SARWIN II
SUITE OF APPLICATIONS
PRELIMINARY
USERS MANUAL

MARCH 22, 1999

IDT

Powering What's Next

Chapter 1 Introduction

The purpose of this manual is to introduce the Windows 95/NT drivers and SARWIN II suite of applications for the IDT77252/IDT77222 ABR SAR.. Figure 1.0 shows where the driver and applications fit into the overall system environment.

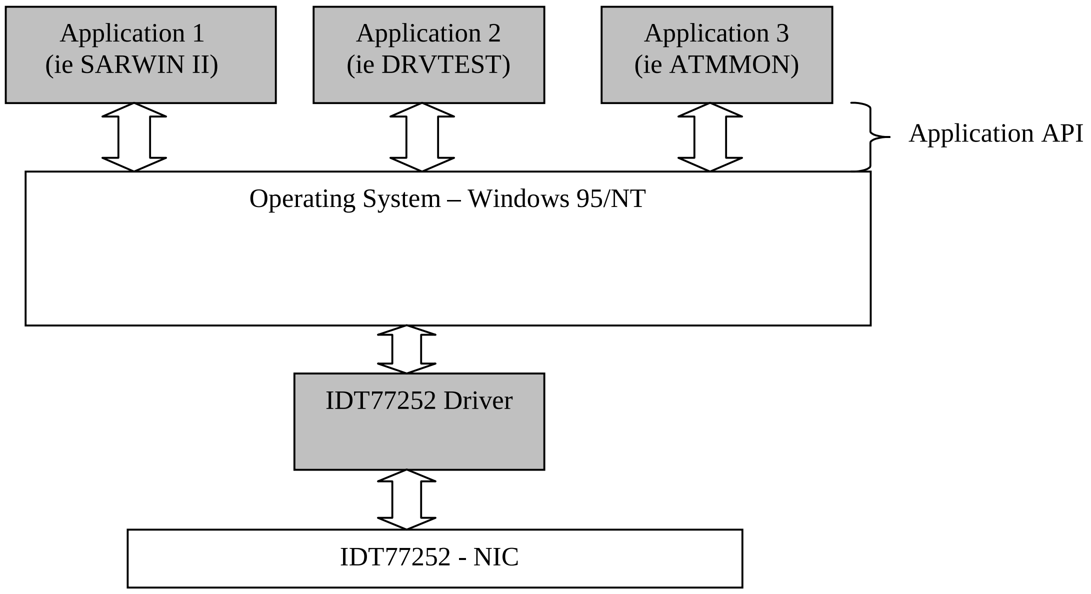

Application 1 (ie SARWIN II)
Application 2 (ie DRVTEST)
Application 3 (ie ATMMON)
Application API
Operating System – Windows 95/NT
IDT77252 Driver
IDT77252 - NIC

Figure 1.0

At the application level; applications can be developed that will act as traffic generators, diagnostics, timing analysis, statistical analysis, class of service selection and custom applications to deal with a particular application. An application that runs on 95 will work on NT and the other way around. The driver presents the same interface in either Windows environment.

1.1 Driver Application Programming Interface

The driver provides the following *device i/o control* calls to the application interface

- Open Connection
- Close Connection
- Read IDT77252/IDT77222 Internal Register
- Write IDT77252/IDT77222 Internal Register
- Read Utility Bus
- Write Utility Bus
- Read SRAM
- Write SRAM
- Retrieve List of Open Connections
- Query Connection Parameters
- Write Data to Connection
- Read Data from Connection
- Set Connection Mode
- Read PCI Configuration Space

- Write PCI Configuration Space
- Get Device Type, Revision and NIC SRAM size

1.2 Driver Parameters

This ABRSAR driver supports both the IDT77252 and IDT 77222 devices. When the driver starts it dynamically determines the device type and configures the internal data structures correctly. It also checks the revision level and determines if it is revision one or two. The number of simultaneous connections, the number of connections and the number and size of the rate tables are effected by the device type and revision. The following sections give the parameters for each configuration.

1.2.1 IDT77252 Revision 01

• Number of Connections	1024
• VPI Range	0-1
• VCI Range	0-511
• Number of Simultaneous Connections:	16
• Number of Rate tables	16

1.2.2 IDT77252 Revision 02

• Number of Connections	1024
• VPI Range	0-1
• VCI Range	0-511
• Number of Simultaneous Connections:	128
• Number of Rate tables	16

1.2.3 IDT77222 Revision 02

• Number of Connections	512
• VPI Range	0
• VCI Range	0-511
• Number of Simultaneous Connections:	128
• Number of Rate tables	4

NOTE:

Revision 01 = Revision A
Revision 02 = Revision B

1.3 Software Installation

The applications and driver are contained in a single self-extracting 'ZIP' file. Create a folder called 'sarwin' on your hard drive. Put the file called 'sarwin.exe' in this folder. Double click sarwin.exe. This will extract all of the SARWINII files.

1. Abrsar_reg.exe
2. Abrsar_sram.exe
3. Atmmon.exe
4. Cellgen.exe
5. Cellrecv.exe
6. Drvtest.exe
7. Idt77252.sys
8. Idt77252.vxd
9. Mfc42.dll
10. Monitor.exe
11. Msvcrt.dll
12. Vxdload.exe

The SARWINII suite of applications is:

1 Abrsar_reg.exe
2 Abrsar_sram.exe
3 Atmmon.exe
4 Cellgen.exe
5 Cellrecv.exe
6 Drvtest.exe

The device drivers for the IDT77252/IDT77222 NIC are:

1 Idt77252.sys – Windows NT 4.0 driver.
2 Idt77252.vxd – Windows 95 driver.

The device driver loaders are:

1 Monitor.exe – Windows NT 4.0.
2 Vxdload.exe – Windows 95.

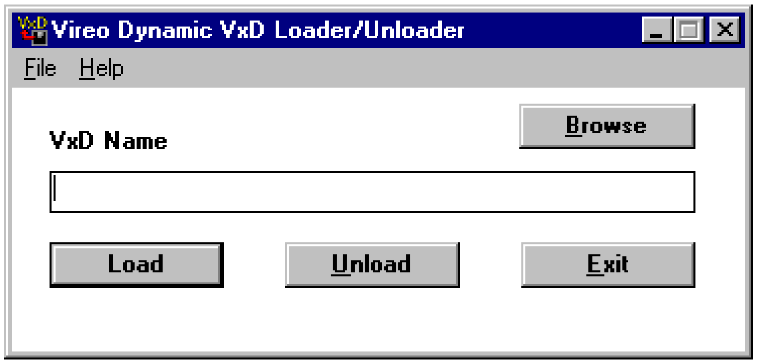
The remaining two files are required *dll's* for the *SARWINII* suite of applications. If theses two *dll's* are not already in your *Windows\system*(Windows 95) or *Winnt\system32*(Windows NT) directory, then copy them to either *Windows\system*(Windows 95) or *Winnt\system32*(Windows NT).

1.4 Driver Installation

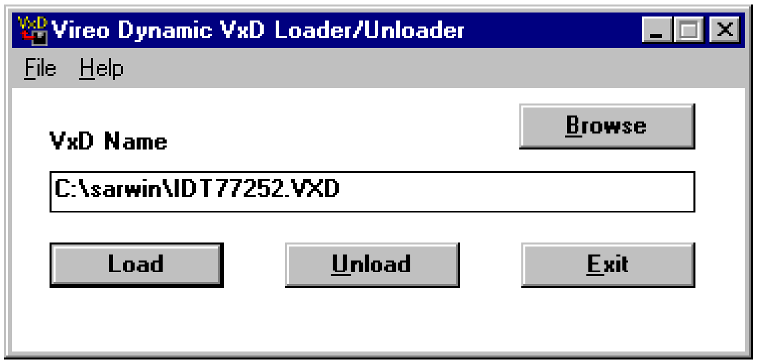
This driver is configured to be loaded and unloaded manually. This means the driver will be loaded after the operating system (either Windows 95 or Windows NT 4.0) has booted. The procedure for loading and unloading is slightly different between Windows 95 and Windows NT. Before installing the driver it is necessary to install the NIC. To do this, first turn off your PC and verify that there is a free PCI slot. If there is, insert the NIC into the free PCI slot and reboot your PC. You may run the NIC in loop back mode by looping the receive to the transmit. If the NIC has a fiber transceiver, then just run a cable from receive to transmit. If the transceiver is twisted pair then you will need a special loop back cable.

1.4.1 Driver Installation – Windows 95

On Windows 95 the plug and play mechanism should find the new card and inform you with a dialog box saying it found a new PCI card. Continue and indicate that you will install the driver later. Later versions of Windows 95 will show a window with 'NEXT' and 'Cancel' buttons. In this case the correct thing is to hit 'NEXT' and then in the next window click the 'Finish' button. To verify that the plug and play mechanism has registered the newly installed NIC; reboot your PC and this time there should be no message stating that a new PCI card was found. Use the program *Vxdload.exe* to load the driver. To invoke *Vxdload* go to directory *sarwin* and double click on the *Vxdload.exe* icon. This will bring up the dialog shown below.

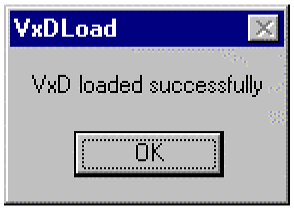


Use the browse button or type in the directory path to the folder that contains the driver. If you created a *sarwin* directory and put the driver in it, then the VxD Name should look as shown below.



C:\sarwin\IDT77252.VXD

Click the *load* button to load the driver. If the driver loaded successfully, then the message below will be displayed.

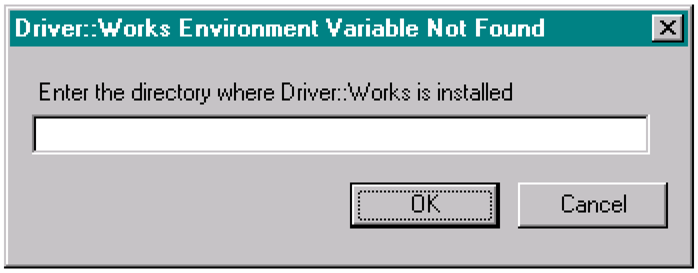


VxD loaded successfully

Click *OK* and dismiss the message box. You are now ready to run the *SARWINII* suite of applications. If you are using a loop back cable you will be able to send and receive cells. The driver may be stopped and unloaded by bringing up *VxDload.exe*, selecting C:\sarwin\IDT77252.vxd and clicking the *Unload* button. **NOTE**: You can not stop the driver if there are any of the SARWINII suite of applications open. It is necessary to stop all activity with the NIC and close the applications prior to stopping the driver.

1.4.2 Driver Installation – Windows NT 4.0

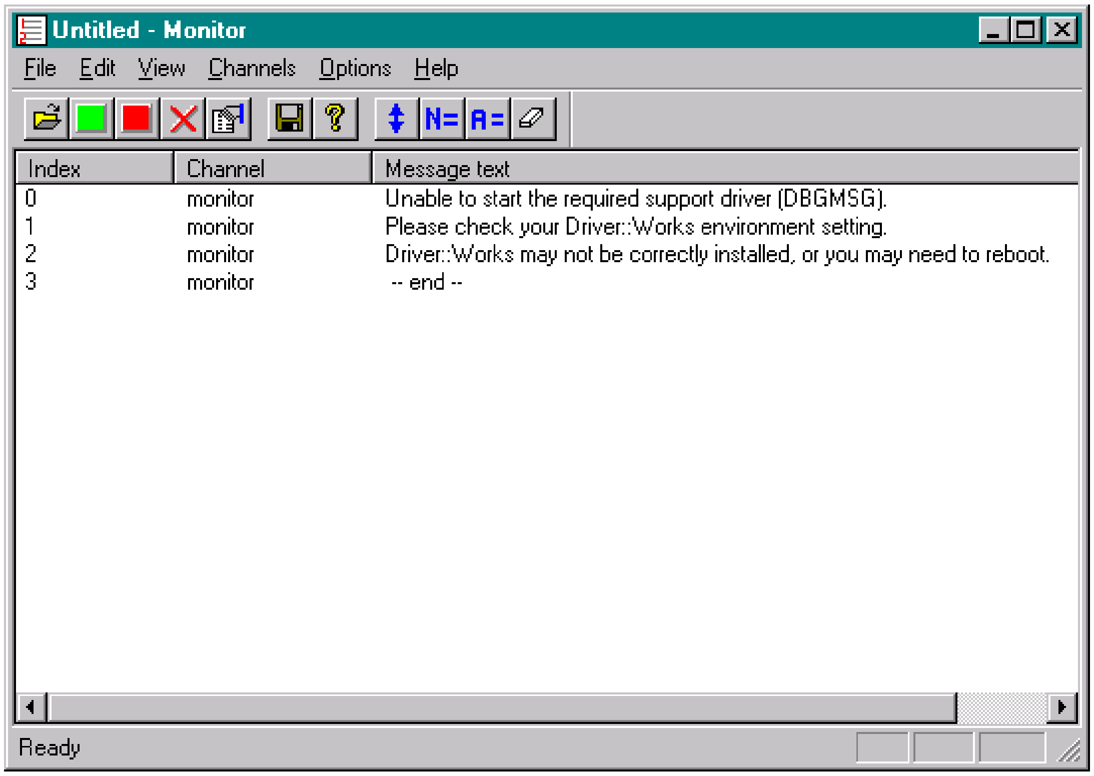
After inserting the NIC and rebooting and installing the software as outlined in section 1.3 you are ready to register the driver. Go into the directory C:\sarwin. Double click the Monitor.exe icon. This will bring up the window shown below.



Driver::Works Environment Variable Not Found

Enter the directory where Driver::Works is installed

You do not need Driver::Works installed to register the driver. Click *Cancel* and the screen shown below will be displayed.

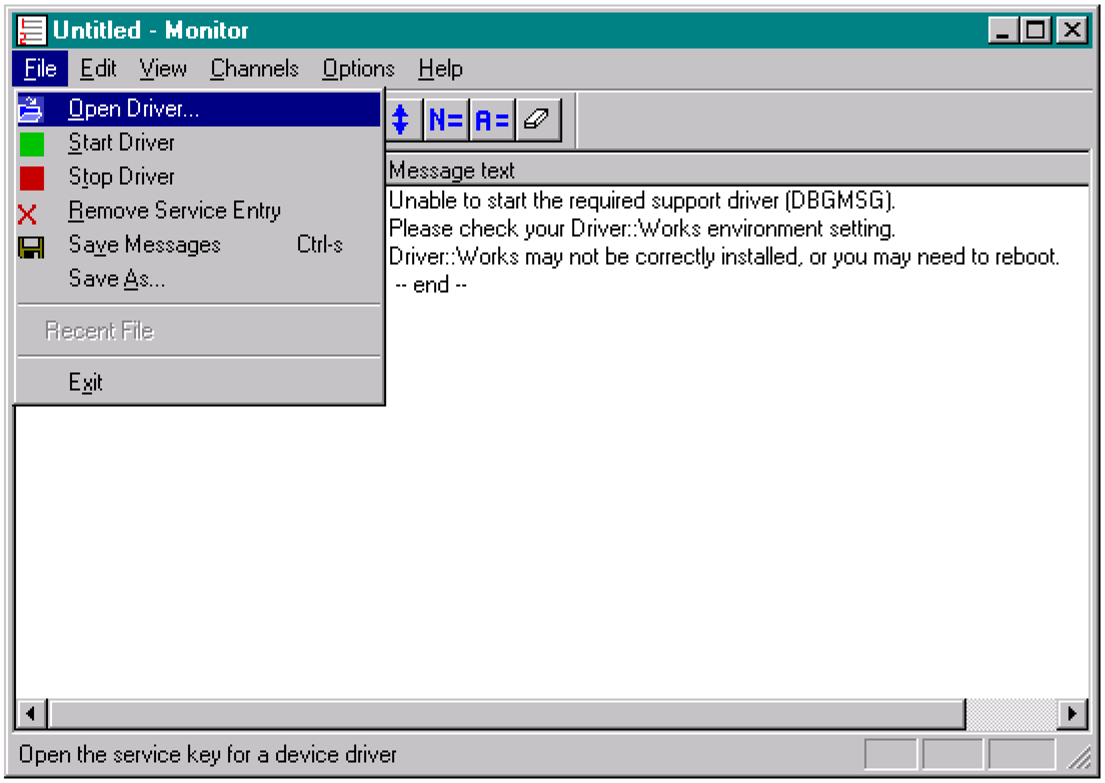


Untitled - Monitor

```
Index   Channel   Message text
0       monitor   Unable to start the required support driver (DBGMSG).
1       monitor   Please check your Driver::Works environment setting.
2       monitor   Driver::Works may not be correctly installed, or you may need to reboot.
3       monitor    -- end --
```

Ready

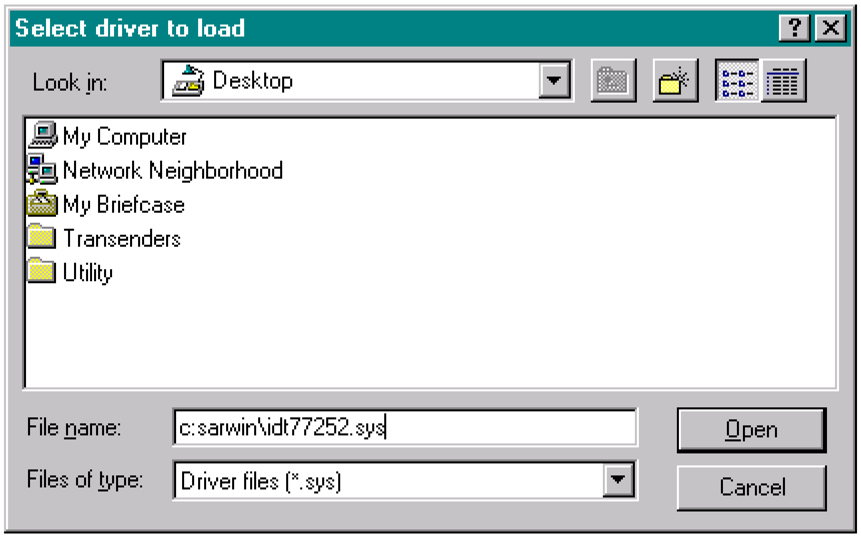
Select the *File* menu item and then select *Open Driver* as shown below.



```
Unable to start the required support driver (DBGMSG).
Please check your Driver::Works environment setting.
Driver::Works may not be correctly installed, or you may need to reboot.
 -- end --
```

Open the service key for a device driver

This selection will display the file selection dialog box shown below. The path to the driver should be entered in the edit box for *File Name*. If you created the *sarwin* directory then the entry as shown below will be correct.

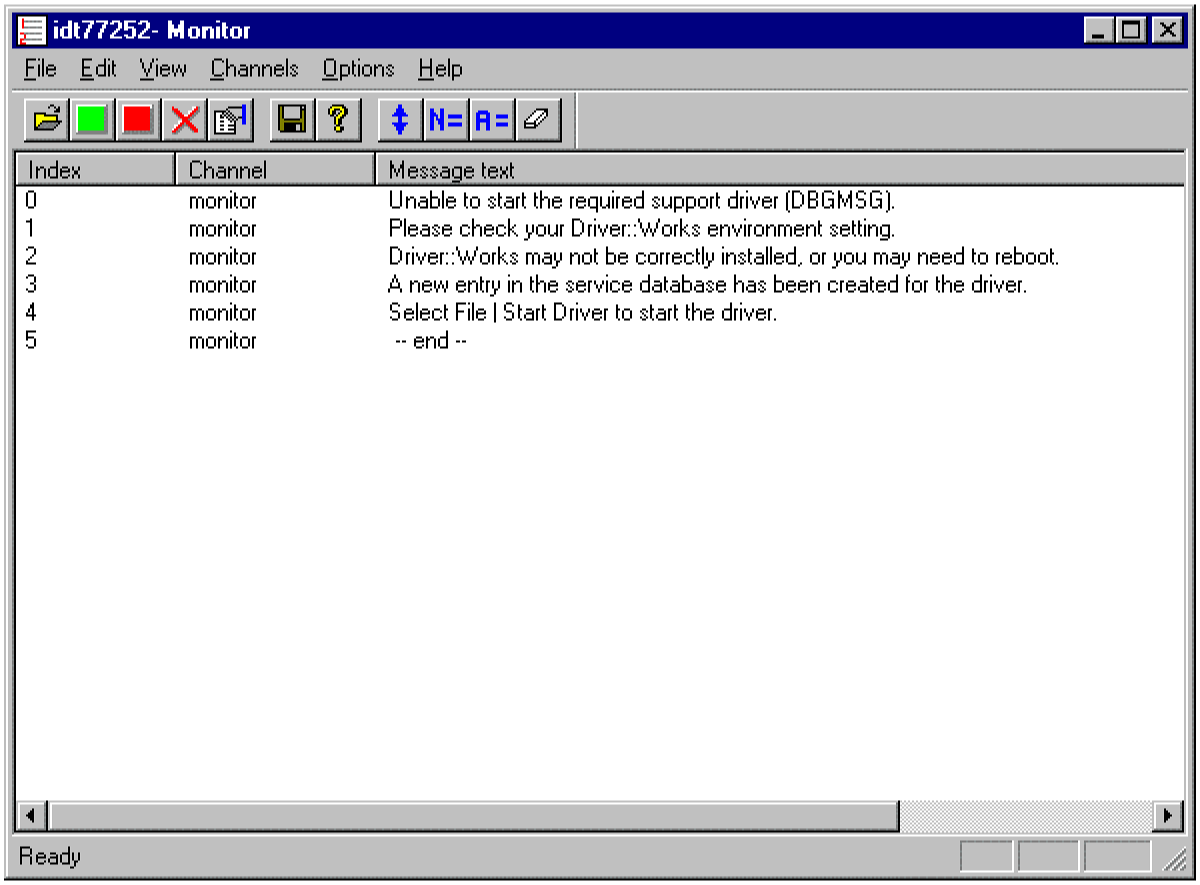


Look in: Desktop

File name: c:sarwin\idt77252.sys

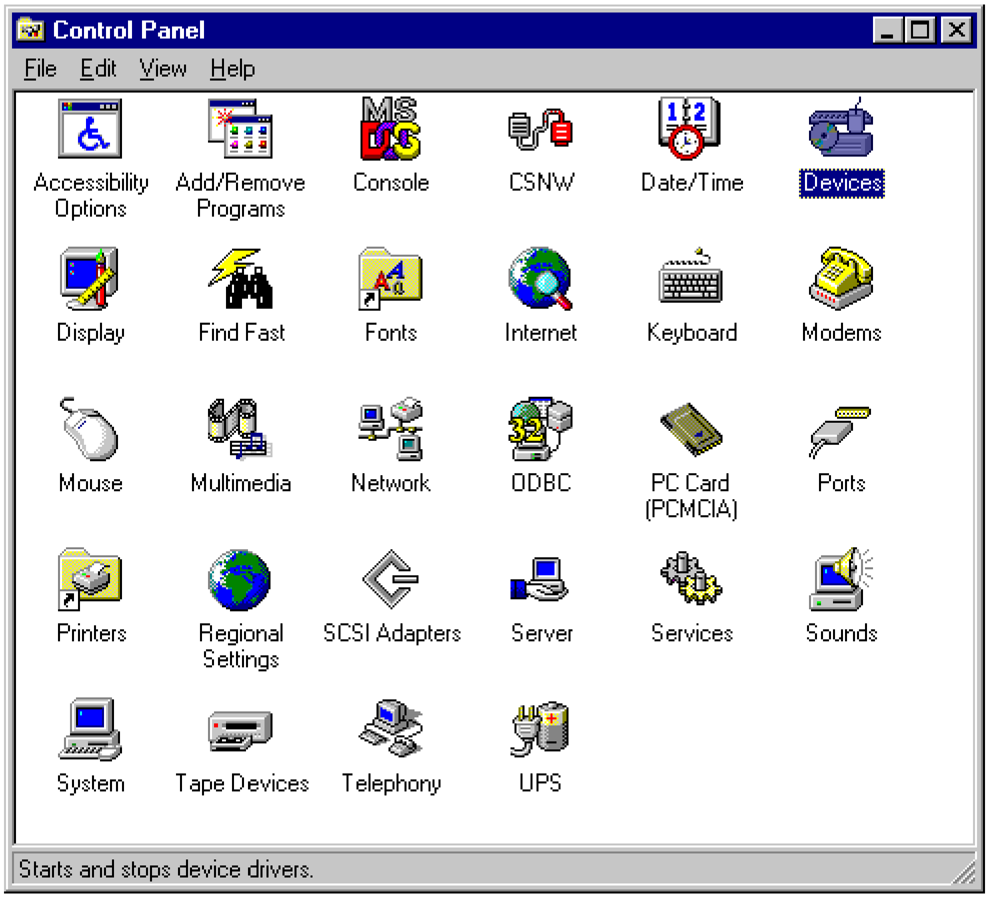
Files of type: Driver files (*.sys)

Click open and the driver will be registered and the screen should look as shown below.



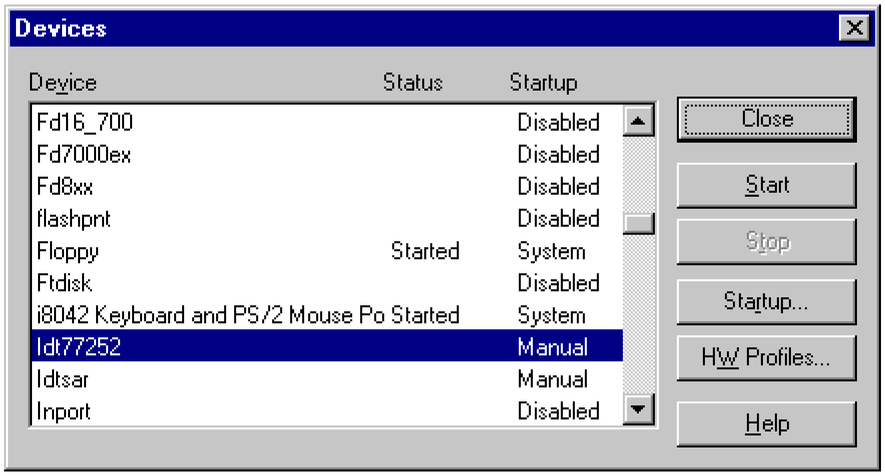
```
Index   Channel   Message text
0       monitor   Unable to start the required support driver (DBGMSG).
1       monitor   Please check your Driver::Works environment setting.
2       monitor   Driver::Works may not be correctly installed, or you may need to reboot.
3       monitor   A new entry in the service database has been created for the driver.
4       monitor   Select File | Start Driver to start the driver.
5       monitor    -- end --
```

This shows *A new Entry in the service data base has been created for the driver.* You may follow the directions and select File | Start to start the driver now. The preferred method is to just exit monitor.exe by closing the window, without starting the driver. You can now use the standard NT facilities for starting and stopping the driver. To do this, open the control panel as shown below.



Starts and stops device drivers.

Double click the *Devices* icon. This will bring up the following dialog box.



Device	Status	Startup
Fd16_700		Disabled
Fd7000ex		Disabled
Fd8xx		Disabled
flashpnt		Disabled
Floppy	Started	System
Ftdisk		Disabled
i8042 Keyboard and PS/2 Mouse Po	Started	System
Idt77252		Manual
Idtsar		Manual
Inport		Disabled

It may be necessary to scroll down so the *idt77252* device is visible. Select it by clicking on the line. To load and start the driver, click the *Start* button. Once the driver is started you may unload and stop it by again bringing up the *Devices* dialog box, selecting *idt77252* and clicking *Stop*. **NOTE**: You can not stop the driver if there are any of the SARWINII suite of applications open. It is necessary to stop all activity with the NIC and close the applications prior to stopping the driver.

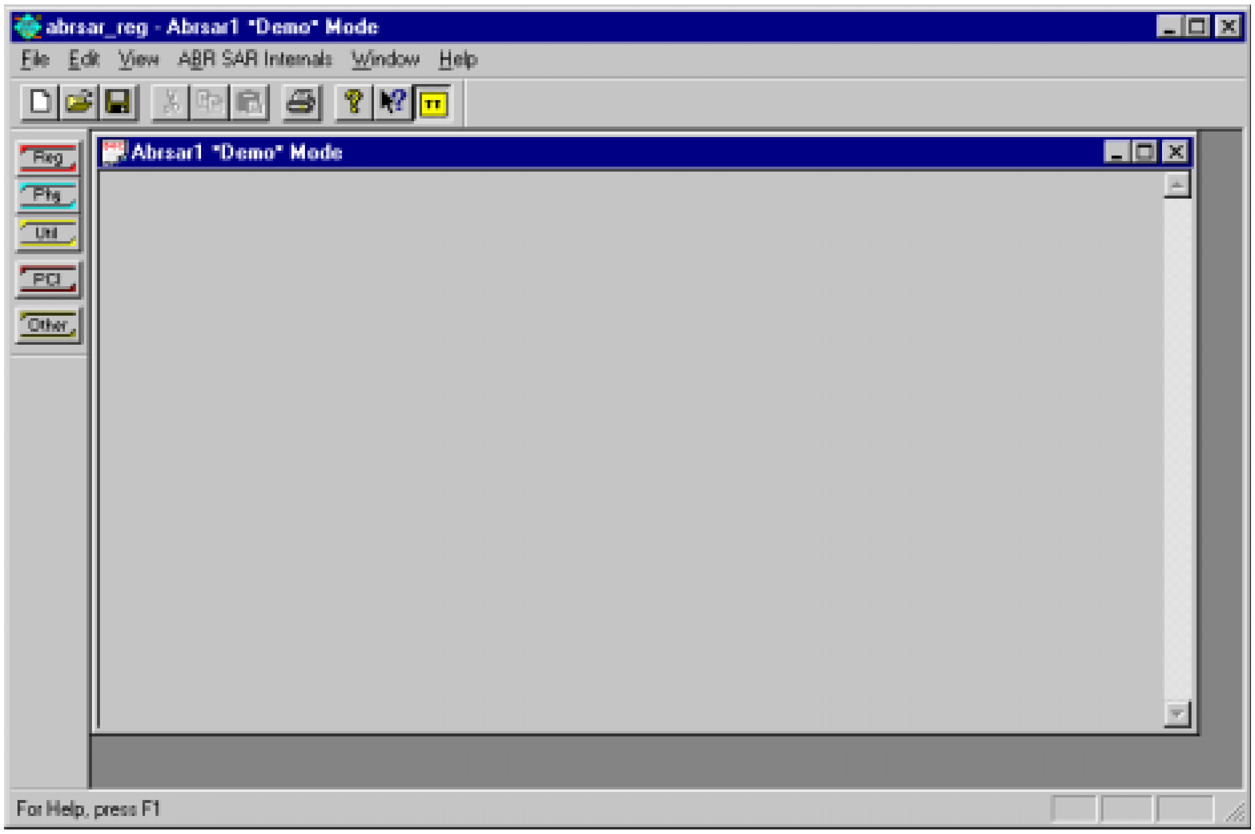
Chapter 2 SARWINII Suite of Applications

The SARWINII suite of applications allow the user to access all of the internal registers, PCI configuration space, and associated SRAM. The SARWINII suite of applications are:

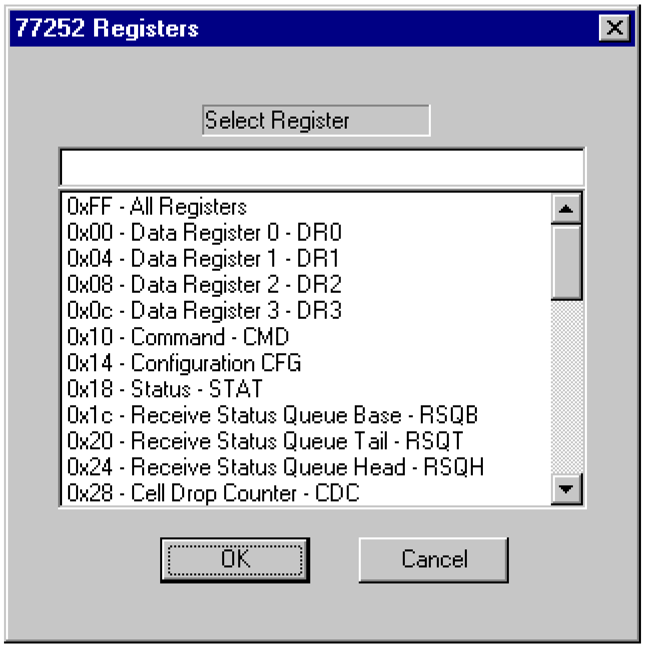
1. Abrsar_reg.exe
2. Abrsar_sram.exe
3. Drvtest.exe
4. Cellgen.exe
5. Cellrecv.exe
6. Atmmon.exe

2.1 Abrsar_reg.exe – Internal Register Access of the ABRSAR

Abrsar_reg.exe allows the user to read and write the registers of the IDT77252/IDT77222 ABRSAR. To launch this application go to the \sarwin directory and double click the abrsar_reg.exe icon. This will bring up the screen shown below. This application can be launched with or without the driver running. If the driver is not running then it goes into *demo* mode and generates random numbers for the register contents.



NOTE: If the driver is running then it is recommended that you do **NOT** write to any of the internal registers. When the driver starts it initializes the ABRSAR and all of the internal registers. Many of these registers point to memory in the host machine that has been allocated out of the system non-paged memory pool. Changing any of these values could **crash** your PC. Clicking on any of the buttons along the left hand side of the window will bring up dialog boxes that will allow access to the associated item. As an example, lets look at the configuration register of the ABRSAR. Click the *Reg* button at the left-hand edge. The following dialog box will be displayed.

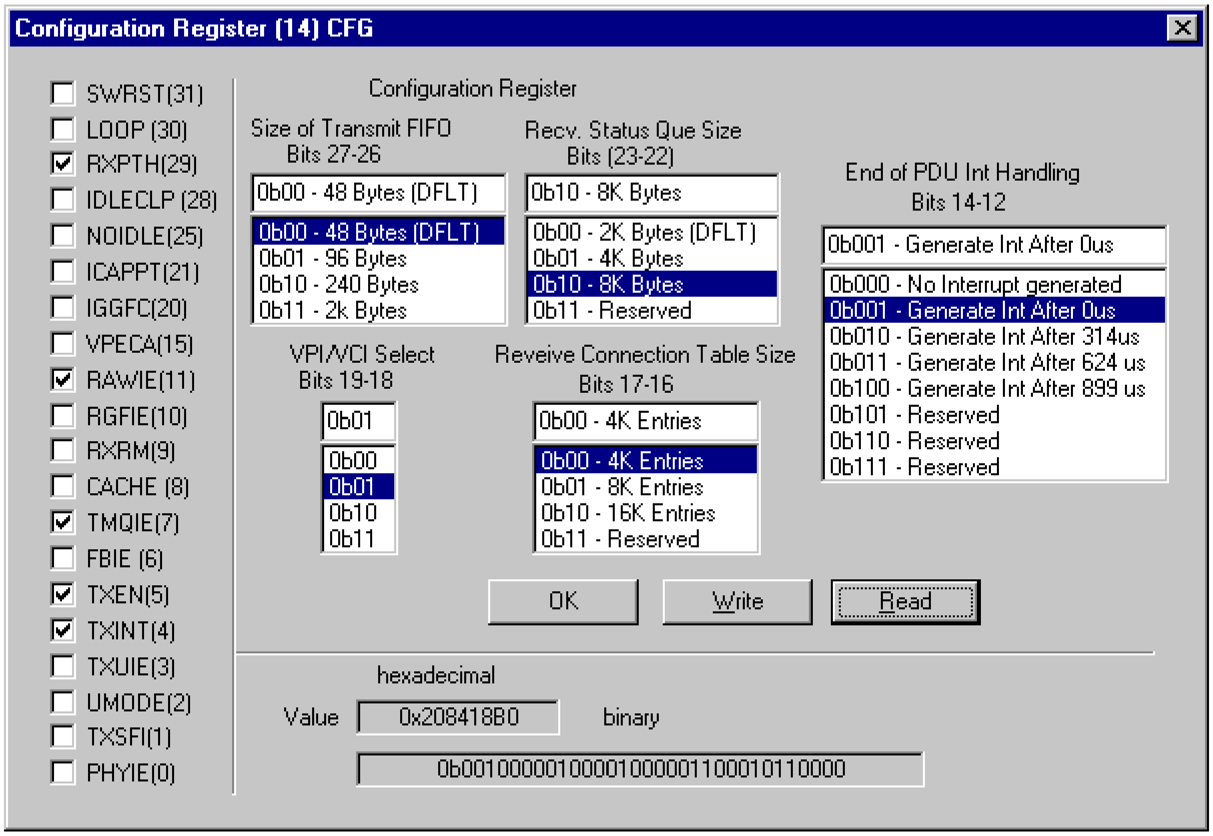

77252 Registers

Select Register

0xFF - All Registers
0x00 - Data Register 0 - DR0
0x04 - Data Register 1 - DR1
0x08 - Data Register 2 - DR2
0x0c - Data Register 3 - DR3
0x10 - Command - CMD
0x14 - Configuration CFG
0x18 - Status - STAT
0x1c - Receive Status Queue Base - RSQB
0x20 - Receive Status Queue Tail - RSQT
0x24 - Receive Status Queue Head - RSQH
0x28 - Cell Drop Counter - CDC

2.1.1 Abrsar_reg – Register Access

Select the configuration register by clicking the line that starts with *0x14*. Click *OK*. The configuration dialog box will be displayed.


Configuration Register (14) CFG

- [ ] SWRST(31)
- [ ] LOOP (30)
- [✓] RXPTH(29)
- [ ] IDLECLP (28)
- [ ] NOIDLE(25)
- [ ] ICAPPT(21)
- [ ] IGGFC(20)
- [ ] VPECA(15)
- [✓] RAWIE(11)
- [ ] RGFIE(10)
- [ ] RXRM(9)
- [ ] CACHE (8)
- [✓] TMQIE(7)
- [ ] FBIE (6)
- [✓] TXEN(5)
- [✓] TXINT(4)
- [ ] TXUIE(3)
- [ ] UMODE(2)
- [ ] TXSFI(1)
- [ ] PHYIE(0)

Configuration Register

Size of Transmit FIFO Bits 27-26: 0b00 - 48 Bytes (DFLT)
- [✓ 0b00 - 48 Bytes (DFLT)]
- 0b01 - 96 Bytes
- 0b10 - 240 Bytes
- 0b11 - 2k Bytes

Recv. Status Que Size Bits (23-22): 0b10 - 8K Bytes
- 0b00 - 2K Bytes (DFLT)
- 0b01 - 4K Bytes
- [✓ 0b10 - 8K Bytes]
- 0b11 - Reserved

End of PDU Int Handling Bits 14-12: 0b001 - Generate Int After 0us
- 0b000 - No Interrupt generated
- [✓ 0b001 - Generate Int After 0us]
- 0b010 - Generate Int After 314us
- 0b011 - Generate Int After 624 us
- 0b100 - Generate Int After 899 us
- 0b101 - Reserved
- 0b110 - Reserved
- 0b111 - Reserved

VPI/VCI Select Bits 19-18: 0b01
- 0b00
- [✓ 0b01]
- 0b10
- 0b11

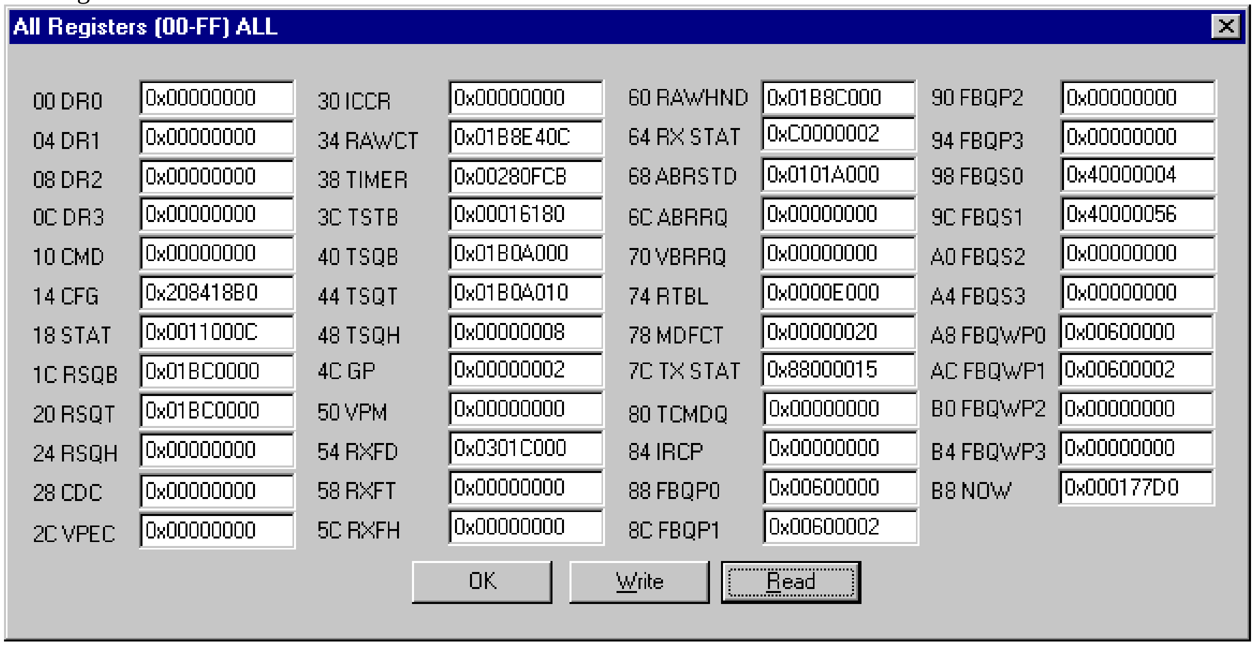
Reveive Connection Table Size Bits 17-16: 0b00 - 4K Entries
- [✓ 0b00 - 4K Entries]
- 0b01 - 8K Entries
- 0b10 - 16K Entries
- 0b11 - Reserved

hexadecimal

Value 0x208418B0

binary 0b00100000100001000001100010110000

With the driver running; the configuration shown above will be set for the 77252 ABRSAR. It is necessary to click the read button to get the register contents displayed. To access the other register the user should repeat the above procedure, selecting different registers. To look at all of the internal registers on one screen the user may select the *0xFF – All Registers* entry in the *77252 Registers* dialog box. This will display the following dialog box.

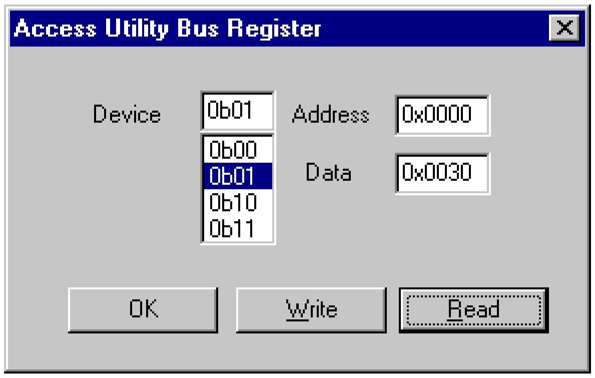


00 DR0	0x00000000	30 ICCR	0x00000000	60 RAWHND	0x01B8C000	90 FBQP2	0x00000000
04 DR1	0x00000000	34 RAWCT	0x01B8E40C	64 RX STAT	0xC0000002	94 FBQP3	0x00000000
08 DR2	0x00000000	38 TIMER	0x00280FCB	68 ABRSTD	0x0101A000	98 FBQS0	0x40000004
0C DR3	0x00000000	3C TSTB	0x00016180	6C ABRRQ	0x00000000	9C FBQS1	0x40000056
10 CMD	0x00000000	40 TSQB	0x01B0A000	70 VBRRQ	0x00000000	A0 FBQS2	0x00000000
14 CFG	0x208418B0	44 TSQT	0x01B0A010	74 RTBL	0x0000E000	A4 FBQS3	0x00000000
18 STAT	0x0011000C	48 TSQH	0x00000008	78 MDFCT	0x00000020	A8 FBQWP0	0x00600000
1C RSQB	0x01BC0000	4C GP	0x00000002	7C TX STAT	0x88000015	AC FBQWP1	0x00600002
20 RSQT	0x01BC0000	50 VPM	0x00000000	80 TCMDQ	0x00000000	B0 FBQWP2	0x00000000
24 RSQH	0x00000000	54 RXFD	0x0301C000	84 IRCP	0x00000000	B4 FBQWP3	0x00000000
28 CDC	0x00000000	58 RXFT	0x00000000	88 FBQP0	0x00600000	B8 NOW	0x000177D0
2C VPEC	0x00000000	5C RXFH	0x00000000	8C FBQP1	0x00600002		

Click the *Read* button repeatedly and you should see the *Timer* and *Now* register changing. If you don't see this behavior then either the driver is not running or there is a problem.

2.1.2 Abrsar_reg – PHY Access

To access the PHY the user should click the *Phy* button along the left-hand edge of the window. This will bring up the dialog box shown below.

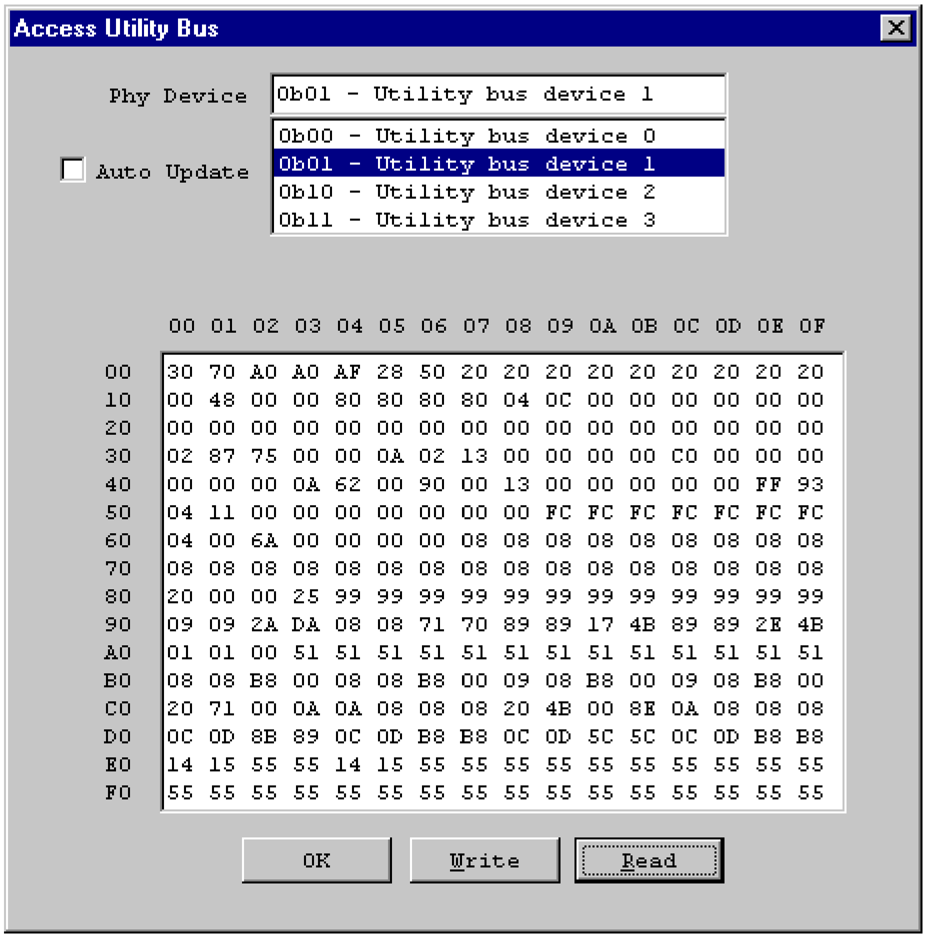


Device: 0b01
Address: 0x0000
Data: 0x0030

The PHY on the NicStAR is device 0b01. If the user selects 0b00 then there will be a message displayed stated that no device exists. The Address range is 0-255.

2.1.3 Abrsar_reg – Utility Bus Access

To display all of the address space on the utility bus the user should click the *Util* button and the following dialog box will be displayed



Phy Device: 0b01 - Utility bus device 1

```
0b00 - Utility bus device 0
0b01 - Utility bus device 1
0b10 - Utility bus device 2
0b11 - Utility bus device 3
```

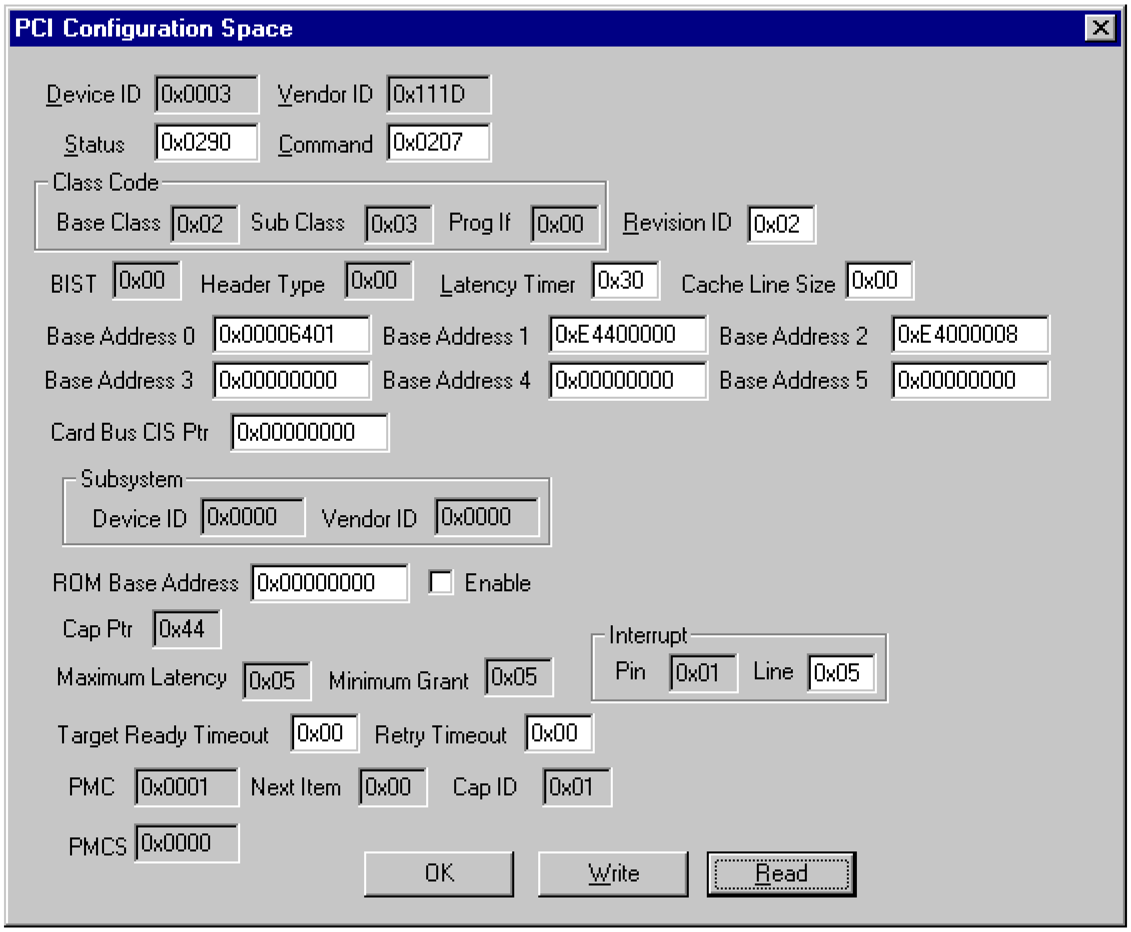
```
     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00   30 70 A0 A0 AF 28 50 20 20 20 20 20 20 20 20 20
10   00 48 00 00 80 80 80 80 04 0C 00 00 00 00 00 00
20   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30   02 87 75 00 00 0A 02 13 00 00 00 00 C0 00 00 00
40   00 00 00 0A 62 00 90 00 13 00 00 00 00 00 FF 93
50   04 11 00 00 00 00 00 00 00 FC FC FC FC FC FC FC
60   04 00 6A 00 00 00 00 08 08 08 08 08 08 08 08 08
70   08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08
80   20 00 00 25 99 99 99 99 99 99 99 99 99 99 99 99
90   09 09 2A DA 08 08 71 70 89 89 17 4B 89 89 2E 4B
A0   01 01 00 51 51 51 51 51 51 51 51 51 51 51 51 51
B0   08 08 B8 00 08 08 B8 00 09 08 B8 00 09 08 B8 00
C0   20 71 00 0A 0A 08 08 08 20 4B 00 8E 0A 08 08 08
D0   0C 0D 8B 89 0C 0D B8 B8 0C 0D 5C 5C 0C 0D B8 B8
E0   14 15 55 55 14 15 55 55 55 55 55 55 55 55 55 55
F0   55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
```

Again the user should select *0b01* and then click read. This will display the entire address space. If the user checks *Auto Update* then the utility bus will be read approximately once a second and the screen updated. This will continue until the user clears the *Auto Update* check box or closes the dialog box..

2.1.4 Abrsar_reg – PCI Configuration Access

To display the PCI configuration space the user should click the *Phy* button on the left hand side of the window. This will display the dialog box below.

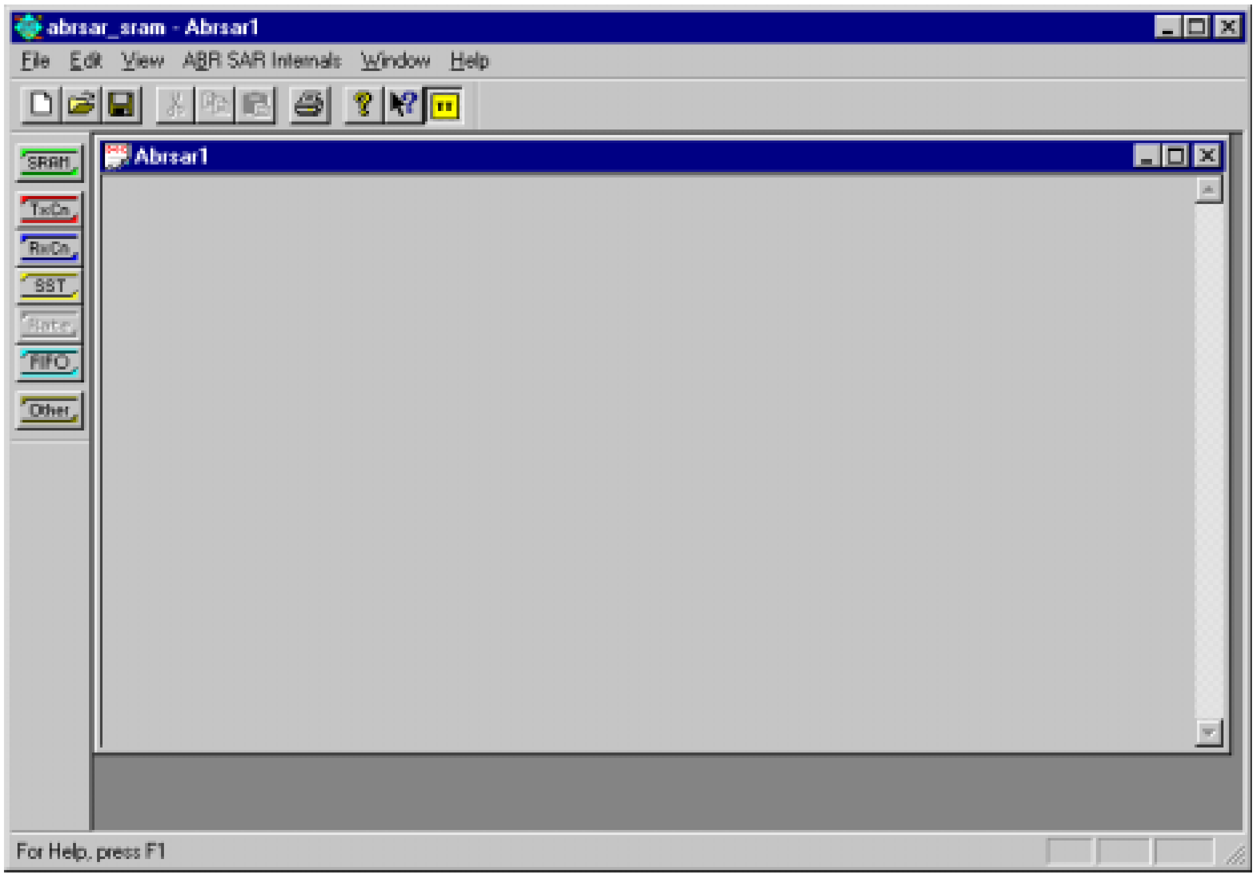

PCI Configuration Space

Device ID 0x0003 Vendor ID 0x111D
Status 0x0290 Command 0x0207
Class Code: Base Class 0x02 Sub Class 0x03 Prog If 0x00
Revision ID 0x02
BIST 0x00 Header Type 0x00 Latency Timer 0x30 Cache Line Size 0x00
Base Address 0 0x00006401 Base Address 1 0xE4400000 Base Address 2 0xE4000008
Base Address 3 0x00000000 Base Address 4 0x00000000 Base Address 5 0x00000000
Card Bus CIS Ptr 0x00000000
Subsystem: Device ID 0x0000 Vendor ID 0x0000
ROM Base Address 0x00000000 ☐ Enable
Cap Ptr 0x44
Interrupt: Pin 0x01 Line 0x05
Maximum Latency 0x05 Minimum Grant 0x05
Target Ready Timeout 0x00 Retry Timeout 0x00
PMC 0x0001 Next Item 0x00 Cap ID 0x01
PMCS 0x0000

The shaded fields are read only. The rest can be written. **NOTE:** Even though these registers can be written, the user should exercise great care when changing any of the values. Addresses should **NOT** be changed. Reference the PCI specification for an in depth description of the fields.

2.2 Abrsar_sram.exe – SRAM access application

Abrsar_sram.exe allows the user to access the SRAM associated with the NicStAR. To launch this application; go to the *sarwin* directory and double click the abrsar_sram.exe icon. This will bring up the screen shown below. This application can be launched with or without the driver running. If the driver is not running then abrsar_sram.exe goes into *demo* mode and generates random numbers for the SRAM content.



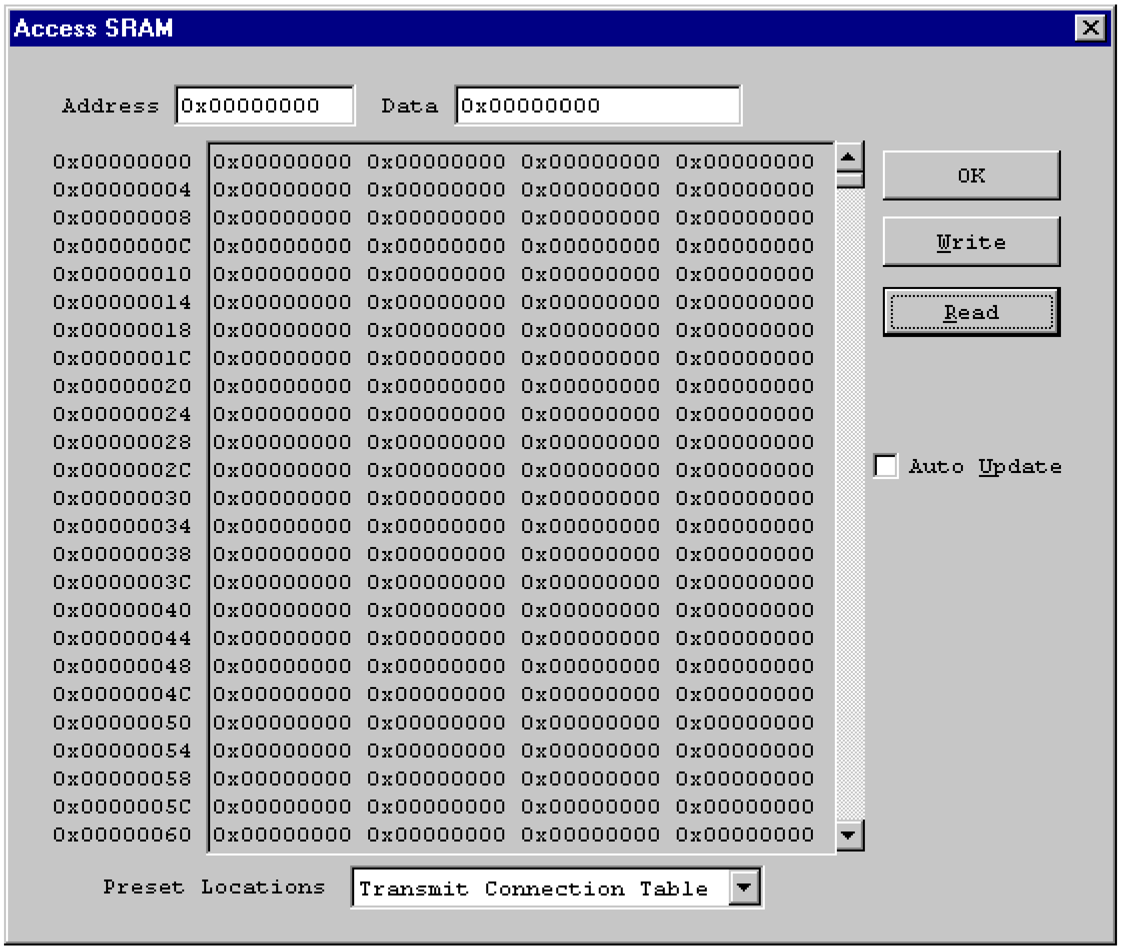
NOTE: If the driver is running then it is recommended that you do **NOT** write to the SRAM. When the driver starts it initializes the ABRSAR and all of the associated SRAM. Most of the SRAM is allocated for use when the driver starts. Changing SRAM could cause the driver to do unexpected operations and could possibly crash your system. Clicking on any of the buttons along the left-hand side of the window will bring up dialog boxes that will allow access to the associated item. The buttons are:

• SRAM	Read and write SRAM at a specified address.
• TxCn	Access the Transmit Connection Table.
• RxCn	Access the Receive Connection table.
• SST	Access the Static Schedule Table
• FIFO	Access the Receive FIFO
• Other	In the final release there will be an *Launcher* associated with this button

The *Other* button is not useful in this release.

2.2.1 Abrsar_sram – SRAM Access

To display SRAM unformatted (raw hex memory dump) the user should click the SRAM button on the left-hand side of the window. The dialog box below will be displayed.



Address 0x00000000 Data 0x00000000

```
0x00000000  0x00000000 0x00000000 0x00000000 0x00000000
0x00000004  0x00000000 0x00000000 0x00000000 0x00000000
0x00000008  0x00000000 0x00000000 0x00000000 0x00000000
0x0000000C  0x00000000 0x00000000 0x00000000 0x00000000
0x00000010  0x00000000 0x00000000 0x00000000 0x00000000
0x00000014  0x00000000 0x00000000 0x00000000 0x00000000
0x00000018  0x00000000 0x00000000 0x00000000 0x00000000
0x0000001C  0x00000000 0x00000000 0x00000000 0x00000000
0x00000020  0x00000000 0x00000000 0x00000000 0x00000000
0x00000024  0x00000000 0x00000000 0x00000000 0x00000000
0x00000028  0x00000000 0x00000000 0x00000000 0x00000000
0x0000002C  0x00000000 0x00000000 0x00000000 0x00000000
0x00000030  0x00000000 0x00000000 0x00000000 0x00000000
0x00000034  0x00000000 0x00000000 0x00000000 0x00000000
0x00000038  0x00000000 0x00000000 0x00000000 0x00000000
0x0000003C  0x00000000 0x00000000 0x00000000 0x00000000
0x00000040  0x00000000 0x00000000 0x00000000 0x00000000
0x00000044  0x00000000 0x00000000 0x00000000 0x00000000
0x00000048  0x00000000 0x00000000 0x00000000 0x00000000
0x0000004C  0x00000000 0x00000000 0x00000000 0x00000000
0x00000050  0x00000000 0x00000000 0x00000000 0x00000000
0x00000054  0x00000000 0x00000000 0x00000000 0x00000000
0x00000058  0x00000000 0x00000000 0x00000000 0x00000000
0x0000005C  0x00000000 0x00000000 0x00000000 0x00000000
0x00000060  0x00000000 0x00000000 0x00000000 0x00000000
```

☐ Auto Update

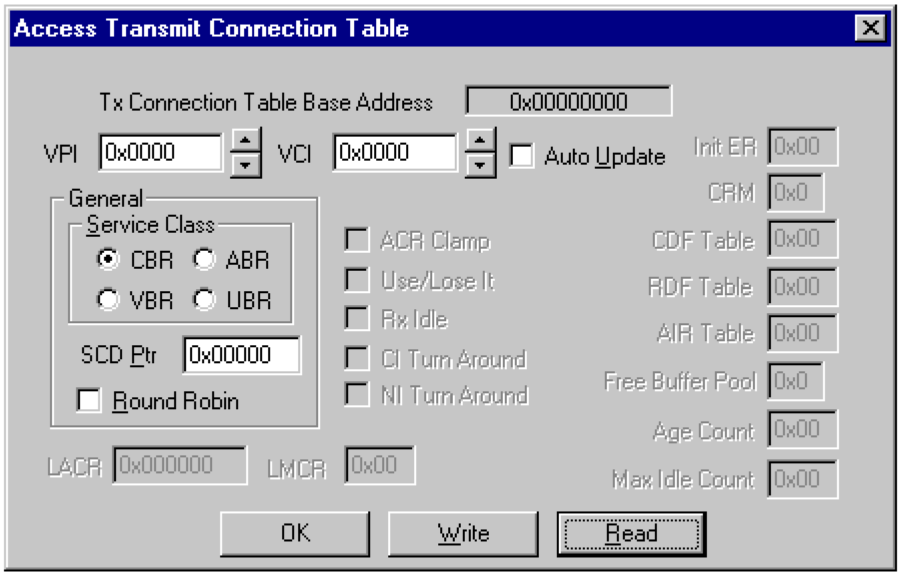
Preset Locations: Transmit Connection Table

The user should click read to read the SRAM. The *Preset Locations* at the bottom of the dialog box can be used to select and jump to the selected item. The Transmit Connection Table always starts at Address zero(0) in the NicStAR. The addresses are **32 bit word** addresses.

The rest of the buttons also access SRAM, however they format the display in the context of the function the SRAM space is allocated for.

2.2.2 Abrsar_sram – Transmit Connection Table Access

Selecting the *TxCn* button and reading connection index 0 will read the data from SRAM locations 0-7. The display would look as shown below.


Access Transmit Connection Table

Tx Connection Table Base Address: 0x00000000

VPI: 0x0000 VCI: 0x0000 ☐ Auto Update

General

Service Class: [✓ CBR] ABR VBR UBR

SCD Ptr: 0x00000

☐ Round Robin

☐ ACR Clamp
☐ Use/Lose It
☐ Rx Idle
☐ CI Turn Around
☐ NI Turn Around

Init ER	0x00
CRM	0x0
CDF Table	0x00
RDF Table	0x00
AIR Table	0x00
Free Buffer Pool	0x0
Age Count	0x00
Max Idle Count	0x00

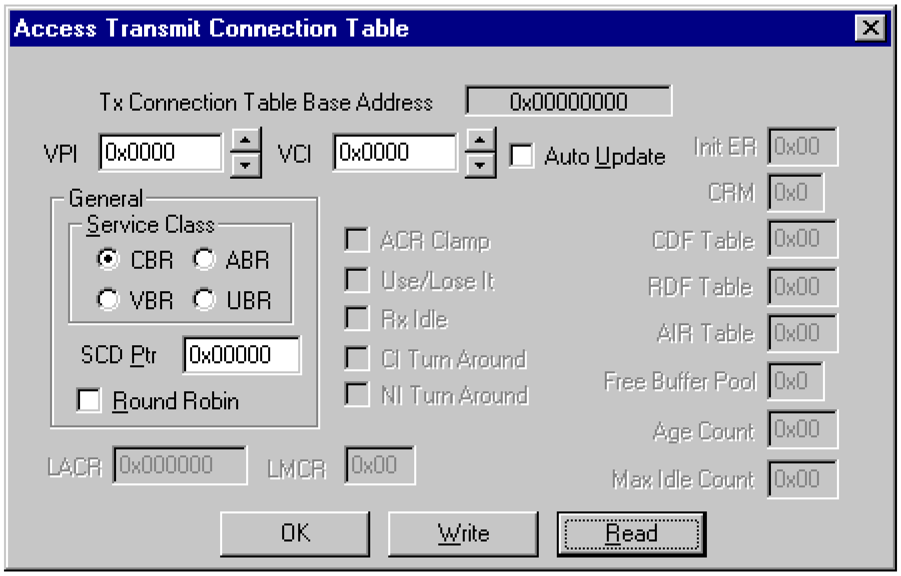
LACR: 0x000000 LMCR: 0x00

The values will be zeros until a connection is opened for the specified VPI/VCI. **NOTE:** Again – even though we give you the ability to write to the transmit connection table, it is almost guaranteed to create problems. The mapping of VPI-VCI to transmit connection table SRAM address depends on the configuration register settings. In that the driver sets this at initialization time, the following mapping will be true for this driver.

For The IDT77252:
SRAM Word Addr. = VPI*512*8 + VCI*8
Where: VPI can be 0 or 1 and VCI is 0-511
For The IDT77222:
SRAM Word Addr. = VCI*8
Where: VPI can only be zero and VCI is 0-511

Abrsar_sram – Receive Connection Table Access

To examine the receive connection table entry for a particular VPI/VCI the user should click the *RxCn* button and enter the desired VPI/VCI and then click *Read*. The dialog box will look like this.


Access Transmit Connection Table

Tx Connection Table Base Address: 0x00000000

VPI: 0x0000 VCI: 0x0000 ☐ Auto Update

General

Service Class: [✓ CBR] ABR VBR UBR

SCD Ptr: 0x00000

☐ Round Robin

☐ ACR Clamp
☐ Use/Lose It
☐ Rx Idle
☐ CI Turn Around
☐ NI Turn Around

Init ER	0x00
CRM	0x0
CDF Table	0x00
RDF Table	0x00
AIR Table	0x00
Free Buffer Pool	0x0
Age Count	0x00
Max Idle Count	0x00

LACR: 0x000000 LMCR: 0x00

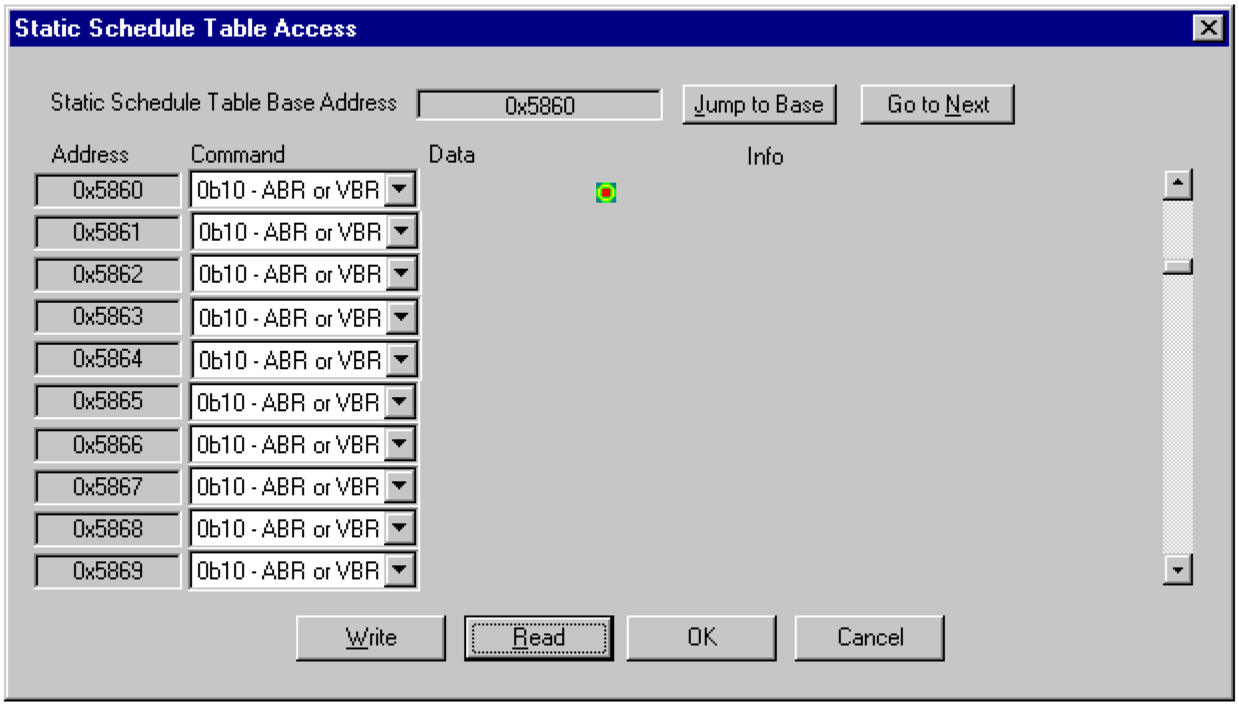
The mapping of VPI-VCI to transmit connection table SRAM address depends on the configuration register settings. In that the driver sets this at initialization time, the following mapping will be true for this driver.

For The IDT77252:
SRAM Word Addr. = VPI*512*4 + VCI*4 + 8192
Where: VPI can be 0 or 1 and VCI is 0-511
For The IDT77222:
SRAM Word Addr. = VCI*4 + 4096
Where: VPI can only be zero and VCI is 0-511

How to open a connection with one of the SAWINII apps will be shown later in the document;.

2.2.3 Abrsar_sram – Static Schedule Table(SST) Access

The static schedule table is created and maintained by the driver. To examine the static schedule table the user should click the *SST* button on the left-hand side of the window. This will display the dialog box below



Static Schedule Table Access

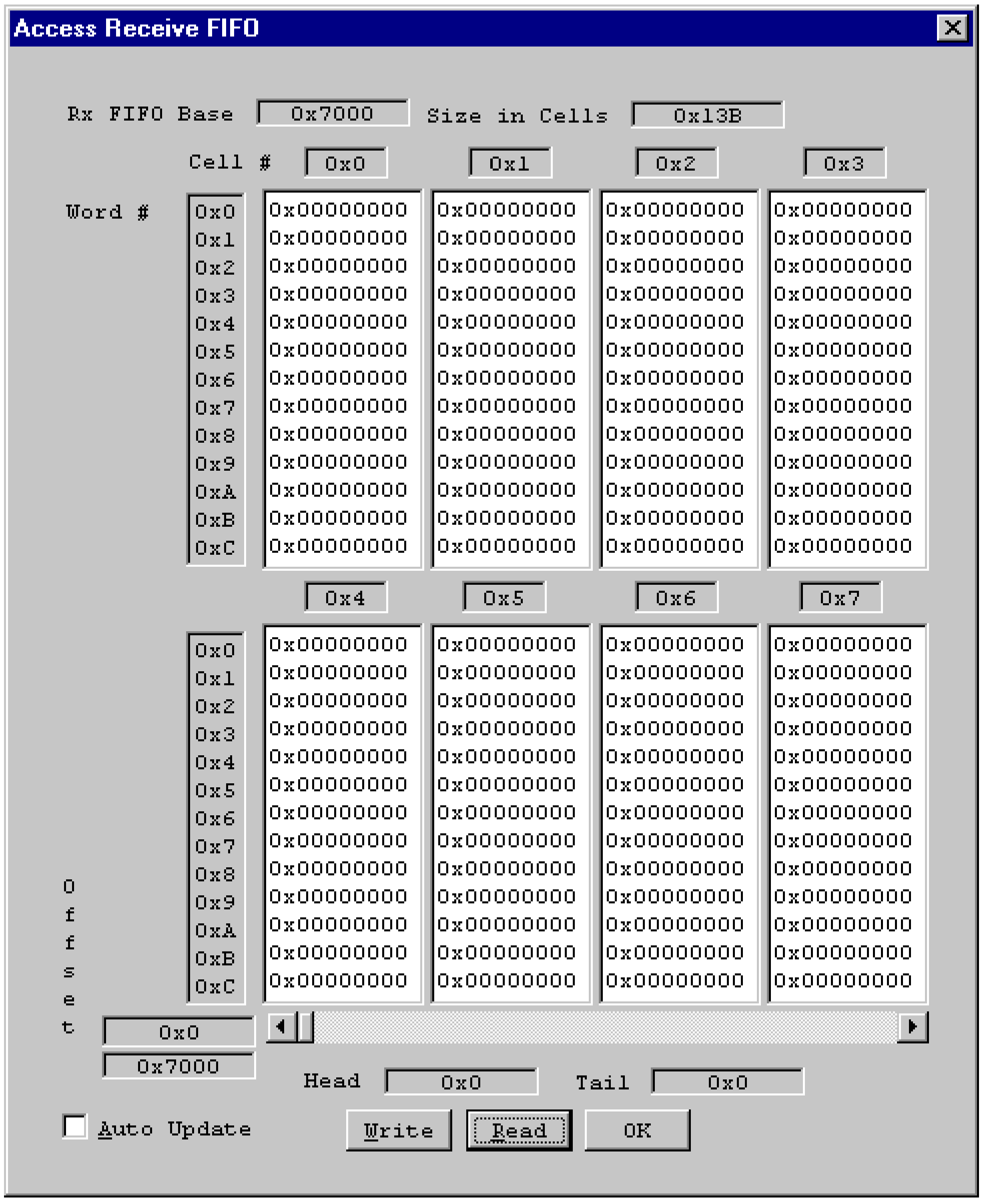
Static Schedule Table Base Address: 0x5860

Address	Command	Data	Info
0x5860	0b10 - ABR or VBR		
0x5861	0b10 - ABR or VBR		
0x5862	0b10 - ABR or VBR		
0x5863	0b10 - ABR or VBR		
0x5864	0b10 - ABR or VBR		
0x5865	0b10 - ABR or VBR		
0x5866	0b10 - ABR or VBR		
0x5867	0b10 - ABR or VBR		
0x5868	0b10 - ABR or VBR		
0x5869	0b10 - ABR or VBR		

The detailed description of the management of the SST is beyond the scope of this user manual. Refer to the IDT77252/222 User Manual.

2.2.4 Abrsar_sram – Receive FIFO Access

To access the contents of the receive FIFO the user should click the *FIFO* button on the left-hand side of the window. This will display the following dialog box.



Rx FIFO Base 0x7000 Size in Cells 0x13B

Word #	Cell # 0x0	0x1	0x2	0x3
0x0	0x00000000	0x00000000	0x00000000	0x00000000
0x1	0x00000000	0x00000000	0x00000000	0x00000000
0x2	0x00000000	0x00000000	0x00000000	0x00000000
0x3	0x00000000	0x00000000	0x00000000	0x00000000
0x4	0x00000000	0x00000000	0x00000000	0x00000000
0x5	0x00000000	0x00000000	0x00000000	0x00000000
0x6	0x00000000	0x00000000	0x00000000	0x00000000
0x7	0x00000000	0x00000000	0x00000000	0x00000000
0x8	0x00000000	0x00000000	0x00000000	0x00000000
0x9	0x00000000	0x00000000	0x00000000	0x00000000
0xA	0x00000000	0x00000000	0x00000000	0x00000000
0xB	0x00000000	0x00000000	0x00000000	0x00000000
0xC	0x00000000	0x00000000	0x00000000	0x00000000

	0x4	0x5	0x6	0x7
0x0	0x00000000	0x00000000	0x00000000	0x00000000
0x1	0x00000000	0x00000000	0x00000000	0x00000000
0x2	0x00000000	0x00000000	0x00000000	0x00000000
0x3	0x00000000	0x00000000	0x00000000	0x00000000
0x4	0x00000000	0x00000000	0x00000000	0x00000000
0x5	0x00000000	0x00000000	0x00000000	0x00000000
0x6	0x00000000	0x00000000	0x00000000	0x00000000
0x7	0x00000000	0x00000000	0x00000000	0x00000000
0x8	0x00000000	0x00000000	0x00000000	0x00000000
0x9	0x00000000	0x00000000	0x00000000	0x00000000
0xA	0x00000000	0x00000000	0x00000000	0x00000000
0xB	0x00000000	0x00000000	0x00000000	0x00000000
0xC	0x00000000	0x00000000	0x00000000	0x00000000

Offset 0x0
0x7000

Head 0x0 Tail 0x0

The user needs to click the *Read* button to read the contents of the receive FIFO. The scroll bar may be used to scoll through the contents. The receive FIFO is written to as cells are received by the hardware. The receive FIFO is set at 4k words in length by the driver. As cells come in the Tail pointer is updated and will wrap when it reaches 4k. As 4k is not divisible by 13 (The size of in incoming cell in words), the cells will appear to scroll through the cell windows.

The *Other* button will be activated in a later release.

2.3 Drvtest.exe - Demonstrate Driver Functionality

Drvtest.exe is an application that allows the user to interact with the NicStAR via the driver. It was initially developed to test the driver and is being released to give the user a simple interface to try some of the various driver API's.

The driver provides the following *device i/o control* calls to the application interface

- Open Connection
- Close Connection
- Read IDT77252/IDT77222 Internal Register
- Write IDT77252/IDT77222 Internal Register
- Read Utility Bus
- Write Utility Bus
- Read SRAM
- Write SRAM
- Retrieve List of Open Connections
- Query Connection Parameters
- Write Data to Connection
- Read Data from Connection
- Read PCI Configuration Space
- Write PCI Configuration Space
- Get Device Type, Revision and NIC SRAM size

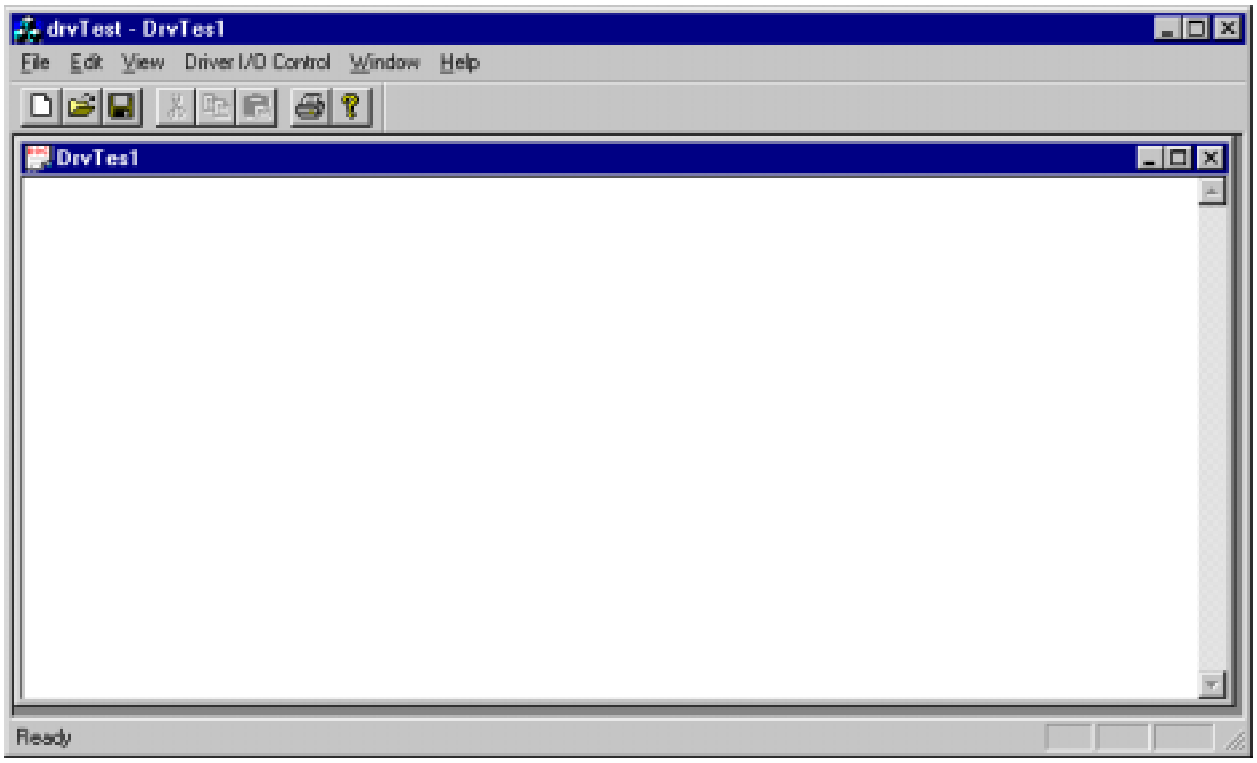
Abrsar_reg.exe and Abrsar_sram.exe exercise the following API's

- Read IDT77252/IDT77222 Internal Register
- Write IDT77252/IDT77222 Internal Register
- Read Utility Bus
- Write Utility Bus
- Read SRAM
- Write SRAM
- Read PCI Configuration Space
- Write PCI Configuration Space

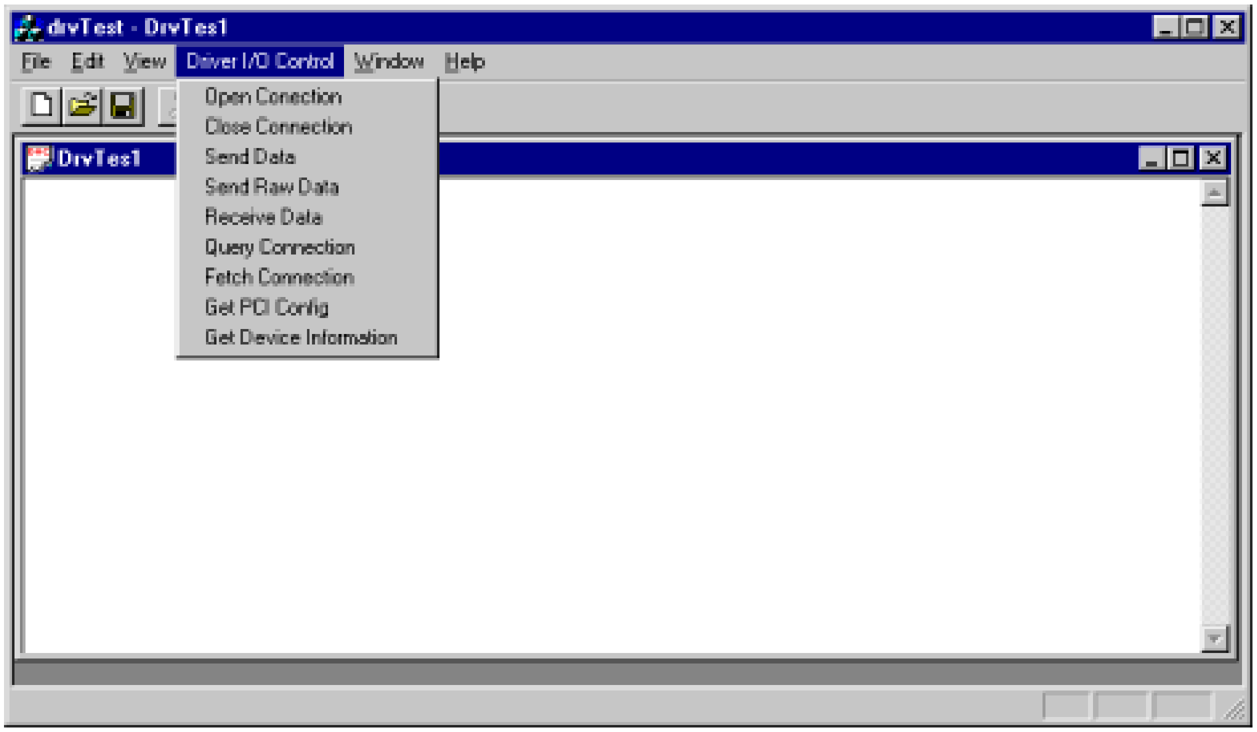
Drvtest.exe exercises the following API's

- Open Connection
- Close Connection
- Retrieve List of Open Connections
- Query Connection Parameters
- Write Data to Connection
- Read Data from Connection
- Read PCI Configuration Space
- Get Device Type, Revision and NIC SRAM size

To launch Drvtest.exe go to the *\sarwin* directory and double click the Drvtest.exe icon. The window shown below will be displayed.

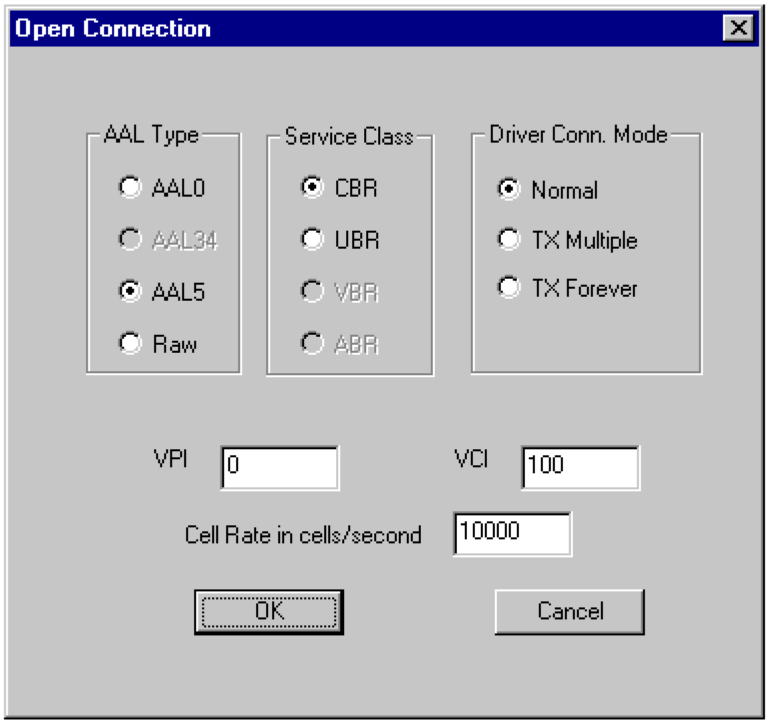


Click on the *Driver I/O Control* menu item and see the drop menu shown below.



2.3.1 Drvtest – Open Connection

Selecting the *Open Connection* menu item will present the following dialog box.



Open Connection

AAL Type: AAL0 / AAL34 / [✓ AAL5] / Raw

Service Class: [✓ CBR] / UBR / VBR / ABR

Driver Conn. Mode: [✓ Normal] / TX Multiple / TX Forever

VPI: 0

VCI: 100

Cell Rate in cells/second: 10000

The above box is set to open an AAL5, CBR connection for VPI 0 and VCI 100. The *Driver Conn. Mode* is set to *Normal*. To open the connection click *OK*.

IMPORTANT:

The driver is a standard Windows driver (VXD for Win95, SYS for WinNT). We have exposed a lot of the low level hardware resources so that the user may gain access and control over the ABRSAR. In doing so there are combinations of things that applications can do that will create system problems. The *Driver Conn. Mode* is one such instant. **When multiple connections are open, all of those connections must be opened in the *Normal* mode. The only time you may select *TX Multiple* or *TX Forever* is when there is one and only one connection open**.

What are the differences between the modes?

***Normal* –** When a connection is opened in *Normal* mode, as the name would imply this is the mode that would normally be used to send and receive cells(data). It means that when an application makes a system call to the driver with a buffer of data to transmit the driver sends that buffer once and completes the operation, returning to the application. On read the driver will fill the buffer with data, if any is available and return the data and a count of the number of bytes read. In this mode due to the overhead in windows attaining full line rate (155Mbits/Sec) is not possible.

TX Multiple – This will change the behavior of the driver when it receives a system call to send a buffer of data. Instead of just sending the buffer of data once and returning it caches the data buffer and enqueues it to be sent, returning at this time to the application. Back at the driver level as soon as the data has been sent the driver re-enqueues the buffer to be sent again. This is repeated until the application level makes a system call to close the connection. **After the app has issued that first send data system call it or any other app must not make another call to send data.** The app may make a system call to query the connection status, which will return cell transmission statistics along with other information. The advantage here is that you do not have the overhead of a Windows system call for each buffer of data transmitted. This mode still does not get to full line rate (about 90% - on a fast machine).

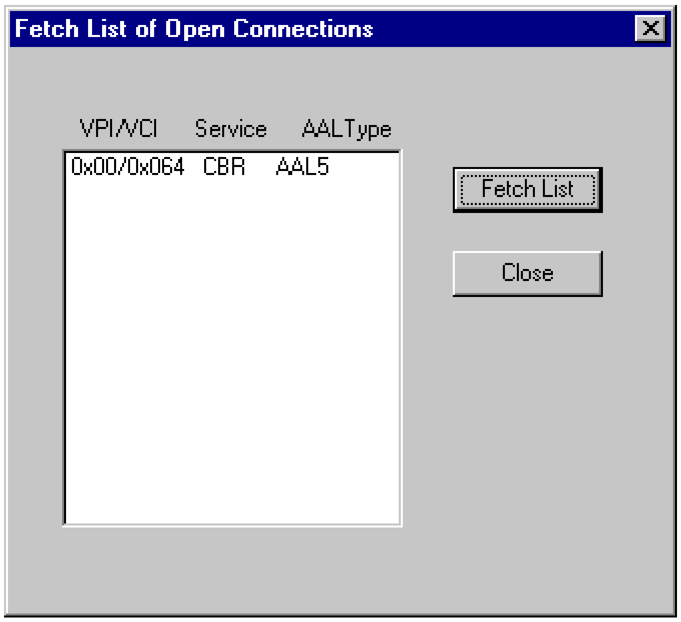
TX Forever – This mode is like the *TX Multiple* and has all the same limitations to the application. It differs in that it uses a feature of the IDT77252/222 that causes the ABRSAR to send the same buffer of data over and over again with no software intervention. The buffer length must be a multiple of 48 bytes. In that the transmission of data is totally controlled by the hardware, the transmission rate will reach line rate.

NOTE: Maximum buffer size in *TX Multiple* or *TX Forever* is 4072 bytes.

The peak cell rate(PCR) can range from 10 – 351415 cells per second. In *Drvtest* or the driver there are no guardrails to keep a user from over booking the bandwidth. That is to say you could open 4 connections with a PCR 0f 100000 cells per second each. If you did that, the results would be unpredictable. This behavior is due to the algorithm in the driver and not due to the parts allocation of bandwidth.

2.3.2 Drvtest – Fetch Connection

Selecting the *Fetch Connection* menu item will present the following dialog box.

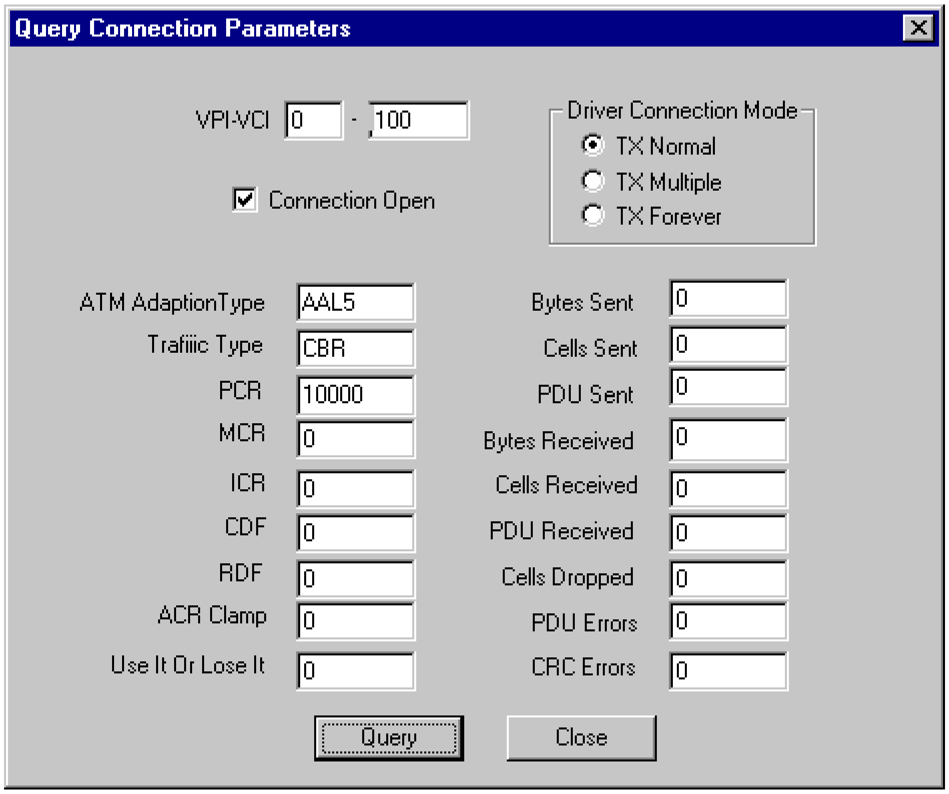


VPI/VCI Service AALType
0x00/0x064 CBR AAL5

If the user had opened a connection in the above section and clicked the *Fetch List* button here, then the above screen should show that a connection is open on VPI/VCI 0x00/0x64 with a service type of CBR and AAL type of AAL5. To dismiss the dialog box click *Close*.

2.3.3 Drvtest – Query Connection

Selecting the *Query Connection* menu item will present the following dialog box.


Query Connection Parameters

VPI-VCI 0 - 100

[✓] Connection Open

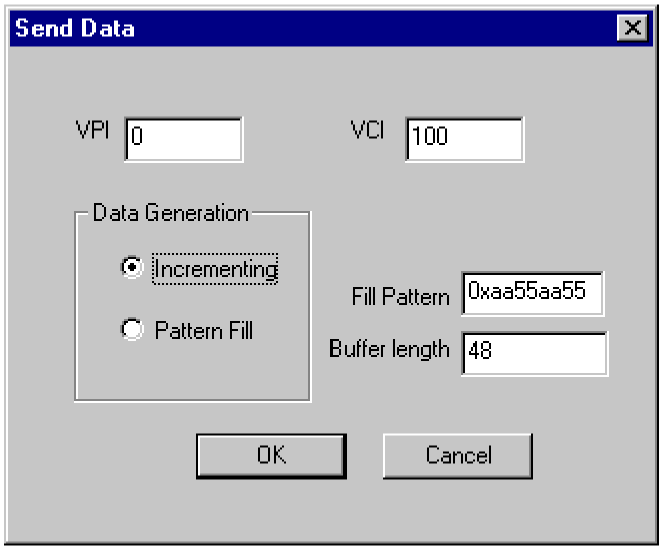
Driver Connection Mode
- [● TX Normal]
- TX Multiple
- TX Forever

ATM AdaptionType	AAL5	Bytes Sent	0
Trafiiic Type	CBR	Cells Sent	0
PCR	10000	PDU Sent	0
MCR	0	Bytes Received	0
ICR	0	Cells Received	0
CDF	0	PDU Received	0
RDF	0	Cells Dropped	0
ACR Clamp	0	PDU Errors	0
Use It Or Lose It	0	CRC Errors	0

When the user enters the connection VPI-VCI and clicks *Query*, the above data will be displayed. This may be used on a connection that is in any *Driver Conn. Mode*. Clicking *Query* repeatedly will update the information each time. The only fields the user should edit are the VPI-VCI. Click *Close* to dismiss the dialog box.

2.3.4 Drvtest – Send Data

Selecting the *Send Data* menu item will present the following dialog box.


Send Data

VPI 0

VCI 100

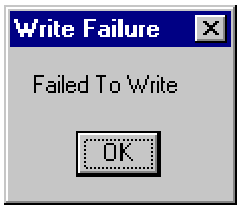
Data Generation

(● Incrementing) ○ Pattern Fill

Fill Pattern 0xaa55aa55

Buffer length 48

Before invoking this function the user **must** have opened a connection on the same VPI-VCI as is selected here. Sending data on a closed connection will result in an error return to the system call to send data. The following message box will be displayed.

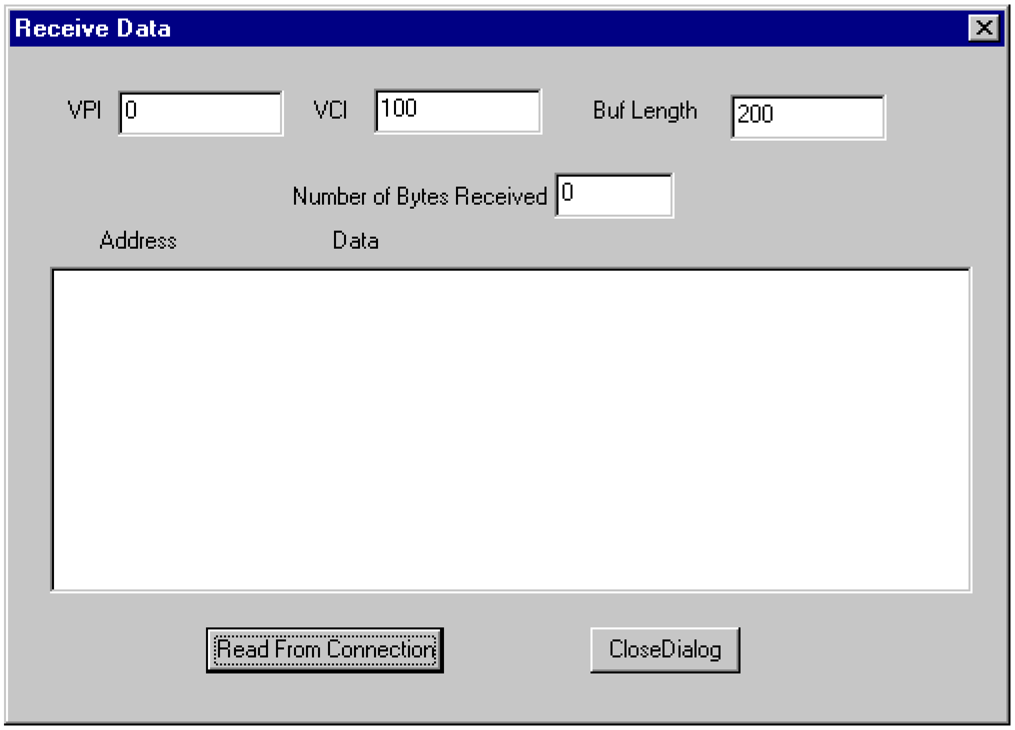

Write Failure

Failed To Write

The user may choose either an incrementing data pattern or specify a 4-byte data pattern. The buffer length may be from 1 – 32k. To send the buffer click *OK*. This will also close the dialog box. If the transmit is looped back to the receive then the user may select *Receive Data* and read the data just sent (see next section).

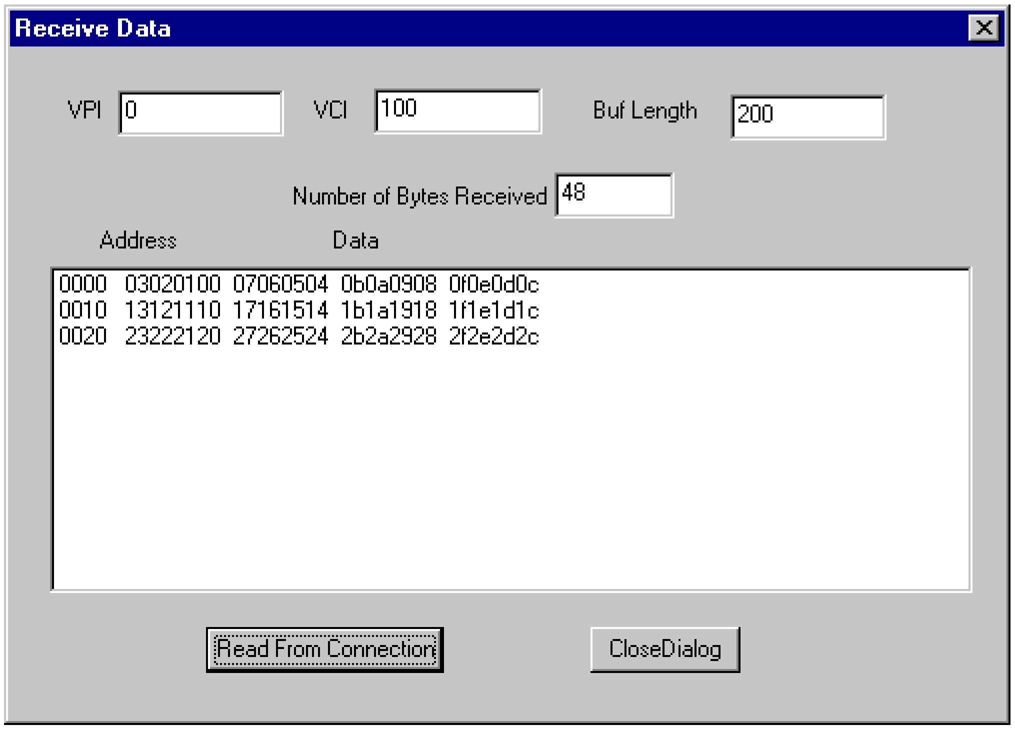
2.3.5 Drvtest – Receive Data

Selecting the *Receive Data* menu item will present the following dialog box.



VPI 0
VCI 100
Buf Length 200
Number of Bytes Received 0
Address Data

The user should enter the VPI-VCI that is to be read from. Throughout *Drvtest* the VPI-VCI defaults to VPI 0, and VCI 100. There is nothing special about this value, it was just convenient. Enter the amount of bytes to read in the *Buffer Length* field. When you open a connection the driver allocates resources to buffer incoming data on that connection. As data is received it is buffered and if no application reads this data and new data continues to arrive it will eventually wrap and overwrite the oldest data received. When the application makes a system call to receive data on a connection, the driver will return the number of bytes in the buffer for that connection up to the number of bytes requested. In other words, if there is more data in the driver's buffer than requested the driver returns the number of byte requested. If there is less data in the driver's buffer than requested, then the driver returns the number of bytes in the buffer. If the user had send 48 bytes of incrementing data in the previous section, with the settings in the *Receive Data* dialog box above, and clicks The *Read From Connections* button the dialog box would look as shown below.



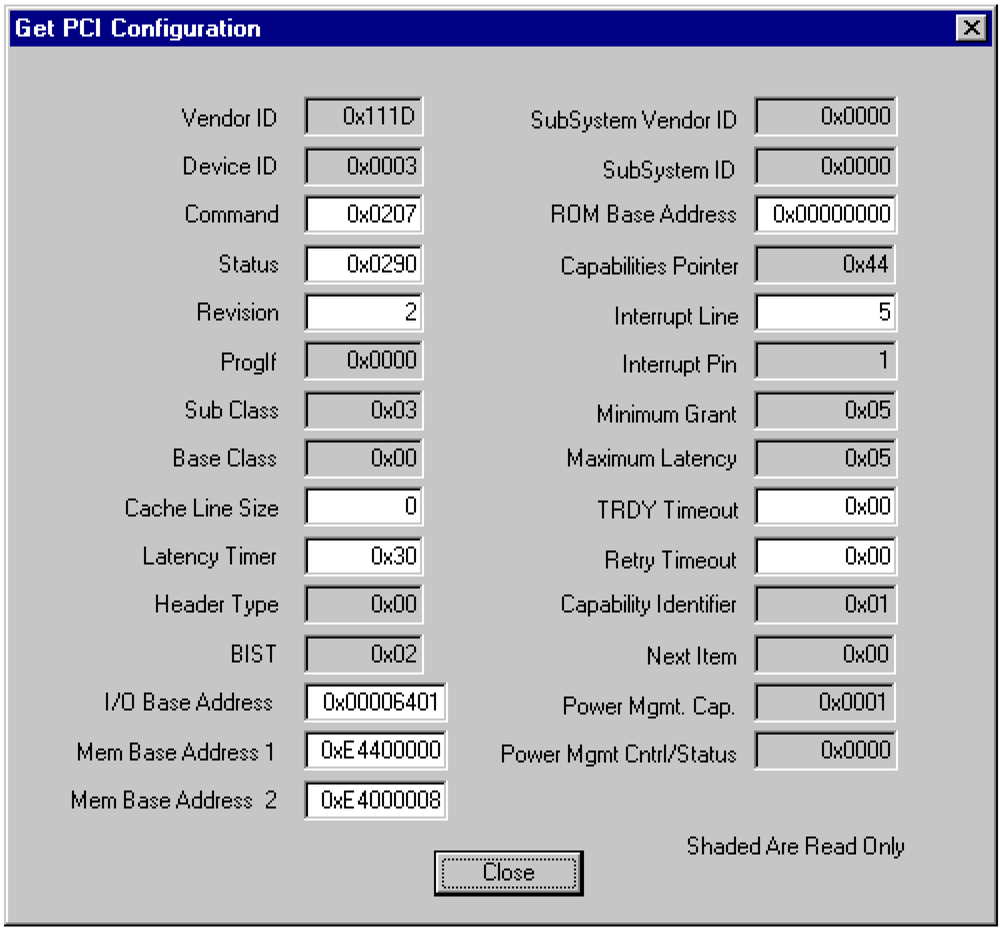
VPI: 0
VCI: 100
Buf Length: 200
Number of Bytes Received: 48

```
0000  03020100 07060504 0b0a0908 0f0e0d0c
0010  13121110 17161514 1b1a1918 1f1e1d1c
0020  23222120 27262524 2b2a2928 2f2e2d2c
```

The 48 bytes of data that was sent earlier is returned by the driver and displayed in the data window. Even though we asked for 200 bytes, there were only 48 to return. The incrementing data pattern shows we are dealing with a little endian machine. Clicking the *Read From Connection* button again will try to read another 200 bytes. Since we have sent no more data, the screen will not change. If we had asked for less bytes than was in the driver's receive buffer, then repeatedly clicking the *Read From Connection* button would continue to extract data from the driver's buffer until it was empty.

2.3.6 Drvtest – Get PCI Configuration

Selecting the *Get PCI Configuration* menu item will present the following dialog box.



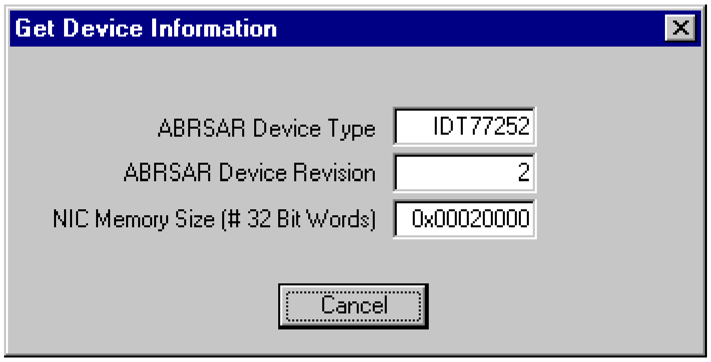
Vendor ID	0x111D	SubSystem Vendor ID	0x0000
Device ID	0x0003	SubSystem ID	0x0000
Command	0x0207	ROM Base Address	0x00000000
Status	0x0290	Capabilities Pointer	0x44
Revision	2	Interrupt Line	5
ProgIf	0x0000	Interrupt Pin	1
Sub Class	0x03	Minimum Grant	0x05
Base Class	0x00	Maximum Latency	0x05
Cache Line Size	0	TRDY Timeout	0x00
Latency Timer	0x30	Retry Timeout	0x00
Header Type	0x00	Capability Identifier	0x01
BIST	0x02	Next Item	0x00
I/O Base Address	0x00006401	Power Mgmt. Cap.	0x0001
Mem Base Address 1	0xE4400000	Power Mgmt Cntrl/Status	0x0000
Mem Base Address 2	0xE4000008		

Shaded Are Read Only

This same function is available in Abrsar_reg.exe. Abrsar_reg allows you to read and write, while Get PCI Configuration will only let you read.

2.3.7 Drvtest – Get Device Information

Selecting the *Get Device Information* menu item will present the following dialog box.



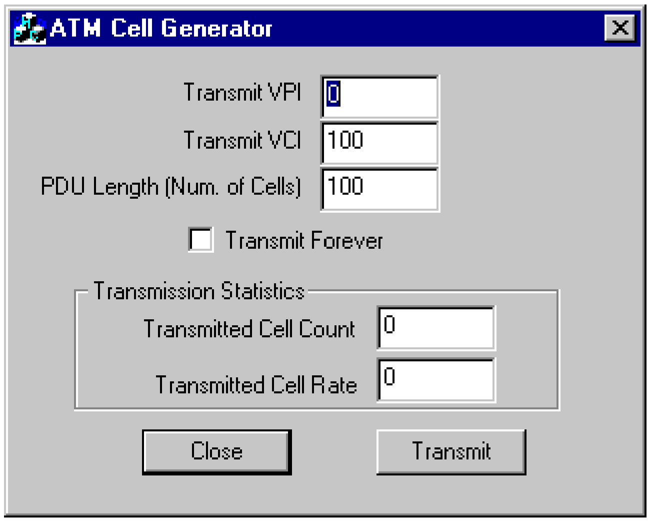
Get Device Information

ABRSAR Device Type	IDT77252
ABRSAR Device Revision	2
NIC Memory Size (# 32 Bit Words)	0x00020000

This returns information about the NicStAR card. There are two possible device types: IDT77252 and IDT77222. Revision will be one(1) or two(2). Memory size depends on how much memory is on the card. Typically it will be 0x00020000 if the device type is 77252 and 0x00004000 if the device type is 77222.

2.4 Cellgen.exe – Generate Cell Traffic

Cellgen.exe will send cell traffic over an open connection. To launch Cellgen.exe go to the \sarwin directory and double click the Cellgen.exe icon. The dialog box shown below will be displayed.


ATM Cell Generator

Transmit VPI: 0
Transmit VCI: 100
PDU Length (Num. of Cells): 100
Transmit Forever: ☐

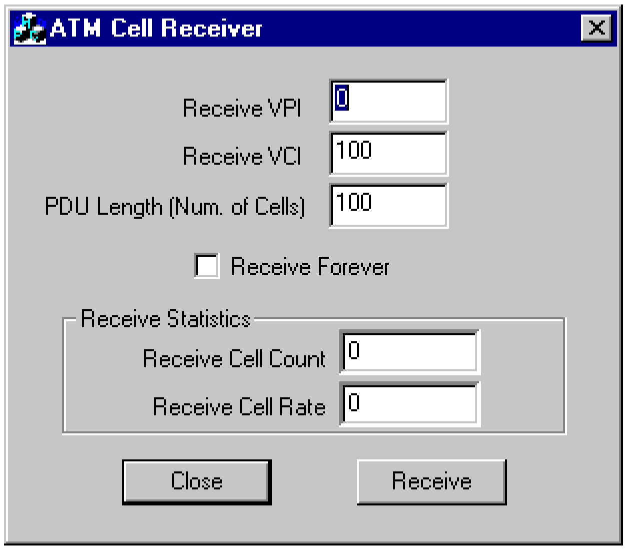
Transmission Statistics
Transmitted Cell Count: 0
Transmitted Cell Rate: 0

The user should enter the VPI-VCI of an open connection (again the default is 0-100). The user may specify the PDU length in cells (48 bytes) by editing the *PDU Length (Num. Of Cells)* field. In the above example we have it set to 100 cells (4800 bytes). This field may range from one (1) cell to 680 cells. If the user clicks *Transmit* the *Transmitted Cell Count* should jump to 100. Click it again and it should be 200. Click the *Transmit Forever* check box. Now click the *Transmit* button. The cell count should be incrementing. If the connection is a CBR connection at 10000 cells per second (PCR = 10000), the cell rate should be showing approximately 10000. Due to the asynchronous nature of cellgen's timer this value will typically range from 9900 to 10100. **NOTE:** do not close this application with the *Transmit Forever* check box checked. Clear the check box and then close cellgen.exe.

As an experiment: set *Transmit Forever* and then click *Transmit*. Now bring up Drvtest.exe and do *Query Connection* on the VPI-VCI that you are transmitting on. Click the query button repeatedly and you will see the transmit and receive statistics incrementing. You can bring up multiple instances of Cellgen.exe to transmit on more than one connection (Don't get carried away).

2.5 Cellrecv.exe - Receive Cell Traffic

Cellrecv.exe will receive cell traffic over an open connection. To launch Cellrecv.exe go to the \sarwin directory and double click the Cellrecv.exe icon. The dialog box shown below will be displayed.

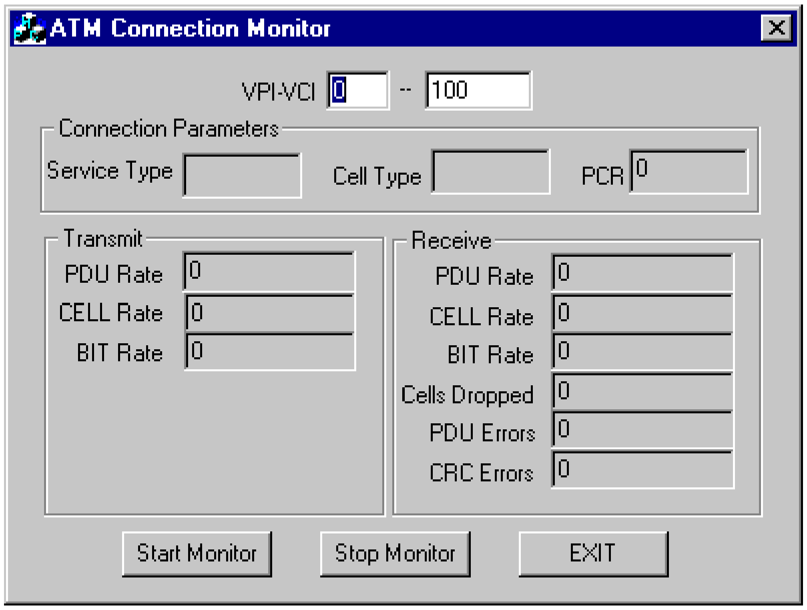


Receive VPI: 0
Receive VCI: 100
PDU Length (Num. of Cells): 100
Receive Cell Count: 0
Receive Cell Rate: 0

The user should enter the VPI-VCI of an open connection (again the default is 0-100). The user may specify the PDU length in cells (48 bytes) by editing the *PDU Length (Num. Of Cells)* field. In the above example we have it set to 100 cells (4800 bytes). This field may range from one (1) cell to 680 cells. Start a transmit session with Cellgen.exe. If the user clicks *Receive* the *Received Cell Count* should jump to 100. Click it again and it should be 200. Click the *Receive Forever* check box. Now click the *Receive* button. The cell count should be incrementing. If the connection is a CBR connection at 10000 cells per second (PCR = 10000), the cell rate should be showing approximately 10000. Due to the asynchronous nature of cellrecv's timer this value will typically range from 9900 to 10100. **NOTE:** do not close this application with the *Receive Forever* check box checked. Clear the check box and then close cellrecv.exe.

2.6 ATMMON.exe ATM Connection Monitor

ATMMON.exe is an application that can monitor the traffic on a user specified connection. To launch ATMMON.exe go to the \sarwin directory and double click the ATMMON.exe icon. The dialog box below will be displayed.



ATM Connection Monitor

VPI-VCI 0 -- 100

Connection Parameters

Service Type:
Cell Type:
PCR: 0

Transmit	
PDU Rate	0
CELL Rate	0
BIT Rate	0

Receive	
PDU Rate	0
CELL Rate	0
BIT Rate	0
Cells Dropped	0
PDU Errors	0
CRC Errors	0

The user should start *Cellgen.exe* and start a transfer forever session on VPI-VCI 0-100. Click *Start Monitor*. The transmit and receive numbers should reflect the current rates on this connection. If this connection is opened as an AAL5 connection, then you may see that the rate shown by the monitor is slightly higher than the rate shown in *Cellgen's* cell rate box. Cellgen is calculating the rate based on the number of payload cells, which is 100 for each PDU. At the driver level because it is AAL5 the actual number of cells in the PDU will be 101. The last cell has no data byte in it, just the 8 byte overhead for AAL5.

APPENDIX A   IDT77252ioctl.h

```
// idt77252ioctl.h
//
// Define control codes for idt77252 driver
//

#ifndef __idt77252ioctl__h_
#define __idt77252ioctl__h_
#ifndef BASE_TYPES_INC
#include "base_types.h"
#endif


#define idt77252_IOCTL_WRTREG CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_RDREG CTL_CODE(FILE_DEVICE_UNKNOWN, 0x801, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_PARAM CTL_CODE(FILE_DEVICE_UNKNOWN, 0x802, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_WRTUTIL CTL_CODE(FILE_DEVICE_UNKNOWN, 0x803, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_RDUTIL CTL_CODE(FILE_DEVICE_UNKNOWN, 0x804, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_OPEN_CONN CTL_CODE(FILE_DEVICE_UNKNOWN, 0x805, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_CLOSE_CONN CTL_CODE(FILE_DEVICE_UNKNOWN, 0x806, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_FETCH_CONN CTL_CODE(FILE_DEVICE_UNKNOWN, 0x807, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_COUNT_CONN CTL_CODE(FILE_DEVICE_UNKNOWN, 0x808, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_QUERY_CONN CTL_CODE(FILE_DEVICE_UNKNOWN, 0x3, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_SEND CTL_CODE(FILE_DEVICE_UNKNOWN, 0x80A, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_RECEIVE CTL_CODE(FILE_DEVICE_UNKNOWN, 0x80B, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_SET_CONN CTL_CODE(FILE_DEVICE_UNKNOWN, 0x80C, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_GET_PCI CTL_CODE(FILE_DEVICE_UNKNOWN, 0x80D, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_SET_PCI CTL_CODE(FILE_DEVICE_UNKNOWN, 0x80E, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define idt77252_IOCTL_GET_INFO CTL_CODE(FILE_DEVICE_UNKNOWN, 0x80F, METHOD_BUFFERED, FILE_ANY_ACCESS)
//
//      Write and read reg (WRTREG and RDREG) Parameters sent:
//              buf + 0         register number
//              buf + 4         data sent/received
//
//      Write and read utility bus has the following buffer meanings:
//              buf + 0         Bus device number -- 0x0 to 0x3
//              buf + 4         Utility bus address 0 - 255
//              buf + 8         Data sent or received to/from utility bus.
//
//
// when calling DeviceIoControl with an IO control code
// of idt77252_IOCTL_PARM
// the IO input buffer is interpreted as follows:
//              buf + 0         Opereration code
//              buf + 4         Sub code field
//              buf + 8         Size of Data Array
//              buf + c         Start of data array
//
//      Operation code(buf+0) for accessing 77222/252 nicstar sram
#define PARAM_READ_SRAM                 0x01    // sub code field has SRAM address - data returned data array
#define PARAM_WRITE_SRAM        0x02    // sub code field has SRAM address
                                                              // - data array has data

#define TX_NORMAL       0       // application sends data each time
#define TX_MULTIPLE     1       // Application sends once - driver send buffer over and over
#define TX_FOREVER      2       // Application sends once - driver sets hardware tx_forever bit

#ifdef __cplusplus
extern "C" {
#endif

#pragma pack(1)

typedef struct _t_IoctlBuf IoctlBuf, *PIoctlBuf;
struct _t_IoctlBuf
        {
        unsigned long  ioctlCode;
        unsigned long  ioctlSubCode;
        unsigned long  ioctlDataSize;
        unsigned long  ioctlData;
        };

typedef struct t_open_conn_param t_open_conn_param;
struct t_open_conn_param {
        // values -> driver
        unsigned short vpi, vci;
        unsigned char service_class;
        unsigned char aal_type;
        unsigned long pcr, mcr, icr, scr, acr, frtt;
        unsigned short nrm, crm, mbs, bt;
        unsigned char rif, rdf, cdf, adtf;
        unsigned char acr_clmp, use_lose_it;
        unsigned long tbe, age_cnt;
        unsigned long max_idle_cnt,max_token_cnt,pcr_token;
        unsigned char locked;
        unsigned long owner;
        int                     mode;
};

typedef struct t_set_conn_param t_set_conn_param;
struct t_set_conn_param {
        // values -> driver
        unsigned short vpi, vci;
        int mode;
        DWORD count;
};

typedef struct t_txconn_rec {
        WORD vpi, vci;
        BYTE enable_abr, enable_vbr;
        // ABR Fields
        WORD init_er;
        WORD lmcr;
        WORD lacr;
        WORD air_table, rdf_table, cdf_table;
        WORD norm_age_count;
        BYTE use_lose_it, acr_clamp;
        WORD crm;
        // cbr
        DWORD rate;
        unsigned long max_idle_cnt,max_token_cnt,pcr_token;
} t_txconn_rec;

typedef struct t_close_conn_param t_close_conn_param;
struct t_close_conn_param {
        // values -> driver
        unsigned short vpi, vci;
};

typedef struct t_query_conn_param t_query_conn_param;
struct t_query_conn_param {
        // values -> driver
        unsigned short vpi, vci;
        // values <- driver
        unsigned char service_class;
        unsigned char aal_type;
        unsigned long pcr, mcr, icr, scr, acr, frtt;
        unsigned short nrm, crm, mbs, bt;
        unsigned char rif, rdf, cdf, adtf;
        unsigned char acr_clamp, use_lose_it;
        unsigned long tbe, age_cnt;
        unsigned long max_idle_cnt,max_token_cnt,pcr_token;
        unsigned char locked;
        unsigned long owner;
        int                     mode;
        // some stuff actually related to status
        unsigned long bytes_sent, cells_sent, pdu_sent;
        unsigned long bytes_received, cells_received, pdu_received;
        unsigned long cells_dropped, pdu_errors, crc_errors;
        // parameters for driver bookkeeping
        t_txconn_rec drvr_param;
};

typedef struct t_count_conn_param t_count_conn_param;
struct t_count_conn_param {
        // values <- driver
        unsigned long count;
};

typedef struct t_vpi_list {
                // an array of vpi vci pairs of size count
                unsigned short vpi, vci;
                unsigned char service_class;
                unsigned char aal_type;
} VPI_LIST;

typedef struct t_fetch_conn_param t_fetch_conn_param;
struct t_fetch_conn_param {
        // values -> driver
        unsigned long count;
        // values <- driver
        VPI_LIST       list[1];
};

typedef struct t_send_conn_param t_send_conn_param;
struct t_send_conn_param {
        // values -> driver
        unsigned short vpi, vci;
        unsigned long length;
        unsigned char buffer[1];      // an array of data elements of size length
};

typedef struct t_receive_conn_param t_receive_conn_param;
struct t_receive_conn_param {
        // values -> driver
        unsigned short vpi, vci;
        // values <-> driver
        unsigned long length;
        // values <- driver
        unsigned char buffer[1];      // an array of data elements of size length
};

typedef struct t_pci_config t_pci_config;
struct  t_pci_config {
    unsigned short  VendorID;
    unsigned short  DeviceID;
    unsigned short  Command;
    unsigned short  Status;
    unsigned char   RevisionID;
    unsigned char   ProgIf;
    unsigned char   SubClass;
    unsigned char   BaseClass;
    unsigned char   CacheLineSize;
    unsigned char   LatencyTimer;
    unsigned char   HeaderType;
    unsigned char   BIST;
    unsigned long   BaseAddresses[6];
    unsigned long      CardBusCISPtr;
    unsigned short     SubsystemVendorID;
    unsigned short     SubsystemID;
    unsigned long      ROMBaseAddress;
    unsigned char      Cap_Ptr;
    unsigned char      Reserved1;
    unsigned short     Reserved2A;
    unsigned long   Reserved3;
    unsigned char   InterruptLine;
    unsigned char   InterruptPin;
    unsigned char   MinimumGrant;
    unsigned char   MaximumLatency;
    unsigned char      TrdyTimeout;
```

```
        unsigned char  RetryTimeout;
        unsigned short Reserved4;
        unsigned char  Cap_ID;
        unsigned char  Next_Item;
        unsigned short PMC;
        unsigned short PMCSR;
        unsigned short Reserved5;
};

typedef struct t_get_info t_get_info;
struct  t_get_info {
        unsigned long  mem_size;              // in 32 bit words
        unsigned char  device_type;   // 0 = 77252, 1 = 77222
        unsigned char  revision;
        unsigned char  param1;
        unsigned char  param2;
        unsigned long  param3;
        unsigned long  param4;
};

#ifdef __cplusplus
};
#endif

#pragma pack()

#endif
```