



IC Library Manager

OPERATOR'S MANUAL



ABI Electronics Limited
Dodworth Business Park
Barnsley
South Yorkshire
S75 3SP
www.abielelectronics.co.uk

CONTENTS

1.	Preface.....	1
1.1.	How to use this manual.....	1
1.2.	Precautions	1
1.2.1.	Host PC.....	1
1.2.2.	Data storage	1
1.2.3.	Programme debugging	1
1.3.	Maintenance.....	2
1.3.1.	Software	2
1.4.	Contacting ABI Electronics Ltd.....	2
1.5.	Copyright and disclaimers.....	2
2.	Introduction	3
2.1.	What is CompactLink?	3
3.	Getting started	4
3.1.	Checklist.....	4
3.2.	System requirements	4
3.3.	Installing security dongle.....	4
3.4.	Installing CompactLink	5
3.5.	Running CompactLink.....	5
3.6.	IC library data structure.....	5
3.7.	Theory of CompactLink operation	6
3.8.	Reviewing the IC library	7
3.9.	Viewing an IC	7
3.10.	Copying/editing an IC.....	8
3.11.	Adding an IC to the USER library.....	8
3.12.	Specifying a functional test for the IC.....	9
3.13.	Developing a functional test.....	11
3.14.	Deleting an IC from the USER library.....	11
3.15.	Printing or exporting a device.....	12
3.16.	Generating library files	12
4.	Writing your own test programmes	15
4.1.	Introduction to PLIP	15
4.2.	Opening the test development and debugging window.....	15
4.3.	Entering and compiling a programme	17
4.4.	Fixing the errors and warnings	17
4.5.	Getting help.....	18
4.6.	Documenting your programme.....	18
4.7.	Connecting to hardware	18
4.7.1.	Connecting to SYSTEM 8 modules.....	19
4.7.2.	Connecting to Compact Professional USB products	20
4.7.3.	Connecting to Compact products with serial cable	20
4.8.	Debugging your programme	21
4.9.	Setting breakpoints	22
4.10.	Debugging techniques	23
4.10.1.	Compiler errors	23
4.10.2.	Run time errors	23
4.10.3.	Logical errors	24

5.	Some common programming concepts	27
5.1.	Digital test programming	27
5.1.1.	Combinational devices – gates, buffers, multiplexers	28
5.1.2.	Sequential devices – counters, registers, latches	28
5.1.3.	Tri-state devices – buffers, bus drivers	29
5.1.4.	LSI and complex devices	29
5.2.	Analogue test programming	30
5.2.1.	Using the DRIVE commands	31
5.2.2.	Using the RESTRICT command	32
5.2.3.	Using parameters	33
5.2.4.	Using the SOURCE command	34
5.3.	Automatic circuit compensation	34
5.3.1.	Compensation by splitting	35
5.3.2.	Compensation by skipping	35
5.3.3.	Compensation by adapting	37
5.3.4.	Compensation by trying	37
6.	Example of a 7400 digital IC test programme for the BFL/CMC	39
6.1.	Defining the IC inputs	39
6.2.	Simple test for a logic NAND gate	40
6.3.	Logic NAND gate test with BFL circuit compensation	40
6.4.	Improved logic NAND gate test with BFL circuit compensation and looping	41
6.5.	Complete programme for logic NAND gate	42
7.	Example of a diode analogue test programme for the AICT	46
8.	Example of an operational amplifier analogue test programme for the LMC	51
9.	<i>PLIP</i> command and function reference	56
9.1.	Introduction	56
10.	Troubleshooting and support	58
11.	Appendices	59
11.1.	Library parameter reference	59
11.2.	CompactLink error/warning messages	61
11.3.	PLIP error messages	63
11.4.	PLIP warning messages	68
11.5.	PLIP run time error messages	69
12.	Index	71

1. Preface

Thank you for purchasing the ABI **CompactLink** IC Library Development Manager software. Please refer to this manual before attempting to install or use the software.

1.1. How to use this manual

This manual is divided into sections describing all aspects of **CompactLink** operation. There is a getting started guide, designed to get you up and working quickly, followed by more detailed instructions on the various functions. We recommend you read at least sections 1 to 3 before using **CompactLink** for the first time.

This manual is written on the assumption that you are already familiar with the SYSTEM 8, BoardMaster and Compact products from ABI. Please refer to the help text and/or manuals for those products if you require further information.

For the latest product information, including an up to date copy of this manual and the latest software, please visit www.abielectronics.co.uk.



This symbol is used where the information given is important to prevent damage to your system or board under test.

1.2. Precautions

1.2.1. Host PC

The **CompactLink** software is designed for use on a PC, laptop or tablet running Microsoft Windows software from XP to Windows 10, and can also be used on the BoardMaster 8000 range of products from ABI. Operation on any other type of PC or with any other operating system is not supported.

1.2.2. Data storage

The **CompactLink** software uses a local database for storing IC details and test programmes. Any new ICs you add to the system are stored in the file "CompactLinkICLibrary.dat" in an application data folder, which is accessible from the Windows Start menu. It is important that this file is backed up regularly to ensure that user devices and test programmes are not lost in the event of hardware failure.

1.2.3. Programme debugging

When testing ICs with the SYSTEM 8 BFL, ATM or AICT modules, backdriving (refer to the SYSTEM 8 or BoardMaster help for an explanation of backdriving) is used to isolate the inputs of the IC under test from the surrounding components.



*When single stepping a **PLIP** test using the **CompactLink** debugger, this may result in extended backdriving times which could cause damage to the BFL/AICT module and/or the board under test. We recommend using an IC in the out of circuit adapter or in an unconnected, powered socket for IC test programme development.*

1.3. Maintenance

1.3.1. Software

The **CompactLink** software is not warranted as being fit for any particular purpose, although ABI will make every effort to ensure that it is suitable for use in conjunction with ABI SYSTEM 8, BoardMaster and Compact products for developing new IC tests. Updated versions of the software are available on our website (www.abielelectronics.co.uk) free of charge. ABI however reserves the right to charge for completely new versions of the software.

1.4. Contacting ABI Electronics Ltd

ABI Electronics Ltd
Dodworth Business Park
Barnsley
South Yorkshire
S75 3SP
United Kingdom

Email: sales@abielelectronics.co.uk
Website: www.abielelectronics.co.uk
Telephone: +44 (0)1226 207420

1.5. Copyright and disclaimers

This manual copyright © 2006-2016 ABI Electronics Ltd. All rights reserved. First published September 2006.

You may make electronic or paper copies of this manual solely for use in conjunction with operating the software as a bona fide customer, but not for any other purpose.

Windows® and Microsoft® are registered trademarks of Microsoft Corporation.

ABI Electronics Ltd reserves the right to make product improvements and/or changes at any time without prior notice, including changes to the software specifications. This manual may therefore not necessarily reflect current software specifications. An updated copy is available for free download on our website www.abielelectronics.co.uk

Whilst ABI makes every effort to ensure the accuracy of this manual, we will not accept liability for damages incurred directly or indirectly from errors, omissions in this manual, or discrepancies between the manual and the **CompactLink** software itself.

2. Introduction

Congratulations on your decision to purchase the **CompactLink** IC Library Manager software.

2.1. What is CompactLink?

CompactLink IC Library Development Manager is designed to allow you to add new ICs, with or without a functional test, to the library of your ABI SYSTEM 8, BoardMaster, ChipMaster Compact or LinearMaster Compact.

The heart of **CompactLink** is **PLIP - PremierLink** IC Programme. **PLIP** is a high-level descriptive test programming language optimised for generation of both analogue and digital IC test programmes. Programmes are compiled into machine code, making them fast and compact, and can be freely added to the SYSTEM 8 or Compact libraries. **CompactLink** contains a sophisticated test programme debugger which allows you to check that your programme executes correctly before including it in your SYSTEM 8 or Compact library.

Note that there are two versions of the software, **PremierLink** and **CompactLink**. If you are running the **CompactLink** version, some of the device information relating to the SYSTEM 8 BFL, ATM and AICT products will not be visible.

It is very important to understand that there are two separate functions involved in adding an IC to the user library. Firstly, the IC number, size, pin-out, power supply pins and other information must be defined which will be described in detail later. After this process is complete, the IC can be included in your user library, and you can perform (depending on the target product) the CONNECTIONS, VOLTAGE, THERMAL and DIGITAL V-I tests on it. However, no functional test is possible unless you write a functional test programme, which is the second main function of **CompactLink**.

3. Getting started

This section is intended to get **CompactLink** up and running quickly. Please read carefully before using for the first time. Detailed descriptions of the meaning of the various parameters and device entries will be given later on in the manual

3.1. Checklist

The following items are included in the **CompactLink** package: -

- **CompactLink** CD
- **CompactLink** USB security dongle
- Operator's manual

The package may also include a serial adapter/cable and/or FLASH IC(s) for first time updating of older ChipMaster or LinearMaster Compact products.

3.2. System requirements

The **CompactLink** software should be installed on a PC, laptop or tablet with the following minimum specification: -

Intel CPU 1GHz or equivalent, 128M RAM, 500MB hard drive space, free USB port, Microsoft Windows operating system (from XP to Windows 10)

For debugging and library updating the PC requirements depend on the type of product you are using. Your PC requires a connection to the SYSTEM 8 BFL/ATM/AICT module, or to the ChipMaster/LinearMaster Compact, to allow test programmes to be developed and added to the library. Depending on the type of product you wish to use, the following will be required: -

Product	Connection
SYSTEM 8 BFL/AICT module in external case	USB cable
SYSTEM 8 BFL/AICT module built into PC	PCI card
SYSTEM 8 ATM module	USB cable
BoardMaster 8000 Plus	PCI card, already fitted
ChipMaster/LinearMaster Compact Professional	USB cable
ChipMaster/LinearMaster Compact (older RS232 versions)	COM port on your PC, or USB to RS232 adapter and serial cable

3.3. Installing security dongle

Before the **CompactLink** software can be used, the USB security dongle must be installed. The drivers are included on the **CompactLink** installation CD and must be installed as follows:

- Ensure you are running Windows on an account with Administrator privileges.

- Insert the **CompactLink** CD in the CD or DVD drive.
- Click “Start, Run, Browse” on your PC and navigate to your CD ROM drive (usually drive D) with the **CompactLink** CD inserted.
- Select the file “CBUSetup.exe” and click “Open”, then click “OK” to start the installation.
- Follow the instruction on the screen.

Once the installation has completed, the USB security token can be inserted into any available USB socket on your computer.

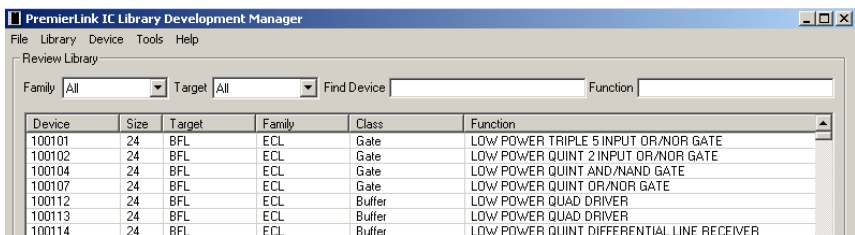
3.4. Installing CompactLink

To install the **CompactLink** software, follow this step by step procedure: -

- Insert the **CompactLink** CD in your CD or DVD drive.
- Click “Start, Run, Browse” on your PC and navigate to your CD ROM drive (usually drive D) with the **CompactLink** CD inserted.
- Select the file “setup.exe” and click “Open”, then click “OK” to start the installation.
- Follow the installation instructions on the screen. We recommend that all options are left at their default values.

3.5. Running CompactLink

To launch the software, click **Start/Programs/CompactLink/CompactLink** on your PC. You can also create a desktop shortcut if you wish to make starting easier. The opening screen (example below) shows the **Review Library** screen displaying a list of devices in the current library and provides a menu to access all software functions.



3.6. IC library data structure

Each device in the library is structured as follows: -

- General Device Information
 - BFL module test information
 - AICT module test information
 - ATM module test information
 - ChipMaster Compact test information

- LinearMaster Compact test information

As shown, the device data is organised into Device Information and Target Information. Each device can have up to five targets, each containing target specific information for the five products supported by **CompactLink** (BFL, AICT, ATM, ChipMaster Compact and LinearMaster Compact). Note that not all devices can be tested on all targets – for example the diode and transistor tests are only applicable for the AICT product.

The entire library of installed ICs is divided into FAMILIES, to make the management of them easier. The current FAMILIES are: -

USER, 74xx, 4xxx, MEMORY, INTERFACE, LSI, MICROS, PAL/PLD, ECL, LINEAR, GENERIC, ANALOGUE, DISCRETE, DATA CONV, OPTO, CONNECTOR, PACKAGE, BUS, INTERCON, PROBE, ALL

The GENERIC, ANALOGUE and DISCRETE libraries contain analogue IC tests and discrete component tests for use with the SYSTEM 8 AICT module. The other libraries contain digital IC tests and are for use with one or more SYSTEM 8 BFL modules. Note that the LINEAR library is designed for use with the BFL module, and is included for backward compatibility with earlier versions of the SYSTEM 8 software.

Note that for Compact products, the devices in the internal library provided with the product are not included in the **CompactLink** database but can still be accessed by entering their numbers in the usual way on the Compact keypad. See the Compact manual(s) for a full device listing.

A full list of all device information entries with their meanings is given in appendix 11.1

3.7. Theory of CompactLink operation

The following steps are required to add an IC to the library for use on the SYSTEM 8 BFL or AICT modules, or with either Compact product. This is a summary – full details are given later in the manual: -

- Add a new device to your USER library using **CompactLink**
- Fill in the device information from the device data sheet
- Enable the target product(s) with which you wish to test the device
- Fill in the target information for the device/target combination
- If functional testing is required: -
 - Enable the functional test for the chosen target product
 - Choose a test for the device, or develop a new test if no suitable test is available
- Generate the USER library files for the target product group (SYSTEM 8 or Compact)
- Copy the generated library files to your SYSTEM 8 folder, or download them to the Compact product
- The added device is now available in your SYSTEM 8 or Compact device list

3.8. Reviewing the IC library

The standard software on the installation CD includes library and test data for all ICs in the current ABI library. These ICs cannot be modified but are available for viewing or copying. The built in library is subdivided into **Families** (74xx, 4xxx, Memory, LSI etc) and **Targets** (BFL, AICT, ATM, ChipMaster, LinearMaster).

By default the entire library is shown, but you can restrict the display to a particular **Family** and/or **Target** by using the combo boxes. If you want to find a particular device, enter its number in the **Find Device** box. You can also filter the list by entering text in the **Function** box – the list will be filtered to show only those entries containing the entered text.

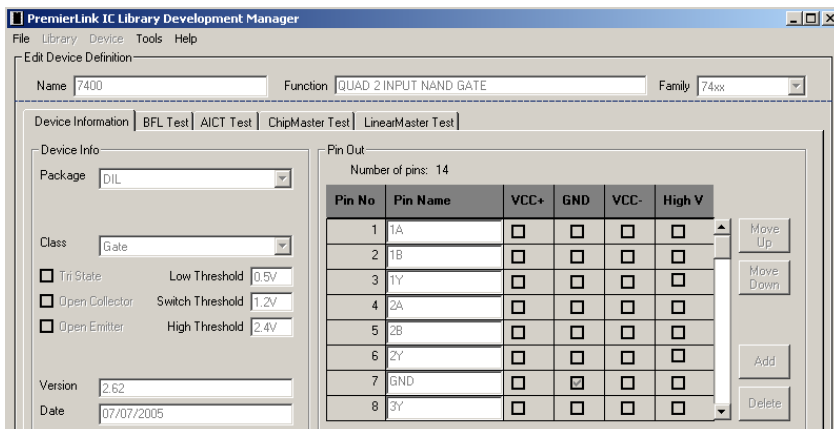
You can also sort the list of devices by clicking on any of the column headings in the **Library Review** display. If you sort on the **Device** column, by default the full device name is used to sort the entries. However, if you click **Tools/Options** you can turn on **Intelligent Sort**, which uses the numeric part of the device name only to produce a more logical list of devices.

3.9. Viewing an IC

There are several ways to select a device for viewing: -

- Click on a device in the list and click **Device/Edit** on the menu
- Right click on a device in the list and choose **Edit** from the popup menu
- Double click on a device in the list
- Enter all or part of the device name in the **Find Device** box, then press the **Enter** key

As an example, set the **Family** and **Target** selections to **All** and clear the **Function** box to display the entire library. Type 7400 in the **Find Device** box and press **Enter**. The 7400 device will then be opened for viewing as shown below (on the **CompactLink** version of the software, the **BFL Test** and **AICT Test** tabs are not visible): -



In this **Edit Device Definition** screen you can see the name, function and family of the 7400 device, along with 5 tabs for further device information.

In the **Device Information** tab you can see the pin out and device specific information such as the package, thresholds and output types. Note that since the 7400 device is part of the built in library, you cannot edit any of these entries.

In the **BFL Test** tab you can see various entries which apply to the 7400 device when testing on the BFL product, including the types of test which are enabled, voltage levels, several library options and the functional test used for this device. Notice that the 7400 device has the **Include device in BFL library** checkbox set for the BFL target product.

The other test tabs contain similar information but the 7400 device is not currently specified for testing on these products.

3.10. Copying/editing an IC

You cannot directly edit any of the devices in the standard library, but you can copy one of the standard devices into the USER library, which is then available for you to edit. As an example of this, carry out the following steps: -

- Return to the **Library Review** screen by clicking **Cancel** if you are still looking at the 7400 device
- Find the 7400 device again by typing into the **Find Device** box
- Choose **Device/Copy** from the menu or right click and select **Copy**
- Enter a new name (e.g. 7400BFL or 7400CMC) for the **New Device Name** and click **OK**

The new 7400BFL/7400CMC device will then be added to your USER library. Locate it in the library list and double click on it to edit. You can now change the various entries for the device without affecting the original 7400 device in the standard library.

On both the ChipMaster and LinearMaster Compact products there is only a numeric keypad available for entering device numbers. Therefore, on the ChipMaster and LinearMaster edit tabs there is a field (**Use Number**) provided for a numeric part number. For example, if the full part number for a new device is LM339N, you may wish to enter the number 339 in the **Use Number** field. The ChipMaster Compact and LinearMaster Compacts contain an internal library of IC tests which is not visible in the **CompactLink** software. When deciding which test to execute, the Compact will give priority to the user library if a device with the same number exists. In the above example the 339 new user test for the LM339N will be executed rather than the built-in LM339 test – if you wish to have both tests available use a different number (e.g. 3390) for the user test.

3.11. Adding an IC to the USER library

To add a new IC to the library, choose **Device/Add** from the menu. You will see the usual **Edit Device Definition** screen but this time with blank or default entries. On the **Device Information** tab, fill in the Name and Function boxes and enter

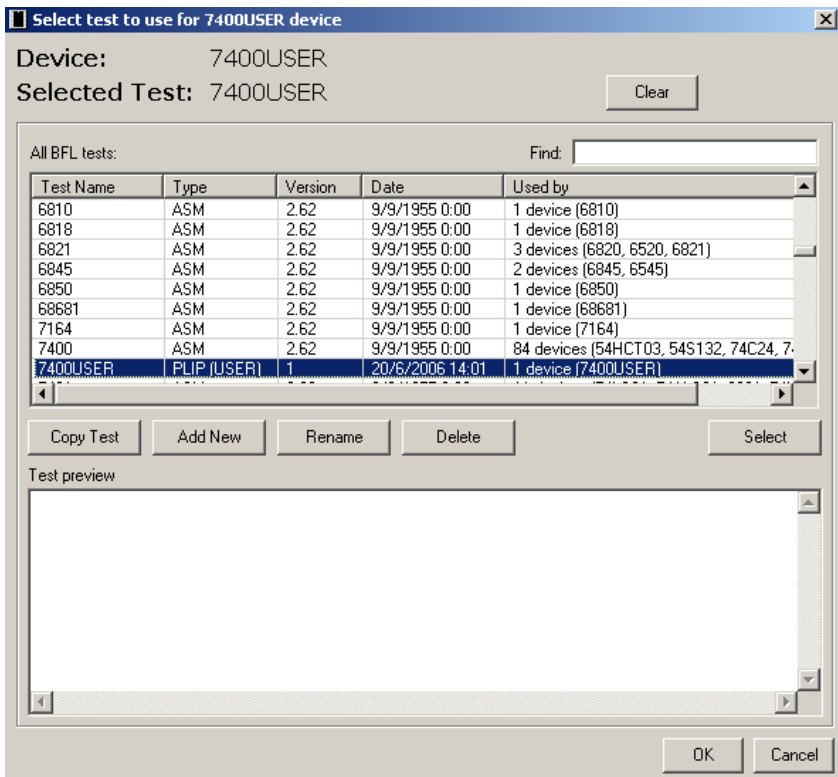
suitable values for the other options. The new device defaults to 14 pins, so you may wish to **Add** or **Delete** pins with the buttons if the device you are adding has a different number of pins. To change the pin name, click on the pin and enter the new name (maximum 8 characters)

Now you need to choose the product(s) with which the new device is to be tested. For example, if you intend to add this device to the BFL library, open the **BFL Test** tab and click **Include device in BFL library**. You can then set the other BFL specific entries as you wish. You must enable at least 1 of the test types (**Connections**, **Voltage**, **V-I** and **Functional**).

3.12. Specifying a functional test for the IC

If you have enabled **Functional** testing for your newly added device, you have to specify which functional test to use. The actual functional tests are stored separately from the devices since several devices with comparable functions and pin-outs will usually share one test. When you enable Functional testing, you will see in the **Functional Test Configuration** area that the **Current Test** entry is blank, showing that as yet no test has been selected for the device.

To select a test, click the **Select Test** button to show the **Select Test** window as shown below: -



The **Select Test** window displays a list of all tests present in the system. There are currently 2 types of test available: -

- **Assembler** language tests. These are reserved for the SYSTEM 8 BFL standard library only and cannot be edited or debugged, nor can they be used with the AICT or Compact products. However, if your device is exactly function and pin compatible with a device in the standard library you can specify an assembler language test if you wish.
- **PLIP (PremierLink IC Programme)** tests. **PLIP** is the **CompactLink** built in test programming language and allows you to develop and debug a test using the integrated debugger, which will be described in detail later in the manual.

If your new device is pin and functional compatible with an existing device it can probably make use of a test already in the list. For example, the 7400 test for the BFL is used by over 80 different devices, which are all pin compatible with the 7400. If you want to specify an existing test, find the test (you can use the **Find** box to quickly locate a test) and click **Select** to associate this test with your new device.

Often you will want to add a new test for a new device. To do this, there are 2 alternatives: -

- If there is a **PLIP** test in the library that is similar to the one you want, you can use Copy Test to make a copy of it with a new name. To do this click **Copy Test**, select the copied test and click **Rename** to give the test your desired name.
- If you want a complete new test, click **Add New** to add a new test, select it then click **Rename** to change the name.

In all cases, the preview window displays the source code for the selected test to help you choose a suitable test for your device.

To delete a test, select the test and click **Delete**, but note that you cannot delete a test from the standard library nor can you delete a test that is allocated to a device.

3.13. Developing a functional test

Test programming and debugging is a complex subject, which we will discuss in detail later. However, to see the required steps, follow the procedure below: -

- If you have not already done so, add a new device to your USER library as described above
- Define the pin names for the new device, since they are required for the functional test
- Enable the device for the chosen target product as described above
- Enable the functional test for the device/target combination
- Click **Select Test** and add a new functional test for the device
- Click **Develop Test** to enter the functional test development and debugging window
- In the **Source Programme** window, enter the **PLIP** code for the test (full details later)
- Use the toolbar **Build Test** button to compile the test and fix any compilation errors
- Connect up your chosen hardware (SYSTEM BFL/AICT modules, or either Compact product)
- Choose **Tools/Configure Hardware** and specify the type of hardware interface to be used
- Use the toolbar **Send Test** button to download the compiled test to the target hardware
- Use the debugging commands and windows to step through the test and confirm that it executes correctly
- Once complete, save the test and close the debugging window
- Generate a new set of USER library files containing the new test

3.14. Deleting an IC from the USER library

There are two ways to delete a device from your USER library: -

- Select the device to be deleted in the **Library Review** screen by clicking on it or by entering all or part of its name in the **Find** box
- Right click on the device and choose **Delete**, or
- Choose **Device/Delete** from the menu

Once you have deleted a device, there is no way to restore it except by re-entering all its information. You should make regular backups of the database file (CompactLinkICLibrary.dat) to ensure you do not lose wanted data.

Note that you cannot delete a device from the standard built-in library.

3.15. *Printing or exporting a device*

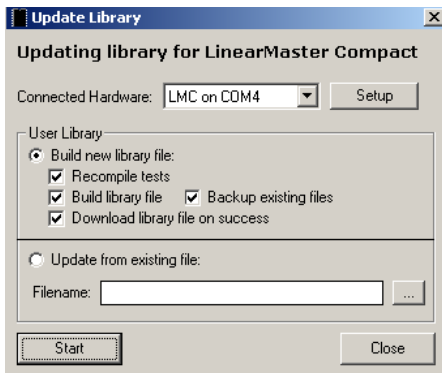
If you wish to have a hard copy record of a device you can create a report and then either print the report or save it to a text file. To do this, select the device in the **Library Review** screen and select **Device/Print/Export** from the menu, or alternatively right click on the device and choose **Print/Export**. In the report window which appears, you then have the choice between **Export**, which saves the device details as a text file, and **Print**, which sends the details to your printer.

3.16. *Generating library files*

Once you have completed adding devices and/or tests, you will probably want to generate a set of library files to include the new device in the library of your SYSTEM 8 or Compact product. There are 3 options which can be selected from the **Library** menu depending on your version: -

- Update SYSTEM 8 User Library. This produces a set of user library files in the correct format for use with your SYSTEM 8 Premier software.
- Update CMC User Library. This produces a ChipMaster Compact user library file (CML file) which can be downloaded and programmed into your ChipMaster Compact over the serial cable.
- Update LMC User Library. This produces a LinearMaster Compact user library file (LML file) which can be downloaded and programmed into your LinearMaster Compact over the serial cable.

As an example, assume we are going to generate a user library file for the LinearMaster Compact. Choose **Library/Update LMC User Library** from the menu and the **Update Library** window appears as follows (assuming the LinearMaster is actually connected to the COM4 port as shown): -



The process has 4 stages enabled by the 4 check boxes as follows: -

- Firstly, you must compile all your user tests. The integrated debugger uses a different format for compiled tests to allow easy debugging, so all tests need to be recompiled at this stage. Select the **Recompile Tests** check box to enable the compiler to compile all LinearMaster Compact tests.
- Enable the **Build library file** check box to generate the library file for the selected product. In this case the file LMCLIB.LML will be generated in the folder specified by the **Tools/Options** menu function.
- If you want to create a backup of any existing user library files, enable the **Backup existing files** check box. A backup folder will be created and the existing library file(s) will be copied to it before being overwritten with the new file(s).
- If you have a Compact product connected (as in this example – for more details see the Compact manual and section 4.7), enable the **Download library file on success** check box. This will cause the generated file(s) to be automatically downloaded to the Compact provided the compilation and file generation processes executed correctly. With SYSTEM 8 this stage is not required as the library files are already created in the correct location.
- If you have previously generated a Compact library file using the above procedure, you can enable the **Update from existing file** check box to skip the compilation and building stages and just download the previously generated file.

Once you have done this you can disconnect the Compact from **CompactLink** and the ICs will then be available in the user library. If you run your SYSTEM 8 software the new devices will appear in the user library for your chosen target product (BFL or AICT or both).

Note: The format of the LinearMaster Compact USER library (LML) file was changed from version 2.07 onwards. If you have a LinearMaster with an old USER library, you should do the following to upgrade your library: -

- Make sure you are using **CompactLink** version 1.16 or greater. Upgrade the software if required before proceeding.

- On the LinearMaster Compact, make sure you are using version 2.07 or greater. Upgrade the LinearMaster if required.
- Rebuild your USER library file as described above and send to the LinearMaster.

4. Writing your own test programmes

4.1. Introduction to PLIP

Once you have added a device to the library you can perform (on the SYSTEM 8 BFL) connections, voltage and V-I tests on it. However, to get the best out of your system you can also add a functional test to the device to perform a truth table or analogue (depending on the target product and the device type) test on the device. Functional tests for all target products are written in a high-level programming language called **PLIP** (**PremierLink** IC Programme).

The **PLIP** test programming language is a high-level language designed specifically for test programming. The syntax is highly descriptive, so that programmes are to a large extent self-commenting, but of course comments can be inserted if required. IC pins are referred to by their names (as defined by the IC device information) to avoid continual reference to the pin-out during programming, and to make the programmes more readable. In addition, related sets of pins can be defined as a pin group, which can then be referred to by its group name. This again greatly improves the readability and understanding of a test programme. See the SET command for further details of this facility.

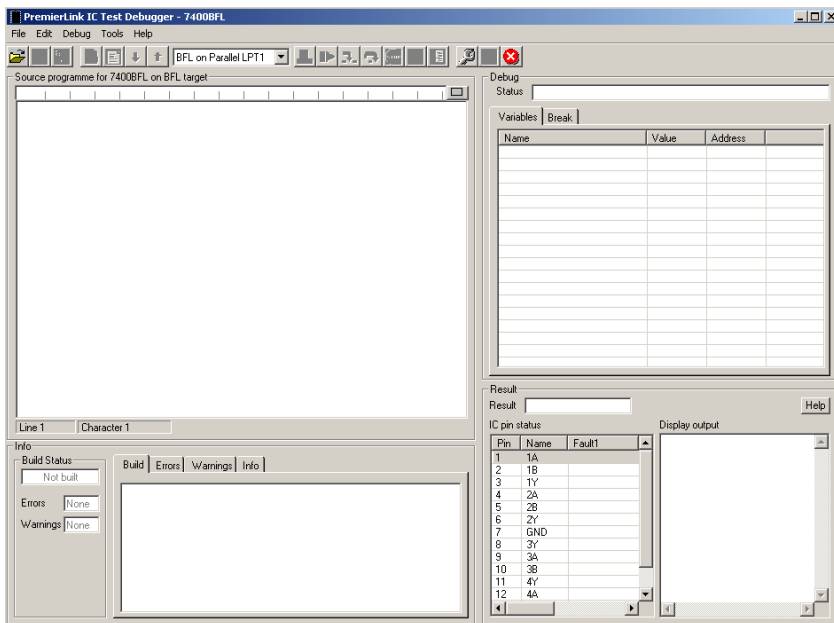
The compiler generates binary data which can be executed in stand alone form by the integral debugger, or combined into library files for use with SYSTEM 8 or Compact products. The SYSTEM 8 and Compact software contains advanced run time error checking traps to ensure that execution errors (e.g. divide by zero, stack overflow, out of range voltages etc) do not cause system crashes.

Any **PLIP** programme contains a combination of COMMANDS, FUNCTIONS and VARIABLES, the meaning of which will become clear if you work through the development example for the 7400 IC given below.


4.2. Opening the test development and debugging window


The first step in writing any test programme is to add a new IC to the library and add a new test for it, as follows: -



- Refer to sections 3.10 to 3.13 and add a 7400BFL (or 7400CMC if working with the ChipMaster Compact) device to the library if not already present.
- Make the sure the pin names for the device are correct (**1A, 1B, 1Y, 2A, 2B, 2Y, GND, 3Y, 3A, 3B, 4Y, 4A, 4B ,VCC**) .
- Enable the **Functional Test** for the BFL or ChipMaster target product.
- Click **Select Test** to open the **Select Test** window.
- Click **Add New** to insert a new, blank test and change its name to 7400BFL or 7400CMC.
- Click **OK** to close the **Select Test** window.
- In the **Edit Device** window, click **Develop Test** to open the test development and debugging window.



The **Source Programme for 7400BFL** window (on the left) is where the test programme code is entered. If you wish you can click the **Full Screen** button at the top right of the **Source Programme** window to expand it to simplify entry of IC test programmes. At the top of the window there is a ruler bar showing the tab stop positions – you can use tabs in your programme to make it more readable and to indent code inside procedures. You can change the tab positions by choosing

Tools/Options/Formatting from the menu or  **Options** from the toolbar, and entering a new **Tab width** value from 2 to 8. While entering the test programme,

you can click the  **Save Test** button on the toolbar to save the current programme in the database. You can also write the programme to a text file, or


read in a text file, using the  **Write Test** and  **Load Test** buttons on the debug toolbar. These functions allow you, if you wish, to write programmes in text format using an external text editor before reading them into the **CompactLink** debugger.

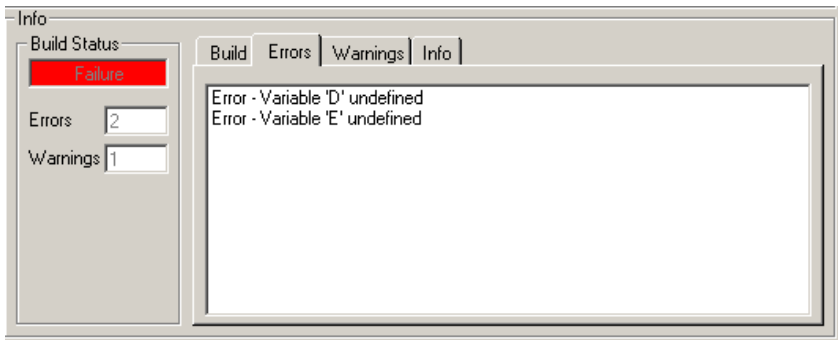
The debug window also contains menu commands and toolbar buttons for compiling and downloading the test, executing and stepping programmes, setting breakpoints and watch values. The meaning of these will become clear as you work through the example 7400 test below.

4.3. Entering and compiling a programme

Before starting to work on a real programme, we will first have a look at the operation of the editing and compilation system used for **PLIP** programs. In the **Source Programme** window, type in the following **PLIP** programme: -



```
A = 0
B = A - D
C = A + E
```

Compile the programme by clicking the  **Build Test** button and observe the results. At the bottom left in the Info window, you will see that the compilation failed with 2 errors and 1 warning as follows



4.4. Fixing the errors and warnings

This is a simple programme using variables and expressions. The variables **A**, **B** and **C** are defined, but the variables **D** and **E** have not been defined yet are used in expressions. This is an error as the compiler shows. **CompactLink** allows you to quickly find the errors in your programme – click on the error message in the **Errors** tab in the **Info** window and observe that the line in the **Source Programme**

window containing the error is highlighted. You can also use the   **Next Problem/Previous Problem** buttons to locate the errors in your programme.

The programme also has a warning that you have not included an **END TEST** command – this is required for every **PLIP** programme because the end of the programme may not necessarily be at the end of the text if procedures are defined later in your programme.

To fix the errors and warnings, amend the programme as follows and recompile: -

```
A = 0
D = 1
E = 2
B = A - D
```

```
C = A + E
END TEST
```

You should now have a result with no errors and no warnings.

4.5. Getting help

CompactLink contains extensive on-line syntax help for **PLIP** programmes. To access this, right click on the programme text in the **Source Programme** window and select **Syntax Help** from the popup menu. **CompactLink** will attempt to find help for the word you have clicked on and will display the correct syntax with examples. When the **PLIP Syntax Guide** is open, you can choose other commands from the combo boxes at the top to learn about all the **PLIP** statements.

Note that if the chosen word in the programme has more than one context, you will be given a list of alternatives to choose from. Help is not available for comment lines or unrecognised words. You can also press **F1** or choose **Help/Syntax** from the menu.

4.6. Documenting your programme

Although **PLIP** is a very readable language, it is not always clear what the intention of the programme is. This is especially true for someone who has not written the programme but has to update or modify it. To help with this you can add comments to your programme to explain, in your own language, what the programme is designed to do. To add a comment, type a ***** character followed by a description of the programme function. The above simple programme could be commented as follows: -

```
* Sample program to show variable definition and
expression use
* Define variables A, D, E and initialise
A = 0
D = 1
E = 2
* Define variables B and C and initialise with
expressions
B = A - D
C = A + E
* Tell PLIP this is the end of the test
END TEST
```

You can also include blank lines to separate out blocks of code to further improve readability.

4.7. Connecting to hardware

Up to now we have used the **CompactLink** software alone with no connection to any form of test hardware. However, to debug programmes you need a hardware

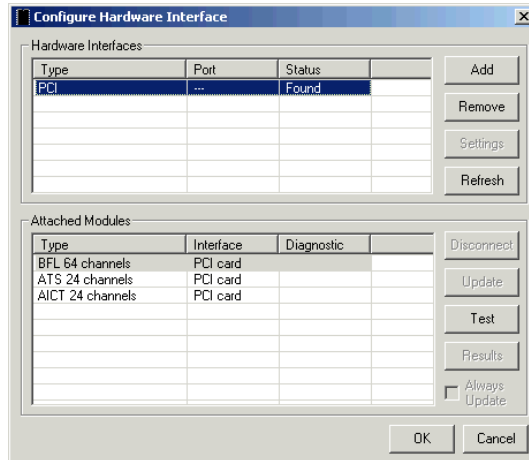
connection to the test product of your choice. Your SYSTEM 8 software will have already been configured to connect to the modules in your system and **CompactLink** makes use of these same connections. For the Compacts, a serial cable is used connected to a COM port on your PC. A summary of the options is shown in the table: -

Product	Connection Options	Comments
SYSTEM 8 BFL SYSTEM 8 AICT	Internal PCI card, or USB External Case	The PCI connection requires the SYSTEM 8 module to be installed in a drive bay in your PC. Refer to your SYSTEM 8 manual for further details. The BoardMaster 8000 Plus has pre-installed PCI cards.
SYSTEM 8 ATM	USB	
ChipMaster/Linear Master Compact Professional	USB	Older Compact products require a USB-RS232 converter and serial cable, available from ABI.

4.7.1. Connecting to SYSTEM 8 modules

Follow these steps to configure your **CompactLink** software to connect to your SYSTEM 8 test hardware: -

- If you have not already done so, install the SYSTEM 8 modules and confirm that they function correctly with the SYSTEM 8 Premier software (see SYSTEM 8 documentation).
- If you have not already done so, install the **CompactLink** software on the PC which is controlling your SYSTEM 8 modules.
- Ensure that all your SYSTEM 8 modules are connected and, if using an external case, ensure that the power is turned on and the USB cable is connected.
- Run the **CompactLink** software and choose **Tools/Configure Hardware** from the main menu or from within the test development and debugging window.
- Click **Add** to add a hardware interface.
- Select an **Interface Type** from the **Type** combo box according to the hardware configuration of the SYSTEM 8 module(s) on your PC. This will either be **PCI** for an internal module or **USB** for a module in an external case.
- The **Status** will be automatically updated by **CompactLink** and will show **Found** if the selected interface is present on your system.
- Click Refresh to update the list of attached modules. Confirm that the list is correct for your configuration.



- If you wish you can click **Test** to run the diagnostics on the attached module. The result is displayed in the list.
- Click **OK** to save the hardware configuration.

4.7.2. Connecting to Compact Professional USB products

Follow these steps to configure your **CompactLink** software to connect to your Compact Professional product: -

- If you have not already done so, install the **CompactLink** software on the PC which is controlling your Compact product(s).
- Connect your ChipMaster or LinearMaster Compact to your PC with a USB cable, turn on and select Cmlink mode (See Compact manual).
- Run the **CompactLink** software and choose **Tools/Configure Hardware** from the main menu or from within the test development and debugging window.
- Click **Add** to add a hardware interface.
- Select **Interface Type** from the **Type** combo box and choose **USB**.
- The **Status** will be automatically updated by **CompactLink** and will show **Found** if the selected interface is present on your system.
- Click Refresh to update the list of attached modules. Confirm that the list is correct for your configuration.
- If you wish you can click **Test** to run the diagnostics on the attached module. The result is displayed in the list.
- Click **OK** to save the hardware configuration.

4.7.3. Connecting to Compact products with serial cable



Follow these steps to configure your **CompactLink** software to connect to your older Compact product with a serial cable: -

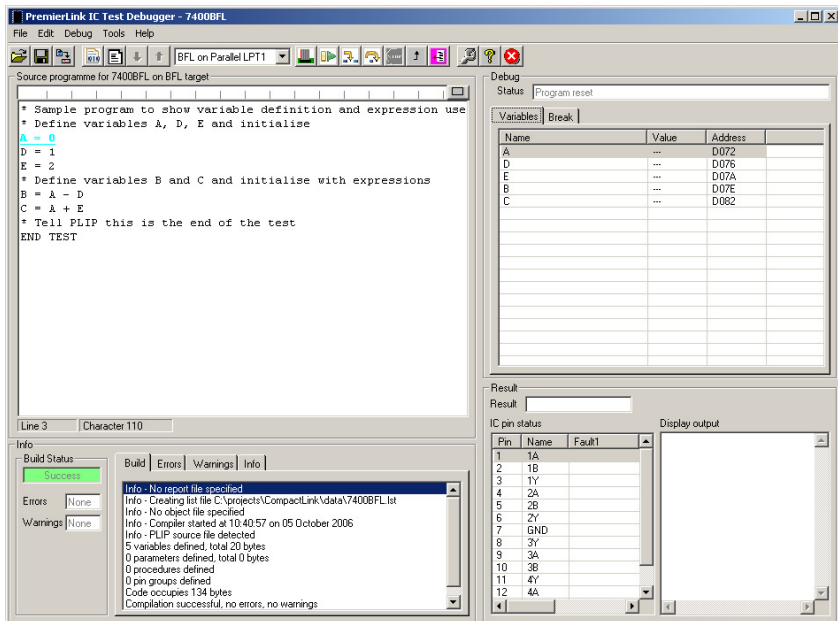
- If you have not already done so, install the **CompactLink** software on the PC which is controlling your Compact product(s).
- Power your Compact Product from mains using a battery eliminator.




- Connect your Compact to a COM port on your PC, either directly or to a USB port via a USB-RS232 converter.
- Turn on and select Cmlink mode (See Compact manual).
- Run the **CompactLink** software and choose **Tools/Configure Hardware** from the main menu or from within the test development and debugging window.
- Click **Add** to add a hardware interface.
- Select **Interface Type** from the **Type** combo box and choose **Serial**.
- Select a port for the interface using the **Port** combo box depending on the COM port used for the serial connection.
- Select the newly added interface by clicking and click **Settings**. Confirm that the serial port settings are Baud rate: 38400, Data bits: 8, Stop bits: 1, Parity: None, Handshaking: Hardware.
- The **Status** will be automatically updated by **CompactLink** and will show **Found** if the selected interface is present on your system.
- Click Refresh to update the list of attached modules. Confirm that the list is correct for your configuration.
- If you wish you can click **Test** to run the diagnostics on the attached module. The result is displayed in the list.
- Click **OK** to save the hardware configuration.

4.8. Debugging your programme

No matter how skilful you are as a programmer, inevitably your programme will have problems (commonly called “bugs”) in it. The purpose of the debugger is to help you identify these problems and fix them before adding the test to your library. As an exercise in using the debugger, enter the short programme as described in sections 4.3 to 4.5 above, then carry out the following steps: -


- Compile the programme by clicking the  **Build Test** button.
- Send the compiled test programme to the hardware by clicking the  Send Test button. You should then see the following display: -





- The 4 execution buttons  **Execute**, **Step In**, **Step Over** and **Reset** are now enabled, and the current execution line (**A = 0**) is highlighted.
- On Compact products only, a further button  **Stop** is displayed but not yet enabled. This enables a running programme to be stopped, and is only enabled during programme execution.
- Note that the 5 variables A to E are listed in the **Variables** debug tab on the right, but the values are shown as "----" since we have not yet executed the program.
- Click the  **Step In** button to step the program. Notice that the variable values are now updated and the execution line moves on to the next line (**D = 1**). If you hover your mouse pointer over any of the variables in the **Source Programme** window, the value will be shown.
- Continue stepping the program and observe the variables updating as the program executes.

4.9. Setting breakpoints

Stepping through the simple programme above is easy enough, but for more complex programmes it can take a long time to step through the entire programme. **Breakpoints** are up to 3 defined locations in your programme where execution can be suspended to allow you to examine variables, check voltages etc. You can then

execute the program and full speed with the  **Execute** button, and the programme will stop at the first breakpoint encountered. To set a breakpoint, do the following: -

- Click on the line where you want to set the breakpoint, e.g. **B = A - D**
- Choose **Debug/Toggle Breakpoint** from the menu, or press **F9**, or right click and choose **Toggle Breakpoint** from the popup menu. The selected line will be bulleted to indicate the breakpoint and an entry will be made in the **Break** debug window on the right
- Click  **Reset** to reset the programme back to the start, then click  **Execute** to run the programme. You will see that the program stops at the selected line.
- You can now examine the variables to confirm that the program has executed correctly.

Note that the **Break** debug window includes two special breakpoints which are enabled by default but you can turn them off if you wish: -

- **Break on first FAIL.** This can be useful when writing IC tests since you can run the programme until the test fails, allowing you to quickly “home in” on problems in your programme.
- **Break at end of test** (only available on Compact products). This causes execution to stop at the end of the programme. This allows the final state of all programme variables to be examined before the test completes.

To remove a breakpoint, click on the line where the breakpoint has been set and choose **Toggle Breakpoint** again to remove it.

4.10. Debugging techniques

Debugging programmes is a complex skill that requires practice and experience. Nevertheless there are some ground rules you can follow to help you avoid errors in your programmes.

There are 3 types of errors that can occur in your programme: -

4.10.1. Compiler errors

Compiler errors occur if you mistype text or use incorrect syntax. These are easily fixed as the **PLIP** compiler provides error messages and the syntax guide helps you get the command right.

4.10.2. Run time errors

Run time errors are caused by illegal operations such as divide by zero which cannot be detected by the compiler as it has no knowledge of the intended values of variables in your programme. For example, if your programme includes the line **GAIN = OUTPUT / INPUT** you should ensure that the value of **INPUT** cannot contain zero. This could be done simply as follows: -

```

IF INPUT <> 0
    GAIN = OUTPUT / INPUT
END IF

```

If a run time error occurs, programme execution will stop and the cause of the error will be displayed in the **Result** window at the bottom right.


4.10.3. Logical errors

These are the most common type of errors in the programme and also the most difficult to find. The following techniques will help: -

- Single step your entire programme. This can be laborious but it will ensure your programme executes according to plan. Use the **Variables** window and the automatic mouse hover variable display to confirm that the variables have the correct values.



Single stepping a BFL or AICT PLIP test which is connected to a board (in-circuit) may result in extended backdriving times. During normal test execution these times are very short, but may cause damage to a board if allowed to occur when developing a test. For this reason we recommend using an IC in the out of circuit adapter or in an unconnected, powered socket during IC test program development.

- Break up your programme into **PROCEDURES** with well-defined input and output values, which can be tested in isolation. Once you have fully tested a procedure, you can use  **Step Over** to execute calls to it without stepping into the procedure itself, which reduces the amount of stepping you need to do.
- Consider what happens in your test if unusual circumstances are present. For example, if you are reading a voltage from an IC pin, remember there may be no power supply to the IC. A faulty IC may also give unusual voltages, which may upset your programme.
- Ensure that you thoroughly understand the function of the IC you are testing. You will be unable to write a functional test programme if you do not know how the IC will react to input signals, so obtain an up to date data sheet for the device.
- If your programme contains complex calculations, split them into several lines using intermediate variables so you can follow the calculation while stepping.
- Ensure you are aware of the order of precedence of operators (see on line syntax guide). For example, consider the programme sequence: -

A = -1

B = 5

CONDITION = A < 0 & B > 3

The **&** operator is a higher order than the relational operators **<** and **>**. Therefore the expression is evaluated as: -

CONDITION = (A < (0 & B)) > 3 which is probably not what you expect. To avoid confusion rewrite as: -

CONDITION = (A < 0) & (B > 3) which makes it clear what you are trying to achieve.


- If your programme uses loops (e.g. **DO WHILE**), ensure that the loop condition can eventually become false and your programme cannot get stuck in the loop. For example, consider the following loop in an AICT or LMC programme: -

```
DO WHILE VOLTAGE(OUTPUT) < 5
  DRIVE INCREMENTAL INPUT WITH 0.05
END DO
```

If the output voltage never exceeds 5V (which could happen, for example, if the IC is faulty) the programme will remain in the loop and will get stuck. To avoid this, try the following: -

```
* Set an execution limit for the loop
LOOP_LIMIT = 1000
* Adjust input voltage until output goes above 5V
DO WHILE (VOLTAGE(OUTPUT) < 5) & (LOOP_LIMIT > 0)
  DRIVE INCREMENTAL INPUT WITH 0.05
  * Count number of times we go round the loop
  LOOP_LIMIT = LOOP_LIMIT - 1
END DO
```

This is far more complex but ensures your program cannot get stuck in a loop. If the loop executes 1000 times the **LOOP_LIMIT** variable will become zero and the loop will exit. Once you are sure your programme is working correctly, you can remove such error trapping code. Note that this uses the **&** logical operator to combine 2 test conditions for the loop condition – ensure you use the brackets as shown to ensure the expression is evaluated as you intend.

On the CMC or LMC you can use the  **Stop** button to force a breakpoint in a loop which is executing indefinitely.

- Use the **DISPLAY** command to show debugging information in the **Display Output** window at the bottom right. For example in the above programme, **DISPLAY VOLTAGE(OUTPUT), VOLTAGE(INPUT)** will show the voltages at the input and output pins so you can see if they are as expected before using them in subsequent calculations, and **DISPLAY 1000 - LOOP_LIMIT** will show how many times your loop executed before exiting.
- When you have tested parts of your programme to your satisfaction, use breakpoints to stop your programme execution after the tested parts so you can then use stepping to test the remainder of the programme.

Further examples of common programme errors are given in the online syntax help.

5. Some common programming concepts

Although **PLIP** is a reasonably simple language to use, some of the concepts involved in IC test programming can be quite complex. The basic principle behind any IC test programme is quite simple: -

- Stimulate the inputs of the device under test with the correct logic levels or analogue voltages.
- Check that the outputs of the device under test respond as expected to the input signals.
- Ensure that the chosen sequence of input signals covers all aspects of device operation.

However, this is not always as simple as it may seem. With the BFL and AICT products test are carried out on devices in working, powered circuit boards. Some inputs of the device under test may be hard wired to the supply rails or to each other. Outputs of the device under test may be linked back to inputs of the same device. Even on the CMC and LMC products where there are no external influences to complicate the test, there still may be problems. To make your programme as successful as possible, always try to meet these objectives before your start: -

- Obtain a sample device of the type you wish to test.
- Obtain an up to date data sheet for the device.
- Start test development in an unconnected, powered IC socket before moving on to an in-circuit test.
- When developing an in-circuit test, obtain a schematic diagram showing the connections to the IC under test, or discover them using a continuity tester.

We will discuss some of these issues in this section.

5.1. Digital test programming

Digital test programming is easier than analogue test programming. The operation of the devices is better defined and there is less mathematics involved in testing. Digital tests can be written for the SYSTEM 8 BFL module and for the ChipMaster Compact

Remember the basic operations in a digital IC test program, stimulate the inputs and check the outputs. The following commands are provided in **PLIP** for digital test programming. For full syntax and examples see the on-line help: -

Stimulus Commands	Use
DRIVE	Drive a logic level onto the input of the IC under test
PULSE	Pulse the input (L->H->L or H->L->H) of the IC under test
Response Commands/Functions	
CHECK THAT	Check that the output of the IC under test is in a specified logic state

CONFLICTS	Check that the outputs of a tri-state IC are not being driven by something else
COMPARE	Compare a group of IC outputs with a specified value
RESPONSE ()	Return a value by reading the logic states on a group of IC outputs
Other Commands	
SET PULL STATE	Set the 10k pull up/down voltage high or low
INPUTS	Define the inputs of the IC under test

5.1.1. Combinational devices – gates, buffers, multiplexers

Combinational logic devices are the simplest type of logic devices – the output logic levels depend purely on the input logic levels, so your programme will probably proceed as follows: -

- Specify the inputs of the IC under test with the **INPUTS** command
- Apply the desired logic levels to the inputs with the **DRIVE** command
- Check that the outputs respond correctly with the **CHECK THAT** command

However, as you will see later in the example for the 7400 QUAD NAND GATE device, there are some traps you can fall in to, mainly concerned with automatic circuit compensation discussed in section 5.3

When writing a test for a combinational logic device, ensure that you cover all possible states to get the best test possible. For example, if you are testing a 4 input gate you will need 16 states to cover all combinations of the 4 inputs. This is best achieved with a loop using the **DO ... WHILE** construction to repeat the test with different input states.

5.1.2. Sequential devices – counters, registers, latches

Sequential devices are far more complex, and in fact the vast majority of digital devices are sequential. The device normally has one or more clock inputs, and the outputs depend on both the current inputs and on the history of the inputs, so you cannot just apply inputs and check the output response. For example, a 4 bit counter can count from 0 to 15 before starting again at 0. If you just apply a pulse to the clock input using the **PULSE** command, this will advance the outputs by 1 state, but unless your programme knows the initial state you cannot check whether the new state after the clock pulse is correct.

To deal with this problem, there are a number of techniques depending on the type of device: -

- If the device has a clear or reset function, test that first, as then the device will be in a known state.
- If there is no clear or reset function, read the current state of the device outputs using the **RESPONSE ()** function and use that in your programme to calculate the next state.

- Some device outputs may not be available externally (e.g. a counter may only have a carry output and the actual counter outputs may not appear externally). In this case you may have to clock the device many times until the carry appears, so that you then know what state the device is in.

In accordance with the above, a typical sequence for a sequential device would be:

-

- Specify the inputs of the IC under test with the **INPUTS** command
- Get the device into a known state using a clear or reset input, or issue clock pulses until a known state is reached
- Apply the desired logic levels to the inputs with the **DRIVE** command
- Apply one or more clock pulses depending on the nature of the device
- Check that the outputs respond correctly with the **CHECK THAT** command

Again, the problem is more complex for in-circuit testing. A device may have a clear or reset pin but it may be hard wired so it is not accessible. A typical test will use several of these techniques to achieve a successful result.

5.1.3. Tri-state devices – buffers, bus drivers

Both combinational and sequential devices may have tri-state outputs – these outputs can be turned off or made high impedance by an enable input, so that other devices on a board can drive the pins in a bus structured system. In out of circuit testing this presents no problem, but with an in-circuit test you need to ensure that the outputs of the device under test are not being driven by anything else and can respond correctly to the inputs.

On the SYSTEM 8 BFL there are BDO (bus disable output) signals to achieve this (see SYSTEM 8 documentation for further details) but **PLIP** provides the **CONFLICTS** command to check that the outputs are free from interference. The **CONFLICTS** command does not influence the test, but it provides a warning on the result display that the device may fail the test. Using the **CONFLICTS** command, a typical combinational tri-state device test sequence would be as follows: -

- Specify the inputs of the IC under test with the **INPUTS** command
- Drive the enable or chip select input to turn off the tri-state outputs
- Use the **CONFLICTS** command to check that the outputs are properly turned off
- Apply the desired logic levels to the inputs with the **DRIVE** command
- Check that the outputs respond correctly with the **CHECK THAT** command

For sequential devices with tri-state outputs similar principles apply.

5.1.4. LSI and complex devices

Testing complex high pin count devices such as CPUs and CPU peripherals is difficult. In many cases, the device data sheet does not specify exactly how the device responds to the inputs, and there may be minor differences in operation between the same devices from different manufacturers. Some devices may

require minimum clock speeds to operate, which means single stepping is impossible. In addition to all this, many devices are so complex that testing every conceivable aspect of device operation may not be feasible because the test would take too long.

Despite this, it is still possible to write tests for complex devices if a few general principles are observed: -

- ICs usually fail because of voltage spikes, static pulses etc on the device pins, so your test should try to ensure every pin is tested in both logic states even if the entire device function cannot be tested
- Many devices need the same sequence of signals repeating many times during a test (for example internal registers that need to be read or written to configure the device operating mode). Use the **PROCEDURE ... END PROCEDURE** construction to write data to registers, so that it can be called from several places in your programme
- If you cannot determine the exact response of the device outputs from the data sheet, use the following technique: -
 - Use **DRIVE** and/or **PULSE** commands to apply logic levels and clock pulses to the inputs
 - Use the **DISPLAY** command with the **RESPONSE ()** function to read the output response and display in the text output window in the **CompactLink** debugger
 - Once you have determined how the IC responds, include **CHECK THAT** commands to test for the expected response
- If the device will not operate at slow speeds, use the debugger breakpoint system in conjunction with the **DISPLAY** command to get debugging information about the test

5.2. Analogue test programming

Analogue ICs, by their very nature, are more difficult to test than digital ICs. Consequently analogue IC tests are often quite complex, even for very simple components such as transistors and diodes. Analogue tests can be written for the SYSTEM 8 AICT module and for the LinearMaster Compact.

The following commands are provided for analogue test programming: -

Stimulus Commands	Use
DRIVE ABSOLUTE	Drive a defined voltage onto the input pin of the IC under test
DRIVE INCREMENTAL	Change the voltage on the input pin by the defined voltage
DRIVE OPENLOOP	Drive a defined voltage onto the input pin without any error compensation
RESTRICT (AICT only)	Clamp the output pin to the specified voltage
UNRESTRICT (AICT only)	Remove the output clamp on the specified pin

SOURCE (AICT only on special channels)	Source the specified current to the given pin
Response Commands/Functions	
COMPARE	Compare the voltage/current with a specified value using a given tolerance
CURRENT () (AICT only)	Return the current from an output pin which has been clamped by the RESTRICT command
VOLTAGE ()	Return the voltage at a pin
Other Commands	
INPUTS	Define the inputs of the IC under test
SET FEEDBACK TO (LMC only)	Configure the LMC feedback network

5.2.1. Using the **DRIVE** commands

The **DRIVE ABSOLUTE** command is the most common command for analogue component stimulus. The specified voltage is applied to the input pin. The **DRIVE ABSOLUTE** and **DRIVE INCREMENTAL** commands have error checking built in to cope with situations where a pin cannot be driven or is shorted or linked to another pin on the AICT – these conditions can be evaluated with the **LASTDRIVE ()** function (see on-line help) and used to control the flow of your programme.

The **DRIVE INCREMENTAL** command is slightly different. This command measures the voltage at the pin before changing it by the specified amount, so you do not need to know the original voltage. For example, when checking the gain of a circuit the actual voltages used are not that important as it is the ratio of them that will give you the gain.

Note that the following 2 programme segments will give the same result: -

```
DRIVE ABSOLUTE INPUT TO VOLTAGE(INPUT) + 0.1
DRIVE INCREMENTAL INPUT BY 0.1
```

The first command measures the input voltage and then increases it by 0.1V. The second command does this internally without first measuring the voltage.

The **DRIVE OPENLOOP** command is used in only a few situations (such as the diode test described below). Normally, the **DRIVE ABSOLUTE** and **DRIVE INCREMENTAL** commands have error checking built in to ensure that the correct voltage is achieved in the presence of varying loads. The AICT output impedance is 34 Ohms, so if a high current load is present the output voltage will drop. The **DRIVE ABSOLUTE** and **DRIVE INCREMENTAL** commands recognise this and boost the voltage to compensate for the drop, but the **DRIVE OPENLOOP** command ignores this, which speeds up the programme operation. If you want to measure the output current, this can be calculated by measuring the difference

between the output voltage programmed and the actual voltage achieved and dividing the result by 34, as in the following example: -

```
VOUT = 1
DRIVE OPENLOOP INPUT TO VOUT
IOUT = ABS(VOUT - VOLTAGE(INPUT)) / 34
```

Here, the **DRIVE OPENLOOP** command is used to output 1V nominal to the pin. The actual output voltage achieved is then measured and the absolute difference is calculated between its value and the original value of 1V. The result is divided by 34 (the AICT output impedance) to give the output current.

5.2.2. Using the **RESTRICT** command

In normal circumstances, the same basic principles apply to developing an analogue IC test: -

- Specify the inputs of the IC under test with the **INPUTS** command
- Apply the desired voltages to the inputs with the **DRIVE ABSOLUTE** or **DRIVE INCREMENTAL** command
- Check that the outputs respond correctly with the **COMPARE** command

However, there are some additional commands available on the AICT for extra types of analogue tests which may be useful in certain circumstances. For example, a voltage regulator by its very nature has an output voltage that never changes in normal circumstances. You can still check its function without changing its output voltage by measuring the magnitude and/or direction of current flow out of its output pin. To do this, use the **RESTRICT** command to clamp the output to a specified voltage, then use the **CURRENT()** function to measure the output current. An example: -

```
* Specify parameters for the min/max currents, output
voltage and tolerance
PARAMETER MIN_CURRENT
PARAMETER MAX_CURRENT
PARAMETER SPEC_VOLTAGE
PARAMETER SPEC_TOL
* Measure the quiescent output voltage of the regulator
OUTPUT_VOLTAGE = VOLTAGE(OUTPUT)
* Check the output voltage using the specified
percentage tolerance
COMPARE OUTPUT_VOLTAGE WITH SPEC_VOLTAGE TOLERANCE
SPEC_VOLTAGE * SPEC_TOL
* Clamp the output to the measured voltage plus a
little bit
RESTRICT OUTPUT TO SPEC_VOLTAGE + 0.1
* The output current should now be close to zero as the
regulator turns off
```

```
COMPARE CURRENT(OUTPUT) WITH MIN_CURRENT TOLERANCE 0.05
* Now repeat with a lower clamp voltage
RESTRICT OUTPUT TO SPEC_VOLTAGE - 0.1
* The output current should now be non-zero as the
regulator turns on
COMPARE CURRENT(OUTPUT) WITH MAX_CURRENT TOLERANCE 0.05
UNRESTRICT OUTPUT
```

This quite complicated example introduces a number of concepts: -

- Firstly, parameters are used (see next section) to specify values used in the test.
- The regulator output voltage is measured with the **VOLTAGE()** function and compared with the specification voltage parameter using the **COMPARE** command, using the percentage tolerance specified in the **SPEC_TOL** parameter.
- The output is then clamped with the **RESTRICT** command to a voltage higher than the specified voltage. The regulator should then turn off and its output current will fall to a low value.
- The **CURRENT()** function is used to measure the output current and compare it with the minimum current parameter.
- This is repeated with the output clamped below the specified voltage. The regulator should then turn on and its output current will be high.
- Finally the clamp is turned off

The **RESTRICT** command automatically turns on the on the specified pin regardless of whether this has been enabled by an **INPUTS** command.

Note that this is a simplified example. In practice, the effects of in-circuit components may affect the results of the current measurements. You should ensure your test is debugged on the actual board you intend to use with the finished test to get your test working correctly.

5.2.3. Using parameters

Parameters are constants used only in AICT/LMC tests. They are initialised with values in the **Device Information** window. This allows many devices with different specifications to share the same test – for example, in the voltage regulator test example in the previous section parameters are used for the output voltage, tolerance, minimum and maximum currents. This allows the same test to be used for voltage regulators with different output voltages.

To specify parameters, enter **PLIP** code in the **Parameters** box in the **Device Information** window for the device under test. For the above example, the parameters might be entered as follows: -

```
MIN_CURRENT = 0
MAX_CURRENT = 0.1
SPEC_VOLTAGE = 5
SPEC_TOL = 0.05
```

When the test is compiled, the given parameters are initialised to the values entered, which can then be used in your programme rather than actual numbers, allowing your test to be re-used.

Similar parameters can be used for test voltages and currents in most types of analogue tests. For example, the forward voltage drop for a Schottky type diode will be less than a normal silicon diode. A parameter can be used to set the expected voltage drop, which can then be used in your test so that the same diode test can be used for both types of diodes.

5.2.4. Using the **SOURCE** command

The AICT has 24 test channels, of which channels 1 to 3 are referred to as “special” channels with an enhanced range of functions. The **SOURCE** command can only be used with channels 1 to 3 and can source a current (rather than the more usual voltage) to the pin. This is useful for several types of test for discrete components such as transistors and diodes.

The **SOURCE** command automatically turns on the driver on the specified pin regardless of whether this has been enabled by an **INPUTS** command.

Note that if the specified current is negative, the **SOURCE** command will actually sink current instead of sourcing it.

Refer to the DIODE test example below for more information.

5.3. *Automatic circuit compensation*

Testing ICs in-circuit is one of the most difficult aspects of device test programming. There are a number of ways in which IC inputs could be wired on a PCB which influence your test strategy: -

- Devices can have unused inputs shorted to supply rails or linked to used inputs
- A device output may be linked to an input of the same device
- Device inputs may be connected to mechanical switches or jumpers which may or may not short the inputs

Your test programme must allow for these situations if you are writing an in-circuit test (e.g. for the BFL or AICT). The following **PLIP** functions are available to allow you to handle this: -

Function	Use
LASTDRIVE ()	Return true (1) if the last DRIVE or PULSE command worked, otherwise false (0)
LEVEL ()	Return the logic levels on driven inputs
LINKED ()	Return true (1) if the given pins are linked, otherwise return false (0)
SHORTED ()	Return true (1) if the pin is shorted to

	either power rail, otherwise return false (0)
CONFLICTS	Check that the outputs of a tri-state IC are not being driven by something else
COMPARE	Compare a group of IC outputs with a specified value
RESPONSE ()	Return a value by reading the logic states on a group of IC outputs
RESULT ()	Return a value (normally 0 = FAIL, 1 = PASS) representing the current test result
SET RESULT TO	Set the test result to PASS or FAIL, overriding the current result

5.3.1. Compensation by splitting

Splitting the test programme into sections is used for multi gate devices where there may be interconnections between gates in the same package. For example, for a quad 2 input gate, each gate should be tested separately rather than testing all gates at the same time. This is achieved by multiple use of the **INPUTS** command as in the following example for a dual 4 input gate: -

```

INPUTS 1A,1B,1C,1D,2A,2B,2C,2D
INPUTS 1A,1B,1C,1D
DO GATE1_TEST
INPUTS 2A,2B,2C,2D
DO GATE2_TEST
END TEST

```

Note carefully the use of the **INPUTS** command in this example. In any IC tests, the very first (and only the first) **INPUTS** command carries out extra tests on input pins (checking for linked or shorted pins, checking drive levels etc) which do not run on subsequent **INPUTS** commands. The results of these tests are important (e.g. they are used in the **LINKED ()** and **SHORTED ()** functions) so they should not be omitted. For this reason the very first **INPUTS** command in your programme should specify ALL the device input pins, even if you only want to use a subset of them as with the 2nd **INPUTS** command.

In this programme the two 4 input gates in the package are tested separately. If the output of gate 1 was connected to an input of gate 2, for example, this will not cause a problem because the inputs of gate 2 are not driven during testing of gate 1, and vice versa. Note that the actual test for each gate will probably require further circuit compensation using one of the techniques described below.

5.3.2. Compensation by skipping

The simplest method of circuit compensation is to just miss out any parts of your programme where the correct input signals could not be achieved. For example, in

the following program segment the programme is written to allow for the clear pin being possibly connected to VCC in-circuit: -

```
SET <Q_OUTPUTS> = QA, QB, QC, QD
DRIVE (CLEAR) HIGH
PULSE (CLEAR) LOW
IF LASTDRIVE ()
    CHECK THAT RESPONSE (<Q_OUTPUTS>) = 0
END IF
```

In this example which is part of a counter test, the programme is attempting to clear the counter and check that all its outputs go low in the following sequence: -

- The 4 bit counter outputs **QA** to **QD** are defined as a pin group to simplify the programme
- The clear pin (note the brackets indicating active low) is driven high to initialise it
- The programme applies a low pulse to the clear pin to clear the counter outputs
- The **LASTDRIVE ()** function is used to see if the **PULSE** command worked correctly
- If the **PULSE** worked, the outputs are tested to check they are all low, otherwise the test is skipped

There are a couple of points to watch out for when using this method of circuit compensation: -

- You should ensure that regardless of the connections that some tests are actually carried out on the IC. If your programme is structured incorrectly it may be possible for the entire testing of the IC to be skipped, so you get a PASS result without actually testing anything!
- You should not rely on the results of a programme segment that may skip a test. In the above example the counter may or may not be cleared depending on the wiring of the clear pin, so you cannot assume the counter is cleared in subsequent test programming
- The **LASTDRIVE ()** function returns different values for AICT tests. Refer to the on-line help for details.

In the above example the programme attempts to pulse the clear pin even though it might be shorted. An alternative way to implement this would be: -

```
SET <Q_OUTPUTS> = QA, QB, QC, QD
IF NOT (SHORTED ( (CLEAR) ))
    DRIVE (CLEAR) HIGH
    PULSE (CLEAR) LOW
    CHECK THAT RESPONSE (<Q_OUTPUTS>) = 0
END IF
```

This programme checks to see if the clear pin is shorted, and if so it just ignores it. Note the brackets that are required because the clear pin is active low.

5.3.3. Compensation by adapting

A more rigorous approach to circuit compensation is to adapt the programme to the prevailing circuit conditions, rather than just to skip sections of it. The procedure is as follows: -

- Drive the inputs of the IC using the **DRIVE** command as usual
- Read the logic levels on the inputs using the **LEVEL()** function
- Calculate the expected response of the IC outputs
- Check that the outputs respond as calculated

An example for a 2-input NAND gate (from a QUAD package) would be: -

```
INPUTS 1A, 1B, 2A, 2B, 3A, 3B, 4A, 4B
INPUTS 1A, 1B
INPUT_STATE = 0
DO WHILE INPUT_STATE <= 3
    DRIVE [1A, 1B] WITH INPUT_STATE
    IF LEVEL([1A, 1B]) = 3
        CHECK THAT 1Y IS LOW
    ELSE
        CHECK THAT 1Y IS HIGH
    END IF
    INPUT_STATE = INPUT_STATE + 1
END DO
```

In this programme the variable **INPUT_STATE** is used for the input (stimulus) data for the IC, but this data is never used in checking the outputs. Instead, the actual logic levels present on the inputs are used to decide what state the output should be in. In this way the NAND gate is correctly tested regardless of the connections to the input pins.

Note that the **LEVEL()** function is similar to the **RESPONSE()** function with one important difference. The **RESPONSE()** function is designed for reading IC outputs (along with the **CHECK THAT** command) and as such includes a text for mid-level outputs (voltage not within valid threshold levels), which is a common fault. However, when driving the inputs of an IC there may be mid-level voltages present depending on circuit conditions, so the **LEVEL()** function is provided to read the IC inputs and ignore any mid-levels to prevent your test from failing.

5.3.4. Compensation by trying

This method of circuit compensation is a last resort method, which can be used, if the above methods do not work for any reason. The idea is to use a test procedure to determine whether the IC can be tested in a particular way or not using the following principle: -

- Remember the current result of the test so far using the **RESULT()** function
- Set the result to PASS using the **SET RESULT TO** command
- Try a test procedure and check the result of it using the **RESULT()** function

- If the result is FAIL, restore the previously remembered result so that the procedure tried is ignored

An example of this is: -

```
* Remember current result
CURRENT_RESULT = RESULT()
* Set to PASS so we can try a test
SET RESULT TO PASS
* Try the test
DO TRY_TEST
* See whether the test failed
IF RESULT() = 0
    *Try failed, restore the previous result to ignore
the test
    IF CURRENT_RESULT = 0
        SET RESULT TO FAIL
    ELSE
        SET RESULT TO PASS
    END IF
END IF
```

A well-designed programme will very rarely need to use this type of construction, but it is included here for completeness. In analogue testing, for example, this technique can be used to try testing a component such as a diode both ways round, so that the user does not have to worry about applying the test probes in the correct way.

6. Example of a 7400 digital IC test programme for the BFL/CMC

Now we are ready to write a complete IC test programme. In this example we will describe how we would write a **PLIP** test programme for a 7400 QUAD NAND GATE IC, and in this way introduce you to the concepts involved in test programming. The programme can be executed on either the BFL or CMC with one minor change.

6.1. Defining the IC inputs

The first step in any test programme is to define the input pins of the IC under test, so that the test target product can switch on the drive on these channels. This is achieved using the **INPUTS** command. Enter the command line into the editing window as follows: -

```
INPUTS 1A, 1B, 2A, 2B, 3A, 3B, 4A, 4B
```

This command tells **CompactLink** that the pins listed are all inputs to the IC, and any pins not listed are assumed to be outputs. Note that for test programming purposes power supply pins are assumed to be outputs from the IC under test. You do not need to refer to the pin numbers directly (although you can do so if you wish by using the syntax **PIN 1**, **PIN 3** etc.) because the compiler will substitute the correct numbers from the device information later.

When **CompactLink** or the target product executes this command line, it will perform various checks on the given pins prior to continuing with the test (depending on the target product). For example, on the BFL it will check the connections between the pins or shorts between pins and either supply rail, and it will check for the presence of changing signals which may interfere with the outcome of the test. On both BFL and CMC it will check that all the given input pins can be properly driven with valid logic levels. All these checks take place on the first, and only the first, **INPUTS** command line in your programme, so the first **INPUTS** command should define all the inputs.

At first sight we could now go on to drive the input pins with a suitable test pattern and check the output, but there is a very important factor which has to be considered first when testing ICs in circuit on the BFL, but this can be ignored on the CMC. This is AUTOMATIC CIRCUIT COMPENSATION, which is one of the most difficult aspects of in-circuit test programming. A 7400 QUAD NAND GATE IC may be connected in many different ways by hard wiring its inputs, and a good test programme will allow for all these connections by adapting itself accordingly. **PLIP** contains all the commands and functions you require to do this, but it is up to you to include these in your programme. Consider a situation where the output of gate 1 of the 7400 IC is connected to the input of gate 2 in the same IC - we would then have a problem, because the system would be driving the gate 2 input, thereby preventing the gate 1 output from responding to the test signals. The solution to this is to enter a second **INPUTS** command, this time only referencing the inputs of the gate being tested. The other inputs will then not be driven,

ensuring that they will not interfere with the output of the gate under test if they should be connected to it. Thus the programme now looks like this: -

```
INPUTS 1A, 1B, 2A, 2B, 3A, 3B, 4A, 4B
INPUTS 1A, 1B
```

6.2. Simple test for a logic NAND gate

Now we are ready to test the first logic gate in the IC, and to do this we need to drive both inputs with all four possible states according to the truth table for a NAND gate, and check that the output responds accordingly. We could do this with the following programme segment: -

```
DRIVE 1A LOW, 1B LOW
CHECK THAT 1Y IS HIGH
DRIVE 1A HIGH, 1B LOW
CHECK THAT 1Y IS HIGH
DRIVE 1A LOW, 1B HIGH
CHECK THAT 1Y IS HIGH
DRIVE 1A HIGH, 1B HIGH
CHECK THAT 1Y IS LOW
```

6.3. Logic NAND gate test with BFL circuit compensation

The above programme would correctly test the 2 input NAND gate according to its truth table and indeed would be sufficient for use on the CMC, but it is not very versatile. Again, on the BFL the problem is AUTOMATIC CIRCUIT COMPENSATION – if you are writing a CMC programme this section is not applicable as you are testing out of circuit only. The above programme assumes that both inputs pins can be driven with all 4 input combinations, but consider for example if the gate is wired as an inverter by connecting its two inputs together. In this case only the first and last **DRIVE/CHECK** combinations would work, but the middle two would fail because the pins are wired together. **PLIP** provides the **LASTDRIVE()** function to overcome this. Consider the following improvement to the above programme segment: -

```
DRIVE 1A LOW, 1B LOW
IF LASTDRIVE()
    CHECK THAT 1Y IS HIGH
END IF
DRIVE 1A HIGH, 1B LOW
IF LASTDRIVE()
    CHECK THAT 1Y IS HIGH
END IF
DRIVE 1A LOW, 1B HIGH
IF LASTDRIVE()
    CHECK THAT 1Y IS HIGH
END IF
```

```
DRIVE 1A HIGH, 1B HIGH
IF LASTDRIVE()
    CHECK THAT 1Y IS LOW
END IF
```

This introduces 2 new programming concepts, **IF** decisions and **FUNCTIONS**. The **IF ... END IF** block is a construction which allows the instructions contained within it to be executed only if a certain condition is true, otherwise they will be skipped. In this case, the condition is evaluated by the **LASTDRIVE()** function. In **PLIP**, a **FUNCTION** is a pre-defined operation or calculation that returns a value to your programme. This value may be a numeric answer or a simple TRUE/FALSE result, as in this case. The **LASTDRIVE()** function provides a TRUE result if the last drive command succeeded, and a FALSE result if it did not. In this way, the output of the IC in the above programme will only be checked if the inputs were driven with the correct values, otherwise the output state will be ignored. In this way the above programme will operate regardless of the wiring of the two inputs pins of the gate under test.

Notice that the programme lines within the **IF ... END IF** blocks are indented (using spaces or tabs). This makes it easier to follow the programme flow (visually) and makes programmes more readable.

6.4. Improved logic NAND gate test with BFL circuit compensation and looping

Whilst the above programme will work, there is a more compact way of achieving the same result using another programming construction, the **DO WHILE ... END DO** construction. This is a commonly found construction allowing blocks of programme code to be repeated until a condition is true. Consider the following programme segment: -

```
DATA = 0
DO WHILE DATA <= 3
    DRIVE [1A,1B] WITH DATA
    IF LEVEL([1A,1B]) = 3
        CHECK THAT 1Y IS LOW
    ELSE
        CHECK THAT 1Y IS HIGH
    END IF
    DATA = DATA + 1
END DO
```

This is more complex and introduces several other programming concepts. Firstly, we have now defined a variable, called **DATA**, which is initialised to the value 0 by the first line in the segment. Variables in **PLIP** are stored in floating point format and can have values ranging from -32767e-99 to +32767e+99. The variable name itself can have up to 30 alphanumeric characters including underscores, but must begin with a letter.

The second line contains a **DO WHILE** condition. All the programme lines in between the **DO WHILE** and **END DO** commands will be executed repeatedly until the condition is false. It follows therefore that the programme must contain code to change the condition otherwise the programme will stick in an endless loop! In this case, the condition is that the value of **DATA** must be less than or equal to 3 for the following programme lines to be executed. Also here, on line 3 we have introduced a modified form of the **DRIVE** command, using square brackets ([]) to group together the two input pins. This form of the **DRIVE** command allows a numeric value to be driven in binary form (i.e. 1 bit at a time) onto the pins contained in the command. In this case, this means that bit 0 of the variable **DATA** is driven onto pin **1A**, and bit 1 is driven onto pin **1B**.

The fourth line contains the **LEVEL ()** function. This is another function used to implement AUTOMATIC CIRCUIT COMPENSATION on the BFL as an alternative to the **LASTDRIVE ()** function mentioned earlier. The **LEVEL ()** function returns the binary value of the actual data driven onto the pins enclosed in the brackets, so that you can test or use this value in your programme. If you are writing a CMC test you can use the input variable **DATA** instead of the **LEVEL ()** function as the inputs cannot be shorted or linked in an out of circuit test. In this case, we are testing the binary value of the logic levels of the input pins, so that we can decide (using the **IF ... ELSE ... END IF** construction) which state to look for on the output pin. You will see that the above programme correctly tests the output pin according to the truth table for a NAND gate. Note here that we have introduced the **ELSE** command as part of the **IF ... ELSE ... END IF** construction. The command lines following **ELSE** and before **END IF** will be executed if the **IF** condition is false.

Finally, after one run through the test programme the value of **DATA** is incremented by 1, and the **END DO** command causes execution to return to the **DO WHILE** command line and repeat the entire programme section. This will continue until the value of **DATA** is 4, when execution will continue after the **END DO** command line. Notice in both the above two programme segments that we have indented, by 4 spaces (or a tab), the code following an **IF** or a **DO WHILE** statement, but the relevant **ELSE**, **END IF** and **END DO** commands revert to the original column on the display. This is not necessary for your programme to work, but it improves the readability of your programme particularly when **DO WHILE ... END DO** or **IF ... ELSE ... END IF** blocks are nested inside each other. We suggest you get into the habit of doing this when you write your programmes.

6.5. Complete programme for logic NAND gate

The complete BFL programme to test the first gate in the package now looks like this: -

```
INPUTS 1A, 1B, 2A, 2B, 3A, 3B, 4A, 4B
INPUTS 1A, 1B
DATA = 0
```

```
DO WHILE DATA <= 3
    DRIVE [1A,1B] WITH DATA
    IF LEVEL([1A,1B]) = 3
        CHECK THAT 1Y IS LOW
    ELSE
        CHECK THAT 1Y IS HIGH
    END IF
    DATA = DATA + 1
END DO
```

On the CMC, the programme would look like this as the LEVEL() function is not required: -

```
INPUTS 1A,1B,2A,2B,3A,3B,4A,4B
INPUTS 1A,1B
DATA = 0
DO WHILE DATA <= 3
    DRIVE [1A,1B] WITH DATA
    IF DATA = 3
        CHECK THAT 1Y IS LOW
    ELSE
        CHECK THAT 1Y IS HIGH
    END IF
    DATA = DATA + 1
END DO
```

It would be a good idea at this stage to test the programme with the debugger to ensure that it functions as expected, before going on to test the other three gates in the package. In this way, if any mistakes are found they can be corrected before continuing with the programme entry. However, for completeness, we will now give the complete programme for all 4 gates in the package. You can use the text editor copy and paste to quickly copy the above block 3 times, then all you need to do is change the pin names for the remaining three gates. The complete programme is as follows. Note that we have added comment lines (beginning with *) to make the programme more readable, and we have also introduced the **END TEST** command to mark the end of the programme. Also, if you are writing a CMC program the LEVEL() functions should be replaced with the variable **DATA** as explained above: -

*** TEST PROGRAMME FOR 7400 QUAD NAND GATE IC**

*** DEFINE ALL INPUT PINS**

INPUTS 1A,1B,2A,2B,3A,3B,4A,4B

*** DEFINE INPUTS FOR GATE 1**

INPUTS 1A,1B

```
* TEST ALL 4 COMBINATIONS OF INPUTS
DATA = 0
DO WHILE DATA <= 3
    DRIVE [1A,1B] WITH DATA
    * GET EXPECTED OUTPUT ACCORDING TO DRIVE LEVELS
    IF LEVEL([1A,1B]) = 3
        CHECK THAT 1Y IS LOW
    ELSE
        CHECK THAT 1Y IS HIGH
    END IF
    * NEXT VALUE OF DATA INPUTS
    DATA = DATA + 1
END DO

* DEFINE INPUTS FOR GATE 2 AND REPEAT ABOVE
INPUTS 2A,2B
DATA = 0
DO WHILE DATA <= 3
    DRIVE [2A,2B] WITH DATA
    IF LEVEL([2A,2B]) = 3
        CHECK THAT 2Y IS LOW
    ELSE
        CHECK THAT 2Y IS HIGH
    END IF
    DATA = DATA + 1
END DO

* REPEAT FOR GATE 3
INPUTS 3A,3B
DATA = 0
DO WHILE DATA <= 3
    DRIVE [3A,3B] WITH DATA
    IF LEVEL([3A,3B]) = 3
        CHECK THAT 3Y IS LOW
    ELSE
        CHECK THAT 3Y IS HIGH
    END IF
    DATA = DATA + 1
END DO

* REPEAT FOR GATE 4
INPUTS 4A,4B
DATA = 0
DO WHILE DATA <= 3
    DRIVE [4A,4B] WITH DATA
```

```
IF LEVEL([4A,4B]) = 3
    CHECK THAT 4Y IS LOW
ELSE
    CHECK THAT 4Y IS HIGH
END IF
DATA = DATA + 1
END DO
```

END TEST

If you have not already keyed in this programme we suggest you do it now. The programme is also included in the test 7400CMC or 7400BFL which are included as sample user devices in the database supplied with the software. After entering

the programme, click the  **Save Test** button on the toolbar to save it.

The above is a very simple example of a test programme, but it does show some of the main features of the language.

7. Example of a diode analogue test programme for the AICT

The following programme is identical to that in the standard library for diode tests. We suggest you paste this programme into a user diode device test and use the debugger to establish exactly how the programme works.

The test uses a variety of techniques to get the correct result and simplify operation for the user: -

- A parameter is used for the forward voltage so that the test can be used for different types of diode
- The **INPUTS** command with no arguments is used because the diode does not have inputs in the conventional sense.
- The procedure **DIODE_SHORT** uses the **RESTRICT** and **SOURCE** commands to measure the voltage across the diode (in both directions) with a specified current. If the voltage is too low (corresponding to a 10 Ohms impedance in this example), the programme decides the diode is shorted and the test fails
- Assuming the diode is not shorted, the procedure **DIODE_RAMP** is used to detect the voltage at which the diode turns on. This is carried out twice with the connections reversed, so the user does not need to know how to apply the probes
- The **DIODE_RAMP** procedure works as follows: -
 - In a loop, the **DRIVE OPENLOOP** (see on-line help) command is used to apply a gradually increasing voltage (initially 0.1V) to the anode with the cathode clamped to 0V
 - The diode current is measured with the **CURRENT ()** function at the cathode pin
 - The voltage across the diode is measured
 - The AICT output current is measured by finding the difference between the voltage output and the voltage across the diode. Dividing this by the AICT output impedance (34 Ohms) gives the output current, which is checked for an excessive value.
 - The change in the output voltage for this step is calculated. If this change is less than a small value (18.75mV) this is recorded
 - If the voltage does not change for 3 steps round the loop, this is defined as the turn on voltage which is recorded by setting **CLAMPED** to 1 and the loop exits
 - If the diode voltage exceeds the maximum forward voltage parameter, the loop exits as this is an error
 - The output voltage is increase by 75mV and the above procedure repeats provided that the output voltage has not reached a pre-defined limit
 - Once the clamped condition is reached, the diode parameters are calculated
- If the **DIODE_RAMP** procedure passed, the test is complete and the diode parameters are displayed

The complete test for the diode is as follows: -

- *Test for a small signal diode
- *Looks for short circuit faults
- *Checks diode can be forward biased and measures Vf
- *Reverse (leakage) test not practical in circuit.

PARAMETER VFMAX

```
*Test parameters
*max drive voltage
V_LIMIT = 11.99
*maximum output current
IOUT_MAX = 120e-3
*voltage step for diode ramp test
STEP = 0.075
*minimum differential test current
MINCURRENT = 2.5e-3
*test current for short circuit
ID1 = 20e-3

*Returned test parameters
VF = 0
VDIODE = 0
*Global variables
TEST_PASS = 0
CLAMPED = 0
SHORT = 0
RD = 1e6
IF1 = 0
IF2 = 0
ID = 0
*Test start **
INPUTS
*initialise test in first direction
ANODE_PIN = ANODE
CATHODE_PIN = CATHODE
*check for short circuit diode first
DO DIODE_SHORT
IF SHORT = 1
    *force result fail
    SET RESULT TO FAIL
    DISPLAY "Short circuit diode junction", NEWLINE
    TEST_PASS = 0
ELSE
    DO DIODE_RAMP
    IF CLAMPED = 1
```

```
TEST_PASS = 1
*diode detected, display results
DISPLAY "Diode detected", NEWLINE
DISPLAY "Anode = pin ", ANODE_PIN, NEWLINE
DISPLAY "Cathode = pin ", CATHODE_PIN, NEWLINE
DISPLAY "Vf = ", ROUND(VF,3), NEWLINE
DISPLAY "Rd = ", ROUND(RD,3), " Ohms", NEWLINE
DISPLAY "@ If = ",ROUND(ID,2), NEWLINE
END IF
*now test other way
CATHODE_PIN = ANODE
ANODE_PIN = CATHODE
DO DIODE_RAMP
IF CLAMPED = 1
TEST_PASS = 1
*display results
DISPLAY "Diode detected", NEWLINE
DISPLAY "Anode = pin ", ANODE_PIN, NEWLINE
DISPLAY "Cathode = pin ", CATHODE_PIN, NEWLINE
DISPLAY "Vf = ", ROUND(VF,3), NEWLINE
DISPLAY "Rd = ", ROUND(RD,3), " Ohms", NEWLINE
DISPLAY "@ If = ",ROUND(ID,2), NEWLINE
END IF
IF TEST_PASS = 0
*force result fail
SET RESULT TO FAIL
END IF

END IF
END TEST

PROCEDURE DIODE_SHORT
*Check for short circuit diode junction
*(Defined as impedance less than 10 ohms)
*Calculate 10 Ohm voltage drop
VMIN = ID1 * 10
RESTRICT CATHODE_PIN TO 0
*source current to anode
SOURCE ID1 TO ANODE_PIN
*measure voltages across the diode
VA1 = VOLTAGE(ANODE_PIN)
VC1 = VOLTAGE(CATHODE_PIN)
VF1 = ABS(VA1 - VC1)
IF VF1 > VMIN
*impedance too high, indicate not a short
SHORT = 0
ELSE
*possible short, check in other direction
```

```
INPUTS
RESTRICT ANODE_PIN TO 0
*source current to anode
SOURCE ID1 TO CATHODE_PIN
*measure voltages across the diode
VA1 = VOLTAGE(ANODE_PIN)
VC1 = VOLTAGE(CATHODE_PIN)
VF1 = ABS(VC1 - VA1)
IF VF1 > VMIN
    *impedance too high, indicate not a short
    SHORT = 0
ELSE
    *diode is short circuit
    SHORT = 1
END IF
END IF
INPUTS
END PROCEDURE

PROCEDURE DIODE_RAMP
ERROR = 0
SINCE_CHANGE = 0
*start voltage
VOUT = 0.1
RESTRICT CATHODE_PIN TO 0
VLAST = 0
DO WHILE ERROR = 0
    *drive new open loop voltage
    DRIVE OPENLOOP ANODE_PIN TO VOUT
    *measure diode current
    IF1 = ABS(CURRENT(CATHODE_PIN))
    *measure actual voltage
    V_ANODE = VOLTAGE(ANODE_PIN)
    V_CATHODE = VOLTAGE(CATHODE_PIN)
    VACTUAL = V_ANODE - V_CATHODE
    *check for excessive current
    IF ((VOUT - VACTUAL) /34) > IOUT_MAX
        *current excessive, stop ramp
        INPUTS
        ERROR = 1
        CLAMPED = 0
    END IF
    *calculate voltage change and compare to initial change
    VCHANGE = VACTUAL - VLAST
    IF VCHANGE < (STEP/4)
        SINCE_CHANGE = SINCE_CHANGE + 1
    ELSE
        SINCE_CHANGE = 0
    END IF
END WHILE
```

```
END IF
*save latest voltage
VLAST = VACTUAL
*if no output voltage changes for 3 steps, assume clamped
IF SINCE_CHANGE > 2
    CLAMPED = 1
    VF1 = VACTUAL
    *and stop the test
    ERROR = 1
END IF
*check in case maximum diode voltage is exceeded
IF VACTUAL > VFMAX
    ERROR = 1
    CLAMPED = 0
END IF
*increment output voltage
VOUT = VOUT + STEP
*check output voltage in case limit is exceeded
IF ABS(VOUT) > V_LIMIT
    VOUT = V_LIMIT
    ERROR = 1
    CLAMPED = 0
END IF
END DO
*Measure resistance if clamped
IF CLAMPED = 1
    *drive new open loop voltage (5mA increment)
    VOUT = VOUT + 0.17
    IF ABS(VOUT) > V_LIMIT
        VOUT = V_LIMIT
    END IF
    DRIVE OPENLOOP ANODE_PIN TO VOUT
    *measure diode current
    IF2 = ABS(CURRENT(CATHODE_PIN))
    *measure actual voltage
    V_ANODE = VOLTAGE(ANODE_PIN)
    V_CATHODE = VOLTAGE(CATHODE_PIN)
    VF2 = V_ANODE - V_CATHODE
    *calculate impedance
    DO CALCULATE_R
    *calculate average diode drop
    VF = (VF1 + VF2)/2
    *calculate average test current
    ID = (IF1 + IF2)/2
END IF
INPUTS
END PROCEDURE
```

```

PROCEDURE CALCULATE_R
*Calculate dynamic impedance
IF (IF2 - IF1) > MINCURRENT
    RD = (VF2 - VF1) / (IF2 - IF1)
ELSE
    RD = 1e6
END IF
END PROCEDURE

```

8. Example of an operational amplifier analogue test programme for the LMC

The following programme is designed for an LM324 quad operational amplifier test on the LinearMaster. We suggest you paste this programme into a user LM324 device test and use the debugger to establish exactly how the programme works.

The test uses a variety of techniques to get the correct result: -

- The test relies on a mid rail voltage for correct operation. Since the test program is not aware of the actual supply voltage used for the test, the mid rail voltage is measured at the start of the test using the LMC feedback network.
- The test uses PARAMETERS for common mode range, saturation voltages and tolerances. This allows the same test to be used for different quad op amp devices.
- The **INPUTS** command with no arguments is used to turn off all output drivers.
- Variables are used for pin names so that procedures **TEST_OPEN_LOOP**, **TEST_BUFFER** and **TEST_GAIN2** can be used for all 4 op amps in the package
- In the open loop test, a ground resistor is used to establish a mid rail voltage on the inverting input. The non-inverting input is then driven by a small voltage either side of this to make the output respond. The **DISPLAY** command is used to show the output saturation voltages achieved. The outputs are tested against the saturation voltage parameters, then the **COMPARE** command is used to force a test fail if the voltages are incorrect.
- In the buffer test, a feedback resistor is used to configure the op amp to have unity gain. The non-inverting input is then driven by a gradually increasing voltage and the output is checked at each stage using the **COMPARE** command.
- In the gain2 test, the feedback network is used to configure the op amp to have a gain of 2. The non-inverting input is then driven by a gradually increasing voltage and the output is checked at each stage using the **COMPARE** command. Note that the difference between the input/output voltages and mid rail voltage is used in the comparison.

* **LM324LMC**

* **Test for LM324 quad op amp on LinearMaster Compact**

```
* Tested in open loop, unity gain and gain of 2

* Define parameters
PARAMETER VCMRNEG
PARAMETER VCMRPOS
PARAMETER VSATNEG
PARAMETER VSATPOS
PARAMETER BUFFERTOL
PARAMETER GAIN2TOL

* Define variables
VIN = 0
VMID = 0
VSUPP = 0
VOUT = 0

* First measure mid rail voltage by following procedure
* 1) Turn off all pins
* 2) Enable the 100R pseudo ground resistor on an input
* 3) Measure the voltage at this pin and save

* Turn all pins off
INPUTS
* Enable 100R ground R on INV1 and no feedback R
SET FEEDBACK TO OUTPUT1, INV1, FB_OFF, GND_100R
* Measure the voltage on INV1 to use for rest of test
VMID = VOLTAGE(INV1)
* Measure the supply voltage
VSUPP = VOLTAGE(V+)

* Set up pins for op amp 1
INPUT_INV = INV1
INPUT_NINV = NINV1
OUTPUT = OUTPUT1
INPUTS NINV1
* Test op amp 1 in open loop mode
DO TEST_OPEN_LOOP
* Test op amp 1 in buffer mode (unity gain)
DO TEST_BUFFER
* Test op amp 1 in gain of 2 mode
DO TEST_GAIN2

* Set up pins for op amp 2
INPUT_INV = INV2
INPUT_NINV = NINV2
OUTPUT = OUTPUT2
INPUTS NINV2
* Test op amp 2 in open loop mode
```

```
DO TEST_OPEN_LOOP
* Test op amp 2 in buffer mode (unity gain)
DO TEST_BUFFER
* Test op amp 2 in gain of 2 mode
DO TEST_GAIN2

* Set up pins for op amp 3
INPUT_INV = INV3
INPUT_NINV = NINV3
OUTPUT = OUTPUT3
INPUTS NINV3
* Test op amp 3 in open loop mode
DO TEST_OPEN_LOOP
* Test op amp 3 in buffer mode (unity gain)
DO TEST_BUFFER
* Test op amp 3 in gain of 2 mode
DO TEST_GAIN2

* Set up pins for op amp 4
INPUT_INV = INV4
INPUT_NINV = NINV4
OUTPUT = OUTPUT4
INPUTS NINV4
* Test op amp 4 in open loop mode
DO TEST_OPEN_LOOP
* Test op amp 4 in buffer mode (unity gain)
DO TEST_BUFFER
* Test op amp 4 in gain of 2 mode
DO TEST_GAIN2

END TEST

PROCEDURE TEST_OPEN_LOOP
* Test op amp in open loop mode as follows
* 1) Connect ground resistor only to inverting input
* 2) Apply small +ve voltage (referred to mid rail)
* 3) Check that the output saturates high
* 4) Repeat with a small negative voltage
* 5) Check that the output saturates low

* Set up the ground resistor on the inverting input
SET FEEDBACK TO OUTPUT, INPUT_INV, FB_OFF, GND_100R
* Output a small positive voltage
DRIVE ABSOLUTE INPUT_NINV TO VMID + 0.1
* Measure the output voltage
VOUT = VOLTAGE(OUTPUT)
DISPLAY "Vsathigh=", VOUT, "V", NEWLINE
* Check if Vout is too low
```



```
IF VOUT < VSUPP - VSATPOS
    * Force a voltage too low fail on the output pin
    COMPARE VOLTAGE(OUTPUT) WITH VSUPP TOLERANCE 0
END IF
* Output a small negative voltage
DRIVE ABSOLUTE INPUT_NINV TO VMID - 0.1
* Measure the output voltage
VOUT = VOLTAGE(OUTPUT)
DISPLAY "Vsatlow=", VOUT, "V", NEWLINE
* Check if Vout is too high
IF VOUT > VSATNEG
    * Force a voltage too high fail on the output pin
    COMPARE VOLTAGE(OUTPUT) WITH 0 TOLERANCE 0
END IF
END PROCEDURE
```

PROCEDURE TEST_BUFFER

```
* Test op amp in unity gain mode as follows
* 1) Connect 1k from output to inverting input
* 2) Apply minimum voltage
* 3) Check that the output follows
* 4) Repeat with stepping voltage up to maximum

* Set up the feedback resistor on the inverting input
SET FEEDBACK TO OUTPUT, INPUT_INV, FB_1K, GND_OFF
* Start testing with Vin at bottom of CM range
VIN = VCMRNEG
* Repeat with increasing values of VIN in 1V steps
DO WHILE VIN < VSUPP - VCMRPOS
    * Drive the input and measure the output
    DRIVE ABSOLUTE INPUT_NINV TO VIN
    VOUT = VOLTAGE(OUTPUT)
    * Display the results for debugging purposes
    DISPLAY "Vin=", VIN, "V", " Vout=", VOUT, "V", NEWLINE
    * Compare output with input voltage
    COMPARE VOUT WITH VIN TOLERANCE 0.3
    VIN = VIN + 1
END DO
END PROCEDURE
```

PROCEDURE TEST_GAIN2

```
* Test op amp in gain of 2 mode as follows
* 1) Connect 1k from output to inverting input
* 2) Connect 1k from inverting input to ground
* 3) Apply minimum voltage
* 3) Check that the output follows
* 4) Repeat with stepping voltage up to maximum
```

```
* Set up feedback/ground resistors on inverting input
SET FEEDBACK TO OUTPUT, INPUT_INV, FB_1K, GND_1K
* Start testing at -1V (referred to mid rail)
VIN = VMID - 1
* Repeat with increasing values of VIN in 0.5V steps
DO WHILE VIN <= VMID + 1
    * Drive the input and measure the output
    DRIVE ABSOLUTE INPUT_NINV TO VIN
    VOUT = VOLTAGE(OUTPUT)
    * Display the results for debugging purposes
    DISPLAY "Vin=",VIN,"V", " Vout=",VOUT,"V",NEWLINE
    * Compare output with gain * diff input voltage
    COMPARE VOUT WITH VMID+2*(VIN-VMID) TOLERANCE 0.3
    VIN = VIN + 0.5
END DO
END PROCEDURE
```

9. *PLIP* command and function reference

9.1. *Introduction*

Full details of all **PLIP** commands and functions are included in the software so you can get help on syntax at any time while developing your programme. To access this on line syntax help, do the following: -

- Click in the **Source Programme** window in the word you wish to look up
- Choose **Help/Syntax** from the menu, or right click and choose **Syntax Help** from the popup menu, or press **F1**
- If the selected word appears in several topics, choose the most applicable topic from the list displayed
- **The PLIP Syntax Guide** will now be displayed

The following words/phrases are used throughout the command/function descriptions: -

expression - a valid expression containing numbers, variables, arithmetic and/or logical operators and functions

condition - an expression that evaluates to either 0 (FALSE) or 1 (TRUE). Usually the expression will include a relational operator (e.g. =, <= etc.) but any expression which gives the result 0 or 1 will work.

pin name - a text string of up to 8 characters giving the name of an IC pin as defined in the IC definition database. Note that you can use the default strings PIN 1, PIN 2 etc. if for some reason you do not wish to use the defined pin names. If the IC pin definition contains several pins with the same name, only the first one will be used by the compiler in programs. The remaining pins **MUST** be referenced by the text PIN 1, PIN 2 etc..

pin name list - a text string containing a list of up to 8 pin names defined as above. Usually the pin name list will be contained in square brackets [].

pin group - a text string that is used as an identifier to refer to a group of pins that are logically connected with each other. The pin group is identified by the use of angular brackets <>. The text string can have a maximum of 30 alphanumeric characters and can contain underscores.

procedure name - a text string which is used as an identifier to refer to a procedure defined in your program. The text string can have a maximum of 30 alphanumeric characters and can contain underscores.

... - this symbol is used in some of the examples to indicate that other, non-specified, program lines are present on these lines.

variable name - a text string that is used as an identifier to refer to a variable defined in your program. The text string can have a maximum of 30 alphanumeric characters and can contain underscores. It can also refer to the pre-defined array using the string ARRAY[].

10. Troubleshooting and support

If you suspect your **CompactLink** software is not functioning correctly, send an email to support@abielectronics.co.uk or your local dealer with full details of the apparent problem. We will respond as soon as possible with advice.

Many apparent faults can easily be solved by a software update, which can be downloaded free of charge in from www.abielectronics.co.uk.

11. Appendices

11.1. Library parameter reference

This section contains detailed information about the meaning of the various information entries for the **CompactLink** device library. All the device parameters are listed in alphabetical order with a brief explanation

Entry	Explanation	Limits
Auto Clip Position	BFL/AICT only. Ticked if automatic clip positioning is possible, usually when the device has at least 1 power and 1 ground pin	
Class	Functional classification of the device	
Connections test	Ticked if the connections test (shorts, liked pins etc) is enabled for the device on the BFL or AICT	
Current test	The currently selected functional test for the device	
Date	Date of last change to device. Not used by system	
Display Note	Ticked if there is a note to display for this device/target combination	
External Comps	Not on AICT. Ticked if the test for this device requires external components (e.g. monostable ICs)	
Family	Logic family of device. New devices are all in the USER family	
Function	Brief description of device function	100 characters max
Functional test	Ticked if the functional test is enabled for the device on this target product	
Ground Clip	BFL only. Ticked if the device may require the ground clip (see SYSTEM 8 help/manual)	
High threshold	Voltages above this value are defined as valid HIGH logic levels	-10V to +10V
Include device in XXX library	Ticked if the device is specified for testing on this target product	
Include in Search	Not on AICT. Ticked if this device is to be included in the target search (IC identifier) function. If the test takes a long time you may want to exclude it to speed up the search	
Input Load Check	Not on AICT. Ticked if the device inputs should be checked for excessive loading. This is the normal case but some good ICs have excessive loading (e.g. Some ULN series ICs)	
Language	When a note is present, this specifies the language to be used	
Last Compiled	Date of last compilation of the test. Automatically updated by the system	
Last Modified	Date of last modification to test.	

Low threshold	Automatically updated by the system Voltages below this value are defined as valid LOW logic levels	-10V to +10V
Name	Alphanumeric name for the device	20 characters max
Note	When an IC is tested, the note text will be displayed as a warning to the user. This field contains the text for the note for the device/target combination	Unlimited
Open collector	Ticked if one of more device outputs is open collector. Ignored for AICT and LMC.	
Open emitter	Ticked if one of more device outputs is open emitter. Ignored for AICT and LMC.	
Package	Package type of the device	
Parameters	When 2 or more devices share the same test, the tests can be configured by using parameters to initialise variables in the test (e.g. to use different voltages for each device). This field contains the parameter initialising code in PLIP format.	Unlimited
Pin out	List of pin names for device. To display a negated pin in SYSTEM 8, enclose all or part of the pin name in brackets.	8 characters per pin max (not including negation brackets)
Power Supply	Not on AICT. Power supply voltage for the device. Note. Currently all devices on the BFL are tested at 5V regardless of this field which is included for future applications.	BFL and CMC: 3V to 5V LMC 2.5V to 10V
Switch threshold	Voltages below this value and above low threshold are MID LOW (invalid). Above this value and below high threshold are MID HIGH (invalid)	-10V to +10V
Technology	When sorting the list of devices in the library, the default order is alphanumeric. However, if Intelligent sort is enabled (Tools/Options/Review) this text field can be used to separate devices in your user library from different device technology groups (e.g. LS, HC, ACT etc). It can be left blank if not required.	20 characters max
Test Bit High	BFL only. Voltage on test bit when in high level	0V to +5V
Test Version	Version identification string for the device. Not used by system	10 characters max
Tri state	Ticked if one of more device outputs is tri state. Ignored for AICT and LMC.	
Type (package)	If the main package type is DISCRETE, this setting is the type of discrete package specified	
Use Number	For Compact products the device number must be numeric. This number will be used to recognise the user test on the Compact when entered on the Compact	7 characters max

	keypad.	
Version	Version identification string for the device. Not used by system	10 characters max
V-I test	Ticked if the V-I test and thermal test is enabled for the device on the BFL.	
Voltage test	Ticked if the voltage test is enabled for the device on the BFL or AICT.	

11.2. CompactLink *error/warning messages*

Message	Meaning	Action
A folder name cannot contain any of the following characters: \ / : * ? " ' < >	You are specifying an invalid character in a folder name	Choose a different name
A maximum of three breakpoints can be set	You cannot set more than 3 breakpoint sin your programme	Remove one of the other breakpoints and set the new one
Are you sure you want to delete 'XXXX'?	You are about to permanently delete the given device	Click Yes to delete or No to abandon the operation
Are you sure you want to delete the test:: XXXX	You are about to permanently remove the specified test	Click Yes to delete, or No to abandon
Build cancelled by user	The USER library generate operation was cancelled by the user	Re generate the USER library files
Build failed	The USER library file generation failed due to an error	Identify and correct the error, then re-generate the USER library files
Cannot set breakpoint on this line	The selected line is either a comment or has no executable code present (e.g. SET pin group command)	Choose another line for your breakpoint
Error - syntax help for 'XXXX' not found	The word under the cursor has no matching syntax help topic	Choose another word
Error - syntax help for this command not found	Missing syntax help for the selected topic	Choose another topic. Contact ABI with details of the problem
Error adding new test	There was an error saving the new test details in the database	Contact ABI with details
Error copying test	There was an error saving the copied test details in the database	Contact ABI with details
Error loading device	There was an error loading the device details from the database	Contact ABI with details
Error loading test	There was an error loading the test details from the database	Contact ABI with details
Error saving device	There was an error saving the device details in the	Contact ABI with details

Error saving test	database There was an error saving the test details in the database	Contact ABI with details
Feature not implemented	The selected function is not present in the software	Contact ABI with details
Memory dump size must be between 1 and 12k (3000H) bytes	The size of the displayed memory block must be between 1 byte and 12k (12,288) bytes	Change the start and/or stop addresses for the memory dump
Must be numeric value from X to Y	The voltage value being entered is invalid	Re-enter according to the limits given
No devices to build	There are no devices in your USER library	Add at least one USER device before continuing
Please enter a test name	You are trying to rename a test with a blank test name	Enter a valid name for the test
Program not built or has errors, cannot set breakpoint	You cannot set breakpoints until you have successfully compiled your programme	Correct and errors and recompile the programme
Search text was not found	The text being sought in the programme was not found	Re-enter the text to search for
Source has been changed, do you wish to save the changes?	The programme source text has been changed. If you exit the debugger now you will lose your changes	Click <i>Cancel</i> and save the changed programme before continuing
Target location does not exist. Do you want to create it?	The chosen location for the generated library files does not exist	Click <i>Yes</i> to create a new folder with the given name
The database file is read only and must have write permissions to be used in <i>CompactLink</i>	The main IC library database cannot be opened	Check that the file <i>CompactLinkICLibrary.dat</i> is present in the <i>CompactLink</i> folder and check that it is not read only. Check that it is not open within another running version of <i>CompactLink</i>
There are devices that use this test. You must select alternative tests for these devices before this test can be removed	You cannot delete a test if one or more devices are using it	Click <i>Yes</i> to attempt to fix the problem Specify alternative tests for the devices before deleting
There was an error opening the library database	The main IC library database cannot be opened	Check that the file <i>CompactLinkICLibrary.dat</i> is present in the <i>CompactLink</i> folder and check that it is not read only. Check that it is not

This will overwrite existing test, continue?	You are loading a text file which will overwrite the existing programme	open within another running version of <i>CompactLink</i> Click <i>Yes</i> if you want to overwrite the programme, otherwise click <i>No</i> and save the existing programme
Unable to find the library database	The main IC library database cannot be opened	Check that the file <i>CompactLinkICLibrary.dat</i> is present in the <i>CompactLink</i> folder
Unable to make database read-write	The main IC library database cannot be set to read/write mode	Check that the file <i>CompactLinkICLibrary.dat</i> is present in the <i>CompactLink</i> folder and check that it is not read only. Check that it is not open within another running version of <i>CompactLink</i>
Unable to proceed, press OK to close <i>CompactLink</i>	The software cannot continue	Click OK and restart the software
Undefined discrete package type	The specified discrete package is undefined	Contact ABI with details
Undefined package type	The specified package is undefined	Contact ABI with details
XXXX already exists, please use a different name	You are trying to rename a test using a name that already exists	Choose a different name

11.3. PLIP error messages

Message	Meaning	Action
Cannot change parameters	Parameters are read only and cannot be changed in a programme	Remove the code which is attempting to change the parameter, or use a variable instead of a parameter
Cannot define procedure inside a 'DO WHILE' loop	Procedures cannot be defined inside a programme loop	Move the procedure elsewhere in your program
Cannot define procedure inside an 'IF ... ELSE ... END IF' construction	Procedures cannot be defined inside a programme IF ... ELSE ... END IF construction	Move the procedure elsewhere in your program
Cannot end test inside procedure	The END TEST command cannot be inside a procedure	Specify the END TEST command before the procedure definition(s) in your programme
'ELSE' found without corresponding 'IF'	The ELSE command did not match up with a corresponding IF	Add an IF statement at the correct location, or remove the ELSE statement
'END DO' without	The END DO command	Add a DO WHILE

corresponding 'DO WHILE'	did not match up with a corresponding DO WHILE	statement at the correct location, or remove the END DO statement
'END IF' without corresponding 'IF'	The END IF command did not match up with a corresponding IF	Add an IF statement at the correct location, or remove the END IF statement
'END PROCEDURE' without corresponding 'PROCEDURE'	The END PROCEDURE command did not match up with a corresponding PROCEDURE	Add a PROCEDURE statement at the correct location, or remove the END PROCEDURE statement
Expecting ')' after expression or function	There was a missing closing bracket in the built-in function call or expression	Use correct syntax
Expecting ')' after function	There was a missing closing bracket in the built-in function call	Use correct syntax
Expecting ')' after round digits expression	There was a missing closing bracket after the round digits expression in the ROUND() built-in function	Use correct syntax
Expecting ',' after 1st pin in 'LINKED()' function	There is a missing comma in the pin list in the LINKED() function	Add the comma
Expecting ',' after 2nd pin in 'SET FEEDBACK TO' command	There is a missing comma in the argument list for the SET FEEDBACK TO command	Use correct syntax
Expecting ',' after expression	There was a missing comma after the given expression	Use correct syntax
Expecting ',' in group pin list	There is a missing comma in the list of pins given for the pin group	Correct the syntax of the pin list
Expecting ',' in input pin list	There is a missing comma in the list of pins given in the INPUTS command	Correct the syntax of the pin list
Expecting ',' in pin list	There is a missing comma in the list of pins given	Correct the syntax of the pin list
Expecting ',' to separate items for display	There is a missing comma in the list of arguments for the DISPLAY command	Use correct syntax
Expecting ']' after array index expression	There is a missing closing square bracket in the ARRAY statement	Use correct syntax
Expecting '"' to terminate string	The string in the DISPLAY command has no terminating double quote character	Use correct syntax
Expecting 'X' after expression or function	The specified character (usually a closing bracket)	Use correct syntax

	was not found after the expression or function	
Expecting 'X' in expression	The specified character (usually an opening bracket) was not found in the expression	Use correct syntax
Expecting '=' after array definition	There is a missing = after the ARRAY statement	Use correct syntax
Expecting '=' after pin group name	Invalid syntax in SET pin group command	Use correct syntax
Expecting '=' after pin list or group	Invalid SYNTAX in CHECK THAT command	Use correct syntax
Expecting '=' after variable name	There is a missing = after the variable definition	Use correct syntax
Expecting '>' to terminate pin group	The specified pin group does not have a closing angle bracket (>)	User correct syntax
Expecting 'BY' after pin name or number	Incorrect syntax of DRIVE INCREMENTAL command	Use correct syntax
Expecting expression	The compiler was expecting an expression, but none was found	Use correct syntax
Expecting 'LOW' or 'HIGH' after pin name or number	Logic level missing	Enter LOW or HIGH as appropriate
Expecting pin name or pin number	Compiler expecting a pin name or number	Use a valid pin name or number. Check the <i>Device Information</i> window for correct pin out
Expecting pin name or pin number, pin expression only valid for AICT/LMC programmes	The pin number can only be an expression for AICT or LMC programmes.	For BFL or CMC tests, use the pin name directly
Expecting string, expression, NEWLINE or CHR() after DISPLAY command	There is no valid argument for the DISPLAY command	Add a valid argument
Expecting 'THAT' before pin name, number, list or group	Invalid SYNTAX in CHECK THAT command	Use correct syntax
Expecting 'TO' after pin name or number	Incorrect syntax in DRIVE command	Use correct syntax
Expecting 'TO' after restrict pin number	Invalid syntax in RESTRICT command	Use correct syntax
Expecting 'TO' after source current expression	Invalid syntax in SOURCE command	Use correct syntax
Expecting 'TOLERANCE' after target compare expression	Invalid SYNTAX in COMPARE command	Use correct syntax
Expecting 'WITH' after actual compare expression	Invalid SYNTAX in COMPARE command	Use correct syntax
Expecting 'WITH' after pin list or group	Incorrect syntax of DRIVE command	Use correct syntax
Expression has more than	The expression is too	Split the expression onto 2

4 bracket levels	complex	or more lines using intermediate variables
Extra ')'	There was an additional closing bracket	Use correct syntax
Hexadecimal number greater than ^FFFF	You are attempting to specify a hexadecimal number greater than ^FFFF	Correct the number or specify in decimal
Input pin 'X' already defined	The given pin number is already used in the INPUTS command pin list	Define each pin only once in the INPUTS command
Invalid character in pin number	Pin number can only contain characters 0-9	Use only digits in the pin number
Invalid exponent, must be integer between -99 and +99	There is a decimal point in the exponent, which is not allowed.	Modify the programme to remove the decimal point.
Invalid pin list	Syntax error in pin list	Use correct syntax
Invalid procedure name	The procedure name contains invalid characters	Procedure names must begin with a letter and can contain only letters, numbers and underscores
Invalid variable name	The variable name contains invalid characters	Variable names must begin with a letter and can contain only letters, numbers and underscores
Missing 'END DO'	The DO WHILE command did not match up with a corresponding END DO	Add an END DO statement at the correct location, or remove the DO WHILE statement
Missing 'END IF'	The IF command did not match up with a corresponding END IF	Add an END IF statement at the correct location, or remove the IF statement
Missing 'END PROCEDURE'	The PROCEDURE command did not match up with a corresponding END PROCEDURE	Add an END PROCEDURE statement at the correct location, or remove the PROCEDURE statement
More than 'X' pins in pin list	There are too many pins in the pin list	Normally a pin list can have 8 pins but in some circumstances (e.g. LINKED() function) this limit may be less. Use the correct no of pins for the command/function
More than 5 nested 'DO WHILE' loops	The loop construction you have used is too complex and/or requires too much memory	Change the structure of your programme to reduce the number of nested loops to 5 or less
More than 5 nested 'IF' constructions	The IF ... ELSE ... END IF construction you have used is too complex and/or requires too much memory	Change the structure of your programme to reduce the nesting to 5 levels or less
More than 8 pins in list	The pin list in the given	Reduce the number of pins

	context can have a maximum of 8 pins	to 8 or less, or split the command over two or more lines
Name 'XXXX' already in use	The procedure name already exists for a variable, procedure or parameter	Use a different name
Number out of range, exponent must be between -99 and +99	The number you are entering is too small or too large, or is of invalid format	Numbers can range from -32767e99 to +32767e99. Check the syntax and use a number within this range
Parameter 'XXXX' already defined	The specified parameter already exists	Use a different name
Parameter 'XXXX' undefined	The specified parameter does not exist	Check the spelling of the parameter name, or define the parameter
Pin 'X' already defined	You have used the specified pin more than once in the CHECK THAT command	Use correct syntax
Pin 'X' defined more than once in pin list	The given pin is included twice or more in the pin list	Include each pin only once
Pin 'X' is undefined	The specified pin does not exist in the device pin-out	Check and correct the pin out and/or the pin name in your programme
Pin defined more than once in pin group 'XXXX'	The given pin is defined more than once in the pin group definition	Include each pin only once
Pin group 'XXXX' already defined	The given pin group name already exists	Use a different name
Pin group 'XXXX' has more than 8 pins	A pin group can have a maximum of 8 pins	Change to 8 pins or less or use 2 different pin group names
Pin group 'XXXX' undefined	The specified pin group does not exist	Check the spelling of the pin group name, or define the pin group
Pin group name 'XXXX' contains spaces	Spaces are not allowed in pin group names	Pin group names must begin with a letter and can contain only letters, numbers and underscores
Pin group name 'XXXX' has more than 30 characters	The pin group name is too long	Use a name with up to 30 characters
Pin is undefined	A pin does not exist in the device pin-out	Check and correct the pin out and/or the pin name syntax in your programme
Pin number 'X' greater than IC size (Y)	The specified pin number is greater than the number of pins for the device	Check the device pin-out and correct the pin number
Pin number 'X' greater than target product size (Y)	The specified pin number is greater than the number of pins on the target product for the device	Specify a device that can be tested on the chosen target product

Procedure 'XXXX' already defined	The procedure name already exists for another procedure	Use a different name
Procedure undefined	The given procedure name does not exist	Check the spelling of the procedure name, if present, or enter a valid procedure. Add an END TEST command before defining procedures
Procedure must be defined after main body of program	Procedures cannot be defined before the compiler reaches and END TEST command, otherwise the procedure could be executed at the wrong time	
Procedure name 'XXXX' has more than 30 characters	The procedure name is too long	Use a name with up to 30 characters
Pulse polarity changed	The polarity of the pulse command has changed within the same command	Ensure all polarity (LOW/HIGH) statements are the same within one PULSE command
String has more than 48 characters	The string in the DISPLAY command has more than 48 characters	Use a shorter string
Symbol table internal error	An internal error occurred	Usually caused by another error. Remove other errors and contact ABI with details if this error remains.
Too many pins in input list for a XX pin IC	There are more pins in the pin list for the INPUTS command than the size of the IC	Check the IC size and correct the pin list
Unrecognised syntax	The compiler does not recognise the syntax	Use correct syntax
Variable 'XXXX' undefined	The specified variable does not exist	Check the spelling of the variable name, or define the variable
Variable name 'XXXX' has more than 30 characters	The variable name is too long	Use a name with up to 30 characters

11.4. PLIP warning messages

Message	Meaning	Action
Command 'XXXX' is only valid for 'XXX' products	The specified command is not valid for the target product for this test	Use the correct command for the target, or change the target
'CONFLICTS PIN 1, PIN 3 ...' is old SLIM format, using 'CONFLICTS [PIN 1, PIN 3 ...]' form	Included for backward compatibility with old SLIM programmes	Use the current syntax with square brackets
'ENDDO' is incorrect syntax, assuming 'END DO'	Incorrect syntax of END DO command	Use correct syntax
'ENDIF' is incorrect syntax, assuming 'END IF'	Incorrect syntax of END IF command	Use correct syntax

'ENDTEST' is incorrect syntax, assuming 'END TEST'	Incorrect syntax of END TEST command	Use correct syntax
Extra characters ignored	There is some additional unnecessary text in the programme	Use the correct syntax
No 'END TEST', one assumed	There is no END TEST command in your programme	Include an END TEST command at the correct location
Test end already defined, ignored	More than one END TEST statement in programme	Use only 1 END TEST statement in your programme in the correct position

11.5. PLIP run time error messages

Message	Meaning	Action
Bad 10 bit DAC value	The voltage is too high for the 10 bit DAC operation.	Modify your programme so that the voltage expression evaluates to the correct voltage
Bad 8 bit DAC value	The voltage is too high for the 8 bit DAC operation.	Modify your programme so that the voltage expression evaluates to the correct voltage
Bad array index	The index of the ARRAY[] must be in the range 0 to 127.	Modify your programme so that the array index expression evaluates to a number in the range 0 to 127
Bad number value	The result of the expression is not legal for the operation being performed. For example, for the ROUND() function the number of decimals places must be 0 or positive.	Modify your programme to ensure that this cannot occur
Bad pin number	The pin number expression evaluates to a number which is either zero, negative, or greater than the IC pin count.	Modify your programmes to ensure that the pin number expression gives the correct result
Bad random seed	The seed for the RANDOM() function must be a non-zero positive number.	Modify your programme so that the argument expression for the RANDOM() function is always positive
Bad tolerance	The TOLERANCE argument for the AICT/LMC COMPARE command must be zero or positive.	Modify your programmes so that the TOLERANCE expression evaluates to a positive number
Divide by zero	Division by zero is impossible.	Modify the programme so that the divisor cannot become zero during execution, or test for this condition to avoid the division operation
High voltage	A voltage was discovered on a pin that is too high for safe operation of the test.	Check the power supply for the board under test and reduce the operating voltage
Invalid on this	The programme contains a	Use the correct target product,

product		command which is invalid on the target product.	or modify the programme.
Not special channel	special	The AICT SOURCE command is only legal for the special channels.	Modify your programmes so that the SOURCE command uses one of the special channels
Power supply overcurrent	supply	Compact products only. The IC under test appears to have a large supply current which cannot be supplied by the product.	Check with a multimeter – the IC cannot be tested and is probably faulty.
Power supply short	supply	Compact products only. The IC under test appears to have a short across its power supply pins.	Check with a multimeter – the IC cannot be tested and is probably faulty.
Stack overflow		The programme has run out of internal stack memory. This usually happens when a procedure is called repeatedly without returning.	Ensure that all procedures complete before they can be called again
Vout higher than PSU		LinearMaster Compact only. The programme is trying to output a voltage greater than the specified supply voltage. This could cause damage to the IC.	Modify the programme and/or change the specified supply voltage.

12. Index

A

Adding an IC · 9
Analogue test programming · 30
Appendices · 61
Automatic circuit compensation · 35

B

Breakpoints · 22

C

Checklist · 4
CompactLink error/warning messages · 63
CompactLink operation · 6
Compiler errors · 23
Compiling a programme · 17
Copying an IC · 8
Copyright · 2

D

Debugging · 21
Debugging window · 15
Deleting an IC · 12
Developing a functional test · 11
Digital test programming · 27
Disclaimers · 2
Documenting programmes · 18

E

Editing an IC · 8
Entering a programme · 17

Example of a 7400 digital IC test · 40
Exporting a device · 12

F

Fixing programme errors · 17
Fixing programme warnings · 17

G

Generating library files · 12
Getting started · 4

H

Hardware connection · 19
Help · 18

I

IC library data structure · 5
Installing **CompactLink** · 5
Introduction · 3
Introduction to PLIP · 15

L

Library parameter reference · 61
Logical errors · 24

M

Maintenance · 2

P

PLIP command and function reference · 58
PLIP error messages · 66
PLIP run time error messages · 72
PLIP warning messages · 72
Precautions · 1
Printing a device · 12
Programming concepts · 27

R

Reviewing the IC library · 7
Run time errors · 24
Running *CompactLink* · 5

S

Specifying a functional test · 9

System requirements · 4

T

Test development window · 15
Troubleshooting and support · 60

V

Viewing an IC · 7

W

Writing test programmes · 15