NetLinx Module Interface Specification

for a

Toshiba
PxxLSA LCD Display

TABLE OF CONTENTS

Introduction 3
Overview 3
Implementation 3
Channels 5
Command Interface 6
String Feedback 8
Device Notes 9
Programming Notes 9
Adding Functions to Modules 9
  Commands to the device 9
  Additional Feedback from the device 10

Revision History

Date	Initials	Comments
10-05-09	DC	Initial release

Introduction

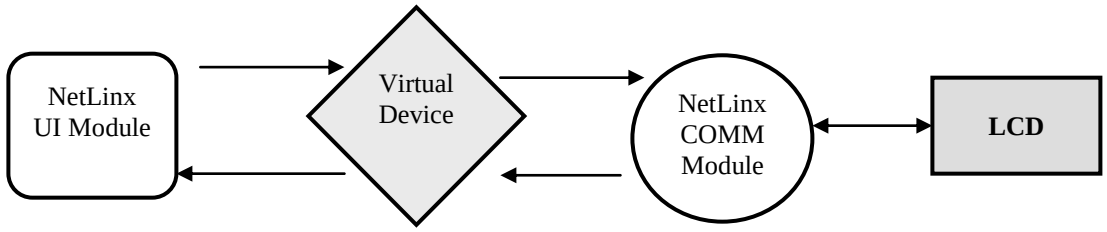
This is a reference manual to describe the interface provided between an AMX NetLinx system and a Toshiba LCD Display. The required communication settings are a baud rate of 9600, 8 data bits, 1 stop bit, no parity, and handshaking off.

Overview

The COMM module translates between the standard interface described below and the **LCD** serial protocol. It parses the buffer for responses from the LCD, sends strings to control the LCD, 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 LCD module, the programmer must perform the following steps:

1. Define the device ID for the LCD that will be controlled.
2. Define the virtual device ID that the LCD 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 LCD,REV0.tpd and module (Toshiba_LCD_UI.axs) have been created for testing.
4. The NetLinx LCD 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 LCD 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
   dvLCD          = 5001:1:0     // The LCD connected to the NetLinx on 1st RS-232 port
   dvLCD TPl      = 10001:1:0    // The touch panel used for output

   vdvLCD         = 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 nTP_BUTTONS[]={1,2,3,4,5,6,7}

DEFINE_START      // Place define_module calls to the very end of the define_start section.
// Main module
DEFINE_MODULE 'Toshiba_PxxLSA_Comm' mdlLCD_APP(dvLCD, vdvLCD)

// Touch panel module
DEFINE_MODULE 'Toshiba_ PxxLSA _UI' mdlLCD_UI(vdvLCD, dvTPArray, nTPButtons)
```

Upon initialization the AMX Comm module will communicate with the Toshiba LCD and information will be exchanged.

Channels

The channels supported by the COMM module are listed below. These channels are associated with the virtual device(s) and are independent of the channels associated with the touch panel device.

Note: An '*' indicates an extension to the standard API.

Channel	Description
251	This channel is used for feedback only. ON: Device is online. OFF: Device is offline.
255	This channel is used for feedback only. ON: POWER is ON. OFF: POWER is OFF

Command Interface

The UI module controls the LCD via command events (NetLinx command *send_command*) sent to the COMM module. The commands supported by the COMM module are listed below.

Command	Description
INPUT=<value>:<ID>	Selects the input source. <value> : 1-6 Inputs may vary with model. See user Manual for details. <ID>: ID of unit to adjust. Two characters- 0-9, A-F. * is used as a wildcard in Either or both positions. "01" is default. INPUT=3:A*
INPUT?<ID>	Request the current input setting. <ID>: ID of unit to adjust. Two characters- 0-9, A-F. * is used as a wildcard in Either or both positions. "01" is default. ***If a wildcard is used, the returned value will show the status of the last unit in the specified string.*** INPUT?31
PASSTHRU=<string>	Allows user the capability of sending commands directly to the LCD without processing by the NetLinx module. User must be aware of the protocol implemented by the unit to use this command. This gives the user access to features which may not be directly supported by the module. See "Adding Functions to Modules" section at the end of this document for more information. The COMM module **automatically adds the required checksum to the command string.** <string> : string to send to unit PASSTHRU=RVER
POWER=<state>:<ID>	Sets the power state. <state> : 0 = off 1 = on <ID>: ID of unit to adjust. Two characters- 0-9, A-F. * is used as a wildcard in Either or both positions. "01" is default. POWER=0:**

POWER?<ID>	Request the current power setting. <ID>: ID of unit to adjust. Two characters- 0-9, A-F. * is used as a wildcard in Either or both positions. "01" is default. ***If a wildcard is used, the returned value will show the status of the last unit in the specified string.*** POWER?AE
VERSION?	Request the version of the module. VERSION?

Table 1 – Send Command Definitions

String Feedback

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

String	Description
INPUT=<value>:<ID>	Reports the input source. <value> : 1-6 Inputs may vary with model. See user Manual for details. <ID>: ID of unit to adjust. Two characters- 0-9, A-F. * is used as a wildcard in Either or both positions. "01" is default. ***If a wildcard is used, the returned value will show the status of the last unit in the specified string.*** INPUT=3:25
POWER=<value>:<ID>	Reports the power status. <value> : 0 power is off 1 power is on <ID>: ID of unit to adjust. Two characters- 0-9, A-F. * is used as a wildcard in Either or both positions. "01" is default. ***If a wildcard is used, the returned value will show the status of the last unit in the specified string.***
VERSION=<version>	Reports the current version number of the NetLinx module. <value> : current version number in xx.yy format VERSION=1.06

Table 2 - String Feedback Definitions

Device Notes

Commands implemented by this interface include those that are commonly used and shared between various display devices. Several of the display commands have been omitted and may be executed using the PASSTHRU command. On occasion, these units will lock up. Depending on the firmware loaded, the lamp LED on the unit may blink, or the fan LED will blink. Once the unit is in this condition, it will not respond to commands until the power switch on the back of the unit has been turned off and then on again. This appears to happen when the power off command is issued.

Programming Notes

The COMM module imposes synchronous operation i.e. the COMM will not send a command to the LCD until it has gotten a response or a timeout from the previous command.

A queue is implemented so that the commands from can be stacked. Commands are pulled off the queue (1) when a reply is received or, (2) after the reply timer times out (200 milliseconds).

The wildcard (*) character works best with an action command such as the POWER= command. For example, the command POWER=1:1* will turn the power on for all units starting with the number 1.

The request commands will pass the request to the device(s) indicated by the ID string. Commands that use wildcard characters will not return a status for each device that matches the wildcard string. Instead, you will receive a return packet with the same ID that you sent (containing the wildcard characters) and the value of the status field will only contain the status of the last device in that particular match. In other words, the wildcards are very useful for turning the unit on, off, or changing all of the input settings but are worthless for retrieving status information. To retrieve status information, you should initiate a separate request for each device.

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_command 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. Be warned however, that you will not see much in the way of data returned. A successful command will return 06. An unsuccessful command will return 15. Information requests will return the standard 7-byte structure where bytes 4-6 contain the actual data requested.