INTERGRATOR i PCI 16 HARDWARE REFERENCE AND CONTROL PROGRAM DEVELOPERS GUIDE

Trademarks
The Comtrol Europe logo and the INTERGRATOR i/PCi logo are trademarks of Comtrol Corporation.
InterGrator is a registered trademark of Comtrol Europe Ltd
Rocket Port is a registered trademark of Comtrol Corporation
WinDriver is a registered trademark of Jungo Ltd (formerly KRFTech Ltd)
Microsoft, MS-DOS, MS, Windows, Windows NT, Windows 95, Windows 98, MSN, MSDN and Visual C++ are registered trademarks of Microsoft Corporation.
Borland, Turbo C++, Borland C++ and Turbo Debugger are registered trademarks of Borland International.
V53 is a registered trademark of NEC Corporation
"Acrobat® Reader Copyright © 1987-2000 Adobe Systems Incorporated. All rights reserved. Adobe and Acrobat are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions"

Product names mentioned in this document may be trademarks and/or registered trademarks of their respective companies.

The Comtrol Corporation Headquarters are in
Maple Grove, Minnesota, USA

The Comtrol Europe Ltd Headquarters are in
Launton, Bicester, England

Comtrol reserves the right to make product and user guide changes without notice

This Guide.
Comtrol Europe Ltd Document Number 981101/PV
Release Rev D, 8th August 2000

Contents

Appendices

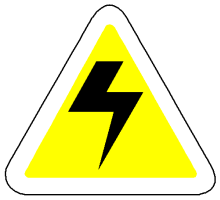
Figures

Tables

1 Introduction

1.1 Who Should Read This Guide

This guide is intended to provide hardware reference information and to assist software engineers in the development of downloadable control programs for the InterGrator i/PCi intelligent serial communications controller from Comtrol Europe Ltd.



Observe the warnings provided in this guide.

Observe anti static precautions

1.2 Conventions and Terminology Used in This Guide

It is important that you read this section.
The term '*Control Program*' refers to the software download by the host computer to the InterGrator i/PCi. The Control Program code is executed by the on-board NEC V53 microprocessor.

The host system software refers to software running on the host computer in which the InterGrator i/PCi controller is installed.

2.0 InterGrator i/PCi Technical Description

The InterGrator i/PCi is an intelligent serial communications controller.

2.1 InterGrator i/PCi Key Features

The following points summarise the key features of the InterGrator i/PCi ;

- NEC V53 processor.
- 1Mb dual port ram (SRAM).
- 16 serial ports (8 x dual channel Zilog Z85230 ESCC's).
- Multiplexed connection to external Interface box.
- Four channel DMA controller (configured as two full duplex DMA channels).
- Serial connection using either the Comtrol Europe Universal Interface (Version 3) or Comtrol Rocket Port Interface box.
- EPROM resident Turbo debugger remote kernel.
- System independent controller reset.
- PCI Bus V2.1 compliant target (PLX9052 PCI Interface chip).
- EPROM start-up and POST firmware.
- Easy upgrade path for existing Comtrol Hostess i 8/16 users.

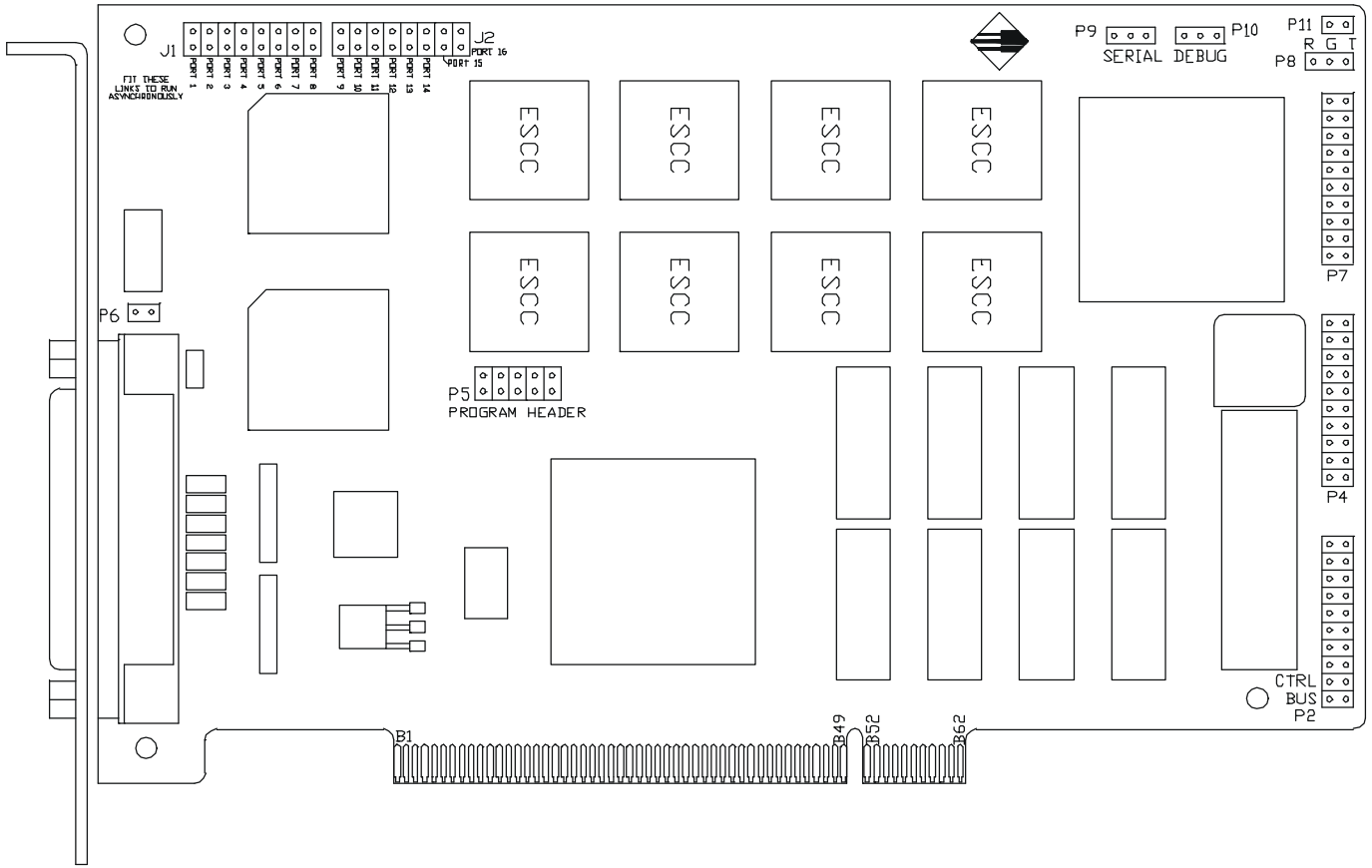


Figure 1 - Comtrol Europe Ltd - InterGrator i/PCi

2.2 InterGrator i/PCi Architectural Overview

The InterGrator i/PCi is an intelligent PCI bus based controller. The PCI interface is implemented as a V2.1 compliant 32 bit slave using the PLX9052 PCI bus interface chip.

The host computer's BIOS communicates with the PLX9052 in order to determine the resources that the InterGrator i/PCi controller requires. The InterGrator i/PCi controller requests a 1Mb memory address space for the dual port ram and 128 byte memory address space for the PLX9052 Run Time Registers (RTR).
Additionally, a 128 byte I/O address space is requested as the PLX9052 run time registers are also mapped into host computer system I/O space.

The allocated resources information is read by the host computer's device driver's initialisation routine. The method used varies between operating systems.

The 1Mb of Dual Port Ram is the main communication channel between the Control Program and the host computer device driver. The Control Program will build data structures such as message queues and FIFO's to effect the transfer of data across this interface.

Considering now, the InterGrator i/PCi on board components. The InterGrator i/PCi processor is the NEC V53 operating at 12Mhz. This may be factory upgraded to a 20Mhz.
This InterGrator i/PCi processor option allows a straightforward upgrade path for users of the existing Comtrol Hostess i 8/16 ISA controller.

The V53 processor shares many of the real mode attributes of the Intel 80x86 processors. It is binary compatible (the instruction set is compatible with the 80x86 in real mode) and register set compatible with the familiar 80x86 segmented architecture. It supports separate memory and I/O address spaces.

The V53 processor is operated in its normal address mode which is analogous with 80x86 real mode with a maximum memory address range of 1Mb. The I/O address range is 64Kb.
The V53 processor supports internal I/O mapped peripherals including a four DMA channel controller, eight channel interrupt controller, three timers and a UART. The I/O address mapping for these devices is set up by the Boot EEPROM.

The sixteen serial channels are implemented using eight dual channel multi-protocol Enhanced Serial Communications Controllers (Zilog Z85230 ESCC, or equivalent).

The Zilog Z85230 ESCC user guide is available in PDF format from :

http://www.zilog.com

Refer to the Z85230 user guide for programming details.

The ESCC I/O signals are routed via a multiplexed data link to an external interface unit. Either the InterGrator Universal Interface or Rocket Port interface can be used. These interfaces handle the signal conversion and level shifting to RS232, 422, and other physical protocols.
The operation of the multiplexed data link is transparent to the software developer.

2.2.1 Dual Port Ram

The Dual Port Ram is the main communication channel between the InterGrator i/PCi and its host computer's device driver. Implemented using 1Mb of SRAM. The Dual Port Ram is addressable from both the V53 processor and the host computer. Certain address ranges are reserved and this is shown in Figure 2. Note that the upper 64K is reserved for the boot EEPROM.
The host computer can address all the InterGrator i/PCi controller's memory. In effect the host computer's access to InterGrator i/PCi memory resource is through a 1Mb window.

As the InterGrator i/PCi is a PCI controller, allocation of resources is made by the host computer's system BIOS at boot time based on configuration information held in the InterGrator i/PCi's on board 'CONFIG EEPROM' and the current host computer configuration.

InterGrator i/PCi Memory Map

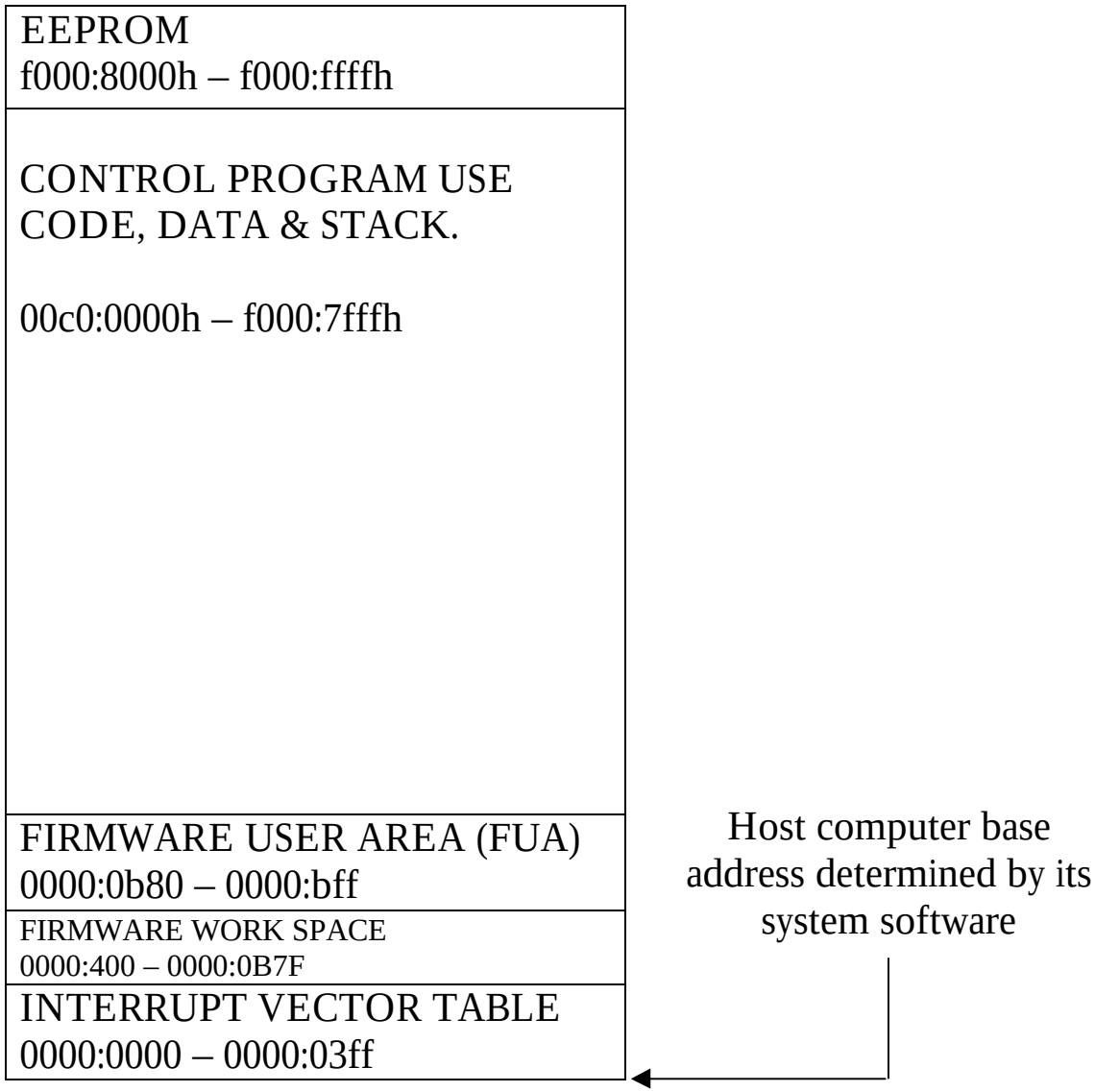


Figure 2 - InterGrator i/PCi Memory Map.

Notes :

1. The address ranges shown are within the V53 processor memory map. Host addressing is dependent on where the host computer 'allocates' the Dual Port Ram in system address space. Your device driver obtains the address from calls to the operating system. The specific details of obtaining the Dual Port Ram system base address and other resources (I/O base address and IRQ) vary between operating systems. Our sample software gives one example.

2. The Firmware User Area (FUA) is also written to address 1000:0000 (within the V53 processor memory map). This is to assist users of early version Hostess i8/16 controllers (identified by a 3 way DIL switch) and also Smart Hostess users which only supported the FUA at this address. More information can be found in Chapter 7.

2.2.2 Interface Choices

The InterGrator i/PCi offers a range of interfaces via either the InterGrator UI (Version 3) or Rocket Port 16 Interface units.

The interface options provided by the InterGrator UI (Version 3) or Rocket Port 16 are described in Chapter 3 and the pin-outs are tabulated in Chapter 8.

3.0 The InterGrator Product Family

3.1 Comtrol InterGrator UI

This section briefly describes the InterGrator UI.

The InterGrator UI is an eight port 19" rack mounted or stackable interface for the InterGrator i/PCi and RocketPort range of multiport serial controllers. Two InterGrator UI units are daisy chained to support 16 ports.

Three versions of the UI exist. The InterGrator i/PCi is designed to work with the *Version 3* Universal Interface.

(Note : UI Versions 1 and 2 can be used but with reduced functionality and no daisy chaining therefore limiting the InterGrator i/PCi to 8 ports only. All future references to the UI in this document infer the VERSION 3 product).

Each DB25 female port on the InterGrator UI can be independently selected as RS232, RS422, RS423, RS485 or Current Loop. Furthermore, in RS232 mode the InterGrator UI can be configured to support synchronous mode clocks (DCE sourced RX clock and DTE sourced TX clock).

The InterGrator UI connects to the InterGrator i/PCi by way of a Comtrol supplied 25 way cable. The cable is of the same type used by the Comtrol Rocket Port controllers and is specifically designed for the high speed multiplexed data link.

The user should refer to the InterGrator UI hardware reference guide for details of the internal configuration jumpers of this interface and how to configure for the different physical protocols.

3.2 Rocket Port 16 Interface

This section briefly describes the Rocket Port Interface.

The Rocket Port interface is a 16 port low profile interface. It connects to the InterGrator i/PCi using the Comtrol supplied 25 way cable.

This cable is specifically designed for the high speed multiplexed data link

The Rocket Port 16 interface is available in two versions. These are RS232 only and RS232/RS422. The selection of physical protocol on the RS232/RS422 interface is by slider switch accessible without removing the cover.

The serial I/O connectors are DB25F.

The user should refer to the Rocket Port interface documentation for full details of this product.

3.3 Comparison Between the InterGrator i/PCi & Hostess i

This section is relevant to Hostess i 8/16 developers migrating to the InterGrator i/PCi. It may also provide useful background information for developers new to Comtrol products. Hostess i8/16 users should also refer to Chapter 7.

The InterGrator i/PCi was designed so that existing Hostess i 8/16 developers could easily port their control programs to the new card. Comtrol have strived to keep the differences from a control program developers perspective to a minimum.

The control program runs on the InterGrator i/PCi (Hostess I) controller. Host computer software in the form of a device driver is needed to provide the interface to application software via the operating system.

As the InterGrator i/PCi is a PCI bus based card, changes will be required to existing Hostess i 8/16 drivers to allow them to work with the new InterGrator i/PCi controller.

With the Hostess i 8/16, a base I/O address (within host system I/O space) was set on DIL switches prior to installation. This defined the base address of four registers, two of which were used to access indirect registers which defined the Dual Port Ram base address etc. Therefore, the Hostess i 8/16 used a proprietary software configuration scheme.

This was documented in the Hostess i 8/16 developers guide. Basically the driver wrote to the controller I/O addresses to set the Dual Port Ram base address (in system memory space), window size & offset. The IRQ (if used) was also software selected.

The InterGrator i/PCi is a PCI controller. The controller's memory, I/O addresses and IRQ are software configured by the host computer system BIOS in response to resource requirements programmed into the InterGrator i/PCi configuration EEPROM. These bus transactions are transparent to the developer.
However, the developer must obtain the allocated resources from the host computer. The mechanism to achieve this varies and is device driver dependent.
Protected mode operating systems that are PCI aware usually provide kernel support routines to assist and these are documented in the relevant Device Driver Kit (DDK).

The accompanying sample software uses the Jungo WinDriver package which provides kernel drivers for Windows 95/98 and Windows NT. The function FindIntergrator() is an example of how to call WinDriver, which in turn calls the host operating system, to get allocated resources.

3.3.1 Point by Point Comparison

There now follows a brief bullet point comparison between the InterGrator i/PCi and the Hostess i 8/16.

- Same V53 processor but InterGrator i/PCi can be factory upgraded to operate at 20Mhz.
- V53 supported timers, DMA, Interrupts are the same on both cards including I/O addressing.
- InterGrator i/PCi uses Zilog 85230 ESCC, which is upward compatible with the Hostess i 8/16 AM85C30 SCC.
- InterGrator i/PCi is equipped with 1Mb of SRAM Dual Ported as standard. The Hostess i 8/16 is fitted with 128Kb of DRAM Dual Ported as standard.
- The InterGrator i/PCi uses active external Interface boxes (Universal Interface or Rocket port 16 interface) to support a wide range of physical protocols.
- The connection is by way of a multiplexed data link. The connecting cable is 25 core (shielded twisted pair + power). Standard DB connector are supported.
- The Hostess i 8/16 connects to a passive interface via a thick 100 core cable
- Both controllers support the Turbo debugger remote kernel. However, the InterGrator i/PCi uses the V53 processor UART for communications with the remote PC. The supported version of Turbo debugger is 3.1.
- The terminal debugger is supported on both cards. Access is via Port 0 (default).

- Both controllers support external H/W Debug/Reset pushbutton controls.

- The Hostess i 8/16 used a DIL switch to set an I/O base address and a proprietary software configuration scheme to configure the Dual Port Ram.
  The InterGrator i/PCi is a PCI card, the host computer system BIOS and the driver must obtain the allocated resources from the operating system as explained in the relevant DDK.

- The Hostess i 8/16 was supplied with 128K Dual Port Ram as standard. Host computer access was via memory windows of 16, 32 or 64K in size (Under 1Mb addressing). If configured above 1Mb then the size would be 128K.
  The InterGrator i/PCi is supplied with 1Mb of Dual Port Ram. The host computer can access the whole of this memory via, in effect, a 1Mb window.

4.0 InterGrator i/PCi Controller Configuration and Installation

Take care when you handle the InterGrator i/PCi, like any electronic device, it is sensitive to static electricity. Use normal anti static precautions such as wearing an earth ground strap.

The InterGrator i/PCi is a PCI bus based controller and takes up one PCI slot.

Configuration of memory, I/O and IRQ is achieved through software. The only hardware configuration is to set the serial ports sync/async operational mode.

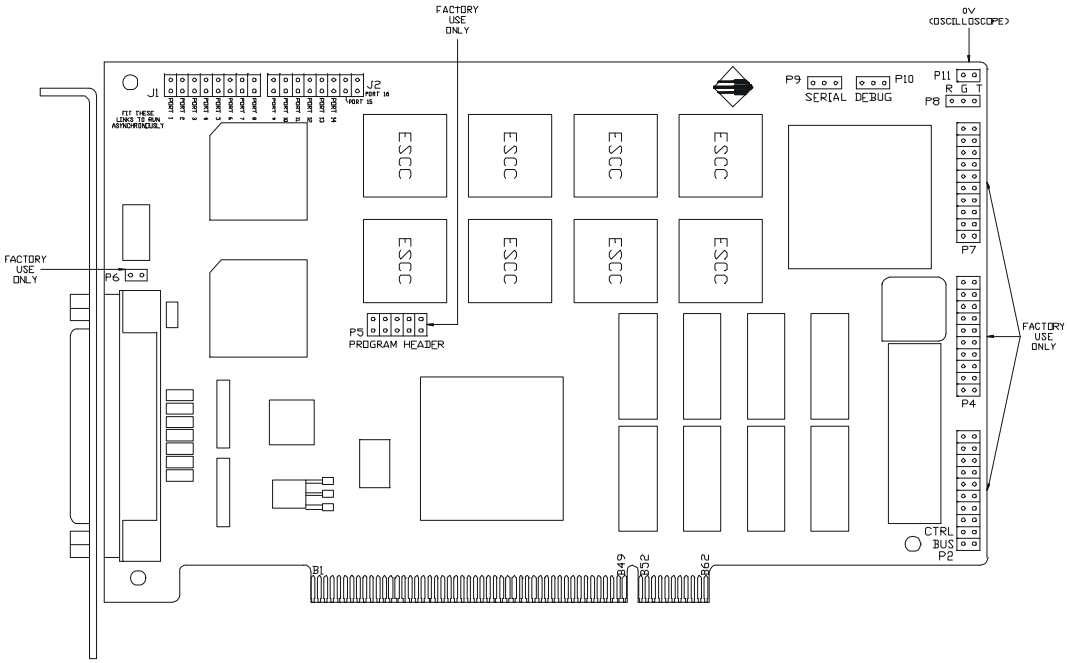


Figure 3 - Factory Use Jumpers

4.1 Configuration of Memory, I/O and IRQ

The configuration of the memory, I/O and IRQ is done by software. There are no user configurable switches or jumpers to set to configure the memory, I/O or IRQ resources.
The InterGrator i/PCi controller resource requirements are programmed into a configuration EEPROM. At reset the EEPROM contents are read automatically by the PLX 9052. When the host computer boots, it will read the resource requirements from the InterGrator i/PCi controller and make allocations based on the host computer system configuration.

The host software is responsible for reading the InterGrator i/PCi controller configuration from the operating system or BIOS.

The host computer system BIOS contains primitives that can be called to assist drivers in this task. However, in protected mode operating systems it is normal and far easier to make use of operating system supplied calls. *As the exact methods vary from operating system to operating system it is not feasible to cover every different method*. However, the sample software does give a typical example.

4.2 ASYNC/SYNC Configuration (J1 & J2)

Each port on the InterGrator i/PCi can be configured to operate in

(i) Async communications mode.
(ii) Sync communications mode with separate clocks.

When using the Rocket Port Interface only option (i) should be used.

Both options (i) and (ii) can be used with the InterGrator Universal Interface (version 3 model). The supported DB25 interface signals for each mode are shown in Chapter 8.

The essential difference is that the sync mode supports a transmit clock (from the InterGrator i/PCi) and a receive clock (to the InterGrator i/PCi). It sacrifices DTR and DSR in order to support these clocks. Also DCD is unavailable in this mode.

If using the InterGrator i/PCi for self clocked synchronous communications (clock embedded in data) set the ports to operate in async mode. The clock will be extracted from the data.

The selection of the communications mode is made by the jumper blocks, J1 - H/W ports 1 to 8 and J2 - H/W ports 9 to 16 *(note that the H/W port numbers are one greater than the software port numbers).*

The port numbers are printed on the InterGrator i/PCi controller PCB. For each port :

For Async use : FIT JUMPER
For Sync use : DO NOT FIT JUMPER

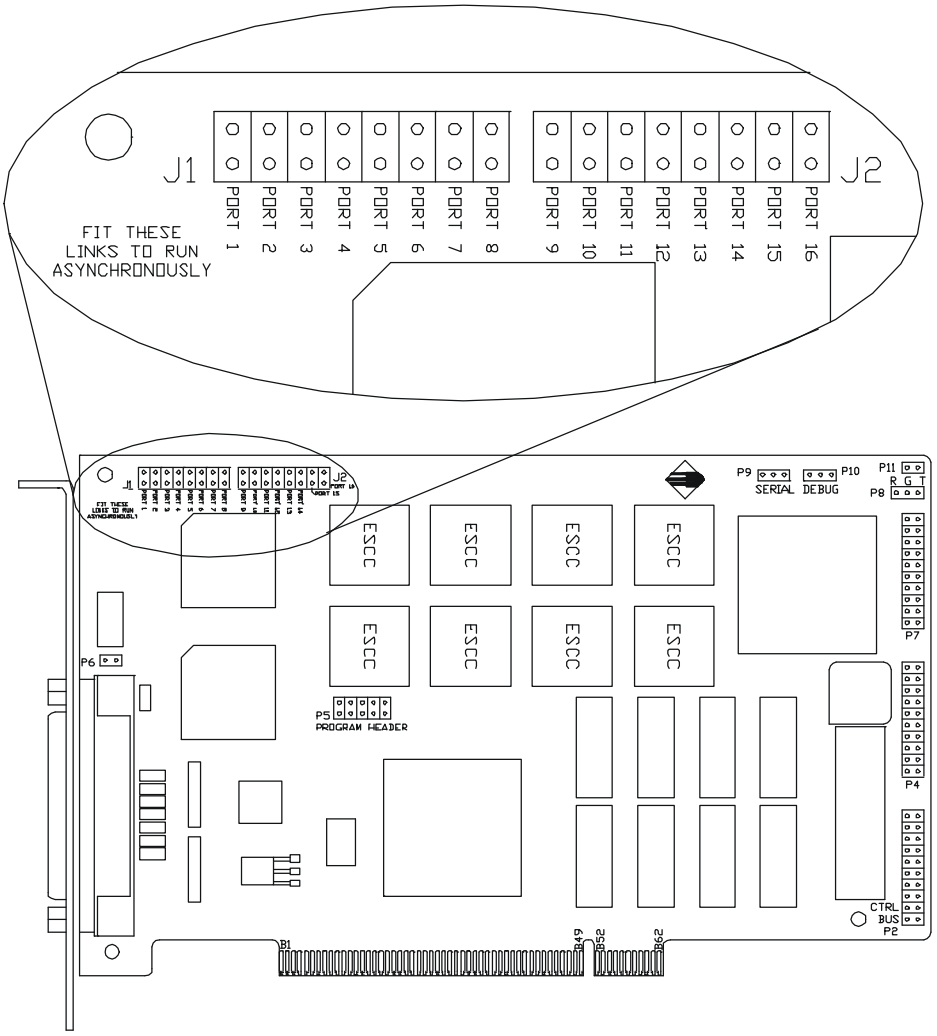


Figure 4 - Async / Sync Jumpers

IMPORTANT NOTE : The InterGrator Universal Interface (UI) has an equivalent set of sync/async jumpers which must be configured to match those of the InterGrator i/PCi. See the InterGrator UI manual.

4.3 Installation of the Debug Backplate

Comtrol provide a backplate on which is mounted a DB9M connector and two push button switches.
This backplate is used during Control Program development. The cables from the connector and switches are connected to headers on the InterGrator i/PCi.

The cable from the DB9M is connected to header P9. This is the serial port for remote debug connection.
The cable from the push button switches is connected to header P10. The DEBUG push button causes an NMI to be generated. The RESET button will reset the controller independently of the host system. Note that the control program is erased following a RESET.

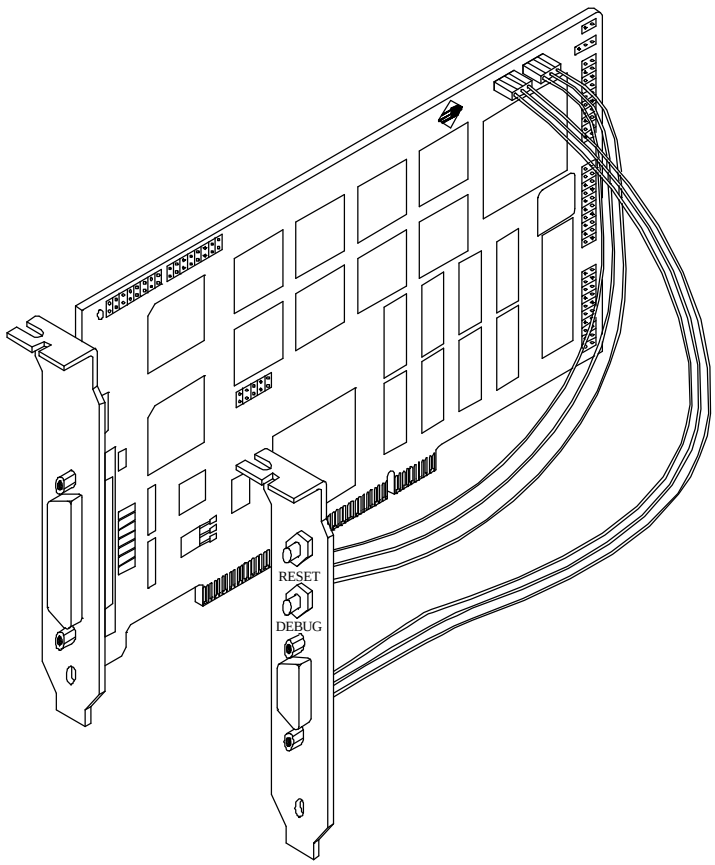


Figure 5 - Connection of Debug Backplate

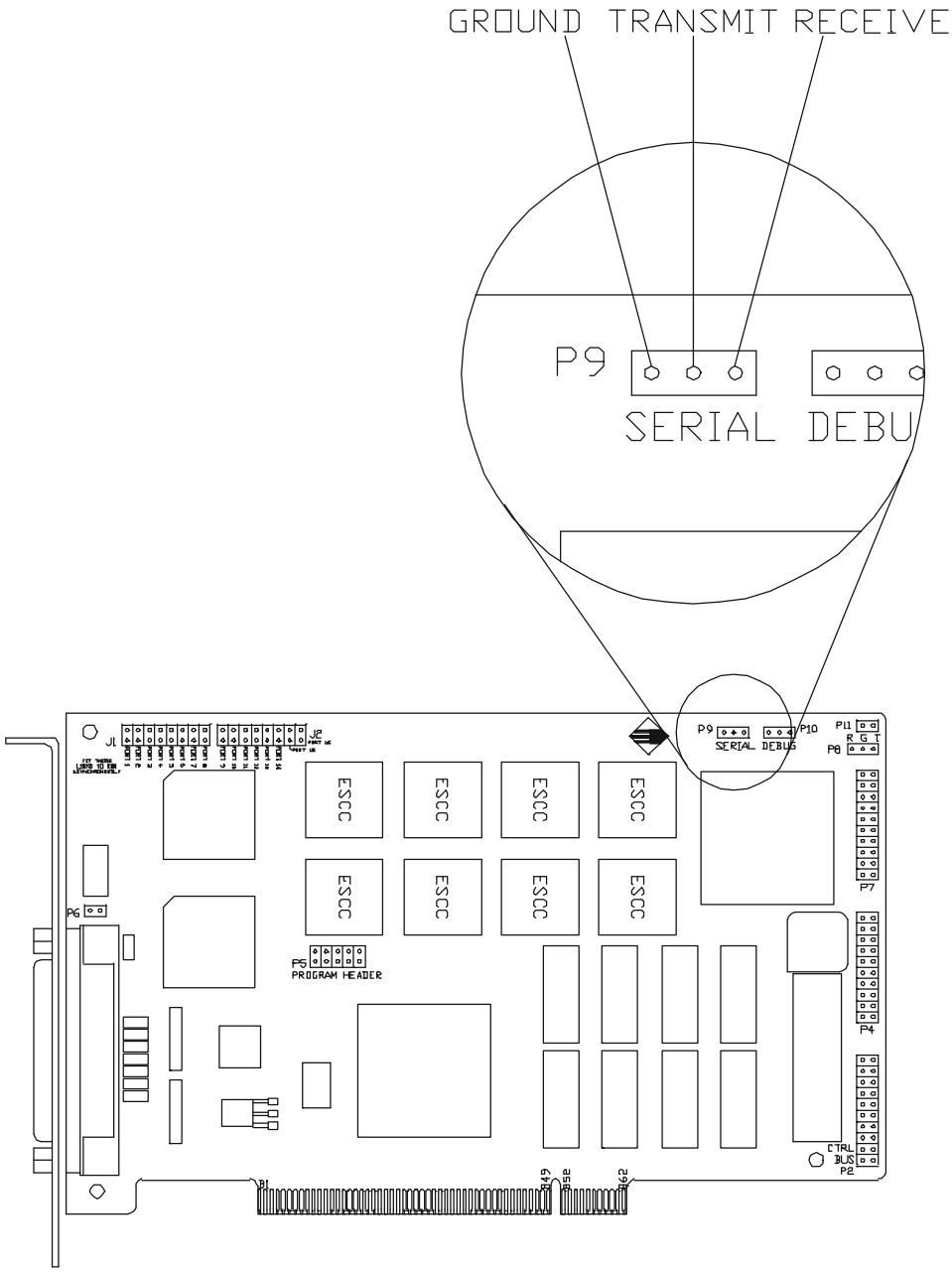


Figure 6 - V53 Processor Serial Header P9

The above pin-outs match those of the Hostess i 8/16 therefore existing cables can be used.

NOTE : Connector V53 serial port connector P8 is for factory use only.

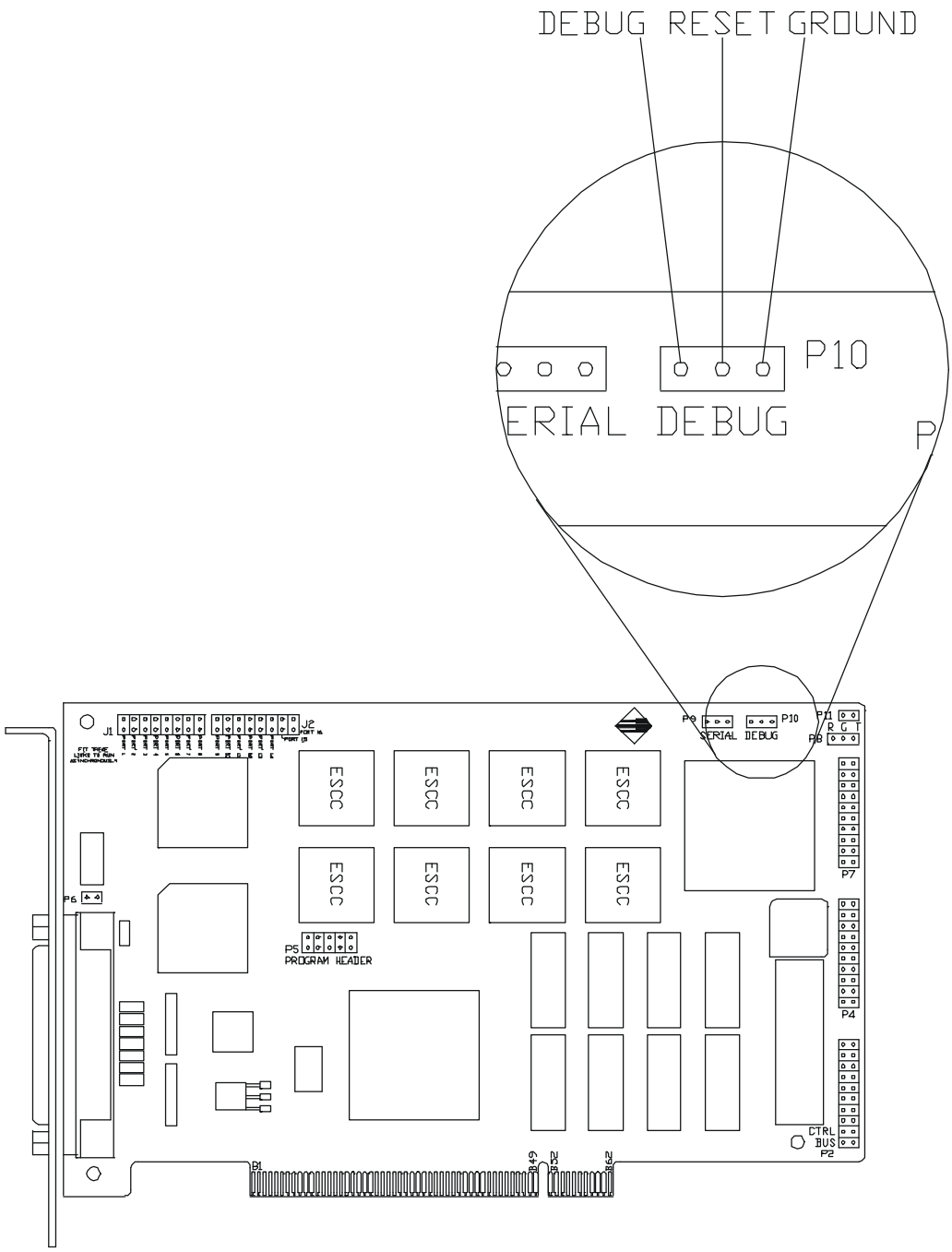


Figure 7 - Connection to Reset/Debug Header

5.0 Host Computer Software / Driver Development

IMPORTANT Note : Regarding port numbering convention.

The software convention numbers the ports starting at 0 (zero).

The Rocket Port Interface box H/W port numbering also starts from 0. However, the InterGrator i/PCi and InterGrator Universal Interface H/W references start at 1.

Thus, InterGrator product port H/W port number references are equal to the software port number reference + 1.

In addition to the InterGrator i/PCi Control Program files, host computer sample software is also available. The InterGrator i/PCi host computer software is designed to work under either Windows 95/98 or Windows NT.

Both chapters 5 & 6 discuss a Dual Port Ram data structure called the Firmware User Area (FUA). This has previously been referred to as the firmware data area.
The FUA is important for the host computer system software, firmware and Control Program. It plays an important role during the Control Program download process but knowledge of only a few of its member variables is required.

Details of the FUA can be found in Appendix A.

5.1 Windows 95/98 and Windows NT Sample Software

The Comtrol supplied host computer software comprises of a dialog based application developed under MS Visual C++, a static library and sample downloadable control program.

Obviously the application cannot access the hardware directly and so kernel drivers must be installed. These drivers are supplied with the sample software. The driver for Windows NT is *'windrvr.sys'* and the driver for Windows 95/98 is *'windrvr.vxd'*.

Both these drivers provide a generic interface to Dual Port memory and I/O on an adapter card through an IOCTL interface. The drivers support an API.

The drivers are written by Jungo Ltd and shipped with their WinDriver device driver development toolkit. Section 5.1.1 below gives a brief overview of this product.

Developers who wish to write their own kernel driver's will need to obtain the Microsoft Windows 95/98/NT DDK's (device driver kits) by obtaining a subscription to the Microsoft Developers Network (MSDN). The MSDN also provides sample kernel drivers.

The DDK's are not required when using the Jungo WinDriver toolkit.

5.1.1 Jungo's WinDriver Toolkit & Kernel Drivers

The WinDriver toolkit is not a Comtrol product, but is provided by Jungo Ltd (formerly KRFtech Ltd).

Jungo's website is at http://www.Jungo.com
At the time of writing, the UK distributor is "Grey Matter" whose website is at http://www.greymatter.co.uk.

The WinDriver package consists of kernel drivers, utilities, wizards, sample software and documentation.

A Jungo developed API is supported which allows user mode programs to access the Dual Port memory or I/O on both PCI and ISA adapter cards. In effect WinDriver allows developers to write and debug their interface software in user mode. WinDriver also supports "*kernel plug-ins*" for time critical code.

The host sample software is supplied with 30 day evaluation versions of the WinDriver physical drivers. If you wish to use WinDriver as the basis for your host computer software driver then you must purchase a registered version from Jungo. For the InterGrator i/PCi you will need to provide the PCI vendor and device ID's which are 11FE and 8001 respectively.

5.1.2 Relationship Between Software Layers

The InterGrator i/PCi sample host computer software uses the WinDriver supplied device drivers to handle the physical interface to the InterGrator i/PCi controller and therefore the Control Program.

At the next layer above a Comtrol library called SSCI.LIB implements an interface to the WinDriver API. The supported functions are described in Section 5.4 and include calls to get the allocated PCI resources, and perform writes and reads to/from memory and I/O.

At the top layer is the MFC based dialog application. The software architecture is shown in Figure 8 below.

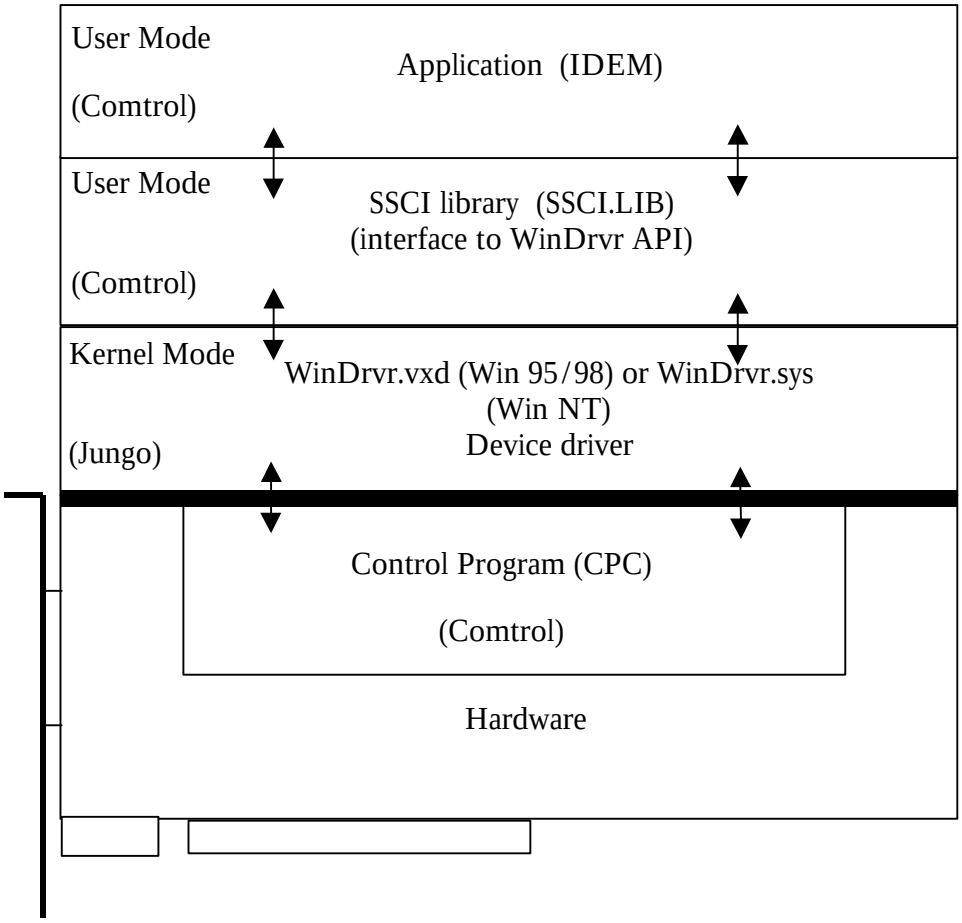


Figure 8 - Relationship Between Software Layers

5.1.3 Application Program Description

A Visual C++ MFC dialog based application is supplied. The project file is IDEM.DSW. This application is intended to provide developers with a basic environment suitable for control program debug and test. The source code gives examples of accessing the InterGrator i/PCi via the SSCI library.
The source to the SSCI library (available to registered WinDriver users) gives examples of calling the WinDriver API, for those developers wishing to work at this level.

The application uses the Jungo WinDriver to interface the hardware. The WinDriver API is encapsulated by the library SSCI.LIB. The functions supported by SSCI are described in Section 5.4.

The source code will provide the necessary insight into downloading control programs to the InterGrator i/PCi, starting the control program, reading/writing data structures, passing commands etc.
The sample software does not demonstrate the use of interrupt handlers with WinDriver which is discussed later.
The following discussions are not a line by line dissection of the sample software. Code fragments are introduced to illustrate the more important points.

Note: References will be made to the PLX9052 PCI Interface chip and certain registers contained therein (known as Run Time Registers or RTR). Two of these registers are important to the interface between the host system and the controller.
The IDEM source code (including the SSCI interface layer) "hides" much of the detail of these registers. Those developers not wishing to base their developments on the sample code and the underlying WinDriver device drivers should refer to Chapter 7 which provides a bit level description of the PLX registers.

5.1.4 InterGrator i/PCi Demo Program (IDEM).

The program is a dialog based MFC application. It serves two purposes ;

- Provides an environment for downloading and exercising a control program.
- Along with the source code for the SSCI library it provides the developer with programming examples on how to develop drivers using the WinDriver device driver development toolkit under Windows NT and Windows 95/98.
  This is its primary purpose.

For those developers familiar with the Hostess i 8/16 and its 'developers toolkit', IDEM and SSCI are broadly equivalent to HITERM and HILIB respectively.

The IDEM application user interface supports the features listed below.

- Automatically opens the WinDriver device driver.
- Reads PCI configuration data and initialises WinDriver for the InterGrator i/PCi.
- Allows the selection and downloading of a control program.
- Check box to allow Turbo Debugger remote kernel start-up.
- Check box to allow automatic update of RX data display, and CTS, DSR and DCD modem input signal displays.
- Open a serial port with selectable parameters.
- Write data (to opened port) command button (data entered into an edit box).
- Read data (from opened port) command button (display data in an edit box).
- Toggle RTS and DTR output modem control signals.
- Poll the states of CTS, DSR & DCD modem input control signals.
- Forces controller generated IRQ (for test purposes).
- Close the serial port.

Feedback is by way of message boxes. Developers can comment these out if wished.

5.2 Opening WinDriver and Getting the Controller Configuration

The IDEM application opens the WinDriver and acquires the controller configuration automatically by making calls to the SSCI functions sWinDriver_*Open* and *sFindIntergrator* in that order. The code can be found in the member CIdemDlg::OnInitDialog() and is also listed below.

sFindIntergrator returns with controller parameters including:

- The base memory address of the Dual Port Ram.
- The base address of the PLX9052 RTR registers in memory space
- The base address of the PLX9052 RTR registers in I/O space
- The allocated IRQ.

The Dual Port Ram is 1Mbyte in size. The system will place it somewhere within the 32 bit address space above the first 1Mbyte. The whole of the Dual Port Ram is accessible to the host computer system. Please note the description in Chapter 2 including the memory map diagram, Figure 2. There are reserved areas in low memory and the upper 64K is mapped to the EEPROM.

The PLX9052 RTR (Run Time Registers) are accessible through either memory address space or I/O address space. The IDEM application accesses them through memory address space. Only two of the registers are of concern to the developer.
These are the '*Interrupt Control/Status Register*' *(INTCSR)* at offset 48h and the '*User I/O, PCI Target Response, EEPROM, Initialisation Control Register*' *(CNTRL)* at offset 50h.

The INTCSR register is used to gate interrupts onto the PCI bus. The CNTRL register User I/O bits are used to issue reset, sysint and IACK signals to the InterGrator i/PCi controller. Both these registers will be dealt with in the download Section 5.3.

Important Note : *Both the INTCSR and CNTRL registers are 32 bit and multi-function. When programming them care must be taken to set or clear ONLY THE INTENDED bits. The others must be preserved otherwise malfunction will almost certainly occur.*

Full information on the PLX 9052 PCI interface chip can be obtained from the PLX web site http://www.plxtech.com

The parameters listed above are stored in a data structure of type ADAPTER_INFO.
The memory addresses are base addresses to be used when accessing the card using WinDriver. Jungo WinDriver refer to these as the 'transfer address' Accessing the memory locations involves adding the correct offset to the transfer address.

The following code fragment shows opening the WinDriver device and scanning for InterGrator i/PCi controllers.

```
//Open WinDriver and check For installed Intergrator i
if( !sWinDriver_Open( &ucWinDriver_Open_ErrCode ) )
{
   AfxMessageBox("WinDriver Open Failed",MB_ICONERROR,0);
   return( FALSE );
}
else
{
   AfxMessageBox("WinDriver Open OK",MB_ICONINFORMATION,0 );

   if( !sFind_Intergrator( adapter_info, &ucCardsInstalled ) )
   {
      AfxMessageBox("Intergrator i/PCI not installed",MB_ICONERROR,0);
      return( FALSE );
   }
   else
   {
       AfxMessageBox("Found Intergrator i/PCI",MB_ICONINFORMATION,0);
   }
 }
```

5.3 Resetting and Downloading

The IDEM application allows the user to download a control program to the InterGrator i/PCi controller by simply selecting a control program file, choosing whether or not to invoke the Turbo Debugger and clicking the DOWNLOAD button. This section describes the mechanism. The member function that handles the downloading is :

CIdemDlg::OnButtonLoad()

This member function can be found in the IdemDlg.cpp file.

The steps that are taken are :

- Set the Firmware User Area Interact flag to 0xAA55
- Reset the controller, completion indicated by the FUA Interact flag being toggled to 0x55AA by the InterGrator i/PCi controller firmware.
- Open the Control Program file.
- Read the Control Program and write to the Dual Port Ram.
- Set the FUA Interact flag to 0xAA55 for the second time
- Start the Control Program running, Good status indicated by the FUA interact flag being toggled to 0x55AA

5.3.1 Resetting

Although source code should be consulted for full details the following code fragments illustrate some important points. For example the reset code fragment from CIdemDlg::OnButtonLoad()

(note: Intergrator is an instance of CIdemApp)

```
//Reset the Intergrator i/PCi and poll for controller ready
Intergrator.Reset_V53( usThisCard );

for(i=0; i<30; i++)
{
       if( Intergrator.Controller_Ready( usThisCard ) )
      {
          break;
      }
      else
      {
          Sleep(1000);
          m_Progress_Load.StepIt();
      }
 }
```

The CIdemApp::Reset_V53() member function actually invokes the reset via the PLX9052 CNTRL register.

```
void CIdemApp::Reset_V53( USHORT usThisCard)
{
          //Forces a reset of usThisCard

          DWORD        dwDpramAddr,dw9052cntrlAddr,dw9052cntrlValue;

          //Reset the Intergrator i/PCi
          dw9052cntrlAddr = adapter_info[usThisCard].dwRtrTaddr + CNTRL;
          dwDpramAddr = adapter_info[usThisCard].dwTaddr;

          //Get copy of 9052 PCI interface chip control register
          dw9052cntrlValue = sMem_indword( dw9052cntrlAddr );

          //Reset the Intergrator i's on board V53 processor
          sMem_outword( dwDpramAddr + IFLAG,0xAA55);
          dw9052cntrlValue &= RESETON;
          sMem_outdword( dw9052cntrlAddr,dw9052cntrlValue );
          Sleep( 100 );
          dw9052cntrlValue |= RESETOFF;
          sMem_outdword( dw9052cntrlAddr,dw9052cntrlValue );
}
```

The above code fragment illustrates some important points regarding accessing the dual port ram and the PLX 9052 RTR registers using the SSCI library and WinDriver.

As already discussed the RTR registers are mapped into both memory and I/O address space. The IDEM application accesses the RTR registers solely through memory address space.
The Dual Port Ram is of course mapped into memory address space. The SSCI function *sFind_Intergrator()* gets the base addresses to these memory spaces (as well as other resources) and writes the values to member variables of the ADAPTER_INFO structure. These member variables are *dwRtrTaddr (PLX 9052 Run Time Registers) and dwTaddr (Dual Port Ram).* These addresses are virtual not physical addresses. They are of the form necessary for use with the WinDriver device driver.

Accessing the required memory location involves adding the offset of that location to either dwRtrTaddr or dwTaddr.

As an example (from the above reset code). The routine pre-sets the interact flag (word variable at the start of the firmware user area FUA) to 0xAA55 and after invoking the reset polls this location until either the flag changes (by the InterGrator i/PCi firmware) to 0x55AA (reset OK) or until the polling time expires.

```
dwDpramAddr = adapter_info[usThisCard].dwTaddr;
.
.
sMem_outword( dwDpramAddr + IFLAG,0xAA55);
.
.

if((sMem_inword( dwDpramAddr + IFLAG ))==0x55AA)
{
        break;
}
```

In the above statements, the local variable dwDpramAddr is set, the base address of the dual port ram. The offset to the interact flag IFLAG (absolute address offset 0x0B80) is added to and the combined address passed to the SSCI functions sMem_outword and sMem_inword. Once the reset has completed the Control Program can be written to the controller.

5.3.2 Downloading

The sample software loads the CPC.BIN Control Program binary at dwDpramAddr + OFFS_CPCSTART.
The value of OFFS_CPCSTART is 0x0C00. The InterGrator i/PCi onboard V53 processor views this address as segment:offset 00C0:0000h or physical address 00C000h.
The Control Program is transferred to the Dual Port Ram using the SSCI library function sBlock_Write_Mem().

sBlock_Write_Mem(*UCHAR *dest_addr, UCHAR *source_buffer,*
DWORD count)

It writes *count* bytes from the *source_buffer* to dual port ram starting at location *dest_addr*.

e.g.

```
sBlock_Write_Mem( dwDpramAddr + OFFS_CPSTART,pucTbuf,
                                                dwBytesToTransfer);
```

The sample IDEM application downloads the relatively small CPC.BIN (circa 6 Kbytes). It reads the whole file into a memory buffer and the sBlock_Write_Mem function writes it in one pass.

5.3.3 Starting the Control Program

Following download, the Control Program can be started. There are two methods of doing this dependent on whether the Turbo debugger remote kernel is to be started or not.

The Control Program is actually started by the function *System_int(),* which via a USER I/O bit in the PLX9052 PCi chip, will generate an interrupt into the V53 processor. The firmware based handler will then read the 'Command' byte location in the Firmware User Area (FUA). If the byte read is 01h then the command is start Control Program. The handler will read the segment offset from other reserved locations in the FUA.

The reserved locations concerned are :

0000:0BAEh - Command byte location

0000:0BB0h - 2 word start address in segment:offset format
First word = segment
Second word = offset

The host computer software must initialise these locations before calling *System_int* as shown in the following code fragment.

```
sMem_outword( dwDpramAddr + FU_CMD,EXEC );
sMem_outword( dwDpramAddr + FU_SEGARG,V53CPSTARTSEG );
sMem_outword( dwDpramAddr + FU_OFFARG,V53CPSTARTOFF );
```

If the Control Program is to be started in Turbo Debugger mode then the INT 27h command has to be jammed in at the start of the Control Program (i.e. at dwDpramAddr+0x0C000). Below is the code fragment. The original word at the start address needs to be saved first, then the opcode and operand needs to be written. Note that the ordering is INTEL or '*Little Endian*'.

The System_Int function invokes the interrupt, the Control Program starts and immediately enters the firmware resident Turbo Debugger remote kernel. The Control Program is now frozen awaiting the Turbo Debugger set-up procedure as described in Chapter 9.

```
if( fDebug_enabled == TRUE )
{
        wSavedword = sMem_inword( dwDpramAddr + 0x0C00 );
        sMem_outword( dwDpramAddr + 0x0c00,0x27CD );
        System_Int( usThisCard );
        Sleep(10);
        sMem_outword( dwDpramAddr + 0x0C00,wSavedword );
        return(TRUE);
}
```

The following code fragment shows the normal start-up and the use of the interact flag for handshake.

```
{
        sMem_outword( dwDpramAddr + IFLAG,0xAA55 );
        System_Int( usThisCard );
        //Check that flag is set
        for(i=0; i<10; i++)
        {
                Sleep(400);
                if( ( sMem_inword( dwDpramAddr + IFLAG ))== 0x55AA )
                        break;
        }

        if( i >= 10 && (sMem_inword( dwDpramAddr + IFLAG )) != 0x55AA )
        {
                return(FALSE);
        }
        else
        {

                return(TRUE);
        }
}
```

The complete member function is listed below.

```
BOOL CIdemApp::Start_ControlProgram( USHORT usThisCard,BOOL
fDebug_enabled )
{
        DWORD dwDpramAddr;
        WORD  wSavedword;
        int         i;

        dwDpramAddr = adapter_info[usThisCard].dwTaddr;

        //Start the control program
        sMem_outword( dwDpramAddr + FU_CMD,EXEC );
        sMem_outword( dwDpramAddr + FU_SEGARG,V53CPSTARTSEG );
        sMem_outword( dwDpramAddr + FU_OFFARG,V53CPSTARTOFF );

        if( fDebug_enabled == TRUE )
        {
                wSavedword = sMem_inword( dwDpramAddr + 0x0C00 );
                sMem_outword( dwDpramAddr + 0x0c00,0x27CD );
                System_Int( usThisCard );
                Sleep(10);
                sMem_outword( dwDpramAddr + 0x0C00,wSavedword );
                return(TRUE);
        }
        else
        {
                sMem_outword( dwDpramAddr + IFLAG,0xAA55 );
                System_Int( usThisCard );
                //Check that flag is set
                for(i=0; i<10; i++)
                {
                        Sleep(400);
                        if( ( sMem_inword( dwDpramAddr + IFLAG ))== 0x55AA )
                                break;
                }

                if( i >= 10 && (sMem_inword( dwDpramAddr + IFLAG )) != 0x55AA )
                {
                        return(FALSE);
                }
                else
                {

                        return(TRUE);
                }
        }
}
```

5.3.4 Control Program Running

Once the Control Program is running, the original firmware based handler that was invoked following a host computer System_int call is replaced. The new Control Program handler is System_isr (See Chapter 6).
The Control Program handler implements the Control Program side of a command passing mechanism that is used to pass Open and Close commands to the Control Program from the host computer. This mechanism is extensible.

The member function *CIdemDlg::Port_open* illustrates the host computer processing. A command message is set up and passed to the member function *CIdemDlg::Qhosticmd* which writes the message to the dual port ram command queue before issuing an interrupt via *CIdemDlg::System_int*.

5.3.5 The CIdem::System_Int Member Function

The System_int function invokes an interrupt into the InterGrator i/PCi V53 processor.

It does this by setting and clearing USER I/O bit 2 in the PLX9052 CNTRL register. This causes an interrupt into the V53 processor PIC on Interrupt line.

This function is called during Control Program start-up and also when a command (e.g. To open a port) is passed to the Control Program.

```
void CIdemApp::System_Int(USHORT usThisCard)
{

        DWORD       dw9052cntrlAddr;
        DWORD       dw9052cntrlValue;

        dw9052cntrlAddr = adapter_info[usThisCard].dwRtrTaddr + CNTRL;
        //Generate the interrupt via the PLX 9052 PCI target I/F chip CNTRL register
        //Set SYSINT bit
        dw9052cntrlValue = sMem_indword( dw9052cntrlAddr );
        dw9052cntrlValue |= SYSINT;
        sMem_outdword( dw9052cntrlAddr,dw9052cntrlValue );

        /*Clear SYSINT bit  NOTE on fast machines eg. 600Mhz a delay may be
        needed before clearing the SYSINT bit*/
        dw9052cntrlValue = sMem_indword( dw9052cntrlAddr );
        dw9052cntrlValue &= NOSYSINT;
        sMem_outdword( dw9052cntrlAddr,dw9052cntrlValue );
}
```

5.4 SSCI Library Functions

This section describes the supported SSCI library functions (at time of publication).
(SSCI = System to Serial Controller Interface)
The functions fall into several groups.

- Initialisation.
- I/O transfers.
- Memory (Dual Port Ram) Read/Write.
- Block Transfers.
- Terminate (Close).

A description of these functions may found in Appendix B.

6.0 Control Program Development

The InterGrator i/PCi is an intelligent serial controller with an embedded V53 processor, 1Mb of Dual Port Ram, Enhanced Serial Communications Controllers (ESCC's) and miscellaneous registers.

Obviously it requires software to operate. The standalone embedded processor design implements this software within non-volatile memory such as an EPROM or Flash memory. The software is then referred to as *firmware.*

Whilst the InterGrator i/PCi has an embedded processor, it is not a standalone design and connects to the host computer via the PCI bus. The InterGrator i/PCi controllers Dual Port Ram is visible in the memory address spaces of both the on-board V53 processor and that of the host computer.

This design allows the host computer's device driver to write or "*download*" the InterGrator i/PCi controller's software, known as the "*control program*", directly to the Dual Port Ram. Once the download is complete the host computer must start the Control Program running.

This "*soft load*" feature has the advantage that an updated or even a new Control Program can be easily distributed. If the control program were EPROM based then any changes would require the fitting of a new hardware component.

Despite the downloadable Control Program, the InterGrator i/PCi still has firmware in the shape of the boot EEPROM. Rather like the host computers BIOS, this EEPROM contains primitives that perform a basic initialisation and power up self-test. It also contains the handlers that support Control Program start-up and debug tools.

The firmware also builds the so called *"firmware user area"* (FUA) in Dual Port Ram at 0000:0b80h. The FUA is important to the download and start-up procedure.

See Chapter 5, host software/driver development for information on downloading.

6.1 Control Program Sample Software

I cannot be prescriptive about the design of Control Programs. The actual software design depends on the nature of your requirements. Ultimately the design rests with you.

However, Comtrol does provide sample software including the source to a Control Program. Previous Hostess i 8/16 users will recognise this Control Program as the CPC control program. It was also supplied as part of the Hostess i developers toolkit.

The Control Program is written in 'C', although there is an 80x86 assembly language start-up routine. Developers will probably not need to modify the start-up routine.
Hostess i 8/16 users will also remember an MS-DOS down-loader called DPLOADER as well as host computer software called HITERM (and it's library HILIB) which acted as a simple 'device driver'. Together the software allowed developers to learn the basics of working with the controller and Control Program development.

This Control Program runs unchanged on the InterGrator i/PCi. One of the development goals was to provide a migration path for existing Hostess i 8/16 users.
However, that as CPC does not generate an IRQ to the host computer (although the Hostess i 8/16 supported the feature), the InterGrator i/PCi controller must be polled.

The InterGrator i/PCi does allow the Control Program to invoke a system interrupt and the mechanism will be described in due course.

6.2 Sample Control Program Description

The sample Control Program software comprises the following files :

- CPC.C Control Program main module
- CPCSTART.ASM Control Program start-up module
- CPC.H Control Program header file
- DPRAM.H Control Program header file
- FIRMUSER.H Control program header file
- FIRMUSER.EQU Start-up module equates
- CLOCATE.EXE Control Program locator
- MAKEFILE Builds Control Program
- CPC.BIN Control Program binary image (o/p from MAKEFILE)
- CPC.TDS Control Program symbol table (for TurboDebugger)

In addition the MAKE process produces intermediate object files.

The V53 processor is a 16 bit processor whose instruction set is compatible with the Intel 80x86 in real mode.

The sample Control Program was developed using 16 bit Borland C/C++ V3.1 Professional edition. In addition to the C/C++ compiler, the package includes TASM, Tlink, Turbo Debugger and other utilities including MAKE.

Borland C/C++ software tools.

- Borland C/C++ 3.1
- Borland TASM 3.1
- Borland TLINK 5.1
- Borland TDSTRIP 3.1
- Borland MAKE 3.6

The CLOCATE utility provided by Comtrol takes the executable file produced by the linker as input and coverts it to a binary image ready for downloading.

Assuming you are using the above software tools then the supplied makefile should be use for building the Control Program. It is compatible with Borland's MAKE utility version 3.6. This makefile ensures that the Control Program binary image is produced (CPC.BIN) and a debug symbol table (CPC.TDS).

The advantage of using these tools is that they are compatible with the version of the Turbo Debugger remote kernel embedded in the InterGrator i/PCi EEPROM. This allows source level debugging of Control Program code.

The above software tools are run under MS DOS or in a Windows 95/98 DOS session.

6.2.1 Control Program 'C' Start-Up Module, CPSTART.ASM

This assembler module contains the Control Program entry point. It sets up the data (DS) and extra (ES) segment and initialises the stack. The Control Program stack size is set to 2048 bytes. It is unlikely that developers will need to modify this module. However, once assembled, the modules object file must be linked to the Control Program.

The CPCSTART.OBJ must be the first module in the list of object files to link. The supplied makefile ensures that this happens. The following flow chart illustrates the sequence of operations.

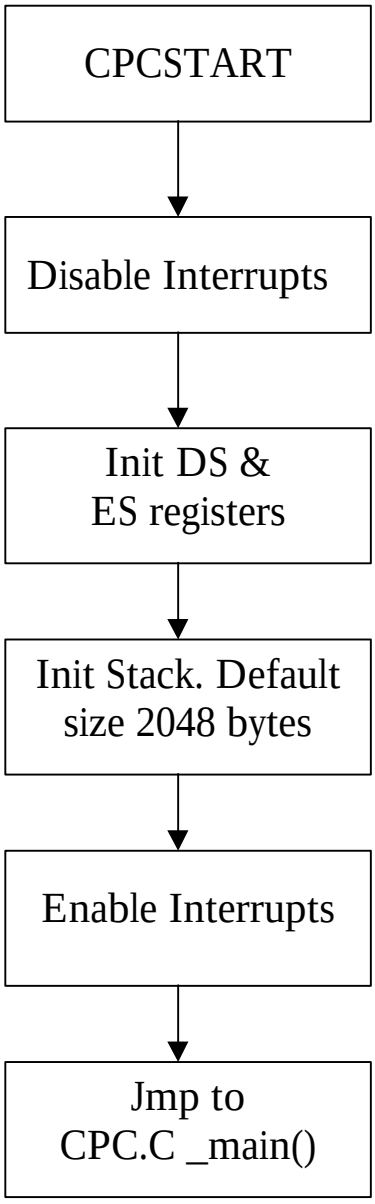


Figure 9 – Control Program Start Up.

6.2.2 Control Program Main Module CPC.C Description

This module comprises initialisation functions, FIFO (queue) handling functions, interrupt handlers, message handlers and a continuously running background processing loop.

6.2.2.1 CPC.C Initialisation

The functions ram_init() and timer_init() handle initialisation. These are called before the background processing loop is entered.

6.2.2.2 CPC.C Dual Port Ram Initialisation

This function initialises the data structures in the Dual Port Ram. It also installs the interrupt service routines for the serial controllers, the Z85230 ESCC's.
It's final task is to set the interact flag in the FUA to indicate to host software that the Control Program is running.

Each port has three structures, these are :

- The line table.
- Transmit FIFO or queue.
- Receive FIFO or queue.

In addition there are two 'global' structures :

- SYSQ – system to controller message queue.
- COMQ –controller to system message queue.

The line table can be likened to a *device control block* in the sense that it is the port control structure.

The transmit FIFO is used by the host computer to pass transmit data down to the Control Program and the receive FIFO is used by the Control Program to pass receive data back up to the host computer software.

The SYSQ and COMQ are used, respectively to pass messages from host computer to InterGrator i/PCi controller and vice versa.

These data structures and their use are further discussed in Section 6.2.2.5

The ram_init() function also installs the interrupt service routines for the ESCC's.
The ESCC's configured for vectored interrupts. Following an interrupt, the processor starts an interrupt acknowledge cycle. The ESCC with the highest priority pending interrupt returns its vector. Each ESCC is programmed with a base vector which will be modified according to the highest priority pending interrupt. This is referred to by the ESCC manufacturer as *vector includes status.*
The ESCC interrupts are described in detail in Section 6.3.

6.2.2.3 CPC.C Timer Initialisation

The InterGrator i/PCi has three timer channels that are embedded in the V53 processor. The boot firmware pre-initialises these to act as mode 2 rate generators. In other words they operate as counters. The count mode is set to binary.
The CPC Control Program initialises timer 1 to generate a periodic interrupt. The interrupt service routine (ISR) entered following an interrupt is in effect a stub that just increments a variable in the FUA and writes an end of interrupt (EOI) command to the programmable interrupt controller.

The period of the interrupt is 1/ count.

The reason for including this code is purely for demonstration purposes. The developer may of course expand the ISR handler to meet their requirements.

For register level programming details refer to the V53 processor user guide. The timer I/O addresses are shown in Table 28.

The prototype for timer_init() function is :

void timer_init(int tnum, unsigned count, void interrupt far (*isr)());

tnum = timer number 0,1 or 2
count = programmed counter (1/count = interrupt period)
*isr = pointer to interrupt service routine.

The operation is described by the following pseudo code.

```
FUNCTION timer_init
BEGIN
  IF tnum < 0 OR tnum > 2 THEN
    RETURN
  ENDIF

  DISABLE interrupts
  CALL FUNCTION set_vector()to install timer ISR
  OUTPUT control_word to timer(tnum)control register
  OUTPUT count to timer(tnum) count register
  ENABLE interrupts
  RETURN
END
```

Use the following formula to calculate the count value :

$$\textbf{Count Value} = \frac{\textbf{TCU clock}}{\textbf{Desired frequency}}$$

Where TCU clock = 750,000 for a 12Mhz InterGrator i/PCi and 1,250,000 for a 20Mhz InterGrator i/PCi.

The Timer Counter Unit (TCU) clock is derived from the external crystal oscillator divided by 32. The crystal oscillator frequency is 24Mhz for a InterGrator i/PCi 12Mhz controller and 40Mhz for an InterGrator i/PCi 20Mhz controller.

Note that the external crystal oscillator is also the source of the CPU clock via a divide by 2 circuit.

In general

$$\textbf{TCU clock} = \frac{\textbf{Oscillator Frequency}}{\textbf{32}}$$

6.2.2.4 CPC.C Background Processing.

After initialisation, the background processing loop is entered. The code loops, testing each active port's transmit FIFO for a pending character. If a character is available it is read from the FIFO and written to that port's ESCC for transmission. The port status is set to busy. When the transmission is complete the ESCC generates a *transmission buffer empty interrupt (TBE).* The TBE handler resets the port to idle and clears the interrupt.

Characters received by an ESCC causes a *receive character available (RCA)* interrupt to be generated. The RCA handler reads the character and writes it to the port's receive FIFO.
The CPC Control Program provides stub handlers for *external status change (ESC*) and *special receive condition (SRC)* interrupts. These handlers simply clear the interrupt and developers should add their own code to process these events.
The ESC interrupt occurs when the ESC detects a change in the *clear to send (CTS*) and/or *data carrier detect (DCD*) modem input signals. An ESC interrupt also occurs when the ESCC detects a *break* condition exists.
The SRC interrupt is generated in response to parity framing or overrun error.
When considering bit or character synchronous protocols there are other conditions that can cause ESC or SRC interrupts to occur as explained in the ESCC user's manual and/or data sheet.

6.2.2.5 CPC.C Main Loop

The following pseudo code describes the main processing loop.

```
DO
BEGIN
     FOR this_port = first port TO last port DO
     BEGIN
          IF this_port.status == line_active THEN
          BEGIN
               CALL FUNCTION deq_tx_data(this_port)
               IF  character  returned THEN
               BEGIN
                    WRITE byte to this_port.ESCC
                    this_port.status = tx_active
               END
               ENDIF
          END
          ENDIF
     END
     ENDFOR
END
LOOP forever
```

The CPC.C implementation is shown below :

```
for(linenum = 0;;linenum = (linenum + 1) % NUMLINES)
{
  lt_p = line[linenum];

  if(!(lt_p->line_status & LINE_ACTIVE))
       continue;

  if(lt_p->line_status & TX_ACTIVE)
       continue;

  deq_tx_data(lt_p);
}
```

There are a couple of points to note. Firstly the variable ‘linenum’ is equivalent to the physical port number and it’s value will be in the range 0 –15.
The pointer lt_p is a points to the line table structure for the current port or line being processed.

Each port has a line table and it is the main control structure. Member variables of the line table structure include the port ESCC base I/O address, line status word, ESCC shadow registers, and FIFO pointers.

The declaration from the DPRAM.H is shown below :

```
struct line_entry
{
  int  io_base;                  /* SCC base I/O address */
  unsigned line_status;          /* line status (defined below) */

  /* SCC register values */
  unsigned char WR2_;            /* write register 2 */
  unsigned char WR3_;            /* write register 3 */
  unsigned char WR4_;            /* write register 4 */
  unsigned char WR5_;            /* write register 5 */
  unsigned char WR12_;           /* write register 12 */
  unsigned char WR13_;           /* write register 13 */
  unsigned char WR15_;           /* write register 15 */
  unsigned char fill1;           /* filler for word alignment */

  /* Transmit queue info */
  int Txq_head;                  /* add data here */
  int Txq_tail;                  /* remove data here */
  char far *Txq_com;             /* ptr to Tx buffer, COM uP address */
  char far *Txq_sys;             /* ptr to Tx buffer, SYS uP address */

  /* Receive queue info */
  int Rxq_head;                  /* add data here */
  int Rxq_tail;                  /* remove data here */
  char far *Rxq_com;             /* ptr to Tx buffer, COM uP address */
  char far *Rxq_sys;             /* ptr to Tx buffer, SYS uP address */

  unsigned char fill2[12];       /* filler */
};
typedef struct line_entry LINE_ENTRY_T;
```

The bit definitions for the line_status member variable are:-

```
#define LINE_ACTIVE  0x0001    /* line is active */
#define TX_ACTIVE    0x0002    /* transmit is active (char is going out) */
```

The line tables, transmit FIFO's, receive FIFO's and the COM and SYS structures are defined by ram_init() at fixed addresses.

This start address of the data structures is also the address of the controller to system message queue structure COMQ. The address is defined by the manifest constant

```
/* COM uP start addr of comq message q */

#define COMQ_ADDR  0x00005000
```

This address will be translated to segment:offset 0000:5000h. The Dual Port Ram data structures exist within the first 64K of Dual Port Ram along with the code, stack and global variables. This can be changed to accommodate a growing Control Program.

For example to place the data structures at a start address of 1000:5000h, change the definition to :

```
#define COMQ_ADDR 0x10005000
```

The COMQ structure is the first of the Dual Port Ram data structures. It is followed by the controller to system message queue, SYSQ, and then, respectively, line tables, transmit FIFO's and receive FIFO's the following table lists these structures and their offsets.

Offset (HEX)	Use	Length in bytes
COMQ		
5000	COMQ head pointer	2
5002	COMQ tail pointer	2
5004	Message area	512 (32 * 16 byte messages)
SYSQ		
5204	SYSQ head pointer	2
5208	SYSQ tail pointer	2
520A	Message area	512 (32 * 16 byte messages)
5408	UNUSED FILLER BYTES	8
LINE TABLES		
5410	Line 0 table	48
5440	Line 1 table	48
5470	Line 2 table	48
54A0	Line 3 table	48
54D0	Line 4 table	48
5500	Line 5 table	48
5530	Line 6 table	48
5560	Line 7 table	48
5590	Line 8 table	48
55C0	Line 9 table	48
55F0	Line 10 table	48
5620	Line 11 table	48
5650	Line 12 table	48
5680	Line 13 table	48
56B0	Line 14 table	48
56E0	Line 15 table	48

Table continued

Offset	Use	Length (bytes)
Transmit FIFO's		
5710	Line 0 TX	512
5910	Line 1 TX	512
5B10	Line 2 TX	512
5D10	Line 3 TX	512
5F10	Line 4 TX	512
6110	Line 5 TX	512
6310	Line 6 TX	512
6510	Line 7 TX	512
6710	Line 8 TX	512
6910	Line 9 TX	512
6B10	Line 10 TX	512
6D10	Line 11 TX	512
6F10	Line 12 TX	512
7110	Line 13 TX	512
7310	Line 14 TX	512
7510	Line 15 TX	512

Offset	Use	Length (bytes)
Receive FIFO's		
7710	Line 0 RX	2048
7F10	Line 1 RX	2048
8710	Line 2 RX	2048
8F10	Line 3 RX	2048
9710	Line 4 RX	2048
9F10	Line 5 RX	2048
A710	Line 6 RX	2048
AF10	Line 7 RX	2048
B710	Line 8 RX	2048
BF10	Line 9 RX	2048
C710	Line 10 RX	2048
CF10	Line 11 RX	2048
D710	Line 12 RX	2048
DF10	Line 13 RX	2048
E710	Line 14 RX	2048
EF10	Line 15 RX	2048

Table 1 - Dual Port Ram Data Structures and Offsets

6.3 Control Program Handled Interrupts

As outlined in Section 6.2.2.4, the background loop can be interrupted by the ESCC's. There are also other hardware interrupts handled by the V53 processor.

6.3.1 Background

The V53 processor has a built in eight channel interrupt controller. The Interrupts are shown in the table below in priority order, INT0 is highest.

Note that INT7 is unconnected and a default invalid interrupt handler is supported by the firmware.

The CPC.C Control Program installs handlers for Host Computer System command, Timer1 and the ESCC's. The unused interrupt vector table entries all point to the invalid interrupt handler.

The CPC.C Control Program makes the installation of interrupt handlers easy by providing the function **setvector()** This function takes the interrupt type number and a pointer to the interrupt handler function, converts the pointer to a SEGMENT:OFFSET address format and installs this in the correct vector table offset. The vector table offset is always 4 x Vector Type number.
The following table lists the interrupts.

Interrupt	V53 PIC Interrupt	Vector Type Number	Vector Table Offset
DMA terminal count	INT0	30h	00C0h
ESCC ports 0 – 7	INT1	Cascade mode	See table 3
ESCC ports 8 – 15	INT2	Cascade mode	See table 3
Host system command	INT3	33h	00CCh
Timer 0	INT4	34h	00D0h
Timer 1	INT5	35h	00D4h
Timer 2	INT6	36h	00D8h
Invalid interrupt	INT7	37h	00DCh

Table 2 - Interrupts

Note that all the interrupt capable devices have an interrupt type number with the exception, apparently of the ESCC's! However, ESCC interrupts are handled in a slightly different manner because of the hardware implementation.

6.3.2 The ESCC Interrupt Hardware Implementation

This is a somewhat simplified description. You will note from the above table that only two interrupt channels are reserved for the ESCC's. These two channels are configured by the boot EPROM to work in cascade mode.

The ESCC interrupts are divided into two groups one for the lower eight channels and the other is the upper eight channels. These two groups are concentrated by on board logic into the two interrupt signals routed to the V53 Processor. Figure 10 below clarifies this.

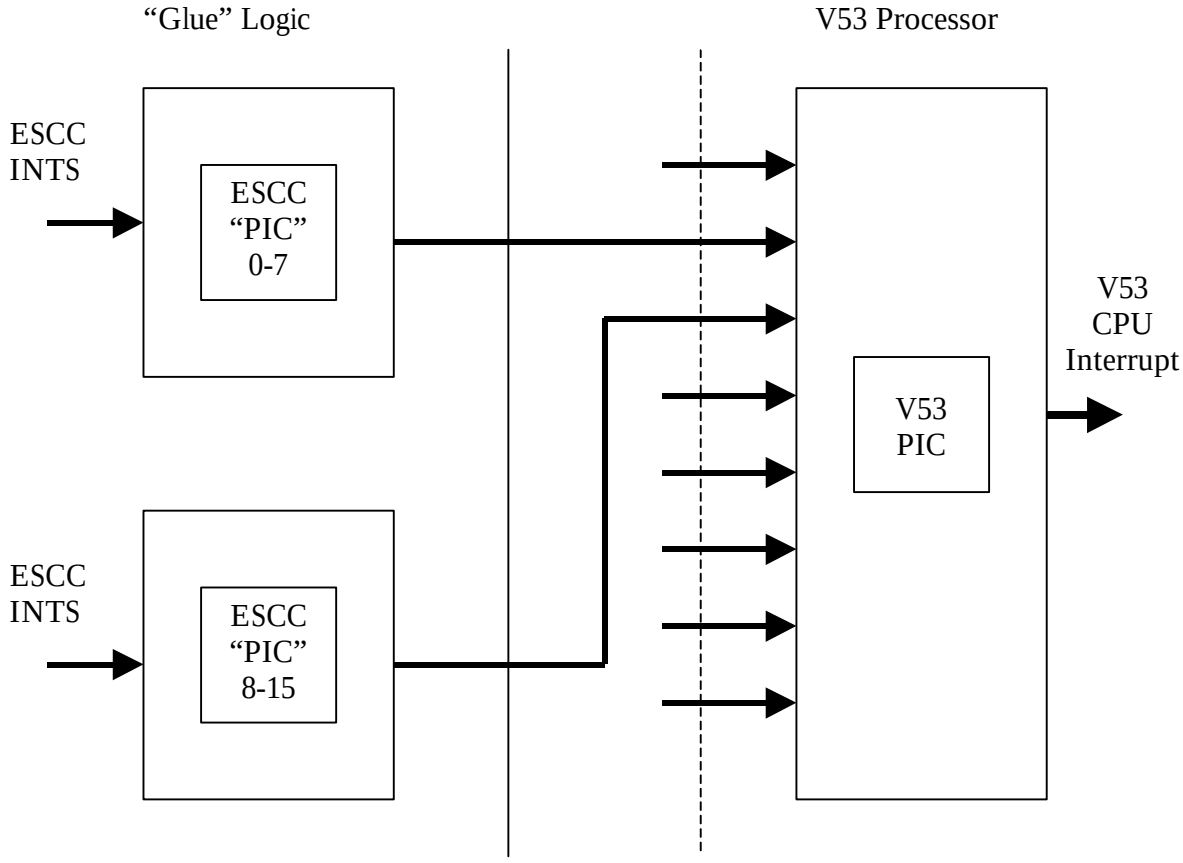


Figure 10 - Interrupts

Each ESCC interrupt signal is routed to logic circuitry which concentrates these into the two interrupt lines routed to the V53 processor INT1 and INT2 inputs.
Both the INT1 and INT2 inputs are configured by the boot EPROM to work in cascade mode.

What this means is that following an interrupt on any of these lines, the V53 processor interrupt acknowledge cycle (IACK) will expect the vector type number to be returned by the interrupting ESCC. The 'glue' logic ensures that the vector is returned by the highest priority ESCC channel. The priorities are ESCC port 0 – highest, port 15 – lowest.

The ESCC port's are programmed by CPC to interrupt when the following conditions occur and to return a pre-programmed vector with a value modified to indicate the interrupting condition (Vector Includes Status mode (VIS) :

- Special Receive Condition (SRC) – In async mode this is used to report errors
- Receive Character available (RCA)
- Transmitter Buffer Empty (TBE)
- External Status Change (ESC) – e.g. CTS change , DCD change , break

In response to these interrupts the V53 processor IACK cycle will request a vector type number. The highest priority ESCC will return the vector which will be the base vector number modified by the highest priority interrupting condition.

As there are 16 ports and each port can generate a vector for each of the above interrupting conditions, there is a total of 64 individual interrupt handlers for the ESCC's . However, these are merely stubs which call a common interrupt service routine passing a pointer to the ports line table.

The interrupt vector table has to be set up with the addresses of these 64 handlers. However, this task is carried out by the Control Program. The developer need only modify the handlers to suit their particular purposes.
Table 3 shows the base vector type programmed into the ESCC. The ESCC is a two channel device (Channel A and Channel B) and each ESCC has one base vector register (WR2) which serves both channels.
The base vector number is modified according to the interrupt condition and channel. Further information can be found in the Z85230 ESCC user guide.

The other columns show the offsets within the V53 processor interrupt vector table for each of the interrupt handlers. The interrupt vector table is fixed in the V53 processor memory map at 0000:0000h to 0000:03ffh. Each four byte entry is initialised with the Segment:offset of the respective interrupt handler.

Port	Base Vector Type	Transmit Buffer Empty (TBE)	External Status Change (ESC)	Receive Character Available (RCA)	Special Receive Condition (SRC)
0	80h	220h	228h	230h	238h
1	80h	200h	208h	210h	218h
2	90h	260h	268h	270h	278h
3	90h	240h	248h	250h	258h
4	A0h	2A0h	2A8h	2B0h	2B8h
5	A0h	280h	288h	290h	298h
6	B0h	2E0h	2E8h	2F0h	2F8h
7	B0h	2C0h	2C8h	2D0h	2D8h
8	C0h	320h	328h	330h	338h
9	C0h	300h	308h	310h	318h
10	D0h	360h	368h	370h	378h
11	D0h	340h	348h	350h	358h
12	E0h	3A0h	3A8h	3B0h	3B8h
13	E0h	380h	388h	390h	398h
14	F0h	3E0h	3E8h	3F0h	3F8h
15	F0h	3C0h	3C8h	3D0h	3D8h

Table 3 - Base Vector Type Programmed into the ESCC

6.3.3 Example Handler

The following example is for the Transmit Buffer Empty (TBE) interrupt service routine. The other interrupts follow a similar process in that an initial stub handler is called which then transfer control to a common isr, passing a pointer to the port's line table. This can be easily understood by referring to the Control Program source code file 'CPC.C'.
This example is for Port 0 (Line 00)

The stub transmit buffer empty handler for Port 0.

```
Void interrupt far line00_TBE()
{
        TBE_isr( line[0] );
        EOI( INTCTL,EOIVAL );
}
```

Below is the common transmit buffer empty ISR

```
void  TBE_isr( LINE_ENTRY_T far *lt_p )
{
        int scc;

        scc = lt_p->io_base;

        lt_p->line_status &= ~TX_ACTIVE;
        OUTB( scc,WR0 );
        OUTB( scc,RESET_TX_INT );
        OUTB( scc,WR0 );
        OUTB( scc,RESET_IUS );
}
```

6.3.4 System To Controller Interrupts (Host System Command)

The host computer software can interrupt the InterGrator i/PCi controller. The method by which this happens is described in Chapter 5 the Host Computer Software section. The V53 processor interrupt is INT3 which has a type number of 33h. The V53 processor interrupt vector table location (containing the handler's segment:offset address) is 00CCh.

System to controller interrupts are used to issue commands to the controller.

The Control Program actions depend on the controller state :

- Awaiting Control Program download and start-up.
- Control Program running.

6.3.4.1 Awaiting Control Program Download

In the first instance, the InterGrator i/PCi controller, once it has completed it's power on self test and basic initialisation will install a host computer system command handler and then enter a perpetual loop. At this stage the host passes commands and arguments to the InterGrator i/PCi controller via reserved memory locations within the Firmware User Area.
The host will invoke the interrupt by setting then clearing a USER I/O bit in the PLX9050 'CTRL' register (See Chapter 5, Host Computer Software).

The FUA command addresses locations are :

0000:0BAEh - One byte command
0000:0BAFh - One byte status.
0000:0BB0h - Sixteen byte 'message' buffer.

Two commands are recognised

01h	Start control program at segement:offset in 'message' buffer.
02h	Memory copy *count* bytes from *source segment: offset* to *destination segment:offset.*

The message format is :
For command 01h - Start Control Program.

0000:0BB0h - Segment address of loaded control program to start.
0000:0BB2h - Offset address of loaded control program to start.

For command 02h – Memory copy.

0000:0BB0h - Segment address of source.
0000:0BB2h - Offset address of source.
0000:0BB4h - Segment address of destination.
0000:0BB6h - Offset address of destination.
0000:0BB8h - Transfer count (two bytes).
When the interrupt occurs the handler checks the value of the byte at offset.

6.3.4.2 Control Program Running

The example Control Program, CPC.C, uses the host command interrupt to implement a mechanism that allows the host to pass run time commands to the InterGrator i/PCi controller. The two supported commands are OPEN and CLOSE although these can be extended using the same approach.

The Control Program installs a new handler **(SYSTEM_ISR)** and sets up a message queue (COMQ) which can support 32 pending messages of up to 16 bytes each. The message format is described in Chapter 5. It can also be determined by studying the Control Program source code.

The Host Computer will queue the message and then interrupt the InterGrator i/PCi controller. The system_isr handler is entered and calls the function deq_com_msg(). The command byte within the message is used to call the command handler via a jump table. The command is processed and on return to the isr, the interrupt cleared down. The system_isr handler code fragment is shown below.

```
void interrupt far system_isr()
{
        /* Set up dispatch table, one function for each Sys uP command */
        static void (*dispatch[NUM_SYSCMD])(void) =
        {
                null_cmd,               /* 0 */
                open,                   /* 1 */
                close,                  /* 2 */
                int_sys                 /* 3 */
        };

        if(!deq_com_msg())                      /* get message from Com uP */
        {
                EOI(INTCTL,EOIVAL);     /* end of interrupt to PIC */
                return;                 /* no message, return */
        }

         if(msg_buf[0] < 0 || msg_buf[0] >= NUM_SYSCMD)
        {
                EOI(INTCTL,EOIVAL);             /* end of interrupt to PIC */
                return;                         /* no message, return */
        }

        (*dispatch[msg_buf[0]])();          /* execute command function */
        EOI(INTCTL,EOIVAL);                 /* end of interrupt to PIC */
}
```

6.4 Interrupting the Host Computer

It is possible for the control program to interrupt the host computer. The interface between the sample Control Program and host computer software is polled. Therefore no use is made of this feature.

However, a function 'int_sys()' is included in the Control Program source for demonstration purposes. The function is set-up to be called from the system_isr(). In other words it can be invoked from the host computer software as a command. This is obviously not used in a practical control program. However, this 'sample' implementation is designed to allow the developer, the opportunity to experiment in a controlled manner.

The sample host software provides an example of setting up an interrupt handler to integrate with the WinDriver drivers. Consult the Jungo manual or on line documentation regarding writing WinDriver hosted interrupt handlers. Do not invoke the 'int_sys()' function unless host software handlers are installed.

It is a three step process for the control program to interrupt the host :

- Write 80h to I/O address EF60h
- Delay
- Write 00h to I/O address EF60h

This actually sets an onboard latch which causes the PLX9052 PCI bus chip, to assert the PCI bus interrupt.

The latch must be cleared by the host computer. It does this by writing to the USER 0 I/O bit of the PLX 9052 CONTROL REGISTER (CTRL).

The PLX9052 control register is a 32 bit register which appears at offset 50h within the chips Run Time Registers (RTR registers).

The RTR registers can be accessed from either memory or address space. The accompanying host computer sample software which uses Jungo's WinDriver, accesses the RTR registers from memory address space.

When writing to the PLX9052 control register it is important to preserve the other bits otherwise a malfunction is likely to occur.

6.5 Direct Memory Access (DMA)

The InterGrator i/PCi supports optional full duplex DMA on ports 0 & 1. The DMA controller is embedded in the V53 processor. The table below lists the registers and I/O addresses.

DMA Register	NEC V53 Abbreviation	InterGrator i/PCi allocated I/O address	Direction (Read/Write)
Initialisation register	DICM	9060h	W
Channel register	DCH	9061h	R/W
Count register (low Byte)	DBC/DCC	9062h	R/W
Count register (high byte)	DBC/DCC	9063h	R/W
Address register (low byte)	DBA/DCA	9064h	R/W
Address register (mid byte)	DBA/DCA	9065h	R/W
Address register (high byte)	DBA/DCA	9066h	R/W
Reserved		9067h	
Device control register (low byte)	DDC	9068h	R/W
Device control register (high byte)	DDC	9069h	R/W
Device mode control	DMD	906Ah	R/W
Device Status	DST	906Bh	R
Reserved		906Ch	
Reserved		906Dh	
Reserved		906Eh	
Device Mask Register	DMK	906Fh	R/W

Table 4 - V53 Processor Direct Memory Access.

DMA is typically used when operating in synchronous communications mode at higher data rates typically 64Kbps. It can equally be useful in asynchronous communications mode if the application employs burst data transfer.

The DMA operates between the Z85230 ESCC (ports 0 &1) and on board memory. The DMA channels are therefore operating in either memory read I/O write OR I/O read memory write.

DMA transfers from controller to host system via the PCI bus are not supported.

Note : The sample control program does not use DMA and you should refer to the V53 User Manual for programming information and register bit definitions. However, the following paragraphs discuss the specifics of the InterGrator i/PCi implementation.

6.5.1 DMA Operational Modes.

The following table lists the DMA channel allocation and shows that the DMA channels are allocated to specific functions. Also shown are the values that must be programmed into the DMA mode register (I/O address 906Ah) transfer direction (TDIR) bit field.

V53 DMA Channel	Port	DMD register TDIR bits 2 & 3	ESCC request signal	Function
0	0	1, 0	DTR/REQ	TX (memory read, I/O write)
1	0	0, 1	Wait/REQ	RX (I/O read, memory write)
2	1	1, 0	DTR/REQ	TX (memory read, I/O write)
3	1	0, 1	Wait/REQ	RX (I/O read, memory write)

Table 5 - DMA Channel Allocation.

6.5.2 Device Mode Register (DMD)

The DMD register contains other bit fields that must be set to specific values as shown below.

Bit	7 6	5	4	3 2	1	0
Name	TMODE	ADIR	AUTI	TDIR	not used	W/ B
Use	0 1 (single mode)	*	*	see Table 5		0 (byte transfer)

Table 6 - DMD Register (906Ah)

The bit fields marked by the asterisks (*) are programmer options. The other bit fields must be programmed as shown in the 'Use' row.

The abbreviations are :
TMODE : Transfer mode.
The InterGrator i/PCi only uses single mode.

ADIR : 0 = Address Increment, 1 = Address decrement
Programmer defined.

AUTI : Auto Initialise 0 = Disable 1 = Enable

TDIR : Transfer Direction
Channel specific, see Table 5

W/B : Word / Byte transfer
The InterGrator i/PCi only uses byte transfer (The Zilog 85230 ESCC is and 8 bit device).

6.5.3 Device Control Register (DDC)

Although this is a 16 bit register, I/O accesses are via two 8 bit registers referred to as DDC low (9068h) and DDC high (9069h).

DDC low is used to enable the DMA and specify the priority scheme either fixed or rotating priority.

Bit	7	6	5	4	3	2	1	0
Name			0	ROT		DDMA		

Table 7 - DDC Low Byte (9068h)

DDMA : 0 = Enable DMA operation, 1 = Disable DMA operation.

ROT : 0 = Fixed priority, 1 = Rotating priority
Fixed priority sets DMA channel 0 to the highest priority and channel 3 to the lowest.
With rotating priority, the last channel serviced becomes the lowest priority. This mode ensures equal servicing of DMA channels.

DDC high is used to specify the bus transfer mode and to control wait state insertion (by external /READY signal) or wait-at-verify mode.

Bit	7	6	5	4	3	2	1	0
Name							WEV	BHLD

Table 8 - DDC High Byte (9069h)

BHLD : DMA transfer mode
ALWAYS SET TO ‘0’ for bus release mode.

WEV : Wait insertion or wait-at-verify mode
ALWAYS SET TO ‘1’ for wait-at-verify.

6.5.4 Initialise Command Register (DICM)

This register is used to reset the DMA controller.

Bit	7	6	5	4	3	2	1	0
Name								RES

Table 9 - DICM (9060h)

RES : 0 = No operation, 1 = DMA controller reset.
Note that the reset bit is cleared automatically by the DMAC.

The table below lists the DMA controller state following reset.

Register	**Post Reset State**
Address (DBA)	No change
Count (DBC)	No change
Channel (DCH)	Channel 0 selected
Mode (DMD)	All bits cleared
Device Control (DDC)	All bits cleared
Status (DST)	All bits cleared
MASK (DMK)	All bits set (all channels masked ie DMA operationally quiescent)

Table 10 - DMA Controller State

6.5.5 Device Channel Register (DCH)

The bit definitions differ between I/O read and I/O write operations.

An I/O read returns a byte which indicates the currently selected channel and read/write access mode for both the count (DBC) and address (DBA) registers.

Bit	7	6	5	4	3	2	1	0
Name				BASE	SEL3	SEL2	SEL1	SEL0

Table 11 - DCH (9061h) I/O Read

SEL0 .. SEL3: Indicates selected channel as shown in the table below.

Channel	SEL3	SEL2	SEL1	SEL0
Zero	0	0	0	1
One	0	0	1	0
Two	0	1	0	0
Three	1	0	0	0

Table 12 - SEL0 .. SEL3 Selected Channel

BASE : Indicates read/write access to DBC, DBA, DCA & DCC as shown in the table below.

Bit Setting	Base Count (DBC)	Base Address (DBA)	Current Count (DCC)	Current Address (DCA)
0	Write only	Write only	Read/Write	Read/Write
1	Read/Write	Read/Write		

Table 13 - Read/Write Access

Bit	7	6	5	4	3	2	1 0
Name						BASE	SELECT CHANNEL (SELCH)

Table 14 - DCH (9061h) I/O Write

SELCH : Selects the channel for programming as shown in the table below.

Channel	SELCH (bit 1)	SELCH (bit 0)
Zero	0	0
One	0	1
Two	1	0
Three	1	1

Table 15 - SELCH

BASE : Selects base and count register access as shown in the table below.

Bit Setting	Base Count (DBC)	Base Address (DBA)	Current Count (DCC)	Current Address (DCA)
0	Write only	Write only	Read/Write	Read/Write
1	Read/Write	Read/Write		

Table 16 - Base and Count Register Access

6.5.6 DMA Base Count (DBC)/Current Count (DCC) Register

This is 16 bit register is accessed as two 8 bit I/O registers. It is a dual mode register read/write register. The DCH register BASE bit determines whether accesses are to the base or current count registers.

Bit	7	6	5	4	3	2	1	0
Name	C7	C6	C5	C4	C3	C2	C1	C0

Table 17 - DBC/DCC Low Byte (9062h)

Bit	7	6	5	4	3	2	1	0
Name	C15	C14	C13	C12	C11	C10	C9	C8

Table 18 - DBC/DCC High Byte (9063h)

NOTE : ***A DMA transfer will occur when the counter reaches 0 (zero). Therefore the DBC is programmed to the actual transfer count - 1.***

6.5.7 DMA Base Address (DBA), Current Address (DCA) Register

This 24 bit register is accessed via three 8 bit I/O locations. The DCH register BASE bit determines whether accesses are to the base or current address registers.

Bit	7	6	5	4	3	2	1	0
Name	A7	A6	A5	A4	A3	A2	A1	A0

Table 19 - DBA/DCA Low Byte (9064h)

Bit	7	6	5	4	3	2	1	0
Name	A15	A14	A13	A12	A11	A10	A9	A8

Table 20 - DBA/DCA Mid Byte (9065h)

Bit	7	6	5	4	3	2	1	0
Name	A23	A22	A21	A20	A19	A18	A17	A16

Table 21 - DBA/DCA High Byte (9066h)

The DBA/DCA address register defines the physical address. The InterGrator i/PCi has 1MB of SRAM. Bits A20 – A24 should be set to zero.

6.5.8 DMA Mask Register (DMK)

The DMK register enables/disables masking for the DMA channels. Following configuration of the registers described previously, unmasking the DMA channel is necessary for DMA cycles to occur.

Bit	7	6	5	4	3	2	1	0
Name					M3	M2	M1	M0

Table 22 - DMK (906Fh)

M0: 0 = Do Not mask DMA request for channel 0
1 = Mask DMA request for channel 0

M1: 0 = Do Not mask DMA request for channel 1
1 = Mask DMA request for channel 1

M2: 0 = Do Not mask DMA request for channel 2
1 = Mask DMA request for channel 2

M3: 0 = Do Not mask DMA request for channel 3
1 = Mask DMA request for channel 3

6.5.9 DMA Status Register (DST)

Bit	7	6	5	4	3	2	1	0
Name	RQ3	RQ2	RQ1	RQ0	TC3	TC2	TC1	TC0

Table 23 - DST (906bh)

TC0 : Terminal Count Channel 0
0=DMA in progress 1=DMA terminal count reached

TC1 : Terminal Count Channel 1
0=DMA in progress 1=DMA terminal count reached

TC2: Terminal Count Channel 2
0=DMA in progress 1=DMA terminal count reached

TC3: Terminal Count Channel 3
0=DMA in progress 1=DMA terminal count reached

RQ0: DMA Request Status Channel 0
0=No request pending, 1=Request pending

RQ1: DMA Request Status Channel 1
0=No request pending, 1=Request pending

RQ2: DMA Request Status Channel 2
0=No request pending, 1=Request pending

RQ3: DMA Request Status Channel 3
0=No request pending, 1=Request pending

6.5.10 Z85230 ESSC, DMA Programming

The following table shows the DMA channel allocation for ports 0 & 1.

V53 DMA Channel	Port	ESCC request signal	Function
0	0	DTR/REQ	TX (memory read, I/O write)
1	0	Wait/REQ	RX (I/O read, memory write)
2	1	DTR/REQ	TX (memory read, I/O write)
3	1	Wait/REQ	RX (I/O read, memory write)

Table 24 - Z85230 ESCC, DMA Programming

The ESCC's are programmed to generate DMA request via write registers WR1 and WR14.

WR1 bits D5, D6 & D7are used to enable the Wait/REQ and WR14 bit D2 is used to enable the DTR/REQ.

For full duplex DMA the bits should be programmed as shown below.

Bit	7	6	5	4	3	2	1	0
Name	1	1	1	*	*	*	*	*

* = See Note.

Table 25 - ESCC WR1 DMA Request (RECEIVE Direction)

Bit	7	6	5	4	3	2	1	0
Name	*	*	*	*	1	*	*	*

* = See Note.

Table 26 - ESCC WR14 DMA Request (TRANSMIT Direction)

* NOTE : *Remember to preserve the remaining bits in these registers*

As the ESCC DTR is not available when WR14 bit D2 = 1, an alternative DTR is provided by the InterGrator i/PCi configuration register at I/O address FE5Eh. The following table gives programming information.

InterGrator i/PCi Register	**I/O Address**	**Direction**
Configuration register 16 Bit I/O	FE5Eh	Write
Bit Position	**Port**	**Value**
1	0	0 = DTR driven by ESCC 1 = DTR driven by Bit 2 Alternative DTR, **Use if DTR/REQ is programmed as a DMA request.**
2	0	Alternative DTR 0 = DTR ON 1= DTR OFF
5	1	0 = DTR driven by ESCC 1 = DTR driven by Bit 6 Alternative DTR, **Use if DTR/REQ is programmed as a DMA request**
6	1	Alternative DTR 0 = DTR ON 1= DTR OFF

Table 27 - I/O Address FE5Eh

6.5.11 InterGrator i/PCi DMA Constraints

When programming the InterGrator i/PCi DMA controller there are several constraints that must be adhered to.

The V53 DMA controller can operate in two modes uPD71037 and uPD71071 mode. These relate to NEC DMA controller devices on which theV53 DMA controller is based.

The V53 DMA controller must be configured for uPD71071 mode by clearing the V53 SCTL register's DMAM bit to '0'.

The V53 OPSEL register DS bit must be set to a '1' to enable the DMA controller.

Furthermore, you must adhere to the DDC and DMD register constraints as outlined in earlier sections.

6.6 InterGrator i/PCi On-Board I/O Addresses

The following are all 8 bit I/O registers with the exception of the DSR poll register and Configuration register.

6.6.1 V53 Processor Internal I/O Register Addresses

For details of V53 Direct Memory Access register addresses refer to Table 4

Timer Register	**NEC V53 Abbreviation**	**InterGrator i/PCi allocated I/O address**	**Direction R/W**
Timer 0 count register	TCT0	9074h	R/W
Timer 1 count register	TCT1	9075h	R/W
Timer 2 count register	TCT2	9076h	R/W
Timer control (mode) word	TMD	9077h	W

Table 28 - V53 Processor Timers.

ICU register (see notes below)	**NEC V53 Abbreviation**	**InterGrator i/PCi allocated I/O address**	**Direction R/W**
Interrupt command register	IPFW	9071h	W

Table 29 - V53 Processor Interrupt Control Unit.

Notes :

- The ICU is initialised by code executed from the boot firmware following a reset and it is not normally necessary for the Control Program to access these registers. The Control Program need only write to the above register following completion of an interrupt service.

The only value that should be written to the Interrupt command register at 9071h is 20h End Of Interrupt (EOI).

Note that NEC documentation sometimes refers to this command as Finish Interrupt (FI).

6.6.2 V53 Processor External I/O Addresses

Z85230 ESCC Port	Command register I/O address	Data Register I/O address
Port 0	E1F4h	E1F6h
Port 1	E1F0h	E1F2h
Port 2	E3F4h	E3F6h
Port 3	E3F0h	E3F2h
Port 4	E5F4h	E5F6h
Port 5	E5F0h	E5F2h
Port 6	E7F4h	E7F6h
Port 7	E7F0h	E7F2h
Port 8	E9F4h	E9F6h
Port 9	E9F0h	E9F2h
Port 10	EBF4h	EBF6h
Port 11	EBF0h	EBF2h
Port 12	EDF4h	EDF6h
Port 13	EDF0h	EDF2h
Port 14	EFF4h	EFF6h
Port 15	EFF0h	EFF2h

Table 30 - Z85230 ESCC I/O Addresses

The DSR poll register is required because the 85230 ESCC does not support this signal. The DSR poll register is accessed by via an I/O word read to FE5Eh.
Each bit represents the state of the DSR modem input signal for one port.

InterGrator i/PCi register	**I/O address**	**Direction**
DSR poll register (16Bit I/O)	FE5Eh	Read
Bit Position	**Port**	**Value**
0	0	0 = DSR ON
1	1	0 = DSR ON
2	2	0 = DSR ON
3	3	0 = DSR ON
4	4	0 = DSR ON
5	5	0 = DSR ON
6	6	0 = DSR ON
7	7	0 = DSR ON
8	8	0 = DSR ON
9	9	0 = DSR ON
10	10	0 = DSR ON
11	11	0 = DSR ON
12	12	0 = DSR ON
13	13	0 = DSR ON
14	14	0 = DSR ON
15	15	0 = DSR ON

Table 31 - DSR Poll Register.

The control program can interrupt the host computer via bit 7 of the interrupt register at EF60h. The control program asserts the bit which will set a hardware latch and assert the PCI bus interrupt into the host.
The control program clears the bit. The interrupt will remain asserted until an IACK is generated by the host device driver.

Notes :-

Note 1. *For the interrupt to be recognised, Bit 6 of the PLX 9052* ***INTCSR*** *register must be set.*

Note 2. *The IACK is generated by the host computer's interrupt handler setting and then clearing Bit 2 of the PLX 9052* ***CNTRL*** *register.*
See Chapter 7.

InterGrator i/PCi Register	I/O address	Direction
Controller to System interrupt 8 Bit I/O	EF60h BIT 7 = 1 = Assert interrupt BIT 7 = 0 = Remove Interrupt	Write

Table 32 - Host Interrupt.

The configuration register is provided for two purposes. The first is to allow the SRAM to be mapped in place of the boot EPROM at address F000:0000h – F000:FFFFh in the V53 memory map. *(NOTE : Do not use this feature if working with the Turbo Debugger remote kernel or the terminal debugger.)*

The second purpose is to provide an 'alternative DTR' signal for those control programs using DMA on ports 0 and/or 1 when using the ESCC DMA REQ signal. When using this signal the DTR pin for the respective ESCC channel is not available. *(NOTE: DTR is not available when the InterGrator i/PCi and Universal Interface is configured for synchronous operation).*

InterGrator i/PCi Register	I/O address	Direction
Configuration register (16 Bit I/O)	FE5Eh	Write
Bit Position	**Port**	**Value**
1	0	0 = DTR driven by ESCC 1 = DTR driven by Bit 2 Alternative DTR
2	0	Alternative DTR 0 = DTR ON 1= DTR OFF
5	1	0 = DTR driven by ESCC 1 = DTR driven by Bit 6 Alternative DTR
6	1	Alternative DTR 0= DTR ON 1= DTR OFF
8	n/a	Allows SRAM to be mapped in place of boot EPROM. F000:0000h – F000:FFFFh 0 = EPROM mapped (default) 1 = SRAM mapped

Table 33 Configuration Register

7.0 Software Migration From Hostess i 8/16

This section is intended to assist existing Comtrol Hostess i 8/16 users to migrate their drivers and Control Programs to the InterGrator i/PCi. It should also be of interest for those porting Smart Hostess control programs to the InterGrator i/PCi.

Note that Smart Hostess developers must also update their control programs to account for the differences between the 80186 and the InterGrator i/PCi's V53. This is outside the scope of this manual and it is recommended the relevant data sheet and/or user guides are consulted.

There are two versions of the Hostess i8/16. The earlier version (version A) can be identified by a 3 position DIL I/O address selection switch. The later version (version B), which is the most numerous type, is identified by a 4 position DIL I/O selection switch. This manual may use the terms 'early' or version A and 'later' or version B to describe the controller.

The Hostess i/S16, is a development of the Hostess i8/16 version B.

The Hostess i8/16 version B featured revised firmware which allowed for a more flexible approach to control program download. The InterGrator i/PCi firmware is derived from the Hostess i8/16 version B firmware.

The Hostess i8/16 version A and the Smart Hostess only supported the control programs loaded to the fixed address of 1000:0080 (in on-board CPU address space).
The Hostess i8/16 and InterGrator i/PCi allow control programs to be downloaded to any starting address in dual port ram by providing a firmware utility command interface to facilitate control program startup.

The most obvious difference between the Hostess i8/16 controllers and the InterGrator i/PCi, is the PCi interface.

The Hostess i8/16 provided four direct access I/O registers to configure the controller's resources, issue controller resets and interrupts.

The InterGrator i/PCi dual port ram base address and IRQ are assigned by the host PC's bios at startup. The host driver gets the configuration via operating system dependent calls.
The RESET, SYSINT and the unique IACK features are handled by the PCi interface chip.

This section continues by describing the PLX9052 PCi interface chip registers that are pertinent to all developers. The PLX9052 registers are also referred to as Run Time Registers or RTR. The equivalent actions to the Hostess i 8/16 reset and system interrupt functions are implemented through these registers.

The PLX 9052 Run Time Registers (RTR) are accessible via both I/O and Memory address space. The base addresses being obtained via system calls. The sample software uses the Jungo WinDriver package as described in Chapter 5 and it accesses the RTR registers via memory address space.

It helps to have a copy of the PLX9052 data sheet to hand whilst reading this chapter. http://www.plxtech.com

The code fragments included in this section are device driver or kernel level and for the sake of illustration access the PLX9052 RTR registers via I/O address space with an assumed base address of 0xFC00.

NOTE: The InterGrator i/PCi specific code fragments shown in this section use _inpd() and _outpd calls which are applicable to Windows 95 and are for illustrative purposes.

7.1 PLX9052 Registers.

There are two registers that the developer must be aware of. These are the INTCSR register and the CNTRL register

7.1.1 INTCSR Register

The INTCSR register is a 32 bit register at offset 4Ch in the PLX9052 RTR register set.

There is no equivalent to this register on the Hostess i 8/16. Only one bit is of concern. However, when programming this bit the developer MUST preserve the state of the other bits

INTCSR bit 6 is a PCI bus interrupt enable control signal. It is set by the driver software to enable the InterGrator i/PCi interrupt to be gated onto the PCI bus and therefore to generate an IRQ.

Bit	Description	Read	Write	Value after reset
6	PCI Interrupt enable. A value of 1 enables PCI interrupt	Yes	Yes	0

Table 34 - INTCSR (4Ch)

NOTE : According to the PLX9052 data sheet only bits 0 – 7 are used. However, it is suggested that in addition to preserving the states of Bits 0-5 & bit 7, bits 8-31 are also preserved.

7.1.2 CNTRL Register

The CNTRL register located at offset 50h within the PLX9052 Run Time Register (RTR) set. It is a 32 bit register and includes *four* 3bit fields that can be used to support user defined I/O bits. The InterGrator I/PCi 16 uses *three* of these fields, programmed for USER I/O use to support the following functions :

- Host to controller interrupt acknowledge (IACK – USER I/O 0)
  (Note : The IACK must be issued by the host in response to a controller interrupt. This has no Hostess i/16 ISA equivalent)
- Host to controller interrupt (SYSINT – USER I/O 2)
- Host to controller reset (RESET – USER I/O 3)

The other USER I/O field (USER I/O 1) is not used by the InterGrator i/PCi 16. However we recommend that it is configured as a USER I/O Pin and the direction bit set up as an I/P.

The USER I/O pins on the PLX9052 are dual function. Following a hard reset, the PLX9052 defines these as USER I/O pins. Even so it is recommended that they are explicitly programmed for user I/O use.

Each 3bit field comprises :

- A definition bit (USER I/O or other PLX9052 defined function)
- A direction bit (relevant to USER I/O only) defining the data bit as an I/P or O/P
- A data bit.

As can be seen the CNTRL register USER I/O bit field is similar to a PIO device.

Bit	Description	InterGrator i/PCi Use	Set Bit to Logic State	Initial** Configuration
0	User I/O 0 or Wait0 pin	User I/O	0	0
1	User I/O 0 direction	Output	1	1
2	User I/O 0 data	IACK*	1=IACK on 0=IACK off	0
3	User I/O 1 or LLOCK pin	User I/O	0	0
4	User I/O 1 direction	Input	0	0
5	User I/O 1 data	Reserved	0	0
6	User I/O 2 or CS2 pin	User I/O	0	0
7	User I/O 2 direction	Output	1	1
8	User I/O 2 data	SYSINT	1= SYSINT on 0= SYSINT off	0
9	User I/O 3 or CS3 pin	User I/O	0	0
10	User I/O 3 direction	Output	1	1
11	User I/O 3 data	RESET	0= RESET on 1= RESET off	1
↓	*Bits 12 –31 must be preserved.*			

Table 35 - CNTRL Register (50h)

NOTE: When programming the CNTRL register, care must be taken to preserve all other bits.

**The IACK bit does not have a Hostess i equivalent.*
*** The initial configuration is with the USER I/O bit settings initialised by the driver.*
Remember that the CNTRL register has other bit fields that must be preserved otherwise operation of the InterGrator i/PCi and/or the host PC will be adversely affected.

7.2 Use of the PLX9052 CNTRL Register USER I/O Bits

The USER I/O Bits 0, 2 and 3 are used to implement the host system to controller interrupt acknowledge (IACK), host system to controller interrupt (SYSINT) and controller reset (RESET) respectively. User I/O bit 1 is not used.

7.2.1 Hostess i 8/16 versus InterGrator i/PCi Reset

The host computer software can perform a software reset on both controllers.

With the Hostess i 8/16 the host computer software will write the value 0 to base I/O address + 3, delay for 100ms and then write 0xff to base I/O address + 3 to remove the reset. So assuming the controller's base I/O address is 0x218 then :

```
outp( 0x21b,0x00 );      /*Set the reset*/
delay( HZ/10 );          /*Delay 1/10 second*/
outp( 0x21b,0xff );      /*Remove the reset*/
```

With the InterGrator i/PCi, prior to accessing any of the PLX9052 CNTRL register functions, the USER I/O bit fields must be configured and the USER I/O pins themselves set to the quiescent state.

```
/*initialise USER I/O bits in PLX 9052 CNTRL register*/
cntrl =  _inpd( RTRbase + 0x50 );     /*Get CNTRL value*/
cntrl  &= 0xFFFFF000;                 /*Clear the four USER I/O bit fields*/
cntrl  |=   0x00000C82;               /* Set up User I/O bit fields (see table 35) */
_outpd( RTRbase+0x50 );
```

The procedure for resetting the InterGrator i/PCi, following CNTRL register initialisation is to set the USER I/O 3 data bit low, delay 100mS and then set the bit high.

e.g.

```
/*Reset the controller*/
cntrl = _inpd( RTRbase+0x50 );      /*Get CNTRL value*/
cntrl &= 0xFFFFF7FF;                /*Clear the USER I/O 3 data bit  RESET=ON*/
_outpd( RTRbase + 0x50,cntrl );     /*Assert reset signal*/
user_delay(100);                    /*User function to generate 100mS delay*/
cntrl |= 0x00000800;                 /*Set the USER I/O 3 data bit RESET=OFF*/
_outpd( RTRbase + 0x50,cntrl );     /*De-Assert reset signal*/
```

7.2.2 Interrupting the Controller

The host computer system can interrupt the V53 processor on the InterGrator i/PCi. The same was true for the Hostess i 8/16.

The mechanism used with the Hostess i 8/16 was simply to perform an I/O write to base I/O address + 2. This was coded simply as (assume base I/O address is 0x218) :

```
outp(  0x21a,0 );
```

The Hostess i 8/16 simply decodes the I/O write and generates an interrupt on the V53 processor PIC's INT 3 line (Vector type number 33h).

The InterGrator i/PCi uses the USER I/O 2 data bit in the PLX9052's CNTRL register. In the quiescent state the bit is low. To generate the interrupt the bit must be taken high and then low. On faster systems (e.g. Pentium 400Mhz systems) it has been found necessary to ensure that a small delay is inserted between setting and clearing the USER I/O 2 data bit (SYSINT bit). This can be achieved by reading the CNTRL contents a second time before clearing the bit (instead of saving a local copy of the CNTRL register contents and using that).

On Pentium 600Mhz systems and above a longer delay will be needed. For example a delay function (eg delay(1) - delay 1mS) or equivalent, can be called. The delay, however implemented, should occur between setting and clearing of SYSINT.

For example, the following driver code works satisfactorily on PC's up tp 300Mhz.

```
cntrl  = _inpd( RTRbase + 0x50 );  /*get a copy of the cntrl register*/
cntrl |= 0x00000100;                /*set USER I/O 2 data bit SYSINT = ON*/
_outpd( RTRbase + 0x50,cntrl );    /*assert SYSINT*/
cntrl &= 0XFFFFEFF;                /*set USER I/O 2 data bit  SYSINT = OFF*/
_outpd( RTRbase + 0x50,cntrl );   /*de-assert SYSINT*/
```

However, on faster machines (eg Pentium 400Mhz) then following method should be used. Of course it can also be used on slower PC's if it is wished.

On Pentium 600Mhz and above an explicit delay will be needed. This will be placed after the /*assert SYSINT*/ line.

```
cntrl  = _inpd( RTRbase + 0x50 );  /*read the cntrl register*/
cntrl |= 0x00000100;                /*set USER I/O 2 data bit SYSINT = ON*/
_outpd( RTRbase + 0x50,cntrl );    /*assert SYSINT*/
cntrl  = _inpd( RTRbase + 0x50 );  /*second read the cntrl register */
cntrl &= 0xFFFFFEFF;                /*set USER I/O 2 data bit SYSINT = OFF*/
_outpd( RTRbase + 0x50,cntrl );    /*de-assert SYSINT*/
```

The Control Program has to clear the interrupt. The method is the same for the InterGrator i/PCi as it was for the Hostess i 8/16.

```
outp( 0x9070, 0x20 );
```

7.2.3 Controller Interrupts the System

The InterGrator i/PCi can interrupt the host computer system (i.e. generate an IRQ). The Hostess i 8/16 also had this capability.

The generation of the IRQ is a Control Program task. The Hostess i 8/16, which is an ISA controller, asserted the IRQ and then removed it. The rising edge triggered the interrupt.

e.g.

```
/*Hostess i 8/16 IRQ generation*/
outp( 0xef60,8 );              /*assert the IRQ*/
asm push ax;                   /*these two instructions simply */
asm pop ax;                    /*cause a small delay*/
outp( 0xef60,0);               /*de-assert the IRQ*/
```

The PCI bus uses level triggered interrupts and so the InterGrator i/PCi IRQ generating mechanism is different. The InterGrator i/PCi supports a Set-Reset (SR) latch. The Control Program generates the 'set' signal. Once the latch is set it will remain so until the host computer system generates an interrupt acknowledge (IACK) signal.

The IRQ must be enabled onto the PCI bus by setting bit 6 of the PLX9052 INTCSR register as shown in the following example :

```
intcsr = _inpd(  RTRbase + 0x4C );   /*get the INTCSR value*/
intcsr |= 0x00000040;                /*Bit 6=1 enables PCI IRQ generation*/
_outpd( RTRbase + 0x4C, intcsr );
```

The InterGrator i/PCi control program IRQ generation is similar to the Hostess i 8/16.

```
/*Intergrator i/PCi IRQ generation*/
outp( 0xef60,0x80 );           /*Set PCI interrupt latch*/
asm push ax;                   /*these two instructions simply */
asm pop ax;                    /*cause a small delay*/
outp( 0xef60,0);               /*Remove the set signal*/
```

Note : The set signal must be removed otherwise the host system IACK will not reset the latch.

As PCI bus interrupts are level triggered, the host computer must issue an Interrupt Acknowledge (IACK). It does this via the USER I/O 0 of the PLX9052 RTR CNTRL register.

On faster PC's (eg Pentium 600Mhz) it may be necessary to code a delay after the /*assert the IACK*/ line.

```
cntrl  = _inpd( RTRbase + 0x50 );   /*read cntrl register*/
cntrl |= 0x00000004;                /*set USER I/O 0 data bit  IACK = ON*/
_outpd( RTRbase + 0x50,cntrl );     /*assert the IACK*/
cntrl  = _inpd( RTRbase + 0x50 );   /*second read  of cntrl register*/
cntrl &=   0xFFFFFFB;               /*set USER I/O 0 data bit IACK = OFF*/
_outpd( RTRbase + 0x50,cntrl );     /*de-assert the IACK*/
```

7.3 Firmware User Area Differences

The InterGrator i/PCi creates a data structure known as the firmware user area. There are two copies, one at 0000:0B80 and one at 1000:0000. Both these addresses are with reference to the V53 memory address space.

The FUA is used provides diagnostic and other information. It is also accessed during the download procedure. The FUA is shown in Table 36 below and can also be found in Appendix A.

There are a few differences when compared to the FUA found on a Hostess i8/16.

- There is no extended memory on the InterGrator i/PCi and so the word at offset 1Ch is not used.
- The identification (ID) number at offset 22h is unique to the InterGrator i/PCi 16. The ID number is the PCi vendor and device ID stored in little endian format. The first word is the device ID and the second the vendor ID.

The FUA at 0000:0B80 is referred to as the primary FUA. The developers support software sample application (IDEM) uses this copy.

The FUA at 1000:0000 is maintained for the benefit of those developer's porting code from older controllers such as the early revision of the Hostess i8/16 and the Smart Hostess.

Both these controllers *(for reasons outside the scope of this document)* supported one FUA copy at 1000:0000. In addition control programs for these cards were downloaded to the fixed address of 1000:0080 (with respect to the V53 CPU). It was normal practice to include a relocator in the control program which was invoked on start up, causing it to relocate to another memory area. Usually a call was made to the firmware (INT 21H) to obtain the first free memory segment.

The later version of the Hostess i8/16 *(identifiable by a four position DIL configuration switch)* supported both FUA structures.
Both the InterGrator i/PCi and the later version Hostess i8/16 support downloading of the control program to different areas of dual port ram. This is supported by the firmware utility command interface which allows the host to specify the control program start address. See the developers support software sample application (IDEM) and also appendix A.
Both controllers still retain the original control program startup procedure as an alternative.

7.4 Control Program Start Up InterGrator i/PCi

This section assumes that the control program has already been downloaded.

The developers sample software application (IDEM) loads the control program to 00c0:0000 (in V53 memory space) and starts it using the utility command interface. This is described in Chapter 5 and can be understood by studying the IDEM source files.

The remainder of this section describes a startup mechanism that is familiar to Smart Hostess and to some Hostess i8/16 users, and is retained by the InterGrator i/PCi for compatibility.

7.4.1 Compatibility Control Program Start Up Mode

This section will be of interest to users with existing control programs developed for early revision Hostess i8/16 and Smart Hostess controllers. The following text assumes that the control program has been downloaded to the address 1000:0080 within V53 memory address space.
To use this method of startup the control program must be downloaded to start at this address. To start the control program we simply issue a SYSINT.

/*If the control program is at 1000:0080 we simply generate a SYSINT*/

```
cntrl  = _inpd( RTRbase + 0x50 );  /*read the cntrl register*/
cntrl |= 0x00000100;               /*set USER I/O 2 data bit SYSINT = ON*/
_outpd( RTRbase + 0x50,cntrl );    /*assert SYSINT*/

/*a delay may require to be added here on fast Pentium PC's  600Mhz and above*/

cntrl  = _inpd( RTRbase + 0x50 );  /*second read the cntrl register */
cntrl &= 0xFFFFFEFF;               /*set USER I/O 2 data bit SYSINT = OFF*/
_outpd( RTRbase + 0x50,cntrl );    /*de-assert SYSINT*/
```

The reason that this works is that the firmware utility command and firmware utility data (see FUA Table 36), are preset to the default command of EXECUTE (0x01h) the control program at 1000:0080.
If downloading to another area of dual port ram then the address in the firmware utility data area has to be changed by the host to the actual control program start address. See Chapter 5 and Appendix A.

With the early Hostess i8/16 controller and the Smart Hostess the control program was usually relocated to the first free memory segment. The relocator had to be built into the control program itself. To find the first free memory segment the software INT 21h handler was called. The handler code is located in the boot eprom and executed by the controller (Do *not confuse this handler with the DOS int 21h handler).*

7.4.2 RAM Query (int 21h)

This software interrupt returns the first segment that is open for control program use in the V53's AX register. This handler is maintained for compatibility with control programs written for the Smart Hostess and Hostess I 8/16 (version A).

7.4.3 Turbo Debugger

Turbo debugger (Borland Turbo Debugger V3.1) is supported by the InterGrator i/PCi.

Note that the chapter describing Turbo debugger assumes that the control program is being loaded directly to 00c0:0000 (within V53 memory address space).

Offset	Description	Use	Length (bytes)
0h	Interaction flag 55AAh = controller active	Download	2
2h	Reserved (area 1)	N/A	2
4h	Unused (area 1)	N/A	2
6h	Firmware Release Number	Information	8
0Eh	Available for control program release number	Information	8
16h	Reserved (area 2)	N/A	2
18h	Reserved (area 3)	N/A	2
1Ah	Controller Ram Map one bit set per 64K found available by POST	Information/ diagnostic	2
1Ch	Reserved (area 4)	N/A	2
1Eh	Controller SCC Port Map one bit set per port found by POST	Information/ diagnostic	2
20h	Reserved (area 5)	N/A	2
22h	Identification number	Information	4
26h	Invalid interrupt flag. *(Normally 00h Set to 01h if an invalid interrupt has occurred. Replaced by Invalid interrupt count)**	diagnostic	1
27h	Invalid interrupt type. *(Normally 00h. Type number of last invalid interrupt which occurred)**	diagnostic	1
28h	Invalid interrupt count *(Normally 0000h. Number of invalid interrupts that occurred. If ffffh reached them terminal debugger is automatically invoked)**	diagnostic	2
2Ah	Heartbeat counter incremented by CPC control program. *User can implement similar function.*	diagnostic	4
2Eh	Firmware utility command	Download	1
2Fh	Firmware utility status	information	1
30h	Firmware utility message buffer	Download	16
40h	Controller pending interrupt flag Set by CPC control program, tested & cleared by IDEM application *(via WinDriver).*	Status	2
42h	Reserved (area 6)	N/A	62

Table 36 - Firmware User Area

** Invalid interrupt handling. The boot EEPROM installs default handlers for all unused S/W and H/W interrupts. These handlers call a common ISR service routine which updates these fields.*

8.0 Interface Connector Pin-Outs

This section gives information on the DB25F pin-outs found on the InterGrator UI and the Rocket Port 16 interface unit.

8.1 Rocket Port 16 Interface Unit

The Rocket Port 16 port interface unit is available in two versions. There is an RS232 only version and an RS232/RS422 external switch selectable version. The physical connectors are DB25F.

8.1.1 Rocket Port 16 Interface DB25F Pin-Outs

RS232 Signal	Abbreviation	DB25F Pin	Direction (DTE) Referred to I/F Unit
Transmit Data	TXD	2	Output
Receive Data	RXD	3	Input
Request To Send	RTS	4	Output
Clear To Send	CTS	5	Input
Data Set Ready	DSR	6	Input
Signal Ground	SG	7	-
Data Carrier Detect	DCD	8	Input
Data Terminal Ready	DTR	20	Output

Table 37 - DB25F RS232 Pin-Outs Rocket Port 16 Interface

RS422 Signal	Abbreviation	DB25F Pin (Rocket Port)	Direction (DTE) Referred to I/F Unit
Transmit Data +	TX+	19	Output
Transmit Data -	TX-	25	Output
Receive Data +	RX+	15	Input
Receive Data -	RX-	17	Input

Table 38 - DB25F RS422 Pin-Outs Rocket Port 16 Interface

8.2 InterGrator UI

The InterGrator UI is an eight port 19" rack mounted unit. Two units are daisy chained to give sixteen ports. The physical connection is via DB25F connectors. There are three versions of the InterGrator UI. Version 3 is used with the InterGrator i/PCi and the DB25F pin-out tables refer to this version.

Only a brief outline of the interface and DB25 pin-outs is given here. Please refer to the relevant user manual for full details and guidance on configuration, to support the alternative physical protocols.

Both RS232 and RS422 are common to both interfaces, but beware; the Rocket Port RS422 pin-outs are different and are taken from the Rocket Port hardware reference guide.

The InterGrator UI RS422 interface uses ITU-T V.11 signal references. That is, the signals are referred to as (A) and (B). The electronic design is such that driver / receiver designation '+' = (A) and '-'= (B).

8.2.1 InterGrator UI (Version 3)

The Version 3 interface supports RS232, RS422, RS485 (two & four wire), RS423 and current loop (active/active, active/passive & passive/passive combinations).

In addition, in RS232 mode, synchronous communications is supported via a DCE sourced receive clock (RXCLK) and DTE sourced transmit clock (TXCLK).
The DB25F connector is shared between these physical protocols. Selection is made by configuring the InterGrator UI internal jumpers (See InterGrator UI user guide) and correct cable wiring.

Tables 40 through 46 show the specific pin-out for each protocol. When manufacturing a cable only connect to those pins required.

Do not use flat cables or cables with all 25 cores connected.

Note : *Pins 14,15 and 16 provide DC power for special applications. Do not connect to these pins unless they are required. Refer to the InterGrator UI user guide and specifications for maximum current available through these pins.*

DB25F Power pins. These pins are permanently powered.	**Voltage**
14	+5V
15	+12V
16	-5V

Table 39 - DB25F DC Power Output Pin-Outs for InterGrator UI (Version 3)

Note : *For all interfaces, Pin 1 is connected to shield.*

RS232 Signal	**Abbreviation**	**DB25F Pin**	**Direction (DTE) Referred to I/F Unit**
Transmit Data	TXD	2	Output
Receive Data	RXD	3	Input
Request To Send	RTS	4	Output
Clear To Send	CTS	5	Input
Data Set Ready	DSR	6	Input
Signal Ground	SG	7	-
Data Carrier Detect	DCD	8	Input
Data Terminal Ready	DTR	20	Output

Table 40 - DB25F RS232 Pin-Outs InterGrator UI (Version 3) ASYNC Mode.

Note : The following table shows the RS232 DB25F pin configuration for Synchronous operation with separate transmit & receive clocks. The DTR and DSR signals are no longer available as they are allocated for clocking. The DCD signal is not supported in this mode. Therefore a custom cable will be required.

RS232 Signal	Abbreviation	DB25F Pin	Direction (DTE) Referred to I/F Unit
Transmit Data	TXD	2	Output
Receive Data	RXD	3	Input
Request To Send	RTS	4	Output
Clear To Send	CTS	5	Input
Receive Clock	RXCLK	6	Input
Signal Ground	SG	7	-
Transmit Clock	TXCLK	20	Output

Table 41 - DB25F RS232 Pin-Outs InterGrator UI (Version 3) SYNC Mode.

RS422 Signal	Abbreviation	DB25F Pin	Direction (DTE) Referred to I/F box
Transmit Data +(B)	TXB	23	Output
Transmit Data -(A)	TXA	11	Output
Receive Data +(B)	RXB	22	Input
Receive Data -(A)	RXA	10	Input

Table 42 - DB25F RS422 Pin-Outs InterGrator UI (Version 3)

RS485 Signal	Abbreviation	DB25F Pin
485 TX/RX + (A)	485 CH A	10
485 TX/RX - (B)	485 CH B	22

Table 43 - DB25F RS485 (2 wire Half Duplex) Pin-Outs InterGrator UI (Version 3)

RS485 Signal	Abbreviation (ITU V.11)	DB25F Pin	Direction (DTE) Referred to I/F Unit
485 Transmit Data + (B)	485 TXB	23	Output
485 Transmit Data - (A)	485 TXA	11	Output
485 Receive Data + (B)	485 RXB	22	Input
485 Receive Data + (A)	485 RXA	10	Input

Table 44 - DB25F RS485 (4 wire Full Duplex) Pin-Outs InterGrator UI (Version 3)

RS423 Signal	Abbreviation	DB25F Pin	Direction (DTE) Referred to I/F Unit
423 Transmit Data	TX423	9	Output
423 Transmit Ground	GND	19	-
423 Receive Data	RX423	10	Input
423 Receive Ground	GND	21	-

Table 45 - DB25F RS423 Pin-Outs InterGrator UI (Version 3)

CL Signal	Abbreviation	DB25F Pin	Direction (DTE) Referred to I/F Unit
Current Loop Transmit +	CL T+	13	Output
Current Loop Transmit -	CL T-	25	Output
Current Loop Receive +	CL R+	12	Input
Current Loop Receive -	CL R-	24	Input

Table 46 - DB25F Current Loop (CL) Pin-Outs InterGrator UI (Version 3)

Refer to the InterGrator UI user guide for information on the use of an external power supply if using active mode current loop configuration.

9.0 Turbo Debugger

This section describes the basics for setting up a remote debug session with Borland's Turbo Debugger and the InterGrator i/PCi with its Turbo Debugger remote kernel. It is not a comprehensive 'user guide' for Turbo debugger and it assumes a level of familiarity with it. Consult the Borland user guides for user and other information.
Note that this section assumes that the control program is being directly loaded to 00C0:0000 within V53 memory address space.

The InterGrator i/PCi firmware contains a Turbo Debugger remote kernel. By attaching a serial cable from header P9 to the COM1 or COM2 port of a remote computer running Borland's Turbo debugger 3.1, it is possible to carry out source level debugging of the Control Program.

In order for this to be possible the Control Program should have been built using tools provided with the Borland C/C++ V3.1 professional edition. The 16 bit programs which are run under MS-DOS or within a Windows 95/98 DOS session are listed below :

- Borland C/C++ 3.1
- Borland TASM 3.1
- Borland TLINK 5.1
- Borland TDSTRIP 3.1
- Borland MAKE 3.6

In addition, the Comtrol supplied utility CLOCATE.EXE must be used to produce the downloadable binary image.

It is recommended that the MAKEFILE supplied with the control program source code is used to control the build. This file is designed for use with the above software.

Existing Hostess i 8/16 developers may already have access to the above Borland software tools.
At the time of writing, the above software is also available packaged with the 'complete compiler edition' of the book 'Teach yourself C in 21 days 4th edition'.

9.1 Setting Up Turbo Debugger

The host computer supporting the InterGrator i/PCi is referred to as the Target PC. The other PC which hosts the Turbo Debugger program is referred to as the Master PC. The Master PC must be running MS-DOS or an MS-DOS session under Windows 95/98. The Turbo Debugger will not work under a Windows NT MS-DOS console session.

Quite often the Borland compiler and associated software, will also be installed on the Master PC. Typically the Control Program to be built and the binary (e.g. CPC.BIN) will be transferred to the Target PC.

As mentioned above, the Control Program binary will need to be on the Target PC ready for download.

The files required on the Master PC are the Control Program source and the Turbo debugger symbol table file CPC.TDS (for the sample Control Program). The symbol table is produced during the build.
The serial connection is made between the InterGrator i/PCi header P9 and the Master PC's COM1 or COM2 port. The cable wiring diagram is shown below.

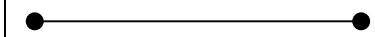
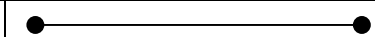
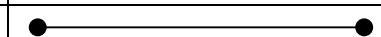
InterGrator i/PCi to Master PC COM1 or COM2		
Header P9		**DB9Female**
G		**5**
T		**2**
R		**3**

Table 47 - InterGrator i/PCi to Master PC COM1 or COM2

The Turbo Debugger is started on the Master PC using the following command :

td –rp<com port number> -rs3

Where <com port number> is 1 for COM1 and 2 for COM2.

e.g. if using COM2 then the command is :

td –rp2 –rs3

Terminate the command with enter or <CR>

The Master PC will respond with a sign on message followed by the string :

“Waiting for handshake from remote driver (Ctrl-Break to quit) and a flashing underline cursor.

At this point Turbo Debugger is waiting for communication with the remote kernel on the InterGrator i/PCi.

This is initiated from the Control Program by forcing the execution of a 80x86 software interrupt instruction. The instruction is :

INT 27h

The accompanying source code for sample software demonstrates how this is achieved. Essentially the Control Program is downloaded but not started. The loader reads and stores the first two bytes of the Control Program and inserts the machine code equivalent of the INT 27h instruction.

The loader will then start the Control Program which will trigger the remote kernel, communication with the master will be established then the Turbo Debugger program will start. The Turbo debugger screen will appear.

Clear the message box (showing the version number, which should be 3.1) by pressing the Esc key.
The CPU window will now appear.

The Control Program is at this point frozen and the Turbo Debugger must be configured for the debug session to commence. The loader will replace the INT 27h instruction with the original bytes saved previously.

9.2 Configuring for the Debug Session.

The following assumes that you are using the sample software. Study the supplied source code to determine the exact methods used to invoke the Turbo Debugger remote kernel. These can be applied to your own specific development.

At this point communication has been established between the Master's Turbo Debugger and the remote kernel on the InterGrator i/PCi. The Control Program is resident on the controller but is currently frozen.

The following steps are now required prior to the debug session :

- Set-up the stack
- Load the symbol table
- Table relocate
- Set entry point address
- Update CS:IP registers
- Prime Turbo debugger

1. Set-up stack
The stack segment is set to 0098h.

a. Press **<ALT> V** to view (i.e. press the ALT key and V simultaneously)

b. Press **R** or use the **<ARROW KEY>** to select **REGISTERS**

c. Use **<ARROW KEY>** to choose **STACK SEGMENT** by highlighting **SS**
and press **<ALT> F10**

d. Press **C** or use the **<ARROW KEY>** to select **CHANGE**

f Enter the value (in this case 98), Press
<ENTER>,<ALT> W for **WINDOW** and then **CLOSE.**

NOTE : *The stack segment (SS) must be set to 0098h, before executing the start-up program. Failure to do so is likely result in your code being corrupted.*

2. Load the symbol table

a. Press **<ALT> F** and select **SYMBOL LOAD.**

b. Choose CPC.TDS from the menu.

3. Table relocate

a. Press **<ALT> F** and select **TABLE RELOCATE**.

b. Enter the value 00C0 in the displayed dialog box.

4. Set entry point

a. Press **<CTRL> G** (i.e. Press the CTRL key and G simultaneously) and enter the address of the entry point in the dialog box. For the sample Control Program this should be CPMAIN.

5. Update CS:IP registers

a. Press **<CTRL> N**

*(This updates the registers for the CS:IP (current segment:instruction pointer***)**

6. Prime Turbo Debugger
NOTE*: This is required so that Turbo debugger recognises that a program for remote debug has been loaded. Failure to complete this step will lead to a message box displaying a 'No program loaded' error.*

a. Enter the first instruction displayed in the CPU window which is indicated and adjacent arrow cursor. For the sample Control Program, this should be CLI.

You are now ready to progress with your debug session. Virtually all the features familiar to traditional (i.e. non-remote) debugging with Turbo Debugger are available.
You can single step the code or run to a pre-set breakpoint, set watches, display source windows, registers, memory etc.

9.3 Debugging Tips

Debugging a Control Program is a step by step process. If we consider a Control Program similar in structure to the sample version, CPC.C, then the first task is to ensure that the main processing loop is working properly using single stepping techniques and 'watching' variables, registers and memory to verify operation.

Then the task of debugging the handlers can begin. The CPC.C Control Program includes handlers to respond to serial port interrupts and to system to controller interrupts issued when a command has been queued.

Special care must be taken when debugging into the handlers.

Let us say that some modifications have been made to the Open function and need to be debugged.

The Open function is called by the System_isr in response to a system to controller interrupt, so the processing is done at interrupt time.
Once entered, the System_isr 'de-queues' the command message (passed from the host software), decodes the command calling the appropriate function via a jump table.

Assuming that the steps discussed in Section 9.2 have been carried out, but the Control Program is not running. (Status message in top right corner of user interface shows READY) Then set a breakpoint at a convenient point in the Open function (just before the new code) and run the control program (press F9). The status message now shows the message RUNNING.

From the host computer software issue the command to open the port. The system queues the message, interrupts the controller and the system_isr will call the open routine and the break point will be hit. The status message will show WAIT and then revert to READY. You can now start to verify/debug the code.

If you need to debug code that has been added to, say the *Receive Character Available (RCA)* interrupt service routine (ISR), then similar principles apply. After the set-up phase, place a breakpoint at the required point in the ISR and run the Control Program then cause the handler to be entered by sending a character into the port.

In general to debug a handler you set a breakpoint in the ISR, then run the Control Program. To hit the breakpoint you must generate an event to cause the ISR to be entered.
If the ISR is not entered then the breakpoint is not hit use the DEBUG switch to break into the Control Program (but note Section 9.4).

9.4 Non Supported Turbo Debugger Feature

One of the Turbo Debugger features not available is the ability to use the <CTRL> BREAK keys to stop a running program that has entered an endless loop.

To get around this a reset/debug push button box can be attached to the InterGrator i/PCi header P10 (silk screened DEBUG on the InterGrator i/PCi PCB).

The push button box supports two switches. The Reset switch allows a hardware reset of the InterGrator i/PCi controller independent of the host computer. This switch should not normally be used in a Turbo Debugger session.

The other switch is a Debug switch. This switch can be used in an attempt to break the Control Program out of an endless loop, for example, by causing an NMI interrupt.
The NMI handler will hand control back to the Turbo Debugger remote kernel at some point in the program.

A typical scenario is as follows :

The code being debugged is run to a pre-set breakpoint but due to a programming error, the breakpoint is never hit. Turbo Debugger displays the RUNNING status (top right corner of the user screen) continuously. At this point the Debug switch can be depressed to return control to Turbo Debugger.

However, it cannot be guaranteed that a situation as just described can be recovered with 100% certainty. The reason for this is that the Control Program and/or the V53 Processor registers may have become corrupted as a result of the original programming error.

10. Terminal or Firmware Debugger

This feature has been ported over from the Hostess i 8/16 controller. This is an alternative to the Turbo Debugger remote kernel. It is a simple debug tool which supports the following features :

- Display/change memory
- Disassembles instructions
- Performs input and output to I/O ports
- Displays registers
- Single steps
- Sets breakpoints

10.1 Connecting to the Terminal Debugger

The debugger is accessed by attaching an ASCII terminal or a terminal emulator (e.g. Windows Hyperterm) via a null modem cable to a port on the interface box. By default the first port (physical Port 0) is used.

A Terminal debugger cable diagram is shown below. This cable is for connecting between the Interface box and a PC's COM port. The cable is terminated with a DB25M at one end and a DB9F at the other.

Terminal Debugger Cable		
DB25 Male (InterGrator i/PCi interface)		**DB9 Female (PC's COMx Port)**
2	● ——— ●	2
3	● ——— ●	3
7	● ——— ●	5

Table 48 - Terminal Debugger Cable

The Terminal debugger communications parameters are :

- 9600 baud
- 8 bits/char
- No parity
- 1 stop bit

10.2 Invoking the Terminal Debugger

The firmware debugger essentially operates as an interrupt service routine (ISR). The InterGrator i/PCi firmware provides access to debugger functions through the following software interrupts.

These assembler statements are coded into the control program.

- **Terminal Debugger (int 20h)**
  A program can invoke the debugger through this interrupt. The firmware configures the first port as the debug console during system initialisation.

- **Terminal Debug Port (int 22h)**
  After the system initialises, a program can change the debug console by executing this interrupt. The CPU AL register is set up with the physical port number (00 – 0Fh) prior to the software interrupt.

10.3 Terminal Debugger Command Summary

Command	Name	Description
B	Byte	Succeeding operations work in byte mode
D	Dump	Displays specified memory range
E	Edit	Edit byte at specified location
F	Fill	Fill specified memory range with specified value
G	Go	Continue execution from current CS:IP with or without breakpoint
I	Input	I/P & display byte or word from specified I/O location
O	Output	O/P's byte or word to specified I/O location
R	Register	Display CPU registers
T	Trace	Single step
U	Unassemble	Disassemble specified memory range
W	Word	Succeeding operations work in word mode

Table 49 - Terminal Debugger Command Summary

10.4 Terminal Debugger Command Syntax

The terminal debugger displays a sign on message and the minus prompt '-'. The following table describes the syntax.

Command	Name	Command Line Syntax
B	Byte	B
D	Dump	D <start segment:offset> <end offset> *or* D <start segment:offset> l <length> *(If end offset or length not specified then 128 bytes dumped)*
E	Edit	E <segment:offset> <byte value> *(1 byte substitution)* E<start segment:offset> *(multi-byte substitution – press space to edit next byte)*
F	Fill	F<start segment:offset> <end offset> <byte val> or F<start segment:offset> l <length> <byte val>
G	Go	G *(go – no break point)* *or* G <break point segment:offset>
I	Input	I <I/O port address>
O	Output	O <I/O port address> <o/p value>
R	Register	R
T	Trace	T
U	Unassemble	U <start segment:offset> <count> *or* U <start segment:offset> *(If no count specified then 16 instruction disassembly)*
W	Word	W

Table 50 - Terminal Debugger Command Syntax

Notes :

1. The segment nmenonic (eg. DS , ES) may be given as the *segment* specifier for D, E, F & U commands.

2. The data value for the I and O commands will be either a byte or word depending on whether the B or W command was most recently invoked.

3. When using the disassembly :

   (i) Jump instructions display the next instructions address as a relative address, not as an absolute address.
   (ii) Non 8086 instructions do not disassemble correctly but do execute correctly. These instructions appear as * data *.
   (iii) Timer, system and SCC interrupts continue to occur when using the terminal debugger.

These ISR's cannot make any assumptions about the state of any registers. This includes the segment registers, which the firmware debugger modifies for it's own use. Because of this, the ISR's must save and initialise the registers and the registers must be restored before exiting the ISRs.

Appendix A – Firmware User Area (FUA)

The FUA is located at the segment:offset 0000:0B80h with respect to the onboard V53 processor. It is 128 bytes in size.
The FUA is accessed during the download and Control Program start up procedure. The sample software is loaded starting at the first free location after the FUA.

Offset	Description	Use	Length (bytes)
0h	Interaction flag 55Aah = controller active	Download	2
2h	Reserved (area 1)	N/A	2
4h	Unused (area 1)	N/A	2
6h	Firmware Release Number	information	8
0Eh	Available for control program release number	information	8
16h	Reserved (area 2)	N/A	2
18h	Reserved (area 3)	N/A	2
1Ah	Controller Ram Map one bit set per 64K found available by POST	Information/ diagnostic	2
1Ch	Reserved (area 4)	N/A	2
1Eh	Controller SCC Port Map one bit set per port found by POST	Information/ diagnostic	2
20h	Reserved (area 5)	N/A	2
22h	Identification number	information	4
26h	Invalid interrupt flag. *(Normally 00h Set to 01h if an invalid interrupt has occurred. Replaced by Invalid interrupt count)**	Diagnostic	1
27h	Invalid interrupt type *(Normally 00h. Type number of last invalid interrupt which occurred)**	Diagnostic	1
28h	Invalid interrupt count *(Normally 0000h. Number of invalid interrupts that occurred. If ffffh reached them terminal debugger is automatically invoked)**	Diagnostic	2

Table Continued

2Ah	Heartbeat counter incremented by CPC control program. *User can implement similar function.*	Diagnostic	4
2Eh	Firmware utility command	Download	1
2Fh	Firmware utility status	Information	1
30h	Firmware utility message buffer	Download	16
40h	Controller pending interrupt flag Set by CPC control program, tested & cleared by IDEM application *(via WinDriver).*	Status	2
42h	Reserved (area 6)	N/A	62

Table A1 - Firmware User Area (continued)

** Invalid interrupt handling. The boot EEPROM installs default handlers for all unused S/W and H/W interrupts. These handlers call a common ISR service routine which updates these fields.*

The interaction flag is used to signal that the InterGrator i/PCi controller is ready for download. It is also used by the downloader and Control Program as a handshake to signal control program start-up. Refer to Section 5.3 for download information.

The host computer software initiates the command by first writing the appropriate values in the Firmware utility fields and then generating a host computer system to InterGrator i/PCi controller interrupt (SYSINT).

The SYSINT is generated by setting and then clearing the USER I/O 2 data bit of the PLX9052 CNTRL register. This register is a member of the PLX9052's RTR registers. It is located at offset 50h from the start of these registers.
The RTR is available in both I/O and Memory address space.

The following table lists the firmware utility commands.

Command	**Action**
01h	Executes a control program. The segment:offset start address is in the firmware message buffer 30h = Segment 32h = offset The control may signal status by writing to the firmware utility status field.
02h	A Copy command that uses the following message buffer parameters 30h = Source segment 32h = Source offset 34h = Destination segment 36h = Destination offset 38h = Transfer count (2 bytes) When complete the firmware utility status field is set to 1 (copy complete)
00h 03h – ffh	Reserved. The controller will ignore these commands (executes null command handler)

Table A2 - Firmware Utility Commands

Appendix B - Description of SSCI Functions.

The SSCI is a library that interfaces to the Jungo device driver's API. Refer to the sample software for examples of use.

The functions fall into the following groups.

- Initialisation.
- I/O transfers.
- Memory (dual port ram) Read/Write.
- Block Transfers.
- Terminate (Close).

Initialisation.
Initialisation functions must be called before any I/O or memory transfers.
Within the initialisation routine the sWinDriver function is called before the sFindIntergrator function.

sWinDriver_Open
Prototype:
BOOL sWinDriver_Open(BYTE *errcode)

Include file:
#include “ssci.h”

Description:
Opens WinDriver device. This function must be called first.
Returns 1 = Open successful, 0 = Open failed

The errcode pointer argument is used to return a value indicating the reason for failure.
1 = Open failed because WinDriver not loaded.
2 = Open failed, WinDriver version earlier than 4.0

---

sFind_Intergrator
int sFind_Intergrator(ADAPTER_INFO *adapter_info,
BYTE *bCardsInstalled)

Include file:
#include “ssci.h”

Description:
This function must be called after sWinDriver_Open but before I/O or memory transfer functions.
The function scans the PCI bus for InterGrator i/PCi controllers. If none are found the function returns 0.
If at least one controller is found then the function returns 1 and updates *bCardsInstalled with the number of controllers found.
The argument *adapter_info is a pointer to an array of structures of type ADAPTER_INFO. The member variables of these structures are updated with the memory, I/O and IRQ resource assignments.

I/O Transfers

sIO_inp

BYTE sIO_inp(DWORD dwIOAddr)

Include file:

#include “ssci.h”

Description:

Return byte read from 8 bit I/O location dwIOAddr.

---

sIO_inpw

WORD sIO_inpw(DWORD dwIOAddr)

Include file:

#include “ssci.h”

Description:

Return word read from 16 bit I/O location dwIOAddr.

---

sIO_inpd

DWORD sIO_inpd(DWORD dwIOAddr)

Include file:

#include “ssci.h”

Description:

Return double word read from 32 bit I/O location dwIOAddr.

Note: Use only to access PLX9052 I/O mapped RTR registers.

sIO_outp

void sIO_outp (DWORD dwIOAddr, BYTE bData)

Include file:

#include “ssci.h”

Description:

Writes 8 bit byte bData to I/O location dwIOaddr

sIO_outpw

void sIO_outpw (DWORD dwIOAddr, WORD wData)

Include file:

#include “ssci.h”

Description:

Writes 16 bit word wData to I/O location dwIOaddr

sIO_outpd

void sIO_outpd (DWORD dwIOAddr, DWORD dwData)

Include file:

#include “ssci.h”

Description:

Writes 32 bit word dwData to I/O location dwIOaddr

Note: Use only to access PLX9052 I/O mapped RTR registers.

Memory (dual port ram) Read/Write

sMem_inbyte

UCHAR sMem_inbyte(DWORD dwAddr)

Include file:

#include “ssci.h”

Description:

Returns the 8 bit byte read from memory address dwAddr.

---

sMem_inword

USHORT sMem_inword(DWORD dwAddr)

Include file:

#include “ssci.h”

Description:

Returns the 16 bit word read from memory address dwAddr.

---

sMem_indword

DWORD sMem_indword(DWORD dwAddr)

Include file:

#include “ssci.h”

Description:

Returns the 32 bit double word read from memory address dwAddr.

Note: Use only to access PLX9052 memory mapped RTR registers.

sMem_outbyte

void sMem_outbyte(DWORD dwAddr, UCHAR Byte_val)

Include file:

#include “ssci.h”

Description:

Writes 8 bit byte, Byte_val to memory location dwAddr

---

sMem_outword

void sMem_outword(DWORD dwAddr,
USHORT Word_val)

Include file:

#include “ssci.h”

Description:

Writes 16 bit word, Word_val to memory location dwAddr

---

sMem_outdword

void sMem_outdword(DWORD dwAddr,
DWORD Dword_val)

Include file:

#include “ssci.h”

Description: Writes 32 bit double word, Dword_val to memory location dwAddr

Note: Use only to access PLX9052 memory mapped RTR registers.

Block Transfers

sBlock_Write_Mem

```
DWORD sBlock_Write_Mem( DWORD dwDestAddr,
                        UCHAR *pucSrcBuf,
                          DWORD  dwCount )
```

Include file:

#include “ssci.h”

Description:

Writes dwCount bytes from user buffer pointed to by *pucSrcBuf to dual port ram starting at location dwDestAddr. dwCount is decremented after each transfer. dwDestAddr and pucSrcBuf are incremented.

---

sBlock_Read_Mem

```
DWORD sBlock_Read_Mem( UCHAR *pucDestBuf,
                       DWORD dwSrcAddr,
                         DWORD  dwCount )
```

Include file:

#include “ssci.h”

Description:

Reads dwCount bytes from dual port ram starting from address dwSrcAddr and writes them user buffer pointed to by *pucDestBuf. dwCount is decremented after each transfer. dwSrcAddr and pucDestBuf are incremented.

Terminate (Close)

sWinDriver_Close

void sWinDriver_Close(void)

Include file:

#include “ssci.h”

Description:

This function is called to close the WinDriver device at the end of the session.

Appendix C – Technical Specification

Environmental

System on

Air temperature	0 to 50°C
Humidity	8 to 80%

System off

Air temperature	-65 to 150°C
Humidity	20 to 80%

Altitude 0 to 3050M

Power

Heat output 15.2 watts

Current consumption

+5V	2.54A
+12V	0.01A
-12V	0.22A

Dimensions

Length	174mm
Width	99mm
Height	13mm

On Board Processor

NEC V53 with 1Mb dual ported RAM

Data Rate

Asynchronous	50 to 76.8 Kbp/s
Synchronous	200 Kbp/s

Interface Options

InterGrator UI

RS232	DB25 female
RS422	DB25 female
RS423	DB25 female
RS485	DB25 female
Current loop	DB25 female

RocketPort 16

RS232	DB25 female
RS232/422	DB25 female
RS232 (Rack mounted)	RJ45

MTBF

11.9 years

Certification

CE Marked

Appendix D - Useful URL's

These URL's and addresses are correct at time of writing.

Comtrol Europe Ltd http://www.comtrol.co.uk
Manufacturers of the InterGrator i/PCi. This site gives information on products services.
The support e-mail address is <support@comtrol.co.uk>

Jungo Ltd http://www.Jungo.com
WinDriver device driver toolkit for Windows NT 4 and Windows 95/98.
The InterGrator i/PCi sample software uses 30 day evaluation versions of the windrvr.sys (NT) and windrvr.vxd (95/98) kernel drivers.
Jungo brochures and order forms for the registered copy of the toolkit can be found in the developers package or alternatively from the above web site. *You must become a registered developer if you intend to include Jungo components in your product*

There is a link to obtain purchasing information.

PLX www.plxtech.com
Manufacturers of the PLX9052 PCI bus I/F chip used on the InterGrator i/PCi.
The URL for the Data sheet is
http://www.plxtech.com/products/9050/previews/9050.htm

The URL for the PLX home page is
http://www.plxtech.com

Zilog www.zilog.com
Manufacturers of the Z85230 ESCC serial communications controller.
The URL for the user guide is
http://www.zilog.com

Appendix E - Glossary

API	Application Programmers Interface
BIOS	Basic Input Output System
CNTRL	Control register (PLX9052).
DEBUG BACKPLATE	InterGrator i/PCi accessory used during development/debug phase.
DIL	Dual In-Line.
DMA	Direct Memory Access
EEPROM	Electrically Erasable Programmable Read Only Memory
ESCC	Enhanced Serial Communications Controller Also known as 85230 or Z85230 and sometimes just as the 'SCC'
FIFO	First In First Out
INTCSR	INTerrupt Control / Status Register (PLX9052 register)
IOCTL	Input Output ConTroL
Jungo	Jungo Ltd. Providers of the WinDriver 95/98/NT User mode device driver toolkit
MFC	Microsoft Foundation Classes
PCI	Peripheral Component Interconnect.

PIO	Programmable Input Output device
POST	Power On Self Test
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver Transmitter
UI	Universal Interface. (Comtrol Interface box option).

Appendix F - DEVELOPERS LICENCE AGREEMENT

This agreement covers the Driver Development Software for the Comtrol Europe Ltd InterGrator i/PCi 16 port Intelligent serial controller.

This software may also be referred to as the :-
Development Software, Developers Software, Developers Support Software, Developers Toolkit, DTK or Sample software either with or without explicit reference to the InterGrator i/PCi product.

By installing the InterGrator i/PCi Driver Development Software you are deemed to agree with the terms of the DEVELOPERS LICENSE AGREEMENT.

1. This agreement is between you and COMTROL where COMTROL refers to Comtrol Europe Ltd, Comtrol Corporation and any subsidiary company or companies of the aforementioned.
2. This agreement covers the use of the Driver Development Software (hereafter referred to as the SOFTWARE) with the Comtrol Europe Ltd InterGrator i/PCi serial communications controller (hereafter referred to as the HARDWARE).
3. The SOFTWARE supplied is protected by International Copyright Treaties.
4. The SOFTWARE may only be used to develop software products that will operate with Comtrol brand HARDWARE.
5. You may not reproduce nor distribute the SOFTWARE in SOURCE CODE FORM *(i.e. human readable)* whether in part or whole.
6. You may distribute executable code derived from the use of the SOFTWARE (with the exception of that described in clause 7), providing that the distribution media bears your copyright notice.
7. You may not distribute the JUNGO LTD (formerly KRFtech Ltd) WinDriver kernel drivers without the permission of the manufacturer JUNGO LTD (Tel +972-9-8870878)
8. You agree to hold COMTROL harmless from all claims, liability, and damage arising from any use of your products which include components derived from the SOFTWARE or HARDWARE.