

General Test Program Development Principles Using TSL/1

Assumptions

It is assumed that the reader is familiar with the basic operation of our μ Master product, and has a reasonable understanding of processor board architecture.

Introduction

The μ Master test system takes control of a board's processor. Once in control, μ Master can be used to access all parts of the board. All testing consists of communicating with the various board components and buses. An unsuccessful communication indicates a defective part.

This application note describes the manual creation of a test program, in our μ Master TSL/1 scripting language, for a PC architecture board. However, the same principles will apply to other board types. Subsequent application notes examine the use of our Automatic Test Generation (ATG) technology. The ATG can greatly improve test development times.

Overview of Test Development

As with all engineering tasks, breaking the test development process into manageable steps is the best approach. The recommended approach consists of four top-level steps:

1. Understand the board structure. That is, divide the board into functional blocks, and create a hierarchical block diagram of the board structure, similar to the one shown in Figure 1.
2. Collect the datasheets for all the major board ICs.
3. Using the block diagram, work from the processor down, and identify the main functional blocks for which a test sequence can be implemented.

4. Create a test sequence for each of the identified functional blocks, using the μ Master TSL/1 language, together with a known-good board on which the test sequence can be run.

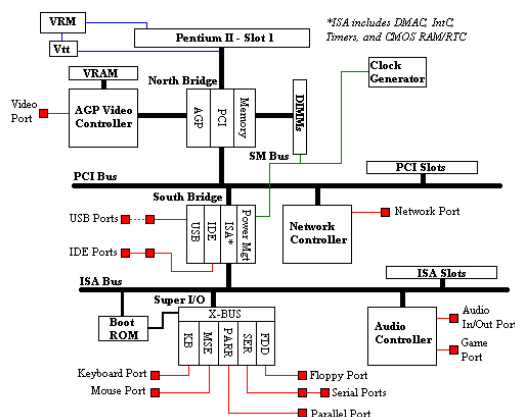


Fig. 1 – Example Architecture

Step 1 - Understanding Board Structure

Using available board documentation divide the board into functional blocks, and draw a block diagram of the board structure, similar to the one shown in Figure 1. Use the following points as a guide to do this:

1. Identify the board's main processor.
2. Identify major buses and their relationships (Host/processor bus, PCI, ISA, PMC, CompactPCI, etc.).
3. Identify any bus bridges (a bridge is an IC which converts one bus type to another bus type).
4. Identify board memory and assign addresses (Table 1 shows a typical PC memory map).
5. Identify major peripheral devices, the I/O components that drive them, and the buses that connect to the I/O devices. The best way to do this is to start at the I/O connector and work backwards. Assign I/O addresses to each I/O device (Table 2 shows a typical PC I/O map).

Address Range	Memory Type
00000000-0009FFFF (640K)	System DRAM
000A0000-000BFFFF	Video Memory
000C0000-000EFFFF	Expansion ROM or Shadow DRAM
000F0000-000FFFFFFF	BIOS ROM or Shadow DRAM
00100000-FFFFFFF	System DRAM (also used for expansion video memory)
FFFF0000-FFFFFFFF	BIOS ROM

Table 1. PC Memory Map

I/O Address	I/O Type
0000-000F	DMA Controller #1
0020-0021	Interrupt Controller #1
0040-0043	System Timer
0060-006F	Keyboard/Mouse Controller
0070-007F	CMOS RAM/RTC
0080-008F	DMA Page Registers
00A0-00A1	Interrupt Controller #2
0170-0177	IDE #2
01F0-01F8	IDE #1
0278-027F	Parallel Port
02F8-02FF	Serial Port
0378-037F	Parallel Port
03BC-03BF	Parallel Port
03C0-03DF	Video
03F0-03F7	Floppy Controller
03F8-03FF	Serial Port
0CF8-0CFE	PCI Configuration Space

Table 2 – PC I/O Map

Step 2 - Get the Datasheets

Collect the datasheets for all the major board ICs. The best sources for these are the websites of the IC manufacturers. See the links at the end of this document.

Step 3 - Define the Necessary Test Sequences

Using the block diagram, identify all the major blocks for which you can write a functional test sequence. A list of possible test sequences for the board example in Figure 1 are:

General Bus Test

North Bridge

- PCI Bridge and Memory Controller
- AGP Bridge

SDRAM

Video

- Video Controller
- Video Memory
- Video Output

South Bridge

- PCI-ISA Bridge
- DMA Controller
- Interrupt Controller
- CMOS RAM/RTC
- USB
 - Controller
 - Input/Output
- IDE
 - Controller
 - Input/Output
- SM-Bus

Network

- Controller
- Input/Output

Boot ROM

Super I/O

- Controller
- Keyboard
- Mouse
- Serial Ports
- Parallel Ports
- Floppy Disk

Audio

- Controller
- Input/Output

Step 4 - Create the Test Sequences

Although there are many types of circuitry to be tested, five types of test sequence structure can be identified which will cover most types of testing:

- Processor Area
- Buses
- Bus Bridges
- Memory
- Input/Output

NOTE - For a full explanation of TSL/1 command syntax consult the μ Master help file.

Testing the Processor Area

When a test begins, μ Master attempts to take control of the processor. If successful it generally indicates that the processor area has no faults. The processor is then used to test the other board areas. This provides further verification of the processor and its surrounding area.

If the processor fails, a Pin Level test can be performed. This should normally be followed by a Power off/on to ensure the board is in a reliable state before any further testing is carried out. See Application Note #3 for details on tracing faults around the processor.

Testing the Buses

Buses can be verified using the following methods:

- Run the μ Master automated ROM Bus test.
- Test any devices attached to the bus in question, which will indirectly verify the bus. For example, running a RAM test verifies the buses from the processor to the RAM device.

The TSL/1 command syntax for the ROMBusTest is:

```
ROMBusTest Start_address End_address [#Label]
```

This test verifies the operation of the buses from the processor to the boot ROM, and diagnoses most data and address bus problems (except shorts). Any defective data or address bus lines found are reported by bus type and line. See Application Note #3 and the μ Master help file for further information.

A sample bus test program written in TSL/1 is shown in Listing 1 below.

```
GENERAL BUS TEST

DefineConst C_BootStart 000FE000
DefineConst C_BootEnd 000FFFFF

'Use the ROM boot block area to perform a basic bus test.
'Host, PCI, and ISA buses are checked for opens and stucks
ROMBusTest C_BootStart C_BootEnd #Bus_Fail

Gprint OK
End

#Bus_Fail 'error handling
Rprint Failure
PrintError
EndError
End
```

Listing 1 - Bus Test

Testing Bus Bridges

A bus bridge converts one bus type to another, e.g. processor bus to PCI bus. All bridge chips contain programmable read/write registers. A read/write test to these registers will verify whether the chip is accessible to the processor. A read/write test consists of writing a test pattern to a register or registers within the bridge using the TSL/1 command:

```
WriteIOByte Port_Address Data
```

The data stored in the register is then verified against the written value using the TSL/1 command:

```
ReadIOByte Port_Address Mask Correct_Data [#Label]
```

The ability to write and read to the bridge verifies that the processor can communicate with the bridge.

Alternatively, most bridges contain a “hard-wired” ID code that can be read back and verified using the ReadIOByte command.

To fully test a bridge's functions, it is necessary to verify its ability to access devices (read or write to memory or I/O) on the other side of the bridge. In Figure 1 for example, if accesses to PCI bus devices such as the South Bridge are possible from the processor, it means that the North Bridge is functioning.

Listing 2 shows an example of a bridge test.

N.B. Some of the command lines in these listings have wrapped onto the following line due to this document's column width. So if you copy and paste text into a µMaster test, you will need to remove unwanted carriage returns.

```
Host-to-pci configuration register test

DefineConst C_PCICnfAddr 0CF8
DefineConst C_PCICnfData 0CFC

'Verify the presence of the North Bridge Host-to-PCI Function
'by verifying it's ID
WriteIODWord C_PCICnfAddr 80000000
ReadIODword C_PCICnfData FFFFFFFF 71908086
#NB_ID_Fail

'Initialise the North Bridge Host-to-PCI function registers.

WriteIODWord C_PCICnfAddr 8000000C
WriteIODWord C_PCICnfData 00004000
WriteIODWord C_PCICnfAddr 80000050
WriteIODWord C_PCICnfData F000000C
WriteIODWord C_PCICnfAddr 80000054
WriteIODWord C_PCICnfData 09000000
WriteIODWord C_PCICnfAddr 80000058
WriteIODWord C_PCICnfData 00111003
WriteIODWord C_PCICnfAddr 8000005C
WriteIODWord C_PCICnfData 11130000
WriteIODWord C_PCICnfAddr 80000060
WriteIODWord C_PCICnfData 08060402
WriteIODWord C_PCICnfAddr 80000064
WriteIODWord C_PCICnfData 08080808
WriteIODWord C_PCICnfAddr 80000068
WriteIODWord C_PCICnfData 33E82F00
WriteIODWord C_PCICnfAddr 8000006C
WriteIODWord C_PCICnfData 0000C003
WriteIODWord C_PCICnfAddr 80000070
WriteIODWord C_PCICnfData 381A1F20
WriteIODWord C_PCICnfAddr 80000074
WriteIODWord C_PCICnfData 01000055
WriteIODWord C_PCICnfAddr 80000078
WriteIODWord C_PCICnfData 2F900000
WriteIODWord C_PCICnfAddr 8000007C
WriteIODWord C_PCICnfData 00000010

'Verify the values written to the registers
WriteIODWord C_PCICnfAddr 8000000C
ReadIODword C_PCICnfData FFFFFFFF 00004000
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000050
ReadIODword C_PCICnfData FFFFFFFF F000000C
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000054
```

```
ReadIODword C_PCICnfData FFFFFFFF 09000000
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000058
ReadIODword C_PCICnfData FFFFFFFF 00111003
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 8000005C
ReadIODword C_PCICnfData FFFFFFFF 11130000
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000060
ReadIODword C_PCICnfData FFFFFFFF 08060402
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000064
ReadIODword C_PCICnfData FFFFFFFF 08080808
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000068
ReadIODword C_PCICnfData FFFFFFFF 33E82F00
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 8000006C
ReadIODword C_PCICnfData FFFFFFFF 0000C003
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000070
ReadIODword C_PCICnfData FFFFFFFF 381A1F20
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000074
ReadIODword C_PCICnfData FFFFFFFF 01000055
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 80000078
ReadIODword C_PCICnfData FFFFFFFF 2F900000
#PCI_Reg_Fail
WriteIODWord C_PCICnfAddr 8000007C
ReadIODword C_PCICnfData FFFFFFFF 00000010
#PCI_Reg_Fail

Gprint OK
End

#NB_ID_Fail
Rprint ID Failure
PrintError
EndError
End

#PCI_Reg_Fail
Rprint Register Failure
PrintError
EndError
End
```

Listing 2 - Bridge Test

Testing Memory

µMaster is supplied with a number of pre-programmed memory tests. These are performed by the following TSL/1 commands:

```
RAMBusTest Start_Address End_Address [#Label]
```

RAMBusTest verifies the buses up to the RAM chips. Bad data and address lines are automatically diagnosed and reported.

```
BasicRAMTest Start_Address End_Address [#Label]
```

BasicRAMTest verifies the buses up to the RAM chips, and also that all RAM cells are readable and writeable. Bad data and address lines, and RAM cells are automatically diagnosed and reported.

```
RWRAMTest Start_Address End_Address [#Label]
```

RWRAMTest performs a more pattern-sensitive RAM test. Use this, if the other RAM tests pass but you still suspect the RAM as being the cause of a problem.

```
DRAMRefreshTest Start_Address End_Address Delay [#Label]
```

DRAMRefreshTest verifies that all DRAM cells are refreshing correctly.

```
ROMTest Start_Address End_Address Correct_CRC [Label]
```

ROMTest performs a CRC check on a ROM device.

NOTE - See the μ Master help files for more information on these RAM/ROM tests.

Often RAM is controlled by a memory controller. This device contains internal registers which must be initialized before the RAM is visible to the processor.

In the PC architecture, the memory controller is part of the Host-PCI bridge, located in the North Bridge. By initializing this device's PCI configuration registers the memory will be turned on. There are a number of points to be aware of when initializing memory:

1. Shadow RAM or Boot ROM: All boards fetch their initial code from a boot ROM. In the PC architecture, the boot ROM is located on the LPC bus. During start up the boot ROM contents is transferred to RAM, and accesses to the boot area are diverted to the RAM. Therefore, the boot address space from 000C0000-000FFFFF can be located in RAM or ROM. This is controlled by specific PCI configuration registers in the Host-PCI bridge. These registers are typically called PAM or

Shadow RAM Enable registers. Before running a RAM or CRC check on this memory area, you should check these registers to ensure that the right device is being pointed at.

2. SDRAM: In addition to programming the memory controller to turn on the RAM, SDRAM contains an internal mode register which must also be programmed before testing can begin.

SDRAM

```
DefineConst C_PCIConfAddr 0CF8
DefineConst C_PCIConfData 0CFC
```

```
'Initialise the North Bridge Host-to-PCI function registers.
```

```
WriteIODWord C_PCIConfAddr 8000000C
WriteIODWord C_PCIConfData 00004000
WriteIODWord C_PCIConfAddr 80000050
WriteIODWord C_PCIConfData F000000C
WriteIODWord C_PCIConfAddr 80000054
WriteIODWord C_PCIConfData 09000000
WriteIODWord C_PCIConfAddr 80000058 'Ensure shadow
RAM is on
WriteIODWord C_PCIConfData 00111003
WriteIODWord C_PCIConfAddr 8000005C
WriteIODWord C_PCIConfData 11130000
WriteIODWord C_PCIConfAddr 80000060
WriteIODWord C_PCIConfData 08060402
WriteIODWord C_PCIConfAddr 80000064
WriteIODWord C_PCIConfData 08080808
WriteIODWord C_PCIConfAddr 80000068
WriteIODWord C_PCIConfData 33E82F00
WriteIODWord C_PCIConfAddr 8000006C
WriteIODWord C_PCIConfData 0000C003
WriteIODWord C_PCIConfAddr 80000070
WriteIODWord C_PCIConfData 381A1F20
WriteIODWord C_PCIConfAddr 80000074
WriteIODWord C_PCIConfData 01000055
WriteIODWord C_PCIConfAddr 80000078
WriteIODWord C_PCIConfData 2F900000
WriteIODWord C_PCIConfAddr 8000007C
WriteIODWord C_PCIConfData 00000010
```

IMPORTANT NOTE

```
'All SDRAMs contain an internal mode register which must be
programmed to
'enable the SDRAMs.This involves following a special sequence
involving the north-bridge
'and the SDRAMs. This sequence is not included in this
example program
```

```
BasicRAMTest 0 9FFFF #RAM_Fail '640k base
BasicRAMTest C0000 FFFFF #RAM_Fail 'Shadow RAM
BasicRAMTest 100000 3FFFFFFF #RAM_Fail '1M-6M
```

```
Gprint OK
End
```

```
#RAM_Fail
Rprint Failure
PrintError
EndError
End
```

Listing 3 - Memory Test

Listing 3 shows an example memory controller initialization and memory test.

Testing Input/Output Devices

Input/Output ICs can be sub-divided into two parts; the section that interfaces to the processor and its buses, and the section that interfaces to the I/O connector. See Figure 2, which illustrates this. Different test methods are used for each.

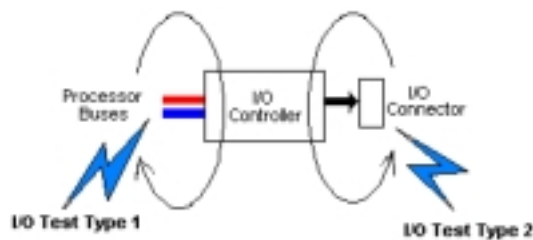


Figure 2 - I/O Types

**I/O Test Type 1:
Processor to I/O Controller**

Most I/O controllers contain internal registers, which can be accessed by the processor's read/write instructions. Using the TSL/1 commands WriteIOByte and ReadIOByte, test patterns can be written to these registers and their contents can be verified. To do this, use the port address of a read/writeable register, and write and verify a series of patterns such as AA and 55 to this port. Listing 4 shows an example where the presence of the serial port controller is verified.

```
Serial Port Controller Access

'Serial Port 1
WriteIOByte 3FF AA ' Write test pattern
ReadIOByte 3FF FF AA #ACCESS_ERROR_1 ' Verify test pattern
WriteIOByte 3FF 55
ReadIOByte 3FF FF 55 #ACCESS_ERROR_1

Gprint OK
End

#ACCESS_ERROR_1
Rprint Failure Serial 1 - Access Controller
PrintError
EndError
End
```

Listing 4 - I/O Access Check using Read/Write Register

Alternatively, if the chip has an ID register, just verify the ID using a read I/O command. Listing 5 shows an example.

```
South Bridge ID Check

DefineConst C_PCICnfAddr 0CF8
DefineConst C_PCICnfData 0CFC

'Verify the presence of the PCI-ISA's PCI ID
'Select SouthBridge using its PCI configuration address
WriteIODWord C_PCICnfAddr 80003800
'Read back and verify SouthBridge ID 71108086
ReadIODword C_PCICnfData FFFFFFFF 71108086
#PCIISA_ID_Fail

Gprint OK
End

#PCIISA_ID_Fail
Rprint ID Failure
PrintError
EndError
End
```

Listing 5 - I/O Access Check using ID Check

More detailed diagnostic information can be derived using the command:

```
IOBusTest8 Port_Addr Mask [#Label]
```

See the µMaster help files User Guide for more information.

**I/O Test Type 2:
I/O Controller to I/O Connector**

This test must verify that the I/O controller can communicate with the external device under its control.

It consists of the following steps:

1. Initialize the I/O controller to enable it to communicate with the external device. This involves writing specific values to some of the I/O Controller's internal registers.
2. Send and verify data to the external device.
3. Receive and verify data from the external device. Data is sent and received to/from the external device using some of the I/O Controller's internal registers.

Data to/from the external device is verified by using one of the following methods:

1. A real peripheral such as a disk drive is connected, and data is written, then read back and verified.
2. A peripheral emulator is used to emulate the operation of the external device's interface, therefore allowing verification of all signals on the I/O connector.
3. Loopback plug
4. An external instrument is used to generate or measure test signals (e.g. an oscilloscope is used to verify a generated audio signal).

Listing 6 shows an example of a serial port test. A loopback plug is used to feed back the Tx line to the Rx line.

```

Serial Ports Tx/Rx

'Program serial controller mode registers for appropriate bit rate,
number data bits, number stop bits etc.
WriteIOByte 3FB 80
WriteIOByte 3F8 06
WriteIOByte 3F9 00
WriteIOByte 3FB 03

'Transmit a character
WriteIOByte 3F8 AA

'Receive and verify transmitted character via loopback
ReadIOByte 3F8 FF AA #TX/RX_ERROR_1

'Transmit and verify another character
WriteIOByte 3F8 55
ReadIOByte 3F8 FF 55 #TX/RX_ERROR_1

Gprint OK
End

#TX/RX_ERROR_1
Rprint Failure Serial 1 - Tx/Rx
EndError
End
    
```

Listing 6 - I/O Test

Programming Example

Here is a sample test for the board in Figure 1.

Functional Area	Test
General Bus Test	Use TSL/1 BusTest Command on boot area (i.e. "BusTest 000FE0000 000FFFFFF #label").
North Bridge	<ul style="list-style-type: none"> • Verify IDs • PCI Configuration Register Test.

SDRAM	<ul style="list-style-type: none"> • Initialize NorthBridge PCI Configuration Registers. • Initialize SDRAM mode register. • Use TSL/1 RAM test commands.
Video	<ul style="list-style-type: none"> • Perform Video Controller Register Test. • Initialize Video Controller. • Test Video Memory using TSL/1 RAM test commands. • Output pattern to screen.
South Bridge	<ul style="list-style-type: none"> • Verify IDs. • PCI Configuration Register Test. • DMA Controller Register Test. • Interrupt Controller Register Test. • CMOS RAM/RTC Test. • USB Test. • IDE Test. • SM-Bus Test (read SDRAM eeprom).
Network	<ul style="list-style-type: none"> • PCI Configuration Register Test. • Transmit and Receive packet of data via external loopback.
Boot ROM Test	CRC using TSL/1 RomTest command.
Super I/O	<ul style="list-style-type: none"> • Configuration Register Test. • Keyboard Controller Test. • Mouse Controller Test. • Serial Port(s) Test. • Parallel Port Test. • Floppy Port Test (read/write to drive).
Audio	<ul style="list-style-type: none"> • Configuration Register Test. • Generate waveform on speaker-out. • Feedback and measure generated signal on line-in and mic-in.

Many of the tests described are supplied as example test programs with the μ Master installation software. These board files can be found in the root directory in which μ Master has been installed. Board files have the extension .brx. They are opened from μ Master's File menu. The following board files should be available:

Example_X86.brx
Example_NonPC.brx

TSL/1 Error Handling

A label can be specified as an optional parameter for many TSL/1 commands. This label identifies a piece of code that will run if the command fails (e.g. if a RAM test fails the code identified by the label is run). The following shows a simple error handling routine:

```
#My_Error  
Rprint This failed due to...  
PrintError  
EndError  
End
```

#My_Error is the label identifying the error handling routine. Any combination of characters can be used, but the first character must be #.

Rprint displays on screen the string following the command in a red font.

PrintError prints detailed, internally generated error information, such as:

- which data bus lines failed.
- what port access caused the failure.
- etc., etc.

EndError denotes the end of the error handling routine.

End stops the test.

Listing 2, Bridge Test shows examples of error handling routines.

In addition to outputting information to the PC display, TSL/1 can send error information to

an external program (e.g. CVI) by using one of the following TSL/1 commands:

ReturnPassOrFail
ReturnUserMsg

Refer to the μ Master Help file for further information on these commands.

Useful websites for databooks.

The following links were valid at the time of writing this Application Note. We apologize if you find them no longer accessible.

Adaptec:
<http://www.adaptec.com/>

Ali Corporation:
<http://www.ali.com.tw/>

AMD:
<http://www.amd.com/>

ARM:
<http://www.arm.com/>

ATI Technologies Inc.:
<http://www.ati.com/>

Atmel Corporation:
<http://www.atmel.com/>

Broadcom:
<http://www.broadcom.com/>

Cirrus Logic Inc.:
<http://www.cirrus.com/>

Conexant:
<http://www.conexant.com/>

Crystal Semiconductor (Cirrus Logic):
<http://www.cirrus.com/>

CSR:
<http://www.csr.com/>

ESS Technology Inc.:
<http://www.esstech.com/>

Freescale Semiconductor:
<http://www.freescale.com/>

Intel:
<http://developer.intel.com/>

LSI Logic:
<http://www.lsil.com/>

Marvell:
<http://www.marvell.com/>

Micron:
<http://www.micron.com/>

National Semiconductor:
<http://www.national.com/>

NVIDIA:
<http://www.nvidia.com/>

O2 Micro:
<http://www.o2micro.com/>

QLogic:
<http://www.qlogic.com/>

Realtek:
<http://www.realtek.com.tw/>

ServerWorks:
<http://www.broadcom.com/>

SigmaTel:
<http://www.sigmatel.com/>

Silicon Image:
<http://www.siliconimage.com/>

Silicon Laboratories:
<http://www.silabs.com/>

SiS:
<http://www.sis.com/>

SMSC:
<http://www.smsc.com/>

Texas Instruments:
<http://www.ti.com/>

TVIA:
<http://www.tvia.com/>

VIA Technologies Inc.:
<http://www.via.com.tw/>

Winbond:
<http://www.winbond.com/>

Contacts for additional information

European Sales and Support:

International Test Technologies,
Larkin House, Oldtown Road,
Letterkenny, County Donegal, Ireland.
Tel: +353 (0)749 188 100
Fax: +353 (0)749 188 128
E-mail: sales@intertesttech.com
Web: www.intertesttech.com

N. American Sales and Support:

International Test Technologies,
2694 21st. Avenue,
San Francisco, CA 94116
Tel: 415 753 5376
Fax: 415 753 3635
E-mail: sales_usa@intertesttech.com

Asian Sales and Support:

International Test Technologies,
32 Maxwell Road,
#03-07 White House,
Singapore 069115.
Tel: +65 9642 3164
E-mail: sales_asia@intertesttech.com

For contact details of our worldwide reps.
please see our website:

http://www.intertesttech.com/ate/company_locations.htm