Australian Monitor
AMD100/AMD200

AMX NetLinx Module Interface Specification

TABLE OF CONTENTS

LIST OF TABLES

Introduction

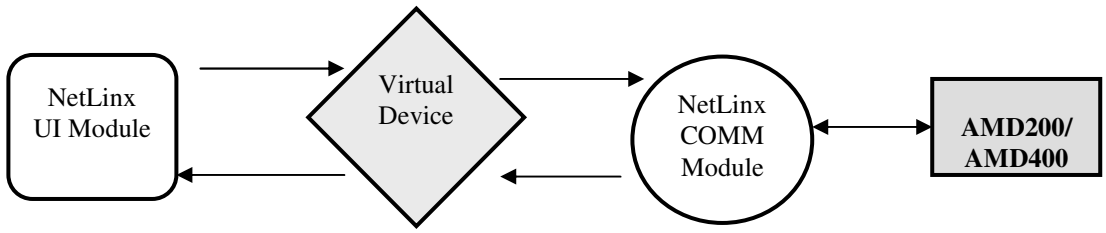
This is a reference manual to describe the interface provided between an AMX NetLinx system and a Australian Monitor AMD100/AMD200. The AMD100/AMD200 supports an RS-485 serial protocol. The required communication settings are a baud rate of 19200, 8 data bits, 1 stop bit, no parity.

Overview

The COMM module translates between the standard interface described below and the AMD100/AMD200 serial protocol. It parses the buffer for responses from the audio processor, sends strings to control the audio processor, and receives commands from the UI module or telnet sessions.

A User Interface (UI) module is also provided. This module uses the standard interface described below and parses the string responses for feedback.

The following diagram gives a graphical view of the interface between the interface code and the NetLinx module.



Implementation

To interface to the AMX AustralianMonitor_AMDX00_Comm.axs module, the programmer must perform the following steps:

1. Define the device ID for the AMD100/AMD200 that will be controlled.
2. Define the virtual device ID that the AMD100/AMD200 COMM module will use to communicate with the main program and User Interface. NetLinx virtual devices start with device number 33001.
3. If a touch panel interface is desired, a touch panel file (AustralianMonitor_AMDX00_Test.TP4) and module (AustralianMonitor_AMDX00_UI.axs) have been created for testing.
4. The NetLinx AMD100/AMD200 module must be included in the program with a DEFINE_MODULE command. This command starts execution of the module and passes in the following key information: the device ID of the audio processor to be controlled, and the virtual device ID for communicating to the main program.

An example of how to do this is shown below.

```
DEFINE_DEVICE
   dvDevice             = 5001:1:0      // The AMD100/AMD200 connected to the NetLinx on 1st RS-485
port
   dvTouchPanel         = 10001:1:0     // The touch panel used for output

      vdvDevice         = 33001:1:0     // The virtual device use for communication between the
                                        // Comm module interface and User_Interface (UI) module interface

DEFINE_VARIABLE
//Define arrays of button channels used on your own touch panel
integer nCHAN_BTN[]={1,2,3,4,5,6,7}
integer nTXT_BTN[]={1,2,3}

DEFINE_START      // Place define_module calls to the very end of the define_start section.
// Comm module
DEFINE_MODULE ‘AustralianMonitor_AMDX00_Comm’ Comm1(vdvDevice, dvDevice)
// User Interface module
DEFINE_MODULE ‘AustralianMonitor_AMDX00_UI' TouchPanelA(vdvDevice, dvTouchPanel,
nCHAN_BTN, nTXT_BTN)
```

Command Interface

The UI module controls the Australian Monitor AMD100/AMD200 via command events (NetLinx command *send_command*) sent to the COMM module. The commands supported by the COMM module are listed below.

Command	Description

BGMSOURCE=<value>	Selects the BGM Source.
	
	<value> : 0..4 = BGM Source Range
	
	BGMSOURCE=1 // Source 1
	BGMSOURCE=0 // Off

BGMSOURCE?	Query the value of the current BGM Source. BGMSOURCE?

FADER=<channel>
:<value>
	Ramp Up, down and stop or sets the specified fader level.
	
	<channel> : 1..12 = channel
	<value> : UP = start ramping up
	DOWN = start ramping down
	STOP = stop ramping
	0..255 = fader level range
	
	Channels:
	
	1 = Mic/Line 1
	2 = Mic/Line 2
	3 = Mic/Line 3
	4 = Mic/Line 4
	5 = Mic/Line 5
	6 = Mic/Line 6
	7 = Mic/Line 7
	8 = Mic/Line 8
	9 = Stereo BGM
	10 = Bass Filter Value
	11 = Treble Filter Value
	12 = Master Volume
	
	FADER=2:UP // Start ramping up
	FADER=1:128 // set to 128 (50%)

FADER?<channel>	Query the specified fader level. <channel> : 1..12 = channel FADER?1

MUTE=<channel>:<value>	Mute/Bypass specified fader channel. <channel> : 1..12 = channel <value> : 0 = mute off 1 = mute on (bypass) MUTE=3:1 MUTE=2:0
MUTE?<channel>	Query the specified channel for mute/bypass status. <channel> : 1..12 = channel MUTE?1
FADERMIN=<channel>:<value>	Set the minimum level for the specified fader channel <channel> : 1..12 = channel <value> : 0..224 = min range Note: Default value is 0. If min level not in valid range or greater than maximum level it will be set to 0 by the module. FADERMIN=1:111
FADERMIN?<channel>	Query the specified fader minimum level. <channel> : 1..12 = channel FADERMIN?1
FADERMAX=<channel>:<value>	Set the maximum level for the specified fader channel <channel> : 1..12 = channel <value> : 0..225 = max range Note: Default value is 201 (0db). If min level not in valid range or less than minimum level it will be set to 201 by the module. FADERMAX=1:201
FADERMAX?<channel>	Query the specified fader maximum level. <channel> : 1..12 = channel FADERMAX?1

GREENMODE=<value>	Set whether the AMD100/AMD200 is in Greenmode or not (Shutdown or Normal Operation)
	
	<value> : 0..1 = Green Mode Range
	
	GREENMODE=1 // Shutdown
	GREENMODE=0 // Normal (Running)

GREENMODE?	Query the value of Greenmode 0..3 = Green Mode Status Range 0 = Normal Mode 1 = Green Mode 2 = Emergency Mode 3 = Emergency Mode, with green mode active GREENMODE?
AMPSTATUS?	Query the status of the AMP 0..2 = AMP Status Range 0 = Amp is running 1 = AMP is in thermal shutdown 2 = AMP is in fault shutdown AMPSTATUS?
VERSION?	Query the communication module version. VERSION?

Table 1 – Send Command Definitions

String Feedback

The NetLinx COMM module provides feedback to the User Interface module for audio processor changes via string events. The strings supported are listed below.

String	Description

BGMSOURCE=<value>	Feedback of the current BGM Source.
	
	<value> : 0..4 = BGM Source Range
	
	BGMSOURCE=1 // Source 1
	BGMSOURCE=0 // Off

FADER=<channel> :<value>	Feedback of the specified fader level. <channel> : 1..12 = channel <value> : 0..255 = fader level range FADER=1:128 // 128 (50%)
MUTE=<channel>:<value>	Feedback of the status of Mute/Bypass for the specified fader channel. <channel> : 1..12 = channel <value> : 0 = mute off 1 = mute on (bypass) MUTE=3:1 MUTE=2:0

FADERMIN=<channel>:<value>	Feedback of the specified minimum fader channel. <channel> : 1..12 = channel <value> : 0..224 = fader min range FADERMIN=1:111
FADERMAX=<channel>:<value>	Feedback of the specified maximum fader channel. <channel> : 1..12 = channel <value> : 0..224 = fader max range FADERMAX=1:201

GREENMODE=<value>	Feedback of the status of whether the AMD100/AMD200 is in Greenmode or not (Shutdown or Normal Operation)
	
	<value> : 0..1 = Green Mode Range
	
	0 = Normal Mode
	1 = Green Mode
	2 = Emergency Mode
	3 = Emergency Mode, with green mode active
	
	GREENMODE=1 // Shutdown
	GREENMODE=0 // Normal (Running)

AMPSTATUS=<value>	Feedback of the current AMP Status.
	
	<value> : 0..2 = AMP Status Range
	
	AMPSTATUS=0 // AMP is running
	AMPSTATUS=1 // AMP is in thermal shutdown
	AMPSTATUS=2 // AMP is in fault shutdown

MODEL=<value>	Model of the AMD (100/200) <value>: current model MODEL=AMD100/AMD200

Table 2 - String Feedback Definitions

Device Notes

- For the initial setup and configuration of this device, use the AMD100/AMD200 Software program.

Adding Functions to Modules

Commands to the device

This module supplies a mechanism to allow additional device features to be added to software using the module. This is the PASSTHRU command, which allows protocol strings to be passed through the module. The device-specific protocol must be known in order to use this feature.

As an example, suppose that a module for a projector has not implemented the 'white balance adjustment' feature. The command that the projector protocol requires is 03H, 10H, 05H, 14H, followed by a checksum. The documentation for the PASSTHRU command specifies that the module will automatically generate the checksum. In this case, the following string should be sent from the UI code to implement 'white balance adjustment'.

```
send_string vdvDevice,"'PASSTHRU=',$03,$10,$05,$14"
```

The reason to use PASSTHRU instead of sending a protocol string directly to the device port is that the device may require command queuing, calculation of checksums, or other internal processing, which would not be done if the string was sent directly. Because of this, it is best to filter all communication TO the device through the module. (The module documentation will indicate any processing that will be automatically done to the PASSTHRU string like checksum calculation.)

Additional Feedback from the device

The module documentation indicates what feedback is provided. If additional feedback is required, a CREATE_BUFFER for the device must be implemented in the user code to process the strings from the device manually. Note that the module will still be processing the response strings independently and sending the interpreted feedback up to the user code.