Contemporary Control
Systems Inc.

Programmer Guide
COM20020/22 Null Stack Driver

for

AI-SRVR
AI-USB
PCI20/22
USB22

DLL Driver for Windows 2K/XP/Vista

version 1.2 4/14/04

Table of Contents

Overview

Contemporary Controls Null Stack Driver

Null Stack Driver Overview

Under Windows 2000/XP/Vista most networking drivers utilize the NDIS model or interface. This arrangement is acceptable for applications that require standard protocols such as TCP/IP, IPX/SPX or NetBEUI. In situations where custom protocols are used, in the NDIS model, a custom protocol driver must be written. The NDIS model also separates the user application from the hardware by several software layers. This can impact the networking performance of an application.

In situations where custom protocols are used, a null stack driver is recommended. A null stack driver simply allows the sending and receiving of raw packets over a network interface device (AI-USB, PCI20/22, AI-SRVR or USB22).

The Contemporary Controls Null Stack Driver provides several functions that allow the sending and receiving of custom ARCNET packets. Also included are functions that provide network status.

The ARCX.DLL and ARCX.LIB files are to be used with user applications written for the AI-USB, PCI20/22, AI-SRVR or USB22.

Provided Files

Along with the driver, a sample application is also provided in both source code and compiled forms. The table below lists the provided files.

File	Description
ARCX.DLL	DLL containing functions for use with the AI-USB, PCI20/22 and USB22
ARCX.LIB	LIB file for resolving links within applications
ARCX.H	Include file that defines how to interface to the DLL
TALK.EXE	Win32 Sample driver application that allows ASCII messages to be sent/received between two or more computers.
TALK.C	C Source code file for Talk.exe sample application
TALK.RC	Resource file for Talk.exe
TALK.DSP TALK.DSW TALK.OPT TALK.NCB	Project files for talk application
RESOURCE.H	Resource include file for Talk.exe

History

In the table below, the history of the driver is listed.

Item	Version	Date	Comments
ARCX.DLL	2.1	12.03.03	First release
ARCX.LIB	2.1	12.03.03	First release

References

[1] COM20022 Data Sheet, Standard Microsystems Corporation 1998

[2] ANSI/ATA 878.1-1999 Local Area Network: Token Bus ARCNET Trade Association 1999

Driver Operation

The intent of the driver is to provide a means to allow Win32 applications, written under Windows 2000/XP/Vista, to send and receive ARCNET packets. There is a similar API available for communicating with ISA/PCI and PCMCIA based cards; however, all communications are handled via DeviceIoControl functions.

ARCX.DLL provides functions that are called directly by the user's application to initialize the AI-USB, PCI20/22 or USB22; check status; send and receive packets; and other tasks.

The descriptions below will describe the driver control codes in more detail. Each function returns a value (0) or a non-zero integer that indicates failure.

As seen in the sample application the driver can be used in two modes. The first mode, or non-event driven mode, allows the application to poll the driver for its status as to the completion of a transmission or to check for received packets. This mode can be demonstrated by commenting out the “EVENT_DRIVEN” compiler directive in the example code. The other mode allows the application to put a thread to sleep when waiting for a transmission to complete or while waiting for packets to be received. In the example program this mode is demonstrated when the compiler directive EVENT_DRIVEN is defined.

IMPORTANT NOTE: The file arcx.h contains several structures that are used to communicate with the driver. These structures require a structure member alignment of 1 byte. A Visual C++ project normally defaults to a structure member alignment of 8 bytes. In order to properly use these structures, either the project must be set to use a structure member alignment of 1 byte or a pragma must be used. See arcx.h for an example of how the pragma can be used.

ARCNET Considerations

Although the driver tries to abstract the network as much as possible, some understanding of ARCNET must be attained to effectively utilize the driver.

ARCNET networks have many built-in features that other networks, such as Ethernet, do not provide. When a node wishes to send a targeted packet to another node the transmitting node first sends a “Free Buffer Inquiry”. The receiving node will respond with its free buffer status. A NAK is used to indicate that the receiving node has no buffer space available. This feature of ARCNET is used to eliminate overruns on the receiving node. The transmitting node will continue trying to send its packet until it reaches the Excessive

NAK limit (see **Com20020Init()**). When the Excessive NAK limit is reached the driver will cease trying to transmit the packet and will communicate this to the application via the **Com20020Status()** function. The WAKE_ON_TX_COMPLETE event will also be set to alert the application. The sending of Free Buffer Inquiries and NAKs are handled by the COM20020.

When a packet has been properly received, the receiving node will send an acknowledgement to the transmitting node. This is performed by the COM20020. The driver will then provide this information to the application via the **Com20020Status()** function. The **bTransmissionComplete** flag is used to indicate, to the transmitting node, that the transmission is complete. The **bTransmissionAcknowledged** flag is used to indicate that the packet has also been received correctly or incorrectly. The **bExcessiveNAKs** flag is used to indicate that the transmission was cancelled due to Excessive NAKs.

If the driver tries to transmit to a non-existent node on a fully functioning network, the transmission will timeout and the driver will respond with a **bTransmissionComplete** set to true and the **bTransmissionAcknowledged** and **bExcessiveNAKs** flags set to false. A fully functioning network is a network with two or more online nodes. If the driver tries to transmit on a network where it is the only node online then the COM20020 cannot provide any timeout status. If the transmission does not complete within a “set period of time”, the **Com20020CancelTX()** function should be used to cancel the last transmission. Any following **Com20020Transmit()** functions will return with an error indicating the inability to transmit until the network contains two or more active nodes. The “set period of time” to wait until canceling a transmission is dependent upon maximum number of nodes in the network and the maximum number of bytes transmitted by each (see ANSI 878.1). A worst case number which can be used is 840ms, when using a standard timeout (see **Com20020Init()**). When using any other timeout, 1680ms can be used.

Hardware Considerations

Each device uses an SMSC COM20022 to interface to the ARCNET network. This chip is backward compatible with the SMSC COM20020. The COM20022 provides an ability to communicate over the ARCNET network at 10Mbps. For more information on the SMSC COM20022 see the SMSC website (www.smsc.com).

Driver Control Codes/Functions

Com20020Init (COM20020_CONFIG *cfg, UCHAR device, UCHAR hardwareType)

This function initializes the AI-USB, PCI20/22, USB22 or AI-SRVR. See notes below for special instructions on initializing a AI-SRVR.

COM20020_CONFIG parameters (not used for AI-SRVR):

- **uiCom20020BaseIOAddress:** This parameter is not used in the NT version of the driver.
- **byCom20020InterruptLevel:** This parameter is not used in the NT version of the driver.
- **byCom20020Timeout:** This is used to indicate the type of extended timeout desired.
  - **STANDARD_TIMEOUT** indicates a standard timeout (not an extended timeout).
  - **QUAD_TIMEOUT** indicates four times the normal timeout
  - **EIGHT_TIMEOUT** indicates eight times the normal timeout
  - **SIXTEEN_TIMEOUT** indicates sixteen times the normal timeout
- **byCom20020NodeID:** This is the node ID used by the driver.
- **bCom20020_128NAKs:** This is the number of NAKs before a transmitted message will stop being retried. A true value will stop the transmission after 128 NAKs. A false value will end the transmission after 4 NAKs. The premature ending of a transmission will be indicated in the **Com20020Status()** function.
- **bCom20020ReceiveAll:** A true value will allow all network traffic to be received by this ARCNET card.
- **byCom20020ClockPrescaler:** This controls the data rate of the Com20020 or Com20022 device. The current available selections are:

Selection	Data Rate
0	2.5 Mbps
1	1.25 Mbps
2	625 kbps
3	312.5 kbps
4	156.25 kbps
5	5.0 Mbps
7	7.25 Mbps
10	10.0 Mbps

- **bCom20020SlowArbitration:** This is curently not used and should be set to false.

- **bCom20020ReceiveBroadcasts:** This controls the card's ability to receive broadcast messages. A true allows the card to receive broadcasts. A false causes the card not to receive broadcast messages.

device is the instance of the AI-USB, PCI20/22 or USB22 in the system to which the application will communicate. For example, if there are four AI-USB devices connected to the computer the first detected AI-USB will have a **deviceNumber** of 0, the second will be 1, the third 2 and the fourth 3. **device** is not used when initializing a AI-SRVR.

hardwareType is '0' for AI-USB or USB22, '1' for PCI20/22, '2' for AI-SRVR.

Return Value: (0) if the Com20020/20022 was properly initialized, or non-zero otherwise.

NOTE: When initializing a AI-SRVR, a pointer to a AISRVR_CONFIG structure must be passed instead of a COM20020 structure. Since you are connecting to an already-initialized AI-SRVR device, the regular initialization parameters are not used. Fill out the **hostname** and **port** values for a AISRVR_CONFIG structure, and call the initialization function as follows:

```
result = Com20020Init( (COM20020_CONFIG *)&aisrvr_cfg ,
device_num , hardware_type );
```

To initialize a AI-SRVR, you may either use auto-detect, or you may supply a hostname and optional port number. Auto-detect only works on when the AI-SRVR is attached to the same subnet as the client computer. If the **hostname** value of the AISRVR_CONFIG structure is '0', auto detect will be used. If the **port** number is '0', the default port 5001 will be used. If you are able to connect to the AI-SRVR, Com20020Init() returns '0', and the rest of the AISRVR_CONFIG structure is filled out with data retrieved from the AI-SRVR.

Com20020Transmit(COM20020_TRANSMIT_BUFFER *txbuf)

This function transmits a packet out onto the network. Upon a (0) return of this function, the Com20020Status() function can be used to check completion of the transmission.

COM20020_TRANSMIT_BUFFER parameters:

- **byDestinationNodeID:** This is the destination node ID for this message. To send a broadcast message use a destination node ID of zero.
- **uiNumberOfBytes:** This is the number of bytes in the data buffer. This number must be between 1 and 508. Also it cannot equal 254, 255 or 256 as these are exception packet sizes and cannot be transmitted by the Com20020/20022.
- **byDataBuffer[508]:** This is the data to be transmitted in the packet. If system codes are used in the message, the first byte (byDataBuffer[0]) will contain the message's system code.

Return Value: (0) if the packet was transmitted.

Com20020Exit(void)

This function will stop all activities of the driver. This function should be called before the end of the application.

Return Value: (0)

Com20020Receive(COM20020_RECEIVE_BUFFER *rxbuf)

This function will either provide one received buffer or indicate that there are no receive packets available. The current number of received packets are also indicated. After the packet has been read by this function it is removed from the driver's buffers.

COM20020_RECEIVE_BUFFER parameters:

- **bySourceNodeID:** This is the source ID of the received packet.
- **byDestinationNodeID:** This is the destination node ID of the received packet.
- **uiNumberOfBytes:** This is the number of data bytes in the received packet.
- **byDataBuffer[508]:** This is the received packet data bytes.
- **dwNumberOfFilledBuffers:** This is the number of received packets currently stored by the driver. This number includes the packet provided in this structure.

Return Value: (0) if one packet has been provided in the output buffer; non-zero if no packets are available.

Com20020Status(COM20020_STATUS *status)

This function will provide the current status of the driver and COM20020/20022 device.

COM20020_STATUS parameters:

- **bReceiveActivity:** This is an indication of traffic on the network. A true indicates there is activity. A false indicates that either there is no activity or the Com20020/20022 is not properly initialized.

- **bPowerOnReset:** This is an indication of whether the device received a power on reset. A true indicates that a power on reset has occurred. A false indicates that no power on reset has occurred. This flag will remain true, after power up until a CLEARPOR command has been sent to the device through the **Com20020Register()** function.

- **bRecon:** This flag indicates the presence of a RECON on the network. This flag is automatically reset by the **Com20020Status()** command.

- **bToken:** This flag indicates a token has been seen on the network sent by a node other than this node.

- **byReceivedMessages:** This is the number of received packets currently stored by the driver.

- **bTransmissionComplete:** This flag is an indication that the previously requested transmission has been completed. In order to assess the proper completion of the message the flag **bTransmissionAcknowledged** must also be inspected.

- **bTransmissionAcknowledged:** This flag indicates the message was acknowledged by the receiving node.

- **bExcessiveNAKs:** This flag indicates that the last transmission was cancelled due to an excessive number of NAKs received from the receiving node during the transmission. This can be due to a misaddressed packet or a node that has turned off its ability to receive packets.

- **dwReserved:** This is currently not used.

Return Value: (0) if this function was successful, non-zero otherwise.

Com20020Register(COM20020_REGISTER *reg)

This function allows the modification of one of the Com20020/20022 registers or can be used to read one Com20020/20022 register.

COM20020_REGISTER parameters:

- **bWrite:** This is used to indicate whether a write or read is being requested. A true indicates a write is being requested and a false indicates a read.
- **byRegister:** This is the register addressed by this function. This can range from 0 to 7. Please refer to the Standard Microsystems Com20020 or Com20022 datasheet for more information.
- **byValue**: If this is a write command then this is the value to be written into the targeted register. If this function is being used to read from the device then this value reflects the value of the register upon return.

Return Value: (0) if the function was successful, non-zero otherwise.

Com20020CancelTX(void)

This function cancels the previously requested transmission. This is useful when a transmission has been requested on a network with only one active node.

Input Buffer: None

Return Value: (0) if the function was successful, non-zero otherwise.

Com20020WakeOnReceive(HANDLE receiveEvent)

This function indicates to the driver that the application wishes to have its event set or "woken" upon receipt of one or more packets. After the event has been set, the null stack driver (version 1.4a) will reset the event and the application does not have to reset the event as before.

Return Value: (0).

Com20020ResetWakeOnReceive(void)

This function indicates to the driver that the application no longer wishes to have its event set or "woken" upon receipt of one or more packets. This function must be called before the application is exited.

Return Value: (0).

Com20020WakeOnTXComplete(HANDLE transmitEvent)

This function indicates to the driver that the application wishes to have its event set or "woken" upon the completion of a transmit operation. After the event has been set the application must then reset its event in order to receive the next event.

Return Value: (0).

Com20020ResetWakeOnTXComplete(void)

This function indicates to the driver that the application no longer wishes to have its event set or “woken” upon the completion of a transmit operation. This function must be called before the application is exited.

Return Value: (0).

Com20020WakeOnRecon(HANDLE reconEvent)

This function indicates to the driver that the application wishes to have its event set or “woken” upon the occurrence of a reconfiguration operation. After the event has been set the application must then reset its event in order to receive the next event.

Return Value: (0).

Com20020ResetWakeOnRecon(void)

This function indicates to the driver that the application no longer wishes to have its event set or “woken” upon the occurrence of a reconfiguration operation. This function must be called before the application is exited.

Return Value: (0).

Com20020GetUSB22OverflowTotal(long *packets , long *data)

This function returns the number of packets and the total number of bytes lost by the USB22 due to overflow. If the PC does not retrieve data from the USB22 fast enough, the USB22 internal buffer can overflow. When this happens, received packets are discarded and a count is maintained of the number of discarded packets and of the total number of bytes lost.

Return Value: (0) if successful, or non-zero if failed.

Com20020GetUSB22FirmwareRevision(short *rev)

This function returns the firmware revision number of the USB22. The major revision number is contained in the upper 8 bits of the returned value, the minor number in the lower 8 bits.

Return Value: (0) if successful, or non-zero if failed.

Com20020UsbVersion(short *version)

This function returns a number in 'version' which indicates whether the USB22 is attached to a USB1.1 or USB2.0 port. The value of 'version' will be 0x0011 for USB1.1, and 0x0022 for USB2.0.

Return Value: (0) if successful, or non-zero if failed.