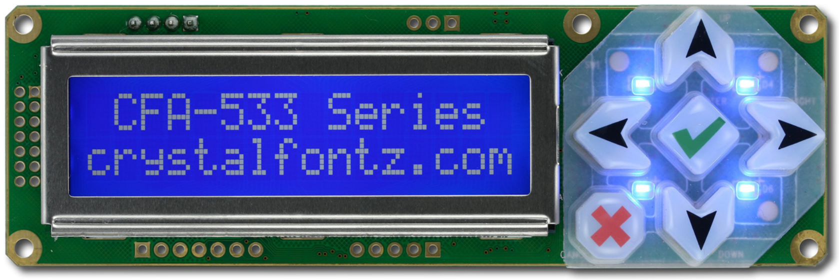
Crystalfontz America, Incorporated

SERIAL
LCD MODULE DATASHEET



Datasheet Release Date 2016-09-30
for the
[CFA533-TMI-KL]

Hardware Version v1.4, Firmware Version s1v2

Crystalfontz America, Incorporated

12412 East Saltese Avenue
Spokane Valley, WA 99216-0357

Phone:	888-206-9720
Fax:	509-892-1203
Email:	techinfo@crystalfontz.com
URL:	www.crystalfontz.com

CONTENTS

FORWARD - - - 5
Revision Information - - - 5
Notices - - - 5
INTRODUCTION - - - 6
CFA533 Interface Choices - - - 6
Similar Display Modules - - - 6
Main Features - - - 6
Display Module Classification Information - - - 7
MECHANICAL SPECIFICATIONS - - - 8
Physical Characteristics - - - 8
Display Module Outline Drawings - - - 9
Keypad Detail Drawing - - - 11
Panel Mounting Application Cutout Drawing - - - 12
ELECTRICAL SPECIFICATIONS - - - 13
System Block Diagram For all CFA533 Series Displays - - - 13
Supply Voltages and Current - - - 14
Absolute Maximum Ratings - - - 14
DC Characteristics - - - 15
ESD (Electro-Static Discharge) Specifications - - - 15
OPTICAL CHARACTERISTICS - - - 16
Optical Specifications - - - 16
Test Conditions and Definitions for Optical Characteristics - - - 16
CONNECTION INFORMATION - - - 19
Jumpers That Can Be Modified - - - 19
Overview of Connection Information - - - 20
Standard (+5v) Power Supply Connection - - - 21
1. Connection through J_PWR Connector (Non-ATX) - - - 21
2. Connection through J8 Connector (Non-ATX) - - - 22
3. Connection through J_RS232 Connector (Non-ATX) - - - 23
ATX Power Supply and Control Connection for Host Power Sense - - - 23
ATX Power Supply Connection - - - 23
ATX Control Connections for Host Power Sense through J_PWR or J8 Connector - - - 24
ATX Keypad Control - - - 27
Data Communications: Details for RS232 Connections - - - 28
GPIO Connections - - - 29
1-Wire (DOW) Device Connections for Optional Accessories - - - 30
Temperature Sensors - - - 30
Other 1-Wire Devices - - - 30
HOST COMMUNICATIONS - - - 31
Packet Structure - - - 31
About Handshaking - - - 33
Report Codes - - - 34
0x80: Key Activity - - - 34
0x82: Temperature Sensor Report - - - 34

CONTENTS, CONTINUED

Command Codes - - - 35
0 (0x00): Ping Command - - - 35
1 (0x01): Get Hardware & Firmware Version - - - 35
2 (0x02): Write User Flash Area - - - 35
3 (0x03): Read User Flash Area - - - 36
4 (0x04): Store Current State as Boot State - - - 36
5 (0x05): Reboot CFA533, Reset Host, or Power Off Host (ATX Required) - - - 37
6 (0x06): Clear LCD Screen - - - 37
7 (0x07): Set LCD Contents, Line 1 - - - 38
8 (0x08): Set LCD Contents, Line 2 - - - 38
9 (0x09): Set LCD Special Character Data - - - 38
10 (0x0A): Read 8 Bytes of LCD Memory - - - 39
11 (0x0B): Set LCD Cursor Position - - - 39
12 (0x0C): Set LCD Cursor Style - - - 39
13 (0x0D): Set LCD Contrast - - - 40
14 (0x0E): Set LCD & Keypad Backlights - - - 40
18 (0x12): Read DWR-DOW-Y17 Temperature Sensors - - - 41
19 (0x13): Set Up Temperature Reporting - - - 42
20 (0x14): Arbitrary 1-Wire (DOW) Transaction - - - 43
21 (0x15): Set Up Live Temperature Display - - - 43
22 (0x16): Send Command Directly to the LCD Controller - - - 44
23 (0x17): Configure Key Reporting - - - 44
24 (0x18): Read Keypad, Polled Mode - - - 45
28 (0x1C): Set ATX Switch Functionality - - - 46
29 (0x1D): Enable/Feed Host Watchdog Reset - - - 48
30 (0x1E): Read Reporting/ATX/Watchdog (debug) - - - 49
31 (0x1F): Send Data to LCD - - - 49
33 (0x21): Set Baud Rate - - - 49
34 (0x22): Set/Configure GPIO - - - 50
35 (0x23): Read GPIO Pin Levels and Configuration State - - - 51
CHARACTER GENERATOR ROM (CGROM) - - - 53
DISPLAY MODULE RELIABILITY AND LONGEVITY - - - 54
Module Reliability - - - 54
Module Longevity (EOL / Replacement Policy) - - - 54
APPENDIX A: CALCULATING THE CRC - - - 55
APPENDIX B: CONNECTING A DS2450 1-WIRE QUAD A/D CONVERTER - - - 66

LIST OF FIGURES

Figure 1. CFA533-TMI-KL Display Module Outline Drawings (2 pages) - - - - - - - 9
Figure 2. Keypad Detail Drawing - - - - - - - 11
Figure 3. Panel Mount Application Cutout Drawing for Optional Bracket - - - - - - - 12
Figure 4. System Block Diagram - - - - - - - 13
Figure 5. Definition of Operation Voltage (V_{OP}) (Negative) - - - - - - - 17
Figure 6. Definition of Response Time (Tr, Tf) (Negative) - - - - - - - 17
Figure 7. Definition of 6:00 O'Clock and 12:00 O'Clock Viewing Angles - - - - - - - 18
Figure 8. Definition of Horizontal and Vertical Viewing Angles (CR>2) - - - - - - - 18
Figure 9. Connection through J_PWR Connector (Non-ATX) - - - - - - - 21
Figure 10. Connection through J8 Connector (Non-ATX) - - - - - - - 22
Figure 11. CFA533-TMI-KL Connection through J_RS232 Connector (Non-ATX) - - - - - - - 23
Figure 12. ATX Host Power Sense through +5v on J_PWR Connector - - - - - - - 24
Figure 13. ATX Power Supply and Control Using Crystalfontz WR-PWR-Y14 Cable - - - - - - - 25
Figure 14. ATX Host Power Sense through GPIO[1] on J8 Connector - - - - - - - 26
Figure 15. J_RS232 Default and Alternate Pin Assignments - - - - - - - 29
Figure 16. Location of GPIO Connections, Resistors, and J_DOW - - - - - - - 30
Figure 17. Character Generator ROM (CGROM) - - - - - - - 53

FORWARD

REVISION INFORMATION

Datasheet Revision History

Datasheet Release: 2016-09-30

- This datasheet was updated to reflect hardware version v1.4 and firmware revision v1.2. For details on the hardware and firmware changes, see Part Change Notifications (PCNs) under the Notices tab on the website page for this display module.
- Datasheet was updated to current standards. Tables, text, and illustrations were improved for readability.

NOTICES

About Variations

Slight variations (for example, contrast, color, or intensity) between lots are normal.

About Volatility

This display has nonvolatile memory.

Additional Fine Print

INTRODUCTION

CFA533 INTERFACE CHOICES

The CFA533 Intelligent LCD Modules are available with four interface choices. The host interface controls the LCD and reads the keypad.

CFA533-*-KC**	I^2C interface
CFA533-*-KL**	Logic level serial interlace
CFA533-*-KS**	Serial interface with “full swing” RS232
CFA533-*-KU**	USB Interface

This datasheet has information for the CFA533-TMI-KL only. Datasheet files for the other interfaces listed above are on the *Datasheets & Files* tab for the web pages of each part number.

SIMILAR DISPLAY MODULES

The CFA533 uses the same command structure as our other Intelligent LCD Modules, including the CFA631, CFA633, CFA635, CFA735. and CFA835 series.

The CFA533 series is mechanically similar to the CFA633 series (available with serial or USB interfaces). The CFA533 series command set is compatible with the CFA633 series. The CFA533 can be used as an economical “drop-in” replacement for most CFA633 series applications that do not need fan capabilities.

The CFA533 series does not have CE certification because it is not an end product. The display module requires power and communications from another system in order to operate. If you need a CE approved display module, please consider our XES635 USB series.

MAIN FEATURES

- ❑ 16 characters x 2 lines LCD with keypad and high-level interface. With the optional drive bay bracket, the display will fit nicely in a 1U rack mount case (35 mm overall height).
- ❑ Only a single supply is needed. Wide power supply voltage range (V_{DD} = +3.3v to +5.0v) is perfect for embedded systems.
- ❑ Backlight and contrast are fully voltage compensated over the power supply range. No adjustments to the contrast setting or backlight brightness are needed.
- ❑ Bidirectional 19200 / 115200 baud logic-level asynchronous serial interface suitable to connect directly to micro-controller UART pins.
- ❑ Integrated LED backlit 6-button translucent silicon keypad with screened legend with the popular arrows, enter and cancel layout. Fully decoded keypad: any key combination is valid and unique.
- ❑ Edge-lit white LED backlight with negative blue STN LCD. Displays light characters on blue background and blue LED backlit keypad.
- ❑ Negative mode display is readable in normally office lighting and dark areas. May be difficult to read in direct sunlight.
- ❑ Advanced digital GPIO control with PWM output.
- ❑ Robust packet-based communications protocol with 16-bit CRC.

- ❑ Non-volatile memory (EEPROM): Set the "power-on" display screen, plus 16-bytes for storing IP, netmask, or system serial number.
- ❑ These options can be added to your display after clicking on the red *Customize and Add to Cart* button:
  - ATX power supply control functionality allows the buttons on the CFA533 to replace the Power and Reset switches on your system, simplifying front panel design. The ATX functionality can also implement a hardware watchdog that can reset host system on host software failure.
  - Temperature monitoring: up to 32 channels at up to 0.5 degrees Celsius with absolute accuracy (using optional connector and Crystalfontz WR-DOW-Y17 cable with DOW sensor).
  - "Live Display" shows up to four temperature readings without host intervention, allowing temperatures to be shown immediately at boot, even before the host operating system is loaded.
  - 1-Wire (DOW) bridge functionality allows control of other 1-Wire compatible devices (ADC, voltage monitoring, current monitoring, RTC, GPIO, counters, identification/encryption). (Additional hardware required.)
- ❑ An optional 5.25-inch half-height drive bay kit with mounting bracket is also available and can be added to your order.
- ❑ Crystalfontz America, Incorporated is ISO 9001:2008 certified.
- ❑ A Declaration for Conformity, RoHS, and REACH:SVHC is available under the *Datasheets & Files* tab on display web pages.

DISPLAY MODULE CLASSIFICATION INFORMATION

CFA	533	-	T	M	I	-	K	L
❶	❷		❸	❹	❺		❻	❼

❶	**Brand**	Crystalfontz America, Inc.
❷	**Model Identifier**	533
❸	**Backlight Type & Color**	T – LED, white
❹	**Fluid Type, Image (positive or negative), & LCD Glass Color**	M – STN, negative blue
❺	**Polarizer Film Type, Temperature Range, & View Angle (O 'Clock)**	I – Transmissive, Temperature Range[1], 6:00
❻	**Special Code 1**	K – Manufacturer's code
❼	**Special Code 2**	L – Serial interface, inverted, logic level

[1]*Temperature Range is -20°C minimum to +70°C maximum*

MECHANICAL SPECIFICATIONS

PHYSICAL CHARACTERISTICS

ITEM	SIZE
Display Module Overall Dimensions	
Width and Height	110.50 (W) x 35.00 (H)
Depth with Keypad and RS232 Connector	25.60 mm nominal 25.90 mm maximum
Viewing Area	61.00 (W) x 15.80 (H) mm
Active Area	56.20 (W) x 11.50 (H) mm
Character Size	2.95 (W) x 5.55 (H) mm
Character Pitch	3.55 (W) x 5.95 (H) mm
Pixel Size	0.550 (W) x 0.60 (H) mm
Pixel Pitch	0.600 (W) x 0.700 (H) mm
Keystroke Travel (approximate)	2.4 mm
Weight	41 grams (typical)

DISPLAY MODULE OUTLINE DRAWINGS

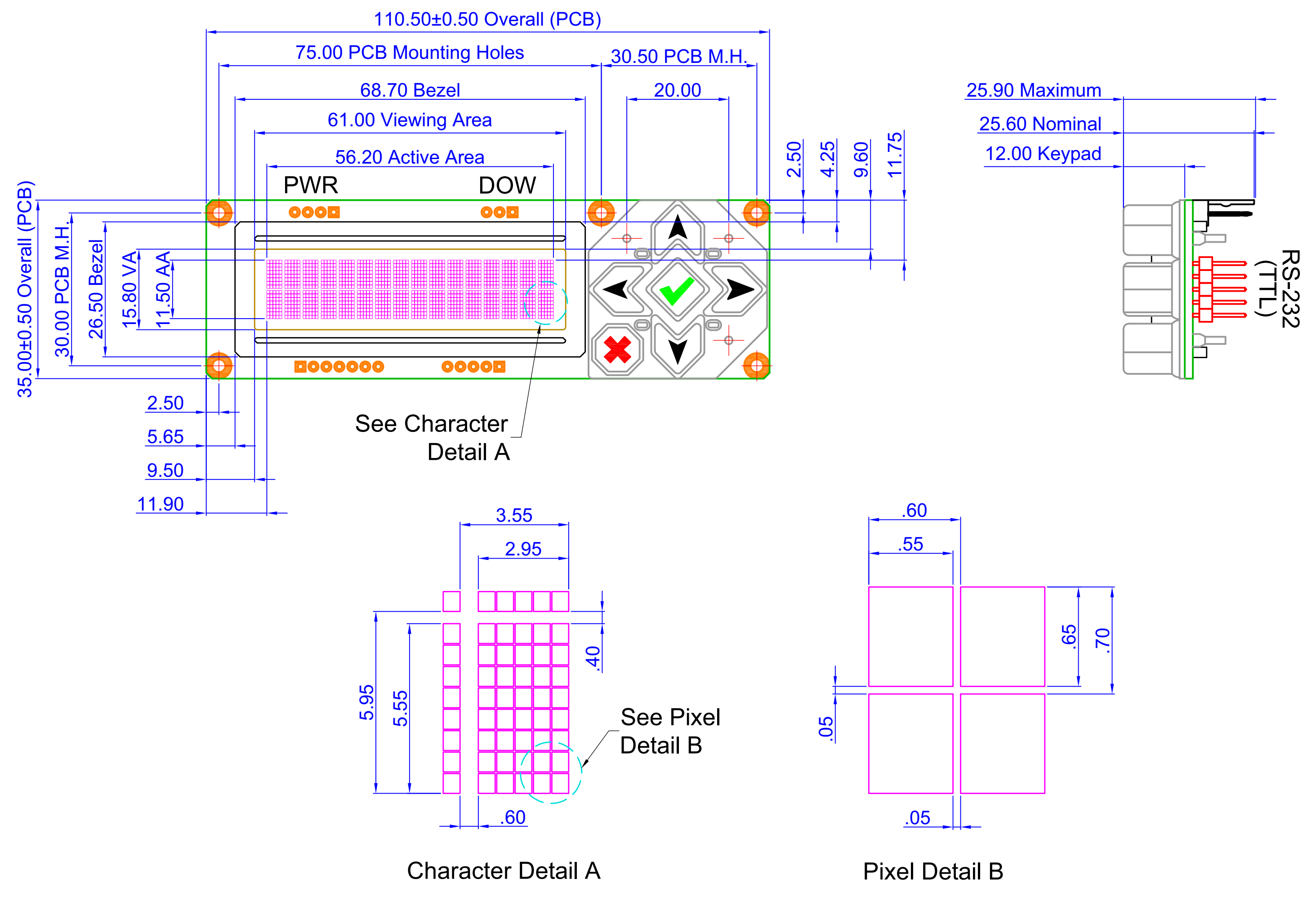
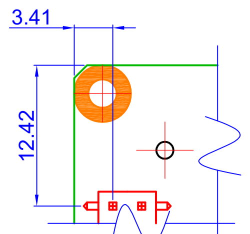


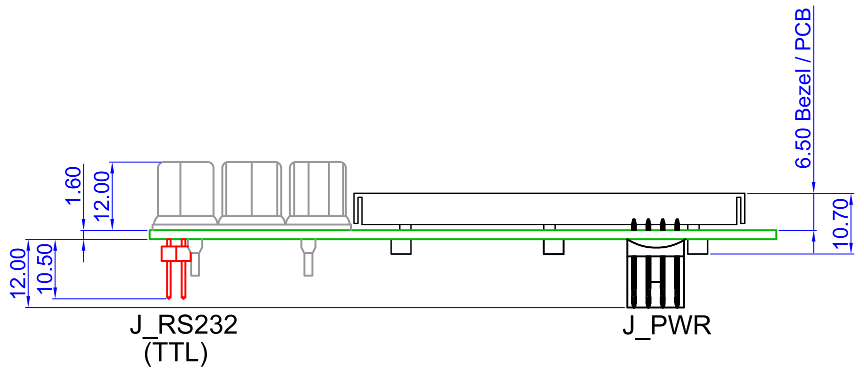
Figure 1. CFA533-TMI-KL Display Module Outline Drawings (2 pages)

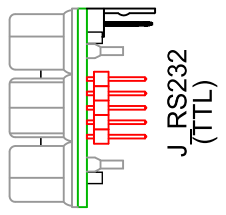
Illustration is deemed accurate but not guranteed.

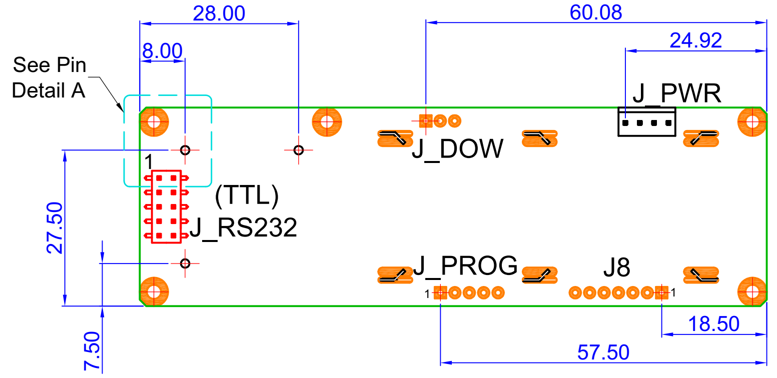
copyright © 2010 by Crystalfontz America, Inc. www.crystalfontz.com/products/	*Part No.(s):* CFA533-TFH-KL CFA533-TMI-KL CFA533-YYH-KL	*Scale:* Not to scale	*Drawing Number:* CFA533_master	*Hardware Rev.:* v1.4
		Units: Millimeters	*Date:* 2016-07-28	*Sheet:* 1 of 2



Pin Detail A





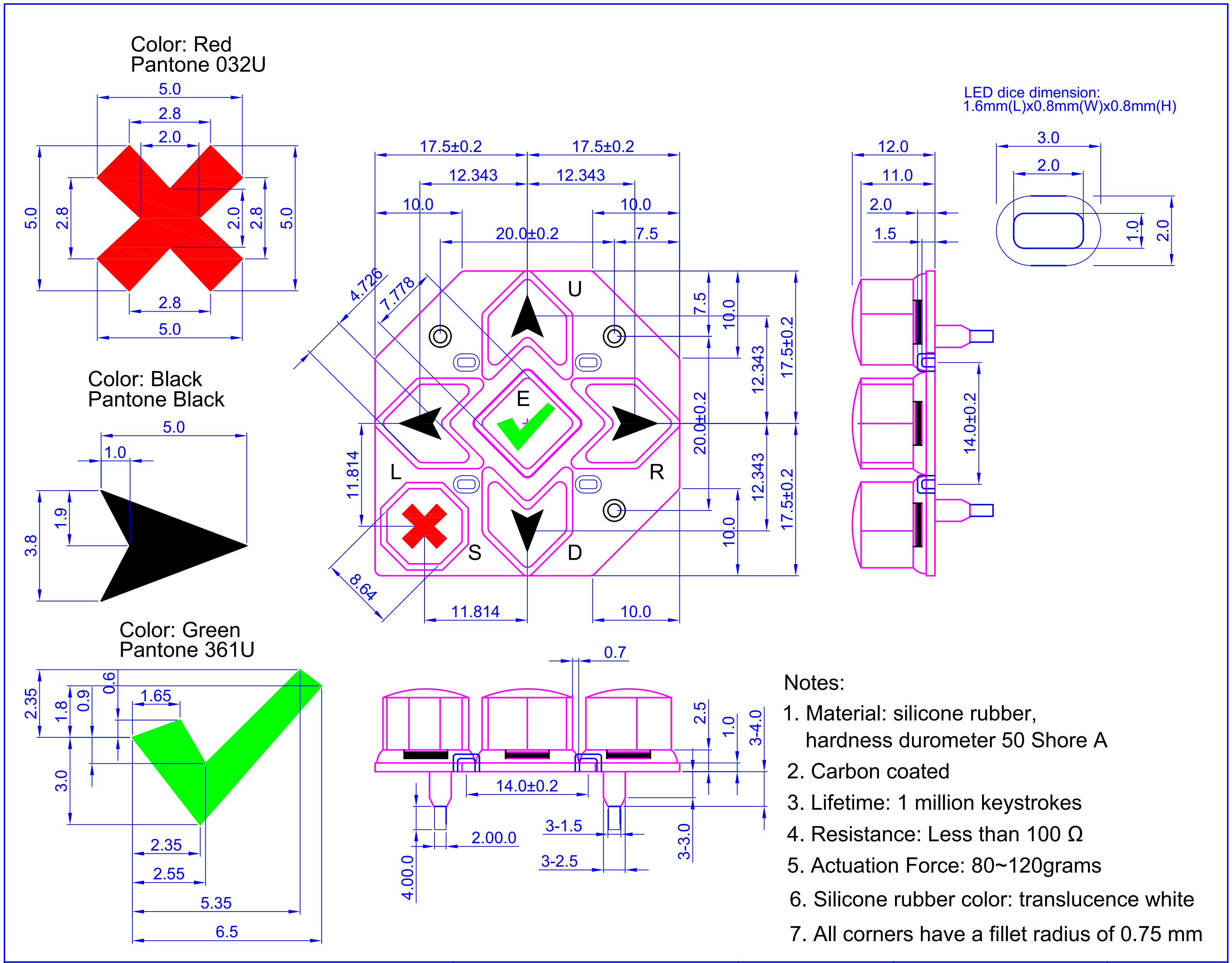


Back View

Illustration is deemed accurate but not guranteed..

Crystalfontz America, Inc. www.crystalfontz.com/products/	*Part No.(s):* CFA533-TFH-KL CFA533-TMI-KL CFA533-YYH-KL	*Scale:* Not to scale	*Drawing Number:* CFA533_master	*Hardware Rev.:* v1.4
		Units: Millimeters	*Date:* 2016-07-28	*Sheet:* 2 of 2

KEYPAD DETAIL DRAWING



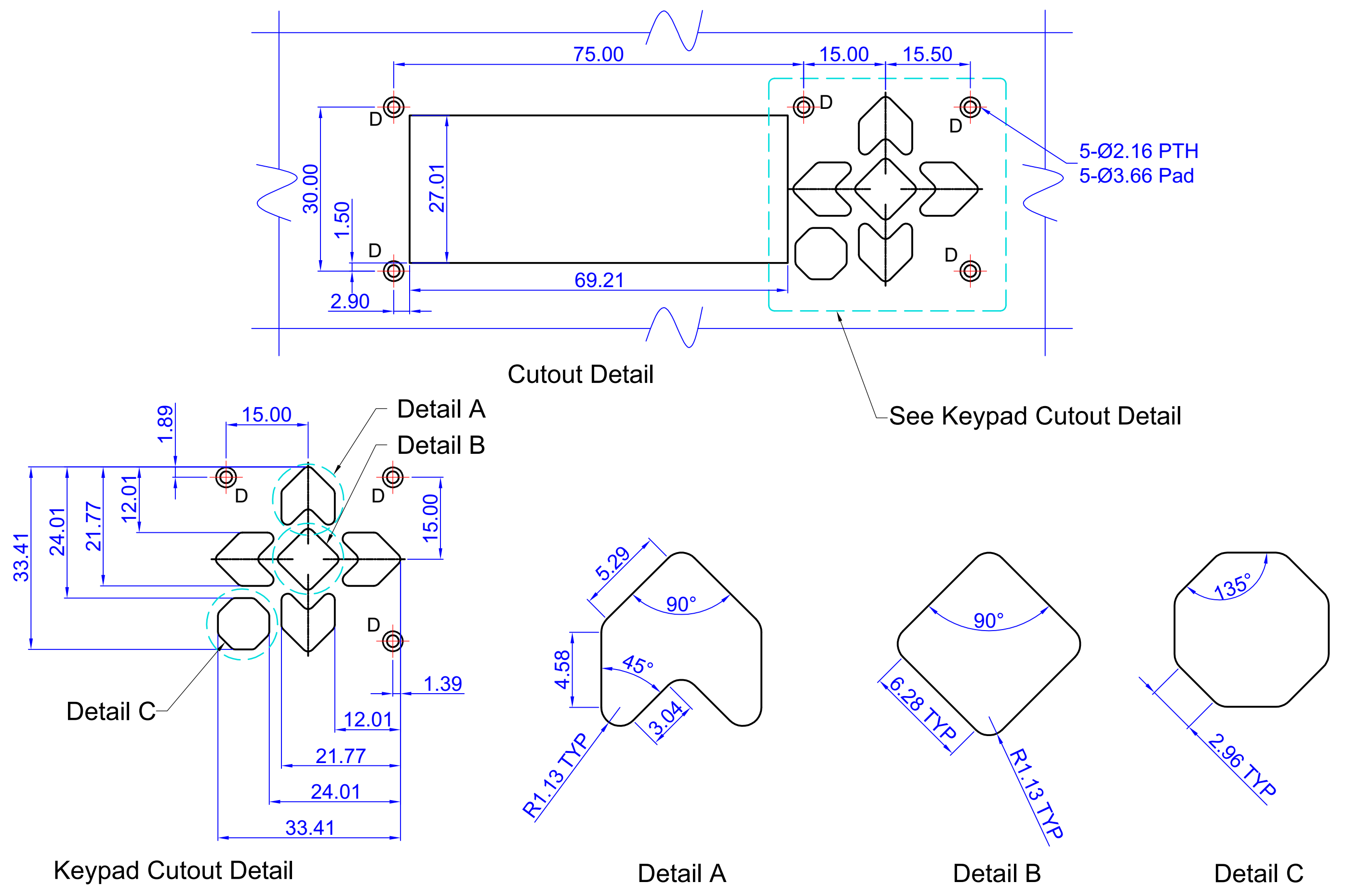
Notes:

1. Material: silicone rubber, hardness durometer 50 Shore A
2. Carbon coated
3. Lifetime: 1 million keystrokes
4. Resistance: Less than 100 Ω
5. Actuation Force: 80~120grams
6. Silicone rubber color: translucence white
7. All corners have a fillet radius of 0.75 mm

copyright © 2009 by **Crystalfontz America, Inc.** www.crystalfontz.com/products/	*Part No.(s):* CFA533 Series Keypad Detail	*Scale:* Not to scale	*Drawing Number:* CFA533_master	*Hardware Rev.:* v1.4
		Units: Millimeters	*Date:* 2016-07-28	*Sheet:* 1 of 1

Figure 2. Keypad Detail Drawing

PANEL MOUNTING APPLICATION CUTOUT DRAWING



Typical mounting hardware at locations "D" (5 places):

- PEM FH-256-8
- Bivar Inc. 9913-5 mm Spacer
- 2-56 "Small Profile" Hex Nut
- Use appropriate screen printed overlay to cover display bezel and mounting hardware, and to protect LCD from scratching. Sample fabrication drawings are available on request.

copyright © 2010 by Crystalfontz America, Inc. www.crystalfontz.com/products/	*Part No.(s):* CFA533 Panel Mounting Application Detail	*Scale:* Not to scale	*Drawing Number:* Panel_master	*Hardware Rev.:* v1.4
		Units: Millimeters	*Date:* 2011/07/25	*Sheet:* 1 of 1

Figure 3. Panel Mount Application Cutout Drawing for Optional Bracket

ELECTRICAL SPECIFICATIONS

SYSTEM BLOCK DIAGRAM FOR ALL CFA533 SERIES DISPLAYS

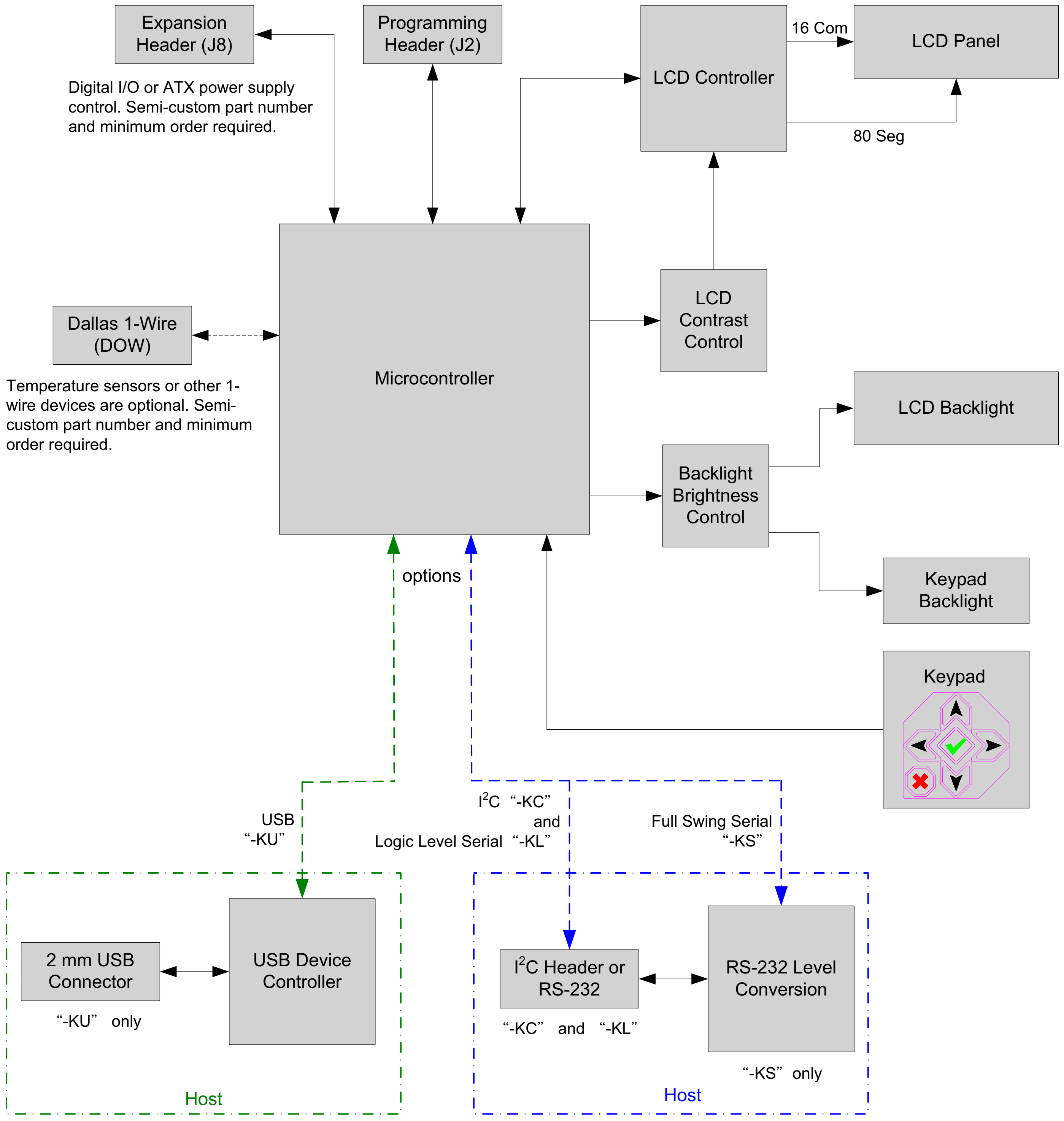


Figure 4. System Block Diagram

SUPPLY VOLTAGES AND CURRENT

GPIO CURRENT LIMITS	SPECIFICATION
Sink	TBD mA
Source	TBD mA

TYPICAL CURRENT CONSUMPTION - TMI	SPECIFICATION
+5v for logic (LCD + micro-controller)	<TBD mA
+5v for logic (LCD + micro-controller) + TMI backlight at full brightness	<TBD mA

ABSOLUTE MAXIMUM RATINGS

ABSOLUTE MAXIMUM RATINGS	SYMBOL	MINIMUM	MAXIMUM
Operating Temperature	T_{OP}	-20°C	+70°C
Storage Temperature	T_{ST}	-30°C	+80°C
Humidity Range (non-condensing)	RH	10%	90%

Note
Extended exposure to the absolute maximum ratings listed above may affect device reliability. Stresses beyond those listed above can cause permanent damage.

Background color changes slightly depending on ambient temperature. This phenomena is reversible.

DC CHARACTERISTICS

	DC CHARACTERISTICS	TEST CONDITIONS	SYMBOL	MINIMUM	TYPICAL	MAXIMUM
CONTROLLER AND BOARD	Supply Voltage for Logic	T_{OP} =-30°C to +70°C	V_{DD} - GND	+3.2v	+3.3v -+5.0v	+5.25v[1]
	Input High Voltage	VDD = +5v	V_{IH}	V_{DD}-1.0v		V_{DD}
	Input Low Voltage		V_{IL}	0v (GND)		+0.6v
	Output High Voltage		V_{OH}	+0.9$_{VDD}$		
	Output Low Voltage		V_{OL}	0v (GND)		+0.1V_{DD}

[1]*Do not exceed* +5.25v *maximum.*

ESD (ELECTRO-STATIC DISCHARGE) SPECIFICATIONS

The circuitry is industry standard CMOS logic and susceptible to ESD damage. Please use industry standard anti static precautions as you would for any other static sensitive devices such as expansion cards, motherboards, or integrated circuits. Ground your body, work surfaces, and equipment.

OPTICAL CHARACTERISTICS

Viewing Direction	6 o'clock

OPTICAL SPECIFICATIONS

ITEM	**SYMBOL**	**CONDITION**	**MINIMUM**	**TYPICAL**	**MAXIMUM**
Test Condition for all: T=25°					
Viewing Angle	Deg θ = 0°	(6 o'clock) CR≥2		45	
	Deg θ = 90°			30	
	Deg θ = 180°			25	
	Deg θ = 270°			30	
Contrast Ratio[1]	CR	θ=ψ= 0		10	15
LCD Response Time[2,3]	T rise			80 ms	160 ms
	T fall			100 ms	200 ms

[1]*Contrast Ratio = (brightness with pixels light)/(brightness with pixels dark).*
[2]*Response Time: The amount of time it takes a pixel to go from active to inactive or back again.*
[3]*For reference only.*

Changes in voltage can result in changes in contrast.

TEST CONDITIONS AND DEFINITIONS FOR OPTICAL CHARACTERISTICS

We work to continuously improve our products, including backlights that are brighter and last longer. Slight color variations from module to module and batch to batch are normal.

- Viewing Angle
  - Vertical (V)θ: 0°
  - Horizontal (H)φ: 0°
- Frame Frequency: 64 Hz
- Driving Waveform: 1/16 Duty, 1/5 Bias
- Ambient Temperature (Ta): 25°C

Definition of Operation Voltage (Vop)

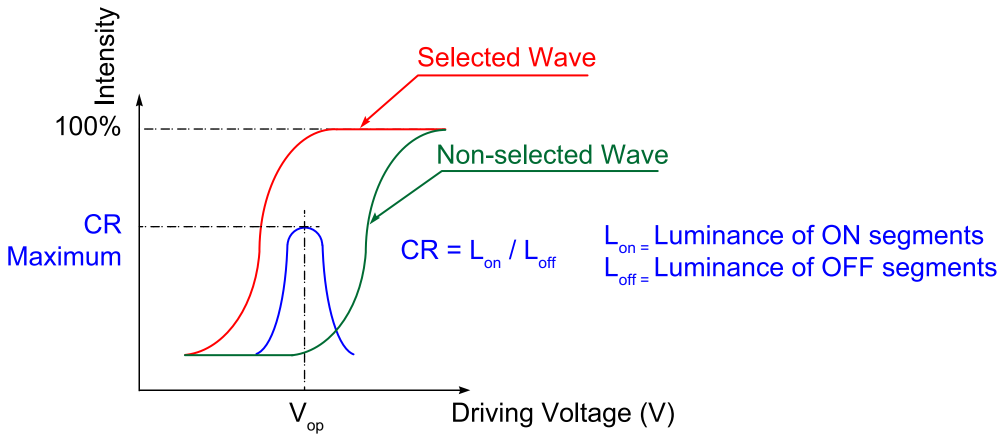


Figure 5. Definition of Operation Voltage (V_{OP}) (Negative)

Definition of Response Time (Tr, Tf)

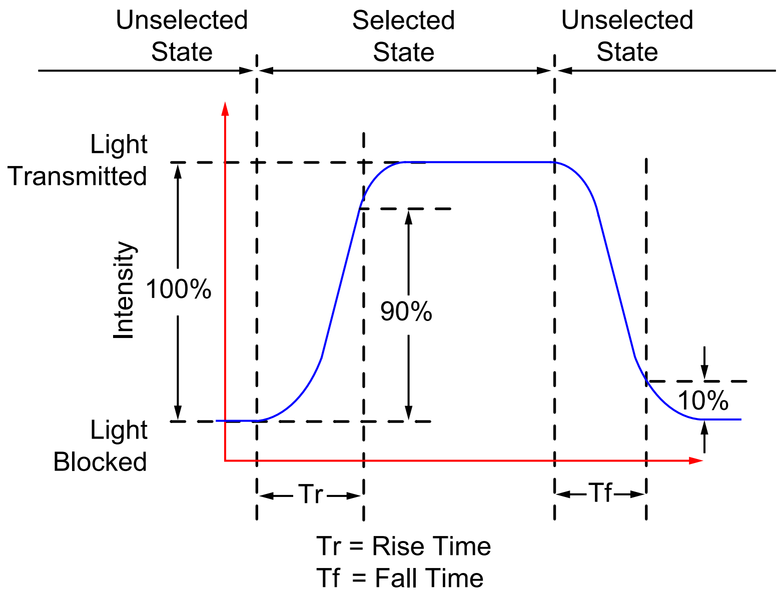


Figure 6. Definition of Response Time (Tr, Tf) (Negative)

Definition of 6 O'Clock and 12:00 O'Clock Viewing Angles

This display module has a 6:00 o'clock viewing angle.

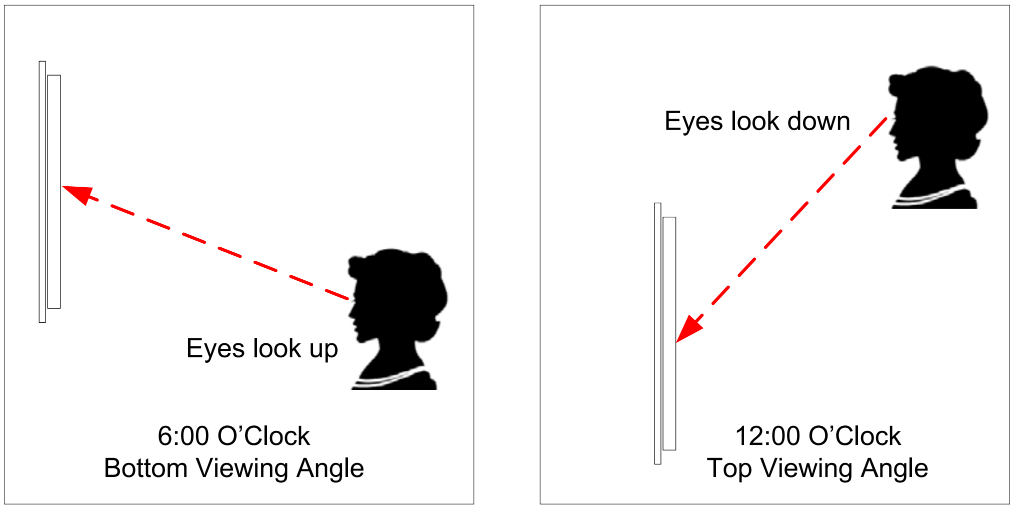


Figure 7. Definition of 6:00 O'Clock and 12:00 O'Clock Viewing Angles

Definition of Vertical and Horizontal Viewing Angles (CR≥2)

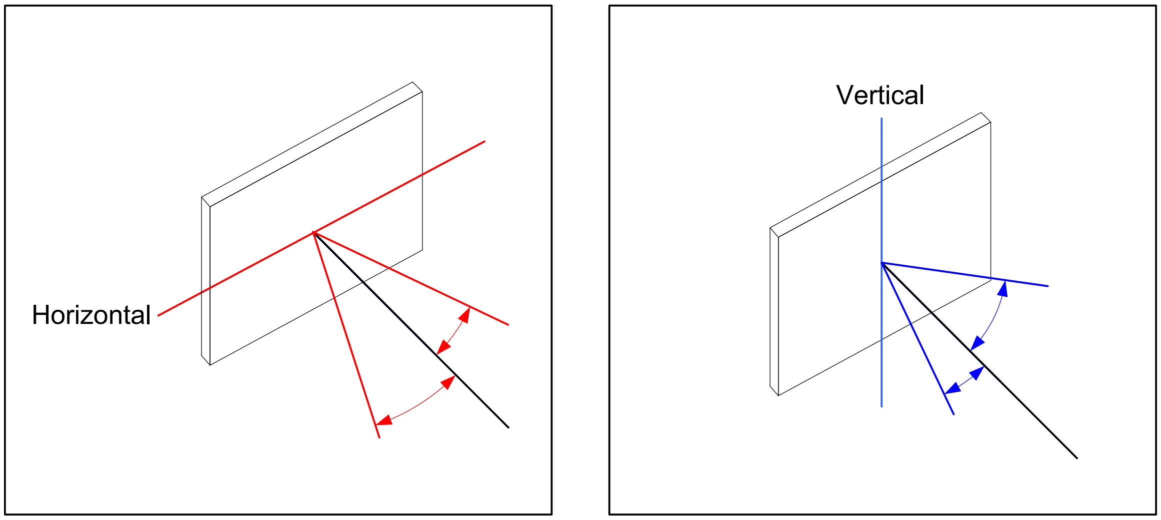
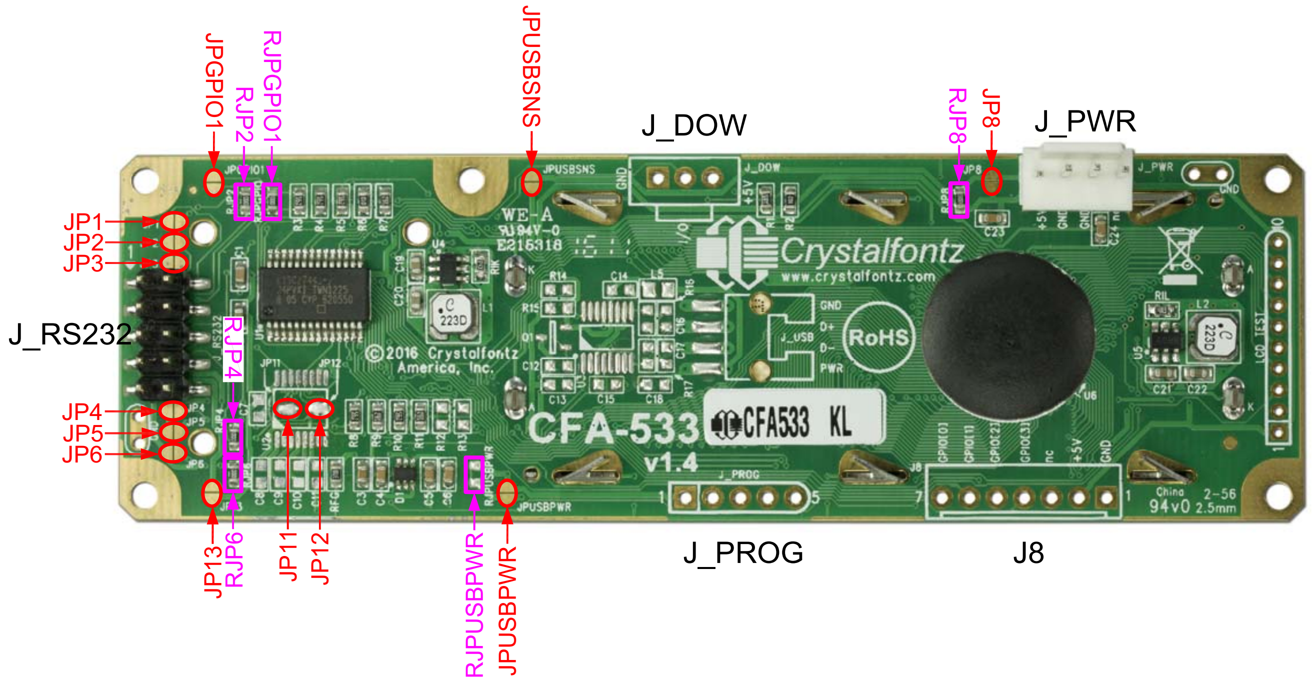


Figure 8. Definition of Horizontal and Vertical Viewing Angles (CR>2)

CONNECTION INFORMATION

JUMPERS THAT CAN BE MODIFIED

The CFA533 series has thirteen jumpers, shown in red below. Not all jumpers are used by all interfaces. Instead of a solder blob, jumpers are closed at the factory with a parallel resistor, labeled in magenta below. If you need to open these jumpers, remove the accompanying resistor. If you want to close any open or close a jumper without a resistor, open by using solder wick or close by melting a ball of solder across its gap.



Jumper Locations and Functions

JUMPER	FUNCTION	-KL
JP1	Alternate RS232 Configuration. See *Figure 15.*.	Open
JP2	Standard RS232 Configuration. See *Figure 15.*.	Closed (0Ω RJP2)
JP3	Alternate RS232 Configuration. See *Figure 15.*.	Open
JP4	Standard RS232 Configuration. See *Figure 15.*.	Closed (0Ω RJP4)
JP5	Alternate RS232 Configuration. See *Figure 15.*.	Open
JP6	Standard RS232 Configuration. See *Figure 15.*.	Closed (0Ω RJP6)
JP8	Connects the display's +5v to +5v on J_PWR. Conflicts with JPUSBSENSE.*	Closed (0Ω RJP8)
JP11	Connects the microprocessor's Serial Tx line to JP1 and JP2.	Closed
JP12	Connects the microprocessor's Serial Rx line to JP3 and JP4.	Closed
JP13	Connects the display's +5v to Pin 4 on J_RS232.	Open

JUMPER	FUNCTION	-KL
JPUSBPWR	Connects the display's +5v to PWR on J_USB.	Open
JPUSBSENSE	Connects the display's ATX SENSE to PWR on J_USB. Conflicts with JP8.*	Open (N/A)
JPGPIO1	When soldered closed or RJPGIO is loaded, bypasses R3. R3 is a 5.6KΩ resistor in series with GPIO1.	Closed (0Ω RJGPIO)
**JP8* and *JPUSBSENSE* both connect to ATX SENSE on the processor. Closing both of these jumpers is not recommended.		

OVERVIEW OF CONNECTION INFORMATION

This section describes your choices of methods to connect power and host power sense to the display module. The section also describes connections for optional accessories.

The host power supply can power the CFA533-TMI-KL in one of two ways:

1. *Standard:* This is the basic method to supply power to the module (“non-ATX”).
2. *ATX:* This method supplies power to the module and has power on, power off, and reset functionality to the host.

For your convenience, here are links to the connection descriptions:

Standard (+5v) Power Supply Connection (Pg. 21)
1. Connection through J_PWR Connector (Non-ATX) (Pg. 21)
2. Connection through J8 Connector (Non-ATX) (Pg. 22)
3. Connection through J_RS232 Connector (Non-ATX) (Pg. 23)

ATX Power Supply and Control Connection for Host Power Sense (Pg. 23)
1. ATX Host Power Sense through +5v on J_PWR Connector (Pg. 24)
2. ATX Host Power Sense through GPIO[1] on J8 Connector (Pg. 26)

ATX Keypad Control (Pg. 27)

Data Communications: Details for RS232 Connections (Pg. 28)

GPIO Connections (Pg. 29)

1-Wire (DOW) Device Connections for Optional Accessories (Pg. 30)
Temperature Sensors (Pg. 30)
Other 1-Wire Devices (Pg. 30)

In the sections listed above, we describe which jumpers, if any, must be opened or closed for the different connection methods. A helpful reference is Jumpers That Can Be Modified (Pg. 19). The table lists the open/close defaults for all jumpers.

STANDARD (+5V) POWER SUPPLY CONNECTION

For a standard power connection from your host to the CFA533-TMI-KL, choose one of the three methods described below.

1. Standard (+5v) Connection through J_PWR Connector
2. Standard (+5v) Connection through J8 Connector
3. Standard (+5v) Connection through J_RS232 Connector

1. Connection through J_PWR Connector (Non-ATX)

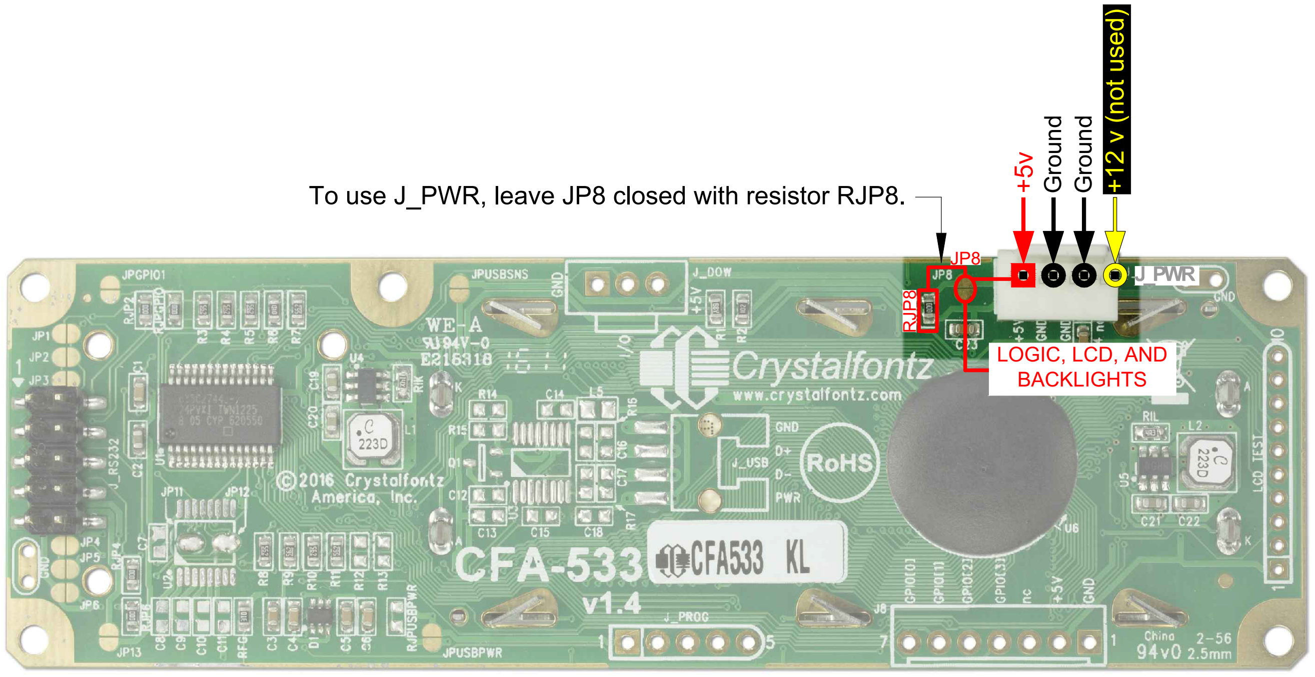

To use J_PWR, leave JP8 closed with resistor RJP8.
+5v
Ground
Ground
+12 v (not used)
JP8
RJP8
J_PWR
LOGIC, LCD, AND BACKLIGHTS

Figure 9. Connection through J_PWR Connector (Non-ATX)

1. JP8 is closed with the RJP8 0KΩ resistor by default. Leave JP8 closed so that the J8 connector is electrically connected to J_PWR.
2. You will need to supply +5v to pin 1 and Ground to pin 2 or 3 on the J_PWR connector. Use the Crystalfontz cable WR-PWR-Y12 (or equivalent) to connect from the host's 4-pin power supply cable connector to the CFA533-TMI-KL's J_PWR connector, You can add the WR-PWR-Y12 cable to your order after you click on the *Customize and Add to Cart* button on the display's web page.

2. Connection through J8 Connector (Non-ATX)

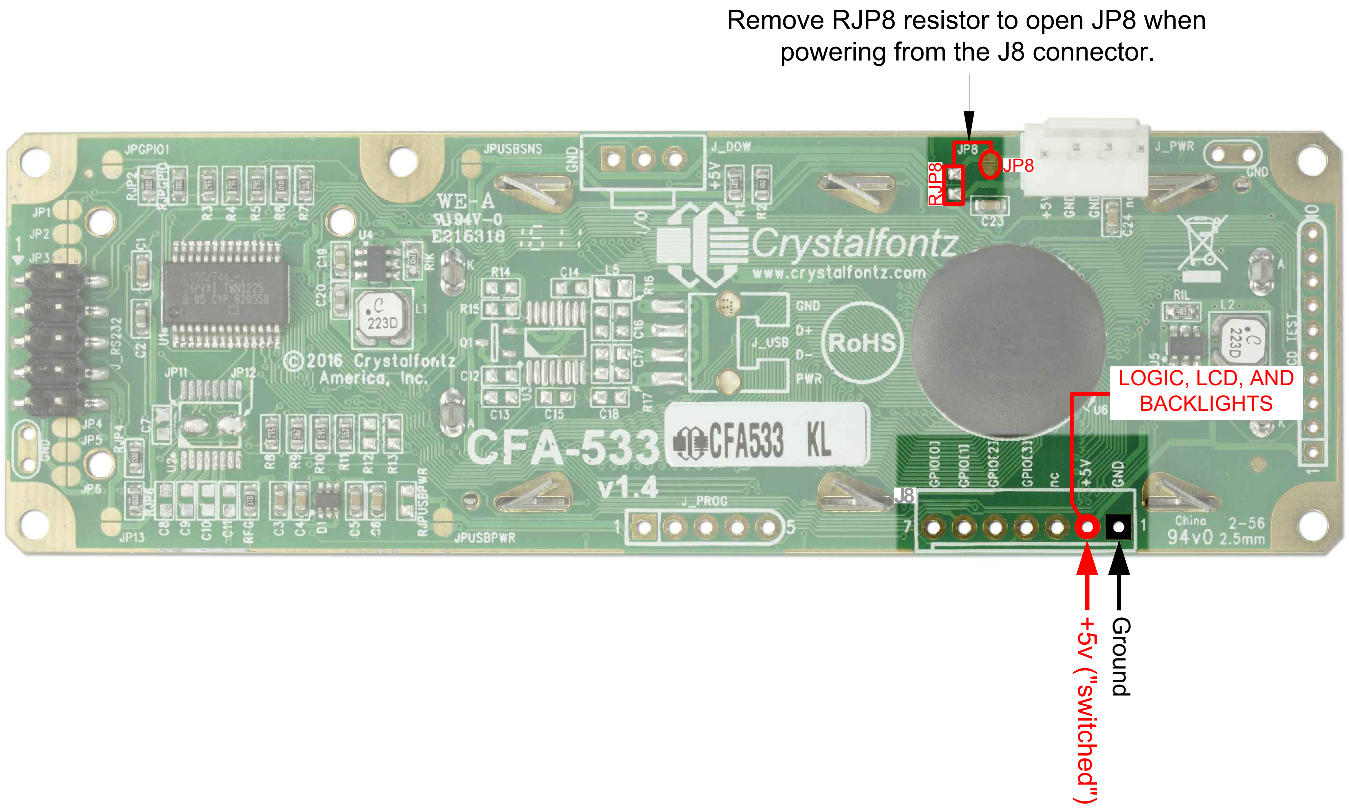

Remove RJP8 resistor to open JP8 when powering from the J8 connector.
RJP8
JP8
LOGIC, LCD, AND BACKLIGHTS
J8
+5v ("switched")
Ground

Figure 10. Connection through J8 Connector (Non-ATX)

1. ***Open JP8 by removing the RJP8 resistor when powering from the J8 connector.*** (JP8 is closed by default. JP8 should remain closed only when powering from J_PWR.)
2. You will need to supply +5v to pin 2 and Ground to pin 1 on the J8 header that you add.

Modification by Crystalfontz

Crystalfontz can configure the display modules so they will be ready to use in your application without modification. We will open JP8, and add a J8 header. You can choose this configuration after you click on the *Customize and Add to Cart* button on the display's web page. You will also be offered suitable cables.

3. Connection through J_RS232 Connector (Non-ATX)

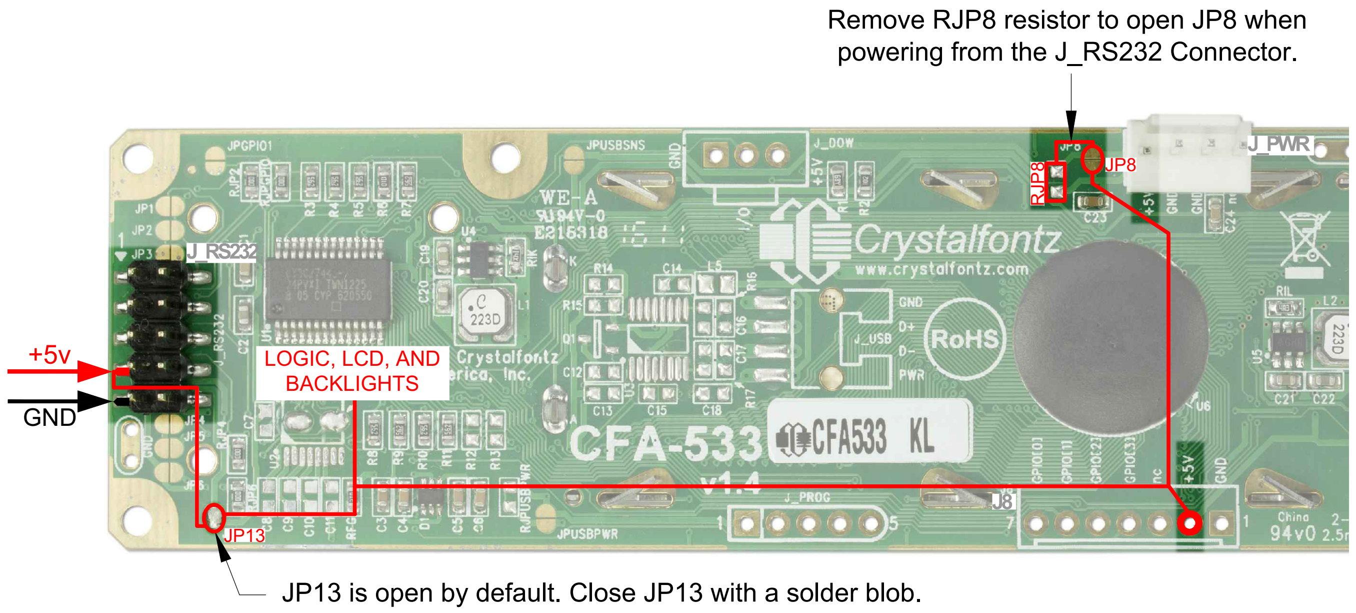

Remove RJP8 resistor to open JP8 when powering from the J_RS232 Connector.
J_PWR
JP8
RJP8
J_RS232
+5v
GND
LOGIC, LCD, AND BACKLIGHTS
JP13
JP13 is open by default. Close JP13 with a solder blob.

Figure 11. CFA533-TMI-KL Connection through J_RS232 Connector (Non-ATX)

The +5v power can be supplied through connector J_RS232, allowing a single cable to contain both power and data connections.

1. ***JP13 is open by default. Close JP13 with a solder blob.***
2. JP8 is closed by default. ***Open JP8 by removing the RJP8 resistor*** when powering from the J_RS232 connector. (JP8 should remain closed only when powering from J_PWR.)
3. If the "Default RS-232 Pin Assignments" (see *Figure 15. on Pg. 29*) are selected, the five connections needed to operate the display module are all on a single column of pins on J_RS232. You can connect a single 0.1-inch spacing 5-conductor cable to connect between the CFA533-TMI-KL and your host.

Customized Parts: Modification by Crystalfontz

Crystalfontz can configure the display modules so they will be ready to use in your application without modification. We will close JP13 and open JP8. For information, please contact technical support (+1-888-206-9720 or email techinfo@crystalfontz.com). We will provide you with a semi-custom part number and pricing. A minimum order quantity may apply.

ATX POWER SUPPLY AND CONTROL CONNECTION FOR HOST POWER SENSE

ATX Power Supply Connection

The CFA533-TMI-KL has the ability to control power on/off and reset functions of an ATX power supply. For this functionality, the CFA533-TMI-KL is powered from the host's V_{SB} signal (V_{SB} is the standby power which is always-on +5v ATX power supply output).

Note

The GPIO pins used for ATX control must not be configured as user GPIO. If ATX Host Power Sense to display module is being used, do not reconfigure the GPIO pins.

ATX Control Connections for Host Power Sense through J_PWR or J8 Connector

For ATX control, choose one of these two connection methods described below.

1. ATX Host Power Sense through +5v on J_PWR Connector
2. ATX Host Power Sense through GPIO[1] on J8 Connector

1. ATX Host Power Sense through +5v on J_PWR Connector

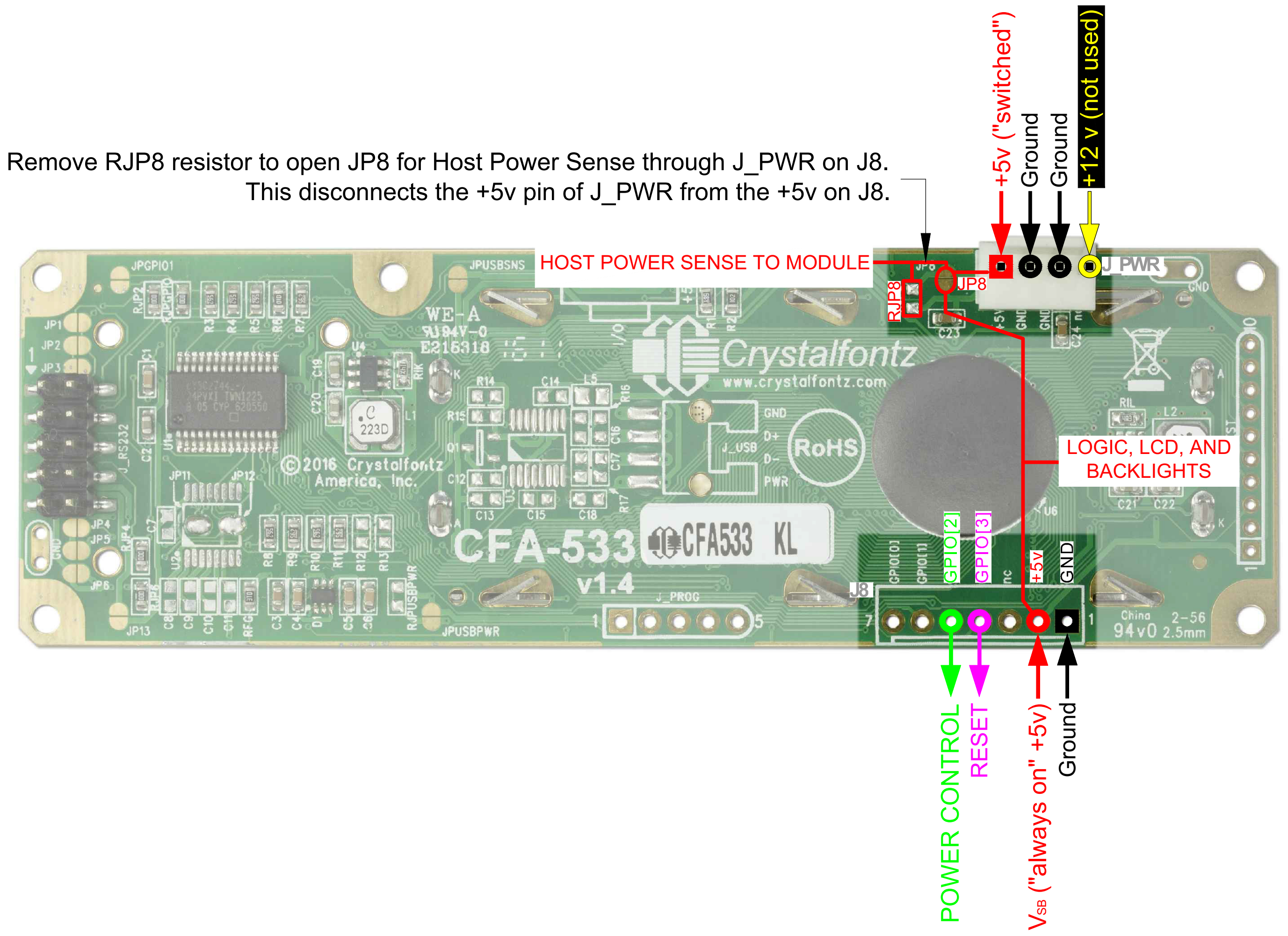

Remove RJP8 resistor to open JP8 for Host Power Sense through J_PWR on J8.
This disconnects the +5v pin of J_PWR from the +5v on J8.
+5v ("switched")
Ground
Ground
+12 v (not used)
HOST POWER SENSE TO MODULE
RJP8
JP8
J_PWR
LOGIC, LCD, AND BACKLIGHTS
GPIO[2]
GPIO[3]
+5v
GND
J8
POWER CONTROL
RESET
VSB ("always on" +5v)
Ground

Figure 12. ATX Host Power Sense through +5v on J_PWR Connector

By default, the pin labeled +5v on the CFA533-TMI-KL's J_PWR connector is electrically connected to the +5v pin on the J8 connector through the normally closed JP8. ***If you want to use the CFA533-TMI-KL to do ATX power supply control, open jumper JP8 by removing the RJP8 resistor.*** This will disconnect the +5v pin of the J_PWR connector from the +5v of the J8 connector. The +5v pin of the J_PWR connector will then function as the "Host Power Sense". The +5v pin of the J8 connector will function as V_{SB} power to the display module.

The motherboard's power switch input is connected to Pin 5 (labeled as GPIO2) of the CFA533-TMI-KL's connector J8 ~~(labeled as GPIO[2])~~. This pin functions as POWER CONTROL. The POWER CONTROL pin is configured as a high-impedance input until the display module wants to turn the host on or off, then it will change momentarily to low impedance output, driving either low or high depending on the setting of POWER_INVERT. (See command 28 (0x1C): Set ATX Switch Functionality (Pg. 46).)

The motherboard's reset switch input is connected to Pin 4 (labeled as GPIO3) of the CFA533-TMI-KL connector J8 ~~(labeled as GPIO[3])~~. This pin functions as RESET. The RESET pin is configured as a high-impedance input until the display module wants to reset the host. Then it will change momentarily to low impedance output, driving either low or high.

The optional Crystalfontz WR-PWR-Y14 or WR-PWR-Y44 cables simplify ATX power supply control connections. JP8 from connector J_PWR is closed by default. When using this cable, open jumper JP8 in order to ensure correct operation.

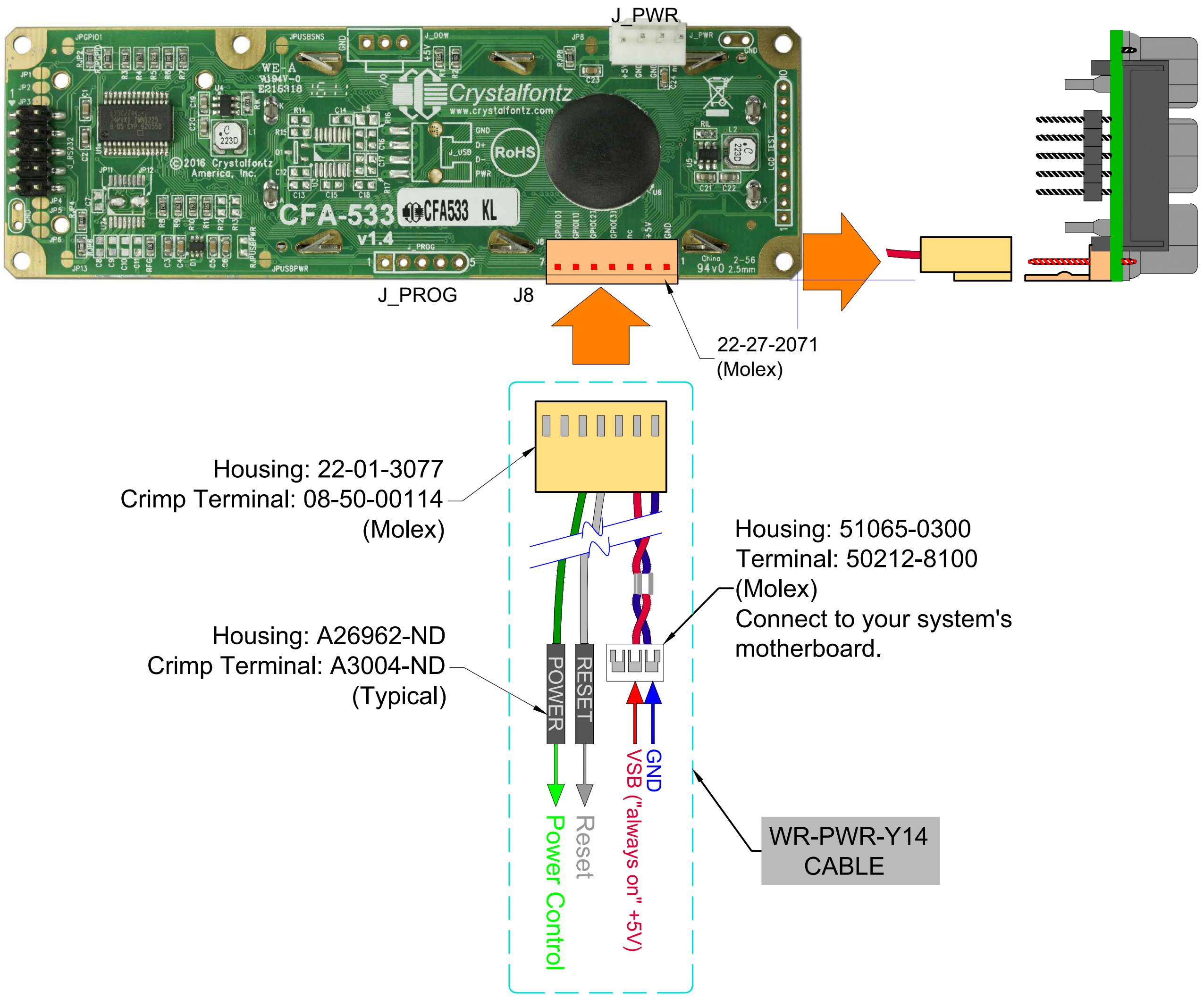


Figure 13. ATX Power Supply and Control Using Crystalfontz WR-PWR-Y14 Cable

Customized Parts: Modification by Crystalfontz

Other modifications are available as custom parts to suit your product requirements. For information, please contact technical support (+1-888-206-9720 or email techinfo@crystalfontz.com). We will provide you with a semi-custom part number and pricing. A minimum order quantity may apply.

2. ATX Host Power Sense through GPIO[1] on J8 Connector

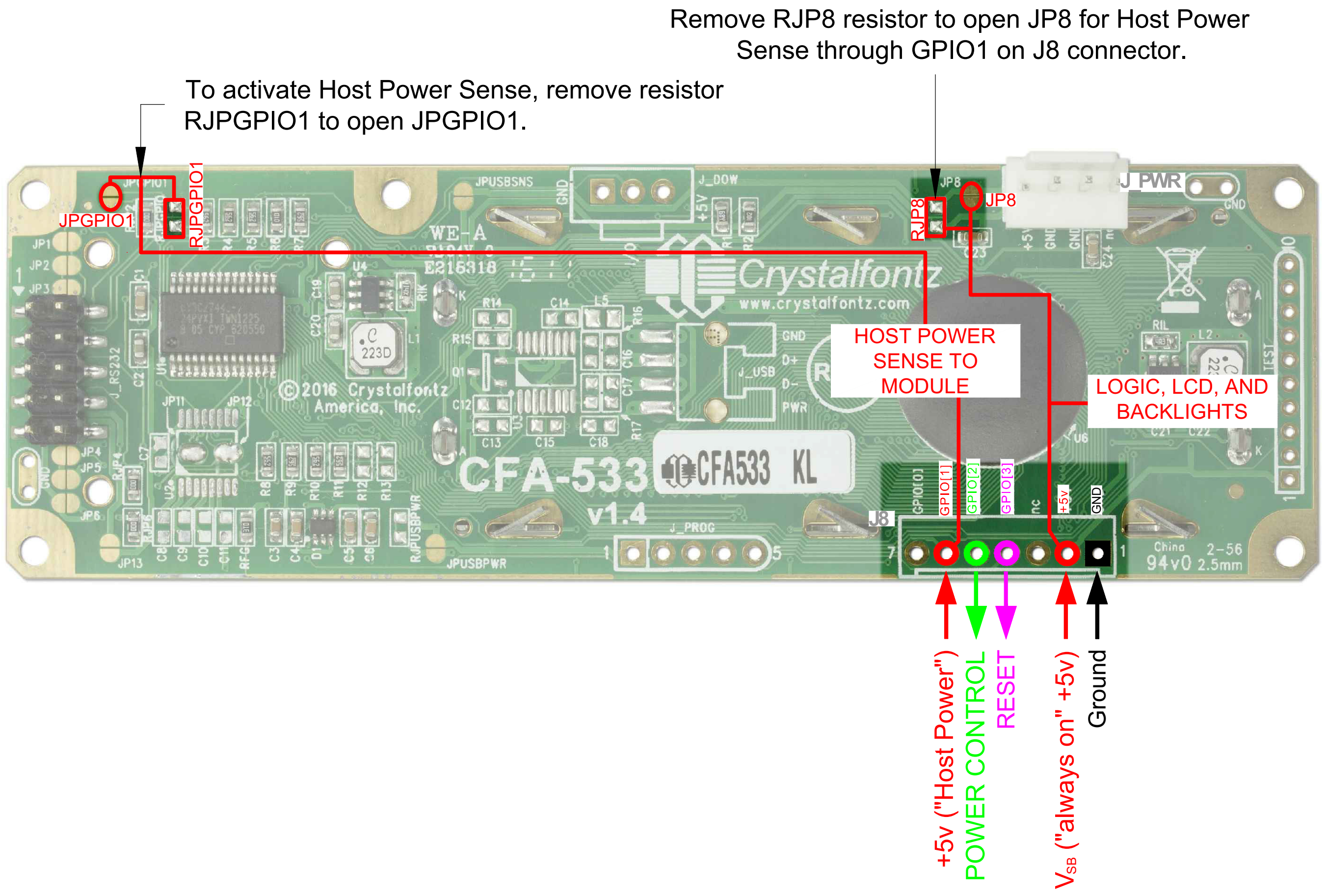

Remove RJP8 resistor to open JP8 for Host Power Sense through GPIO1 on J8 connector.
To activate Host Power Sense, remove resistor RJPGPIO1 to open JPGPIO1.
JPGPIO1
RJPGPIO1
RJP8
JP8
HOST POWER SENSE TO MODULE
LOGIC, LCD, AND BACKLIGHTS
GPIO[0]
GPIO[1]
GPIO[2]
GPIO[3]
nc
+5v
GND
J8
+5v ("Host Power")
POWER CONTROL
RESET
VSB ("always on" +5v)
Ground

Figure 14. ATX Host Power Sense through GPIO[1] on J8 Connector

The CFA533-TMI-KL can be configured to sense host power through GPIO[1] on connector J8. In addition to +5$_{VSB}$, Ground, Power Control (GPIO[2]), and Reset Control (GPIO[3]) connections, you will need to supply connection to the host's +5v power to GPIO[1]. JP8 is closed by default. ***To properly function, JP8 must be opened by removing RJP8 resistor. To activate Host Power Sense, remove resistor RJPGPIO1 to open JPGPIO1.***

The POWER-ON SENSE can be provided through Pin 6 of J8 (GPIO[1]). This option is only provided to allow backwards compatibility for legacy CFA633 applications. R3 is loaded in series with GPIO1 with a 5.6KΩ 0805 SMT resistor for this functionality.

Here is an excerpt from command 28 (0x1C): Set ATX Switch Functionality (Pg. 46):

Note

The GPIO pins used for ATX control must not be configured as user GPIO. The pins must be configured to their default drive mode in order for the ATX functions to work correctly.

These settings are factory default but may be changed by the user. Please see command 34 (0x22): Set/Configure GPIO (Pg. 50). These settings must be saved as the boot state.

To ensure that GPIO[1] will operate correctly as ATX SENSE, user GPIO[1] must be configured as:

```
DDD = "011: 1=Resistive Pull Up, 0=Fast, Strong Drive Down".
F = "0: Port unused for user GPIO."
```

This configuration can be assured by sending the following command:

```
command = 34
length = 3
data[0] = 1
data[1] = 0
data[2] = 3
```

To ensure that GPIO[2] will operate correctly as ATX POWER, user GPIO[2] must be configured as:

```
DDD = "010: Hi-Z, use for input".
F = "0: Port unused for user GPIO."
```

This configuration can be assured by sending the following command:

```
command = 34
length = 3
data[0] = 2
data[1] = 0
data[2] = 2
```

To ensure that GPIO[3] will operate correctly as ATX RESET, user GPIO[3] must be configured as:

```
DDD = "010: Hi-Z, use for input".
F = "0: Port unused for user GPIO."
```

This configuration can be assured by sending the following command:

```
command = 34
length = 3
data[0] = 3
data[1] = 0
data[2] = 2
```

These settings must be saved as the boot state.

ATX Keypad Control

Once configured by the host software (see command 28 (0x1C): Set ATX Switch Functionality (Pg. 46)), the following functions may be individually enabled:

- **System power on.** If POWER-ON SENSE is low (0th), pressing the green check key (Enter key) for 0.25 seconds will turn the unit on by driving POWER CONTROL line for the pulse width set by command 28 (0x1C): Set ATX Switch Functionality (Pg. 46) (1.0 seconds default).
- **System hard power off.** If POWER-ON SENSE is high (+5v power, V_{DD}) pressing the red X key (Cancel key) for 4 seconds will turn the system off by driving the POWER CONTROL line. The line will be driven for a minimum of the pulse width set by command 28 (0x1C): Set ATX Switch Functionality (Pg. 46) (1.0 seconds default). If the user continues to press the key, the CFA533-TMI-KL will continue to drive the line for up to an additional 5 seconds.

- **System hard reset.** If POWER-ON SENSE is high (+5v power, V_{DD}) pressing the green check key (Enter key) for 4 seconds will reset the system by driving the RESET line for 1 second. The CFA533-TMI-KL will reboot itself immediately after resetting the host.

Since the host and display module must look off if the host’s power is off, the CFA533-TMI-KL can be configured to monitor the POWER-ON SENSE line and blank its display any time the POWER-ON SENSE line is low.

DATA COMMUNICATIONS: DETAILS FOR RS232 CONNECTIONS

JP2, JP4, and JP6 are closed at the factory, selecting the “Default RS-232 Pin Assignments” (see *Figure 15. on Pg. 29*). This connection allows a low-cost ribbon cable (Crystalfontz WR-232-Y08) to connect the CFA533-TMI-KL to a host's DB-9 COM port.

If you are connecting the CFA533-TMI-KL to a host system (such as a PC motherboard) that has a similar 10-pin 0.1-inch connector, rather than a standard RS-232 DB-9 connector common to rear panels, the pinouts may need to be changed from “Default” to “Alternate”.

For an RS-232 connection, you can modify the display modules or have Crystalfontz modify them for you.

Choose one of two methods described below to make the connection.

Method 1: Use Crystalfontz WR-232-Y22 cable
The easiest method is to use a Crystalfontz WR-232-Y22 cable. Connect the single end of the WR-232-Y22 cable to the CFA533-TMI-KL. On the double end of the WR-232-Y22 cable, one connector will work for host connections that use “default” numbering; the other end will work for host connections that use the “alternate” numbering.

Method 2: Use Straight-through 10-pin to 10-pin ribbon cable
Use a straight-through 10-pin to 10-pin ribbon cable (for example, CW Industries’ C3AAG-1018G-ND cable available from Digi-Key). The pin order of your motherboard's header will determine if the CFA533-TMI-KL's pin assignments need to be “Default” or “Alternate”. Open or close jumpers JP1-JP6 as necessary to set the CFA533-TMI-KL to “Default” or “Alternate” that matches your motherboard.

Customized Parts: Modification by Crystalfontz

Other modifications are available as custom parts to suit your product requirements. For information, please contact technical support (+1-888-206-9720 or email techinfo@crystalfontz.com). We will provide you with a semi-custom part number and pricing. A minimum order quantity may apply.

Please note that the CFA533-TMI-KL can be powered through this header. Please refer to 3. Connection through J_RS232 Connector (Non-ATX) (Pg. 23).

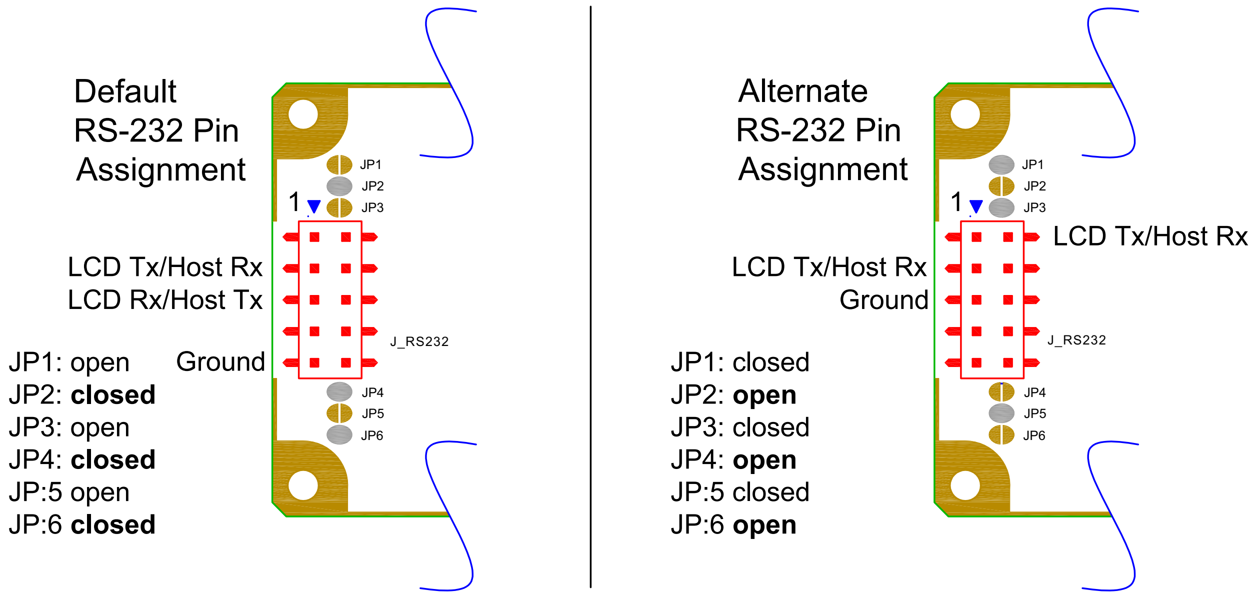


Figure 15. J_RS232 Default and Alternate Pin Assignments

GPIO CONNECTIONS

The CFA533-TMI-KL has five General-Purpose Input/Output (GPIO) pins. The GPIO are port pins from the CFA533-TMI-KL's micro-controller brought out to connectors. As an output, a GPIO can be used to turn on an LED, or perhaps drive a relay. As an input, a GPIO can be used to read a switch or a button. Most of the GPIOs have a default function that allows the display module to perform some special purpose activity with the pin.

```
GPIO[0] = J8, Pin 7
GPIO[1] = J8, Pin 6 (may be used as ATX Host Power Sense, has R3 in series)
GPIO[2] = J8, Pin 5 (default is ATX Host Power Control)
GPIO[3] = J8, Pin 4 (default is ATX Host Reset Control)
GPIO[4] = J_DOW, Pin 2 (default is DOW I/O -- has 1KΩ resistor hardware pull-up: R2)
```

GPIO[0], GPIO[2] and GPIO[3] are connected directly from the micro-controller port pin to the connector pin.

GPIO[1] has a series 5.6KΩ resistor in R3.

GPIO[4] is also used as the DOW I/O pin. Since the DOW requires a pull-up on the I/O pin, a 1KΩ resistor in R2 is loaded to pull GPIO[4] to V_{DD} (+5v power).

Please refer to commands 34 (0x22): Set/Configure GPIO (Pg. 50) and 35 (0x23): Read GPIO Pin Levels and Configuration State (Pg. 51) for additional details concerning the GPIO operation.

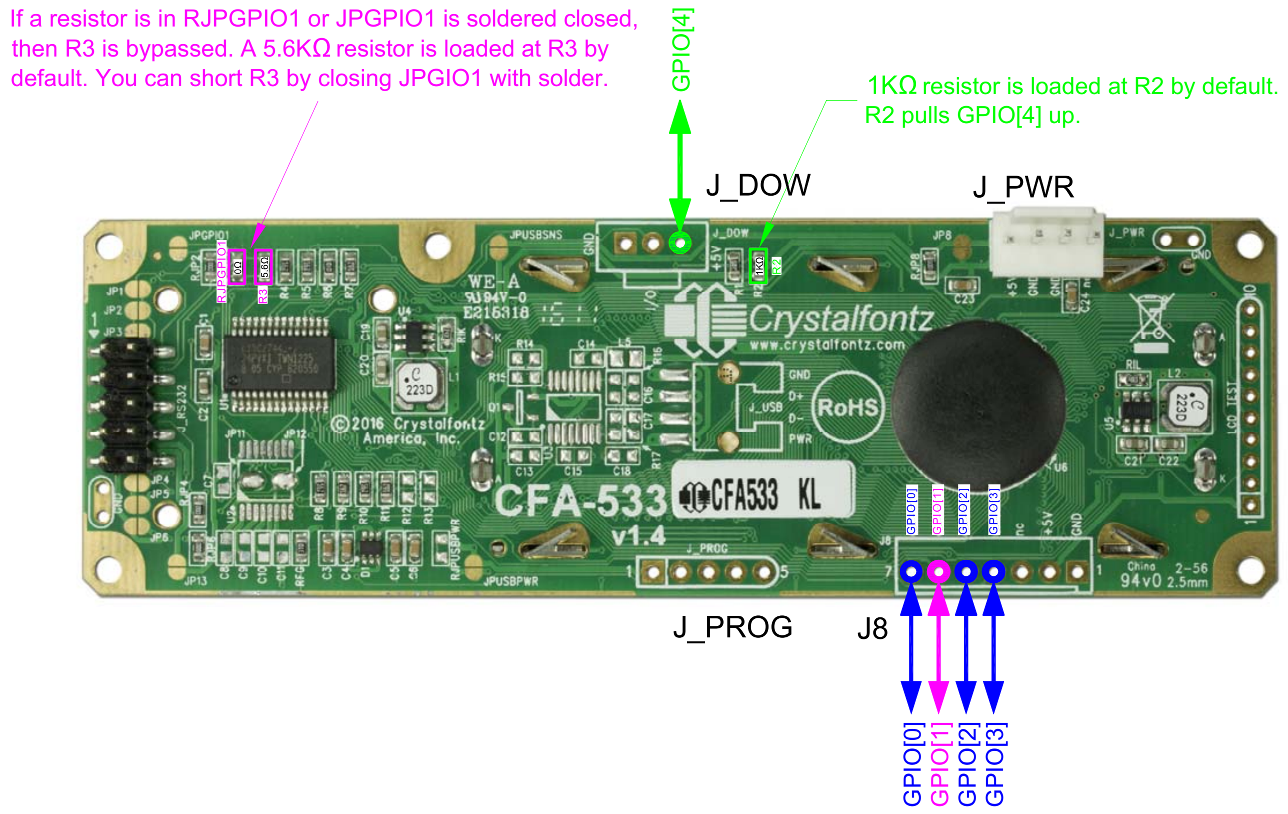

If a resistor is in RJPGPIO1 or JPGPIO1 is soldered closed, then R3 is bypassed. A 5.6KΩ resistor is loaded at R3 by default. You can short R3 by closing JPGIO1 with solder.
GPIO[4]
1KΩ resistor is loaded at R2 by default. R2 pulls GPIO[4] up.
J_DOW
J_PWR
J_PROG
J8
GPIO[0]
GPIO[1]
GPIO[2]
GPIO[3]

Figure 16. Location of GPIO Connections, Resistors, and J_DOW

1-WIRE (DOW) DEVICE CONNECTIONS FOR OPTIONAL ACCESSORIES

Temperature Sensors

The these displays support Maxim's 1-Wire (DOW) temperature sensors. (Dallas One Wire - uses the standard Dallas Semiconductor 1-Wire protocol for data transfers.) When you order these displays through our website, you can configure your display module to include a DOW mating connector and DOW temperature sensor cables WR-DOW-Y17.

The Crystalfontz WR-DOW-Y17 has a DS18B20 attached to a "daisy chain" cable. If a WR-DOW-Y17 is ordered at the same time as the display, Crystalfontz can load the WR-DOW-Y17's mating connector into the display's DOW position. For reference, the mating connector for the WR-DOW-Y17 is Molex 0705430002 available from Digi-Key or other parts suppliers.

The temperature sensor can be configured to be automatically read and displayed on the display's in °C or °F (see command 21 (0x15): Set Up Live Temperature Display (Pg. 43)). Independently, any temperature sensor can be configured to report to the host (see 19 (0x13): Set Up Temperature Reporting (Pg. 42)). The sensors configured to be reported are updated once each second.

Other 1-Wire Devices

Other 1-Wire devices may be connected to the 1-Wire bus, with the display acting as a bridge between RS-232 and the 1-Wire bus (see command 21 (0x15): Set Up Live Temperature Display (Pg. 43)). The total number of 1-Wire devices supported is 32, including directly supported temperature sensors and any other user-provided 1-Wire devices. (See display's DOW connection location in Figure 16. on Pg. 30 above.) The display module can send up to 15 bytes and receive up to 14 bytes. This will be sufficient for many devices but some devices require larger transactions and cannot be fully used with the display module.

The display has a 1KΩ resistor hardware pull-up on the J_DOW connector's I/O line.

Connect the 1-Wire sensors as detailed in the sensor's datasheet.

HOST COMMUNICATIONS

Note

Because there is no difference in communications and commands for I^{2}C variants (part numbers ending in "-KC"), serial variants (part numbers ending in "-KL" or "-KS") and *USB* variants (part numbers ending in "-KU") of the CFA533, the Host Communications section of this Datasheet uses the shorter term "CFA533" instead of "CFA533-TMI-KL".

The CFA533 series (includes CFA533-TMI-KL) communicates with its host using an RS-232 interface. The port settings are 19200 baud, 8 data bits, no parity, 1 stop bit by factory default. The speed can be set to 115200 baud under software control (see command 33 (0x21): Set Baud Rate (Pg. 49)).

PACKET STRUCTURE

All communication between the CFA533 and the host takes place in the form of a simple and robust CRC checked packet. The packet format allows for very reliable communications between the CFA533 and the host without the traditional problems that occur in a stream-based serial communication (such as having to send data in inefficient ASCII format, to "escape" certain "control characters", or losing sync if a character is corrupted, missing, or inserted).

Note

Reconciling packets is recommended rather than using delays when communicating with the display module. To reconcile your packets, please ensure that you have received the acknowledgment packet from the packet most recently sent before sending any additional packets to the display module. This practice will guarantee that you will not have any dropped packets or missed communication with the display module.

All packets have the following structure:

```
<type><data_length><data><CRC>
```

`type` is one byte, and identifies the type and function of the packet:

```
TTcc cccc
|||| ||||--Command, response, error or report code 0-63
||---------Type:
             00 = normal command from host to CFA533
             01 = normal response from CFA533 to host
             10 = normal report from CFA533 to host (not indirect response to a command
                  from the host)
             11 = error response from CFA533 to host (a packet with valid structure but
                  illegal content was received by the CFA533)
```

`data_length` specifies the number of bytes that will follow in the data field. The valid range of `data_length` is 0 to 18.

`data` is the payload of the packet. Each `type` of packet will have a specified `data_length` and format for `data` as well as algorithms for decoding `data` detailed below.

`CRC` is a standard 16-bit CRC of all the bytes in the packet except the CRC itself. The CRC is sent LSB first. At the port, the CRC immediately follows the last used element of data `[]`. See APPENDIX A: CALCULATING THE CRC (Pg. 55) for several examples of how to calculate the CRC in different programming languages.

The following concept may be useful for understanding the packet structure.

```
typedef struct
  {
  unsigned char
    command;
  unsigned char
    data_length;
  unsigned char
    data[data_length];
  unsigned short
    CRC;
  }COMMAND_PACKET;
```

Crystalfontz supplies a demonstration and test program cfTest along with its C source code. Both will work with the CFA533 modules. Included in the cfTest source is a CRC algorithm and an algorithm that detects packets. The algorithm will automatically re-synchronize to the next valid packet in the event of any communications errors. Please follow the algorithm in the sample code closely in order to realize the benefits of using the packet communications.

ABOUT HANDSHAKING

The nature of CFA533's packets makes it unnecessary to implement traditional hardware or software handshaking.

The host should wait for a corresponding acknowledge packet from the CFA533 before sending the next command packet. The CFA533 will respond to all packets within 250 mS. The host software should stop waiting and retry the packet if the CFA533 fails to respond within 250 mS. The host software should report an error if a packet is not acknowledged after several retries. This situation indicates a hardware problem — for example, a disconnected cable.

Please note that some operating systems may introduce delays between when the data arrives at the physical port from the CFA533 until it is available to the user program. In this case, the host program may have to increase its timeout window to account for the additional overhead of the operating system.

The CFA533 can be configured to send several types of report packets along with regular acknowledge packets. The host should be able to buffer several incoming packets and must guarantee that it can process and remove packets from its input buffer faster than the packets can arrive given the baud rate and the reporting configuration of the CFA533. For any modern PC using reasonably efficient software, this requirement will not pose a challenge.

The report packets are sent asynchronously with respect to the command packets received from the host. The host should not assume that the first packet received after it sends a command is the acknowledge packet for that command. The host should inspect the `type` field of incoming packets and process them accordingly.

REPORT CODES

The CFA533 can be configured to report two items. The CFA533 sends reports automatically when the data becomes available. Reports are not sent in response to a particular packet received from the host. The two report types are:

0x80: Key Activity

If a key is pressed or released, the CFA533 sends a Key Activity report packet to the host. Key event reporting may be individually enabled or disabled by command 23 (0x17): Configure Key Reporting (Pg. 44).

```
type: 0x80
data_length: 1
data[0] is the type of keyboard activity:
       KEY_UP_PRESS              1
       KEY_DOWN_PRESS            2
       KEY_LEFT_PRESS            3
       KEY_RIGHT_PRESS           4
       KEY_ENTER_PRESS           5
       KEY_EXIT_PRESS            6
       KEY_UP_RELEASE            7
       KEY_DOWN_RELEASE          8
       KEY_LEFT_RELEASE          9
       KEY_RIGHT_RELEASE        10
       KEY_ENTER_RELEASE        11
       KEY_EXIT_RELEASE         12
```

0x81: (reserved)

0x82: Temperature Sensor Report

If any of the up to 32 temperature sensors is configured to report to the host, the CFA533 will send Temperature Sensor Reports for each selected sensor every second. See the command 19 (0x13): Set Up Temperature Reporting (Pg. 42) below.

```
type: 0x82
data_length: 4
data[0] is the index of the temperature sensor being reported:
       0 = temperature sensor 1
       1 = temperature sensor 2
       . . .
       31 = temperature sensor 32
data[1] is the LSB of Temperature_Sensor_Counts
data[2] is the MSB of Temperature_Sensor_Counts
data[3] is DOW_crc_status
```

The following C function will decode the Temperature Sensor Report packet into °C and °F:

```
void OnReceivedTempReport(COMMAND_PACKET *packet, char *output)
  {
  //First check the DOW CRC return code from the CFA533
  if(packet->data[3]==0)
    strcpy(output,"BAD CRC");
  else
    {
    double
      degc;
    degc=(*(short *)&(packet->data[1]))/16.0;

    double
      degf;
    degf=(degc*9.0)/5.0+32.0;

    sprintf(output,"%9.4f°C =%9.4f°F",
            degc,
            degf);
    }
  }
```

COMMAND CODES

Below is a list of valid commands for the CFA533. Each command packet is answered by either a response packet or an error packet.The low 6 bits of the `type` field of the response or error packet is the same as the low 6 bits of the `type` field of the command packet being acknowledged.

0 (0x00): Ping Command

The CFA533 will return the Ping Command to the host.

```
type: 0x00 = 0₁₀
valid data_length is 0 to 16
data[0-(data_length-1)] can be filled with any arbitrary data
```

The return packet is identical to the packet sent, except the type will be 0x40 (normal response, Ping Command):

```
type: 0x40 | 0x00 = 0x40 = 64₁₀
data_length: (identical to received packet)
data[0-(data_length-1)] = (identical to received packet)
```

1 (0x01): Get Hardware & Firmware Version

The CFA533 will return the hardware and firmware version information to the host.

```
type: 0x01 = 1₁₀
valid data_length is 0
```

The return packet will be:

```
type: 0x40 | 0x01 = 0x41 = 65₁₀
data_length: 16
data[] = "CFA533:hX.X,sYvY"

hX.X is the hardware version, for example, "h1.4"
sYvY is the firmware version, for example, "s1v2".
```

2 (0x02): Write User Flash Area

The CFA533 reserves 16 bytes of nonvolatile memory for arbitrary use by the host. This memory can be used to store data such as a serial number, IP address, gateway address, netmask, or any other data required. All 16 bytes must be supplied.

```
type: 0x02 = 2₁₀
valid data_length is 16
data[] = 16 bytes of arbitrary user data to be stored in the CFA533's non-volatile memory
```

The return packet will be:

```
type: 0x40 | 0x02 = 0x42 = 66₁₀
data_length: 0
```

3 (0x03): Read User Flash Area

This command will read the User Flash Area and return the data to the host.

```
type: 0x03 = 3₁₀
valid data_length is 0
```

The return packet will be:

```
type: 0x40 | 0x03 = 0x43 = 67₁₀
data_length: 16
data[] = 16 bytes user data recalled from the CFA533's non-volatile memory
```

4 (0x04): Store Current State as Boot State

The CFA533 loads its power-up configuration from nonvolatile memory when power is applied. The CFA533 is configured at the factory to display a welcome screen when power is applied. This command can be used to customize the welcome screen, as well as the following items:

- Characters shown on LCD (display module), which are affected by:
  - command 6 (0x06): Clear LCD Screen (Pg. 37).
  - command 7 (0x07): Set LCD Contents, Line 1 (Pg. 38).
  - command 8 (0x08): Set LCD Contents, Line 2 (Pg. 38).
  - command 31 (0x1F): Send Data to LCD (Pg. 49).
- Special character font definitions (command 9 (0x09): Set LCD Special Character Data (Pg. 38)).
- Cursor position (command 11 (0x0B): Set LCD Cursor Position (Pg. 39)).
- Cursor style (command 12 (0x0C): Set LCD Cursor Style (Pg. 39)).
- Contrast setting (command 13 (0x0D): Set LCD Contrast (Pg. 40)).
- LCD backlight setting (command 14 (0x0E): Set LCD & Keypad Backlights (Pg. 40)).
- Keypad backlight setting (command 14 (0x0E): Set LCD & Keypad Backlights (Pg. 40)).
- Settings of any live displays (command 21 (0x15): Set Up Live Temperature Display (Pg. 43)).
- Key press and release masks (command 23 (0x17): Configure Key Reporting (Pg. 44)).
- ATX function enable and pulse length settings (command 28 (0x1C): Set ATX Switch Functionality (Pg. 46)).
- Baud rate (command 33 (0x21): Set Baud Rate (Pg. 49)).
- GPIO settings (command 34 (0x22): Set/Configure GPIO (Pg. 50)).

You cannot store the temperature reporting (although the live display of temperatures can be saved). You cannot store the host watchdog.The host software should enable this item once the system is initialized and it is ready to receive the data.

```
type: 0x04 = 4₁₀
valid data_length is 0
```

The return packet will be:

```
type: 0x40 | 0x04 = 0x44 = 68₁₀
data_length: 0
```

5 (0x05): Reboot CFA533, Reset Host, or Power Off Host (ATX Required)

This command instructs the CFA533 to simulate a power-on restart of itself, reset the host, or turn the host's power off. The ability to reset the host may be useful to allow certain host operating system configuration changes to complete. The ability to turn the host's power off under software control may be useful in systems that do not have ACPI* compatible BIOS.

*****_**A**dvanced **C**onfiguration and **P**ower **I**nterface) is an industry specification for the efficient handling of power consumption in desktop and mobile computers._

Note

The GPIO pins used for ATX control must not be configured as user GPIO, and must be configured to their default drive mode in order for the ATX functions to work correctly. These settings are factory default, but may be changed by the user. Please see command 34 (0x22): Set/Configure GPIO (Pg. 50).

Rebooting the CFA533 may be useful when testing the boot configuration. It may also be useful to re-enumerate the devices on the 1-Wire (DOW) bus. To reboot the CFA533, send the following packet:

```
type: 0x05 = 5₁₀
valid data_length is 3
data[0] = 8
data[1] = 18
data[2] = 99
```

To reset the host, assuming the host's reset line is connected to GPIO[3] as described in command 28 (0x1C): Set ATX Switch Functionality (Pg. 46), send the following packet:

```
type: 0x05 = 5₁₀
valid data_length is 3
data[0] = 12
data[1] = 28
data[2] = 97
```

To turn the host's power off, assuming the host's power control line is connected to GPIO[2] as described in command 28 (0x1C): Set ATX Switch Functionality (Pg. 46), send the following packet:

```
type: 0x05 = 5₁₀
valid data_length is 3
data[0] = 3
data[1] = 11
data[2] = 95
```

In any of the above cases, the return packet will be:

```
type: 0x40 | 0x05 = 0x45 = 69₁₀
data_length: 0
```

6 (0x06): Clear LCD Screen

Sets the contents of the LCD screen DDRAM to ' ' = 0x20 = 32_{10} and moves the cursor to the left-most column of the top line.

```
type: 0x06 = 6₁₀
valid data_length is 0
```

The return packet will be:

```
type: 0x40 | 0x06 = 0x46 = 70₁₀
data_length: 0
```

Clear LCD Screen changes the LCD. The LCD contents is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

7 (0x07): Set LCD Contents, Line 1

Sets the 16 characters displayed for the top line of LCD screen.

> Note
> Please use this command only if you need backwards compatibility with older CFA633 display modules. For new applications, please use the more flexible command 31 (0x1F): Send Data to LCD (Pg. 49) which is also supported by the CFA631 and CFA635.

```
type: 0x7 = 7₁₀
valid data_length is 16
data[] = top line's display content (must supply 16 bytes)
```

The return packet will be:

```
type: 0x40 | 0x07 = 0x47 = 71₁₀
data_length: 0
```

Set LCD Contents, Line 1 is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

8 (0x08): Set LCD Contents, Line 2

Sets the 16 characters displayed for the bottom line of LCD screen.

> Note
> Please use this command only if you need backwards compatibility with older CFA633 display modules. For new applications, please use the more flexible command 31 (0x1F): Send Data to LCD (Pg. 49) which is also supported by the CFA631 and CFA635.

```
type: 0x08 = 8₁₀
valid data_length is 16
data[] = bottom line's display content (must supply 16 bytes)
```

The return packet will be:

```
type: 0x40 | 0x08 = 0x48 = 72₁₀
data_length: 0
```

Set LCD Contents, Line 2 is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

9 (0x09): Set LCD Special Character Data

Sets the font definition for one of the special characters (CGROM).

```
type: 0x09 = 9₁₀
valid data_length is 9
data[0] = index of special character that you would like to modify, 0-7 are valid
data[1-8] = bitmap of the new font for this character
```

`data[1-8]` are the bitmap information for this character. Any value is valid between 0 and 63, the msb is at the left of the character cell of the row, and the lsb is at the right of the character cell.

`data[1]` is at the top of the cell,

`data[8]` is at the bottom of the cell.

The return packet will be:

```
type: 0x40 | 0x09 = 0x49 = 73₁₀
data_length: 0
```

Set LCD Special Character Data is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

10 (0x0A): Read 8 Bytes of LCD Memory

This command will return the contents of the LCD's DDRAM or CGROM. This command is intended for debugging.

```
type: 0x0A = 10₁₀
valid data_length is 1
data[0] = address code of desired data

data[0] is the address code native to the LCD controller:
      0x40 (\064) to 0x7F (\127) for CGRAM
      0x80 (\128) to 0x8F (\143) for DDRAM, line 1
      0xC0 (\192) to 0xCF (\207) for DDRAM, line 2
```

The return packet will be:

```
type: 0x40 | 0x0A = 0x4A = 74₁₀
data_length: 9
```

`data[0]` of the return packet will be the address code.
`data[1-8]` of the return packet will be the data read from the LCD controller's memory.

11 (0x0B): Set LCD Cursor Position

This command allows the cursor to be placed at the desired location on the CFA533's LCD screen. If you want the cursor to be visible, you may also need to send a command 12 (0x0C): Set LCD Cursor Style (Pg. 39).

```
type: 0x0B = 11₁₀
valid data_length is 2
data[0] = column (0-15 valid)
data[1] = row (0-1 valid)
```

The return packet will be:

```
type: 0x40 | 0x0B = 0x4B = 75₁₀
data_length: 0
```

Set LCD Cursor Position is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

12 (0x0C): Set LCD Cursor Style

This command allows you to select among four hardware generated cursor options.

```
type: 0x0C = 12₁₀
valid data_length is 1
data[0]: cursor style (0-3 valid)
       0 = no cursor
       1 = blinking block cursor
       2 = underscore cursor
       3 = blinking underscore (Note: This behavior is not the same as the CFA633 series
           which is: blinking block plus underscore.
```

The return packet will be:

```
type: 0x40 | 0x0C = 0x4C = 76₁₀
data_length: 0
```

Set LCD Cursor Style is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

13 (0x0D): Set LCD Contrast

This command sets the contrast or vertical viewing angle of the display. (Initiated by the host, responded to by the CFA533.)

- CFA633 Compatible
  If only one byte of data is supplied, then it is the "CFA633 Compatible" version of the command. Requires 1 byte (0-200) are valid, but only (0-50) are useful for this LCD.

```
type: 0x0D = 13₁₀
   valid data_length is 1
   data[0]: contrast setting (0-50 valid)
          0 = light
         16 = about right
         29 = dark
      30-50 = very dark
```

The return packet will be:

```
type: 0x40 | 0x0D = 0x4D = 77₁₀
data_length: 0
```

- CFA533 Enhanced
  If two bytes of data are supplied, then the command takes advantage of the CFA533s native enhanced contrast resolution. Requires 2 bytes.
  - The first byte data[0] is ignored, any value from 0 to 255 is accepted.
  - The second byte data[1] controls the CFA533 contrast with better resolution.

```
type: 0x0D = 13₁₀
valid data_length is 1
data[0]: required but ignored
data[1]: contrast setting (0-200 valid)
       0-99 = lighter
        100 = no correction
    101-200 = darker
```

The return packet will be:

```
type: 0x40 | 0x0D = 0x4D = 77₁₀
data_length: 0
```

Set LCD Contrast is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

14 (0x0E): Set LCD & Keypad Backlights

This command sets the brightness of the LCD and keypad backlights. (Initiated by the host, responded to by the CFA533.)

- If one byte is supplied, both the keypad and LCD backlights are set to that brightness (CFA633 compatible).

```
type: 0x0E = 14₁₀
valid data_length is 1
data[0]: keypad and LCD backlight power setting (0-100 valid)
       0 = off
   1-100 = variable brightness
```

The return packet will be:

```
type: 0x40 | 0x0E = 0x4E = 78₁₀
data_length: 0
```

- If two bytes are supplied, the LCD is set to the brightness of the first byte, the keypad is set to the brightness of the second byte.

```
type: 0x0E = 14₁₀
valid data_length is 2
data[0]: LCD backlight power setting (0-100 valid)
       0 = off
   1-100 = variable brightness

data[1]: keypad backlight power setting (0-100 valid)
       0 = off
   1-100 = variable brightness
```

The return packet will be:

```
type: 0x40 | 0x0E = 0x4E = 78₁₀
data_length: 0
```

Set LCD & Keypad Backlight is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

18 (0x12): Read DWR-DOW-Y17 Temperature Sensors

When power is applied to the CFA533,WR-DOW-Y17 it detects any devices (WR-DOW-Y17) connected to the 1-Wire (DOW) bus and stores the device's information. This command will allow the host to read the device's information.

Note

The GPIO pin used for DOW must not be configured as user GPIO. It must be configured to its default drive mode in order for the DOW functions to work correctly.

These settings are factory default but may be changed by the user. Please see command 34 (0x22): Set/Configure GPIO (Pg. 50).

In order for the DOW subsystem to be enabled and operate correctly, user GPIO[4] must be configured as:

```
DDD = "111: 1=Hi-Z, 0=Slow, Strong Drive Down".
F = "0: Port unused for user GPIO."
```

This state is the factory default, but it can be changed and saved by the user. To ensure that GPIO[4] is set correctly and the DOW operation is enabled, send the following command:

```
command = 34
length = 3
data[0] = 4
data[1] = 100
data[2] = 7
```

This setting must be saved as the boot state, so when the CFA533 reboots it will detect the WR-DOW-Y17 temperature sensors.

```
type: 0x12 = 18₁₀
valid data_length is 1
data[0] = device index (0-31 valid)
```

The return packet will be:

```
type: 0x40 | 0x12 = 0x52 = 82₁₀
data_length: 9
data[0]   = device index (0-31 valid)
data[1-8] = ROM ID of the device
```

If data[1] is 0x22 (WR-DOW-Y17 temperature sensor), then that device can be set up to automatically convert and report the temperature every second. See the command 19 (0x13): Set Up Temperature Reporting (Pg. 42).

19 (0x13): Set Up Temperature Reporting

This command will configure the CFA533 to report the temperature information to the host every second.

```
type: 0x13 = 19₁₀
valid data_length is 4
data[0-3] = 32-bit bitmask indicating which temperature sensors are enabled to report
            (0-255 valid in each location)

data[0]
08 07 06 05   04 03   02 01  Enable Reporting of sensor with
|  |  |  |    |  |    |  |   device index of:
|  |  |  |    |  |    |  |--  0: 1 = enable, 0 = disable
|  |  |  |    |  |    |-----  1: 1 = enable, 0 = disable
|  |  |  |    |  |---------  2: 1 = enable, 0 = disable
|  |  |  |    |------------  3: 1 = enable, 0 = disable
|  |  |  |-----------------  4: 1 = enable, 0 = disable
|  |  |--------------------  5: 1 = enable, 0 = disable
|  |-----------------------  6: 1 = enable, 0 = disable
|--------------------------  7: 1 = enable, 0 = disable

data[1]
16 15 14 13   12 11   10 09  Enable Reporting of sensor with
|  |  |  |    |  |    |  |   device index of:
|  |  |  |    |  |    |  |--  8: 1 = enable, 0 = disable
|  |  |  |    |  |    |-----  9: 1 = enable, 0 = disable
|  |  |  |    |  |--------- 10: 1 = enable, 0 = disable
|  |  |  |    |------------ 11: 1 = enable, 0 = disable
|  |  |  |----------------- 12: 1 = enable, 0 = disable
|  |  |-------------------- 13: 1 = enable, 0 = disable
|  |----------------------- 14: 1 = enable, 0 = disable
|-------------------------- 15: 1 = enable, 0 = disable

data[2]
24 23 22 21   20 19   18 17  Enable Reporting of sensor with
|  |  |  |    |  |    |  |   device index of:
|  |  |  |    |  |    |  |-- 16: 1 = enable, 0 = disable
|  |  |  |    |  |    |----- 17: 1 = enable, 0 = disable
|  |  |  |    |  |--------- 18: 1 = enable, 0 = disable
|  |  |  |    |------------ 19: 1 = enable, 0 = disable
|  |  |  |----------------- 20: 1 = enable, 0 = disable
|  |  |-------------------- 21: 1 = enable, 0 = disable
|  |----------------------- 22: 1 = enable, 0 = disable
|-------------------------- 23: 1 = enable, 0 = disable


data[3]
32 31 30 29   28 27   26 25  Enable Reporting of sensor with
|  |  |  |    |  |    |  |   device index of:
|  |  |  |    |  |    |  |-- 24: 1 = enable, 0 = disable
|  |  |  |    |  |    |----- 25: 1 = enable, 0 = disable
|  |  |  |    |  |--------- 26: 1 = enable, 0 = disable
|  |  |  |    |------------ 27: 1 = enable, 0 = disable
|  |  |  |----------------- 28: 1 = enable, 0 = disable
|  |  |-------------------- 29: 1 = enable, 0 = disable
|  |----------------------- 30: 1 = enable, 0 = disable
|-------------------------- 31: 1 = enable, 0 = disable
```

Any sensor enabled must have been detected as a 0x22 (DS1822 temperature sensor) or 0x28 (DS18B20 temperature sensor) during DOW enumeration. This can be verified by using the command 18 (0x12): Read DWR-DOW-Y17 Temperature Sensors (Pg. 41).

The return packet will be:

```
type: 0x40 | 0x13 = 0x53 = 83₁₀
data_length: 0
```

20 (0x14): Arbitrary 1-Wire (DOW) Transaction

The CFA533 can function as an RS-232 to 1-Wire (DOW) bridge. The CFA533 can send up to 15 bytes and receive up to 14 bytes. This will be sufficient for many devices, but some devices require larger transactions cannot be fully used with the CFA533.

This command allows you to specify arbitrary transactions on the 1-Wire bus. The 1-Wire commands follow this basic layout:

This command allows you to specify arbitrary transactions on the 1-Wire bus. 1-Wire commands follow this basic layout:

```
<bus reset>        //Required
<address_phase>    //Must be "Match ROM" or "Skip ROM"
<write_phase>      //optional, but at least one of write_phase or read_phase must be sent
<read_phase>       //optional, but at least one of write_phase or read_phase must be sent
```

Please see APPENDIX B: CONNECTING A DS2450 1-WIRE QUAD A/D CONVERTER (Pg. 66) for an example of using this command.

```
type: 0x14 = 20₁₀
valid data_length is 2 to 16
   data[0] = device_index (0-32 valid)
   data[1] = number_of_bytes_to_read (0-14 valid)
data[2-15] = data_to_be_written[data_length-2]
```

If `device_index` is 32, then no address phase will be executed. If `device_index` is in the range of 0 to 31, and a 1-Wire device was detected for that `device_index` at power on, then the write cycle will be prefixed with a "Match ROM" command and the address information for that device.

If `data_length` is two, then no specific write phase will be executed (although address information may be written independently of `data_length` depending on the value of `device_index`).

If `data_length` is greater than two, then `data_length-2` bytes of `data_to_be_written` will be written to the 1-Wire bus immediately after the address phase.

If `number_of_bytes_to_read` is zero, then no read phase will be executed. If `number_of_bytes_to_read` is not zero then `number_of_bytes_to_read` will be read from the bus and loaded into the response packet.

The return packet will be:

```
type: 0x40 | 0x14 = 0x54 = 84₁₀
data_length: 2 to 16
data[0] = device index (0-31 valid)
data[data_length-2] = Data read from the 1-Wire bus. This is the same as
                      number_of_bytes_to_read from the command.
data[data_length-1] = 1-Wire CRC
```

21 (0x15): Set Up Live Temperature Display

You can configure the CFA533 to automatically update a portion of the LCD with a live temperature reading. Once the display is configured using this command, the CFA533 will continue to display the live reading on the LCD without host intervention. The Set Up Live Temperature Display is one of the items stored by command 4 (0x04): Store Current State as Boot State (Pg. 36), so you can configure the CFA533 to immediately display system temperatures as soon as power is applied.

The live display is based on a concept of display slots. There are 4 slots, and each of the 4 slots may be enabled or disabled independently.

Any slot may be requested to display any data that is available. For instance, slot 0 could display temperature sensor 3 in °C, while slot 1 could simultaneously display temperature sensor 3 in °F.

Any slot may be positioned at any location on the LCD, as long as all the digits of that slot fall fully within the display area. It is legal to have the display area of one slot overlap the display area of another slot, but senseless. This situation should be avoided in order to have meaningful information displayed.

```
type: 0x15 = 21₁₀
valid data_length is 7 or 2 (for turning a slot off)
data[0]: display slot (0-3)
data[1]: type of item to display in this slot
       0 = nothing (data_length then must be 2)
       1 = (invalid)
       2 = temperature (data_length then must be 7)
data[2]: index of the sensor to display in this slot:
       0-31 are valid for temperatures (and the temperature device must be attached)
data[3]: number of digits
       for a temperature: 3 digits (-XX or XXX)
       for a temperature: 5 digits (-XX.X or XXX.X)
data[4]: display column
       0-13 valid for a 3-digit temperature
       0-11 valid for a 5-digit temperature
data[5]: display row (0-1 valid)
data[6]: temperature units(0 = deg C, 1 = deg F)
```

If a 1-Wire CRC error is detected, the temperature will be displayed as "ERR" or "ERROR".

The return packet will be:

```
type: 0x40 | 0x15 = 0x55 = 85₁₀
data_length: 0
```

22 (0x16): Send Command Directly to the LCD Controller

The controller on the CFA533 is the Neotec NT7070B (HD44780 compatible). Generally you won't need low-level access to the LCD controller but some arcane functions are not exposed by the CFA533's command set. This command allows you to access the CFA533's LCD controller directly. Note: It is possible to corrupt the CFA533 display using this command.

```
type: 0x16 = 22₁₀
data_length: 2
data[0]: location code
       0 = "Data" register
       1 = "Control" register
data[1]: data to write to the selected register
```

The return packet will be:

```
type: 0x40 | 0x16 = 0x56 = 86₁₀
data_length: 0
```

23 (0x17): Configure Key Reporting

By default, the CFA533 reports any key event to the host. This command allows the key events to be enabled or disabled on an individual basis. The key events set to report are one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

```
#define KP_UP     0x01
#define KP_ENTER  0x02
#define KP_CANCEL 0x04
#define KP_LEFT   0x08
#define KP_RIGHT  0x10
#define KP_DOWN   0x20
```

type: 0x17 = 23_{10}
data_length: 2
data[0]: press mask (0-63 valid)
data[1]: release mask (0-63 valid)

Valid values of the mask are \000-\063.

The return packet will be:

type: 0x40 | 0x17 = 0x57 = 87_{10}
data_length: 0

Configure Key Reporting is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

24 (0x18): Read Keypad, Polled Mode

In some situations, it may be convenient for the host to poll the CFA533 for key activity. This command allows the host to detect which keys are currently pressed, which keys have been pressed since the last poll, and which keys have been released since the last poll.

This command is independent of the key reporting masks set by command 23 (0x17): Configure Key Reporting (Pg. 44). All keys are always visible to this command. Typically both masks of command 23 would be set to "0" if the host is reading the keypad in polled mode.

```
#define KP_UP     0x01
#define KP_ENTER  0x02
#define KP_CANCEL 0x04
#define KP_LEFT   0x08
#define KP_RIGHT  0x10
#define KP_DOWN   0x20
```

type: 0x18 = 24_{10}
data_length: 0

The return packet will be:

```
type: 0x40 | 0x18 = 0x58 = 88₁₀
data_length: 3
data[0] = bit mask showing the keys currently pressed
data[1] = bit mask showing the keys that have been pressed since the last poll
data[2] = bit mask showing the keys that have been released since the last poll
```

28 (0x1C): Set ATX Switch Functionality

The combination of the CFA533 with the Crystalfontz WR-PWR-Y14 cable can be used to replace the function of the power and reset switches in a standard ATX-compatible system. The ATX Power Switch Functionality is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

Note

The GPIO pins used for ATX control must not be configured as user GPIO. The pins must be configured to their default drive mode in order for the ATX functions to work correctly.

These settings are factory default but may be changed by the user. Please see command 34 (0x22): Set/Configure GPIO (Pg. 50). These settings must be saved as the boot state.

To ensure that GPIO[1] will operate correctly as ATX SENSE, user GPIO[1] must be configured as:

```
DDD = "011: 1=Resistive Pull Up, 0=Fast, Strong Drive Down".
F = "0: Port unused for user GPIO."
```

This configuration can be assured by sending the following command:

```
command = 34
length = 3
data[0] = 1
data[1] = 0
data[2] = 3
```

To ensure that GPIO[2] will operate correctly as ATX POWER, user GPIO[2] must be configured as:

```
DDD = "010: Hi-Z, use for input".
F = "0: Port unused for user GPIO."
```

This configuration can be assured by sending the following command:

```
command = 34
length = 3
data[0] = 2
data[1] = 0
data[2] = 2
```

To ensure that GPIO[3] will operate correctly as ATX RESET, user GPIO[3] must be configured as:

```
DDD = "010: Hi-Z, use for input".
F = "0: Port unused for user GPIO."
```

This configuration can be assured by sending the following command:

```
command = 34
length = 3
data[0] = 3
data[1] = 0
data[2] = 2
```

These settings must be saved as the boot state.

The RESET (GPIO[3]) and POWER CONTROL (GPIO[2]) lines on the CFA533 are normally high-impedance. Electrically, they appear to be disconnected or floating. When the CFA533 asserts the RESET or POWER CONTROL lines, they are momentarily driven high or low (as determined by the AUTO_POLARITY, RESET_INVERT or

POWER_INVERT bits, detailed below). To end the power or reset pulse, the CFA533 changes the lines back to high-impedance.

FOUR FUNCTIONS MAY BE ENABLED BY COMMAND 28

Function 1: KEYPAD_RESET

If POWER-ON SENSE (GPIO[1]) is high, holding the green check key for 4 seconds will pulse RESET (GPIO[3]) pin for 1 second. During the 1-second pulse, the CFA533 will show "RESET", and then the CFA533 will reset itself, showing its boot state as if it had just powered on. Once the pulse has finished, the CFA533 will not respond to any commands until after it has reset the host and itself.

Function 2: KEYPAD_POWER_ON

If POWER-ON SENSE (GPIO[1]) is low, pressing the green check key for 0.25 seconds will pulse POWER CONTROL (GPIO[2]) for the duration specified by in data[1] or the default of 1 second. During this time the CFA533 will show "POWER ON", then the CFA533 will reset itself.

Function 3: KEYPAD_POWER_OFF

If POWER-ON SENSE (GPIO[1]) is high, holding the red X key for 4 seconds will pulse POWER CONTROL (GPIO[2]) for the duration specified by in data[1] or the default of 1 second. If the user continues to hold the power key down, then the CFA533 will continue to drive the line for a maximum of 5 additional seconds. During this time the CFA533 will show "POWER OFF".

Function 4: LCD_OFF_IF_HOST_IS_OFF

If LCD_OFF_IF_HOST_IS_OFF is set, the CFA533 will blank its screen and turn off its backlight to simulate its power being off any time POWER-ON SENSE is low.

Note

By default there is an internal POWER-ON-SENSE connected to the +5v pin of J_PWR, selected by setting data[2] to 1. Alternatively, GPIO[1] may be configured to act as POWER-ON-SENSE through R23 of 5K, and specifying data[2] as 0. The CFA533 will still be active (since it is powered by V_{SB}, standby power which is always-on), monitoring the keypad for a power-on keystroke. Once POWER-ON SENSE goes high, the CFA533 will reboot as if power had just been applied to it.

```
#define AUTO_POLARITY           0x01 //Automatically detects polarity for reset and
                                     //power (recommended)
#define RESET_INVERT            0x02 //Reset pin drives high instead of low (ignored if
                                       AUTO_POLARITY is set)
#define POWER_INVERT            0x04 //Power pin drives high instead of low (ignored if
                                       AUTO_POLARITY is set)
#define LCD_OFF_IF_HOST_IS_OFF  0x10
#define KEYPAD_RESET            0x20
#define KEYPAD_POWER_ON         0x40
#define KEYPAD_POWER_OFF        0x80

type: 0x1C = 28₁₀
data_length: 1, 2 or 3
data[0]: bit mask of enabled functions
data[1]: (optional) length of power on & off pulses in 1/32 second
       1 = 1/32 sec
       2 = 1/16 sec
      16 = 1/2 sec
     254 = 7.9 seconds
     255 = Assert power control line until host power state changes

data[2]: (optional) atx_sense_on_floppy (default setting)
       0: sense ATX host state on P2.1 (J8, pin 6 / GPIO [1] -- R3 must be loaded)
       1: sense ATX host state on P0.7 (JPWR,+5v -- recommended configuration))
```

The return packet will be:

```
type: 0x40 | 0x1C = 0x5C = 92₁₀
data_length: 0
```

29 (0x1D): Enable/Feed Host Watchdog Reset

Some systems use hardware watchdog timers to ensure that a software or hardware failure does not result in an extended system outage. Once the host system has booted, a system monitor program is started. The system monitor program would enable the watchdog timer on the CFA533. If the system monitor program fails to feed the CFA533's watchdog timer, the CFA533 will reset the host system.

Note

The GPIO pins used for ATX control must not be configured as user GPIO. They must be configured to their default drive mode in order for the ATX functions to work correctly. These settings are factory default, but may be changed by the user. Please see the note under command 28 (0x1C): Set ATX Switch Functionality (Pg. 46) or command 34 (0x22): Set/Configure GPIO (Pg. 50).

```
type: 0x1D = 29₁₀
data_length: 1
data[0] = enable/timeout

If timeout is 0, the watchdog is disabled.

If timeout is 1-255, then this command must be issued again within timeout seconds to feed
the watchdog and avoid a watchdog reset.

To turn the watchdog off once it has been enabled, simply set timeout to 0.

If the command is not re-issued within timeout seconds, then the CFA533 will reset the
host (see command 28 for details). Since the watchdog is off by default when the CFA533
powers up, the CFA533 will not issue another host reset until the host has once again
enabled the watchdog.
```

The return packet will be:

```
type: 0x40 | 0x1D = 0x5D = 93₁₀
data_length: 0
```

30 (0x1E): Read Reporting/ATX/Watchdog (debug)

This command can be used to verify the current items configured to report to the host, as well as some other miscellaneous status information. Please note that the information returned by the CFA533 is not identical to the information returned by other modules.

```
type: 0x1E = 30
data_length: 0
```

The return packet will be:

```
type = 0x40 | 0x1E = 0x5E = 94₁₀
data_length: 15
data[0] = 0 (reserved)
data[1] = temperatures 1-8 reporting status (as set by command 19)
data[2] = temperatures 9-15 reporting status (as set by command 19)
data[3] = temperatures 16-23 reporting status (as set by command 19)
data[4] = temperatures 24-32 reporting status (as set by command 19)
data[5] = key presses (as set by command 23)
data[6] = key releases (as set by command 23)
data[7] = ATX Power Switch Functionality (as set by command 28)
data[8] = current watchdog counter (as set by command 29)
data[9] = User Contrast Adjust (as set by command 13, data[1])
data[10] = Key backlight setting (as set by command 14, data[1])
data[11] = atx_sense_on_floppy (as set by command 28)
data[12] = 0 (reserved)
data[13] = CFA633-style contrast setting (as set by command 13, data[0])
data[14] = LCD backlight setting (as set by command 14, data[0])
```

Please Note: Future firmware versions may return fewer or additional bytes.

31 (0x1F): Send Data to LCD

This command allows data to be placed at any position on the LCD.

```
type: 0x1F = 31₁₀
data_length: 3 to 18
data[0]: col = x = 0 to 15
data[1]: row = y = 0 to 1
data[2-21]: text to place on the LCD, variable from 1 to 16 characters
```

The return packet will be:

```
type: 0x40 | 0x1F = 0x5F = 95₁₀
data_length: 0
```

Send Data to LCD is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

33 (0x21): Set Baud Rate

This command will change the CFA533's baud rate. The CFA533 will send the acknowledge packet for this command and then change its baud rate to the new value. The host should send the baud rate command, wait for a positive acknowledge from the CFA533 at the old baud rate, and then switch itself to the new baud rate. The baud rate must be saved by the command 4 (0x04): Store Current State as Boot State (Pg. 36) if you want the CFA533 to power up at the new baud rate.

The factory default baud rate is 19200.

```
type: 0x21 = 33₁₀
data_length: 1
data[0]:
       0 = 19200 baud
       1 = 115200 baud
```

The return packet will be:

```
type: 0x40 | 0x21 = 0x61 = 97₁₀
data_length: 0
```

34 (0x22): Set/Configure GPIO

The CFA533 has five pins for user-definable general-purpose input / output (GPIO). These pins are shared with the DOW and ATX functions. Be careful when you configure the GPIO if you want to use the ATX or DOW at the same time.

The architecture of the CFA533 allows great flexibility in the configuration of the GPIO pins. They can be set as input or output. They can output constant high or low signals or a variable duty cycle 100 Hz PWM signal. (PWM Pulse Width Modulation is a way to simulate intermediate levels by switching a level between full on and full off. PWM is typically used to control the brightness of LED backlights, relying on the natural averaging done by the human eye.)

In output mode using the PWM (and a suitable current limiting resistor), an LED may be turned on or off and even dimmed under host software control. With suitable external circuitry, the GPIOs can also be used to drive external logic or power transistors.

The CFA533 continuously polls the GPIOs as inputs at 32 Hz. The present level can be queried by the host software at a lower rate. The CFA533 also keeps track of whether there were rising or falling edges since the last host query (subject to the resolution of the 32 Hz sampling). This means that the host is not forced to poll quickly in order to detect short events. The algorithm used by the CFA533 to read the inputs is inherently “debounced”

The GPIOs also have “pull-up” and “pull-down” modes. These modes can be useful when using the GPIO as an input connected to a switch since no external pull-up or pull-down resistor is needed. For instance, the GPIO can be set to pull up. Then when a switch connected between the GPIO and ground is open, reading the GPIO will return a "1". When the switch is closed, the input will return a "0".

Pull-up/pull-down resistance values are approximately 5kΩconfirm. Do not exceed current of 25 mA per GPIO.

GPIO[1] may be connected to the host’s power in order to sense the host’s power on/off state. There is a resistor R3 in series with GPIO[1] to limit the possibility of latchup. To use GPIO[1] as a general-purpose input/output, you may need to change R3 with a resistor suitable for your application. It is loaded with a 5.6KΩ resistor that is suitable for most applications.

Note

The GPIO pins may also be used for ATX control through header J8 and temperature sensing through the CFA533’s DOW header. By factory default, the GPIO output setting, function, and drive mode are set correctly to enable operation of the ATX and DOW functions. **The GPIO output setting, function, and drive mode must be set to the correct values in order for the ATX and DOW functions to work. Improper use of this command can disable the ATX and DOW functions.**The cfTest will work with this CFA533 module and may be used to easily check and reset the GPIO configuration to the default state so the ATX and DOW functions will work.

The GPIO configuration is one of the items stored by the command 4 (0x04): Store Current State as Boot State (Pg. 36).

```
type: 0x22 = 34₁₀
data_length:
 2 bytes to change value only
 3 bytes to change value and configure function and drive mode

data[0]: index of GPIO to modify
       0 = GPIO[0] = J8, Pin 7
       1 = GPIO[1] = J8, Pin 6 (may be ATX Host Power Sense, as configured by
                               command 28, data[2])
       2 = GPIO[2] = J8, Pin 5 (default is ATX Host Power Control)
       3 = GPIO[3] = J8, Pin 4 (default is ATX Host Reset Control)
       4 = GPIO[4] = J_DOW, Pin 2 (default is DOW I/O -- has a 1KΩ resistor hardware
           pull-up: R2)
   5-255 = reserved

 Please note: Future versions of this command on future hardware models may accept
              additional values for data[0], which would control the state of future
              additional GPIO pins

data[1]: Pin output state (actual behavior depends on drive mode):
       0 = Output set to low
    1-99 = Output duty cycle percentage (100 Hz nominal)
     100 = Output set to high
 101-255 = invalid

data[2]: Pin function select and drive mode (optional, 0-15 valid)
  ---- FDDD
  ||||  |||-- DDD = Drive Mode (based on output state of 1 or 0)
  ||||  |     =====================================================
  ||||  |     000: 1=Fast, Strong Drive Up, 0=Resistive Pull Down
  ||||  |     001: 1=Fast, Strong Drive Up, 0=Fast, Strong Drive Down
  ||||  |     010: Hi-Z, use for input
  ||||  |     011: 1=Resistive Pull Up,     0=Fast, Strong Drive Down
  ||||  |     100: 1=Slow, Strong Drive Up, 0=Hi-Z
  ||||  |     101: 1=Slow, Strong Drive Up, 0=Slow, Strong Drive Down
  ||||  |     110: reserved, do not use
  ||||  |     111: 1=Hi-Z,                  0=Slow, Strong Drive Down
  ||||  |
  ||||  |----- F = Function
  ||||        =====================================================
  ||||        0: Port unused for GPIO. It will take on the default
  ||||           function such as ATX, DOW or unused. The user is
  ||||           responsible for setting the drive to the correct
  ||||           value in order for the default function to work
  ||||           correctly.
  ||||        1: Port used for GPIO under user control. The user is
  ||||           responsible for setting the drive to the correct
  ||||           value in order for the desired GPIO mode to work
  ||||           correctly.
  ||||------- reserved, must be 0
```

The return packet will be:

```
type: 0x40 | 0x22 = 0x62 = 98₁₀
data_length: 0
```

35 (0x23): Read GPIO Pin Levels and Configuration State

Please see command 34 (0x22): Set/Configure GPIO (Pg. 50) for details on the GPIO architecture.

```
type: 0x23 = 35₁₀
data_length: 1

data[0]: index of GPIO to query
       0 = GPIO[0] = J8, Pin 7
       1 = GPIO[1] = J8, Pin 6 (may be ATX Host Power Sense, as configured
                                by command 28, data[2])
       2 = GPIO[2] = J8, Pin 5 (default is ATX Host Power Control)
       3 = GPIO[3] = J8, Pin 4 (default is ATX Host Reset Control)
       4 = GPIO[4] = J_DOW, Pin 2 (default is DOW I/O -- has a 1KΩ resistor hardware
           pull-up: R2)
   5-255 = reserved

 Please note: Future versions of this command on future hardware models may accept
 additional values for data[0],which would return the status of future additional GPIO
 pins

returns:
  data[0]: index of GPIO read
  data[1]: Pin state & changes since last poll
    ---- -RFS
    |||| ||||-- S = state at the last reading
    |||| |||--- F = at least one falling edge has
    |||| |||        been detected since the last poll
    |||| ||---- R = at least one rising edge has
    |||| ||         been detected since the last poll
    |||| |----- reserved

      (This reading is the actual pin state, which may or may not agree with the pin
       setting, depending on drive mode and the load presented by external circuitry.
       The pins are polled at approximately 32 Hz asynchronously with respect to this
       command.Transients that happen between polls will not be detected.)
  data[2]: Requested Pin level/PWM level
    0-100 = Output duty cycle percentage
      (This value is the requested PWM duty cycle. The actual pin may or may not be
       toggling in agreement with this value, depending on the drive mode and the load
       presented by external circuitry)
  data[3]: Pin function select and drive mode
    ---- FDDD
    |||| ||||-- DDD = Drive Mode
    |||| |      =======================================================
    |||| |      000: 1=Fast, Strong Drive Up, 0=Resistive Pull Down
    |||| |      001: 1=Fast, Strong Drive Up, 0=Fast, Strong Drive Down
    |||| |      010: Hi-Z, use for input
    |||| |      011: 1=Resistive Pull Up,     0=Fast, Strong Drive Down
    |||| |      100: 1=Slow, Strong Drive Up, 0=Hi-Z
    |||| |      101: 1=Slow, Strong Drive Up, 0=Slow, Strong Drive Down
    |||| |      110: reserved
    |||| |      111: 1=Hi-Z,                  0=Slow, Strong Drive Down
    |||| |
    |||| |----- F = Function
    ||||        =======================================================
    ||||        0: Port unused for GPIO. It will take on the default
    ||||           function such as ATX, DOW or unused. The user is
    ||||           responsible for setting the drive to the correct
    ||||           value in order for the default function to work
    ||||           correctly.
    ||||        1: Port used for GPIO under user control. The user is
    ||||           responsible for setting the drive to the correct
    ||||           value in order for the desired GPIO mode to work
    ||||           correctly.
    ||||------- reserved, will return 0
```

CHARACTER GENERATOR ROM (CGROM)

To find the code for a given character, add the two numbers that are shown in bold for its row and column. For example, the Greek letter "β" is in the column labeled “224d” and in the row labeled “2d”. So 224 + 2 = 226. When you send a byte with the value of 226 to the display, the Greek letter "β" will be shown.

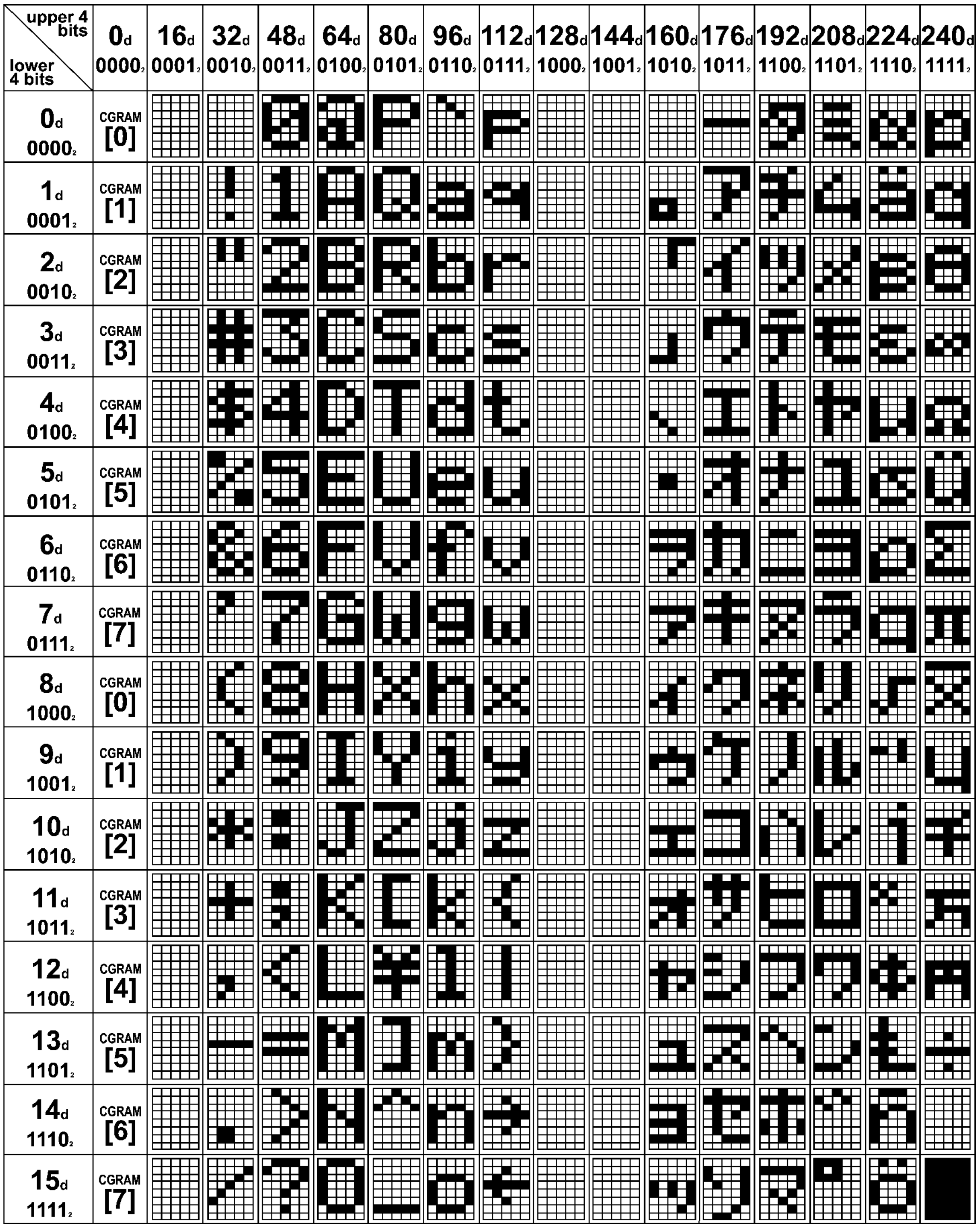

upper 4 bits
lower 4 bits
0d 0000₂
16d 0001₂
32d 0010₂
48d 0011₂
64d 0100₂
80d 0101₂
96d 0110₂
112d 0111₂
128d 1000₂
144d 1001₂
160d 1010₂
176d 1011₂
192d 1100₂
208d 1101₂
224d 1110₂
240d 1111₂
0d 0000₂
1d 0001₂
2d 0010₂
3d 0011₂
4d 0100₂
5d 0101₂
6d 0110₂
7d 0111₂
8d 1000₂
9d 1001₂
10d 1010₂
11d 1011₂
12d 1100₂
13d 1101₂
14d 1110₂
15d 1111₂
CGRAM [0]
CGRAM [1]
CGRAM [2]
CGRAM [3]
CGRAM [4]
CGRAM [5]
CGRAM [6]
CGRAM [7]
CGRAM [0]
CGRAM [1]
CGRAM [2]
CGRAM [3]
CGRAM [4]
CGRAM [5]
CGRAM [6]
CGRAM [7]

Figure 17. Character Generator ROM (CGROM)

DISPLAY MODULE RELIABILITY AND LONGEVITY

Note: We work to continuously improve our products, including backlights that are brighter and last longer. Slight color variations from module to module and batch to batch are normal.

MODULE RELIABILITY

ITEM	SPECIFICATION	
LCD portion (excluding Keypad and Backlights)	50,000 to 100,000 hours (typical)	
Keypad	1,000,000 keystrokes	
White* LED Display and Blue LED Keypad Backlights ** We recommend that the backlight of the white LED back-lit modules be dimmed or turned off during periods of inactivity to conserve the white LED backlight lifetime.*	*Power-On Hours*	*% of Initial Brightness*
	<10,000	>90%
	<50,000	>50%

MODULE LONGEVITY (EOL / REPLACEMENT POLICY)

Crystalfontz is committed to making all of our display modules available for as long as possible. For each module we introduce, we intend to offer it indefinitely. We do not pre-plan a module's obsolescence. The majority of modules we have introduced are still available.

We recognize that discontinuing a module may cause problems for some customers. However, rapidly changing technologies, component availability, or low customer order levels may force us to discontinue ("End of Life", EOL) a module. For example, we must occasionally discontinue a module when a supplier discontinues a component or a manufacturing process becomes obsolete. When we discontinue a module, we will do our best to find an acceptable replacement module with the same fit, form, and function.

In most situations, you will not notice a difference when comparing a "fit, form, and function" replacement module to the discontinued module it replaces. However, sometimes a change in component or process for the replacement module results in a slight variation, perhaps an improvement, over the previous design.

Although the replacement module is still within the stated Datasheet specifications and tolerances of the discontinued module, changes may require modification to your circuit and/or firmware. Possible changes include:

- *Backlight LEDs.* Brightness may be affected (perhaps the new LEDs have better efficiency) or the current they draw may change (new LEDs may have a different VF).
- *Controller.* A new controller may require minor changes in your code.
- *Component tolerances.* Module components have manufacturing tolerances. In extreme cases, the tolerance stack can change the visual or operating characteristics.

Please understand that we avoid changing a module whenever possible; we only discontinue a module if we have no other option. We will post Part Change Notices (PCN) on the product's webpage as soon as possible. If interested, you can subscribe to future part change notifications.

APPENDIX A: CALCULATING THE CRC

ALGORITHMS TO CALCULATE THE CRC

Below are eight sample algorithms that will calculate the CRC of a CFA533 packet. Some of the algorithms were contributed by forum members and originally written for the CFA631 and CFA635. The CRC used in the CFA533 is the same one that is used in IrDA, which came from PPP, which to at least some extent seems to be related to a CCITT (ref: Network Working Group Request for Comments: 1171) standard. At that point, the trail was getting a bit cold and diverged into several referenced articles and papers, dating back to 1983.

The polynomial used is $X^{16} + X^{12} + X^5 + X^0$ (0x8408)
The result is bit-wise inverted before being returned.

Algorithm 1: "C" Table Implementation

This algorithm is typically used on the host computer, where code space is not an issue.

```
//This code is from the IRDA LAP documentation, which appears to
//have been copied from PPP:
//
// http://irda.affiniscape.com/associations/2494/files/Specifications/IrLAP11_Plus_Er-
rata.zip
//
//I doubt that there are any worries about the legality of this code,
//searching for the first line of the table below, it appears that
//the code is already included in the linux 2.6 kernel "Driver for
//ST5481 USB ISDN modem". This is an "industry standard" algorithm
//and I do not think there are ANY issues with it at all.
typedef unsigned char ubyte;
typedef unsigned short word;
word get_crc(ubyte *bufptr,word len)
  {
  //CRC lookup table to avoid bit-shifting loops.
  static const word crcLookupTable[256] =
    {0x00000,0x01189,0x02312,0x0329B,0x04624,0x057AD,0x06536,0x074BF,
     0x08C48,0x09DC1,0x0AF5A,0x0BED3,0x0CA6C,0x0DBE5,0x0E97E,0x0F8F7,
     0x01081,0x00108,0x03393,0x0221A,0x056A5,0x0472C,0x075B7,0x0643E,
     0x09CC9,0x08D40,0x0BFDB,0x0AE52,0x0DAED,0x0CB64,0x0F9FF,0x0E876,
     0x02102,0x0308B,0x00210,0x01399,0x06726,0x076AF,0x04434,0x055BD,
     0x0AD4A,0x0BCC3,0x08E58,0x09FD1,0x0EB6E,0x0FAE7,0x0C87C,0x0D9F5,
     0x03183,0x0200A,0x01291,0x00318,0x077A7,0x0662E,0x054B5,0x0453C,
     0x0BDCB,0x0AC42,0x09ED9,0x08F50,0x0FBEF,0x0EA66,0x0D8FD,0x0C974,
     0x04204,0x0538D,0x06116,0x0709F,0x00420,0x015A9,0x02732,0x036BB,
     0x0CE4C,0x0DFC5,0x0ED5E,0x0FCD7,0x08868,0x099E1,0x0AB7A,0x0BAF3,
     0x05285,0x0430C,0x07197,0x0601E,0x014A1,0x00528,0x037B3,0x0263A,
     0x0DECD,0x0CF44,0x0FDDF,0x0EC56,0x098E9,0x08960,0x0BBFB,0x0AA72,
     0x06306,0x0728F,0x04014,0x0519D,0x02522,0x034AB,0x00630,0x017B9,
     0x0EF4E,0x0FEC7,0x0CC5C,0x0DDD5,0x0A96A,0x0B8E3,0x08A78,0x09BF1,
     0x07387,0x0620E,0x05095,0x0411C,0x035A3,0x0242A,0x016B1,0x00738,
     0x0FFCF,0x0EE46,0x0DCDD,0x0CD54,0x0B9EB,0x0A862,0x09AF9,0x08B70,
     0x08408,0x09581,0x0A71A,0x0B693,0x0C22C,0x0D3A5,0x0E13E,0x0F0B7,
     0x00840,0x019C9,0x02B52,0x03ADB,0x04E64,0x05FED,0x06D76,0x07CFF,
     0x09489,0x08500,0x0B79B,0x0A612,0x0D2AD,0x0C324,0x0F1BF,0x0E036,
     0x018C1,0x00948,0x03BD3,0x02A5A,0x05EE5,0x04F6C,0x07DF7,0x06C7E,
     0x0A50A,0x0B483,0x08618,0x09791,0x0E32E,0x0F2A7,0x0C03C,0x0D1B5,
     0x02942,0x038CB,0x00A50,0x01BD9,0x06F66,0x07EEF,0x04C74,0x05DFD,
     0x0B58B,0x0A402,0x09699,0x08710,0x0F3AF,0x0E226,0x0D0BD,0x0C134,
     0x039C3,0x0284A,0x01AD1,0x00B58,0x07FE7,0x06E6E,0x05CF5,0x04D7C,
     0x0C60C,0x0D785,0x0E51E,0x0F497,0x08028,0x091A1,0x0A33A,0x0B2B3,
     0x04A44,0x05BCD,0x06956,0x078DF,0x00C60,0x01DE9,0x02F72,0x03EFB,
     0x0D68D,0x0C704,0x0F59F,0x0E416,0x090A9,0x08120,0x0B3BB,0x0A232,
     0x05AC5,0x04B4C,0x079D7,0x0685E,0x01CE1,0x00D68,0x03FF3,0x02E7A,
     0x0E70E,0x0F687,0x0C41C,0x0D595,0x0A12A,0x0B0A3,0x08238,0x093B1,
     0x06B46,0x07ACF,0x04854,0x059DD,0x02D62,0x03CEB,0x00E70,0x01FF9,
     0x0F78F,0x0E606,0x0D49D,0x0C514,0x0B1AB,0x0A022,0x092B9,0x08330,
     0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78};

  register word
    newCrc;
  newCrc=0xFFFF;
  //This algorithm is based on the IrDA LAP example.
  while(len--)
    newCrc = (newCrc >> 8) ^ crcLookupTable[(newCrc ^ *bufptr++) & 0xff];

  //Make this crc match the one’s complement that is sent in the packet.
  return(~newCrc);
  }
```

Algorithm 2: “C” Bit Shift Implementation

This algorithm was mainly written to avoid any possible legal issues about the source of the routine (at the request of the LCDproc group). This routine was “clean” coded from the definition of the CRC. It is ostensibly smaller than the table driven approach but will take longer to execute. This routine is offered under the GPL.

```
typedef unsigned char ubyte;
typedef unsigned short word;

word get_crc(ubyte *bufptr,word len)
  {
  register unsigned int
    newCRC;
  //Put the current byte in here.
  ubyte
    data;
  int
    bit_count;
  //This seed makes the output of this shift based algorithm match
  //the table based algorithm. The center 16 bits of the 32-bit
  //"newCRC" are used for the CRC. The MSb of the lower byte is used
  //to see what bit was shifted out of the center 16 bit CRC
  //accumulator ("carry flag analog");
  newCRC=0x00F32100;
  while(len--)
    {
    //Get the next byte in the stream.
    data=*bufptr++;
    //Push this byte’s bits through a software
    //implementation of a hardware shift & xor.
    for(bit_count=0;bit_count<=7;bit_count++)
      {
      //Shift the CRC accumulator
      newCRC>>=1;

      //The new MSB of the CRC accumulator comes
      //from the LSB of the current data byte.
      if(data&0x01)
        newCRC|=0x00800000;

      //If the low bit of the current CRC accumulator was set
      //before the shift, then we need to XOR the accumulator
      //with the polynomial (center 16 bits of 0x00840800)
      if(newCRC&0x00000080)
        newCRC^=0x00840800;
      //Shift the data byte to put the next bit of the stream
      //into position 0.
      data>>=1;
      }
    }
  //All the data has been done. Do 16 more bits of 0 data.
  for(bit_count=0;bit_count<=15;bit_count++)
    {
    //Shift the CRC accumulator
    newCRC>>=1;

    //If the low bit of the current CRC accumulator was set
    //before the shift we need to XOR the accumulator with
    //0x00840800.
    if(newCRC&0x00000080)
      newCRC^=0x00840800;
    }
  //Return the center 16 bits, making this CRC match the one's
  //complement that is sent in the packet.
  return((~newCRC)>>8);
  }
```

Algorithm 2B: "C" Improved Bit Shift Implementation

This is simplified algorithm that implements the CRC.

```
  unsigned short get_crc(unsigned char count,unsigned char *ptr)
    {
    unsigned short
      crc;   //Calculated CRC
    unsigned char
      i;     //Loop count, bits in byte
    unsigned char
      data;  //Current byte being shifted

    crc = 0xFFFF; // Preset to all 1's, prevent loss of leading zeros

    while(count--)
      {
      data = *ptr++;
      i = 8;
      do
        {
        if((crc ^ data) & 0x01)
          {
          crc >>= 1;
          crc ^= 0x8408;
          }
        else
          crc >>= 1;
        data >>= 1;
        } while(--i != 0);
      }
    return (~crc);
    }
```

Algorithm 3: "PIC Assembly" Bit Shift Implementation

This routine was graciously donated by one of our customers.

```
;=====================================================================
; Crystalfontz CFA533 PIC CRC Calculation Example
;
; This example calculates the CRC for the hard coded example provided
; in the documentation.
;
; It uses "This is a test. " as input and calculates the proper CRC
; of 0x93FA.
;=====================================================================
#include "p16f877.inc"
;=====================================================================
; CRC16 equates and storage
;---------------------------------------------------------------------
accuml      equ     40h         ; BYTE - CRC result register high byte
accumh      equ     41h         ; BYTE - CRC result register high low byte
datareg     equ     42h         ; BYTE - data register for shift
j           equ     43h         ; BYTE - bit counter for CRC 16 routine
Zero        equ     44h         ; BYTE - storage for string memory read
index       equ     45h         ; BYTE - index for string memory read
savchr      equ     46h         ; BYTE - temp storage for CRC routine
;
seedlo      equ     021h        ; initial seed for CRC reg lo byte
seedhi      equ     0F3h        ; initial seed for CRC reg hi byte
;
polyL       equ     008h        ; polynomial low byte
polyH       equ     084h        ; polynomial high byte
;=====================================================================
;  CRC Test Program
;---------------------------------------------------------------------
        org         0           ; reset vector = 0000H
;
        clrf        PCLATH      ; ensure upper bits of PC are cleared
        clrf        STATUS      ; ensure page bits are cleared
        goto        main        ; jump to start of program
;
; ISR Vector
;
        org         4           ; start of ISR
        goto        $           ; jump to ISR when coded
;
        org         20          ; start of main program
main
        movlw       seedhi      ; setup intial CRC seed value.
        movwf       accumh      ; This must be done prior to
        movlw       seedlo      ; sending string to CRC routine.
        movwf       accuml      ;
        clrf        index       ; clear string read variables
;
main1
        movlw       HIGH InputStr ; point to LCD test string
        movwf       PCLATH      ; latch into PCL,
        movfw       index       ; get index
        call        InputStr    ; get character
        movwf       Zero        ; setup for terminator test
        movf        Zero,f      ; see if terminator
        btfsc       STATUS,Z    ; skip if not terminator
        goto        main2       ; else terminator reached, jump out of loop
        call        CRC16       ; calculate new crc
        call        SENDUART    ; send data to LCD
        incf        index,f     ; bump index
        goto        main1       ; loop
;
main2
        movlw       00h         ; shift accumulator 16 more bits.
        call        CRC16       ; This must be done after sending
        movlw       00h         ; string to CRC routine.
        call        CRC16       ;
;
```

```
        comf        accumh,f    ; invert result
        comf        accuml,f    ;
;
        movfw       accuml      ; get CRC low byte
        call        SENDUART    ; send to LCD
        movfw       accumh      ; get CRC hi byte
        call        SENDUART    ; send to LCD
;
stop    goto        stop           ; word result of 0x93FA is in accumh/accuml
;=====================================================================
; calculate CRC of input byte
;---------------------------------------------------------------------
CRC16
        movwf       savchr      ; save the input character
        movwf       datareg     ; load data register
        movlw       .8          ; setup number of bits to test
        movwf       j           ; save to incrementor
_loop
        clrc                    ; clear carry for CRC register shift
         rrf       datareg,f    ; perform shift of data into CRC register
        rrf         accumh,f    ;
        rrf         accuml,f    ;
        btfss       STATUS,C    ; skip jump if if carry
        goto        _notset     ; otherwise goto next bit
        movlw       polyL       ; XOR poly mask with CRC register
        xorwf       accuml,F    ;
        movlw       polyH       ;
        xorwf       accumh,F    ;
_notset
        decfsz      j,F         ; decrement bit counter
        goto        _loop       ; loop if not complete
        movfw       savchr      ; restore the input character
        return                  ; return to calling routine
;=====================================================================
; USER SUPPLIED Serial port transmit routine
;---------------------------------------------------------------------
SENDUART
        return                  ; put serial xmit routine here
;=====================================================================
; test string storage
;---------------------------------------------------------------------
        org     0100h
;
InputStr
        addwf   PCL,f
        dt      7h,10h,"This is a test. ",0
;
;=====================================================================
        end
```

Algorithm 4: "Visual Basic" Table Implementation

Visual BASIC has its own challenges as a language (such as initializing static arrays), and it is also challenging to use Visual BASIC to work with "binary" (arbitrary length character data possibly containing nulls—such as the "data" portion of the CFA533 packet) data. This routine was adapted from the C table implementation. The complete project can be found in our forums.

```
'This program is brutally blunt. Just like VB. No apologies.
'Written by Crystalfontz America, Inc. 2004 http://www.crystalfontz.com
'Free code, not copyright copyleft or anything else.
'Some visual basic concepts taken from:
'http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=21434&lngWId=1
'most of the algorithm is from functions in 633_WinTest:
'https://www.crystalfontz.com/product/633WinTest#docs
'Full zip of the project is available in our forum:
'http://www.crystalfontz.com/forum/showthread.php?postid=9921#post9921
```

```
Private Type WORD
   Lo As Byte
   Hi As Byte
End Type

Private Type PACKET_STRUCT
   command As Byte
   data_length As Byte
   data(22) As Byte
   crc As WORD
End Type

Dim crcLookupTable(256) As WORD

Private Sub MSComm_OnComm()
'Leave this here
End Sub

'My understanding of visual basic is very limited--however it appears that there is no way
'to initialize an array of structures. Nice language. Fast processors, lots of memory, big
'disks, and we fill them up with this . . this . . this . . STUFF.
Sub Initialize_CRC_Lookup_Table()
  crcLookupTable(0).Lo = &H0
  crcLookupTable(0).Hi = &H0
  . . .
'For purposes of brevity in this data sheet, I have removed 251 entries of this table, the
'full source is available in our forum:
'http://www.crystalfontz.com/forum/showthread.php?postid=9921#post9921
  . . .
  crcLookupTable(255).Lo = &H78
  crcLookupTable(255).Hi = &HF
End Sub

'This function returns the CRC of the array at data for length positions
Private Function Get_Crc(ByRef data() As Byte, ByVal length As Integer) As WORD
  Dim Index As Integer
  Dim Table_Index As Integer
  Dim newCrc As WORD
  newCrc.Lo = &HFF
  newCrc.Hi = &HFF
  For Index = 0 To length - 1
    'exclusive-or the input byte with the low-order byte of the CRC register
    'to get an index into crcLookupTable
    Table_Index = newCrc.Lo Xor data(Index)
    'shift the CRC register eight bits to the right
    newCrc.Lo = newCrc.Hi
    newCrc.Hi = 0
    ' exclusive-or the CRC register with the contents of Table at Table_Index
    newCrc.Lo = newCrc.Lo Xor crcLookupTable(Table_Index).Lo
    newCrc.Hi = newCrc.Hi Xor crcLookupTable(Table_Index).Hi
  Next Index
  'Invert & return newCrc
  Get_Crc.Lo = newCrc.Lo Xor &HFF
  Get_Crc.Hi = newCrc.Hi Xor &HFF
End Function

Private Sub Send_Packet(ByRef packet As PACKET_STRUCT)
  Dim Index As Integer
  'Need to put the whole packet into a linear array
  'since you can't do type overrides. VB, gotta love it.
  Dim linear_array(26) As Byte
  linear_array(0) = packet.command
  linear_array(1) = packet.data_length
  For Index = 0 To packet.data_length - 1
    linear_array(Index + 2) = packet.data(Index)
  Next Index
  packet.crc = Get_Crc(linear_array, packet.data_length + 2)
```

```
    'Might as well move the CRC into the linear array too
    linear_array(packet.data_length + 2) = packet.crc.Lo
    linear_array(packet.data_length + 3) = packet.crc.Hi
    'Now a simple loop can dump it out the port.
    For Index = 0 To packet.data_length + 3
      MSComm.Output = Chr(linear_array(Index))
    Next Index
  End Sub
```

Algorithm 5: “Java” Table Implementation

This code was posted in our forum by user “norm” as a working example of a Java CRC calculation.

```
  public class CRC16 extends Object
    {
    public static void main(String[] args)
      {
      byte[] data = new byte[2];
      // hw - fw
      data[0] = 0x01;
      data[1] = 0x00;
      System.out.println("hw -fw req");
      System.out.println(Integer.toHexString(compute(data)));

      // ping
      data[0] = 0x00;
      data[1] = 0x00;
      System.out.println("ping");
      System.out.println(Integer.toHexString(compute(data)));

      // reboot
      data[0] = 0x05;
      data[1] = 0x00;
      System.out.println("reboot");
      System.out.println(Integer.toHexString(compute(data)));

      // clear lcd
      data[0] = 0x06;
      data[1] = 0x00;
      System.out.println("clear lcd");
      System.out.println(Integer.toHexString(compute(data)));

      // set line 1
      data = new byte[18];
      data[0] = 0x07;
      data[1] = 0x10;
      String text = "Test Test Test  ";
      byte[] textByte = text.getBytes();
      for (int i=0; i < text.length(); i++) data[i+2] = textByte[i];
      System.out.println("text 1");
      System.out.println(Integer.toHexString(compute(data)));
      }
    private CRC16()
      {
      }
    private static final int[] crcLookupTable =
      {
      0x00000,0x01189,0x02312,0x0329B,0x04624,0x057AD,0x06536,0x074BF,
      0x08C48,0x09DC1,0x0AF5A,0x0BED3,0x0CA6C,0x0DBE5,0x0E97E,0x0F8F7,
      0x01081,0x00108,0x03393,0x0221A,0x056A5,0x0472C,0x075B7,0x0643E,
      0x09CC9,0x08D40,0x0BFDB,0x0AE52,0x0DAED,0x0CB64,0x0F9FF,0x0E876,
      0x02102,0x0308B,0x00210,0x01399,0x06726,0x076AF,0x04434,0x055BD,
      0x0AD4A,0x0BCC3,0x08E58,0x09FD1,0x0EB6E,0x0FAE7,0x0C87C,0x0D9F5,
      0x03183,0x0200A,0x01291,0x00318,0x077A7,0x0662E,0x054B5,0x0453C,
      0x0BDCB,0x0AC42,0x09ED9,0x08F50,0x0FBEF,0x0EA66,0x0D8FD,0x0C974,
      0x04204,0x0538D,0x06116,0x0709F,0x00420,0x015A9,0x02732,0x036BB,
      0x0CE4C,0x0DFC5,0x0ED5E,0x0FCD7,0x08868,0x099E1,0x0AB7A,0x0BAF3,
      0x05285,0x0430C,0x07197,0x0601E,0x014A1,0x00528,0x037B3,0x0263A,
      0x0DECD,0x0CF44,0x0FDDF,0x0EC56,0x098E9,0x08960,0x0BBFB,0x0AA72,
      0x06306,0x0728F,0x04014,0x0519D,0x02522,0x034AB,0x00630,0x017B9,
      0x0EF4E,0x0FEC7,0x0CC5C,0x0DDD5,0x0A96A,0x0B8E3,0x08A78,0x09BF1,
      0x07387,0x0620E,0x05095,0x0411C,0x035A3,0x0242A,0x016B1,0x00738,
      0x0FFCF,0x0EE46,0x0DCDD,0x0CD54,0x0B9EB,0x0A862,0x09AF9,0x08B70,
      0x08408,0x09581,0x0A71A,0x0B693,0x0C22C,0x0D3A5,0x0E13E,0x0F0B7,
      0x00840,0x019C9,0x02B52,0x03ADB,0x04E64,0x05FED,0x06D76,0x07CFF,
      0x09489,0x08500,0x0B79B,0x0A612,0x0D2AD,0x0C324,0x0F1BF,0x0E036,
      0x018C1,0x00948,0x03BD3,0x02A5A,0x05EE5,0x04F6C,0x07DF7,0x06C7E,
      0x0A50A,0x0B483,0x08618,0x09791,0x0E32E,0x0F2A7,0x0C03C,0x0D1B5,
      0x02942,0x038CB,0x00A50,0x01BD9,0x06F66,0x07EEF,0x04C74,0x05DFD,
      0x0B58B,0x0A402,0x09699,0x08710,0x0F3AF,0x0E226,0x0D0BD,0x0C134,
      0x039C3,0x0284A,0x01AD1,0x00B58,0x07FE7,0x06E6E,0x05CF5,0x04D7C,
      0x0C60C,0x0D785,0x0E51E,0x0F497,0x08028,0x091A1,0x0A33A,0x0B2B3,
      0x04A44,0x05BCD,0x06956,0x078DF,0x00C60,0x01DE9,0x02F72,0x03EFB,
      0x0D68D,0x0C704,0x0F59F,0x0E416,0x090A9,0x08120,0x0B3BB,0x0A232,
      0x05AC5,0x04B4C,0x079D7,0x0685E,0x01CE1,0x00D68,0x03FF3,0x02E7A,
      0x0E70E,0x0F687,0x0C41C,0x0D595,0x0A12A,0x0B0A3,0x08238,0x093B1,
      0x06B46,0x07ACF,0x04854,0x059DD,0x02D62,0x03CEB,0x00E70,0x01FF9,
      0x0F78F,0x0E606,0x0D49D,0x0C514,0x0B1AB,0x0A022,0x092B9,0x08330,
      0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78
      };
    public static int compute(byte[] data)
      {
      int newCrc = 0x0FFFF;
      for (int i = 0; i < data.length; i++ )
        {
        int lookup = crcLookupTable[(newCrc ^ data[i]) & 0xFF];
        newCrc = (newCrc >> 8) ^ lookup;
        }
      return(~newCrc);
      }
    }
```

Algorithm 6: “Perl” Table Implementation

This code was translated from the C version by one of our customers.

```
#!/usr/bin/perl

use strict;

my @CRC_LOOKUP =
  (0x00000,0x01189,0x02312,0x0329B,0x04624,0x057AD,0x06536,0x074BF,
   0x08C48,0x09DC1,0x0AF5A,0x0BED3,0x0CA6C,0x0DBE5,0x0E97E,0x0F8F7,
   0x01081,0x00108,0x03393,0x0221A,0x056A5,0x0472C,0x075B7,0x0643E,
   0x09CC9,0x08D40,0x0BFDB,0x0AE52,0x0DAED,0x0CB64,0x0F9FF,0x0E876,
   0x02102,0x0308B,0x00210,0x01399,0x06726,0x076AF,0x04434,0x055BD,
   0x0AD4A,0x0BCC3,0x08E58,0x09FD1,0x0EB6E,0x0FAE7,0x0C87C,0x0D9F5,
   0x03183,0x0200A,0x01291,0x00318,0x077A7,0x0662E,0x054B5,0x0453C,
   0x0BDCB,0x0AC42,0x09ED9,0x08F50,0x0FBEF,0x0EA66,0x0D8FD,0x0C974,
   0x04204,0x0538D,0x06116,0x0709F,0x00420,0x015A9,0x02732,0x036BB,
   0x0CE4C,0x0DFC5,0x0ED5E,0x0FCD7,0x08868,0x099E1,0x0AB7A,0x0BAF3,
   0x05285,0x0430C,0x07197,0x0601E,0x014A1,0x00528,0x037B3,0x0263A,
   0x0DECD,0x0CF44,0x0FDDF,0x0EC56,0x098E9,0x08960,0x0BBFB,0x0AA72,
   0x06306,0x0728F,0x04014,0x0519D,0x02522,0x034AB,0x00630,0x017B9,
   0x0EF4E,0x0FEC7,0x0CC5C,0x0DDD5,0x0A96A,0x0B8E3,0x08A78,0x09BF1,
   0x07387,0x0620E,0x05095,0x0411C,0x035A3,0x0242A,0x016B1,0x00738,
   0x0FFCF,0x0EE46,0x0DCDD,0x0CD54,0x0B9EB,0x0A862,0x09AF9,0x08B70,
   0x08408,0x09581,0x0A71A,0x0B693,0x0C22C,0x0D3A5,0x0E13E,0x0F0B7,
   0x00840,0x019C9,0x02B52,0x03ADB,0x04E64,0x05FED,0x06D76,0x07CFF,
   0x09489,0x08500,0x0B79B,0x0A612,0x0D2AD,0x0C324,0x0F1BF,0x0E036,
   0x018C1,0x00948,0x03BD3,0x02A5A,0x05EE5,0x04F6C,0x07DF7,0x06C7E,
   0x0A50A,0x0B483,0x08618,0x09791,0x0E32E,0x0F2A7,0x0C03C,0x0D1B5,
   0x02942,0x038CB,0x00A50,0x01BD9,0x06F66,0x07EEF,0x04C74,0x05DFD,
   0x0B58B,0x0A402,0x09699,0x08710,0x0F3AF,0x0E226,0x0D0BD,0x0C134,
   0x039C3,0x0284A,0x01AD1,0x00B58,0x07FE7,0x06E6E,0x05CF5,0x04D7C,
   0x0C60C,0x0D785,0x0E51E,0x0F497,0x08028,0x091A1,0x0A33A,0x0B2B3,
   0x04A44,0x05BCD,0x06956,0x078DF,0x00C60,0x01DE9,0x02F72,0x03EFB,
   0x0D68D,0x0C704,0x0F59F,0x0E416,0x090A9,0x08120,0x0B3BB,0x0A232,
   0x05AC5,0x04B4C,0x079D7,0x0685E,0x01CE1,0x00D68,0x03FF3,0x02E7A,
   0x0E70E,0x0F687,0x0C41C,0x0D595,0x0A12A,0x0B0A3,0x08238,0x093B1,
   0x06B46,0x07ACF,0x04854,0x059DD,0x02D62,0x03CEB,0x00E70,0x01FF9,
   0x0F78F,0x0E606,0x0D49D,0x0C514,0x0B1AB,0x0A022,0x092B9,0x08330,
   0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78);

#   our test packet read from an enter key press over the serial line:
#    type: 80        (key press)
#    data_length: 1      (1 byte of data)
#    data = 5

my $type: '80';
my $length = '01';
my $data = '05';
my $packet = chr(hex $type) . chr(hex $length) . chr(hex $data) ;


my $valid_crc = '5584' ;

print "A CRC of Packet ($packet) Should Equal ($valid_crc)\n";

my $crc = 0xFFFF ;

printf("%x\n", $crc);

foreach my $char (split //, $packet)
  {
  # newCrc = (newCrc >> 8) ^ crcLookupTable[(newCrc ^ *bufptr++) & 0xff];
  # & is bitwise AND
  # ^ is bitwise XOR
  # >> bitwise shift right
  $crc = ($crc >> 8) ^ $CRC_LOOKUP[($crc ^ ord($char) ) & 0xFF] ;
  # print out the running crc at each byte
  printf("%x\n", $crc);
  }

# get the complement
$crc = ~$crc ;
$crc = ($crc & 0xFFFF) ;

# print out the crc in hex
printf("%x\n", $crc);
```

Algorithm 7: For PIC18F8722 or PIC18F2685

This code was written by customer Virgil Stamps of ATOM Instrument Corporation for our CFA635 module.

```
; CRC Algorithm for CrystalFontz CFA-635 display (DB535)
; This code written for PIC18F8722 or PIC18F2685
;
; Your main focus here should be the ComputeCRC2 and
; CRC16_ routines
;
;===================================================================
ComputeCRC2:
        movlb   RAM8
        movwf   dsplyLPCNT      ;w has the byte count
nxt1_dsply:
        movf    POSTINC1,w
        call    CRC16_
        decfsz  dsplyLPCNT
        goto    nxt1_dsply
        movlw   .0               ; shift accumulator 16 more bits
        call    CRC16_
        movlw   .0
        call    CRC16_
        comf    dsplyCRC,F       ; invert result
        comf    dsplyCRC+1,F
        return
;===================================================================
CRC16_ movwf:
        dsplyCRCData             ; w has byte to crc
        movlw   .8
        movwf   dsplyCRCCount
_cloop:
        bcf     STATUS,C         ; clear carry for CRC register shift
        rrcf    dsplyCRCData,f   ; perform shift of data into CRC
                                 ;register
        rrcf    dsplyCRC,F
        rrcf    dsplyCRC+1,F
        btfss   STATUS,C         ; skip jump if carry
        goto    _notset          ; otherwise goto next bit
        movlw   0x84
        xorwf   dsplyCRC,F
        movlw   0x08             ; XOR poly mask with CRC register
        xorwf   dsplyCRC+1,F
_notset:
        decfsz  dsplyCRCCount,F  ; decrement bit counter
        bra _cloop               ; loop if not complete
        return
;===================================================================
; example to clear screen
dsplyFSR1_TEMP  equ     0x83A    ; 16-bit save for FSR1 for display
                                 ; message handler
dsplyCRC        equ     0x83C    ; 16-bit CRC (H/L)
dsplyLPCNT      equ     0x83E    ; 8-bit save for display message
                                 ; length - CRC
dsplyCRCData    equ     0x83F    ; 8-bit CRC data for display use
dsplyCRCCount   equ     0x840    ; 8-bit CRC count for display use
SendCount       equ     0x841    ; 8-bit byte count for sending to
                                 ; display
RXBUF2          equ     0x8C0    ; 32-byte receive buffer for
                                 ; Display
TXBUF2          equ     0x8E0    ; 32-byte transmit buffer for
                                 ; Display
;-------------------------------------------------------------------
ClearScreen:
        movlb   RAM8
        movlw   .0
        movwf   SendCount
        movlw   0xF3
        movwf   dsplyCRC         ; seed ho for CRC calculation
        movlw   0x21
        movwf   dsplyCRC+1       ; seen lo for CRC calculation
        call    ClaimFSR1
        movlw   0x06
        movwf   TXBUF2
        LFSR    FSR1,TXBUF2
        movf    SendCount,w
        movwf   TXBUF2+1         ; message data length
        call    BMD1
        goto    SendMsg
;===================================================================
; send message via interrupt routine. The code is made complex due
; to the limited FSR registers and extended memory space used
;
```

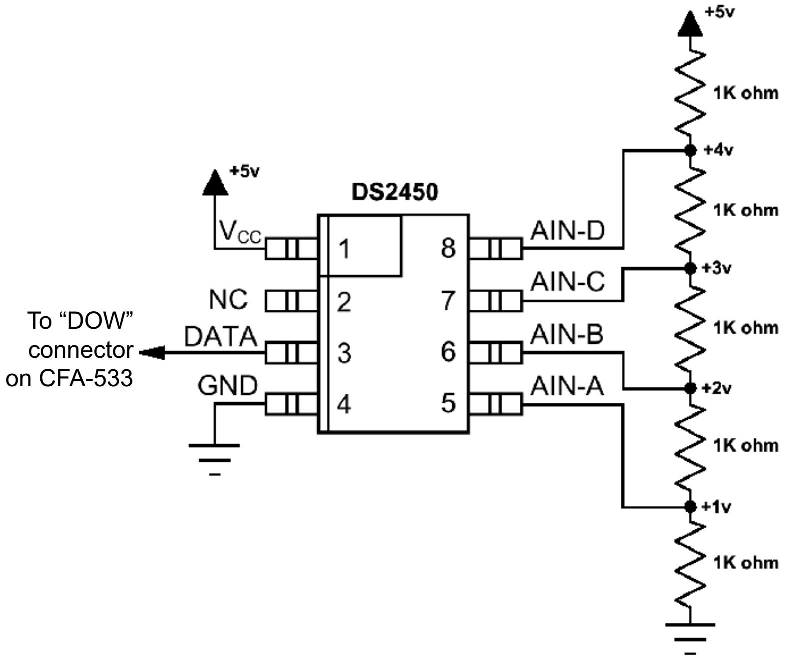
```
; example of sending a string to column 0, row 0
;--------------------------------------------------------------------
SignOnL1:
        call    ClaimFSR1
        lfsr    FSR1,TXBUF2+4     ; set data string position
        SHOW    C0R0,BusName      ; move string to TXBUF2
        movlw   .2                ;
        addwf   SendCount         ;
        movff   SendCount,TXBUF2+1
                                  ; insert message data length
        call    BuildMsgDSPLY
        call    SendMsg
        return
;====================================================================
; BuildMsgDSPLY used to send a string to LCD
;--------------------------------------------------------------------
BuildMsgDSPLY:
        movlw   0xF3
        movwf   dsplyCRC          ; seed hi for CRC calculation
        movlw   0x21
        movwf   dsplyCRC+1        ; seed lo for CRC calculation
        LFSR    FSR1,TXBUF2       ; point at transmit buffer
        movlw   0x1F              ; command to send data to LCD
        movwf   TXBUF2            ; insert command byte from us to
                                  ; CFA-635
        BMD1    movlw .2
        ddwf    SendCount,w       ; + overhead
        call    ComputeCRC2       ; compute CRC of transmit message
        movf    dsplyCRC+1,w
        movwf   POSTINC1          ; append CRC byte
        movf    dsplyCRC,w
        movwf   POSTINC1          ; append CRC byte
        return
;====================================================================
SendMsg:
        call    ReleaseFSR1
        LFSR    FSR0,TXBUF2
        movff   FSR0H,irptFSR0
        movff   FSR0L,irptFSR0+1
                                  ; save interrupt use of FSR0
        movff   SendCount,TXBUSY2
        bsf     PIE2,TX2IE
                                  ; set transmit interrupt enable
                                  ; (bit 4)
        return
;====================================================================
; macro to move string to transmit buffer
SHOW macro src, stringname
        call    src
        MOVLF   upper stringname, TBLPTRU
        MOVLF   high stringname, TBLPTRH
        MOVLF   low stringname, TBLPTRL
        call    MOVE_STR
        endm
;====================================================================
MOVE_STR:
        tblrd   *+
        movf    TABLAT,w
        bz      ms1b
        movwf   POSTINC1
        incf    SendCount
        goto    MOVE_STR
ms1b:
        return
;====================================================================
```

APPENDIX B: CONNECTING A DS2450 1-WIRE QUAD A/D CONVERTER

This appendix describes a simple test circuit that demonstrates how to connect a DS2450 4-channel ADC to the CFA-533's DOW (Dallas One Wire - The DS2450 uses the standard Dallas Semiconductor 1-Wire protocol for data transfers) connector. It also gives a sample command sequence to initialize and read the ADC.

Up to 32 DOW devices can be connected to the CFA-533. In this example the DS2450 appears at device index 0. Your software should query the connected devices using command 18 (0x12): Read DWR-DOW-Y17 Temperature Sensors (Pg. 41) to verify the locations and types of DOW devices connected in your application.

Please refer to the DS2450 Data Sheet and the description for command 20 (0x14): Arbitrary 1-Wire (DOW) Transaction (Pg. 43) more information.



Appendix B Figure 1. Test Circuit Schematic

Start 633WinTest (works with CFA-533) and open the Packet Debugger dialog.

Select Command 20 = Arbitrary DOW Transaction, then paste each string below into the data field and send the packet. The response should be similar to what is shown.

```
//Write 0x40 (=64) to address 0x1C (=28) to leave analog circuitry on
//(see page 6 of the data sheet)
<command 20> \000\002\085\028\000\064
<response> C=84(d=0):2E,05,22  //16 bit "i-button" CRC + 8-bit "DOW" CRC
                               //Consult "i-button" docs to check 16-bit CRC
                               //DOW CRC is probably useless for this device.

//Write all 8 channels of control/status (16 bits, 5.10v range)
<command 20> \000\002\085\008\000\000 // address = 8, channel A low
<response> C=84(d=0):6F,F1,68         // 16-bits, output off

<command 20> \000\002\085\009\000\001 // address = 9, channel A high
<response> C=84(d=0):FF,F1,AB         // no alarms, 5.1v

<command 20> \000\002\085\010\000\000 // address = 10, channel B low
<response> C=84(d=0):CE,31,88         // 16-bits, output off

<command 20> \000\002\085\011\000\001 // address = 11, channel B high
<response> C=84(d=0):5E,31,4B         // no alarms, 5.1v

<command 20> \000\002\085\012\000\000 // address = 12, channel C low
<response> C=84(d=0):2E,30,A3         // 16-bits, output off

<command 20> \000\002\085\013\000\001 // address = 13, channel C high
<response> C=84(d=0):BE,30,60         // no alarms, 5.1v

<command 20> \000\002\085\014\000\000 // address = 14, channel D low
<response> C=84(d=0):8F,F0,43         // 16-bits, output off

<command 20> \000\002\085\015\000\001 // address = 15, channel D high
<response> C=84(d=0):1F,F0,80         // no alarms, 5.1v

//Read all 4 channels of control/status (check only)
<command 20> \000\010\170\008\000
<response> C=84(d=0):00,01,00,01,00,01,00,01,E0,CF,01

//Repeat next two commands for each conversion (two cycles shown)

//Start conversion on all channels
<command 20> \000\002\060\015\000
<response> C=84(d=0):3A,03,28

//Read all 8 channels
<command 20> \000\010\170\000\000
<response> C=84(d=0):00,33,DF,64,84,96,6A,C8,5A,6B,BE

//Decoded response:
0x3300 = 13056 1.016015625 volts (channel A)
0x64DF = 25823 2.009541321 volts (channel B)
0x9684 = 38532 2.998553467 volts (channel C)
0xC86A = 51306 3.992623901 volts (channel D)

//Start conversion on all channels
<command 20> \000\002\060\015\000
<response> C=84(d=0):3A,03,28

//Read all 8 channels
<command 20> \000\010\170\000\000
<response> C=84(d=0):6B,33,B2,64,97,96,42,C8,0F,C9,0A

//Decoded response:
0x336B = 13163 1.024342346 volts (channel A)
0x64B2 = 25778 2.006039429 volts (channel B)
0x9697 = 38551 3.000032043 volts (channel C)
0xC842 = 51266 3.989511108 volts (channel D)
```