrees, D2 indicates 20 to 29.9 degrees, etc.)

Programming for Analog Output Control

A feature of the LabPro that is not used as often as it should be is the analog output line and its built-in function generator. The analog output line is accessed by connecting the voltage probe which came with LabPro to the CH4 connector. The analog voltage out is across the two connectors of this probe. This section will explain how you can control the analog output line in your programs.

The analog output line is controlled using only the 401 command. This one command allows you to set a steady DC voltage output level, or to turn on the LabPro's function generator. When the function generator is used, you can set the waveform, amplitude, offset, and period of the waveform. The format of this command is:

{401, *waveform, amplitude, offset, period*}

By changing the parameters you may change the output value. The details of setting up the parameters are explained below. Note that sine waveforms are very different from other waveforms and they are explained in a special section below.

Waveform: There are 7 possible settings for waveform:

Waveform	Description
0	OFF
1	DC Output
2	Ramp Up (Sawtooth)
3	Ramp Down (Sawtooth)
4	Triangle
5	Square
6	Sine

Amplitude: This is the peak to peak voltage in units of 0.0024 volts, except in the case of sine waveforms. The practical range is 0 to 1667. This parameter cannot be negative.

Offset: the base voltage relative to ground in units of -0.0012 volts, except in the case of sine waveforms. The practical range is 0 to 3333. For most simple waveforms and for positive DC voltages, just set this to zero.

Period: time (in milliseconds) to complete one cycle of the waveform. The range is 5 to 2000, which corresponds to a range of 200 Hz to 0.5 Hz.

In practical use, here is how you will probably want to use the 401 command:

For Ramp (Sawtooth), Triangle, and Square Waveforms made up of only positive voltages:
Choose the waveform you want (2 = Ramp Up, 3 = Ramp Down, 4 = Triangle, 5 = Square) . Use zero for Offset. Set the amplitude parameter by dividing the peak voltage you want for the waveform by 0.0024 volts. Set the period as you want, remembering it is in milliseconds. To calculate the period if you know the frequency, use this calculation:

period (in ms) = 1/ frequency * 1000

For Positive DC (steady) Voltages
Use waveform = 1, offset and period = 0. Set the amplitude parameter by dividing the positive voltage you want by 0.0024 volts. For example, to get a steady +2.4 volts output, use 401, 1, 1000, 0,0

For Negative DC (steady) Voltages
This is a little strange, because you set things up for negative voltages using the offset parameter. Use waveform = 1, amplitude and period = 0. Set the offset parameter by dividing the negative voltage you want by -0.0012 volts.

For example, to get a steady –2.4 volts output, use 401,1, 0,2000,0

To Set the Output Voltage to Zero
Use waveform = 1, amplitude, offset, and period = 0. Note that this is not quite the same as setting the analog output waveform = 0 (Off), which leaves the analog output disconnected and perhaps floating at a non-zero voltage.

For Sine Waveforms Centered at Zero
Sine waveforms are very different from all others. The reason is that these waveforms are created using a look-up table of sines stored in the LabPro. All other waveforms are created by simple calculations done by the LabPro microprocessor. Only a few different amplitudes are possible, and the amplitude and offset parameters have different meanings than with other waveforms.

For sinewaves centered at zero, your only choices for amplitude are

4-Volt Sine Waveform (8 volts peak-to-peak)
401,6,0,3072,T, where T is period in milliseconds

2-Volt Sine Waveform (4 volts peak-to-peak)
401,6,1,1536,T, where T is period in milliseconds

1-Volt Sine Waveform (2 volts peak-to-peak)
401,6,2,768,T, where T is period in milliseconds

If you want to experiment with waveforms other than the ones described above, such as ramp, triangle, and square waveforms that include negative voltages, try changing the offset value. The general equation for voltage control is

V_{out} = 2.4mV*amplitude –1.2mV*offset

Here are some examples of possible waveforms:

For a 100 Hz, ramp up waveform, 5 volts peak to peak, centered at zero, use:
401,2,2083,2083,10

For a 50 Hz, ramp down waveform, 2 volts peak to peak, centered at –1 V, use:
401,3,792,1667,20

For a square waveform, 4 volts peak to peak, centered at zero, use:
401,5,1667,1667,T (T is period in ms)

Analog Output Programming Challenges

- Write a program to set up a 2 Hz triangle waveform with an amplitude of 3 volts.
- Write a program to produce a 4-volt sine wave with a frequency controlled by user input.
- Write a program to produce a steady DC voltage (positive or negative) controlled by user input.

Using the DCU with a Calculator

This section is intended to introduce you to how the DCU works with a TI graphing calculator and how it is programmed. Skip this section if you are not using a calculator with the DCU.

You do not need any extra hardware connected to the DCU to work through these examples. If you have a small speaker, you can use it for one part of this section. We try to make things as simple as possible, but introduce you to all of the following:

- Getting the calculator software loaded.
- Connecting the DCU to LabPro.
- Running the DCUTOGGL (8-character version of "DCU Toggle") program to control the DCU lines.
- Running the simple DCU program DCUCOUNT.
- Examining and modifying the DCUCOUNT program.

Follow the steps below to get your DCU operating using the DCUTOGGL and DCUCOUNT programs:

1. Load all the programs from the appropriate folder of DCU programs into the calculator using the TI-GRAPH LINK cable. There are separate folders of DCU sample programs on the CD for TI-73, TI-83/83+/84+, TI-86, TI-92, and TI-92+.
2. Connect the DCU to the connector on a LabPro labeled DIG/Sonic 1 or the connector on the top right side of the CBL 2 labeled DIG/Sonic. Make sure this connector locks in place.
3. Connect a LabPro power supply (IPS) to the round connector on the DCU. Note the LabPro and CBL 2 do not have an on/off switch and they will turn on or off automatically.
4. Connect the calculator to the LabPro/CBL 2 using the round calculator link cable. Make sure this connector locks in place.
5. Start the DCUTOGGL program. If you complete all of the steps above correctly, the green LED on the DCU should turn on momentarily. If the green LED does not come on, double-check all the steps. Note that the electronics in the DCU can sense when it is properly connected to the LabPro/CBL 2 and when the unit has power.
6. Try pressing the 1 key on the calculator. If everything is working properly, the red LED labeled 1 should go on. Press the 1 key again and it should go off. Try to turn on and off the other LEDs, using the 2 through 6 keys. Note that there are some combinations of LEDs that are not allowed, so that in some cases when you turn on an LED, some others may go off. This DCUTOGGL program is very useful in testing hardware that you connect to the DCU.
7. Quit the DCUTOGGL program by pressing the + key.
8. Start the DCUCOUNT program. The red LEDs should go through a series of patterns, with a change every second. There are 16 steps that are the possible output patterns of the DCU. They represent sending the numbers 0, 1, 2, 3, 4, … 15 to the DCU from the calculator, followed by a 0 to turn all the outputs off again. Notice that the first 12 steps correspond to binary counting using the first 4 red LEDs. The numbers 12 to 15 control the status of the last two lines.

To give you an introduction to what DCU programs are like, the DCUCOUNT program code is listed below. This is TI-83 code, but the code for other calculators is similar. There are a number of things to notice about this short program:

prgmDCUINIT	Calls a subprogram named DCUINIT. This subprogram initializes the LabPro/CBL 2.
Disp "COUNTING"	Displays message on calculator screen

{1,31,17,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0}→L6	This line sets up a list which will be used as a 1 command to set up the DIG/Sonic 1 digital output channel of LabPro.
Send(L6)	Sends this list to the CBL 2/LabPro.
{3,1,17,0}→L6	Sets up a list starting with a 3, followed by some other numbers. This command will control the output from the LabPro/CBL 2.
Send(L6)	Sends this list to the LabPro/CBL 2.

When we ran this program, we saw that the DCU went through a series of steps with the output pattern changing. How is that accomplished? The 1 command sets up a sequence of outputs, with 17 patterns. The 17 numbers that follow are the patterns. The 3 command tells the LabPro to step through each pattern, taking 1 second per pattern.

It's now time to customize the DCUCOUNT program. We will make three changes. The first one is to make the program step through the patterns more slowly. Examine the program code on the calculator screen. The details of doing this are slightly different on various TI graphing calculators. On TI-73, TI-83, TI-83 Plus, and TI-84 calculators you do this by pressing the PRGM key, and then the right arrow to select EDIT and then the ENTER key. Scroll down the list of programs until you come to DCUCOUNT and then press ENTER. Refer to your calculator manual for information on how to edit the code on other calculators. The program should now be listed on the screen.

Recall that the DCUCOUNT program initially stepped through the sequence one second at a time. The 3 command controlled the pace of this process. The second number of the 3 command is originally 1, so the steps of the sequence each last 1 second. Try moving the cursor to the 1 and changing it to a 2. Then press 2nd and then QUIT and run the program again. The program should then go through the sequence more slowly.

Now we want to add a feature to the program by using a subprogram. We want it to buzz a speaker before the LED display sequence starts. If you have a small speaker available, connect the speaker wires between the D1 connection and a Ground connector. If you do not have a speaker, you can just watch the LEDs and see if the buzzing action is taking place. Again display the DCUCOUNT program on the calculator screen. Now, we need to add the following lines to the program, just after the prgmDCUINIT.

```
50→F
5→T
prgmDCUBUZZ
```

These lines will tell the program to execute the subprogram DCUBUZZ using 50 for the frequency (F) and 5 for the number of seconds duration (T). You add this code to the program by going to the start of the line after where you want to add the new code and pressing 2nd INS (for insert). Then type in the lines. You will need to use the ALPHA key to type the letters and the PRGM key, selecting from a list of commands to insert the prgm character. Refer to your calculator manual if you are not familiar with editing programs. Now press 2nd and QUIT and run the program again. The program should first buzz the speaker at a low frequency for 5 seconds then go through the pattern. If you do not have a speaker, just look at the #1 LED; it should flicker.

For our final change, we want to add to the end of the program. We will have the program wait for a key press and then turn on the D1 and D2 lines and leave them on. To do this we want to add a Pause command (for calculator keyboard input) followed by a 2001, 3 command to turn on both lines 1 and 2 of the DCU.

```
Pause
{2001,3}→L6
Send(L6)
```

The final, modified program should look like this:

```
prgmDCUINIT
50→F
5→T
prgmDCUBUZZ
Disp "COUNTING"
{1,31,17,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0}→L6
Send(L6)
{3,1,17,0}→L6
Send(L6)
Pause
{2001,3}→L6
Send(L6)
```

You now have one example of how a calculator program can control the DCU. This example shows how subprograms can be used to make programming easier. We encourage you to experiment with our other subprograms to develop your own programs. There is a lot more to learn. The best advice we have is to experiment. The best way to learn programming is to actually do it.

Additional Notes on Calculator Programming

This section offers specific tips for people writing TI calculator programs. For more information, also see the *LabPro Technical Reference Manual*, *Getting Started with the CBL 2*, and the manuals that came with your calculator. The exact details of programming differ somewhat between calculator models. You should carefully study the manual that came with the calculator you are using. Concentrate on the sections about programming. Also, examine some of the programs that you use with the calculator. No matter what calculator model you are using, the commands you send to the LabPro/CBL 2 are the same. All the sample programs listed in this manual use the syntax of the TI-83, or TI-84 calculators. To see versions of the same programs for other calculators, simply open the appropriate version of the program from the CD, using TI-GRAPH LINK or TI Connect.

Our number one tip on programming for TI calculators is that most programming should be done using a computer and typing on a keyboard. Use TI-GRAPH LINK or TI Connect for the editing and then send the program to the calculator. Typing or modifying a program on the calculator keyboard is much harder. Also, most versions of TI-GRAPH LINK have built-in tools to make writing lines of code for the LabPro/CBL 2 easier. Check the TI manuals for information on how to use this feature.

Interrupting the Program

Some of our calculator programs are very simple and do not even have a graceful way of stopping them. Remember that on all TI graphing calculators, repeatedly pressing the [On] key will stop the program.

Always start with the DCUINIT SubProgram

You will save yourself a lot of trouble if you make the first line of every program: `prgmDCUINIT` . If you do this, your programs will provide good feedback if they cannot communicate with the LabPro and the LabPro power-down feature will be turned off, so you do not get unexpected results.

Setting the DCU Output

If you are using a LabPro and CBL 2, the command 2001 makes it easy to set the output pattern of the DCU. To use it, you simply use:

```
{2001,D}→L6
Send(L6)                    Set output pattern to D
```

Where D is the output pattern you want to set and hold. It can be any integer from 0 to 15.

To turn just the first digital output line on and leave it on until you send another command to turn it off, use

```
{2001,1}→L6
Send(L6)
```

Having the DCU Output Hold a Setting for a Specific Time

Many times you want to turn on a line for a specific time period. The easiest way to do this is to use the DCUPWRON subprogram. First set the output pattern you want in the variable D, and the time the pattern is to hold (in seconds) in the variable T. After that, call the subprogram. Here is an example of turning on the first three DCU lines for 10 seconds.

```
7→D
10→T
prgmDCUPWRON
```

Using -1 in the Command 3 Line of DCU Calculator Program

As with programs for sensors, you can use -1 as the number of readings in the command 3 line. In this type of program, the LabPro/CBL 2 will step through the patterns of outputs each time the Get command is encountered. You must set up a sensor channel, and then use a Get command to step you through the pattern. Here is a sample program showing how the DCU can be controlled in this way:

```
prgmDCUINIT                 Initialize the LabPro/CBL 2
{1,1,1}→L6                  Set up CH1 to read an AutoID sensor
Send→(L6)
{1,31,6,1,2,4,8,13,14}→L6   Set up digital output to turn on six lines in order, one at a time
Send(L6)
{3,1,-1,0}→L6               Set up the LabPro/CBL 2 for one step at a time
Send(L6)
Lbl A                       Label for looping
Get(I)                      Get the (meaningless) reading from the LabPro/CBL 2
Goto A                      Repeat
```

This program turns on the six lines of the DCU, one at a time, in order. Notice that the timing of the program execution is still controlled by second number in the 3 command.

Calculator users may also be confused about the keystroke to use to type the "–" character. TI uses two different "-"character on their keyboards. For the TI 83/84 families, you want to use the key labeled "(-)", not the "-", subtract key.

Reading a Sensor at the Same Time that You Control the DCU

Often you will want to read the status of a sensor and use this to decide what the program should do with the DCU. For example, you may want to check a temperature and turn on a fan motor if the temperature gets too hot. Here is a sample program that shows this idea. This program is for a DCU-controlled alarm system. It monitors an analog sensor on channel 1 and sets off a buzzer, connected to D1 when the voltage goes over one volt.

```
prgmDCUINIT                 Initialize the LabPro/CBL 2
{1,1,1}→L6                  Set up an AutoID sensor in channel 1
Send(L6)
1→V                         Sets the limit at 1 unit
0→S                         Initialize reading from the sensor
{3,1,-1,0}→L6               Take a reading
Send(L6)
While S<V
Get(S)                      Get the sensor reading until it is equal to or greater than V
End
1→D                         Set output pattern to be used
10→T                        Set how long the output pattern will be held
prgmDCUPWRON                Turn on the buzzer
```

Notice this program uses live (real time) data collection, with a –1 in the 3 command. When data collection is set up this way, you can loop through a section of code and execute a GET statement many times. Each time through, one reading is taken from the sensor. When the conditions to break out of the loop are met, the DCUPWRON subprogram is used to turn on the buzzer.

Taking a Single Reading While Not Collecting Data

Sometimes you simply want to take one reading from an analog sensor, say to make a decision on what to do in a DCU or analog output program. The 9 command is the easy way to do this. Here is a sample program

```
prgmDCUINIT                 Initialize the LabPro/CBL 2.
{1,1,1}→L6                  Set up CH1 for reading an AutoID temperature probe
Send(L6)
Lbl A                       Label for looping
{9,1}→L6                    Send 9 command
Send(L6)
Get(L1)                     Get value from sensor, reading result into a list
L1 (1) →S                   Take the first number and the list and set equal to the reading.
Disp S                      Display the temperature reading on the calculator
If S<23                     Start a loop that will repeat until 23 degrees is reached
Goto A                      End of the loop
{2001,1}→L6                 Turn on line 1
Send(L6)
```

This program will loop until the temperature reaches 23 degrees and then turn on DCU line 1.

Making Sure the LabPro/CBL 2 Finishes Its Work before Your Program Moves On

One thing that can cause confusion when programming the DCU is when the calculator goes on with its program before the LabPro/CBL 2 has time to finish what it is doing. This can cause two different types of confusion. For example, consider this program:

```
prgmDCUINIT                 Initialize the LabPro/CBL 2.
{1,31,2,0,7}→L6             Set up the digital output to turn the first 3 lines on and off
Send(L6)
{3,1,100,0}→L6              Go through 100 steps, each taking 1 second
```

```
Send(L6)
Disp "DONE"                    Display message on calculator when program is finished
```

When you execute a program like this, you may be surprised that the calculator indicates that the program is finished, but the LabPro/CBL 2 and DCU are still doing something with the digital output lines. This is because the LabPro/CBL 2 got its command to do the sequence of outputs and they are continuing, even though the calculator went on executing the rest of the program.

A slightly different situation happens when you send commands to the LabPro/CBL 2 to have the DCU do something else before the first operations sent are completed. Consider this program, for example:

```
prgmDCUINIT                    Initialize the LabPro/CBL 2
{1,31,2,0,7}→L6                Set up digital output to turn on and off the first three lines
Send(L6)
{3,1,10,0}→L6                  Go through 10 steps, taking 1 second for each step
Send(L6)
{2001,0}→L6                    Turn off power to all lines.
Send(L6)
```

If you try this program, it will not operate the way you might expect. If the DCU is connected, you might expect to see the first three red LEDs flash on and off for ten seconds and then the power should go off. Instead, the LEDs will briefly flash, but the program quickly ends with the power turned off. Why?

The problem is that the LabPro/CBL 2 starts executing the sequence of 10 steps, but then is sent a new command telling it to turn the power off. It interrupts what it was working on and follows the new instruction.

There is a solution to these problems. The trick is to have the LabPro/CBL 2 do a (probably unnecessary) sensor reading at the same time. Then you can use a GET calculator program statement to retrieve the sensor reading from the LabPro/CBL 2. In this case, the calculator program will know that it cannot go on until the LabPro/CBL 2 has finished its operation and returned readings to the GET instruction. Here is the same program revised so that first sequence of outputs will be completed before the power is turned off:

```
prgmDCUINIT                    Initialize the LabPro/CBL 2
{1,1,1}→L6                     Set up CH1 to read a sensor
Send(L6)
{1,31,2,0,7}→L6                Set up digital output to turn on and then off the first three lines
Send(L6)
{3,1,10,0}→L6                  Go through 10 steps, each taking 1 second
Send(L6)
Get(L1)                        Get the reading. (The program will wait.)
{2001,0}→L6                    Set up digital output lines for turning power off to all lines
Send(L6)
```

Here are some things to remember about the programming trick discussed in this section:

- You do not need to have a sensor connected when you use this trick.
- It makes no difference what value is returned in the GET statement. It is just a way of forcing the calculator program to wait for the LabPro/CBL 2 to finish its work.
- If you happen to want your program to take a sensor reading at the same time it is executing some pattern of digital outputs, then this same idea will work, and produce useful data.

Another good reason for using subprograms in your DCU programming is to avoid confusion of this sort. If you need to turn on a line for a specific time, use the subprogram DCUPWRON. Simply specify the output you want (D), and the time you want the pattern held (T), and all this will be taken care of for you.

Reading a Sensor as a Program Goes through a Sequence of Outputs

Sometimes you will want to read the status of a sensor at the same time you are stepping through a sequence of output patterns. For example, you may want to flash a LED, or operate a stepper motor while you are monitoring a sensor. Here is a sample program that shows how this can be done:

`prgmDCUINIT`	Initialize the LabPro/CBL 2.
`{1,31,4,5,9,10,6}→L6`	Set up digital output lines for running a stepper motor.
`Send(L6)`	
`{1,1,1}→L6`	Set up CH1 for reading an AutoID temperature probe
`Send(L6)`	
`0→T`	Initialize a variable to represent temperature
`While T<23`	Start a loop that will repeat until 23 degrees is reached
`{3,.2,4,0}→L6`	Sample sensor and change the digital output
`Send(L6)`	
`Get(L4)`	Get the temperatures
`mean(L4)→T`	Calculate the mean temperature
`Disp T`	Display the mean temperature reading on the calculator
`End`	End of the loop

This program will operate a stepper motor (or just flash the LEDs on the DCU) as it reads temperatures. It repeats this until the temperature reaches 23 degrees and then it stops.

Getting User Input

In calculator programs you often want to ask for user input. The DCUTEMPC program is a good example. It uses the following code to ask the user to enter a temperature.

```
Disp "ENTER MIN TEMP"
Input W
```

The value typed in will be stored in variable W.

Later in the program, the program goes into a data collection loop. To allow the user to specify when the data collection loop should end, the following code is used:

```
Lbl A
getKey→K
If K≠95
Goto A
```

Keycode 95 represents the + key on a TI-83 calculator. The loop will repeat until this key is pressed.

An alternative way to allow the user to control when data collection ends would be to use the statement below:

While getKey=0.

[data collection code here]

End

In this example, the data collection code will be repeated until any key is pressed.

Subroutines

We have also included a number of small subroutines on the CD for you to use. On the TI calculators these are actually separate programs, which we refer to as *subprograms*. These subroutines are intended to supply sections of code that you can use to write your programs. Major operations that you want your program to handle can be replaced using one of the subroutines. Many programs can be written by linking together these subroutines with a few lines of new code. As an example, consider the program DCUTRAP2. As mentioned earlier, this program is used to make a bug trap. It uses a photogate to sense when a bug is inside a box. It then turns on a motor to knock the lid of the box closed. Here is a generic description of this program, with some comments explaining what the program does.

ClrHome	Set up the calculator screen
Disp "READY FOR ACTION"	Display "READY FOR ACTION"
prgmDCUINIT	Use the subroutine DCUINIT to initialize the LabPro/CBL 2
prgmDCUCHKD	Check to see if the photogate is working
ClrHome	Clear the calculator screen again.
prgmDCUWAITD photogate	Use the subroutine DCUWAITD to monitor the status of the and wait until the photogate is blocked
1→D to turn on	Set the value of the variable D, used to specify which outputs
1→T output is on	Set the value of the variable T, used to control the time the
prgmDCUPWRON	Use the subroutine DCUPWRON to turn on the motor, connected to D1 (D=1), for 1 second (T=1)
2→D	Set D to 2 because the buzzer is connected to the D2 line
prgmDCUPULSK connected to D2	Use the subroutine DCUPULSK to turn on the buzzer, and leave it on until a key is pressed.

Notice that there is not much to the main program. All the tricky stuff is done in the subroutines DCUINIT, DCUCHKD, DCUWAITD, DCUPWRON, and DCUPULSK.

Subroutines are used by calling them in the code with the value of certain variables set. For example, with the DCUPWRON subroutine, you call it with D set for the digital output pattern you want on and T set to the time you want the pattern to stay on.

Using the DCU with REALbasic and a Macintosh Computer

Introduction

This reference guide is intended to be used by people who want to get their DCU up and running as quickly as possible using a Macintosh computer and REALbasic. It assumes that the user has some familiarity with RealSoft's REALbasic. Information on this programming language is at www.realsoftware.com. We have two versions of programs using REALbasic. For Macintosh classic (OS 9) users, we have programs written using RealSoft's REALbasic 3.5.1. For Mac OS X users we have similar programs written with REALbasic 5.5.2. The sample screens below are from the OS 9 version, but the OS X versions are similar.

Setting Up the Files:

Using the LabPro with REALbasic in OS X

On the CD that came with your DCU you will find a folder Mac OS X REALbasic DCU". Copy this folder to your hard disk in a convenient location.

To use the LabPro with REALbasic on Mac OS X, you must first have two support files in the proper locations.

- **VSTLabProUSBRB** is a plugin. You will need to put a copy of this plugin in your REALbasic plugins folder.
- The package **VST_USB.bundle** is required to use Labro via a USB connection. It should be placed in the same folder as the program you are using. We have placed this bundle in the folder with all the DCU programs on the CD. Just makes sure that when you copy REALbasic programs to other folders, to keep this bundle with them, or the programs will not work.

Using the LabPro with REALbasic in Mac Classic (OS 9)

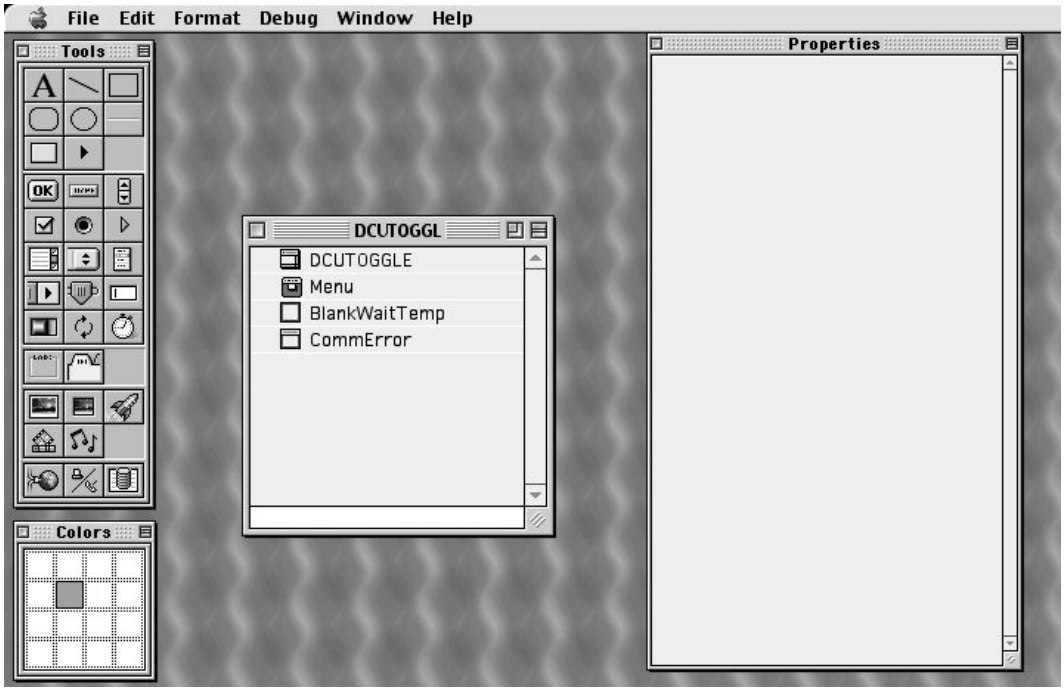
On the CD that came with your DCU you will a folder Mac OS9 REALbasic DCU". Copy this folder to your hard disk in a convenient location

- If you plan to use a USB connection, make sure the extension LabProUSB is installed on your computer. This extension comes with the Vernier Logger *Pro* 2 or Logger *Pro* 3 program. If you do not have this program installed on your computer, you can download a free demonstration version from.www.vernier.com/drivers. This file should be placed in the System Folder/Extensions folder.

Trying out a REALbasic Program - DCUTOGGLE

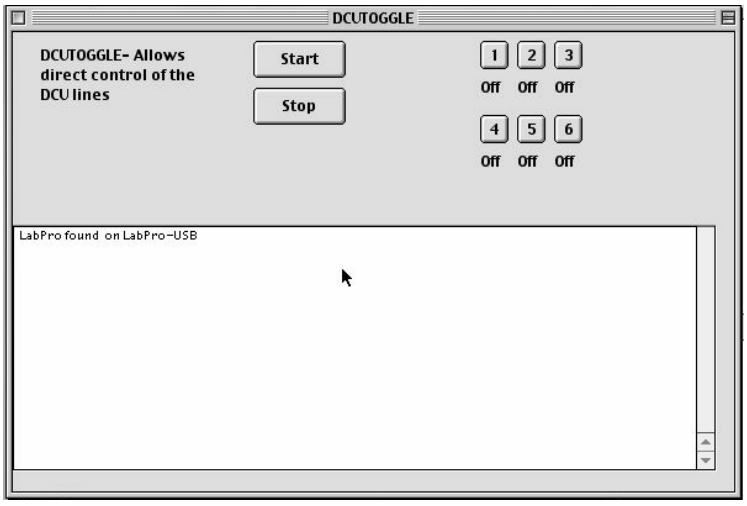
To see a REALbasic program in action controlling the DCU, try the DCUTOGGL program. This is a program to control the DCU lines with mouse clicks. Connect the DCU to the connector on the top right side of the LabPro labeled DIG/Sonic 1. Make sure this connector locks in place. Connect a LabPro power supply (IPS) to the round connector on the DCU. Connect LabPro to the Macintosh computer using the USB or serial port cable.

Navigate to the location on your hard drive where you put the DCU REALbasic folder and open it. Inside you will see a number files. Double-click on this on file called DCUTOGGL. This will start REALbasic with the DCU program open. On the right side of the screen you will see a window labeled Properties. On the left side of the screen you will see a window of Tools and another window labeled Colors. For now, we will ignore these windows. In the center of the screen you should see a window labeled DCUTOGGL. This is the important window for now.



Initial REALbasic Screen.

Let's try running this program. Select the Run command from the Debug menu at the top of the screen. This will bring up another window. You should see the user interface for the DCUTOGGL program, which includes Start, and Stop buttons and buttons for controlling the six output lines of the DCU. The words "LabPro Found" should appear in the ListBox at the bottom of the screen if all your hardware is connected properly. You will get an error message if there is a problem with the hardware. If this happens check the power to the LabPro and the connection to the computer. If that does not help, try removing the power from the LabPro and then plugging it back in.



DCUTOGGL Program User Interface

The six buttons on the screen correspond to the DCU lines 1-6. This program will allow you to turn these lines on and off. Try clicking the mouse on the "1" button on the screen. The red LED labeled 1 should go on. Press the 1 button again and it should go off. Try to turn the other LEDs on and off using the 2 through 6 buttons on the screen. Note that there are some combinations of LEDs that are not allowed, so that in some cases when you turn on an LED, some others may go off. The DCUTOGGL program is very useful in testing hardware for your future projects that you connect to the DCU.

Note that the Start button does nothing in this program. It is there to maintain consistency with our other programs. When you are finished experimenting with the DCUTOGGL program, click on the Stop button to terminate communication with the LabPro and then quit the program. REALbasic will still be running.

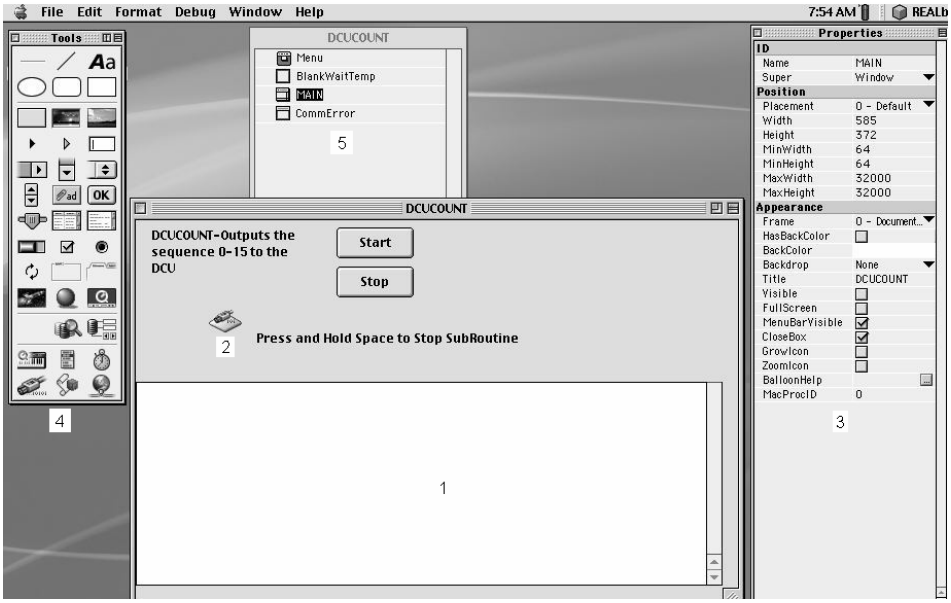
Opening the DCUCOUNT Program

Choose Open from the REALbasic File menu and navigate to the DCU sample programs folder. Within that folder is a file DCUCOUNT. Open this file. Answer "no" to the question about saving changes to the last program. When the file opens, near the center of the screen you will see a Window (called the Project Window) that lists the various other windows that make up the program. It has an entry "MAIN". Double-click on this entry.

An Overview of REALbasic

REALbasic uses a lot of windows to display information. The program is built around a main window to which you add buttons, labels, text boxes, list boxes, and controls. Many of the objects on this window have REALbasic code (instructions for what the computer should do) associated with them. For example, pressing a button will result in the code attached to the button action being executed. This is why REALbasic is referred to as an *Event Driven* language. Pressing a button is considered an *Event*.

The Windows used by REALbasic:



View of the Main DCU Window (Mac Classic Version)

Key parts of a REALbasic program are labeled with numbers on the figure above. We will try to explain what each of these numbered windows and objects does below:

1. This is the main window of the program with the user interface. It is technically referred to as the Window Editor. It controls what the application that we have developed looks like. Notice that there are several different objects on this window, including two buttons, several labels and a large white listbox at the bottom. This window is how the program gathers input from the user and also how it displays output to the user as well.

2. **(Mac Classic Only)** One object displayed on the screen displayed above is only used on the Mac Classic version of these programs. It looks like a Mac modem/printer symbol. This is the Serial Communications Control. This control allows us to communicate with the LabPro via the Serial Port. It is important to the program, but you should not have to change anything about it in your projects. It does not appear in the program window when you run the application. It is only visible in the Design Environment view. The serial communications control has a name: LabPro. It has

properties associated with it and clicking on it will display those properties in the Properties window to the right.

3. This is the Properties window. It will become more important as you go on with our development of applications. Many objects in REALbasic have properties associated with them, such as the name, caption, location, etc. This window is where these properties are set and changed.
4. This is the Tools Window. You can drop items from this window onto the main window to add features like buttons, labels, text windows, list boxes and controls. We will not be using these controls now, but as you program your own applications they will become invaluable.
5. This is the Project window of the application. It shows all of the other windows that are associated with this program. In this program there are a few other windows, plus a default Menu window.

Running the DCUCOUNT Program

The DCUCOUNT is a program that we created to count through the 16 possible DCU outputs and then turn all the lines off. Each of the numbers 0–15 represents one of the 16 patterns to the DCU. So a program that goes through each of the 16 outputs on the DCU would be a program that sends the numbers 0–15 to the LabPro for use by the DCU. It sends out a zero at the end to make sure all the lines are off.

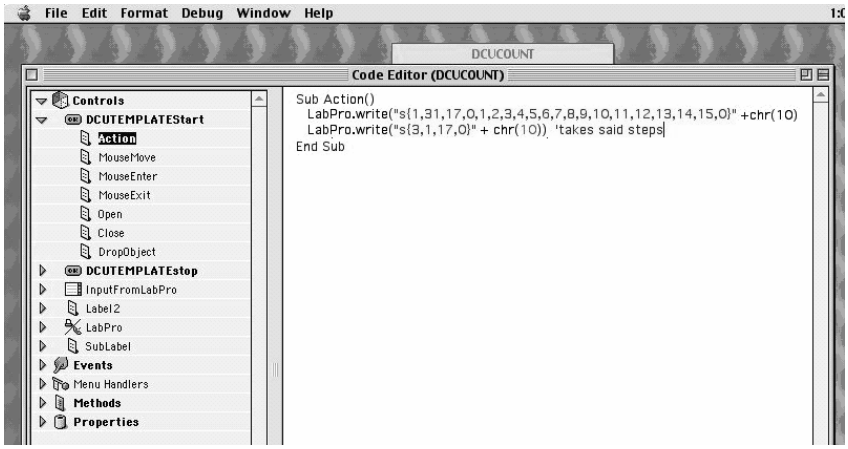
Let's try running this program. Select the Run command from the Debug menu at the top of the screen. This changes the appearance of the screen. You are now seeing the *Runtime Environment* of REALbasic. This is the actual running application with the DCUCOUNT window and its user interface displayed. You should see the words "LabPro Found" written in the ListBox at the bottom of the screen. The green LED of the DCU should be on. There are no red DCU lights on and the program seems to be waiting for some sort of input from the user.

Try clicking on the Start button. This button starts sending the output sequence to the LabPro. It sends each of the possible outputs sequentially, holding each output for 1 second. Wait for all 17 outputs to cycle through and press the Start button again. The sequence should start over again. You can continue doing this as many times as you like. When you are done, press the Stop button. This button will cleanly discontinue communication with the LabPro and terminate the execution of the program. It will return you to the design view that we saw earlier, the *Design Environment* view of REALbasic.

Examining the REALbasic Code

Since the act of you clicking on the Start button started the sequence of LED flashes, there must be some code associated with that button that sends out the correct sequence to the DCU. Let's take a look at this code.

With the DCUCOUNT user interface window open on the screen, but not running, double-click on the Start button. This will bring up a new screen that we haven't seen before. This is the Code Editor where the code is displayed and edited. By double-clicking on the Start button, the code view has opened with the cursor placed in the section of code that is associated with the clicking of the Start button.



View of the code that is associated with the Start button

Let's look at exactly what we have here. On the left side of this window you should see a hierarchical listing of entries. The item at the top of this list is Controls. This item is opened in the hierarchical view and we see several sub items in this list, including DCUTEMPLATEStart and DCUTEMPLATEStop. These two entries correspond to the Start and Stop button that you see on the window. Now notice that the DCUTEMPLATEStart entry is also expanded and we see several *Events* below it. The one that is selected is labeled "Action". Events contain the code which executes when the action takes place. In this case the code executes when the user clicks on the Start button.

The syntax of the information that LabPro expects to receive is

"s{command}" + chr(10))

The command is just the string of integers we plan to send to the LabPro. The s, the quotation marks,and the curly brackets are required; they make sure that the data you want to send out is in a window readable by the LabPro. The +chr(10)) at the end of the line simply adds a termination character to the information being sent out so that the LabPro will know that it has received an entire command.

We said earlier that we wanted to send out all the possible output codes to the DCU. How is that accomplished? Earlier sections of this manual explained the commands to control the DCU in this way. We first set up the sequence of steps that we want to send to the DCU with a 1 command. Then we issue a 3 command for the LabPro to actually start through the sequence of outputs. In this case we are using the following command to set up the sequence:

{1,31,17,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0}

Here the 1 tells the LabPro we are setting up a channel, the 31 tells the LabPro that we want to setup the DIG/Sonic channel, the 17 tells the LabPro that there are 17 elements in the sequence. The numbers that follow are the actual outputs we would like to send.

The 1 command sets up the pattern. The output is started with a 3 command. The syntax we use for this 3 command is:

{3,1,17,0} {3, number of seconds between steps, number of steps, triggering}

So, now we need to see how to send these commands with REALbasic. We want to send these commands anytime the Start button is pressed, so we put it in the code associated with the Start button. Any code that we put here will be executed when this Start button is clicked. Outputting information to the LabPro consists of only one command.

LabPro.Write (information to be sent out of serial or USB port)

LabPro is the name of the Serial Communications Control which handles addressing the port to which LabPro is connected. The "Write" part of the command tells the computer that you want to output information to the device

connected to the serial. The information in the parentheses is simply what you want to output. Even though this code was originally set up for use with a serial port connection to a LabPro, it also works with USB connections. On Mac OS 9, if you have the proper USB extension on your computer, REALbasic sends the data via USB connection to LabPro. On Mac OS X, if you have the right files in the right places, we have written some special code which allows the same LabPro Write command to send data to the LabPro via USB connection.

So the code we activate when we click the Start button is:

```
LabPro.Write("s{1,31,17,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0}" + chr(10))
LabPro.Write("s{3,1,17,0}" + chr(10))
```

Modifying the Program

It's now time to customize the DCUCOUNT program and make it do something different. We are going to modify the program to make it go through the counting more slowly, but also make two other changes. We will modify the program to make it turn on DCU lines 1 and 2 for 3 seconds before it starts the counting sequence. Also, we will buzz a speaker at 50 Hz for 5 seconds before the pattern starts. The speaker hardware is not really required. If you don't have a speaker that you can connect to the DCU, you will be able to visually check and make sure the program is doing what it should.

Let's go through the changes one at a time.

Changing the rate of going through the pattern:

This is the easy change to make to the code. The second number in the 3 command sets the time between steps of an output sequence. Change the line so it reads:

```
LabPro.Write("s{3,2,17,0}" + chr(10))
```

Turn on DCU lines 1 and 2 for 3 seconds before it start the counting sequence:

Now, to turn on the DCU lines, we often use a 2001 command. The format is

2001, [output pattern]

To turn on DCU lines 1 and 2, we want to send out a 3 for the pattern, so we want to add the line:

```
LabPro.Write("s{2001,3}" + chr(10))
```

To get the program to hold here for 3 seconds, we need a REALbasic Sleep statement:

```
Sleep (3000) 'wait 3 seconds
```

This will have the program wait at this point for 3 seconds. Note that the REALbasic Sleep commands use milliseconds for the time.

Buzz a speaker at 50 Hz for 5 seconds

For the final change we need to make the speaker buzz. Code we have included in the REALbasic methods will make this pretty easy. All the sample REALbasic programs on the CD include a number of subroutines, stored in *methods* for you to use. One of them is named DCUBUZZ, and it does exactly what we would like here. If you look down this list at the left side of the window you shall see a heading labeled Methods. Click on this entry to expand it. Once it expands, you shall see many entries in the left side of the window. This is a list of the methods that we have written to aid you in developing your applications.

We can add this method to our code. The DCUBUZZ method requires two parameters, the frequency of the buzz in Hz and the duration of the buzz in seconds. Here is what the DCU Start button code will look like after adding in the appropriate DCUBUZZ command and the other changes have been made.

```
Sub Action()
    LabPro.Write("s{2001,3}" + chr(10)) ' turn on DCU lines 1 and 2
    Sleep 3000 'wait 3 seconds, REALbasic Sleep commands use milliseconds for time
    DCUBUZZ 50, 5
```

```
        LabPro.Write("s{1,31,17,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0}" + chr(10))
        LabPro.Write("s{3,2,17,0}" + chr(10))
    End Sub
```

Note that in the DCUBUZZ line, the 50 is the frequency that we specified before and the 5 is the duration.

Now make all these changes to the program and try running it again (select Run from the Debug menu). If you have a speaker handy, connect it between the D1 and Ground lines. When you click on the Start button, you should see the 1st DCU channel flicker for 5 seconds and if you have a speaker, hear a buzz from it. Then the LEDs on the DCU should cycle through the 17 outputs, more slowly than before.

Exit REALbasic, or open another DCU sample program if you like. You may or may not want to save the changes you made. You also can save your changed version of the program with a different name.

This is an example of how you can use the raw material we provide in the window of REALbasic code to create your own DCU control programs. REALbasic is an extremely powerful programming tool, but like most programming languages, you must become familiar with the commands and syntax of the system. For more details about LabPro programming, refer to the *DCU Technical Reference Manual*. For more information about programming in REALbasic, read the manuals that came with REALbasic. The best advice we have is to experiment.

Additional Notes on REALbasic Programming

This section offers specific tips for people writing REALbasic programs. We hope to warn you about some of the things that can cause confusion when you start DCU programming.

REALbasic Windows

When using REALbasic, you may find it confusing dealing with the various parts of a sample program. Here are some tips. To see the user interface of the program, select Project from the REALbasic Window menu and then double-click on the window you want. It is often named "Main" or sometimes has the name of the file (for example DCUCOUNT). To see the code associated with one of the buttons, double-click on that button. Once you have displayed some of the code, you can also choose "Code Editor" or "Source Code" from the Windows menu to get back to the code.

Sometimes in REALbasic, windows get hidden behind one another strangely. Try moving things around if you lose a window.

REALbasic Methods

We have provided you with subroutines, called *Methods* in the REALbasic that you can easily use. These methods are included with each of the sample programs. So, for example, you can attach a method that monitors a photogate and reports the status. By doing this you can fairly easily set a program up so that when the user clicks on a button, the program will begin monitoring a photogate. This is much simpler than trying to write your own code to monitor the photogate.

To examine or modify these methods, select Code Editor or Source Code from the Window menu of REALbasic or double-click on one of the button in the user interface window. You will see a hierarchical menu at the left side of the window. Open the Methods item and you will see each of the subroutines of the program listed as a separate method. Click on any method and the code for that method is displayed to the right.

Methods are used by listing the method name in the code with any parameters to be passed to the method following the name. For example with the DCUPWRON subroutine, you call to it with a parameter set for the digital output pattern you want on and T set to the time you want the pattern to stay on. The code below turns on the D1 line for 2 seconds.

```
    DCUPWRON,1,2
```

Major operations that you want your program to handle can be replaced using one of the subroutines. Many

programs can be written by linking together these subroutines with a few lines of new code. As an example, consider the program DCUTRAP2. This program is used to make a bug trap. It uses a photogate to sense when a bug is inside a box. It then turns on a motor to close the lid of the box. When the program opens the Method DCUINIT is automatically called. The rest of the program is three buttons for the operations and each one has code assigned to its Action.

For the Check button, the Action is: DCUCHKD	(this is a method to check the photogate status)
For the Trap button, the Action is:	
DCUWAITD	(this is a method to check the photogate status and wait until the photogate is blocked)
DCUPWRON,1,1	(this is a method to turn the power on (to D1 for 1 second)
DCUPULSK 2	(this is a method pulse the power on and off (to D2))
For the Stop button, the Action is: DCUOFF	(this is a method to shut off all lines)

All the complicated code for this program is in the methods we have provided.

How commands are sent to LabPro for controlling the DCU

LabPro is controlled by sending a string of integers, called a *command*, out through the serial or USB port. In REALbasic, we use the following code, which sends data to LabPro:

```
LabPro.Write("s{integer string}" + chr(10))
```

The command is in the curly brackets. The rest of the code tells the computer where to send the data (LabPro.Write), and sends a carriage return at the end of the command ends (chr(10)). This format works for all the versions of REALbasic and for both serial and USB connections if you have the driver files loaded on your computer as suggested at the start of the REALbasic section of this manual. If you wanted to have the LabPro turn on line 1 using the 2001 command, you would use the following line in REALbasic:

```
LabPro.Write("s{2001,1}" + chr(10))
```

Interrupting the Program

When writing the subroutines such as DCUCHKD, and DCUWAITD for REALbasic, we were not able to find a way to provide a functional Stop button. Instead we designed these methods to stop on the press of the Space Bar. There is code to check for a press of the Space Bar and break out of the While loops if it is pressed. The relevant code is:

```
SubRoutineStopButtonClicked = false
While SubRoutineStopButtonClicked = false '[start loop]

   `[main code of loop]

    if keyboard.asynckeyDown(&h31) then SubRoutineStopButtonClicked =
true
wend
```

This tests the keyboard during the loop to see if the Space Bar (key label &h31) has been depressed. If it has been, the variable `SubRoutineStopButtonClicked` is set to true and the While loop end. This same idea needs to be used in programs similar to DCUWARNV, which include loops. If you do not handle this properly, you can get into a situation were the REALbasic program is running and there is no way to stop it.

Getting Data Back from LabPro

In REALbasic programs that ask LabPro to return data or other information, there are two ways to handle the data.

Normally, whenever data are available on the serial or USB port of the computer, the Data Available subroutine (listed under the serial control (LabPro) would automatically takes over and processes the data. We have written programs to operate that way and it works fine. In the sample programs we provide on the CD, we have chosen to handle things differently. We do not allow the Data Available subroutine to take control and instead use code like this in the main part of the program.

```
Dim Buffer as String 'A string that will be used for reading values from
LabPro
Dim CurrentValue as Double
labpro.flush
labpro.write ("s{3,.01,1,0}" + chr(10))
labpro.write ("g" + chr(10))
Buffer = LabPro.ReadAll         `Reads the incoming data
labpro.flush 'clears the buffer
lblcurrent.text = mid(labpro.lookahead,5,1) + mid(labpro.lookahead,7,1) +
"." + mid(labpro.lookahead,8,1)
```

Data Format of LabPro Data

LabPro sends data back in a format similar to this:

```
|{ +n.nnnnnE+nn}|
```

In order to get this data translated to floating point numbers (value) we us the following lines of REALbasic:

```
Mantissa = Val(Mid(Buffer, 1, 8))
Exponent = Val(Mid(Buffer, 10, 3))
```

Current value = Mantissa * Pow(10,exponent)

New versions of REALbasic can handle this string to number conversion more easily. With them you can simply use:

```
Buffer = Trim(Mid(Buffer, 4, 13))
Current value = Val(Buffer)
```

We chose to use the more complex mantissa, exponent version in both the OS 9 and OS X versions for consistency.

Collecting Data as you Control Outputs

The DCUTEMPC program is a good example of this kind of programming. It uses a -1 command in the 3 command and monitors a temperature as this loop continues. If the temperature hits limits, the program branches to other code.

Making Sure LabPro Finishes before the REALbasic Program Moves On

One thing that can cause confusion when programming the DCU occurs when the computer goes on with its program before the LabPro has time to finish what it is doing. For example, you may send commands to the LabPro to have the DCU set an output pattern before an earlier series of operations are completed. Consider this program, for example:

```
LabPro.Write("s{1,31,2,0,7}" + chr(10))     Set up the digital output to
                                            turn the first 3 lines on and
                                            off
LabPro.Write("s{3,1,10,0}" + chr(10))       Go through 10 steps, each
                                            taking 1 second
LabPro.Write("s{2001,0}" + chr(10))         Set up digital output lines
                                            for turning power off to all
                                            lines
```

If you try this program, it will not operate the way you might expect. If the DCU is connected, you might expect to see the first three red LEDs flash on and off for ten seconds and then the power should go off. Instead, the LEDs will briefly flash, but the program quickly ends with the power turned off. Why? The problem is that LabPro starts executing the sequence of 10 steps, but then is sent a new command telling it to turn the power off. It interrupts what it was working on and follows the new instruction.

There is a solution to this problem. Use the Sleep subroutine (method) which is included in all of our sample programs to keep program from moving on, while you wait for LabPro to finish a command. The format of the command is Sleep(X), where X is the number of milliseconds to delay. For example, in the sample program above, you could use

```
Sleep (10000)
```

This will cause the program to pause for 10 seconds. We use this approach frequently in our sample programs.

Another good reason for using subprograms in your DCU programming is to avoid confusion of this sort. If you need to turn on a line for a specific time, use the subprogram DCUPWRON. Simply specify the output you want (D), and the time you want the pattern held (T), and all this will be taken care of for you.

Power Control and DCU_INIT

Whenever you use LabPro for Digital Output, you probably want to set the LabPro to leave power on at all times. You do this with the following command

```
LabPro.Write("s{102,-1}" + chr(10))
```

We do this in the DCUINIT method, and the best way to start a DCU program is to simply use a call to the DCUINIT method as described below.

The DCUINIT method does several things. First it opens the serial or USB port. It then checks to see if LabPro is properly attached to the port and powered up. This is done by sending a {7} command to the LabPro (a system status request). If properly connected and powered, LabPro responds by sending back a long string of characters. We have the computer sleep for 200 milliseconds and then examine the data returned. If we verify that a LabPro is properly connected, we indicate so in the list box. If the proper string of data is not returned, a Dialog box is displayed to inform the user of an error in communication. This dialog allows the user to retry the connection until it is successful. Finally, the DCU_INIT routine sets the LabPro power to be always on.

REALbasic Data-Collection Programs for LabPro

We have included some sample programs to demonstrate programming for collecting data using just sensors with LabPro (no DCU) on the CD. There is a simple real-time (live) data collection program, and non-real-time data collection program. There are versions of these data collection programs in both the Mac Classic and Mac OS X folders.

REALbasic Developer Articles on using LabPro

The July/August 2004 issue of REALbasic developer magazine featured a cover story on the use of LabPro in your REALbasic programming. The following September/October issue continued the discussion on LabPro programming. These articles were written by Willam H. Murray and Chris H. Pappas. They offer lots of sample programs and suggestions for writing REALbasic programs for LabPro. You can order back issues of this journal at www.rbdeveloper.com.

Using the DCU with LabVIEW and a Windows, Macintosh, or Linux Computer

Introduction

This reference guide is to help people who want to use LabVIEW to get their DCU up and running as quickly as possible. It assumes that the user has some familiarity with LabVIEW programming. The first section covers information on how to run, examine and modify one of the sample VIs. The next section, *Additional Notes on LabVIEW Programming*, provides information about where to look for good examples, how to find and use the various Express and subVIs, and troubleshooting help.

This guide is written with the assumption that you are using LabVIEW 7. This means that the Express VIs are used in examples and frequently discussed. If you are a LabVIEW 6 user you do not have access to Express VIs, but the information is just as valid. Our LabVIEW 6 VIs have replaced the Express VIs with subVIs that have the same name, the same icon, and the same functionality. The programs look and perform the same. The only difference, of course, is that you will not see the big blue field that surrounds the icon, and you must configure a subVI by wiring in values, not with the use of a popup dialog box.

Separate versions of DCU sample programs for each of these computer operating systems are provided on the DCU program CD:

LabVIEW 7.0 or 7.1

- Windows computer with either serial or USB connection to LabPro
- Macintosh OS X computer with USB connection to LabPro
- Linux computer with serial connection to LabPro

LabVIEW 6.1

- Windows computer with either serial or USB connection to LabPro
- Macintosh OS 9 computer with serial or USB connection to LabPro
- Linux computer with serial connection to LabPro

Setting Up the Files

On the CD that came with your DCU there is a folder that combines the name of the operating system (Windows, Linux, Mac OS 9, or Mac OS X) and the version of LabVIEW (LabVIEW 6.1 or LabVEW 7.0). Copy the appropriate folder to your hard disk and follow the directions below:

1. The folder that you moved to your hard disk contains two subfolders named "LabPro" and "DCU Programs". Move the folder named LabPro to the user.lib folder that is found in the National Instruments/LabVIEW directory. The LabPro folder contains the subVIs that are used in all of the DCU VIs we provide. None of the VIs we provide will work, unless you have this folder.
2. Move the folder named DCU Programs to a convenient location on your hard drive.
3. Read the notes below to confirm that you have the appropriate USB or Serial drivers installed.

If you will be using a USB connection from the LabPro you may need to install a USB driver:

- Windows: You need to install USB drivers, unless you have previously installed the Vernier program Logger *Pro* 3.3. If you do not have Logger *Pro* 3.3 already installed on your computer, then run the LabPro USB Installer. You can find this installer at www.vernier.com/drivers.
- Mac OS X: You must have the USB driver installed. We do not have an installer for the Mac OSX driver. Therefore, you must have Logger *Pro* 3.3 installed. Installing a demo version of Logger *Pro* will also install this driver. You can find a demo at www.vernier.com/downloads.

- Mac Classic, OS 9: Make sure the extension LabProUSB is installed on your computer. This extension comes with the Vernier Logger *Pro* 2 or Logger *Pro* 3 program. You can find this installer at www.vernier.com/drivers.

If you will be using a Serial connection from the LabPro, and running LabVIEW 7, you may need to install the NI-VISA driver software. This driver is part of the LabVIEW installation CDs from National Instruments. If you did not install this driver when you installed LabVIEW, do it now. It can also be downloaded from the NI website (www.ni.com).

An Overview of LabVIEW

LabVIEW is a graphic programming language. LabVIEW programs are known as virtual instruments, or VIs. A LabVIEW VI has two main parts: the Front Panel and the Block Diagram. The Front Panel is the user interface with buttons and controls and also displays. The Block Diagram is the code controlling how the VI functions. To toggle between the Front Panel and Block Diagram displays on screen, press the Control-E keys (Windows), Alt-E keys (Linux) or Open Apple-E keys (Macintosh).

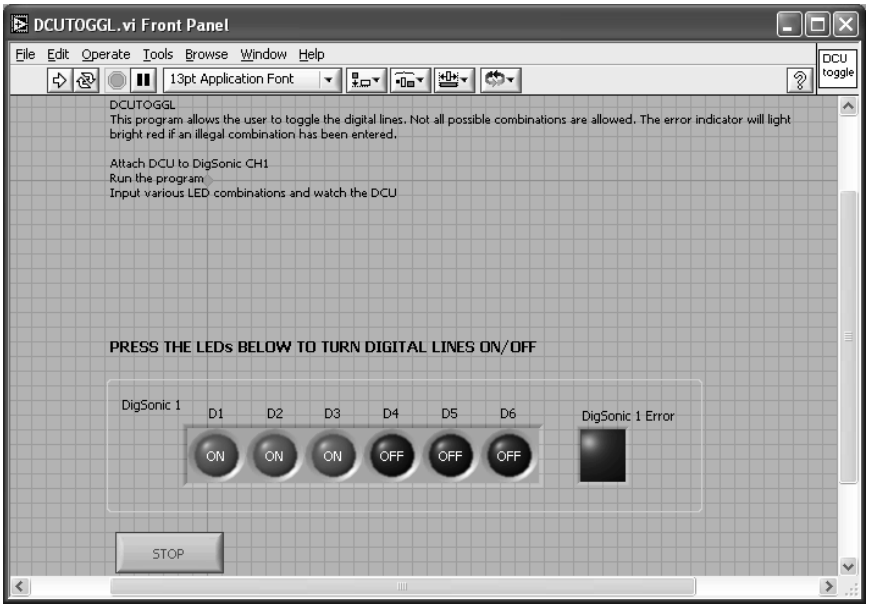
We have provided a collection of VIs and Express VIs that perform the low-level commands to the LabPro. These VIs will be the building blocks of our DCU programs and for your own custom programs. Study the Block Diagrams of the sample VIs and the information within this section to help you understand how to use these LabPro VIs.

Trying out a LabVIEW Program—DCUTOGGLE

To see a LabVIEW program in action, controlling the DCU, try the DCUTOGGL VI. This VI allows you to control the DCU output lines with button presses. First, prepare your hardware:

1. Connect the DCU to the connector on the top right side of the LabPro labeled DIG/Sonic 1. Make sure this connector locks in place.
2. Connect a LabPro power supply (IPS) to the round connector on the DCU.
3. Connect a LabPro to the computer using the USB or serial port cable.

Now, navigate to the location on your hard drive where you put the DCU Programs folder and open the folder. Inside you will see a number of files. Double-click on the file named DCUTOGGL.vi. This will start LabVIEW with the DCU program open. You will see the front panel of the LabVIEW program.



Front Panel of DCUTOGGL.vi

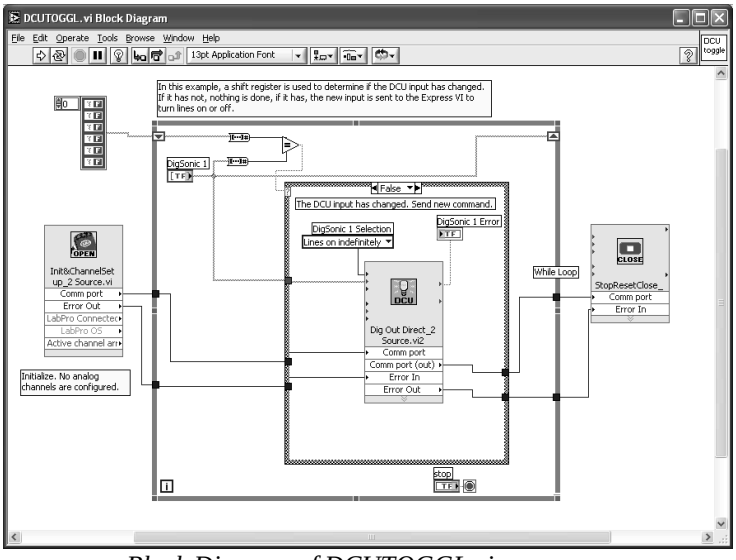
Let's try running this VI. Click on the Run button (white arrow at the left side of the toolbar). This will start the VI running, and you should see the green LED of the DCU turn on. Problems that might occur are usually solved with the following steps:

1. Check your connections.
2. Check the power to the LabPro.
3. Remove the power from the LabPro and then plug it back in.
4. If you are using a USB connection, look over the "Setting Up the Files" section above.
5. Look over the "LabVIEW Troubleshooting Help" section of this manual

The six buttons on the screen correspond to the DCU lines 1–6. This program will allow you to turn these lines on and off. Try clicking the mouse on the "D1" button on the screen. Look at your DCU, the red LED labeled 1 should go on. If you have any electrical device connected to D1, it should be turned on. Press the D1 button again and it should go off. Try to turn the other LEDs on and off using the D2 through D6 buttons on the screen. Note that there are some combinations of LEDs that are not allowed (see the table of digital outputs in the "DCU Overview" section). If you make a click that would set up an output pattern that is not permitted, the red error warning button on the right of the screen will go on, and the output pattern will not change. The DCUTOGGL VI is very useful in testing hardware for your future projects that you connect to the DCU.

When you are finished experimenting with the DCUTOGGL program, click on the Stop button at the bottom of the screen to terminate communication with the LabPro and end the program. It is not recommended that you stop the program by clicking on the red Stop icon on the toolbar. This button should be reserved for emergency stops only, because it will abruptly stop the program and could leave the LabPro or USB port in an unusual state.

If you want to examine the Block Diagram of this VI, choose Show Block Diagram from the Windows menu. This VI uses three Express VIs to communicate with LabPro. We will discuss these VIs later in this manual.



Block Diagram of DCUTOGGL.vi

Opening the DCUCOUNT Program

Choose Open from the LabVIEW File menu and open the DCUCOUNT.vi. If asked, answer no to the question about saving changes to the last program. We will take a look at this simple VI as a way of learning how LabPro is controlled by LabVIEW.

Running the DCUCOUNT Program

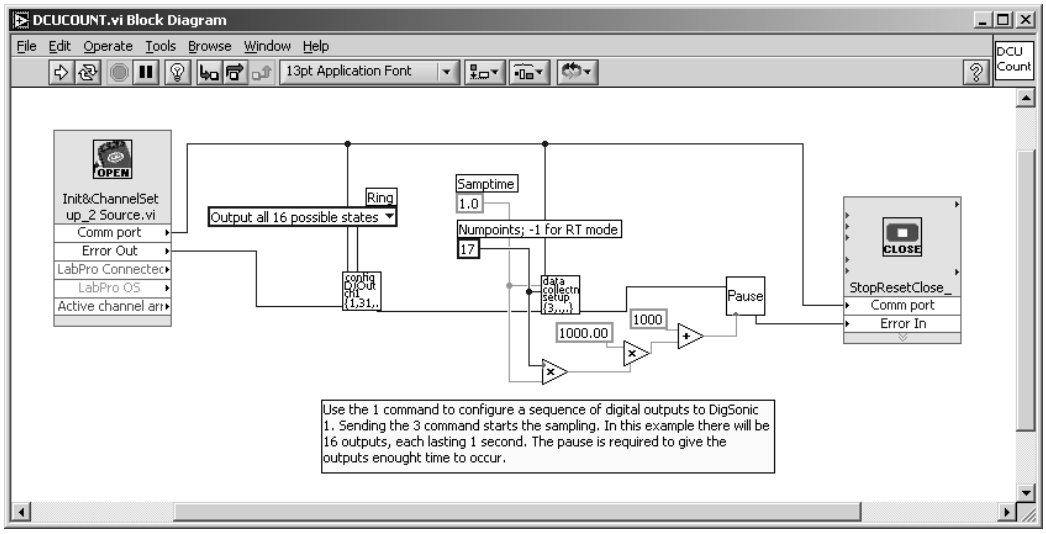
DCUCOUNT.vi is a program that we created to count through the 16 possible DCU outputs. Each of the numbers 0–15 represents one of the 16 outputs to the DCU. Therefore, a program that goes through each of the 16 outputs on

the DCU would be a program that sends the numbers 0-15 to the LabPro for use by the DCU. It sends out a zero at the end to make sure all the lines are off, for a total of 17 steps in the pattern.

Let's try running this program. Make sure your DCU is connected to DIG/Sonic 1 and then click on the Run button. This will start the VI running. The program begins executing the code on the Block Diagram that sends the output sequence to the LabPro. It sends each of the 17 outputs sequentially, holding each output for 1 second. Wait for all 17 outputs to cycle through and press the Run button again. The sequence should start over again.

Examining the LabVIEW Code

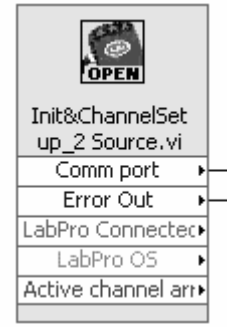
Clicking on the Run button began the code that told the LabPro to send a sequence of LED flashes. So now let's take a look at this code, found in the Block Diagram (remember, an easy way to toggle between the Front Panel and Block Diagram displays on screen is to press the Control-E keys (Windows), Alt-E keys (Linux) or Open Apple-E keys (Macintosh).



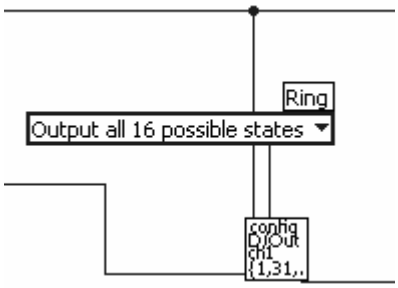
Block Diagram of DCUCOUNT.vi

You will see that this program is made up of two Express VIs, three subVIs and some Numeric functions. The execution order of this program is left to right, but this is not controlled by the position of the VIs on the block diagram. LabVIEW uses *data flow* programming, meaning that a sub or Express VI will not execute until all of its input data are available. Notice that the blue data wire, originating from the Error Out node of the Init&ChannelSetup Express VI, is connected through all of the sub and Express VIs. This forces the program to flow from left to right.

All of the VIs in this example are provided in the LabPro folder from the CD (placed by you in the user.lib folder of LabVIEW). On the next page is an explanation of what each of the five main parts of DCUCOUNT does:



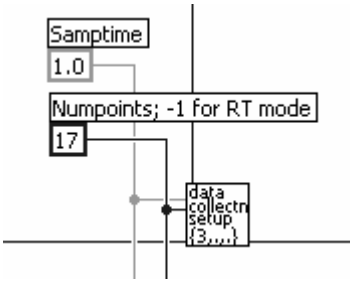
The Express VI, Init&ChannelSetup_2, begins the program. Express VIs are a new feature of LabVIEW 7. They contain, and hide, a lot of low-level code. Express VIs can be configured with a dialog box that appears when you double click the Express VI (double click on it to see the configuration dialog box). Alternatively, they can be configured by wiring data into their input nodes during program execution (just like a subVI). This Express VI tests for a LabPro connected to the computer, notes how it is connected (the comm. Port), initializes the LabPro, and configures the selected channels. If it does not find a LabPro connected to the computer, an error dialog box will appear.



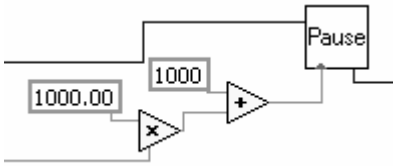
When you want to send a pattern of outputs to the DCU (as compared to a single output), you set up the pattern using a 1 command. The subVI, Config_Digital Output CH1.vi, allows you to easily do this by choosing from a list of possible output patterns. The inputs used here are Comm port, Ring, and Error In. There are also inputs for the "Operation" and "List of Values" if you choose to create a custom sequence. The Error In line is connected to the Error Out line of the previous VI. This pattern of connecting the Error Out of one object to the Error In of the next object is a standard practice used in all the VIs in this manual. Ring contains a list of the six most popular output patterns you might want to pick:

- Custom; input operation and list values
- Flash 3 LEDs
- Output all 16 possible states
- Run Stepper Motor CCW
- Run Stepper Motor CW
- Flash each of the 6 of the LEDs in order

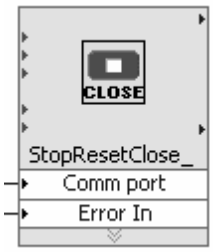
In this program we chose the "Output all 16 possible states" pattern. In other projects you will primarily be choosing to create a custom output pattern. For example, you may want lines 1 and 2 to alternate on and off. To perform a custom pattern you will be choosing the first option from the Ring list called "Custom; input operation and list values", and inputting values into the Operation and List of Values nodes. See the section in *Additional Notes on LabVIEW programming* for more information on custom sequences.



The 1 command was used to set up the digital output pattern. LabPro must now be told how quickly the pattern should be executed (time between steps), how many patterns to output (number of steps) and when to start the output pattern. The subVI, Config_DataCollectionSetup.vi, sends a 3 command to the LabPro, starting the output pattern. The Samptime input determines the time between steps and the Numpoints input determines the number of steps. There are also inputs for Error in and Comm port.



The next part of the Block Diagram is used to provide a delay in the program to allow the sequence of outputs to execute on the LabPro. Otherwise, we might start an output sequence and then end the program before the sequence has had time to happen. How long should this delay be? We can calculate it, since the time between steps and the number of steps are both known. The subvi, pause.vi, expects an input in milliseconds. We can take the time between steps, and multiply it by the number of steps to get the time delay needed in seconds. To convert the delay to milliseconds we should multiply by 1000. In this VI, we do this and then we add one second (1000 milliseconds) to the delay just to make sure there is time for the pattern to finish executing.

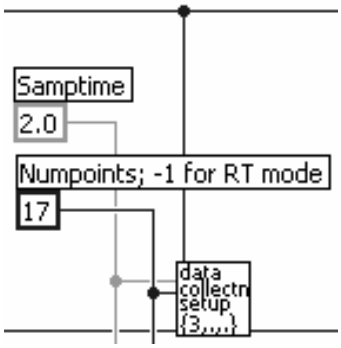


Once the pause time elapses, the last piece of code becomes active. StopResetClose_2 is an Express VI whose function is to properly end the program and shut down the LabPro, including freeing up the computer's USB or comm port. It is very important to end a program by properly closing the computer's USB or comm Port.

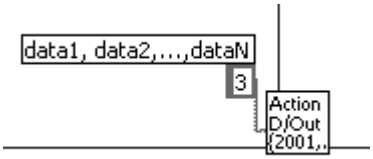
Modifying the Program

It's now time to customize DCUCOUNT and make it do something different. We are going to make two changes. We will modify the program to make it turn on DCU lines 1 and 2 for 3 seconds before the counting sequence starts and also modify the program to make it go through the counting more slowly.

The easy change to make to the code is to change the rate of going through the sequence. In the part of the code below, all we have to do is to change the Samptime input. Let's change the time between steps from 1 second to 2 seconds.

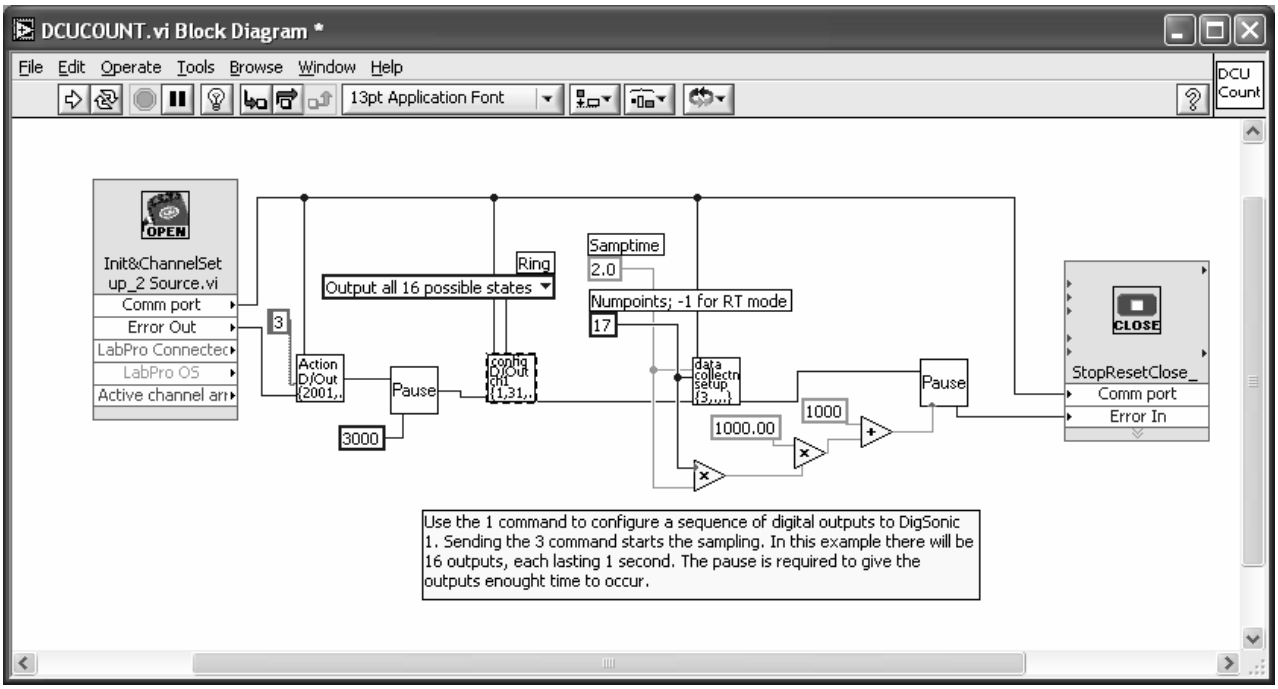


Now, let's enter the code that will turn on lines 1 and 2 before the sequence starts.



The LabPro folder has a collection of subVIs and Express VIs for many different tasks. The one we will use, pictured above, is named Action_Digital Out.vi. This simple subVI sends a 2001 command to the LabPro, which tells the LabPro to perform a direct digital output (for more information on the 2001 command refer to the *Programming for the DCU* section of this manual). The inputs nodes into the VI are labeled: "data1, data2,…,data N", "Comm port", and "Error In". To set DCU output lines 1 and 2 both on, we need to send a data value of 3 to the LabPro. To do this we create a constant for the subVI's "data1, data2,…,data N" node and wire in a value of "3". Note that this node expects the input to be a string, not a numeric - so make sure your constant is a string constant. The Action_Digital Out subVI turns on the two lines, now you need to add a delay so that the two lines will stay on for 3 seconds before the sequence starts. One way to perform this delay is with the use of the pause.vi, which was discussed above. The input to this subVI is delay time in milliseconds, so we should use 3000 ms (=3 seconds).

The block diagram picture below shows the modified VI. Note that the Action_Digital Out and pause subVIs had to be added into the code right after the LabPro was initialized. To do this we had to break the error line and weave it through the new subVIs to make them happen in the right order. Also, the Action_Digital Out subVI requires the comm line to be wired, so it knows which comm port the LabPro is using.



Modified Block Diagram of DCUCOUNT.vi

Now, add this code to your program and try running it again. Watch the DCU LEDs. You should see the 1st and 2nd DCU LEDs go on for three seconds and then the LEDs cycle through the 16 outputs, more slowly than before.

Exit LabVIEW, or open another DCU sample program if you like. You may or may not want to save the changes you made. You also can save your changed version of the program with a different name.

This is an example of how you can use the raw material we provide in the window of LabVIEW code to create your own DCU control programs. LabVIEW is an extremely powerful programming tool, and we have provided a lot of VIs for controlling LabPro for data collection and analog and digital output. To learn how to properly use these VIs in your programs refer to the next section of this manual, *Additional Notes on LabVIEW Programming*. Better yet, spend some time opening, trying out, and examining the sample VIs in the DCU VIs folder. Try making minor changes and see if they work. The best way to learn programming is to actually do it.

Additional Notes on LabVIEW Programming

Additional LabPro Sample VIs

The DCU CD contains LabVIEW VIs that are focused on controlling the DCU and analog output lines. We have posted additional sample VIs that demonstrate other ways to control the LabPro. Visit our web site for more documents and sample VIs at: www.vernier.com/labview

Collecting Data to Control Outputs

Some of the most interesting programs include the use of sensor data for feedback and control. Example programs that demonstrate how to collect data from a sensor, and then perform a specific digital output based on the sensor reading, include: DCUALARM, DCUSUN, DCUTEMPC, and DCUWARNV. The best way to combine data collection with digital or analog output is to collect the data using the single data point technique. This style allows you to go back and forth between collecting data and performing an output relatively seamlessly. DCUALARM, DCUSUN, and DCUTEMPC all use this style. The drawback to this technique is that the data collection is slow, and data is collected only when asked for, not at a defined interval. DCUWARNV combines data collection and digital output using the real time (RT) data collection technique. A sampling period is configured for the analog channel, which means LabPro will spit out data at the sampling period until told to stop. You will find that this style of data

collection does not mix well with analog output. It can only be done if you re-configure the sampling period after every time you set an analog output.

If you are just interested in writing a program to collect sensor data, without intermixing digital or analog output, study the DCUWARNV program. Remove the code used for digital output (the Case Structure), and add a control for the "Sample Time" node found in the RTSamplingSetup Express VI. Perform these two steps and you will have created a simple data collection program.

Direct Digital Output

A direct digital output simply means that a digital line (or lines) is turned on, and remains on, until modified or turned off. This is simply a matter of sending LabPro a 2001, 2011, or 2012 command. We use the 2001 command frequently in the DCU programs with a subVI called "Action_Digital Out.vi". Look at the program called DCUTEMPC to see an example of how this subVI is used. In this program, a sensor is monitored until the data reaches a certain value, at which time a line is turned on or off. The subVI creates a 2001 command string, and all you must do is enter the value (as a string) that will tell the DCU what lines to turn on and off.

For more information on the 2001 command and the values that are used for turning lines on, see the *Programming for the DCU* section of this manual.

Digital Output Sequence

For a good example of performing a digital output sequence, open and study the program DCUMASS. In this program a custom sequence of 5 outputs is sent (1,2,4,8,0). Run the program, you will see the DCU's LEDs flash 4 times, the fifth output is a zero, so the LEDs will be turned off for this last output. Now, go to the block diagram and change the Numpoints constant. It is currently 5, change it to 10. Run the program and note that the sequence performs 10 outputs (1,2,4,8,0,1,2,4,8,0). Try Numpoints of 12, now the sequence will be (1,2,4,8,0,1,2,4,8,0,1,2).

Now change the Samptime front panel control from 0.12 to 1.0. Notice how the 12 outputs are slowed down, each one lasting for 1 second.

Now go to the block diagram and change the sequence from 1,2,4,8,0 to 1,2. This new sequence has 2 elements, and therefore the "Operation" constant must be changed from 5 to 2. Run the program. If you have left the Numpoints equal to 12 the output pattern will be (1,2,1,2,1,2,1,2,1,2,1,2), each output lasting 1 second.

For more information on sequence control see the *Programming for the DCU* section of this manual.

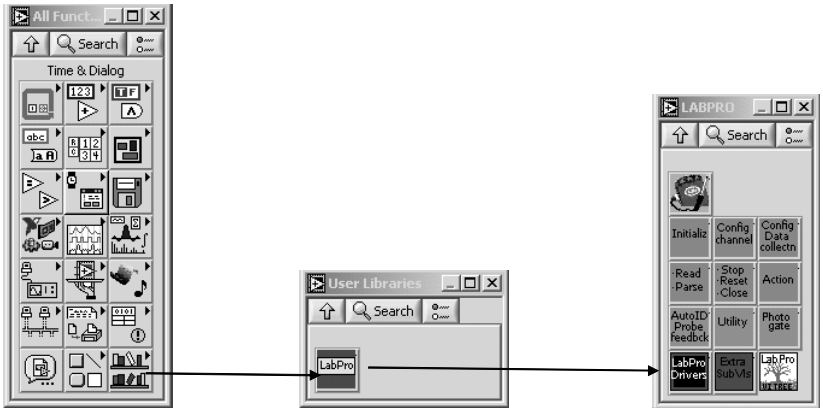
Using the LabPro subVIs and Express VIs

Programming to control the LabPro using LabVIEW is very different than programming in other languages, for two reasons. LabVIEW's graphical interface makes things easier and Vernier has provided more complete, higher-level code to support this programming. In fact, learning to program LabPro with LabVIEW really boils down to learning how to use a collection of VIs (Virtual Instruments) supplied on the DCU CD.

The best way to access these VIs is to find and open a working sample VI. Test the sample VI to prove that you have communication with LabPro and that the function (such as a digital output) works. Once you have convinced yourself that your hardware is working then you can take the next step of altering the program to create your own custom program.

If you will be starting a program from scratch, or if you need to add LabPro subVIs or Express VIs to a sample VI, there are two good methods to access the VIs:

1. Open a sample VI and then highlight and copy the subVIs, Express VIs or code you wish to use; paste these into the block diagram of your new VI.
2. From the Block Diagram, open the LabPro palette in your functions palette (Functions>>All Functions>>User Libraries>>LabPro) to access all of the subVIs and Express VIs. Once located, simply drag and drop the required subVI or Express VI onto your block diagram. A description of what this palette contains is provided later in this section. (NOTE - Installing the folder "LabPro" in the user.lib folder is required to provide access to these VIs in the functions palette.)

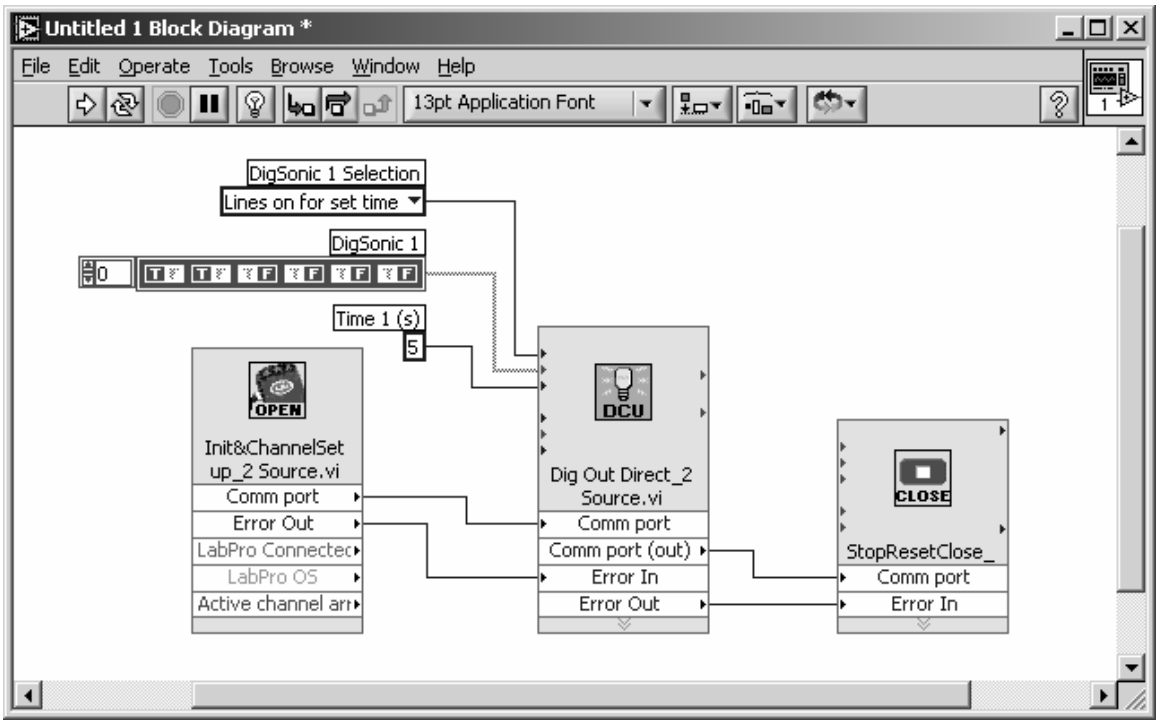


The LabPro Palette is Located in the User Libraries Palette.

Express VIs and subVIs

The LabPro folder contains Express VIs and also subVIs. Express VIs were introduced in LabVIEW 7.0 to provide an interactive means of configuring the inputs. Express VIs allow you to double click on them to bring up a dialog box, providing you with an easy method for configuring your program. As you know, a standard subVI is configured by the values that are wired into the subVIs connector nodes. An Express VI can also be configured in this manner. This means Express VIs can be configured in two ways; wiring in a value or inputting a value in the Express VI's dialog box. If configuration values are wired, they will take precedent over values that are input into the Express VI's dialog box. In most cases, Express VIs will be configured in both ways, with the more important inputs configured in the dialog box, and the less important inputs wired into the nodes. With our Express VIs you will always be required to wire in Comm Port and the Error In, at a minimum.

The LabPro Express VIs were created by us using a special toolkit from National Instruments; you will not be able to create your own Express VIs without having this toolkit installed. We have created a selection of Express VIs that perform most of the major functions of the LabPro. It is possible to view the subVIs that were used to create the Express VIs by right clicking (open-apple or command clicking on a Mac) on the Express VI and choosing Open Front Panel. The sample DCU programs do not use all of our Express VIs. You may want to take the time to look at the rest of these by going to the LabPro palette in the User Libraries function palette. In fact, in most examples we have chosen not to use our Express VIs in order to keep the commands more transparent. If you are looking over our examples, do not be confused when you see a section of code consisting of a handful of subVIs that could be replaced by one of our Express VIs—we wanted you to see more of the details.

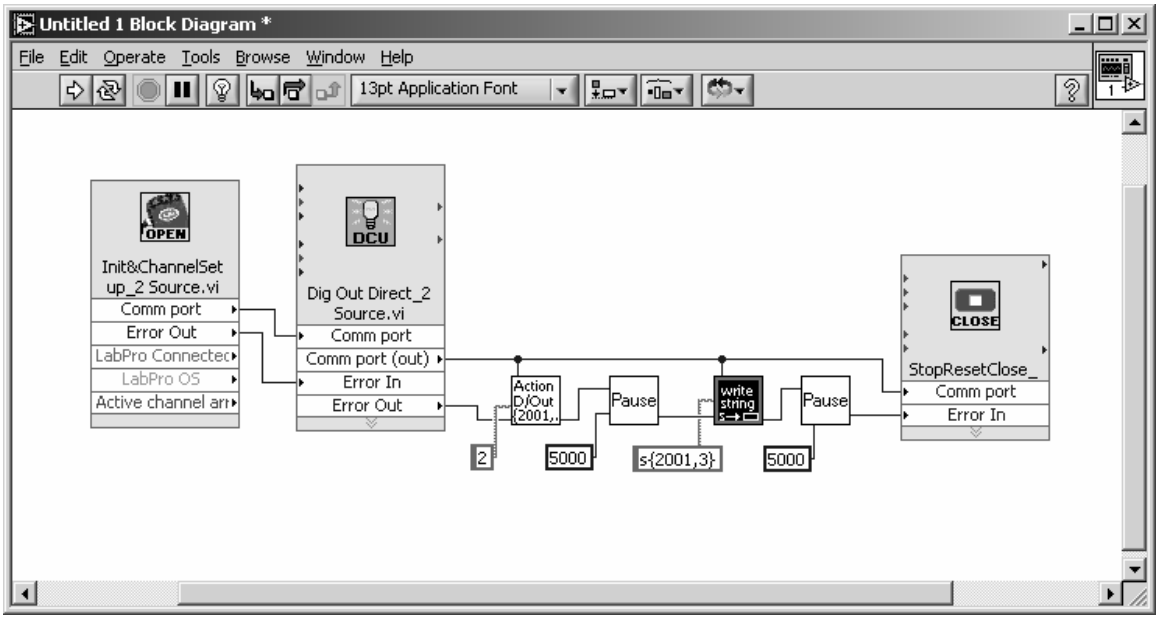


Example Program using LabPro Express VIs to turn on lines D1 and D2 for 5 seconds.

Low-Level VIs and High-Level VIs

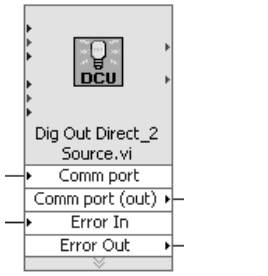
The LabPro collection of VIs includes both low-level and high-level VIs. The low-level VIs provide easy, transparent access to the LabPro command and communication code. The high-level VIs hide the LabPro command strings and the communication code. We have provided high-level VIs, such as the Express VIs, to allow users to create data acquisition and control programs without having to learn the low-level details, such as the LabPro command strings. But if you look at the block diagram of these high-level VIs, and dig into the subVIs within these VIs, you will eventually find the low-level subVIs that perform the communication and send the string commands to LabPro.

When you are writing your program, it is best to look for an Express VI or subVI in the palette that performs the required function. Using these high-level VIs should make your program cleaner, easier to follow, and faster to create. However, if you are interested in writing programs by sending string commands directly to the LabPro, or if the high-level VIs do not provide enough flexibility, then you will want to use the low-level VIs.

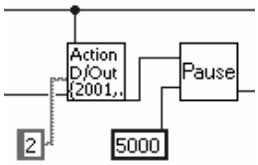


This example program demonstrates the difference between high-level and low-level VIs. The Dig Out Direct Express VI hides the 2001 command and is therefore considered high-level. The subVIs that send 2001 commands show portions of the 2001 command string and are therefore considered low-level.

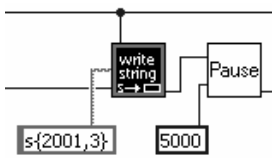
The sample program above highlights the difference between the high-level and low-level subVIs. In this program, line D1 is turned on for five seconds (configured by double clicking the Dig Out Direct Express VI), then line 2 is turned on for five seconds (5000 milliseconds), and then lines one and two are turned on for five seconds (remember that sending a 3 turns on lines 1 and 2). In every instance, a 2001 string command is sent to the LabPro. The actions (outputs) are isolated in the diagrams below. With the first output, the s{2001,1} command is completely hidden within the Express VI. Someone could write a program, using this Express VI to turn on the digital lines, without knowing anything about a 2001 command string. In the second output, the 2001 command is partially hidden. The user inputs a 2 (as a string, not a number) and this gets concatenated within the subVI to create the s{2001,2} command string. In the third output, the user must create the entire command string. This means, unlike the Express VI, the user must have some knowledge of the LabPro commands.



First output using Express VI



Second output using subVI



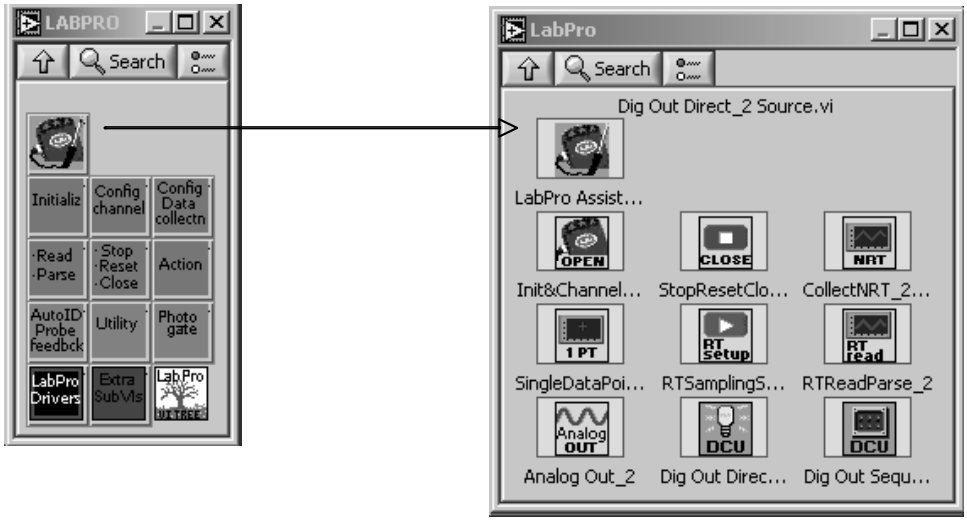
Third output using LabPro command

The LabPro palette contains both the high-level and low-level VIs to provide for programming flexibility and to match the type of programming you wish to do. In any case, review sample programs to better understand the VIs and how to program the LabPro using LabVIEW.

LabPro VI Palette Overview

Here is a summary of the VIs found in the LabPro folder and accessed from the User Libraries function palette. When you have properly placed the LabPro folder in the user.lib folder you will have access to the LabPro palette of VIs by going to the User Libraries function palette.

The VIs in this palette are the building blocks of a LabPro program - sending and reading commands to LabPro via the serial or USB port. When you build a LabPro program you should place the appropriate VI on your block diagram, provide the proper inputs, and wire the VI in such a way that it will occur in the proper sequence. Use the DCU Manual, the *LabPro Technical Reference Manual*, and example VIs to help you understand how to use these VIs properly. In addition, when a VI is placed on the block diagram you can turn on the Show Context Help (in the Help menu) and get information about the VI by placing your cursor over the VI. Here are the most commonly used VIs:



The LabPro Express VI palette is accessed by placing your cursor over the LabPro icon.

The top palette contains the LabPro Express VIs. These are very high-level VIs that perform most of the functions of the LabPro. A description of the ten Express VIs follows:

LabPro Assistant: Use this Express VI to verify that you have a good connection between your computer and the LabPro. This is a stand-alone Express VI that collects a single data point from a Vernier Analog Auto-ID probe. All of the steps required to collect one single data point occur within this VI, including: Open the port, initialize LabPro, configure the channel for an Auto-ID probe, collect a single data point and close the port. This VI is not meant for use within a larger program, just as a very easy way to get started with LabVIEW data collection.

Init&ChannelSetup: This Express VI locates and resets the LabPro, identifies the comm line it is connected to, and returns the firmware version of the LabPro (if you want to know it). Use this Express VI to configure your analog channels or your digital channels for the motion detector. For example, you can specify that you want to read an AutoID sensor on CH1 and a Motion Detector on DIG/Sonic 2.

StopResetClose: This Express VI stops any data collection or output, powers down LabPro, and closes the computers' USB or serial ports. You must end your program by closing the port, otherwise there can be unexpected and unwanted behavior.

CollectNRT: Non-real time (NRT) collection simply means that data are not returned from LabPro until all of the requested data are collected. This Express VI is used to collect data in NRT from the "active" analog channels 1-4 (made active in the Init&ChannelSettup Express VI). Once all of the data is available, this Express VI will return the data in array format.

Single Data Point: This Express VI lets you read one data point from any channel configured by the Init&ChannelSettup Express VI.

RTSamplingSetup: This Express VI lets you set the rate of data collection for real-time data collection. In typical RT data collection programs the "active" channels (configured in the Init&ChannelSetup Express VI) will begin collecting data as soon as this Express VI is called. The Express VI called RTReadParse is then placed in a loop to collect the data.

RTReadParse: This Express VI reads real-time data after you have configured the channels and set the data rate. Note that there is nothing to configure in this VI.

Analog Out: This Express VI controls the analog output driver on LabPro Channel 4. Output begins immediately regardless of data collection mode and will remain active until LabPro is reset or disabled (this can be done with this Express VI or with the StopResetClose Express VI).

Dig Out Direct: This Express VI configures the DigSonic channel(s) to perform digital output. LabPro's DigSonic channels can be used to perform digital output during a sampling run. This provides a good way to turn on a fan, light bulb or buzzer, for example. Sending digital output using this method does not stop sampling, and it should not affect the analog channels.

Dig Out Sequence: This Express VI lets you control the DCU lines in a sequence. This is the way to set a digital output pattern for running a stepper motor, turning a servo motor, flashing lines on and off and many other operations.

The next set of VIs that are discussed can be found in the nine subpalettes circled.

Moving down the palette, we will now look at some of the VIs found in the next nine subpalettes. Not all of the VIs are summarized, but it should give an idea of what is available.

Configure_Digital Output CH1: Located in the Configure Channel subpalette. There is a separate VI for configuring CH2.

This subVI is used to set the output value on DIG/Sonic 1 to a series of output states. LabPro outputs one element for each sample. This command is used for sequential digital outputs used for turning a stepper motor, flashing all of the LEDs, or turning a servomotor, for example. Several common output states are pre-programmed in this subVI, and can be selected from the Ring control. In addition, this subVI allows you to create your own custom sequence by selecting the "custom,.." option from the ring control, and then sending a string value to represent the operation and list of values. The string command that is created in this subVI is s{1,31, operation, list of values}. The inputs to the VI follow:

Comm port: port number of the open port

Ring: Ring control with various pre-programmed options to run

Operation: If you choose "custom" from the ring control you will need to input this parameter. 0 = clears the channel; 1-32 = Count (number of output patterns in list)

List of values: Values may be from 0 to 15; must have one value for each count.

Configure_Analog CH1: Located in the Configure Channel subpalette. There are similar VIs for CH2, CH3 and CH4

This subVI sets up channel 1 for data collection. These VIs do not appear in the DCU example programs because the Init&ChannelSetup Express VI performs the same function. However, these VIs provide more configuration options for the channels. The string command that is created in this subVI is s{1,channel,operation,post proc, delta,equ}. The inputs to the VI follow:

Comm port: port number of the open port

Operation: indicates the units of measure the channel should return; use 1 to Auto-ID the sensor

Post-Proc: enables post processing to take place on data collected. Since data is not stored in RT mode, no post processing is allowed

Delta: always set to zero

Equ: Indicates if conversion equation is applied to data before it is returned

ms to wait after: amount of pause after sending the command

Config_DataCollectionSetup: Located in the Data Collection subpalette.

This program sets up the LabPro for data sampling. There are a lot of parameters that can be used with this command, but only the first few are given a connector on this subVI. The string command that is created in this subVI is s{3, samptime, numpoints, trigtype, trigchchan}. The inputs to the VI follow:

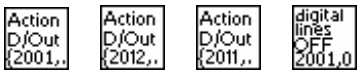
Comm port: port number of the open port

samptime: sets time between samples in seconds. The possible values are 0.00002 to 16,000 seconds. May also be -1 to repeat the previous sampling

Numpoints: Indicates the number of data points per input to be recorded. The possible values are 1 to 12,287. Setting this value to -1 puts LabPro into realtime data collection mode.

Trigtype: Indicates what events must occur on the trigger channel to start sampling (default is 0)

TrigChan: Indicates on which active channel the trigger conditions occur.



Action_Digital Out: Located in the Action subpalette. Note that there are similar subVIs called Action Digital Lines Off, Action_DigOut2011 and Action_DigOut2012. Action Digital Lines Off sends an s{2001,0} command to turn off all lines. The 2001 command effects both ports, whereas the 2011 and 2012 command only effect the DIG/Sonic 1 port and DIG/Sonic 2 port, respectively.

This subVI is used to output data to the digital output port during a sampling run. One to sixteen data points may be output. Sending this command does not stop the sampling and it should not affect the analog channels. This subVI is used for such things as turning on a light bulb or turning a DC motor. In most cases, just a single data value is used (e.g s{2001,1}), although there is an option of sending more than one. Note that the input for the data value into this subVI is a string, not a numeric. String commands are sent to LabPro, and in this case the input to the subVI is simply a part of the 2001 command string. The string command that is created in this subVI is s{2001, data1, data2, data3,...dataN}. The inputs to the VI follow:

Comm port: port number of the open port

data1, data2,....,dataN: Data values must be between 0-15. For values outside this range, behavior is undefined.

Action_Analog Output: Located in the Action subpalette.

This subVI sets parameters to control the analog output driver in LabPro. The analog output is present on line 1 of CH4. Therefore, the voltage clips supplied with LabPro may be used to connect the analog output to external circuits. Once the channel has been configured, the output is enabled immediately regardless of data collection mode, and will remain active until the unit is reset or until it is disabled. Waveforms other than DC values are divided into steps of discrete values. Since this analog out shares the CH4 Vin pin, you may monitor the output voltage by configuring CH4 to read the analog input. Note that Vout = 0.0024*amplitude - 0.0012*offset. The string command that is created in this subVI is s{401, waveform, amplitude, offset, period}. The inputs to the VI follow:

Comm port: port number of the open port

waveform: ring control with all of the possible waveform values

amplitude: height of the wave. The possible values are from 0 to 4095

offset: indicates where the wave will be centered. The possible values are from 0 to 4095

period: indicates how often (in milliseconds) the waveform will repeat itself. The number must be an integer between 5 and 2000. As a result, the range of possible frequencies is 0.5 Hz to 200 Hz.

Pause: Located in the Utility subpalette.

This subVI is used for providing LabPro the required time to finish a command or to hold a command on for a specified amount of time. It simply waits the specified number of milliseconds. No command is sent to LabPro.

StopCollect: Located in the Stop Close subpalette.

Use this subVI to stop data collection. When collecting in realtime mode you will need to send this command to halt the collection. This subVI sends the s{6,0} command to stop sampling.

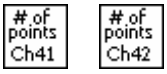
Photogate_Config Ch: Located in the Photogate subpalette.

This subVI configures DIG/Sonic 1 and/or DIG/Sonic 2 to be used with the photogate. Remember that you must have at least one analog channel activated before sending the Command 3 to start sampling. The string command that is created in this subVI is s{12,4x,10}. The inputs to the VI follow:

Comm port: port number of the active port

ms to wait after: pause following the write to LabPro

Photogate channel: ring control that allows you to pick which channel(s) to configure



Photogate_NumPts_Ch41: Located in the Photogate subpalette. There are subVIs for DIG/Sonic 1 and DIG/Sonic 2.

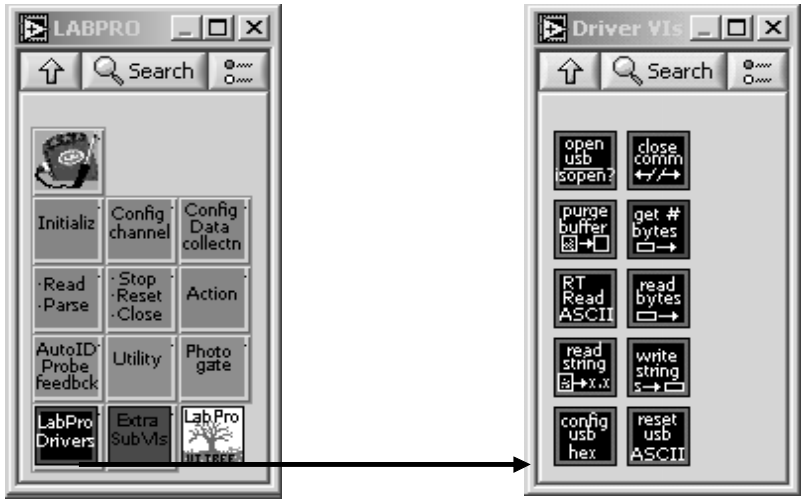
This subVI is used to get the number of transitions recorded by a photogate connected to DIG/Sonic Channel 1. Use this subVI in a realtime data collection loop for tracking the number of transitions in realtime. This subVI reads all

of LabPro's data and parses out the transitions recorded. The string command that is created in this subVI is s{12,4x,0}. The inputs to the VI follow:

Comm port: port number of the active port

ms to wait after: amount of pause between writing the command and reading the data

bytes to read: the number of bytes to read. 19 bytes for the photogate data



This subpalette contains our low-level driver VIs.

The last subpalette discussed contains the low-level LabPro Driver VIs. We will not go into detail about these different VIs and what their function is. Use these VIs to create commands that do not exist in the LabPro palette, to create your own custom subVIs, or to sends commands directly to LabPro. Even if you do not use these VIs you should be aware that they exist, and that if you dig deep into the high-level VIs you will always find these VIs doing the communication work.

LabVIEW Troubleshooting Help

Problems with communication over the serial port: NI-VISA drivers were not installed. This driver is a National Instruments driver found on the second installation CD labeled "Device Driver Reference CD". If you are using the LabVIEW Student Edition this version does not contain the driver sets. This driver needs to be installed or it can be downloaded from the National Instruments website for free. Go to www.ni.com

Problems with communication over the USB port: The USB driver must be installed and the LabProUSB.dll file must be present on your computer. The LabProUSB.dll is part of the LabPro folder of the DCU CD and it should be placed into the user.lib folder of your computer. The USB driver (called windrvr6.sys) is automatically installed when you install Logger *Pro* 3.3. An installer is also available from our website at www.vernier.com/drivers. In some instances you may have to have LabPro plugged in before starting LabVIEW.

LabVIEW quits unexpectedly: Do not unplug the USB port while LabVIEW is running. Restart your computer.

Problems finding LabPro: Do not have other programs (such as Logger *Pro*) running that also use the serial or USB ports. In some instances you must have permission for the serial port and USB port.

Old and new driver files have become intermixed: Delete old 5.1 VIs and place new "LabPro" folder in the user.lib directory

The Initialization VI states that OS = 0.00: This can happen if you are on an international machine and your computer is setup with the decimal separator as a comma. Fix this by changing the computer's setting or by changing specific string to decimal and decimal to string functions.

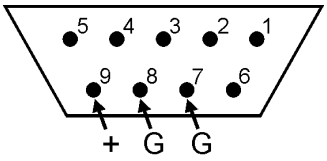
Four beeps from LabPro: You have sent an illegal command to LabPro or you have sent commands faster than about 30 commands/second. If there is no delay between sending commands (using the WriteString.vi) LabPro's buffer will overflow or commands will get chopped up and 4 beeps will occur.

Problems the second time a VI is run: If you do not properly stop and close the port you may create any number of problems. Do not stop a program with the Stop icon tool in the LabVIEW toolbar.

An error value of 107367xxxx is returned from the Serial Port Init.vi: VISA must be installed. Also, on Linux and Mac platforms a "warning" is given when running VISA. This warning is sent in all cases do to the fact that you are not allowed to change the buffer size in VISA if running on Linux or Mac OS X.

Connecting Devices to the DCU

For connecting electrical devices, use the 9-pin, sub-D socket on the side of the DCU. There are connections for all six digital lines, plus power and ground. The diagram below shows the connections to the holes on the socket on the cable. This is an easy thing to confuse; these are not the pins on the DCU box.

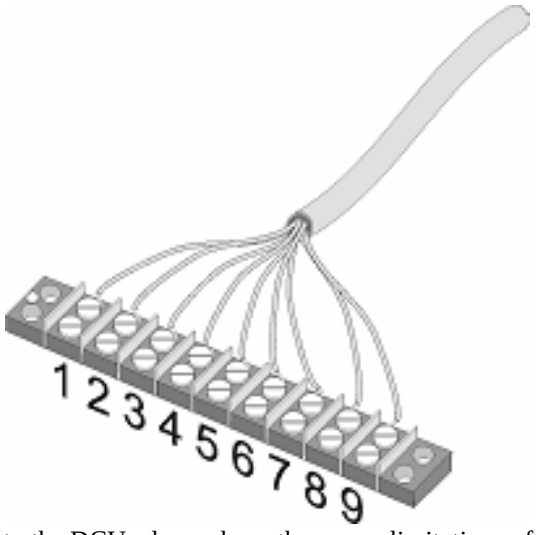


The End of the Cable that Plugs into the DCU, Looking into the Holes

The holes labeled 1, 2, 3, 4, 5, and 6 are the digital output lines. G is for ground and + is for the power from the power supply of the DCU.

We provide one cable to plug into this socket, with the lead wires identified for you to use on your first projects. To build most projects, you will want to make connections to this cable. For testing, twisting the wires together is probably ok, but eventually you will want to solder the leads or make some other connection. Whatever you do, make sure you insulate the leads so that they cannot accidentally touch each other.

We recommend using a terminal strip like this one shown below. A terminal strip like this is included in the Vernier Robotics & Control Kit (RCK-DCU).



As you build devices to connect to the DCU, always keep the power limitations of the DCU in mind. For the entire DCU, it should not exceed the current limit of your power supply. This limit is 300 mA when using the CBL 2 power supply, 600 mA when using the LabPro power supply, and 1000 mA when using the ULI power supply. If you are using a different power supply, check the current rating. In general, you will not damage the DCU by trying to draw too much current, but the circuit will not work properly. It may also be possible that you could damage the power supply. The current draw for any one line should not exceed 600mA, no matter what. Remember that Ohm's law controls the current that flows through the device:

Current (amperes) = Voltage (volts) / Resistance (ohms)

In some cases you can check the resistance of the device with a meter and calculate how much current it will draw using Ohm's law.

Making Additional Cables to Connect Projects to the DCU

If you need additional cables, you can find the 9-pin, sub-D sockets at most electronic stores; for example, part number 276-1538 from Radio Shack will work. You would have to solder lead wires to each of the connections you use. Even better is to find assembled cables that you can use; for example, Radio Shack #26-152, can be cut in half to produce two useful cables. To identify the lead wires on a cable, use a meter which will indicate conductivity. The meter allows you to determine which wire on your cable connects to each of the holes on the connector. The easiest way to do this is to stick a paper clip in one of the holes on the plug. Connect one probe of the meter to this paper clip inserted in a hole in the end of the cable and then touch each of the bare wires with the other probe until you find one that connects (meter reads near zero resistance). This is even easier if your meter has a setting to make a sound when conductivity is found. As you determine the pattern of wires, either label the leads or make a table indicating a color code of the wires to each hole.

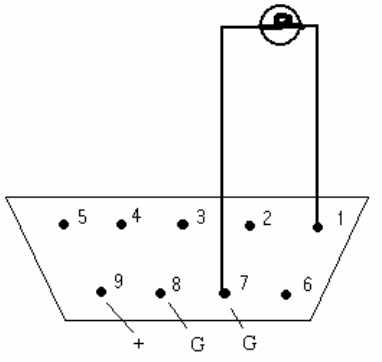
Safety Note:

When making your own cables, if you leave the + wire exposed and it touches a ground wire, you could have a direct short of the DCU power supply. If you leave it exposed, it could short out against one of the other bare wires. This could damage the power supply. To avoid this, once you identify the 9 lead wires on the cable, cut the bare wire off of the + connection. This lead is not used very often. We have cut this wire on the prepared cable we include with the DCU.

What can you connect to the DCU output lines? The general answer is any electrical device which is meant to run on DC electricity at a voltage that matches the voltage you are using for your power DCU power supply. Most of the time the LabPro or CBL 2 power supply is used with the DCU, and it is a 6-volt power supply. If you are using this power supply, you should use 6-volt devices. Thousands of different motors, stepper motors, lamps, buzzers, solenoids, and other devices are designed to run at 6-volts. The Vernier Robotics and Control Kit (RCK-DCU) includes many devices of that sort, ready to use with the DCU.

Connecting One Simple, Non-Polarized Device

To connect a simple, non-polarized, electrical device such as a lamp, DC motor (not a stepper or servo motor), resistor, or electromagnet that you just want to turn on or off, use this wiring pattern:

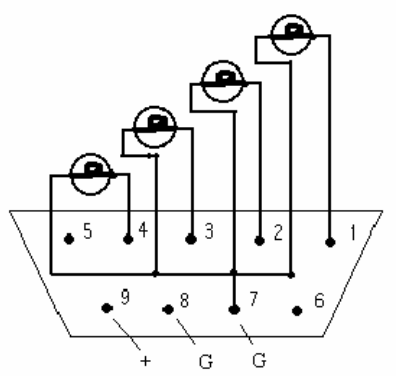


Wiring a Simple DC Electrical Device

The device is shown in this diagram as a small lamp, but it could be any electrical device that does not have positive and negative leads.

Connecting More Than One Simple Device

If you want to wire a number of simple electrical devices, repeat this wiring using additional digital output lines. The wire to hole #8 can also be used for ground connections. These devices will be turned on when the corresponding digital output (D) line is on (see table in hardware section).



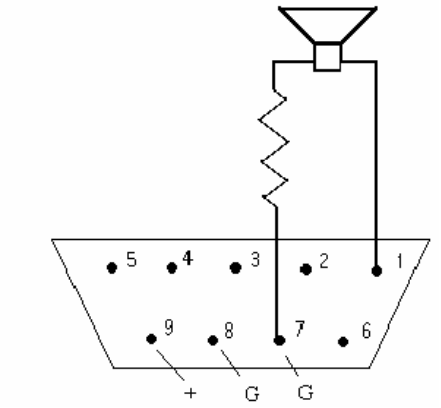
Wiring of Several Simple Electrical Devices

Notes:

- Three DC devices can be turned on completely independently and in any combination if they are connected between D1, D2, D3, and ground.
- Six DC devices could be connected between the six output terminals and ground. Four (those connected to D1, D2, D3, and D4) can be used in almost any pattern, except that you could not have D3 and D4 on at the same time. The devices connected to D5 and D6 can be turned on only when all the devices connected to D1 through D4 are off.

Connecting a Speaker

If you connect a small speaker to the DCU, you will probably want to put a resistor in series with it so that the speaker is not too loud. The resistor should be a power resistor, rated at least at 0.5 watts and a fairly low resistance. The larger the resistance, the quieter the speaker will be. When using a speaker, you must use a program that turns the power to an output line on and off at a frequency of a hundred hertz or so. Our sample calculator program DCUBUZZ does this.

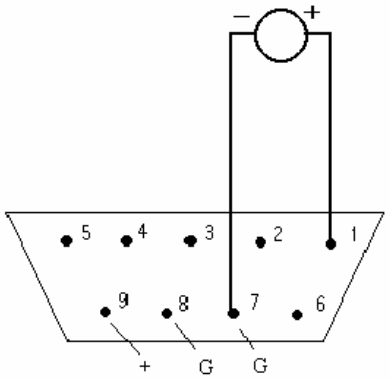


Wiring a Speaker

Connecting a Polarized Electrical Device

Some electrical devices are *polarized*; that is, they have a positive and a negative side. They therefore must be wired in one particular way for use with the DCU. Examples include some buzzers, a few lamps, and complex electronic devices. LEDs are given special treatment below. For devices that have positive and negative sides, make sure you

connect the negative side to ground. Devices wired this way will be on any time D1 is high.

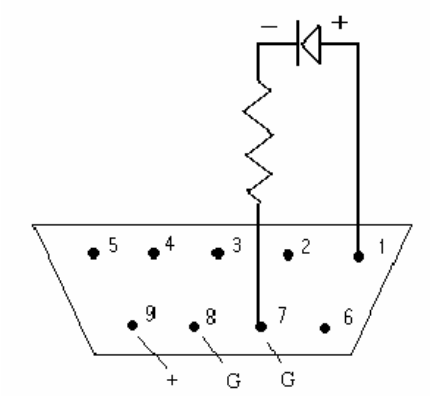


Wiring a Polarized Electrical Device with Positive and Negative Leads

You can wire up to six polarized devices in the same way using connections 1–6 for the positive leads and the G connections for the negative leads.

Connecting an LED

Simple LEDs have positive and negative sides, so make sure you connect the negative side to ground. Also, most LEDs need to be wired with a resistor in series with them, in order to limit the amount of current that flows. Typically, this resistor has a value of 220 ohms or higher. Check the specifications on your specific LED, if possible. The flattened side of the plastic part of the LED marks the negative side. The positive lead is usually longer than the negative lead.



Wiring a LED with Positive and Negative Leads

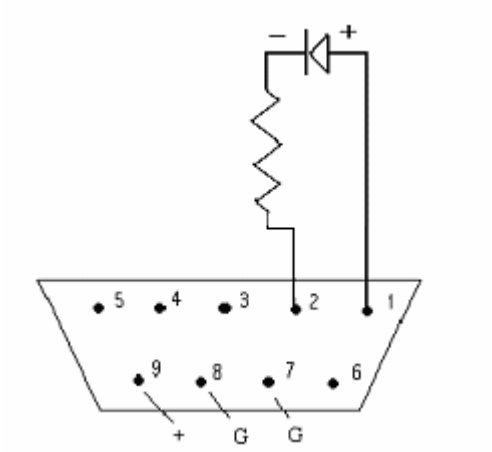
You can wire up to six polarized devices in the same way using connections 1–6 for the positive leads and the G connections for the negative leads.

Connecting a Bi-Color LED

Bi-color LEDs can be wired so that the current flows either way through them. They will produce different colors depending on the direction the current flows. If you wire one (with a current-limiting resistor in series) between D1 and D2, you can cause the current to flow either way as follows

If you send a 1 to the DCU, line D1 will be high and D2 will be low and you will get one color.

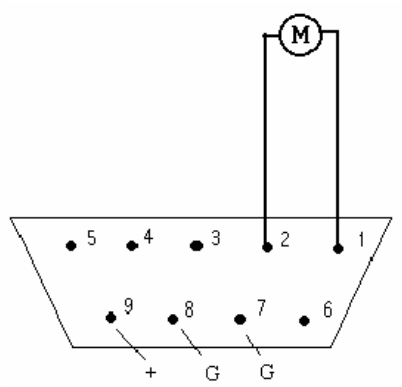
If you send a 2 to the DCU, line D2 will be high and D1 will be low, and you will get the other color.



Wiring a Bi-Color LED

Connecting a Motor for Running in Either Direction

Simple DC motors can be wired as shown above and they will either be off or on, rotating in one particular direction. If you want to have the ability to run the motor in either direction, you have to wire it as shown below.



Wiring of a Simple DC Motor to Run Either Direction

For a motor wired this way, you will get one direction of rotation if D1 is high and D2 is low. You will get the opposite rotation if D2 is high and D1 is low. It will be off for all other patterns. You can connect a second motor wired this way to D3 and D4, and even a third motor connected to D5 and D6. The chart below shows the output patterns of the DCU and the direction of rotation of three motors that can be wired using output pairs 1-2, 3-4, and 5-6.

Output	D1	D2	D3	D4	D5	D6	Motor 1-2	Motor 3-4	Motor 5-6
0	—	—	—	—	X	X	Off	Off	Off
1	+	—	—	—	X	X	**CW**	Off	Off
2	—	+	—	—	X	X	**CCW**	Off	Off
3	+	+	—	—	X	X	Off	Off	Off
4	—	—	+	—	X	X	Off	**CW**	Off
5	+	—	+	—	X	X	**CW**	**CW**	Off
6	—	+	+	—	X	X	**CCW**	**CW**	Off
7	+	+	+	—	X	X	Off	**CW**	Off
8	—	—	—	+	X	X	Off	**CCW**	Off
9	+	—	—	+	X	X	**CW**	**CCW**	Off
10	—	+	—	+	X	X	**CCW**	**CCW**	Off
11	+	+	—	+	X	X	Off	**CCW**	Off
12	X	X	X	X	—	—	Off	Off	Off
13	X	X	X	X	+	—	Off	Off	**CW**
14	X	X	X	X	—	+	Off	Off	**CCW**
15	X	X	X	X	+	+	Off	Off	Off

Controlling Motors connected to D1-D2, D3-D4, and D5-D6
*(**CW** = Clockwise, **CCW** = Counterclockwise)*

Note:

- If a third simple motor is connected to D5 and D6, the third motor can be run only when the first two are off, but it can be run in either direction.

Connecting LEGO® and Fishertechnik® Motors

We have used the motors from the LEGO. Mindstorms Kit and the Fishertechnik motors with the DCU. Both companies make motors that are normally run on 9 volts. They will work pretty well connected to a DCU powered by a LabPro power supply (6 volts). The advantage of using these motors is that there are lots of gears and pulleys designed for use with them and they have nice systems for mounting the motors. LEGO motors are available from PITSCO, www.pitsco-LEGOeducation.com. Fishertechnik information is at info@fischertechnik.de.

Connecting Stepper Motors

Stepper motors are very different from simple "DC"or "commutator" motors which have two lead wires. Stepper motors are used in cases where you want to have exact control of a motion. Examples include the positioning of the head on a disk drive or the laser in a CD-ROM player.

There are basically two types of stepper motors that you may want to use: *unipolar* and *bipolar*. You can identify which type you have with this rule:

- Bipolar stepper motors have 4 lead wires.
- Unipolar stepper motors have more than 4 wires, usually 5, 6, or 8.

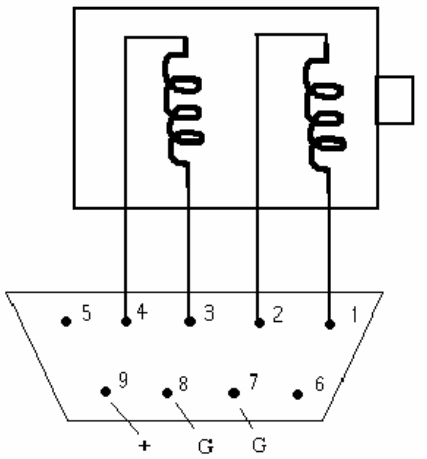
No matter what type of stepper motor you have, remember that it needs to match the voltage you are using on your DCU power supply. If you are using the LabPro/CBL 2 power supply you should use 6-volt stepper motors.

Identifying the leads on a stepper motor can be tricky. It helps if you have a diagram provided by the manufacturer. Unfortunately, you are often using a surplus stepper motor and need to figure it out yourself. First, determine which type of stepper motor it is. Next, look for patterns. Examine the wires carefully. Refer to the diagrams below which symbolically show how the two types of stepper motors are wired inside. Use a meter to measure resistance. Remember that a coil will have a few ohms of resistance. Use a little trial and error and you will be able to get it going.

If you use the stepper motor which comes with the Vernier Robotics & Control kit (RCK-DCU), the leads are identified in the documentation which comes with it.

Bipolar Stepper Motors

To connect a bipolar stepper motor directly to the DCU, wire it as shown here:

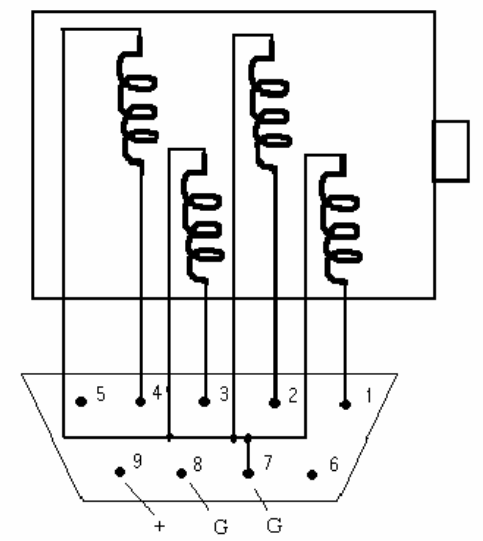


Direct Connection of a Bipolar Stepper Motor

Wired this way, you can control the bipolar stepper motor using the DCUSTEP. You can also use the DCUSTEP3 program, which is an improved, more complicated program for controlling stepper motors.

Unipolar Stepper Motors

Unipolar stepper motors are more difficult to figure out for wiring than bipolar ones. Often all the ground wires are the same color or similar colors. Use a meter and the "trial and error method" to figure out the wiring pattern. Wire them as shown below.



Direct Connection of a Unipolar Stepper Motor

The programs for unipolar stepper motor are the same as for bipolar. Wired as shown above, you can control the unipolar stepper motor using the DCUSTEP or DCUSTEP3 programs.

There are three different methods of driving a stepper motor: *Normal, Half-Step*, and *WaveDrive*. The DCUSTEP and DCUSTEP3 programs use the Normal method. In this case, electromagnets inside the stepper motor are always turned on two at a time as the motor steps. In the Half-Step Method, intermediate steps with only one electromagnet on at a time are included. This gives you more precision in the positioning of the stepper motor. The Wave method of driving a stepper motor has only one electromagnet on at a time. For this reason, the Wave Drive method uses less electricity, but the motor will have less torque.

Below are the DCU output patterns to use for rotating a stepper motor clockwise and counterclockwise in each of the three drive modes. We use the Normal drive method in all our programs, but you may want to try the others.

For "Normal" Stepper Motor Rotation:

- 5,9,10,6 for clockwise
- 6,10,9,5 for counterclockwise

For "Half-Step" Stepper Motor Rotation:

- 5,1,9,8,10,2,6,4 for clockwise
- 4,6,2,10,8,9,1,5 for counterclockwise

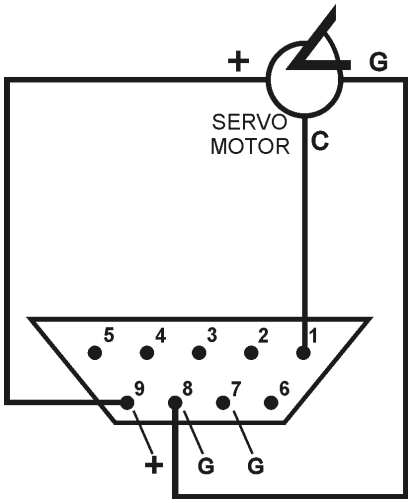
For "Wave Drive" Stepper Motor Rotation:

- 1, 8, 2,4 for clockwise
- 4, 2, 8, 1 for counterclockwise

There are other ways to connect stepper motors. If you use a stepper motor control integrated circuit (IC), you will need fewer wires to control the stepper motors. This will allow you to control two stepper motors with the DCU. One type of stepper motor control IC uses just two lines to control the stepper motor. One line is held high or low to indicate the direction the motor should rotate. The other line is toggled on and off one time for each step the motor is to move. This type of stepper motor control IC is assumed for use with our sample calculator program DCUSTEP2. Other stepper motor control ICs may operate differently.

Servo Motors

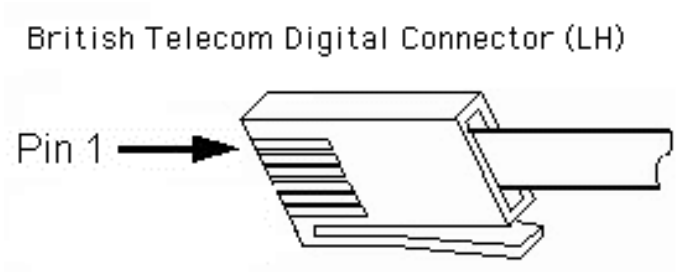
Servo motors are very popular in robotics because they have a lot of torque for their size and power used. Servo motors are controlled by pulse-width modulation, that is, a square-wave, on-off pattern is sent to their control line. The length of time that the square wave is at the high voltage is varied to control the servo motor rotor's position. The servo motor will hold its position firmly as long as the square wave continues. We can create the necessary square waveform from the DCU. The DCUSERVO program is an example. It will produce the necessary square wave pattern for most servo motors.



The easiest way to connect a servo motor is to connect the motor's power line to the DCU + connector and the motor's ground lead to ground, and the signal line to D1. This is the connection assumed by the DCUSERVO programs.

You can actually control a servo motor directly from the LabPro or CBL 2 Dig/Sonic port (without the DCU). The reason is that the signal line used to control the servo motor requires very little power. Connect the +5V, and Ground lines directly to the appropriate pins of the Dig/Sonic port, and then connect the control line of the servo motor to pin 1. The connections on the Dig/Sonic ports are shown below:

Pin1	I/O1
Pin2	I/O2
Pin3	I/O3
Pin4	PWR (5.3V)
Pin5	GND
Pin6	I/O4



British Telecom Digital (BTD)–Left Hand

Connecting Devices to the LabPro Analog Out Line

The Analog output of LabPro is used by connecting the standard Voltage Probe which comes with the LabPro (VP-BTA) to CH4. This analog output signal can be used for lots of projects, but keep in mind the limitation of +/- 4 Volts, and 100 mA maximum, with a frequency range of 0.5 to 200 Hz.

Here are some projects we have done using the analog output lines of LabPro. These can be done using the speaker, lamp, and bi-color LED that is included in the Vernier Robotics and Control Kit (RCK-DCU).

Drive a stretched string from a small speaker. Use a small speaker with a paper clip glued to it. Mount the small speaker with the cone pointed upward. Connect an elastic cord to the paper clip. The other end of the cord should be attached to a solid object, such as a ring stand. For software to drive the function generator of the LabPro, you can use either the FUNCTGEN or ANOUT programs. When LabPro produces the right frequencies, you can produce a number of standing-wave patterns.

We have used several different small 3 to 5 inch speakers. Some work better than others. The speaker included in the Robotic and Control Kit (RCK-DCU) was selected because it works well and has a higher than usual impedance so that it does not draw too much current.

We like to use a stretchy cord, like the kind that is used on name tags at conferences. We usually have a string length of a little less than a meter. Because of the fact that the analog output frequency resolution is much better at low frequencies than higher ones, you should use a low fundamental frequency in this demo. Start with the loosest string you can get that vibrates nicely.

Start the FUNCTGEN program and set it for a 4 volt, sine wave output. We usually start with a frequency of about 20-30 Hz. Experiment with the length and tension until a good standing wave pattern is obtained with the second or third harmonic. Carefully (and slightly) adjust the tension (length) until you get maximum amplitude of vibration at the antinodes. We usually get a movement of more than a centimeter at the antinodes. After you get a good pattern, go back to the fundamental frequency. You can calculate what it is. Next try other frequencies. Sometimes we can get up to the 5 and 6th harmonics displayed pretty well.

Drive a mass on a spring to explore resonance: Use a small speaker with a paper clip glued to it. Mount the speaker with the cone pointed downward. Hang a light-weight, flexible spring from the paper clip and then hang a small mass from the spring. Let the mass oscillate up and down naturally and estimate the natural frequency. Now use the analog output line of LabPro to drive the speaker with a sine waveform. Try to find the resonance frequency. Note the response at frequencies close to the natural frequency (resonance).

Build an Audio Voltmeter: Connect the analog output line to the speaker. Write a program to read a voltage probe (or really any analog probe) connected to CH1. Set up a 4-volt sine wave analog output and have the program adjust the frequency depending on the sensor reading. The result is a system which responds with frequency changes as a

sensor is used. This is a great tool for blind students and also can be helpful in troubleshooting intermittent sensors and circuits. The human ear is amazingly good at hearing changes in frequency.

Power a Bi-Color LED: Connect the analog output line to a bi-color LED. The most common ones produce red when the current flows one way and green when the current flows the other way. If you send a slow sine waveform out the analog output line you get a fascinating color transition from red to green to red. If you send a high-frequency sine wave to the LED, it will appear orange.

Control the DC voltage to a lamp and monitor its current: This is a great experiment to see the change in resistance of a lamp filament as it heats up. You can also monitor the light produced by the lamp as it goes through the cycle. Note that many lamps will draw more current than the LabPro analog output lines can supply. A number 48 lamp, rated at 2 V and 60 mA is perfect for these demonstrations. One of these lamps is included in the Vernier Robotics & Control Kit (order code RCK-DCU) and also in the Vernier Circuit Board (order code VCB). You can use either the ANOUT or FUNCTGEN program to control the LabPro analog output line. You might want to modify the ANOUT program to add the plotting of the output voltage. You can also use Logger *Pro* 3 on Macintosh or Windows, which has the advantage of automatically plotting a graph of the voltage and current vs. time.

One note about how the function generator of LabPro works may help you understand how to best use this feature. When you adjust the frequency of the analog output you are really controlling the period of the waveform, in millisecond increments, from 5 to 2000 ms. This means that you have very good resolution for frequency control at low frequencies, but not very good resolution at higher frequencies. For example, the very highest frequency is 200 Hz, followed by 166 Hz, 143 Hz, 125 Hz, etc. On the low frequency end, the lowest frequency is 0.5 Hz, and there are 1000 steps to 1 Hz. In the mid range, at 10 Hz, the steps are about 0.1 Hz.

DCU Project Ideas

The exciting thing about the Digital Control Unit is the projects you can build using it for control. Here are some projects ideas, with a few tips and suggestions. Most of these projects can be done with the Vernier Robotics & Control Kit (RCK-DCU).

Flashing Lamps and LEDs

You can use almost any lamp that has a voltage rating to match the DCU power supply you are using. Many different types are available from electronic supply houses, including Radio Shack. Lamps are almost always non-polarized, so they do not have to be oriented in one particular way.

LEDs (Light Emitting Diodes) are very inexpensive and last almost forever. LEDs are polarized, so they must have their positive and negative leads oriented properly. Normally the longer lead of an LED is positive. Also, the negative lead is usually marked with a flattened side on the LED. In most cases, you should connect a current-limiting resistor in series with the LED. This will limit the amount of current that flows through the LED. Without this resistor, the LED may quickly burn out.

There are now LEDs of many different colors; there are even two-color LEDs that will produce red if the current flows one way and green if the current flows the other way. If you quickly switch the polarity of a bipolar LED, you get orange.

It is easy to get the lamps and LEDs to flash in any pattern you want. Many of the sample programs included do this.

Moving Displays and Kinetic Sculpture

Simple DC motors can be used to create rotating or moving displays. Mount a colorful disk on a motor and have it spin to attract attention. Wrap a string around the shaft of a motor and hang a sign from it so that when the motor turns, the sign moves up or down. Consider using LEGO or Fischertechniks construction kits for these projects.

Solenoids

Solenoids are electromagnets that can be turned on to make a piece of metal move a short distance. When using solenoids, check the voltage rating and also make sure that the current the solenoid draws is not too large.

Beeps and Buzzers

Almost all computer systems have a "beep" sound that can be made when the computer wants to get the attention of the operator. Most calculators don't make sound, but you can add one, using the DCU. There are several different types of buzzers that you can use. Some buzzers need only to have a steady voltage applied and they will produce a sound. The sound continues until the circuit is turned off. Our program DCUTOGGL can be used for this type of buzzer. Other buzzers and simple speakers need to be pulsed; that is, the power to them must be turned on and off at a high frequency. The calculator subprogram DCUBUZZ is an example of this.

Mass Driver

An interesting project to try using electromagnets is to make a device to accelerate magnets. Start with a permanent magnet, such as a cow magnet. Find a clear plastic tube that the magnet will fit through. Wrap wire to make four or six electromagnets spaced along the tube. Connect the electromagnets to the digital out lines. Use a program to turn on the electromagnets in a quick sequence so that the magnet is pulled along. Timing is critical. The subroutine DCUMASS is an example of this kind of program.

You may also want to try using three electromagnets that can switch their polarity. Some commercial mass drivers operate this way.

Temperature-Controlled Environment

Combining sensors with control via the DCU allows you to experiment with feedback. Projects of this sort are often educational and interesting. Use a temperature probe connected to the LabPro/CBL 2. For heating air, you can use a small lamp. For heating a small amount of water, you can use a simple resistor. A fan can be wired to move air for cooling.

The program DCUTEMPC can be used for this type of control. It turns on a heater connected to D1 until the temperature reaches a specified reading. If the temperature exceeds another specified reading, it turns on a fan connected to D2.

Live Traps Activated with a Photogate

A fun project is to try to catch flies, bugs, or mice in a DCU-controlled live trap. The easiest way to do this is by using a photogate as a sensor to detect when the animal is in position to be captured. Photogates have an infrared beam which the animal blocks, sending a signal to the LabPro/CBL 2. When this happens, have the DCU turn on a motor or a stepper motor to move a door to catch the animal. We have used a guillotine-style door, a dropping box, or a motor that hits the lid of a hinged box to knock it into place. Be creative.

The program DCUTRAP2 is an example of a program that uses a DC motor to close a trap. The program DCUTRAP3 does a similar thing using a stepper motor to lower a door.

Alarms Controlled by Motion Detectors

Another interesting project is to set up a warning system using a Motion Detector. Motion Detectors have a range up to six meters when oriented carefully. The program DCUALARM is an example. We have used this for sounding a buzzer whenever someone walks in range of the motion detector. We have also used it outside (with the buzzer inside) to detect deer and alert us to look outside so we can see them. Note that it is not possible to use motion detectors in conjunction with the DCU on the CBL 2.

Activating Camera Shutters

Some cameras have connectors so that they can be activated by an electrical signal. To get photos of animals in the wild, you can use this idea to have the DCU trip the camera shutter when an animal is detected by either a photogate or a Motion Detector.

Stepper Motor Projects

Stepper motors are great for projects. You can control the exact position of the rotating part of the motor. There are many surplus stepper motors around, left over from disk drives and similar devices. Some stepper motors have as few as 48 steps in a complete rotation. Many come with built-in gear systems to provide hundreds or even thousands of steps per rotation. This also increases the torque of the motor.

Servo Motor Projects

Servo motors have a lot of torque, so they allow you to do things that require short motion with a lot of force. We have used them to lift the end of a dynamics track to start a cart rolling, to launch rubber bands, and to simulate an assembly-line sorting system.

A DCU-Controlled Robot Using DC Motors

One popular project is to build a robot with two wheels controlled by the DCU. A third wheel rotates freely to allow the robot to turn. You can have all of the following motions:

- Forward - Both motors moving forward
- Backward - Both motors moving backward
- Turn Right - Left motor rotating forward, right motor rotating backward
- Turn Left - Right motor rotating forward, left motor rotating backward

The DCU was partly designed around this project. There are output patterns for all of these motions using the D1,D2, D3, and D4 lines. This allows you to move the robot anywhere you want. Also, we wanted to have some other lines (D5 and D6) which could be used for other operations. For example, you could have the robot move around until it reaches a position, and then use D5 and D6 do something like sound a buzzer, raise a flag, pick up something, etc. The program DCUCAR is for this kind of robot.

A DCU-Controlled Robot Using Stepper Motors

Two stepper motors can also be used for the two wheels of the robot. In general, this gives you much better control of the motion of the robot, but it will probably be slower to move. The program DCUCARS is for use with this kind of robot. Note that to control two stepper motors with a single DCU, you must use stepper motor control ICs.

Sensor on a Robot for Feedback

Try combining a robot with sensors to provide added control over its motion. For example:

- Mount a Motion Detector on the robot so that it turns and tries a new direction when it gets within a specified distance of an obstacle.
- Build an edge detector that can tell that the robot has reached the table edge and tell it to stop and back up.
- Mount Magnetic Field Sensors on the robot so it can navigate using the earth's magnetic field.
- Mount two light sensors pointed at the tabletop so that the robot can sense whether the tabletop is white (reflecting a lot of light) or black (not reflecting much light). Program the robot so that it follows a black line drawn on the table, correcting its motion whenever it starts to move off the line.
- Mount a motion detector on the robot and write a program that has the robot move just up to an object and then stop.

Sun Seeker

Another interesting project using feedback is to build a light seeker. Mount two light sensors so that they point in slightly different directions on an apparatus that can be rotated. Write a program that compares the light level detected by the two light sensors and then rotate the apparatus toward the one which detects the most light. If this pattern continues, the light seeker should always end up pointed at the brightest light source. This is the basic concept used on some solar panels to track the sun.

Tea Maker

We have made a stepper motor driven automatic tea maker. The device consists of a tea bag, a temperature probe, and a conductivity probe mounted so that the stepper motor can raise or lower them into a tea cup. The program starts by lowering everything into a cup of hot water. The stepper motor gently raises and lowers the tea bag to help with the infusing. At the same time the program is monitoring the conductivity which increases as the tea dissolves. When the conductivity reaches a set limit, the apparatus is raised such that the tea bag is pulled out of the water but the temperature probe is still in the tea. The temperature is then monitored until it drops to a previously specified, perfect drinking temperature. Then the buzzer goes off and the apparatus is raised out of the way.

Appendix
A

Sources of Electronic Devices

Here are some sources of electronic devices, such as lamps, LEDs, motors, buzzers, solenoids, and stepper motors.

Vernier Robotics and Control Kit

Vernier Software & Technology sells a Robotics and Control Kit (order code RCK-DCU). It contains connection hardware and assorted components, such as: motors (DC, servo, and stepper), a fan, a speaker, lamps and holders, LEDs, and a buzzer for use with the DCU and LabPro analog output lines. The parts in this kit were selected to be easy to use with the DCU and LabPro analog output. It includes a manual describing 14 projects for student construction, programming and experimentation. Each project includes feedback and control extensions and other challenges.

Resources for Teachers Interested in Calculator-Controlled Robots

Math Machines
Fred Thomas and Bob Chaney have recently started a company making equipment to help teachers teach math. They make several interesting types of robots, laser pointer rotators, and other interesting devices. Check out http://mathmachines.net.

Electronic Component Sources

All Electronics
9005 S. Vermont Avenue
Los Angeles, CA 90006
800-826-5432
allcorp@allcorop.com
http://www.allcorp.com

(good for stepper motors, DC motors, and new or surplus items of all kinds)

Mouser Electronics
958 N. Main
Mansfield, TX 76063-4827
800-346-6873
http://www.mouser.com
(new electronic components, but no motors)

JameCo Electronic Components
1355 Shoreway Road
Belmont, CA 94002-4100
800-831-4242
http://www.jameco.com
(new electronic parts of all kinds, including motors)

Digi-Key
701 Brooks Ave. South
Thief River Falls, MN 56701-0677
800-344-4539
www.digikey.com
(new electronic components, but no motors)

Radio Shack
(stores everywhere)
800-843-7422
www.radioshack.com

Appendix
B

Calculator Programs and Subprograms

Here is a list of all the calculator programs and subprograms provided on the CD. We list calculator variables used by the programs. Many programs also use list L_6.

Program Name	All Variables Used in Calculator Programs	Description of Program	For stand-alone programs, a list of subprograms used	LabPro/CBL2 Notes
DCU Programs:				
DCU2	I	Turns on D1–D6 in succession on DCU connected to DIG/Sonic 1 and then turns on D1–D6 in succession on DIG/Sonic 2	DCUINIT	**LabPro only**
DCUALARM	V,S,D,T	Waits until Motion Detector detects an object closer than a specified distance. Turns on D1 for 10 seconds. The + key ends this program.	DCUINIT DCUWAITM(V) DCUPWRON(D,T) DCUOFF	**LabPro only**
DCUCAR	D,K,T	Allows you to control a car, driven by two DC motors, using the four arrow keys. T controls the time the motor is on for each keypress. The + key ends this program.	DCUINIT DCUWHEEL(D,T) DCUOFF	
DCUCARS	D,K,N	Allows you to control a car, driven by two stepper motors (and a stepper motor IC), using the four arrow keys. N controls the number of steps taken for each keypress. The + key ends this program.	DCUINIT DCUWHELS(D,N) DCUOFF	
DCUCOUNT		Counts 0–15 to show the resulting LED displays.	DCUINIT	
DCUMASS		Program to turn on D1, D2, D3, and D4 in order to accelerate a magnet through a tube (mass driver).	DCUINIT DCUOFF	

DCUSERVO	P = Pulse Width (ms) N = # of output patterns in 1 command J = loop counter L1 = pattern	Controls a servo motor connected to D1. This program works for many servo motors, including the one in the Vernier Robotic & Control kit.	DCUINIT DCUOFF	
DCUSTEP	T,D,N,I	Simple program that allows user to specify direction and number of steps (up to 12,000) for a directly-connected stepper motor (unipolar or bipolar). Change the value of T in the first line to control speed.	DCUSTEP1 (D,N) DCUINIT DCUOFF	
DCUSTEP3	T, D,N,I,P	Improved program that directly controls a stepper motor. This version is best to use if you plan to use the stepper motor for several different motions, one after the other. It keeps track of the stepper motor position as it moves.	DCUINIT DCUOFF	
DCUSUN	X, = tolerance S,T = inputs D = digital output K = Keypress	This program assumes that you have two auto-ID Vernier light sensors connected to CH1 and CH2 and a DC motor connected to the D1 and D2 output lines of the DCU. If the two light sensors are mounted pointing a few degrees apart and so they can be rotated by the motor, the program will turn the motor in the direction of the brightest light, so the light sensors follow a light source.	DCUINIT DCUOFF	
DCUTEMPC	W, V, S, D,K	Program that monitors temperature. Turns on heater (D1) if temperature is below minimum value (W) and turns on fan (D2) if temperature is above maximum value (V).	DCUINIT DCUOFF	

DCUTOGGL	L_1,L_2,L_3, I,A,N,B,W,V,K,T ,U	Program that allows the user to toggle the digital lines with the keypad. Note that it takes a second or two for this program to respond to a keypress. Also, not all possible combinations of outputs are possible, so sometimes the program will shut off lines before it turns on a line.	DCUINIT DCUOFF	
DCUTRAP2	D,T	Used in a photogate-triggered live trap. Monitors photogate to detect animal, then turns on D1 to take action, then pulses D2.	DCUINIT DCUCHKP DCUWAITP DCUPWRON(D,T) DCUPULSK(D)	
DCUTRAP3	D,N,T	Another live trap program, this one uses stepper motors. Monitors photogate connected to analog channel CH1 to detect animal, then activates a directly connected stepper motor to lower a trap door.	DCUINIT DCUSTEP1(D,N) DCUWAITP DCUCHKP DCUOFF	
DCUWARNV	D,T,V	Warns when the signal gets too high by turning on D1.	DCUINIT DCUWAITV(V) DCUPWRON(D,T) DCUOFF	
Analog Output:				
ANOUT	W = waveform F = frequency T = period A = amplitude (V) B = amplitude in units of 0.0024 V) S = offset	This is a simple program to show how you can control the analog output line of the LabPro and the function generator. Use this program if you want to examine the code to see how to control analog output.	DCUINIT DCUOFF	**LabPro only**
FUNCTGEN	A, K, L, M, O, P, Q, S, V, W	This program is a fairly complete function generator program. Use to set a DC voltage output, or set up a output waveform to test hardware connected to the analog output line of LabPro.	DCUINIT DCUOFF	**LabPro only**

Here is a list of all the subprograms provided on the disks.

Subprogram Name	All Variables Used in Calculator Programs	Description of Program	Variables that must be set.	LabPro/CBL2 Notes
DCUBUZZ (subprogram)	F,P,T,N, I	Produces an on/off signal on digital-out line 1 for a specified time and at a specified frequency. For use with speakers and simple buzzers.	F (Frequency) and T (Time) must be predefined. Also, F * T must be <= 512.	On CBL 2 or LabPro, the limit is 12,000 steps.
DCUCHKD (subprogram)	S	Checks and reports the status of a photogate connected to digital channel 2 (Blocked or Unblocked). The + key ends this program.		**LabPro Only**
DCUCHKP (subprogram)	S	Checks and reports the status of a photogate connected to the CH1 analog input (Blocked or Unblocked). The + key ends this program.		
DCUINIT (subprogram)	Z	Initialization subprogram for the CBL/LabPro and DCU. It makes sure that the link between the calculator and CBL/LabPro is OK and initializes the unit It also turns off all the DCU lines. This program should be called at the beginning of any main program.		
DCUOFF (subprogram)		Turns off all digital power lines. This subprogram should be called at the end of every main program.		
DCUPULSK (subprogram)	K,D, J	Turns the output lines specified by D on and off relatively slowly. The program continues until a key is pressed.	D (digital output pattern) must be predefined.	
DCUPWR2 (subprogram)	I,N,T, D	Turns on lines specified by D at half power for a specified time (maximum ~120 seconds).	T, D	
DCUPWR3 (subprogram)	I,N,T, D	Turns on lines specified by D at one-third power for a specified time (maximum ~120 seconds).	T, D	

DCUPWRON (subprogram)	T,D,I	Turns on line D for a time T. Forces calculator to wait for CBL 2.	T (Time) and D (digital output) must be predefined.	
DCUSTEP1 (subprogram)	D,N,I	Subprogram that directly controls a stepper motor.	D (Direction) and N (Number of Steps)	
DCUSTEP2 (subprogram)	D,N,I	Controls a stepper motor that uses an IC controller. Line D1 is pulsed to move a step and line D2 is used to set the direction of movement.	D (Direction) and N (Number of Steps) must be predefined.	
DCUWAITD (subprogram)	S	Waits for a photogate connected to digital channel 2 to become blocked.		**LabPro Only**
DCUWAITM (subprogram)	S,V,K	Waits for the motion detector to read an object closer than a specified distance. The + key ends this program.	V (Distance limit)	**LabPro Only (since CBL 2 only has one DIG/Sonic port)**
DCUWAITP (subprogram)	S	Waits for a photogate connected to analog channel CH1 to become blocked.		
DCUWAITV (subprogram)	V,S	Waits until the signal level on channel one exceeds a specified value.	V (Minimum Value) must be predefined	
DCUWHEEL (subprogram)	T,D,I	DC motor control program. Used to move DC motor-controlled car. Takes direction and steps as an argument and moves accordingly.	T (Time) and D (Direction; 1=forward, 2=backward, 3=left, 4=right).	
DCUWHELS (subprogram)	N,D,I, T	Stepper Control Program. Used to move a stepper driven car, Takes direction and number of steps as arguments and moves accordingly. Change the value of T in the first line to control the speed.	N (Steps) and D (Direction; 1=forward, 2=backward, 3=left, 4=right)	

Appendix
C

Selected DCU Calculator Program Lists

Here are some of the DCU programs listed for you to examine. The programs listed here are TI-83 versions, but the versions for other calculators are similar. To see versions of the same programs for other calculators, simply open the appropriate version of the program from the disk, using TI-GRAPH LINK. You can also use TI-GRAPH LINK to modify, print, and move these programs to your calculator.

Program: DCUSTEP

```
prgmDCUINIT
.1→T
ClrHome
Lbl A
Disp "T",T
Disp "ENTER DIRECTION"
Disp "0 OR 1"
Prompt D
Disp "NO. OF STEPS?"
Prompt N
ClrHome
prgmDCUSTEP1
Goto A
prgmDCUOFF
```

Program: DCUSTEP1

```
Lbl A
If D=0
Then
{1,31,4,5,9,10,6}→L6
Else
{1,31,4,6,10,9,5}→L6
End
Send(L6)
{1,1,14}→L6
Send(L6)
Disp "DIRECTION",D
Disp "STEPS",N
{3,T,N,0}→L6
Send(L6)
Get(I)
{1,31,1,0}→L6
Send(L6)
{3,.1,1,0}→L6
Send(L6)
Get(I)
```

Program: DCUMASS

```
prgmDCUINIT
{1,1,14}→L6
Send(L6)
{1,31,5,1,2,4,8,0}→L6
Send(L6)
Disp "FIRE"
{3,.12,5,0}→L6
Send(L6)
Get(I)
prgmDCUOFF
```

Program: DCUWARNV

```
prgmDCUINIT
ClrHome
Disp "SET LIMIT"
Prompt V
prgmDCUWAITV
Disp "LEVEL"
Disp "EXCEEDED"
Disp "OUTPUT 1 ON"
1→D
5→T
prgmDCUPWRON
prgmDCUOFF
```

Program: DCUALARM

```
prgmDCUINIT
Disp "ENTER DISTANCE"
Disp "LIMIT"
Input V
ClrHome
10→S
Disp "WAITING FOR "
Disp "DISTANCE"
Disp "TO BE LESS"
Disp "THAN"
Disp V
ClrHome
Output(1,1,"PRESS + TO QUIT")
prgmDCUWAITM
If S≠0
Then
1→D
10→T
ClrHome
prgmDCUPWRON
End
ClrHome
prgmDCUOFF
```

Program: DCUCAR

```
ClrHome
1→T
prgmDCUINIT
Disp "READY FOR ACTION"
Disp "USE ARROWS"
Disp "FOR MOTION"
Disp "PRESS [+] TO"
Disp "QUIT"
Lbl A
0→D
getKey→K
If K=25
1→D
If K=34
2→D
If K=24
3→D
If K=26
4→D
If K=95
Goto Z
If D=0
Goto A
prgmDCUWHEEL
Goto A
Lbl Z
prgmDCUOFF
```

Program: DCUWAITM

```
{1,12,2}→L6
Send(L6)
6→S
Send(L6)
Output(2,1,"WAITING FOR")
Output(3,1,"DISTANCE")
Output(4,1,"TO BE LESS")
Output(5,1,"THAN")
Output(5,6,V)
{3,1,-1,0}→L6
Send(L6)
While S>V
Get(S)
Output(7,1,S)
getKey→K
If K=95
0→S
End
```

Program: DCUWHEEL

```
{1,1,14}→L6
Send(L6)
Disp "DIRECTION"
If D=1
Then
Disp "FORWARD"
{1,31,2,5,0}→L6
Goto A
End
If D=2
Then
Disp "BACKWARD"
{1,31,2,10,0}→L6
Goto A
End
If D=3
Then
Disp "LEFT"
{1,31,2,6,0}→L6
Goto A
End
If D=4
Then
Disp "RIGHT"
{1,31,2,9,0}→L6
Lbl A
Send(L6)
Disp T
Disp "SECONDS"
{3,T,2,0}→L6
Send(L6)
Get(I)
```