JX Series
with IF-12 Control Module

TABLE OF CONTENTS

DRAWINGS

(Shipped configuration included. Other schematics available on request)

DRWG #	DESCRIPTION
4-058	Signal I/O Module Schematic
11-21-50	IF-12 LAN/RS232/GPIB Control Module Schematic
16-16-50	34 Pin Mesa II EIF Module
24-04-00	JX Expansion Chassis
24-20-00	JX Expansion Chassis High Power
24-20-30	JX Expansion Chassis High Power Wiring Diagram
24-21-00	JX/256 Mainframe Chassis
24-21-91	JX/256 Front and Rear Panel
24-21-30	JX/256 Power Supply Wiring Diagram
24-21-31	JX/256 Control Wiring Diagram

1.0 ADDENDUM

This page is intentionally left blank

2.0 GETTING STARTED

Unpack the unit and make sure it has arrived undamaged. Inspect for dents, bent handles, major scratches and missing or loose parts. Note that many of the items listed individually on the packing list are already installed within the chassis, rather than being packed separately.

Compare the Shipped Configuration List on the last page of the Quick Start Guide that shipped with the unit with the included packing slip to verify that all components and ordered parts have been received. If any purchased items are missing please contact your Sales Representative at 585-381-4740 or sales@cytec-ate.com. Utilize the Shipped Configuration List to identify which drawings and diagrams refer to the specific unit ordered.

Next, set up the chassis on either a bench or rack. The front handles allow the unit to be bolted to a standard 19 inch rack. No special setup tools are needed.

For AC powered units, a Power Cord should be included in the box. Plug one end into the chassis and the other into a three prong commercial AC outlet. The unit will operate from one of two AC voltage ranges: 100/140 or 200/260. There is a fuse holder built into the AC input that can be rotated to switch between 110 VAC and 220 VAC.

Install the appropriate remote control cable to the controlling computer: RS232, IEEE488 (GPIB) or Ethernet. Cytec provides a one to one RS232 D9 cable but does not provide Ethernet or GPIB cables with the unit.

Turn the unit ON via the toggle switch located at the AC input. The front panel Power LED should illuminate.

Study the sections of this manual which deal with your control interface (RS232 or Ethernet), as well as the controlling command syntax. A group of programming examples are included in appendices at the end of the manual and provide a good structure to work from. Drivers can also be downloaded from Cytec's web site at: http://www.cytec-ate.com/downloads

You should now be able to begin writing useful code. **Always write and debug code thoroughly before hooking up live signals to the matrix!** This equipment gives you full control over what is switched to where and will not prevent you from making potentially harmful connections. That is, nothing in the system prevents the switching of excessive power, which can damage or destroy the relay contacts or digital switches.

3.0 GENERAL

The JX Series may function as a high density 1xN multiplexer, a 2xM matrix, discrete switch points or some combinations of these. This series is typically used for data acquisition, component testing, bed of nails testers, and cable testers. A modular design concept is used, so that interchangeable switch modules may be assembled into various sized matrices, multiplexers, or individual switch point configurations. The JX Series switch modules (which are composed of 1, or 2 pole relays) are capable of switching the following signals depending on the modules chosen: Low level current to 1 picoamp, low level voltage with a DC offset less than 1 microvolt, high power to 2000 VA, and high current to 8 amps, and signals with a bandpass of up to 80 MHz.

3.1 CHASSIS DESCRIPTION

The JX Series includes the JX/256 mainframe and expansion chassis. A mainframe is a stand-alone unit which may be controlled either remotely ***(See Section 5.0)*** or via a keypad manual control. An expansion chassis must be wired to and controlled via a MESA control chassis.

The JX/256 mainframe is shown on **Drwg. #24-21-00**. The JX chassis holds an 17-slot card cage, a +5V power supply for driving the logic and a +12V power supply for driving the relays ***(Section 3.3)***. The front panel may hold the optional keypad manual control with LCD Display. The signal backplane motherboard ***(Section 3.5)*** interconnects the control and switch modules and also busses signals from the switch modules ***(Section 4.0)*** to the signal I/O module ***(Section 4.6)*** located in the chassis' last slot. Sixteen of the card cage's seventeen slots hold switch modules, while the last slot on the right, looking at it from the rear holds the signal I/O module.

A variety of available switch module types allow the system to be custom tailored to the user's specific requirements. Each switch module is typically built with either 16 or 32 electromechanical or solid state relays, and input/output signals are wired to connectors located at the rear edge of the module. Since the JX/256 chassis is modular, it may contain from one to sixteen switch modules. The chassis must always one control module, and one signal I/O module, however. The power supply operates from the AC line supply via a fused line cord adapter on the rear panel which also supplies the ON/OFF Switch.

The expansion chassis **(Drwg. #24-04-00)** differs from the mainframe in that it cannot operate as a stand-alone device and must be controlled remotely from a MESA control chassis. The JX/256 expansion chassis holds the same components as the JX/256 mainframe with two exceptions: the +5V power supply is always absent and the control module is replaced by the expansion interface module (**Drwg. #4-041-1)**. The expansion interface module plugs into the JX/256 backplane and is wired out to the chassis interface connector on the rear panel. Note that the JX/256 mainframe and expansion chassis differ from each other physically, and a mainframe cannot be adapted for use as an expansion chassis (or vice-versa). Refer to the "Shipped Configuration" sheet to determine which of the chassis drawing pertains to the system purchased.

3.2 SPECIFICATIONS

Dimensions:	19" Rack Mounting x 5.25" High x (15" or 11" Deep)
Weight:	Maximum weight with full complement of modules less than 32 lbs.
Maximum Power:	110 W @ 100-130 Vac or 110 W @ 200-260 Vac
Environment:	
Operating:	0C to 50C @ 95% Relative Humidity
Storage:	-25C to 65C @ 95% Relative Humidity
Capacity:	16 Switch Modules, 1 Signal I/O Module & 1 Control Module per chassis
Expansion Capacity:	Up to 16 Expansion Chassis with one MESA Unit
Display:	1 Power LED
Control Mode:	TCP/IP & RS232 Standard; IEEE488 as option.
	Break Before Make in Multiplexer Mode

Refer to Switch Module and Control Module sections of this manual for specifications on these topics.

3.3 POWER SUPPLY

The JX/256 mainframe holds two power supplies as shown in **Drwg. # 24-21-30**. One power supply (PS/5) provides 3.0 A at +5V and is used to power the logic circuitry. The second power supply (PS/12) provides 3.4 A at +12V and is used to drive the relays.

The power supplies will operate from 100-140 volts or 200-260 volts at 47-63 Hz.

The power supplies are wired to the selectable AC input module on the rear panel, which also holds the chassis' ON/OFF switch. The user can select one of two AC voltage ranges: 110/120 Volts or 220/240 volts AC. To change the selected voltage, remove the fuse cartridge using a small blade screw driver or a similar tool. Select the desired voltage by matching the arrow on the fuse cartridge to the arrow located on the input module's lower right corner. *Replace the fuse cartridge making sure the voltage selection arrow aligns with the arrow located on the Input Module*.

Two fuses are held in the fuse cartridge, with 220/240 VAC fused at 1.0 amp and 110/120 VAC fused at 2.0 amp.

3.3.1 OPTIONAL POWER SUPPLY (EXPANSION/HP CHASSIS)

For the expansion chassis, power is normally supplied from the MESA controller via the expansion interface module as shown in **Drwg. #4-041-1**. The JX expansion chassis may be ordered with an optional internal +12 V, 3.4 A PS/12 power supply or a +12 V, 7.5 amp supply. This additional power supply will be included only when the power from the MESA is not sufficient to drive all of the relays; as in the case when the user's application requires more than 150 relays to be closed simultaneously. This expansion chassis, the JX/256-E-PS is shown in **Drwg. #24-04-00 for the 3.4 A version and 24-20-00 for the 7.5 A version**.

3.4 FRONT PANEL

The JX/256 Mainframe's front panel will have a Power LED if the unit does not have the optional Keypad and LCD Display. If the Keypad and LCD have been ordered, there is no separate Power LED (the LCD reports power status).

A JX/256-E Expansion chassis built with the optional +12V power supply will also have a front panel Power LED.

3.5 MOTHERBOARDS

The JX/256 signal backplane **Drwg. #24-08-50** is used to bus control signals from the control module to the switch modules present in the system. The backplane also serves to bus the signals from the switch modules to the signal I/O module. The backplane is comprised of seventeen slots, the middle sixteen devoted to switch Modules, and one to a signal I/O module.

Power
Power is bussed to all 18 slots. Pins 4 and 19 on the backplane carry the ground signal and are connected together, both on the backplane and on each individual module. The +5V logic voltage is bussed on pin 5, likewise the +12V relay voltage is bussed on pin 18. These particular bus lines have links at their midpoints, splitting them into two sections which can be operated with separate power supplies.

Controls
The control module is located on the back of the front panel.

The controls for Switch Select, Latch/Unlatch, Mux/Clear, Switch Strobe, Status Strobe and Status Output are also bussed to all module slots.

For module selection, the backplane can be considered in two sections with modules 0 through 7 in one section, and modules 8 through 15 in the other section. Module select 0 on pin 8 of the control module is connected to pin 7 of switch modules 0 & 8. Similarly, module select 1 on pin

9 of the control module is connected to pin 7 of switch modules 1 & 9, and this pattern is repeated for module select 2 through 7.

Bit 8 on pin S of the control module is bussed to pin 15 of modules 0 through 7, and bit 8 on pin 16 of the control module is bussed to pin 15 of modules 8 through 15.

Signals

The backplane is bussed to handle 1, 2 or 3 pole relays, depending on the installed module type. There are also two sets of bussing for each switch module, since each of these may be separated into two separate 8 x 1 matrices.

Switches 0 through 7 are referenced as the "A" signals with Hi on pin #1, Lo on pin #2 and Shield on pin #3. Switches 8 through 15 are referenced as the "B" signals with Hi on pin #22, Lo on pin #21 and Shield on pin #20.

Although the third pole of the relay is referenced as Shield, as this is its most usual application, it can be used for any signal type.

4.0 SWITCH MODULES

Cytec currently offers over 20 different switch modules for installation in the JX/256-MF and JX/128-MF chassis. Connector options include D Subminiature, Header, or Screw Terminal connectors. A selection of switch modules can be viewed at the following link to Cytec's website: https://cytec-ate.com/application-guide/general/general_purpose_mods/

Note that the 'JX Modular Mux Test Systems' item should be checked under the 'Filters' heading to display the switch modules for this system. Both Schematics and Data Sheets can be accessed by clicking on the link located on the same line as the switch module name.

5.0 IF-12 GPIB / RS232 / LAN CONTROL MODULE

Introduction

CYTEC's IF-12 RS232/LAN Control Module is designed to control single chassis mainframes. Three forms of remote control are available on the module: IEEE488 (GPIB), RS232 and Ethernet LAN. An optional manual control is also available. All four interfaces may be active and used simultaneously.

Interface options: GPIB, RS232 and LAN are standard, but the Manual Control must be specified when purchasing the system. On some systems where panel space is limited, only two of the three interface connectors may be included.

5.1 LAN INTERFACE

Dynamic IP Address (DHCP): The Cytec IF12 is set at the factory to attempt to obtain an address from a DHCP server when the application boots. If you are connected to a network with a DHCP server, then the device IP address, network mask and gateway should be configured automatically. If your PC is on the same DHCP network, you will be able to communicate with the device after a short boot period of less than 10 seconds.

Static IP Address: If the module is plugged in to a network that does not have a DHCP server, you must provide a static IP address, network mask and gateway. These addresses should be provided by your network administrator.

Auto IP Address: The factory application contains an auto IP negotiation system. This allows the device to automatically configure its address in the absence of a central DHCP server, and without the need for a static IP address. This scheme is utilized as a fallback that will activate when both dynamic and static IP addresses fail to initialize. In order to communicate with a device in auto IP mode, the host system must support auto IP. Auto IP support is included in both Windows and OS X operating systems. By default, auto IP addressing starts in the 169.XXX.XXX.XXX address range.

Find Your Device: Our recommended option to locate the device is to use a local discover utility. You can do this by navigating to the Cytec web site and downloading the tool localdiscover.exe from https://cytec-ate.com/discover-cytec-local. The executable sends out a request to all Cytec devices on the local network. It opens a browser page on the first device to respond that lists all of the discovered devices, or a page that show that no devices were found.

Note: If these options are failing, there may be a firewall issue blocking the applications from sending the UDP broadcast that is used to locate Cytec devices. Always grant Cytec applications the ability to get through your OS firewall and ensure that UDP port 20034 is open for use.

5.2 RS232 INTERFACE

Signal Connections

The control module is pre-configured at the factory to operate as Data Communications Equipment (DCE) per the EIA RS232D Standard. In this configuration, the module transmits on the RxD Pin and receives on the TxD Pin. RTS is required to be high for the control module to transmit and CTS is output high by the control module to indicate a ready for data state and low when busy. The RS232 rear panel connector is a D9P (male) and can be run directly from a D9 computer COM port with a straight through (one to one) D9S to D9S cable. A null modem cable will not work with the factory default settings! Adaptors are available at any computer store to convert from D25 to D9. Do not use any adaptor that also acts as a null modem converter. If you are building your own cables, consult CYTEC Corp., for D25 to D9 pin out conversion.

D9P (male) PIN OUTS

Pin	Signal	Function
1	DCD	Not Used.
2	RxD	Data out of Control Module.
3	TxD	Data in to Control Module.
4	DTR	Not Used
5	Common	Signal Ground.
6	DSR	Not Used
7	RTS	Control Module requires + V to transmit.
8	CTS	Control Module provides +V when ready
9	RI	Not Used

Find Your Device

Open Device Manager on Windows computers and navigate to Ports. The COM port number will be bracketed next to the device description.

Configure Your Device

The RS232 interface can be accessed using any standard terminal emulation program such as PuTTY which can be downloaded from putty.org. Enter the COM port number in the field for the Serial line to connect to. The default values set at the factory are:

- Speed(baud): 9600
- Data bits: 8
- Stop bits: 1
- Parity: None
- Flow control: RTS/CTS (Hardware)

The first thing you should do is turn on Echo. This will enable you to see what you are typing. Make sure you turn Echo back off when you are done with the terminal session. Echo being left on will normally interfere with programs written specifically to control the switch.

Echo
Echos the characters back to your screen while you type them so you can see what you type.

Command:	"E 0 73"	Turns Echo Off
	"E 1 73"	Turns Echo On

Answerback
Answerback allows the Control Module to return information to the COM port. Answerback should almost always be left on. If Answerback is enabled, the Answerback byte **must** be read back by the requesting device. Failure to do so could have unpredictable results.

Command:	"A 0 73"	Turns Answerback Off
	"A 1 73"	Turns Answerback On

Verbose
Verbose causes the system to return more specific information when you request status or read answerback characters. It is sometimes helpful when troubleshooting but it slows the interface down a lot. While there may occasionally be a good reason to turn on Verbose during a puTTY or Hyperterm session, it is almost never used in a programmatic interface. All of the same information can be generated in code based on the non-verbose responses without slowing down the RS232 interface.

Command:	"V 0 73"	Turns Verbose Off
	"V 1 73"	Turns Verbose On

Baud Rate
Baud rate is set at the factory at 9600 Baud. Change is under software control and the control module must be connected to a serial interface to effect the change.

Baud	Baud# n
2400	4
4800	5
9600	6
19200	7
38400	8
57600	9
115200	10
230400	11 (untested, you should consider LAN)

Command:	"P19 n 73"	
	"P19 7 73"	sets baud rate to 19200.

If the Baud rate is inadvertently set to an unknown rate, the default value may be restored. See the section on Setting Defaults for the procedure. Obviously as soon as you reset the Cytec baud rate you will no longer be able to communicate with the switch until you reset the baud rate on your controlling computer or communication device.

CTS/RTS Handshake

The Clear to Send (CTS) and Request To Send (RTS) hardware handshaking functions may be modified by the 'P6' command.

Command:"P6 handshake 73"

handshake = 0	Handshaking off
handshake = 1	Handshaking on (default)

Example

"P6 0 73"	Turn handshaking off.

5.3 IEEE488 INTERFACE

Also known as GPIB (General Purpose Interface Bus), IEEE-488 is the international standard for a parallel interface used for attaching sensors and programmable instruments to a computer. When connecting IEEE-488 cables, some rules apply. The total number of devices should be 15 or less. The total length of all cables should not exceed 2 meters multiplied by the number of connected devices, up to a maximum of 20 meters. And no more than three connectors should be stacked together.

Find Your Device

Our recommended option to locate the device is to use NI Measurement & Automation Explorer (NI MAX), which can be downloaded from their website. Search for instruments in the application and the Cytec device should be found at default GPIB address 7.

Configure Your Device

GPIB Address:

Command syntax: “P14 n 73”.

For example, “P14 8 73” sets the GPIB address to 8.

5.3.1 IEEE488.2 SPECIFIC MATRIX COMMANDS

These commands are ignored by the RS232 interface.

***IDN? - Revision Number** (Same as Cytec “N” - Revision Command)

Syntax: “*IDN?”

The 'IDN?' command will cause the matrix to return its current revision number followed by an end of line.

Send:	"*idn?"	Request revision number.
Receive:	"Cytec VDX/32x32 11-01-13 1.0" eol	Text string indicating rev.

***RST - Reset** (same as C - Clear command)

The '*RST' command will clear (open all switches) in the matrix.

Send:	"*rst"	Reset.
Receive:	"0" eol	Returns ‘0'.

5.4 CONFIGURING TCP/IP PARAMETERS FROM A SERIAL CONNECTION

To change parameters you will need to access the serial interface using any standard terminal emulation program from the COM port on your computer. Once you have established a serial connection the following commands can be used for configuration:

D command returns a list of current settings:

A1, E1, V0 Answerback = ON, Echo = ON, Verbose = OFF
Baudnumber = 6, RS Handshaking = 1
IP Address = 10.0.0.144
Netmask = 255.255.255.0
Gateway = 0.0.0.0
Port0 = 8080, Port1 = 8081
TCP idle = 60
Telnetlock = 0, Telnet Echo = 0
Battery Ram = 0, Default List = 0

IFConfig command is used to set the static IP address. The syntax for this command is:

ifconfig aaa.aaa.aaa.aaa nnn.nnn.nnn.nnn

a = ip address in dotted decimal format n = subnet mask in dotted decimal format

Example: ifconfig 10.0.0.100 255.0.0.0

Typing ifconfig and hitting the enter key will return the current settings.

Since you may be connected via Telnet to do this, **the IP address will not actually change until you reboot the Cytec switch**. This helps prevent anyone from mistakenly setting the IP to an unknown address by accident. It is a good idea to double check the settings with the D command before you reboot.

HOSTS command sets the gateway for TCP/IP sockets. The syntax for this command is:

HOSTS xxx.xxx.xxx.xxx

Example: hosts 10.0.0.100

Typing hosts and hitting the enter key will return the current settings.

SNET TCP PORT command sets the Port number for TCP/IP sockets. The syntax for this command is:

SNET TCP PORT n m where n = equals one of two sockets and m is the port number

Example:

snet tcp port 0 8088 socket 0 is port #8088
snet tcp port 1 8089 socket 1 is port #8089

Port numbers must be between 1024 and 65535.
The Telnet port (23) may also be available. See TELNETLOCK command.

SNET TCP Idle command sets the socket life for the connection. The syntax for this command is:

SNET TCP Idle n (n=seconds) (1 to 3600 sec)
Default = 60 sec
SNET TCP Idle (display)
TCP Idle = 60
SNET TCP Idle 0 Socket never dies until the computer that established the socket kills it.

Setting the TCP Idle to 0 will force the socket to stay alive until the program that established the socket kills it.

WARNING: This can lead to issues if there is a network disconnect or the computer that established the socket locks up. If the computer that establishes the socket cannot kill the socket, no one will be able to connect to the switch until the Cytec unit is rebooted.

TCPAnswerback – Answerback

Syntax: TCPANSWERBACK n n = 0, 1 or 2

Answerback will enable or disable the transmission of a single character followed by an end of line upon the completion of all commands. The Answerback character will be a 1 or 0 depending on what command is sent. It is used to verify that the command was accepted and can verify completion of relay control commands. See **Section 5.5.2**

Eg.	"TCPANSWERBACK 0 "	Turn answerback off.
	"TCPANSWERBACK 1 "	Turn answerback on
	"TCPANSWERBACK 2 "	Turn answerback plus terminator on

Note: TCPANSWERBACK 2:
This setting appends a set of square brackets to the answerback byte.

Eg.	Send:	"L0 0"	Latch Module 0 Switch 0.
	Receive:	"1[]"	End of line follows the terminator

5.5 RUNNING THE CYTEC FACTORY APPLICATION ON THE IF12

The factory application that is included with the IF12 Control module includes:

- System Parameter Settings
- Matrix Parameter Settings
- Remote Switch Control
- List/Config Management
- File Management
- Custom Labeling

The URL request in the browser should look like the following:

http://<Device IP>

Where <Device IP> is replaced with the corresponding IP address. For more information on finding the IP address of your device, please see the device discovery section of this manual.

5.6 COMMAND FORMAT/COMPLETION

COMMAND FORMAT

All commands consist of at least one ASCII character indicating the command followed by optional values. After the command string is sent, an End of Line Character must be sent to affect the command.

If values are included with the command, the first value does not need to be separated from the command; all subsequent values MUST be separated by spaces or commas, eg. L1 2.

Multiple commands may also be sent on one line. Commands must be separated by a semi-colon character. Command line length is limited to 19 characters so avoid abusing this feature.

Examples:	"L2 7;C"	Connects Input 2 to Output 7 then clear
	"U4 7;L 1 2"	Unlatch Mod 4, Sw 7 then Latch Mod 1, Sw 2

COMMAND COMPLETION

A code representing the last requested switch point status (open or closed) and command completion will be stored by the matrix.

If the LAN or RS232 answerback function is enabled, a single character followed by end of line will be sent upon completion of all commands. Answerback may also include a termination character.

Note: Command Completion is NOT updated until the matrix finishes the requested operation.

Command Completion Codes – See Section 5.8 for error and completion codes

5.6.1 END OF LINE CHARACTER (EOL)

A received end of line character will cause the control module to execute the ASCII command string. The end of line character may be sent as a carriage return (CR) or New Line / Line Feed (NL/LF) for RS232 interfaces and a New Line / Line Feed (NL/LF) for IEEE488 interfaces or LAN interfaces. The IEEE488 also allows for the END control line being true with the last data character to initiate the command.

Valid end of Lines:

CR, LF or NL	LAN, RS232 or IEEE488
CR and END	IEEE488
LF/NL and END	IEEE488

Note that the terms New Line and Line Feed are often used to mean the same thing.
Both are expressed as \n in most programming languages and are shown on the ASCII table as "LF".
LF = Line Feed / New Line represented as \n, on ASCII table it is Decimal 10, or Hex A (0xA).
CR = Carriage Return represented as \r, on ASCII table it is = Decimal 13 or Hex D (0xD).

When any data is returned from the switch, the data will also be followed by an End Of Line character (EOL).

Notes - All Interfaces: Upon requesting status output characters MUST be received by the requesting device. Failure to do this will prevent further use of the matrix.

Access Code
Some commands require an access code number to be included with the command. This code prevents inadvertent operation of system modifying commands. The access code is 73.

5.7 SETUP COMMANDS

Matrixsize command sets the matrix size. The syntax for this command is:

matrixsize mtx# #mods #rlys

mtx#: For mainframes this is 0, for Mesa expansion systems this is the matrix number for the expansion chassis.

#mods: The maximum number of modules for the chassis.

#rlys: The maximum number of relays per module.

Example: matrixsize 0 16 8 Sets the # of modules to 16 and the maximum number of relays per module to 8 for a mainframe chassis (mtx is 0).

Typing matrixsize and hitting the enter key will return the current settings and chassis type (for Mesa systems all of the expansion chassis settings will be returned as a list).

P Commands (Except for communications settings these are set at the factory to the correct value for your system and should not need to be altered)

- P0 n 73 Set maximum number of matrices to 'n'. For Mainframes n = 1, for Mesa Control n = number of expansion chassis

- P6 n 73 n can be 1 or 0. 1 turns RTS/CTS handshaking on, 0 turns RTS/CTS handshaking off. This setting only applies to serial communication.

- P7 n 73 n can be 1 or 0. 1 turns Use RAM on, 0 turns Use RAM off.

- P8 n 73 n can be 0 to 6. Sets the default list (configuration) to load at power up if Use RAM is on.

- P10 n 73 Set maximum number of modules to 'n' for a Mainframe system. For Mesa Systems sets the maximum number of modules for Matrix 0;

- P11 n 73 Set maximum number of modules in Matrix 1 of a Mesa System to 'n'.

- P12 n 73 Set maximum number of modules in Matrix 2 of a Mesa System to 'n'.

- P13 n 73 Set maximum number of modules in Matrix 3 of a Mesa System to 'n'.

- P14 n 73 Set GPIB address to 'n'. n can be 0 to 31.

- P19 n 73 Set the baud number to ' n'. See RS232 configuration section for corresponding baud rate to baud number.

- P20 n 73 Set maximum number of relays to 'n' for a Mainframe system. For Mesa Systems sets the maximum number of relays for Matrix 0;

- P90 n 73 Set the system ID number to 'n'. Used in large systems to differentiate between chassis.

5.8 SWITCH COMMANDS

General Notes

For LAN and RS232, after sending any command the Cytec control will return an integer Answerback character if Answerback (TCPAnswerback for LAN) is ON. Answerback/TCPAnswerback is turned on by default and is Cytec's preferred operation since it allows you to verify commands are accepted before continuing.

If the command was a switch operation command such as Latch (L) or Unlatch (U), the character will be a meaningful status response where 1 = switch latched and 0 = switch unlatched. This may be used to verify that the command was received correctly.

Any other commands sent will also generate an answerback character which may be either a 1 or

0 and either character will indicate the command was received but the value is meaningless so either is acceptable.

Answerback may be turned off when using LAN or RS232 although it is not recommended. Answerback can also include a termination character for the LAN or RS232 interface.

Error Characters

If a command is sent incorrectly, an error character will be generated and added to the answerback character. Since the answerback character may be a 1 or 0, there may be two values for error characters as described below.

Answerback returned:

Dec	Hex	
1	30	Latch completed without errors.
0	31	Unlatch completed without errors.
2 or 3	32 or 33	Unknown command, first character unrecognizable.
4 or 5	34 or 35	Incorrect entries, number or type of entries incorrect.
6 or 7	36 or 37	Entries out of limits, switch point out of usable range.
8 or 9	38 or 39	Invalid access code, number 73 not included when required.

Delays to Prevent Errors

It is important to recognize that with modern computers and control interfaces, it is possible to stream commands to the switch matrix faster than the relays can physically operate. Many electro-mechanical relays may take between 2 to 20 ms to close or open. This can result in unpredictable results if certain operations are streamed together without considering this delay.

A good example of this type of problem occurs if a Latch command is sent and is immediately followed by a status request. Many of Cytec's products actually base status on current flow through the relay drives so it is possible to send a command and request status before the relay has physically operated, resulting in incorrect status feedback.

Typically, a 5 to 20 ms delay between commands requiring feedback can ensure that this is never an issue.

L,U,X – Latch, Unlatch, Multiplex Commands

Syntax: Cmd Switch
Cmd Module, Switch
Cmd Matrix, Module, Switch
The specified switchpoint is operated on. Note: For mainframe systems the matrix number will be 0.

(Cmd = 'L', 'U' or 'X')
L = Latch = Turn switch ON Closes the specified point, all others unaffected.

U = Unlatch = Turn switch OFF Opens the specified point, all others are unaffected.
X = Multiplex = Clear + Latch Opens all points, then Latches the specified point.

E.g. “U0 2 3” Matrix 0, Module 2, Switch 3 is opened. (OFF)
“L0 1 3” Matrix 0, Module 1, Switch 3 is closed. (ON)
“L0 1” Module 0, Switch 1 is closed. (ON)
“L2 3 7” Matrix 2, Module 3, Switch 7 is closed. (ON)
“X0 3 0” Clear all switch points (turn them all OFF) then Latch Matrix 0, Module 3, Switch 0.

If a single integer value is sent, the control module assumes it is a switch value and defaults to the last module value sent. If two integers are sent, the control module assumes they are a module and switch value and defaults to the last matrix value sent.

E.g. “L3 2 3” Matrix 3, Module 2, Switch 3 is closed (ON). Then, “L1 4” Assumes Matrix 3. Matrix 3, Module 1, Switch 4 is closed (ON). Then,“L5” Assumes Matrix 3, Module 1. Matrix 3, Module 1, Switch 5 is closed (ON).

Some Cytec programming examples may refer to Mod #, Rly # (Relay #). The terms Switch (Sw) and Relay (Rly) mean the same thing. For Unidirectional matrix switches the Module # may be thought of as Input #, and the Switch or Relay # may be thought of as the Output #.

C - Clear Command

Syntax: C
All points in the chassis are opened.

E.g. "C" All switches in the chassis are opened.

For IEEE488.2, The C command is the same as the *RST (reset) function.

5.9 STATUS COMMAND

The Status and Interrogate commands return information to the user so they can determine what state each switch point is in before proceeding. The commands can be used to simply check the switch configuration, to verify connections, or to prevent unwanted connections.

The information returned by these commands can be different depending on what type of system you have. Please find the Status or Interrogate section for your specific system before writing code that is dependent on the returned values.

Syntax:	S	Returns Status of entire mainframe chassis.
	S0 Module#	Returns Status of specified Module#.
	S0 Module# Switch#	Returns Status of specified Switch point.

Status may be requested of a single switch point or for the entire chassis. After receipt of the Status command the Matrix will return a character or string of characters representing the status, open or closed, of a switch point or switch points. A one, '1', signifies a closed switch point (ON) and a zero, '0', an open switch point (OFF).

In the case of a single switch point Status a single character is returned followed by an end of line.

The S command sent by itself will return a row / column pattern of 1's and 0's that mimic the LED display on the front panel (if equipped), where the columns are the Module # and the Rows are the Switch #.

Eg. A 16 Module, 8 Switch Matrix (your configuration may be different):

Send:	"S"	Status of chassis
Receive:	"0001000100000000" eol	Switch 0, Module 3 and 7 closed
	"0000000000000000" eol	Switch 1, none closed
	"1111111111111111" eol	Switch 2, all closed
	"1000000000000001" eol	Switch 3, Module 0 and 15 closed
	"1010101010101010" eol	Switch 4, odd Modules closed
	"0101010101010101" eol	Switch 5, even Modules closed
	"0110000000000000" eol	Switch 6, Module 1 and 2 closed
	"0000000000000110" eol	Switch 7, Module 13 and 14 closed
	"0"eol	Answerback character

If the Answerback function is on, the last 1 or 0 before the EOL will be the Answerback character and the value is a "don't care". For LAN Communication, if Answerback + termination character is ON (TCPAnswerback set to 2), the termination characters [] will be the last characters returned – this is in order to enable a coder to read the entire response with one chunk of code, (read until char returned equals ']').

5.10 INTERROGATE COMMAND

Syntax: I

The Interrogate function will return a list of all closed (ON) switch points. Each switch point will be followed by an “end of line” (EOL). The switch point is listed as the Module# and then Switch#. For matrix applications such as a 16x16 this often translates into “Input # then Output #” or “Output # then Input #”. Since many systems are bi-directional Input vs Output may be dependent on how you are using it. For uni-directional systems, such as VDM, or DXM, the input vs output relationship will be carved in stone and you should be familiar with it.

<I> Request interrogation.
Receive: <Module#><comma><space><Switch#><EOL>

Eg.	Send	“I”	
	Receive	“0, 0” eol	Module 0, Switch 0 Closed. End of line.
		“1, 6” eol	Module 1, Switch 6 Closed. End of line.
		“3, 2” eol	Module 3, Switch 2 Closed. End of line.
		“0” eol	Answerback character (if enabled). End of line.

For system such as a DX/256x256 the “I” command may return up to 256 addresses. Be sure your buffer size can handle the amount of returned data.

For Unidirectional matrix switches, specifically DX, DXM, VDX, VDM and TX, the Module # may be thought of as Input #, and the Switch or Relay # may be thought of as the Output #.

.

5.11 OTHER COMMANDS

F - Front Panel
Syntax: F n 73 n = 0 or 1

Front panel lock-out will be initiated by the receipt of a 0 character and enabled by the receipt of a 1 character followed by the access code. The access code prevents inadvertent lock-out from occurring. Lock-out will prevent any operation of the system from the front panel until it is terminated from the remote (F 1) or power is turned off then on. Preset to panel enabled at power on.

Eg.	"F 0,73"	Lock-out local operation.
	"F 1 73"	Enable local operation.

P - Program
Syntax: P n1,n2,73

The program command allows the operator to setup matrix dependent variables. These include matrix switch configuration and certain interface functions. Use of the P commands is complicated and varies greatly between systems. Your system should have been provided with the correct P command set-up.

If you need to change the matrix configuration, number of allowed modules, or other obscure set-up configurations on your system we recommend you contact Cytec and we can walk you through the P commands needed for your specific system. Please provide the serial # of your system when you contact us.

N - Revision Number (Same as IEEE488 *IDN?)
Syntax: N

The 'N' command will cause the matrix to return its current revision number followed by an integer identifier, followed by an end of line.

Eg. Send: "N" Request revision Number
Receive: "Cytec 11-21-60, IF-12, 2.12, 0" eol Text string indicating revision.

Where: “Cytec” = manufacturer.
“11-21-60" = control module board number.
“2.12" = Firmware Revision # (example).
“0" = Integer identifier.

Note: When requesting the Revision number, all characters must be received before the system can be resumed. *The text string received from the ‘N’ Command will vary depending on the type of system.*

Integer identifier

The N command now includes a single byte which can be used as an identifier for Cytec systems. The identifier is a single byte integer so it may 0 to 255. WE do not assign this and it has no meaningful relationship to any product. It is simply a number which may be assigned to a chassis so that the end user can acknowledge that a specific Cytec chassis is communicating. It is up to the customer to assign the number and keep track of it. It allows them to poll multiple chassis and know that the one they are talking to is, for example, the JX/256 that they assigned the identifier “13" to.

Command to enter or change the number:

P90 n 73 where n is the number from 0 to 255

5.12 INPUT / OUTPUT vs MODULE / SWITCH NOMENCLATURE

Most of the switching systems sold by Cytec are completely bi-directional and can be used in a variety of ways by the customer so it is impossible for us to use the terms Input and Output, even though it is what probably makes the most sense to the end user when connecting signals to the switch.

We label and control the switches using Module# and Switch# to avoid this confusion since for most systems either can be considered an Input or an Output.

5.13 LIST MANAGEMENT

Lists can be set most easily through the device webpage, which can be accessed by typing the IP address for the system into any browser address bar. Currently, Cytec switches allow only nine saved lists and list 0 is always the current latched points. Valid values for n are 1-9.

- BS n 73: Saves the current latched switchpoints in List n
- BL n 73: Clears the switch and loads the switch points in List n.
- BD n 73: Displays the switch points in List n.
- BC n 73: Clears List n.

5.14 MATRIX COMMAND SUMMARY

COMMAND	FUNCTION
L sw L mod, sw	Latch switch point.
U sw U mod, sw	Unlatch switch point.
X sw X mod, sw	Multiplex switch point.
C	Clear entire system.
S S mod, sw	Return status.
I	Interrogate Closed Points.
F 0/1 73	Disable/Enable Front Panel.
P parameter value 73	Program parameter.
N	Revision Number

RS232 Specific Commands

R baud, RTS/CTS 73	Baud Rate, RTS/CTS operation.
A 0/1 73	Disable/Enable Answerback.
E 0/1 73	Disable/Enable Echo.
V 0/1 73	Disable/Enable Verbose.

TCP/IP Specific Commands

TCPANSWERBACK 0/1/2	Disable/Enable Answerback/Termination
IFCONFIG aaa.aaa.aaa.aaa nnn.nnn.nnn.nnn	a = ip address in dotted decimal format n = subnet mask in dotted decimal format
SNET TCP PORT n m	Where n = equals one of two sockets and m is the port number

5.15 IF-12 (RS232/LAN/GPIB) DEFAULT CONFIGURATION SETTINGS

System parameters can be set most easily through the device webpage, which can be accessed by typing the IP address for the system into any browser address bar.

Default Values

TCP Settings:

Port 0	8080
Port 1	8081
Socket Timeout	60 seconds
TCPAnswerback	1 (on)
TelNet Lock	0 (off)

Serial Settings:

Answerback	1 (on)
Verbose	0 (off)
Echo	0 (off)
Baudrate	9600
RS Handshake	1 (RTS/CTS)

GPIB Settings:

GPIB Address	7

Front Panel Settings:

Mux Config	0 (Single 32) only for 16 or 32 channel systems
Front Panel	1 (on)

Miscellaneous Settings:

Use RAM (startup)	0 (off)
Default List	0 (currently latched switchpoints)
Sys ID Number	0

5.16 LCD DISPLAY/KEYPAD MANUAL CONTROL OPTION

The Keypad/Display option allows manual control of the matrix from the front panel. Keypad operation is always enabled at power on but may be disabled by the remote command, 'F 0 73'.

Display
The display contains two lines with sixteen characters per line. The top line displays matrix commands and numeric entry. The bottom line displays the status of the entry or operation. The display will also show the last command entered from the remote computer interface when the front panel is enabled.

Keypad
The keypad consists of ten numeric keys, four function keys, a space key and an enter key.

Key	Function
0-9	Numeric entries.
space	Delimits between numeric entries.
L	Latch operation.
U	Unlatch operation.
X	Multiplex operation.
C	Clear operation.
ENTR	Execute displayed operation.

Operation
A matrix command key, **L**, **U**, **X** or **C**, MUST be pressed before numeric entry keys. Pressing any key except a matrix command key causes the message **Enter Cmd First** to be displayed. After pressing a matrix command key the command and a cursor are displayed. The switch point to be operated on may now be entered with the numeric and space keys. The entry format is the same as described in the MATRIX OPERATION section and described briefly by the following table:

Command Key	Display Line 1	Line 2
L	Lat _	Enter Point
U	Unl _	Enter Point
X	Mux _	Enter Point
C	Clr _	Enter Matrix

The numeric keypad now allows selection of the Module and Relay (Input and Output) to be operated on. Each entry may be multiple digits and a space must be pressed between selections.

Key	Line 1 Display	Line 2 Message
L	Lat _	Enter Point
1	Lat 1_	
Space	Lat 1 _	
2	Lat 1 2_	
3	Lat 1 23_	
Enter Key	Lat 1 23_	1

The **ENTR** key may now be pressed to execute the displayed operation. If the displayed entry is incorrect or the operation is not desired, pressing any matrix command key will clear the display and restart the entry.

Status Display
After the **ENTR** key is pressed, the displayed operation is attempted to be executed by the control module. If the execution is successful, a **Point Closed** or **Point Open** message will be displayed on line 2. If the operation cannot be executed, an error message will be displayed.

Line 2 Message	Status
Ready	Displayed after power on.
Enter Point	The ENTR key has not been pressed, command and selection mode.
Point Closed	The selected point was closed.
Point Open	The selected point was opened.
Points Open	All points opened, Clear operation.
***Err: limits	The selected point is outside the programmed size of the matrix.
***Err: entry	An incorrect entry was selected.

Front Panel Disable
The 'F' command allows enabling or disabling front panel operation. If the front panel is disabled, no operation can be performed from the keypad.

Remote Command	Line 1	Line 2
F 0 73	Panel	Disabled
F 1 73	Panel	Enabled

Contrast and LED Backlight Adjustment
Controls are provided to adjust the LCD contrast and LED backlight level. These controls should only need adjustment in extremely bright or dim environments or for acute viewing angles. Both LCD and LED circuits have temperature sensing elements that will automatically adjust the output level for changes in the ambient temperature.

5.17 MEMORY SANITATION PROCEDURE

Cytec's IF-12 uses the NXP MCF54415 microprocessor with 32 MB of non-volatile flash memory which is used to store user lists and labels for the webpage factory application and 128kB of user parameter storage. If the unit is ever removed from service or needs to be sanitized for disposal the memory can be erased using one of the following methods.

1) Easiest with least damage. Contact Cytec for factory application .bin file at: sales@cytec-ate.com or 1-585-381-4740.

2) Permanent. Remove cover. Locate IF-12. Remove the NetBurner core board by disconnecting the LAN cable and prying the module off the IF-12 board. Destroy the NetBurner board. Unit is non functional until IF-12 has been replaced.

APPENDIX - EXAMPLE PROGRAMS

Java LAN Programming Example:

```
import java.net.*;  // for Socket
import java.io.*;   // for IOException and Input/OutputStream

public class if12_lantester
{
        static final int N_MODS = 4;
        static final int N_RLYS = 12;

        /**----------------test if12 utility functions--------------------------------------*/

        public static void main(String[] args) throws IOException, InterruptedException
        {
            if  (args.length != 2) // Test for correct # of args. IP Address and Port
                throw new IllegalArgumentException("Parameter(s): <IP Address> [<Port>]");

            String server = args[0];       // Server name or IP address
            int servPort = Integer.parseInt(args[1]);  // Port Number

            // Create socket that is connected to server on specified port
            Socket socket = new Socket(server, servPort);
            System.out.println("Connected to server...sending string");
            InputStream in = socket.getInputStream();
            OutputStream out = socket.getOutputStream();

            if12_lan if12 = new if12_lan();

            // Initialize Device: Turn Verbose & Echo off, Answerback on
            if (if12.init_LAN(in,out) < 0)
                throw new SocketException("Error Initializing Device");

            // Clear Device: Unlatch all relays
            if (if12.matrix_clear(in,out) != 48)
                throw new SocketException("Error clearing Device");

            // Latch and Unlatch Relays
            for (int mod =0; mod < N_MODS; mod++)
            {
                for (int rly=0;rly<N_RLYS;rly++)
                {
                    if (if12.point_ops(in,out,'L',0,mod,rly) != 49)
                    {
                        System.out.printf("Error latching Mod %d Rly %d\n",mod,rly);
```

```
                    break;
                }
                System.out.printf("Latched Mod %d Rly %d\n",mod,rly);
                if (if12.point_ops(in,out,'U',0,mod,rly) != 48)
                {
                    System.out.printf("Error unlatching Mod %d Rly %d\n",mod,rly);
                    break;
                }
                System.out.printf("Unlatched Mod %d Rly %d\n",mod,rly);
            }
        }

        socket.close();  // Close the socket and its streams
    }
}

public class if12_lan
{
    private int bytesRcvd,bytesSent;
    private byte[] rcvBuffer = new byte[256];

    public if12_lan()
    {
        bytesRcvd = 0;
        bytesSent = 0;
    }
    /**------------------Initialize Device-----------------------------------*/
    public int init_LAN(InputStream in, OutputStream out) throws IOException,
            InterruptedException
    {
        String str =new String("E0 73;V0 73;TCPANSWERBACK 1\n");
        // Convert string to bytes for writing to output stream
        byte[] byteBuffer = str.getBytes();
        // Send the encoded string to the if12
        out.write(byteBuffer);
        Thread.sleep(1000); //Wait one second
        // Receive the response from the device
        if ((bytesRcvd = in.read(rcvBuffer,0,9)) != 9)
            return -1;
        return 0;
    }
    /**--------------------clear matrix-----------------------------------*/
    public int matrix_clear(InputStream in,OutputStream out) throws IOException
    {
        String str = new String("C\n");
        // Convert string to bytes for writing to output stream
```

```
            byte[] byteBuffer = str.getBytes();
            // Send the encoded string to the if12
            out.write(byteBuffer);
            // Receive the response from the device
            if ((bytesRcvd = in.read(rcvBuffer,0,3)) == -1)
                return -1;
            return rcvBuffer[0];
        }
        /**------------------switchpoint operations----------------------------*/
        public int point_ops(InputStream in, OutputStream out,char cmd,int mtx,
            int mod,int rly) throws IOException, InterruptedException
        {
            //Format command string to send to device
            String cmd_line = String.format("%c%d %d %d\n",cmd,mtx,mod,rly);
            // Convert string to bytes for writing to output stream
            byte[] byteBuffer = cmd_line.getBytes();
            // Send the encoded string to the if12
            out.write(byteBuffer);
            Thread.sleep(100); //Wait 1/10 second
            // Receive the response from the device
            if ((bytesRcvd = in.read(rcvBuffer,0,3)) == -1)
                return -1;
            return rcvBuffer[0];
        }
}
```

C LAN Programming Example

```
/* Cytec Matrix Test Program for LAN  */
/* This program uses Microsoft's WS2_32 Library */
/* and winsock2.h. These are available in the */
/* Microsoft SDKs and can be downloaded from */
/* Microsoft's Developer Network */
/* https://msdn.microsoft.com/en-us/default.aspx */

#include <stdio.h>
#include <winsock2.h>
#include <stdlib.h>   /* for exit() */

int init_LAN(int sock);
int point_ops(int sock,int cmd, int mtx, int mod, int rly);
int matrix_clear(int sock);
void DieWithError(char *errorMessage);
```

```
#define MAX_MTX   1
#define MAX_MOD  4
#define MAX_RLY 12

int main(int argc, char *argv[])
{
        int sock;
        char *servIP = "10.0.0.144";  /*Default IP Address*?
        struct sockaddr_in servAddr; /* IP address */
        unsigned short servPort = 8080;  /* Port */
        int mtx, mod, rly,status;

        if (argc == 3)
        {
            servIP = argv[1];
            servPort = atoi(argv[2]);
        }


        WSADATA wsaData;          /* Structure for WinSock setup communication */
        WSAStartup(0x202, &wsaData);       /* Load Winsock 2.2 DLL */

        /* Create a reliable, stream socket using TCP */
        if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP))<0)
            DieWithError("socket() failed");

        /* Construct the server address structure */
        memset(&servAddr, 0, sizeof(servAddr));     /* Zero out structure */
        servAddr.sin_family      = AF_INET;         /* Internet address family */
        servAddr.sin_addr.s_addr = inet_addr(servIP);   /* Server IP address */
        servAddr.sin_port        = htons(servPort); /* Server port */

        /* Establish the connection to the server */
        if (connect(sock, (struct sockaddr *) &servAddr, sizeof(servAddr))<0)
                DieWithError("connect() failed");

        /* Initialize Device using init_LAN Function */
        init_LAN(sock);

        /* Send Clear Command to Device with matrix_clear Function*/
        if ((status = matrix_clear(sock)) != 48)
            printf("Error clearing device/n");

        /* Simple looping through switchpoints */

        for(mtx=0; mtx<MAX_MTX; mtx++)
```

```
        {
          for (mod=0; mod<MAX_MOD; mod++)
          {
            for (rly=0; rly<MAX_RLY; rly++)
            {
            if (((status = point_ops(sock,'L',mtx,mod,rly))) !=49)
                printf("Error point %d %d %d not closed\n",mtx,mod,rly);
                else
                printf("Latched point %d %d\n",mod, rly);

                if (((status = point_ops(sock,'U',mtx,mod,rly))) !=48)
                printf("Error point %d %d %d not open\n",mtx,mod,rly);
                else
                printf("Unlatched point %d %d\n",mod, rly);
            }
          }
        }
        closesocket(sock);
        WSACleanup();  /* Cleanup Winsock */
        return 0;
}
/*--------------Initialize---------------------------*/

int init_LAN(int sock)
{
        char rcvString[40];     /* Buffer for device response */
        int rcvStringLen;       /* Length of device response */
        void DieWithError(char *errorMessage);
        /* Initialize Device */
        if ((send(sock, "E0 73;V0 73;TCPANSWERBACK 1\n", 28, 0)) != 28)
        {
            DieWithError("send() failed");
        }
        Sleep(1000);     /* Wait for Response from Device */
        if ((rcvStringLen = recv(sock, rcvString, 9, 0)) < 9)
        {
            DieWithError("recv() failed or connection closed prematurely");
        }
        rcvString[rcvStringLen] = '\0';
        return 0;
}

/*-----------------Clear Matrix----------------------*/

int matrix_clear(int sock)
{
```

```
        char rcvString[8];     /* Buffer for device response */
        int rcvStringLen;      /* Length of device response */

        if ((send(sock,"C\n",2,0)) != 2)
            DieWithError("send() failed");

        Sleep(200);   /* Wait for Response

        /* Receive Response from Device */
        if ((rcvStringLen = recv(sock, rcvString, 10, 0)) <= 0)
            DieWithError("recv() failed or connection closed prematurely");
        rcvString[rcvStringLen] = '\0';
        int status = rcvString[0] & 0x3f;

        return status;
}
/*-----------Switchpoint Operation----------------  */

int point_ops(int sock,int cmd, int mtx, int mod, int rly)
{
        char cmd_str[40];          /* Formatted command string */
        char rcvString[8];     /* Buffer for device response */
        int rcvStringLen;      /* Length of device response */

        /* Format String */
        sprintf(cmd_str,"%c%d %d %d\n",cmd,mtx,mod,rly);

        /* Send Command to Device */
        if ((send(sock,cmd_str,strlen(cmd_str),0)) != strlen(cmd_str))
                DieWithError("send() failed");

        Sleep(200);   /* Wait for Response

        /* Receive Response from Device */
        if ((rcvStringLen = recv(sock, rcvString, 10, 0)) <= 0)
            DieWithError("recv() failed or connection closed prematurely");
        rcvString[rcvStringLen] = '\0';
        int status = rcvString[0] & 0x3f;

        return status;
}

/*------------Error Handling Function---------------------*/

 void DieWithError(char *errorMessage)
```

```
{
        fprintf(stderr,"%s: %d\n", errorMessage, WSAGetLastError());
        getchar();
        exit(1);
}
```

LabWindows RS232 Programming Example

```
*== Cytec Main Frame Control Include File rs232.h
==================================*/

int RS232port;

/*== GLOBAL FUNCTION DECLARATIONS
=======================================*/

int CYRS232Initialize (int com_port, int baud_rate);
int CyIf3_read (char *buf);
int CyIf3_write (char *buf);
int CyIf3_close (void);


/*============================== END */

#include <ansi_c.h>
#include <utility.h>
/*=============================================================*/
/* Cytec Main Frame RS232 LabWindows/CVI Driver Module         */
/*=============================================================/*

#include <rs232.h>
#include <formatio.h>
#include "CYRS232.h"

/*= STATIC VARIABLES
=====================================================*/
/* port contains the number of the port opened for the instrument module.  */
/* cmd is a buffer for RS-232 I/O strings.                                 */
/* rscnt contains the number of bytes transferred during a read or write.  */
/* CyIf3_err: the error variable for the instrument module                 */
/*=============================================================*/
//static int port;
static char cmd[26];
static int rscnt;
static int CyIf12_err;
```

```
/*= UTILITY ROUTINES
=====================================================*/
int CyIf12_invalid_short_range (short val, short min, short max, int err_code);
int CyIf12_invalid_integer_range (int val, int min, int max, int err_code);
int CyIf12_invalid_longint_range (long val, long min, long max, int err_code);
int CyIf12_invalid_real_range (double val, double min, double max, int err_code);
int CyIf12_read_data (char *buf, int cnt, int term);
int CyIf12_write_data (char *buf, int cnt);
int CyIf12_device_closed (void);
void CyIf12_setup_arrays (void);

int main()
{
        CYRS232Initialize(12,9600);

}

/*=================================================================*/
/* This function opens a com port for the instrument module, queries for   */
/* ID, and initializes the instrument to a known state.                    */
/*=================================================================*/
int CYRS232Initialize(int com_port, int baud_rate)
{
        char s[40];

   if (CyIf12_invalid_integer_range (baud_rate, 110, 19200, -2) != 0)
      return -14;


   CyIf12_err = OpenComConfig (com_port, "", baud_rate, 0, 8, 1, 512, 512);
   if (CyIf12_err<0) {
      return CyIf12_err;
   }
   CyIf12_err = SetComTime (com_port, 1.0);
   if (CyIf12_err<0) {
      return CyIf12_err;
   }


/*
   Set port to the number of the port just opened.
*/

   RS232port = com_port;

/* Initialize communication, Answerback ON, Verbose, Echo OFF */
```

```
        Fmt (s, "A1,73;V0,73;E0,73\r");
        CyIf12_err = (ComWrt (RS232port, s, StringLength(s)));
        if (CyIf12_err<0)  {
            return CyIf12_err;
        }
        Delay(.1);

        CyIf12_err = ComRdTerm(RS232port, s, 40, '\r');
        if (CyIf12_err<0) {
            return CyIf12_err;
        }
        Delay (1.0);

        FlushInQ (com_port);
        FlushOutQ (com_port);

   return CyIf12_err;
}

/*============================================================*/
int CyIf12_read (char *buf)
{
        return(CyIf12_read_data (buf, 40, '\r'));

}

int CyIf12_write (char *buf)
{
        return (CyIf12_write_data (buf, StringLength(buf)));

}


/*============================================================*/
/* This function closes the port for the instrument module and sets the    */
/* port to zero.                                              */
/*============================================================*/
int CyIf12_close (void)
{
/*  Check for device closed */

   if (CyIf12_device_closed())
      return CyIf12_err;

/*
   Close the com port.  If error, set CyIf3_err = rs232err+300.
```

```
*/

   CloseCom(RS232port);
   if (rs232err != 0) {
      CyIf12_err = rs232err+300;
      return CyIf12_err;
   }

   RS232port = 0;
   return CyIf12_err;
}

/* = UTILITY ROUTINES ==========================================*/

/*=============================================================*/
/* Function: Invalid Short Range                               */
/* Purpose:  This function checks a short to see if it lies between a      */
/*           minimum and maximum value.  If the value is out of range, set */
/*           the global error variable to the value err_code.  If the      */
/*           value is OK, error = 0.                           */
/*=============================================================*/
int CyIf12_invalid_short_range (short val, short min, short max, int err_code)
{
  if ((val < min) || (val > max)) {
    CyIf12_err = err_code;
    return -1;
  }
  return 0;
}
/*=============================================================*/
/* Function: Invalid Integer Range                             */
/* Purpose:  This function checks an integer to see if it lies between a   */
/*           minimum and maximum value.  If the value is out of range, set */
/*           the global error variable to the value err_code.  If the      */
/*           value is OK, error = 0.                           */
/*=============================================================*/
int CyIf12_invalid_integer_range (int val, int min, int max, int err_code)
{
  if ((val < min) || (val > max)) {
    CyIf12_err = err_code;
    return -1;
  }
  return 0;
}
/*=============================================================*/
/* Function: Invalid Long Integer Range                        */
```

```
/* Purpose:  This function checks a long integer to see if it lies between */
/*           a minimum and maximum value.  If the value is out of range,   */
/*           set the global error variable to the value err_code.  If the  */
/*           value is OK, error = 0.  The return value is equal to the     */
/*           global error value.                                           */
/*=========================================================================*/
int CyIf12_invalid_longint_range (long val, long min, long max, int err_code)
{
   if (val < min || val > max) {
      CyIf12_err = err_code;
      return -1;
   }
   return 0;
}
/*=========================================================================*/
/* Function: Invalid Real Range                                            */
/* Purpose:  This function checks a real number to see if it lies between  */
/*           a minimum and maximum value.  If the value is out of range,   */
/*           set the global error variable to the value err_code.  If the  */
/*           value is OK, error = 0.                                       */
/*=========================================================================*/
int CyIf12_invalid_real_range (double val, double min, double max, int err_code)
{
  if ((val < min) || (val > max)) {
    CyIf12_err = err_code;
    return -1;
  }
  return 0;
}
/*=========================================================================*/
/* Function: Device Closed                                                 */
/* Purpose:  This function checks to see if the module has been            */
/*           initialized.  If the device has not been opened, a 1 is       */
/*           returned, 0 otherwise.                                        */
/*=========================================================================*/
int CyIf12_device_closed (void)
{
   if (RS232port == 0)  {
      CyIf12_err = 232;
      return -1;
   }
   return 0;
}

/*=========================================================================*/
/* Function: Read Data                                                     */
```

```
/* Purpose:  This function reads a buffer of data from the instrument. The */
/*           return value is equal to the global error variable.          */
/*========================================================================*/
int CyIf12_read_data (char *buf, int cnt, int term)
{
   rscnt = ComRdTerm(RS232port, buf, cnt, term);
         FlushInQ (RS232port);

   return rscnt;
}

/*========================================================================*/
/* Function: Write Data                                                   */
/* Purpose:  This function writes a buffer of data to the instrument. The  */
/*           return value is equal to the global error variable.          */
/*========================================================================*/
int CyIf12_write_data (char *buf, int cnt)
{
   rscnt = ComWrt (RS232port, buf, cnt);

   return rscnt;
}

/*========================================================================*/
/* This function is called by the init routine to initialize global arrays  */
/* This routine should be modified for each instrument to include           */
/* instrument-dependent commmand arrays.                                    */
/*========================================================================*/
void CyIf12_setup_arrays (void)
{
}
/*= THE END
==========================================================================*/
```

LabWindows GPIB Programming Example

```
/*========================================================================*/

/*= Cytec IF-11 IEEE488 Control Module Include File ======================*/

/*== GLOBAL CONSTANT DECLARATIONS
========================================*/

/* Replace 10 with the maximum number of devices of this type being used.  */
#define IF12_MAX_INSTR    10
```

```
/*== GLOBAL FUNCTION DECLARATIONS
=========================================*/
int if12_init (int, int, int *);

/** INSERT INSTRUMENT-DEPENDENT FUNCTION DECLARATIONS HERE **/

int if12_operate(int, int, int, int, int *);
int if12_write (int, char *);
int if12_read (int, int, char *, int *);
int if12_close (int);

/*=== END INCLUDE FILE
=====================================================*/


/*============================================================================*/

#include <gpib.h>
#include <utility.h>
#include <formatio.h>
#include "cy_if12.h"


/*= INSTRUMENT TABLE
=====================================================*/
/* address array: contains the GPIB addresses of opened instruments.       */
/* bd array: contains the device descriptors returned by OpenDev.          */
/* instr_cnt: contains the number of instruments open of this model type.  */
/*============================================================================*/
static int address[IF12_MAX_INSTR + 1];
static int bd[IF12_MAX_INSTR + 1];
static int instr_cnt;

/*= STATIC VARIABLES
=====================================================*/
/* cmd is a buffer for GPIB I/O strings.                                   */
/* if12_err: the error variable for the instrument module                  */
/* ibcnt: contains the number of bytes transferred by GPIB reads and       */
/*        writes.  See the GPIB library I/O Class for more information     */
/*============================================================================*/
static char cmd[50];
static int if12_err;

/*= UTILITY ROUTINES
=====================================================*/
```

```
int if12_open_instr (int, int *);
int if12_close_instr (int);
int if12_invalid_integer_range (int, int, int, int);
int if12_device_closed (int);
int if12_read_data (int, char *, int);
int if12_write_data (int, char *, int);
int if12_set_timeout (int, int, int *);
void if12_setup_arrays (void);

/*=============================================================*/
/* Function: Initialize                                        */
/* Purpose:  This function opens the instrument, queries the instrument    */
/*           for its ID, and initializes the instrument to a known state.  */
/*=============================================================*/
int if12_init (addr, rest, instrID)
int   addr;
int   rest;
int * instrID;
{
        int  ID;

        if (if12_invalid_integer_range (addr, 0, 30, -1) != 0)
            return if12_err;
        if (if12_invalid_integer_range (rest, 0, 1, -3) != 0)
            return if12_err;

        if (if12_open_instr (addr, &ID) != 0)
            return if12_err;

        if (rest) {
            if (if12_write_data (ID, "C", 1) != 0) {
                if12_close_instr (ID);
                return if12_err;
            }
            Delay(0.01);
        }
        if12_setup_arrays ();
        *instrID = ID;

        return if12_err;
}

/*=============================================================*/
/* - Operations: Latch, Unlatch, Multiplex, Clear and Status --- */

int if12_operate (instrID, Operation, Module, Relay, Status)
```

```
int instrID, Operation, Module, Relay;
int *Status;
{
        char s[20];

        *Status = -1;
        if (Operation == 'C') {
            Fmt(s,"C");
            if (if12_write_data(instrID, s, StringLength(s)) != 0)
                return if12_err;
            Delay(0.01);
        }
        else {
            Fmt(s,"%c %d %d", Operation, Module, Relay);
            if (if12_write_data(instrID, s, StringLength(s)) != 0)
                return if12_err;
        }
        if (if12_read_data(instrID, s, 2) != 0)
            return if12_err;
        *Status = s[0] & 0xf;
        return if12_err;
}

/*=============================================================*/
/* Function: Write To Instrument                               */
/* Purpose:  This function writes a command string to the instrument.      */
/*=============================================================*/
int if12_write (instrID, cmd_string)
int   instrID;
char *cmd_string;
{
        if (if12_invalid_integer_range (instrID, 1, IF12_MAX_INSTR, -1) != 0)
            return if12_err;
        if (if12_device_closed(instrID) != 0)
            return if12_err;

        Fmt (cmd, "%s<%s", cmd_string);
        if (if12_write_data (instrID, cmd, NumFmtdBytes()) != 0)
            return if12_err;

        return if12_err;
}

/*=============================================================*/
/* Function: Read Instrument Buffer                            */
/* Purpose:  This function reads the output buffer of the instrument.      */
```

```
/*=============================================================*/
int if12_read (instrID, numbytes, in_buff, bytes_read)
int   instrID;
int   numbytes;
char *in_buff;
int  *bytes_read;
{
        if (if12_invalid_integer_range (instrID, 1, IF12_MAX_INSTR, -1) != 0)
            return if12_err;
        if (if12_device_closed(instrID) != 0)
            return if12_err;

        *bytes_read = 0;
        if (if12_read_data (instrID, in_buff, numbytes) != 0)
            return if12_err;

        *bytes_read = ibcnt;

        return if12_err;
}

/*=============================================================*/
/* Function: Close                                             */
/* Purpose:  This function closes the instrument.              */
/*=============================================================*/
int if12_close (instrID)
int instrID;
{
        if (if12_invalid_integer_range (instrID, 1, IF12_MAX_INSTR, -1) != 0)
            return if12_err;
        if (if12_device_closed (instrID))
            return if12_err;

        if12_close_instr (instrID);

        return if12_err;
}

/*= UTILITY ROUTINES ========================================*/

/*=============================================================*/
/* Function: Open Instrument                                   */
/* Purpose:  This function locates and initializes an entry in the  */
/*           Instrument Table and the GPIB device table for the     */
/*           instrument.  The size of the Instrument Table can be changed  */
/*           in the include file by altering the constant           */
```

```
/*          IF12_MAX_INSTR.  The return value of this function is equal */
/*          to the global error variable.                                */
/*=======================================================================*/
int if12_open_instr (addr, ID)
int  addr;
int *ID;
{
        int i, instrID;

        instrID = 0;
        if12_err = 0;

/* Check to see if the instrument is already in the Instrument Table. */

        for (i = 1; i <= IF12_MAX_INSTR; i++)
            if (address[i] == addr) {
                instrID = i;
                i = IF12_MAX_INSTR;
             }

/* If it is not in the instrument table, open an entry for the instrument. */

        if (instrID <= 0)
            for (i = 1; i <= IF12_MAX_INSTR; i++)
                if (address[i] == 0) {
                    instrID = i;
                    i = IF12_MAX_INSTR;
                 }

/* If an entry could not be opened in the Instrument Table, return an error.*/

        if (instrID <= 0) {
            if12_err = 220;
            return if12_err;
        }

/* If the device has not been opened in the GPIB device table (bd[ID] = 0),*/
/*  then open it.                                                         */

        if (bd[instrID] <= 0) {
            if (instr_cnt <= 0)
                CloseInstrDevs("if12");
            bd[instrID] = OpenDev ("", "if12");
            if (bd[instrID] <= 0) {
                if12_err = 220;
                return if12_err;
```

```
            }
            instr_cnt += 1;
            address[instrID] = addr;
        }

/*  Change the primary address of the device    */

        if (ibpad (bd[instrID], addr) < 0) {
            if12_err = 233;
            return if12_err;
        }

        *ID = instrID;
        return if12_err;
}

/*=============================================================*/
/* Function: Close Instrument                                  */
/* Purpose:  This function closes the instrument by removing it from the   */
/*          GPIB device table and setting the address and bd[instrID] to  */
/*          zero in the Instrument Table.  The return value is equal to   */
/*          the global error variable.                         */
/*=============================================================*/
int if12_close_instr (instrID)
int instrID;
{
        if (bd[instrID] != 0) {
            CloseDev (bd[instrID]);
            bd[instrID] = 0;
            address[instrID] = 0;
            instr_cnt -= 1;
        }
        else
            if12_err = 221;

        return if12_err;
}

/*=============================================================*/
/* Function: Invalid Integer Range                             */
/* Purpose:  This function checks an integer to see if it lies between a   */
/*          minimum and maximum value.  If the value is out of range, set */
/*          the global error variable to the value err_code.  If the      */
/*          value is OK, error = 0.  The return value is equal to the     */
/*          global error value.                                */
/*=============================================================*/
```

```
int if12_invalid_integer_range (val, min, max, err_code)
int val;
int min;
int max;
int err_code;
{
        if (val < min || val > max)
            if12_err = err_code;
        else
            if12_err = 0;

        return if12_err;
}

/*=============================================================*/
/* Function: Device Closed                                     */
/* Purpose:  This function checks to see if the module has been */
/*          initialized.  If the device has not been opened, set the */
/*          global error variable to 232, 0 otherwise. The return value */
/*          is equal to the global error value.                */
/*=============================================================*/
int if12_device_closed (instrID)
int instrID;
{
        if (bd[instrID] <= 0)
            if12_err = 232;
        else
            if12_err = 0;

        return if12_err;
}

/*=============================================================*/
/* Function: Read Data                                         */
/* Purpose:  This function reads a buffer of data from the instrument. The */
/*          return value is equal to the global error variable.  */
/*=============================================================*/
int if12_read_data (instrID, buf, cnt)
int   instrID;
char *buf;
int   cnt;
{
        if (ibrd(bd[instrID], buf, (long)cnt) <= 0)
            if12_err = 231;
        else
            if12_err = 0;
```

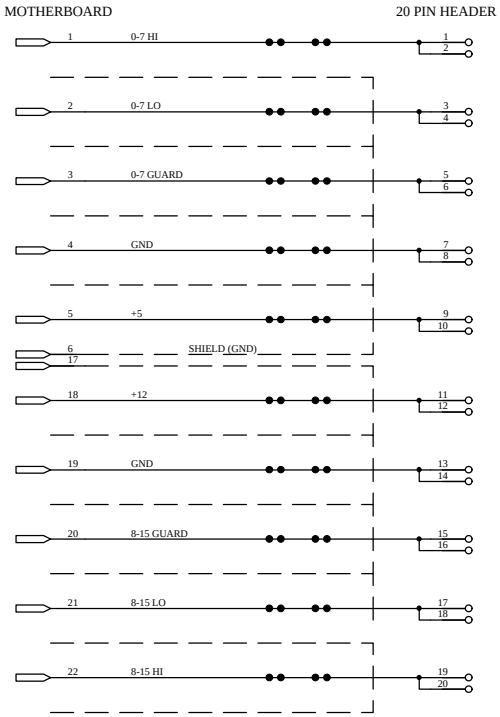
```
        return if12_err;
}

/*=============================================================*/
/* Function: Write Data                                        */
/* Purpose:  This function writes a buffer of data to the instrument. The  */
/*           return value is equal to the global error variable.           */
/*=============================================================*/
int if12_write_data (instrID, buf, cnt)
int   instrID;
char *buf;
int   cnt;
{
        if (ibwrt(bd[instrID], buf, (long)cnt) <= 0)
            if12_err = 230;
        else
            if12_err = 0;

        return if12_err;
}

/*=============================================================*/
/* Function: Set Timeout                                       */
/* Purpose:  This function changes or disables the timeout of the device.  */
/*           Refer to the LabWindows Standard Libraries Reference Manual   */
/*           for timeout codes. The return value is equal to the global    */
/*           error variable.                                               */
/*=============================================================*/
int if12_set_timeout (instrID, tmo_code, old_timeout)
int  instrID;
int  tmo_code;
int *old_timeout;
{
        *old_timeout = ibtmo (bd[instrID], tmo_code);
        if (ibsta <= 0)
            if12_err = 239;
        else
            if12_err = 0;

        return if12_err;
}

/*=============================================================*/
/* Function: Setup Arrays                                      */
/* Purpose:  This function is called by the init routine to initialize     */
```

```
/*          static arrays.                                  */
/*          This routine should be modified for each instrument to      */
/*          include instrument-dependent commmand arrays.               */
/*=============================================================*/
void if12_setup_arrays ()
{
}
/*=== THE END ===============================================*/
```

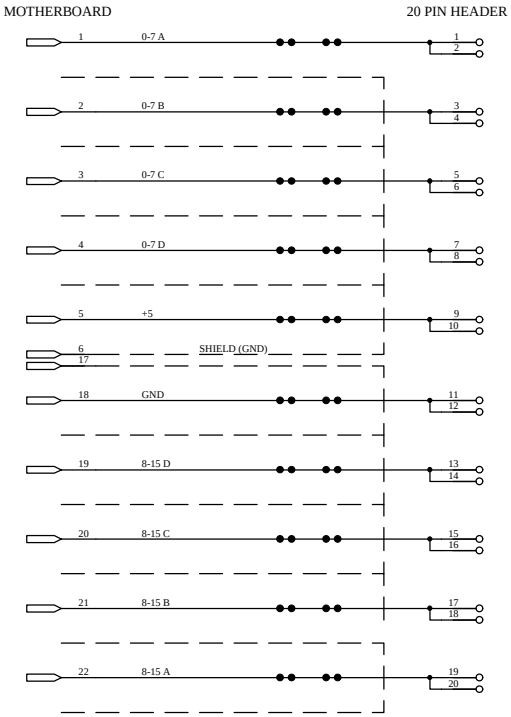
LabView Drivers

Labview Drivers are available for download at https://cytec-ate.com/downloads/drivers/

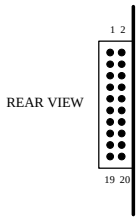
JX16/L1 L2 L3
SIGNAL CONNECTIONS



JX16/4AB
SIGNAL CONNECTIONS



REVISED FROM:

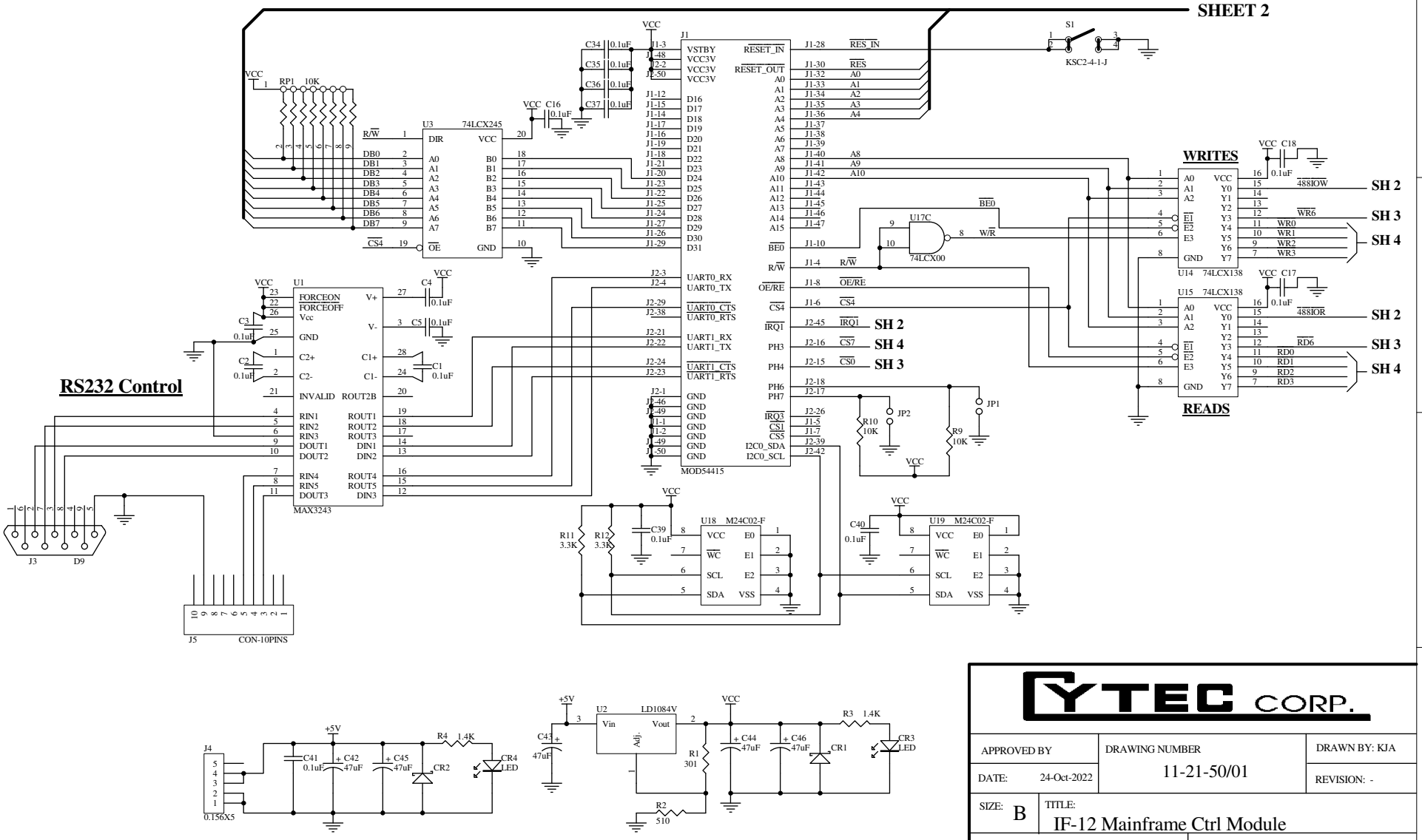


CYTEC CORP.

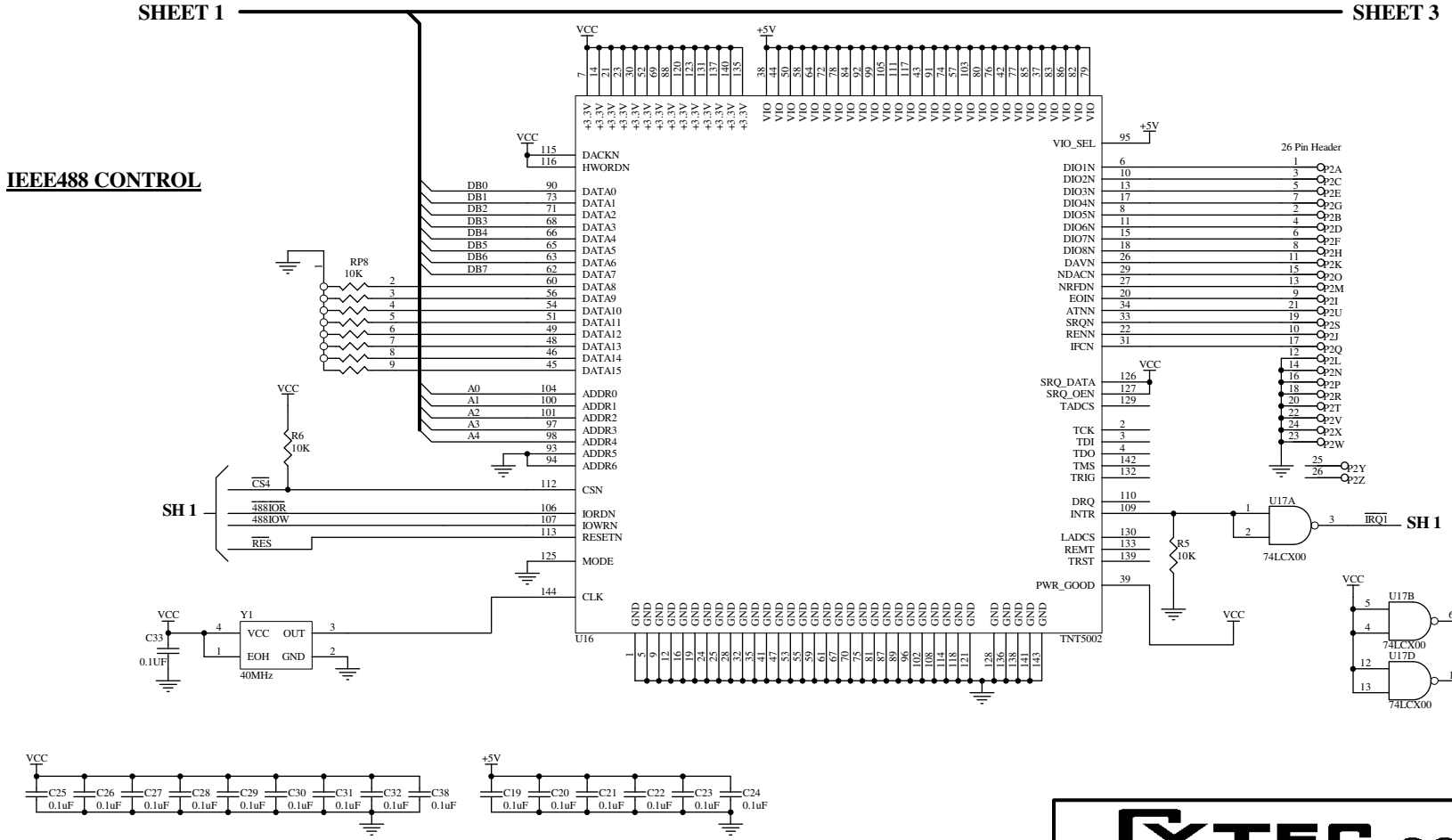
APPROVED BY	DRAWING NUMBER	DRAWN BY: RHB
DATE: 25-May-2000	4-058	REVISION:
SIZE: B	TITLE: JX SIGNAL I/O MODULE	
F:\BDrawings\jx-sch\4-058mod.Sch		SHEET 1 OF 1

MAIN CONTROLLER

SHEET 2



YTEC CORP.			
APPROVED BY		DRAWING NUMBER	DRAWN BY: KJA
DATE:	24-Oct-2022	11-21-50/01	REVISION: -
SIZE: B	TITLE: IF-12 Mainframe Ctrl Module		
11-21-50-01.sch		SHEET 1 OF 4	

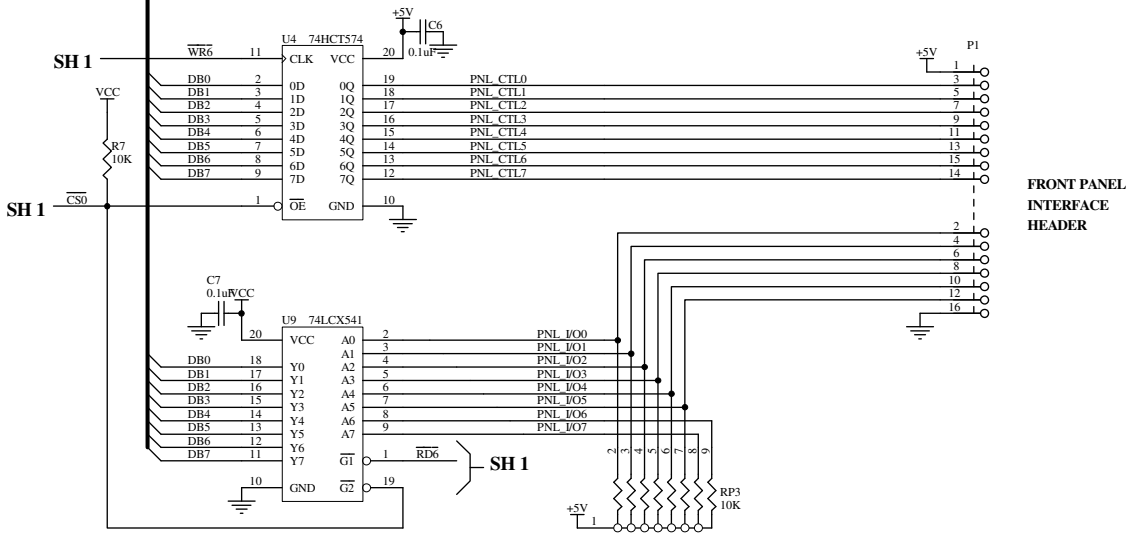


APPROVED BY	DRAWING NUMBER	DRAWN BY: KJA
DATE: 24-Oct-2022	11-21-50/02	REVISION: -
SIZE: B	TITLE: IF-12 Mainframe Ctrl Module	
11-21-50-02.sch	SHEET 2 OF 4	

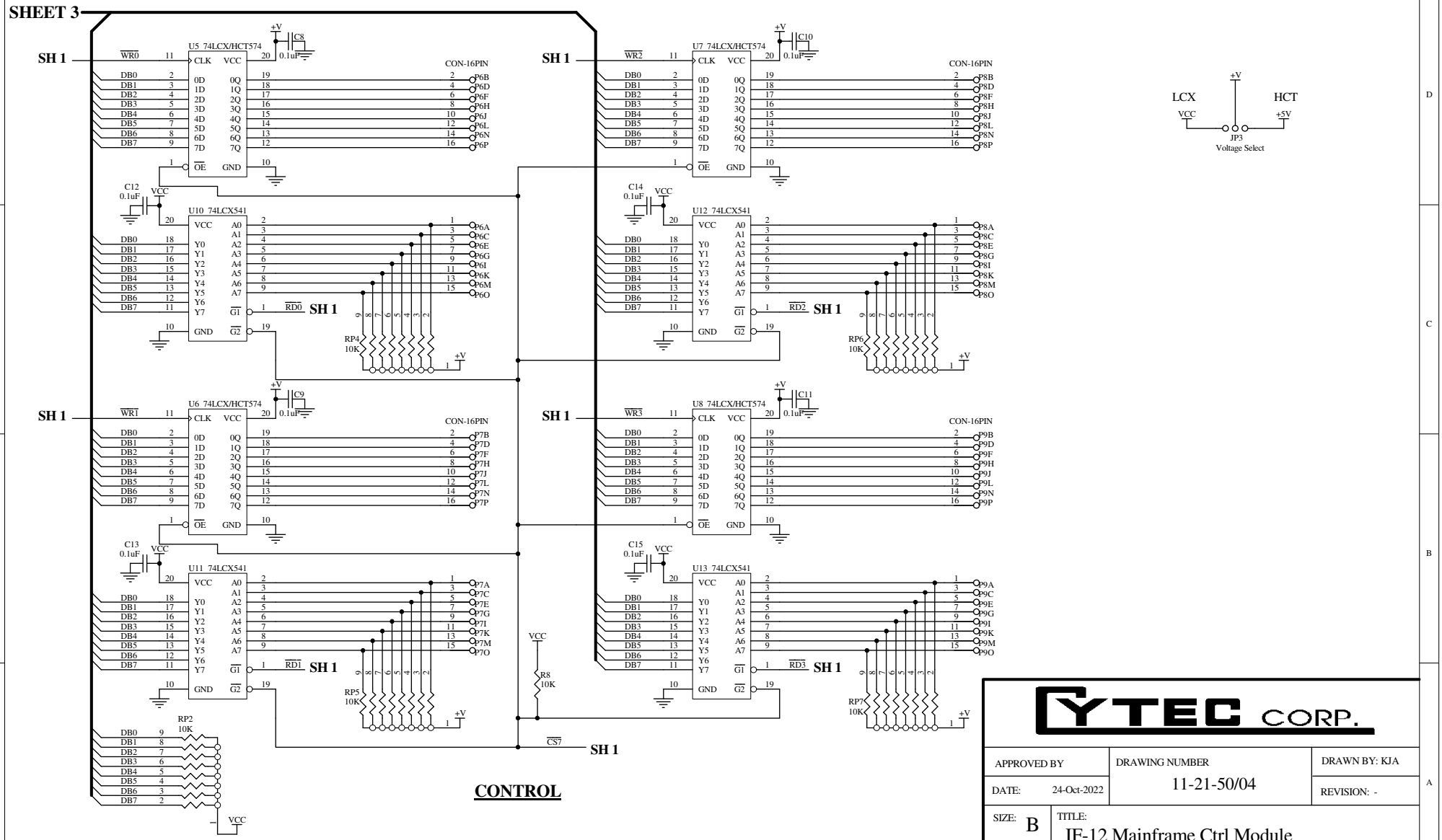
SHEET 2

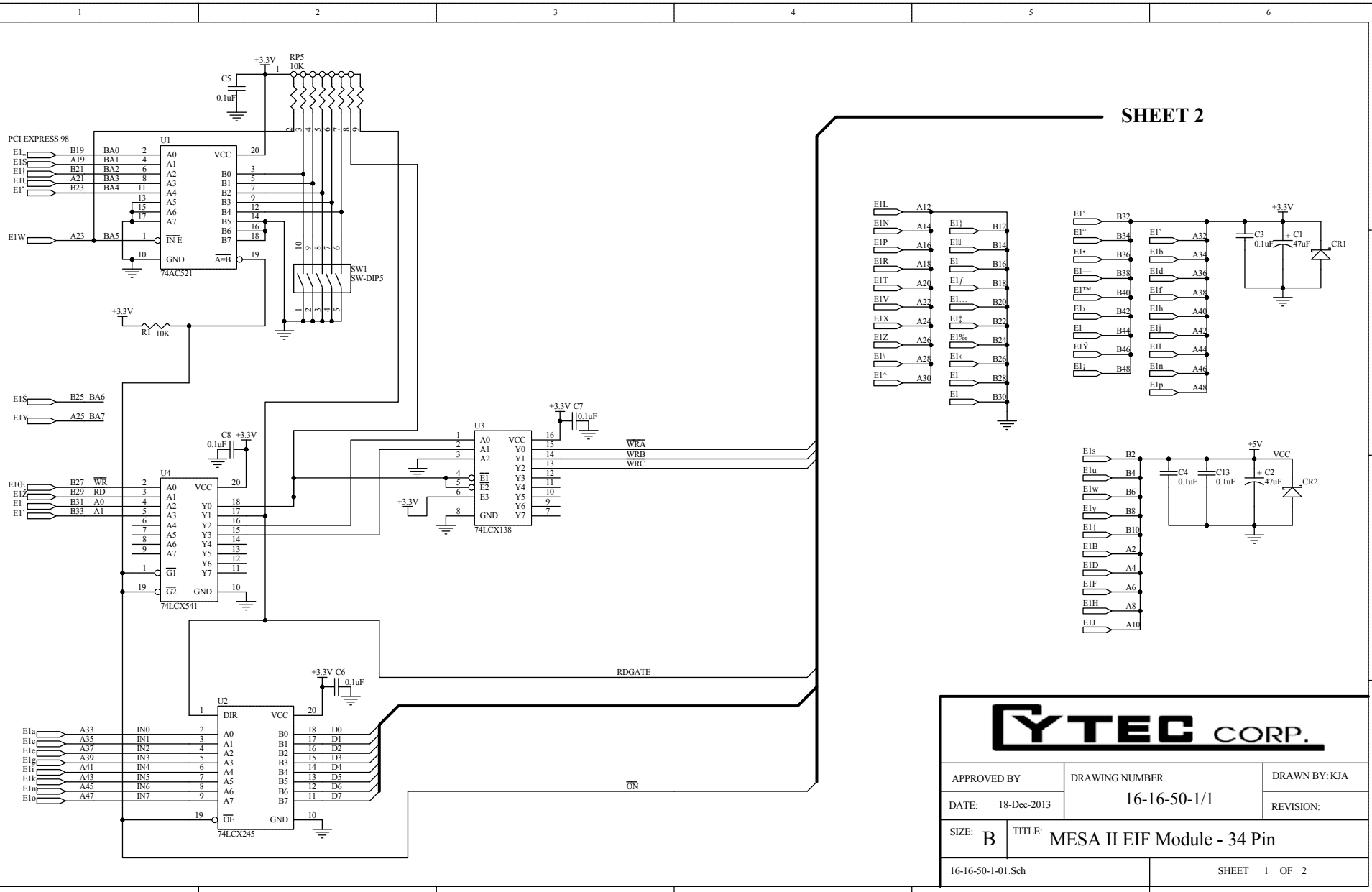
SHEET 4

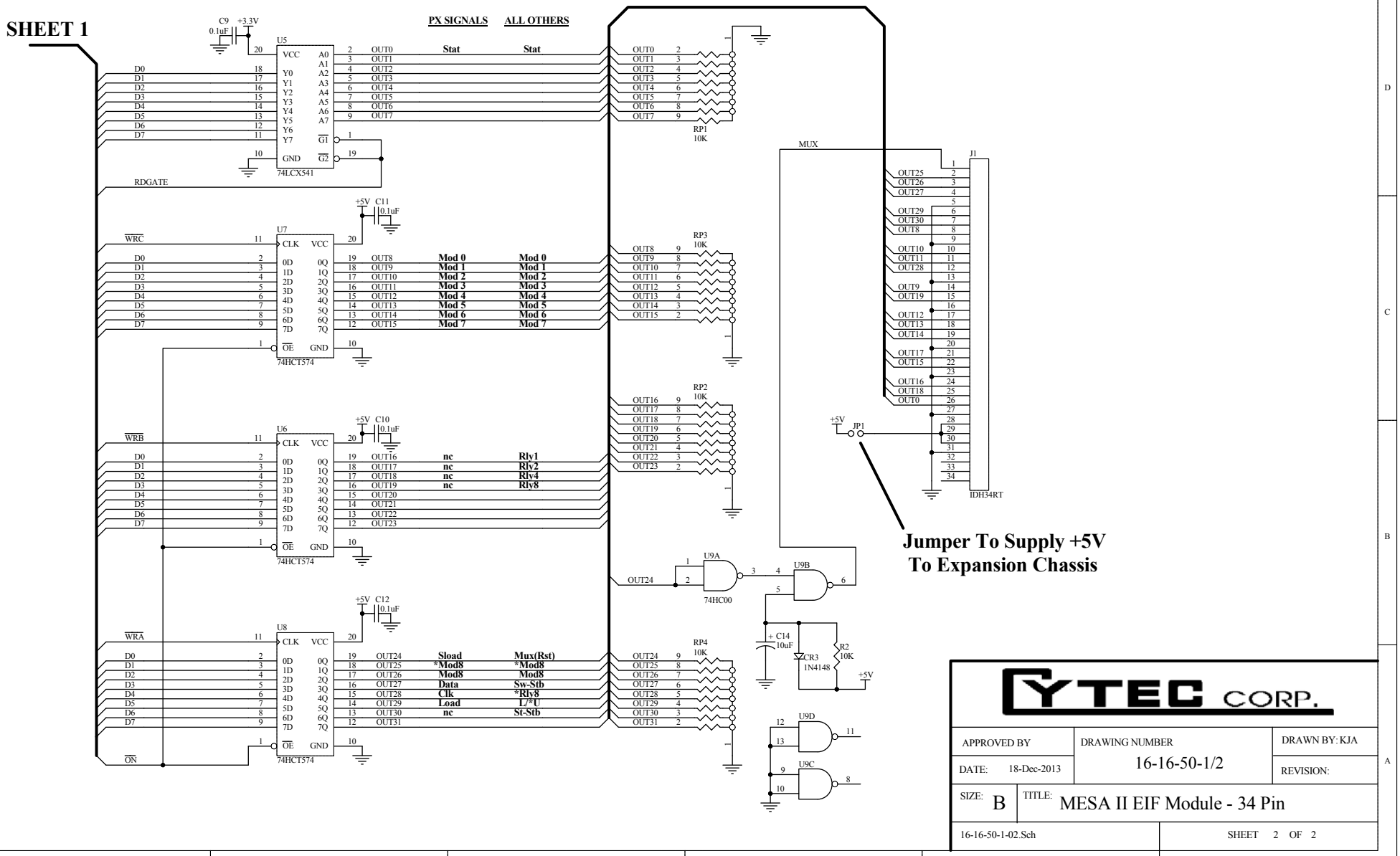
LCD CTRL



CYTEC CORP.		
APPROVED BY	DRAWING NUMBER	DRAWN BY: KJA
DATE: 24-Oct-2022	11-21-50/03	REVISION: -
SIZE: B	TITLE: IF-12 Mainframe Ctrl Module	
11-21-50-03.sch		SHEET 3 OF 4


SHEET 3
SH 1
U5 74LCX/HCT574
U7 74LCX/HCT574
U6 74LCX/HCT574
U8 74LCX/HCT574
U10 74LCX541
U12 74LCX541
U11 74LCX541
U13 74LCX541
CON-16PIN
P6B P6D P6F P6H P6J P6L P6N P6P
P8B P8D P8F P8H P8J P8L P8N P8P
P7B P7D P7F P7H P7J P7L P7N P7P
P9B P9D P9F P9H P9J P9L P9N P9P
P6A P6C P6E P6G P6I P6K P6M P6O
P8A P8C P8E P8G P8I P8K P8M P8O
P7A P7C P7E P7G P7I P7K P7M P7O
P9A P9C P9E P9G P9I P9K P9M P9O
C8 C9 C10 C11 C12 C13 C14 C15
0.1uF
+V
VCC
RP2 10K
RP4 10K
RP5 10K
RP6 10K
RP7 10K
R8 10K
RD0 RD1 RD2 RD3
CS7
LCX
HCT
+5V
JP3
Voltage Select
CONTROL
CYTEC CORP.
APPROVED BY
DRAWING NUMBER
11-21-50/04
DRAWN BY: KJA
DATE: 24-Oct-2022
REVISION: -
SIZE: B
TITLE:
IF-12 Mainframe Ctrl Module
11-21-50-04.sch
SHEET 4 OF 4


SHEET 2
PCI EXPRESS 98
U1
74AC521
SW1
SW-DIP5
RP5
10K
C5
0.1uF
R1
10K
U4
74LCX541
U3
74LCX138
U2
74LCX245
C8
C7
C6
RDGATE
ON
C1
C2
C3
C4
C13
CR1
CR2
+3.3V
+5V
VCC
GND
TYTEC CORP.
APPROVED BY
DATE: 18-Dec-2013
DRAWING NUMBER
16-16-50-1/1
DRAWN BY: KJA
REVISION:
SIZE: B
TITLE: MESA II EIF Module - 34 Pin
16-16-50-1-01.Sch
SHEET 1 OF 2

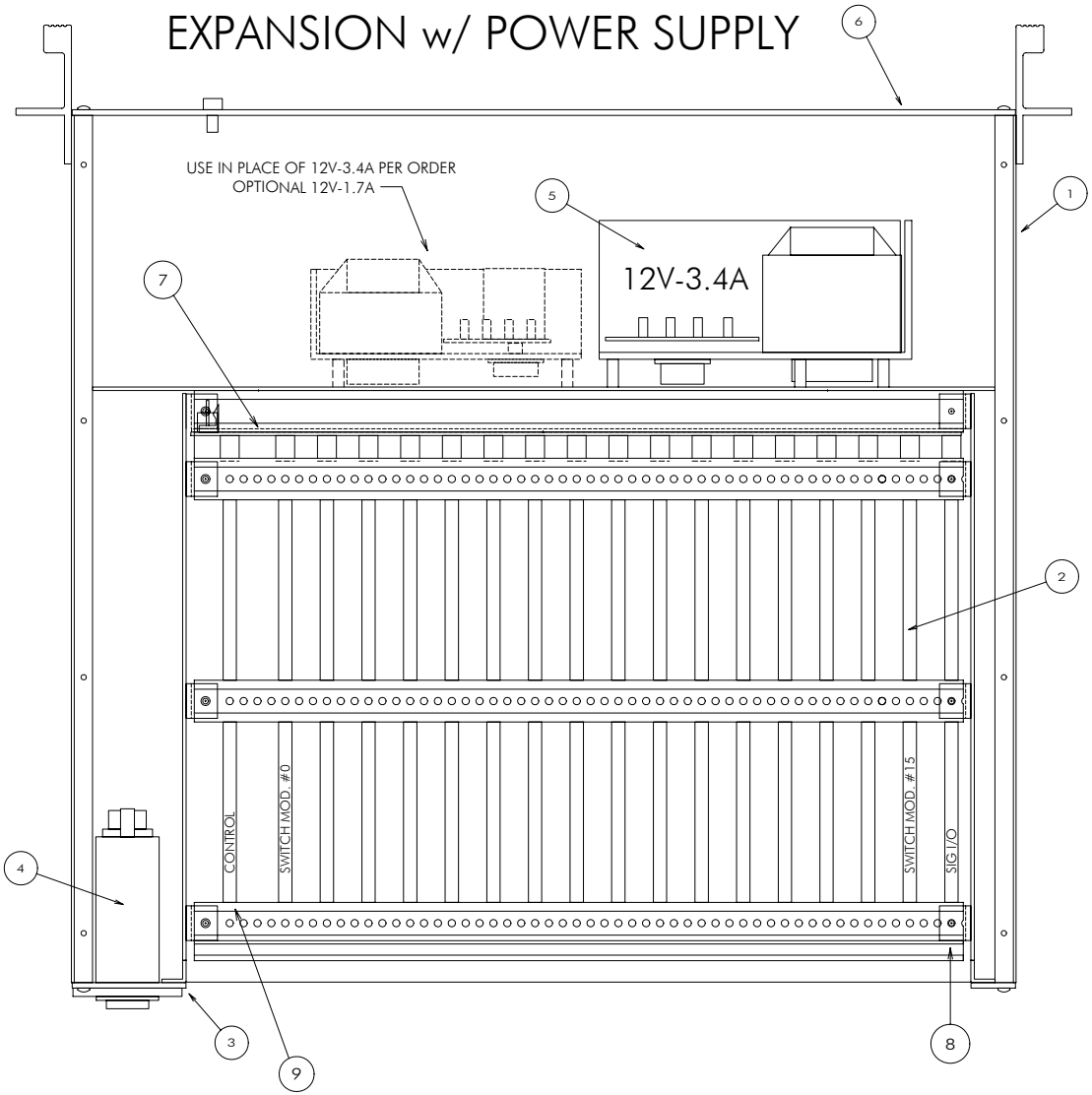


Stat	Stat
Mod 0	Mod 0
Mod 1	Mod 1
Mod 2	Mod 2
Mod 3	Mod 3
Mod 4	Mod 4
Mod 5	Mod 5
Mod 6	Mod 6
Mod 7	Mod 7
nc	Rly1
nc	Rly2
nc	Rly4
nc	Rly8
Sload	Mux(Rst)
*Mod8	*Mod8
Mod8	Mod8
Data	Sw-Stb
Clk	*Rly8
Load	L/*U
nc	St-Stb

APPROVED BY	DRAWING NUMBER	DRAWN BY: KJA
DATE: 18-Dec-2013	16-16-50-1/2	REVISION:
SIZE: B	TITLE: MESA II EIF Module - 34 Pin	
16-16-50-1-02.Sch		SHEET 2 OF 2

APPROVED 6/24/19

EXPANSION w/ POWER SUPPLY



1. 14-20-70 x 3U 15.62" CHASSIS ASSY
   15.62" DEEP x 19" WIDE x 5-1/4" HIGH
2. 24-01-20 JX/256 CAGE ASSY
3. 24-02-21 REAR PANEL
4. AC SELECTABLE POWER ENTRY
5. 12V-3.4A POWER SUPPLY
   OR OPTIONAL 12V-1.7A POWER SUPPLY
6. 15-14-22 5-1/4" FRONT PANEL
7. 24-08-10 BACKPLANE
8. 4-058 SIG I/O (JX Only)
9. 4-041-1 EXP. INTERFACE MODULE
   PER A.P.D.- OMIT L2 FERRITE

24-02-30 POWER WIRING DIA.

6-015-3 MESA E.I.F. ASSY

C	4/23/03	DR	UPDATED WIRE DIA.
B	5/1/02	DR	UPDATED ASSY LIST
A	11/12/01	DR	UPDATED PARTS LIST

CYTEC CORP.

TOL. +/- .010" U.O.S.

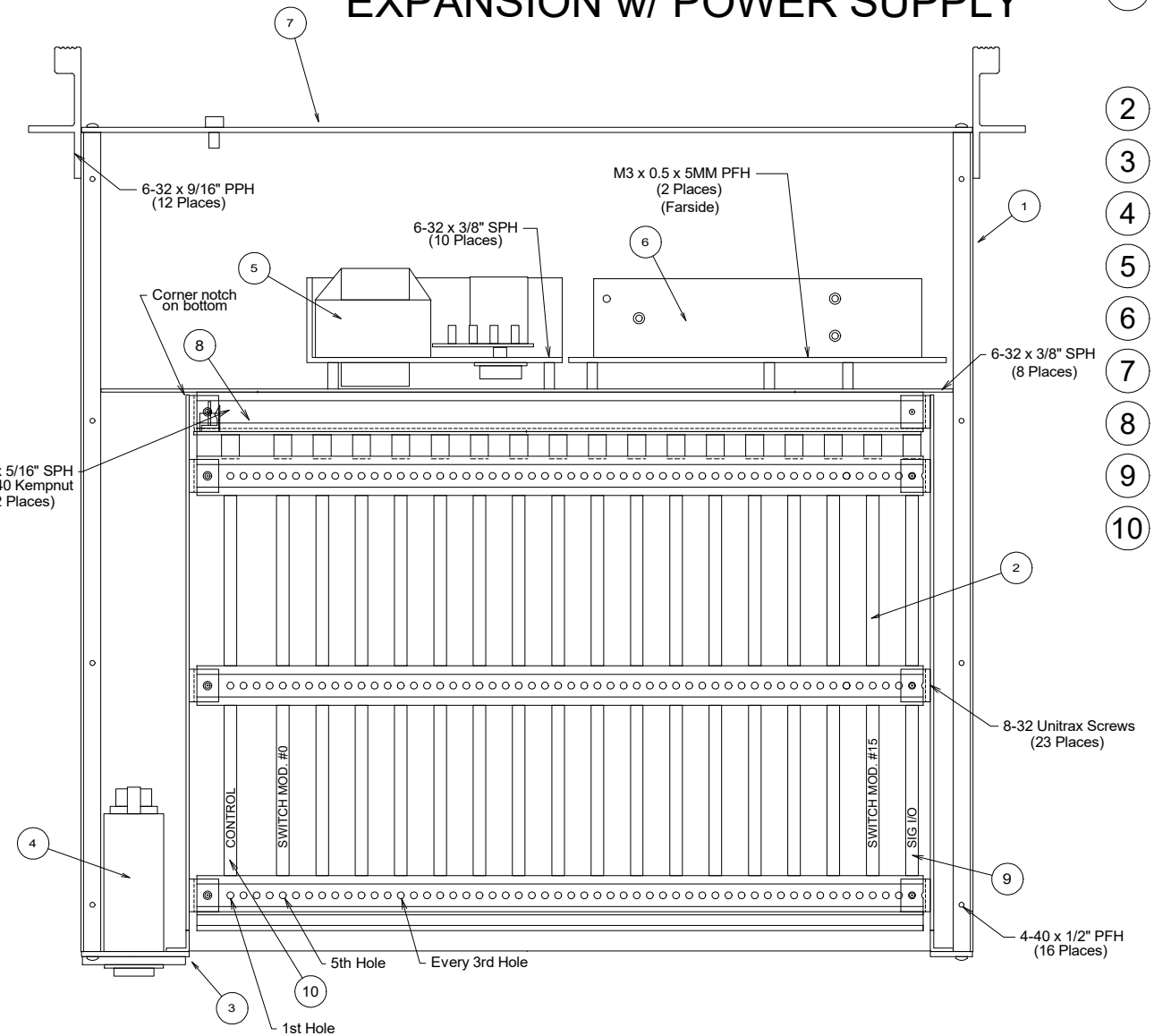
DATE: 7/6/95 | CAD FILE: 24-04-00.DW2 | DRAWN BY: JAR

SIZE: D | TITLE: JX/256 & RSS/256 E - P.S. ASSY

COMMENTS:

DRAWING NUMBER: 24-04-00

EXPANSION w/ POWER SUPPLY



1. Chassis Assembly
   Fabrication BOM: 24-20-140
   Assembly BOM: 24-20-240
2. 24-01-20 JX/256 Cage Assy
3. 24-02-21 Rear Panel
4. AC Selectable Power Entry
5. 5V-3A Power Supply
6. 12V-8.5A LS100-12 P.S.
7. Front Panel w/ Power LED
8. 24-17-10 High Power Backplane
9. 4-058 SIG I/O
10. 4-041-1 Control Interface Mod.
    Per A.P.D.- Omit L2 Ferrite

Cytec P/N	Description	Qty
FAS1008	4-40 x 5/16" Slotted Panhead	12
FAS1013	4-40 Kempnut	12
FAS1006	4-40 x 1/2" Phillips Flathead	16
FAS1004	6-32 x 3/8" Slotted Panhead	18
FAS1007	6-32 x 9/16" Phillips Panhead	12
FAS1058	8/32 Unitrax Screw	23
FAS1149	M3 x 0.5 x 5MM Phillips Flathead	2

A	10/16/23	PV	Changed 12v to LS Supply
-	7/19/13	DR	Drawing Start

CYTEC CORP.

TOL. +/- .010" U.O.S.

DATE:	CAD FILE:	DRAWN BY:
7/19/13	24-20-00 Rev A.dcd	DR

SIZE: D

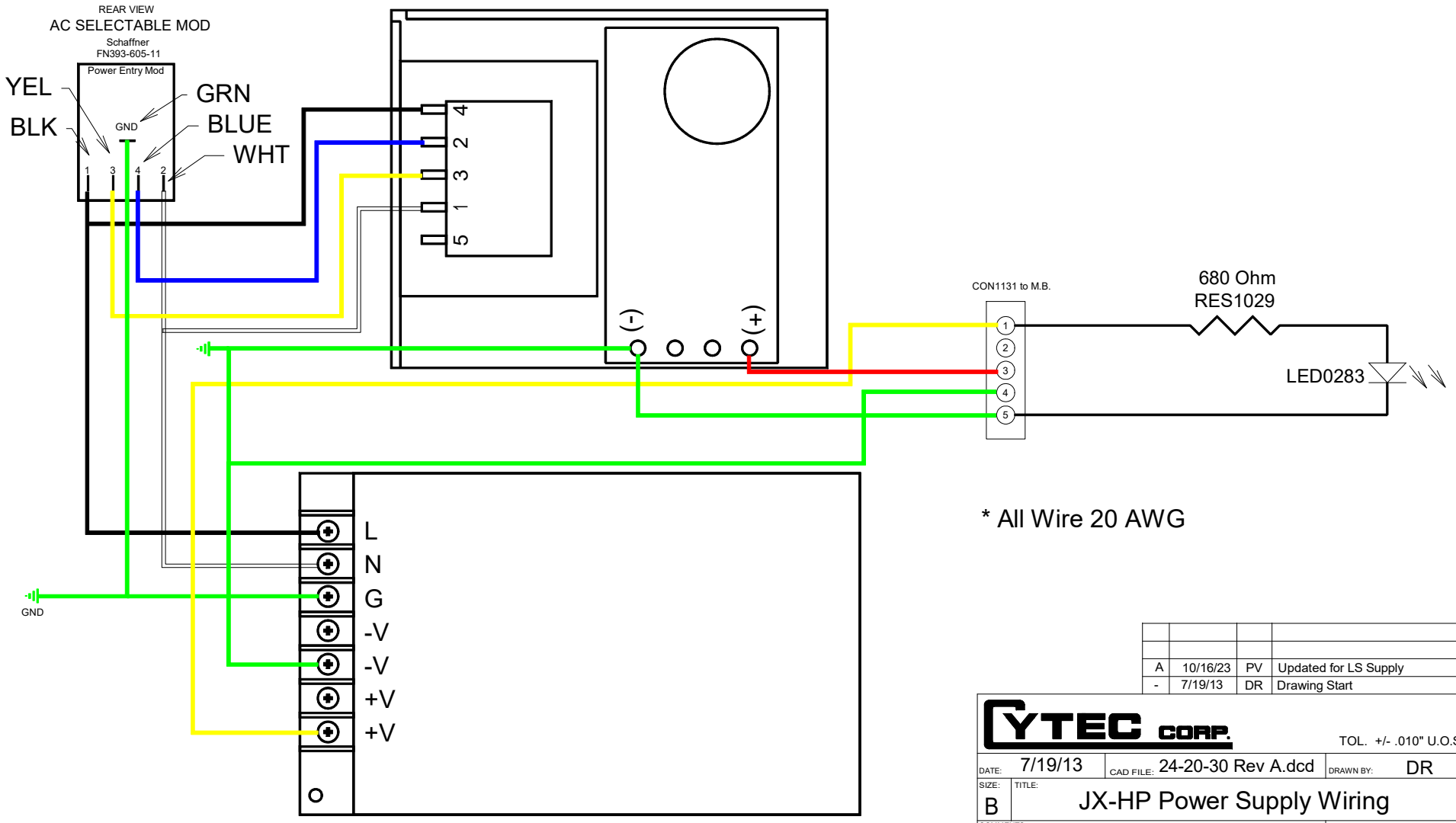
TITLE: JX/256 EXPANSION HIGH POWER ASSY

COMMENTS:

DRAWING NUMBER: 24-20-00

AC POWER SUPPLY WIRING

Standard



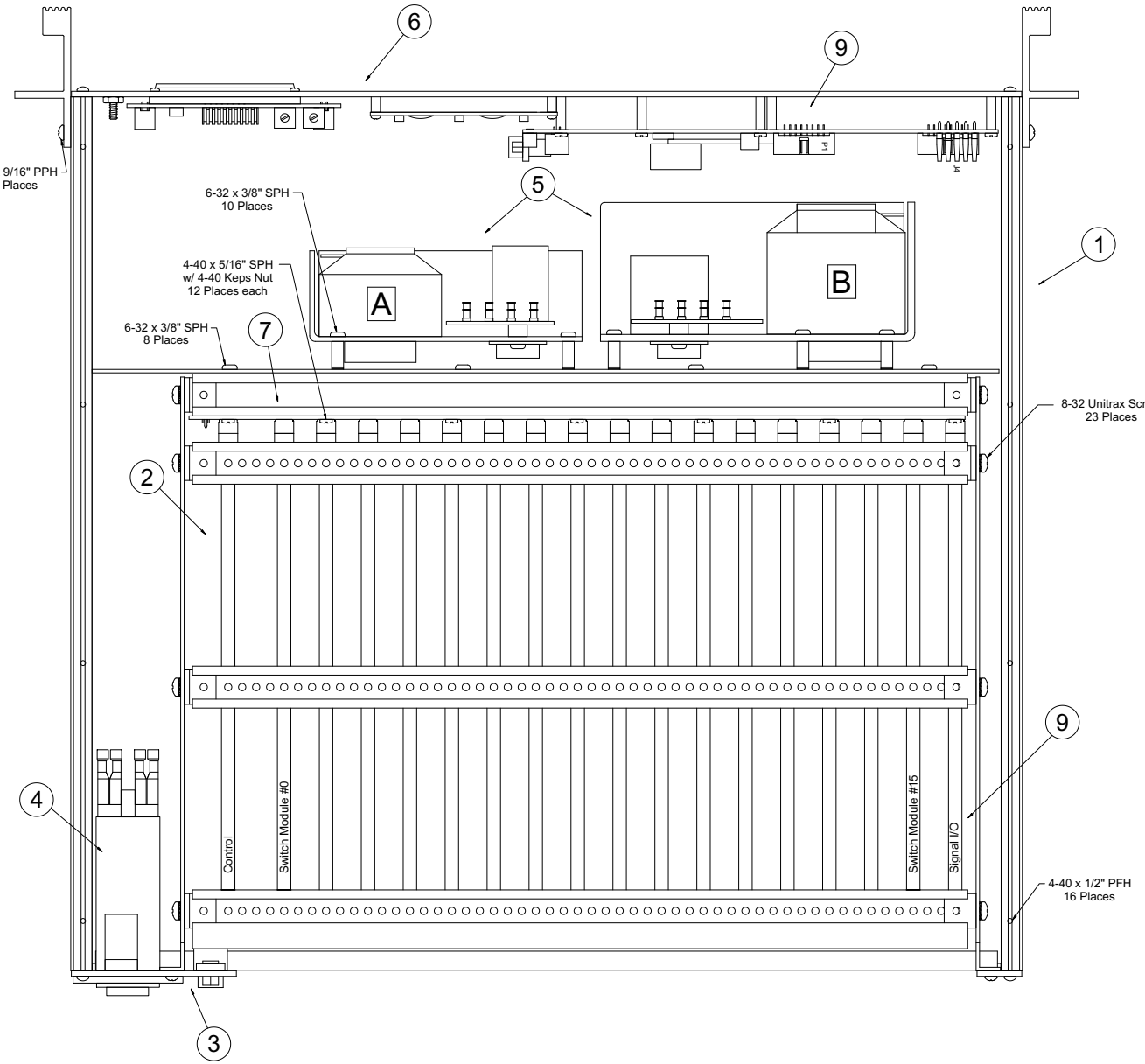
* All Wire 20 AWG

A	10/16/23	PV	Updated for LS Supply
-	7/19/13	DR	Drawing Start

CYTEC CORP.

TOL. +/- .010" U.O.S.

DATE: 7/19/13	CAD FILE: 24-20-30 Rev A.dcd	DRAWN BY: DR
SIZE: B	TITLE: JX-HP Power Supply Wiring	
COMMENTS:		DRAWING NUMBER: 24-20-30



1. Chassis Assembly
   Fabrication BOM: 24-21-140
   Assembly BOM: 24-21-240
2. JX/256 Cage Assembly (24-01-20)
3. JX Rear Panel (24-21-21)
4. Power Entry Module (SWX1000)
5. Power Supplies:
   - A 5V - 3A (PWS0293)
   - B 12V - 3.4A (PWS0292)
6. Front Panel as Ordered:
   No Manual Control (24-21-22)
   Manual Control (24-22-22) Shown
7. JX/256 Motherboard Module (24-08-10)
8. JX Signal I/O Module (4-058)
9. Control Interface Module

Cytec P/N	Description	Qty
FAS1008	4-40 x 5/16" Slotted Panhead	12
FAS1013	4-40 Keps Nut	12
FAS1006	4-40 x 1/2" Phillips Flathead	16
FAS1004	6-32 x 3/8" Slotted Panhead	18
FAS1007	6-32 x 9/16" Phillips Panhead	12
FAS1058	8-32 Unitrax Screw	23

A	4/9/19	PV	Added New Interfact Mod 24-26-10
-	8/19/14	DR	Drawing Start

CYTEC CORP

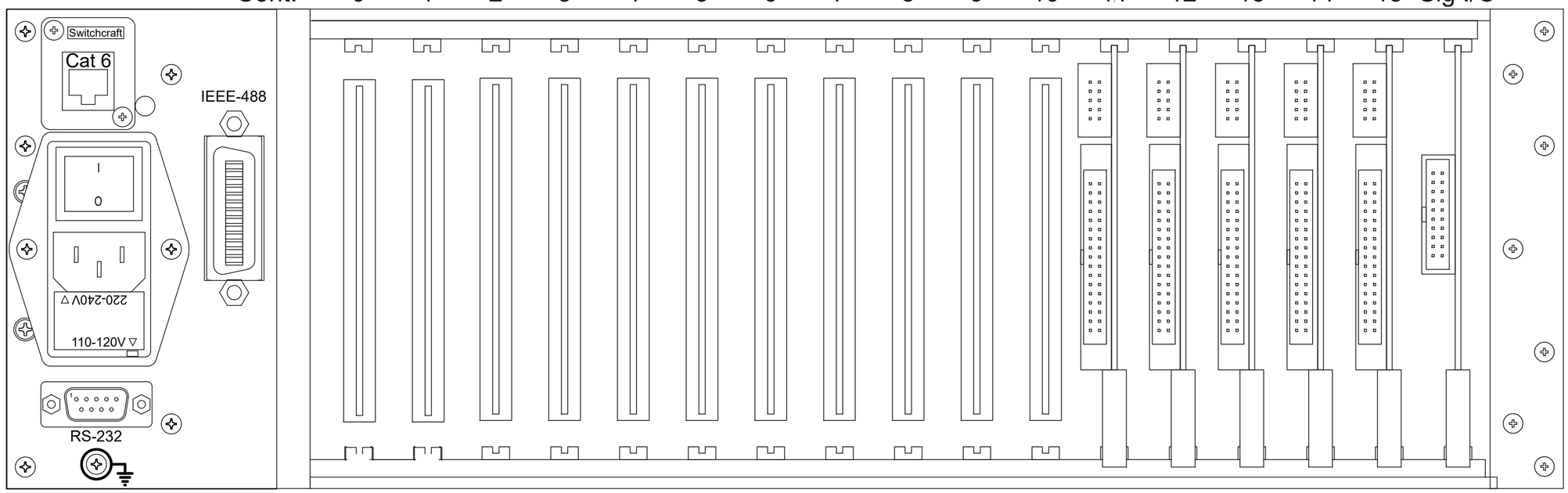
TOL. +/- 0.01" U.O.S.

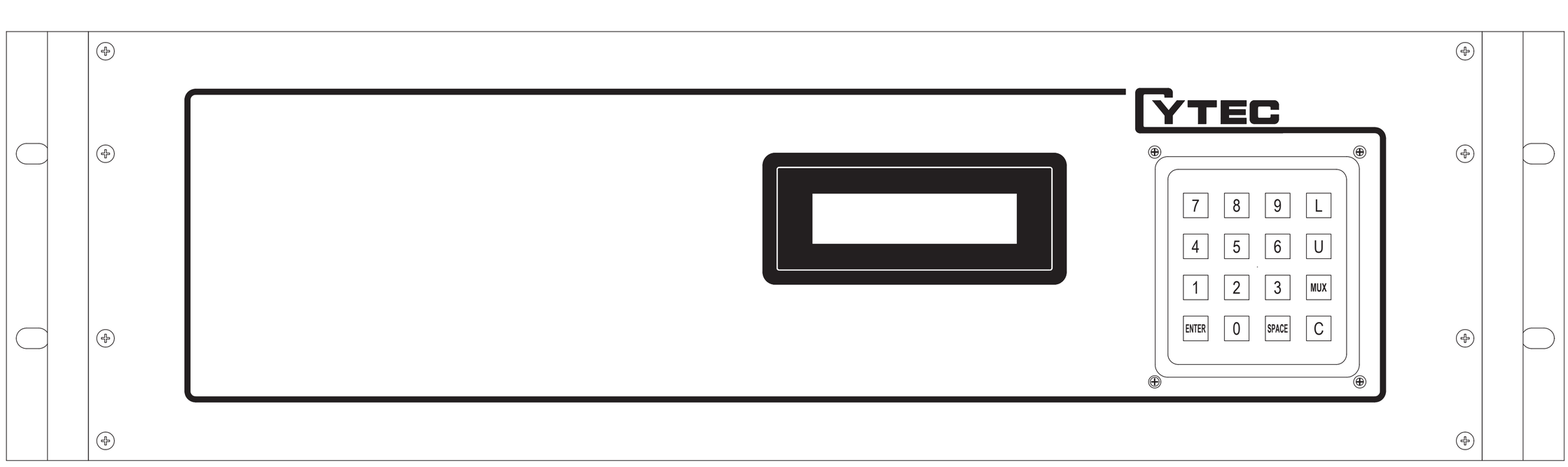
Date: 8/19/14 | CAD File: 24-21-00 Rev A.dxf | Drawn By: DR

Size: A | Title: JX/256-MF w/ IF-9 to IF-12

Comments:

Drawing Number: 24-21-00

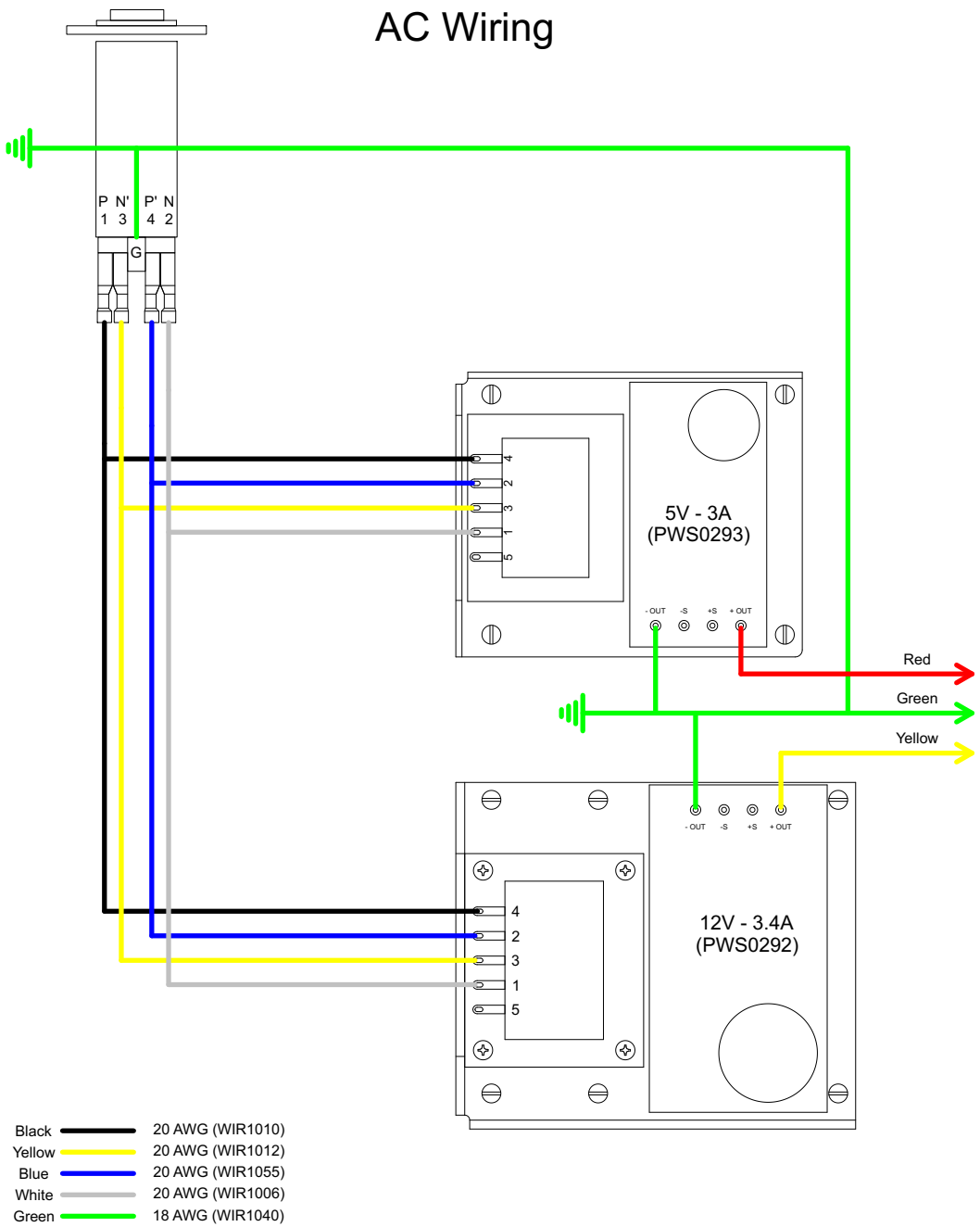




YTEC CORP.		TOL. +/- .010" U.O.S.
DATE: 5/25/25	CAD FILE: 24-21-91.ai	DRAWN BY: FHC
SIZE: D	TITLE: JX/256-MF- Front and Rear - IF-12	
COMMENTS:		DRAWING NUMBER: 24-21-91

Standard A/C Selectable
Schaffner P/N: FN393-605-11
SWX1000

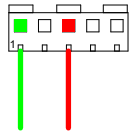
AC Wiring



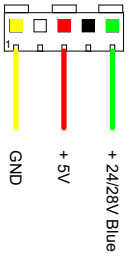
Black	20 AWG	(WIR1010)
Yellow	20 AWG	(WIR1012)
Blue	20 AWG	(WIR1055)
White	20 AWG	(WIR1006)
Green	18 AWG	(WIR1040)

DC Wiring

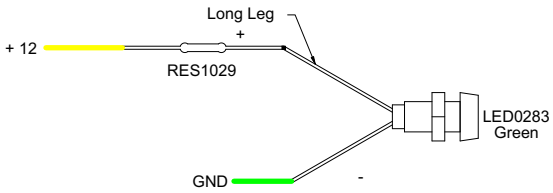
Control Module
CON1131
J3



24-08-10
JX/256 Motherboard
CON1131
J2



Front Panel LED Wiring



-	8/19/14	DR	Drawing Start

YTEC CORP

TOL. +/- 0.01" U.O.S.

Date: 8/19/14	CAD File: 24-21-30.dxf	Drawn By: DR

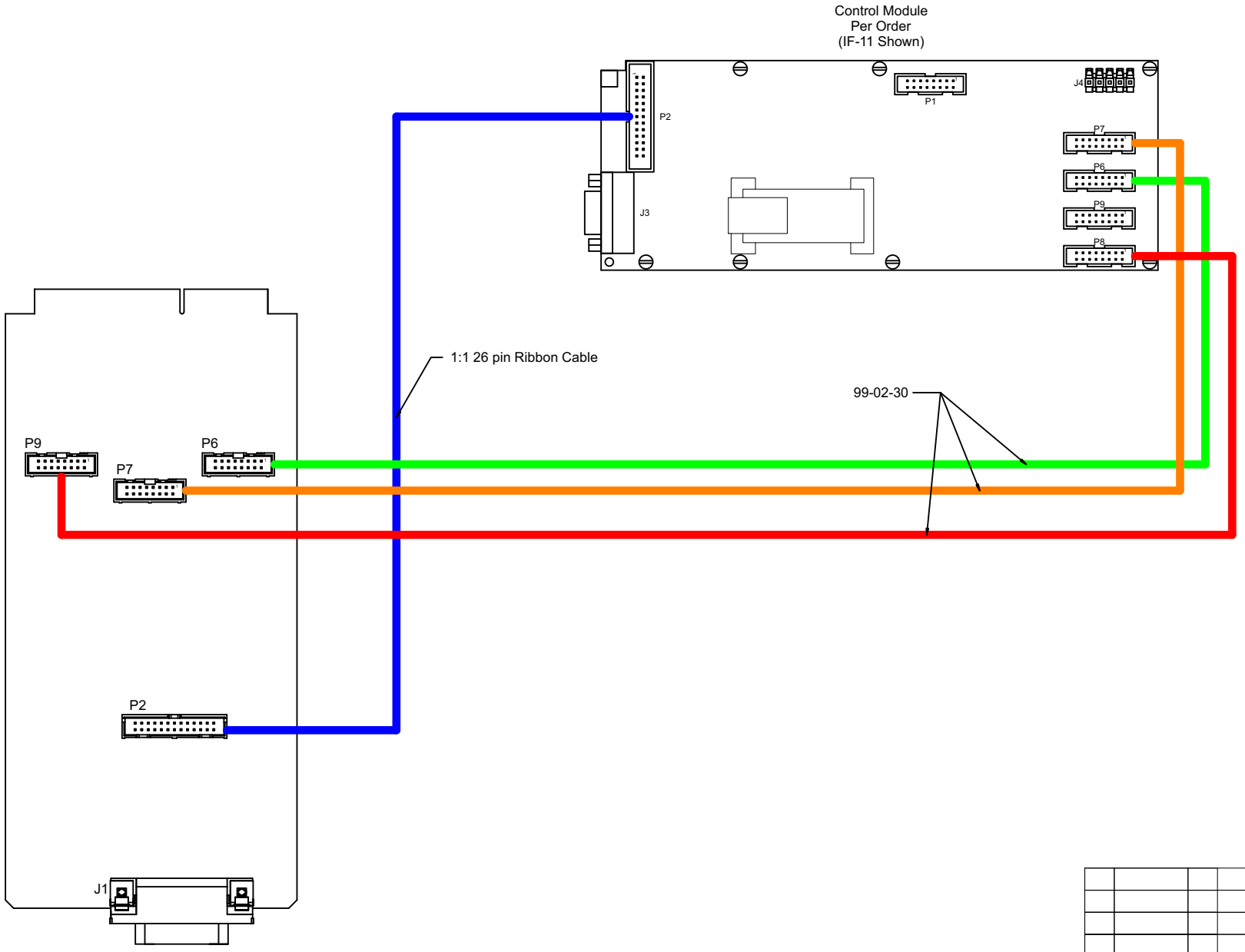
Size: A

Title: JX Power Supply Wiring

Comments:

Drawing Number: 24-21-30

Deburred		Initial	



24-26-10
JX mainframe Adapter Module

-	4/9/19	PV	Drawing Start

YTEC CORP

TOL. +/- 0.01" U.O.S.

Date: 4/9/19	CAD File: 24-21-31.dxf	Drawn By: PV

Size: A

Title: JX/256 Control Wiring Diagram

Comments:

Drawing Number: 24-21-31