



Crystalfontz America, Incorporated

INTELLIGENT USB LCD MODULE SPECIFICATIONS



Crystalfontz Model Number	CFA533-YYH-KU
Hardware Version	Revision 1.0, March 2010
Firmware Version	Revision u1.0, April 2010
Data Sheet Version	Revision 2.0, December 2010
Product Pages	http://www.crystalfontz.com/product/CFA533-YYH-KU

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



REVISION HISTORY

HARDWARE	
2010/06/06	Hardware version: v1.0 (revision number has not changed) Changes: We are transitioning to an improved keypad from “10.5” millimeters to “12.00” millimeters high. See PCN 10282 .
2010/03/18	Hardware version: v1.0 Changes since hardware version (v0.1): As the inventory of hardware version v0.1 depletes, we are phasing in the improved v1.0. The v1.0 should be a drop-in replacement for any v0.1 application. 1. Added a JPUSBSENSE jumper to help user configuration. See Jumper Locations and Functions. 2. Changed JPFPG to RFG. RFG connects 1MΩ between logic ground and frame ground. See Jumper Locations and Functions. 3. PCB layout changes: <ul style="list-style-type: none">- Added more breaks to the frame ground (FG) trace that surrounds the board so there is not a loop.- Added masking on FG trace where connectors may be hand soldered to ease assembly.- Moved vias from under the zebra near the ends of the bezel to assure no shorts between the vias and the bezel.- Made mounting hole annular ring (pads) larger to avoid component damage by tools during assembly.- Resistors are loaded wherever needed. 4. Module weight increased from “40” grams to “41” grams.
2007/12/31	Hardware version: v0.1 New module.

FIRMWARE	
2010/05/15	Current firmware version: u1.0 Changes since last version (u0.x→u1.0): Firmware version now matches all modules that are hardware version v1.0.
2009/04/15	Firmware version: u0.x→u1.0 New firmware.



DATA SHEET

2010/12/16

Current Data Sheet version: **v2.0**

Changes since Data Sheet version (v1.1):

- Wherever listed or shown, changed keypad from “10.5” millimeters to “12.00” millimeters high. This transition started 2010/06/06. See [PCN 10282](#).
- Wherever listed, deleted dash (“-”) from module part numbers to match how they now appear on our website without the dash (“”).
- In [Features \(Pg. 8\)](#), added “RoHS compliant”, “Factories are ISO certified”, and that materials are in compliance with EU Directive “REACH”.
- In [Ordering Information \(Pg. 10\)](#) section,
 - Changed part numbers for I²C variants of CFA-533 modules to end in “-KC” instead of “-KI”.
 - For I²C variants, deleted note “Semi-custom part number and minimum order may apply.” Modules may be ordered through the website using its standard part number, no minimum required.
- In [Physical Characteristics \(Pg. 11\)](#),
 - Changed module depth dimension to include keypad “12.00” millimeters high (formerly 10.5 millimeters high).
 - Module weight increased from “40” grams to “41” grams.
- Updated information in [Jumper Locations and Functions \(Pg. 14\)](#) due to hardware revision v1.0. Added a JPUSBSENSE jumper to help user configuration. Changed JFIG to RFG. RFG connects 1MΩ between logic ground and frame ground. Confirm: mention this for all interfaces?
- Added [OPTICAL CHARACTERISTICS \(Pg. 17\)](#) section, with definition of viewing angles.
- In [Temperature and Humidity Ratings \(Pg. 19\)](#), added Humidity Range specification.
- Expanded and improved information on all power and control connections. See [CONNECTION INFORMATION \(Pg. 21\)](#), particularly
- Because in hardware version 1.0, resistors are loaded by default, all references to “resistors not loaded” were removed.
- In command [8 \(0x08\): Set LCD Contents, Line 2 \(Pg. 38\)](#), corrected screen display text from
data[] = TOP line's display content (must supply 16 bytes)
to
data[] = BOTTOM line's display content (must supply 16 bytes)
- In command [11 \(0x0B\): Set LCD Cursor Position \(Pg. 39\)](#), corrected from
data[0] = column (0-19 valid)
data[1] = row (0-3 valid)
to
data[0] = column (0-15 valid)
data[1] = row (0-1 valid)
- In command [12 \(0x0C\): Set LCD Cursor Style \(Pg. 40\)](#), corrected from “3 = blinking block plus underscore” to “3 = blinking underscore”. This behavior is not the same as the CFA633 series which is “3 = blinking block plus underscore”.
- In command [22 \(0x16\): Send Command Directly to the LCD Controller \(Pg. 45\)](#), corrected “The Neotec [NT7070B](#) controller on the CFA-533 is **S6A0073** compatible.” to “The controller on the CFA-533 is a Neotec [NT7070B](#) (**HD44780** compatible).”
- In command [33 \(0x21\): Set Baud Rate \(Pg. 50\)](#), corrected from “data_length: 1” to “data_length: 0”. Command has not changed.
- In command [35 \(0x23\): Read GPIO Pin Levels and Configuration State \(Pg. 52\)](#)
 - Replaced upper case “X” with lower case “x”.
 - Corrected “data_length” from “4” to “1”.
- Added more information in [CARE AND HANDLING PRECAUTIONS \(Pg. 56\)](#).



DATA SHEET (Continued)	
2010/12/16	<ul style="list-style-type: none">● In APPENDIX B: SAMPLE CODE AND CALCULATING THE CRC (Pg. 64),<ul style="list-style-type: none">- In Sample Code (Pg. 64), added hyperlinks to free downloadable sample code available on our website.- In sample code for Algorithm 1: "C" Table Implementation (Pg. 64) and Algorithm 2: "C" Bit Shift Implementation (Pg. 65), added typedefs for "ubyte" and "word".- Added sample code Algorithm 7: For PIC18F8722 or PIC18F2685 (Pg. 73).● In addition to list above, wherever needed, made minor changes in text and illustrations to improve clarity.
2009/06/20	Data Sheet version: v1.1 Changes since last revision (v1.0): <ul style="list-style-type: none">● Note about semi-custom part number and minimum order in Ordering Information table was deleted. I²C can be ordered directly from our website.● Section on ATX POWER SUPPLY POWER AND CONTROL CONNECTIONS modified for clarification.● For command 33 (0x21): Set Baud Rate, corrected data[1] to data[0].● In section on POWER CONNECTIONS THROUGH USB, corrected part number link for the Tyco Electronics/Amp connector.● Corrected explanation for CHARACTER GENERATOR ROM (CGROM).● Minor wording and formatting changes.
2009/04/15	Data Sheet version: v1.0 New Data Sheet.

The Fine Print

Certain applications using CrystalFontz America, Inc. products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications"). CRYSTALFONTZ AMERICA, INC. PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. Inclusion of CrystalFontz America, Inc. products in such applications is understood to be fully at the risk of the customer. In order to minimize risks associated with customer applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazard. Please contact us if you have any questions concerning potential risk applications.

CrystalFontz America, Inc. assumes no liability for applications assistance, customer product design, software performance, or infringements of patents or services described herein. Nor does CrystalFontz America, Inc. warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of CrystalFontz America, Inc. covering or relating to any combination, machine, or process in which our products or services might be or are used.

The information in this publication is deemed accurate but is not guaranteed.

Company and product names mentioned in this publication are trademarks or registered trademarks of their respective owners.

Copyright © 2010 by CrystalFontz America, Inc., 12412 East Saltese Avenue, Spokane Valley, WA 99216-0357 U.S.A.



CONTENTS

MAIN FEATURES	8
Comparison to CFA633	8
Features	8
Module Classification Information	9
Ordering Information	10
MECHANICAL SPECIFICATIONS	11
Physical Characteristics	11
LCD Module Outline Drawings	12
Jumper Locations and Functions	14
Keypad Detail Drawing	15
Panel Mounting Application Cutout Drawing	16
OPTICAL CHARACTERISTICS	17
ELECTRICAL SPECIFICATIONS	18
System Block Diagram	18
Supply Voltages and Current	19
Temperature and Humidity Ratings	19
ESD (Electro-Static Discharge) Specifications	20
CONNECTION INFORMATION	21
Overview of Connection Information	21
Standard (5+v) and Data Communications Connections	21
1. Standard (+5v) Power Supply and USB Data Communications through USB Connector	22
2. Standard (+5v) Connection through J_PWR Connector	23
3. Standard (+5v) Connection through J8 Connector	24
ATX Power Supply and Control Connection for Host Power Sense	24
ATX Power Supply Connection	24
ATX Control Connections for Host Power Sense	25
ATX Keypad Control	29
GPIO Connections	30
Dallas Semiconductor 1-Wire Device Connections for Optional Accessories	31
Temperature Sensors	31
Other 1-Wire Devices	32
HOST COMMUNICATIONS	32
Packet Structure	32
About Handshaking	34
Report Codes	34
0x80: Key Activity	34
0x82: Temperature Sensor Report	35
Command Codes	35
0 (0x00): Ping Command	35
1 (0x01): Get Hardware & Firmware Version	35
2 (0x02): Write User Flash Area	36
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	37



CONTENTS, CONTINUED

6 (0x06): Clear LCD Screen	38
7 (0x07): Set LCD Contents, Line 1	38
8 (0x08): Set LCD Contents, Line 2	38
9 (0x09): Set LCD Special Character Data	39
10 (0x0A): Read 8 Bytes of LCD Memory	39
11 (0x0B): Set LCD Cursor Position	39
12 (0x0C): Set LCD Cursor Style	40
13 (0x0D): Set LCD Contrast	40
14 (0x0E): Set LCD & Keypad Backlight	41
18 (0x12): Read DOW Device Information	41
19 (0x13): Set Up Temperature Reporting	42
20 (0x14): Arbitrary DOW Transaction	44
21 (0x15): Set Up Live Temperature Display	44
22 (0x16): Send Command Directly to the LCD Controller	45
23 (0x17): Configure Key Reporting	45
24 (0x18): Read Keypad, Polled Mode	46
28 (0x1C): Set ATX Switch Functionality	47
29 (0x1D): Enable/Feed Host Watchdog Reset	49
30 (0x1E): Read Reporting/ATX/Watchdog (debug)	49
31 (0x1F): Send Data to LCD	50
33 (0x21): Set Baud Rate	50
34 (0x22): Set/Configure GPIO	51
35 (0x23): Read GPIO Pin Levels and Configuration State	52
CHARACTER GENERATOR ROM (CGROM)	54
LCD MODULE RELIABILITY AND LONGEVITY	55
Module Reliability	55
Module Longevity (EOL / Replacement Policy)	55
CARE AND HANDLING PRECAUTIONS	56
APPENDIX A: QUALITY ASSURANCE STANDARDS	58
APPENDIX B: SAMPLE CODE AND CALCULATING THE CRC	64
Sample Code	64
Algorithms to Calculate the CRC	64
Algorithm 1: "C" Table Implementation	64
Algorithm 2: "C" Bit Shift Implementation	65
Algorithm 2B: "C" Improved Bit Shift Implementation	66
Algorithm 3: "PIC Assembly" Bit Shift Implementation	67
Algorithm 4: "Visual Basic" Table Implementation	69
Algorithm 5: "Java" Table Implementation	70
Algorithm 6: "Perl" Table Implementation	72
Algorithm 7: For PIC18F8722 or PIC18F2685	73
APPENDIX C: CONNECTING A DS2450 1-WIRE QUAD A/D CONVERTER	76



LIST OF FIGURES

Figure 1. CFA533-YYH-KU Module Outline Drawing (two pages below) -----	12
Figure 2. Jumper Locations and Functions -----	14
Figure 3. Keypad Detail Drawing -----	15
Figure 4. Panel Mounting Application Cutout Drawing -----	16
Figure 5. Definition of 6:00 O'clock and 12:00 O'clock Viewing Angles -----	17
Figure 6. System Block Diagram -----	18
Figure 7. CFA533-YYH-KU Current Usage -----	20
Figure 8. Standard (+5v) Power Supply and USB Data Communications through USB Connector -----	22
Figure 9. Standard (+5v) Connection through J_PWR Connector -----	23
Figure 10. Standard (+5v) Connection through J8 Connector -----	24
Figure 11. ATX Power Supply and Host Power Sense through PWR on USB Connector via JPUSBSENSE -----	25
Figure 12. ATX Host Power Sense through +5v on J_PWR Connector -----	26
Figure 13. ATX Host Power Sense through GPIO[1] on J8 Connector -----	27
Figure 14. ATX Power Supply and Control Using Crystalfontz WR-PWR-Y14 Cable -----	28
Figure 15. Location of GPIO Connections, Resistors, and J_DOW -----	31
Figure 16. Character Generator ROM (CGROM) -----	54



MAIN FEATURES

COMPARISON TO CFA633

If your project does not need the fan connections, the CFA533 series is an economical replacement for the CFA633 series. In November 2010, we issued a Technical Bulletin that describes the differences between the CFA533 series and three versions of the CFA633 series. Please see [Part Change Notice #10291](#).

FEATURES

- 16 characters x 2 lines LCD with keypad and high-level interface. 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.
- The CFA533 series is mechanically similar to the the [CFA633](#) series. The CFA533 series command set is compatible with the [CFA633](#) series. The CFA533 can be used as a “drop-in” replacement for most [CFA633](#) series applications that do not need fan capabilities.
- Full-speed USB interface.
- Edge-lit yellow-green LED backlight with positive yellow-green STN LCD. Displays dark characters on yellow background. Integrated yellow LED backlit 6-button translucent silicon keypad.
- Positive mode display is sunlight readable and also readable in dark areas.
- Fully decoded keypad: any key combination is valid and unique.
- Robust packet-based communications protocol with 16-bit CRC.
- Nonvolatile memory capability (EEPROM):
 - Customize the “power-on” display settings.
 - 16-byte “scratch” register for storing data such as IP address, netmask, system serial number . . .
- Optional capabilities. CrystalFontz can make these modifications for you. A semi-custom part number and minimum order may apply.
 - ATX power supply control functionality allows the buttons on the CFA533-YYH-KU 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 Dallas 1-Wire 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.
 - RS-232 to Dallas Semiconductor 1-Wire bridge functionality allows control of other 1-Wire compatible devices such as ADC, voltage monitoring, current monitoring, RTC, GPIO, counters, and identification/encryption. (Additional hardware required.)
- RoHS compliant.
- Factories have ISO certification.
- Product materials are in compliance with the regulations related to the EU Directive 2006/121/EC for Registration, Evaluation, Authorization and Restriction of Chemicals (REACH).



- ❑ If you need a CE or MET (UL type) approved LCD module, please see our XES635 USB series (<http://www.crystalfontz.com/product/XES635BK-YYE-KU.html>, <http://www.crystalfontz.com/product/XES635BK-TMF-KU.html>, or <http://www.crystalfontz.com/product/XES635BK-TFE-KU.html>). The CFA533-YYH-KU does not have CE or MET (UL type) certification because it is not an end product. The module requires power and communications from another system in order to operate.









MODULE CLASSIFICATION INFORMATION

$$\frac{\text{CFA}}{\text{①}} - \frac{\text{533}}{\text{②}} - \frac{\text{Y}}{\text{③}} \frac{\text{Y}}{\text{④}} \frac{\text{H}}{\text{⑤}} - \frac{\text{K}}{\text{⑥}} \frac{\text{U}}{\text{⑦}}$$

①	Brand	Crystalfontz America, Inc.
②	Model Identifier	533
③	Backlight Type & Color	Y – LED, yellow-green
④	Fluid Type, Image (positive or negative), & LCD Glass Color	Y – STN, positive yellow-green
⑤	Polarizer Film Type, Temperature Range, & View Angle (O’Clock)	H – Transflective, Wide Temperature Range ¹ , 6:00
⑥	Special Code 1	K – Manufacturer’s code
⑦	Special Code 2	U – USB interface
¹ Wide Temperature Range is -20°C minimum to +70°C maximum		



ORDERING INFORMATION

PART NUMBER	FLUID	LCD GLASS COLOR	IMAGE	POLARIZER FILM	BACKLIGHT COLOR/TYPE
CFA533-YYH-KU (USB)	STN	yellow-green	positive	transflective	LCD: yellow-green edge LED Keypad: yellow-green LEDs 
CFA533-TMI-KC (I ² C)	STN	blue	negative	transmissive	LCD: white edge LEDs Keypad: blue LEDs 
CFA533-TMI-KL ("logic-level" RS-232)	STN	blue	negative	transmissive	LCD: white edge LEDs Keypad: blue LEDs 
CFA533-TMI-KS ("full swing" RS-232)	STN	blue	negative	transmissive	LCD: white edge LEDs Keypad: blue LEDs 
CFA533-TMI-KU (USB)	STN	blue	negative	transmissive	LCD: white edge LEDs Keypad: blue LEDs 
CFA533-YYH-KC (I ² C)	STN	yellow-green	positive	transflective	LCD: yellow-green edge LEDs Keypad: yellow-green LEDs 
CFA533-YYH-KL ("logic-level" RS-232)	STN	yellow-green	positive	transflective	LCD: yellow-green edge LEDs Keypad: yellow-green LEDs 
CFA533-YYH-KS ("full swing" RS-232)	STN	yellow-green	positive	transflective	LCD: yellow-green edge LEDs Keypad: yellow-green LEDs 



MECHANICAL SPECIFICATIONS

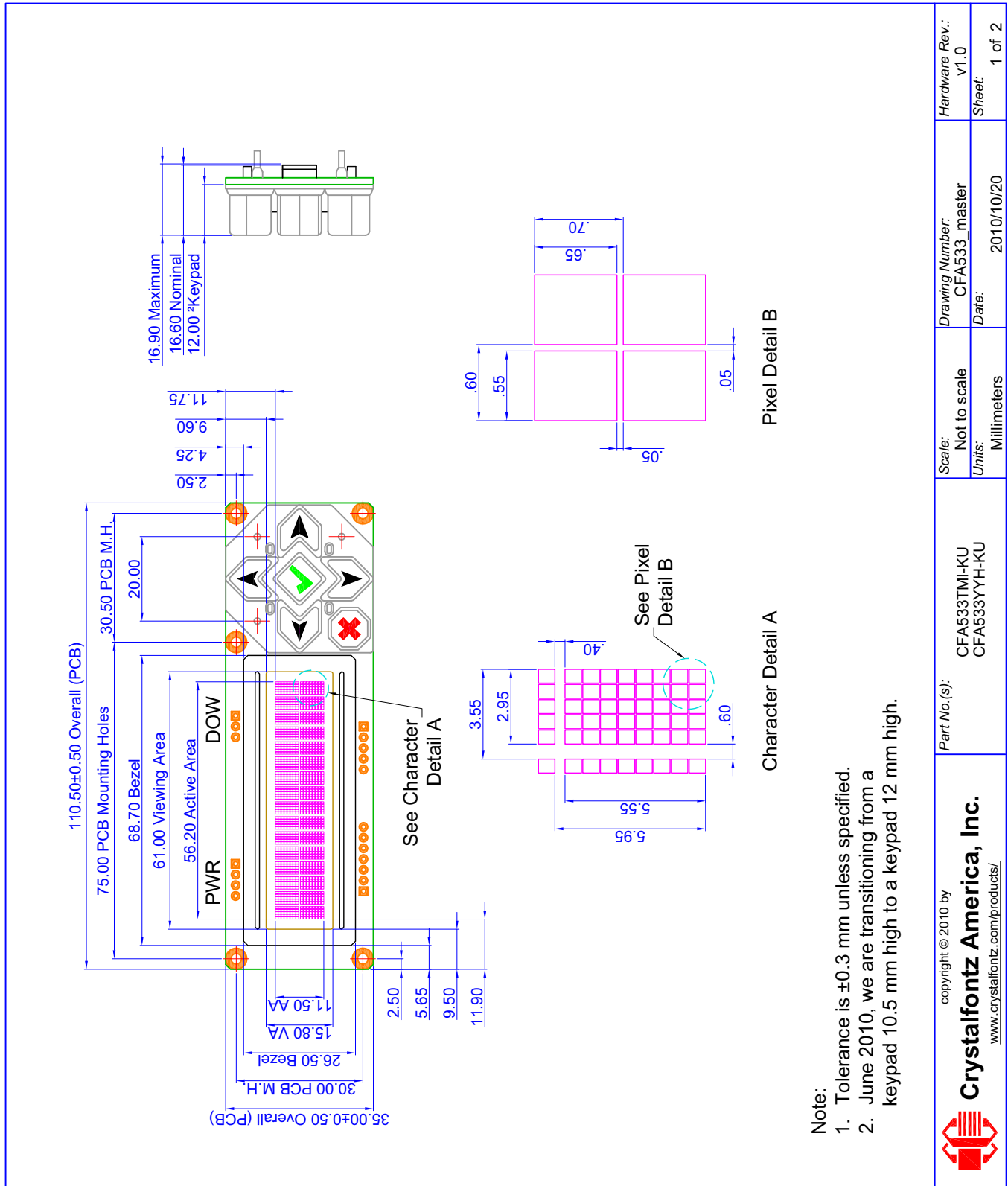
PHYSICAL CHARACTERISTICS

ITEM	SIZE
LCD Module Overall Dimensions	
Width and Height	110.5 (W) x 35.0 (H)
Depth with Keypad	16.60 mm nominal 16.90 mm maximum
Viewing Area	61.0 (W) x 15.8 (H) mm
Active Area	56.2 (W) x 11.5 (H) mm
Character Size	2.95 (W) x 5.55 (H) mm
Character Pitch	3.55 (W) x 5.95 (H) mm
Pixel Size	0.55 (W) x 0.65 (H) mm
Pixel Pitch	0.60 (W) x 0.70 (H) mm
Keystroke Travel (approximate)	2.4 mm
Weight	41 grams (typical)



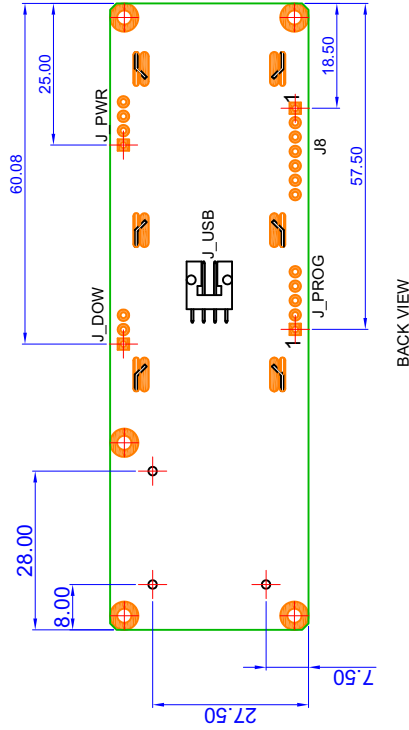
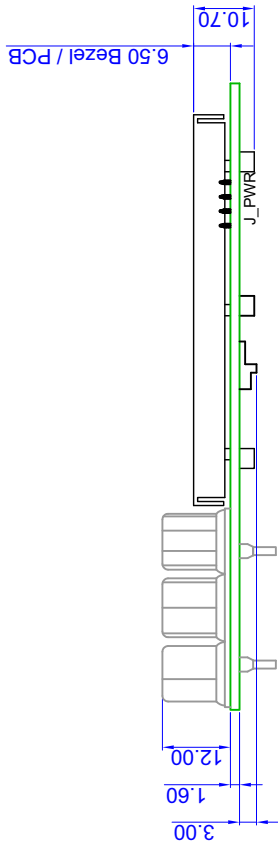
LCD MODULE OUTLINE DRAWINGS

Figure 1. CFA533-YYH-KU Module Outline Drawing (two pages below)



Note:
 1. Tolerance is ±0.3 mm unless specified.
 2. June 2010, we are transitioning from a keypad 10.5 mm high to a keypad 12 mm high.


 copyright © 2010 by CrystalFontz America, Inc. www.crystalfontz.com/products/	Part No.(s): CFA533TMI-KU CFA533YYH-KU	Scale: Not to scale Units: Millimeters	Drawing Number: CFA533_master Date: 2010/10/20	Hardware Rev.: v1.0 Sheet: 1 of 2
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------	-------------------------------------------	------------------------------------------------------	-----------------------------------------



BACK VIEW

Note:

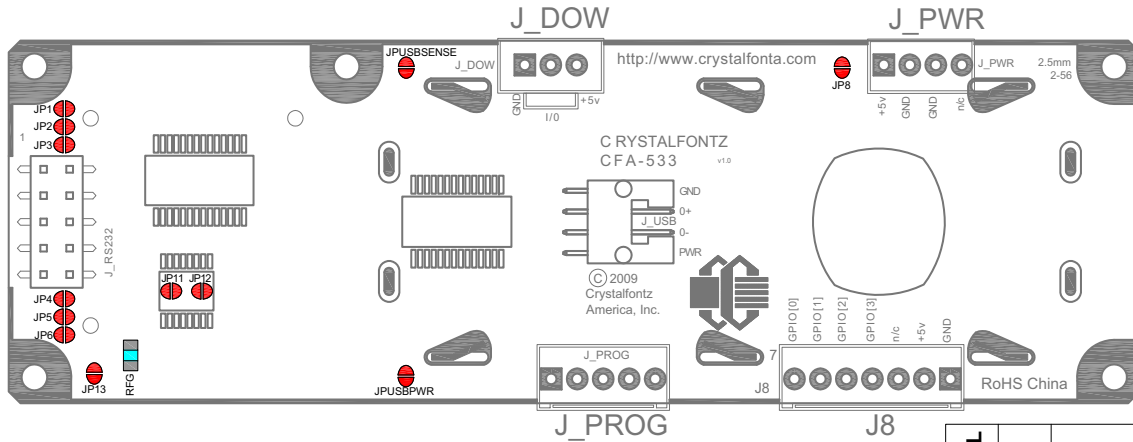
1. Tolerance is ± 0.3 mm unless specified.
2. June 2010, we are transitioning from a keypad 10.5 mm high to a keypad 12 mm high.

 copyright © 2010 by Crystalfontz America, Inc. www.crystalfontz.com/products/	Part No.(s): CFA533-TMI-KU CFA533-YYH-KU	Scale: Not to scale Units: Millimeters	Drawing Number: CFA533_master Date: 2010/10/27	Hardware Rev.: V1.0 Sheet: 2 of 2
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------	-------------------------------------------------	---------------------------------------------------------	--------------------------------------------



JUMPER LOCATIONS AND FUNCTIONS

The CFA533-YYH-KU has thirteen jumpers. Not all jumpers are used by all interfaces. Jumpers may be closed by melting a ball of solder across their gap. You may re-open the jumpers by removing the solder. Use solder wick to remove solder.



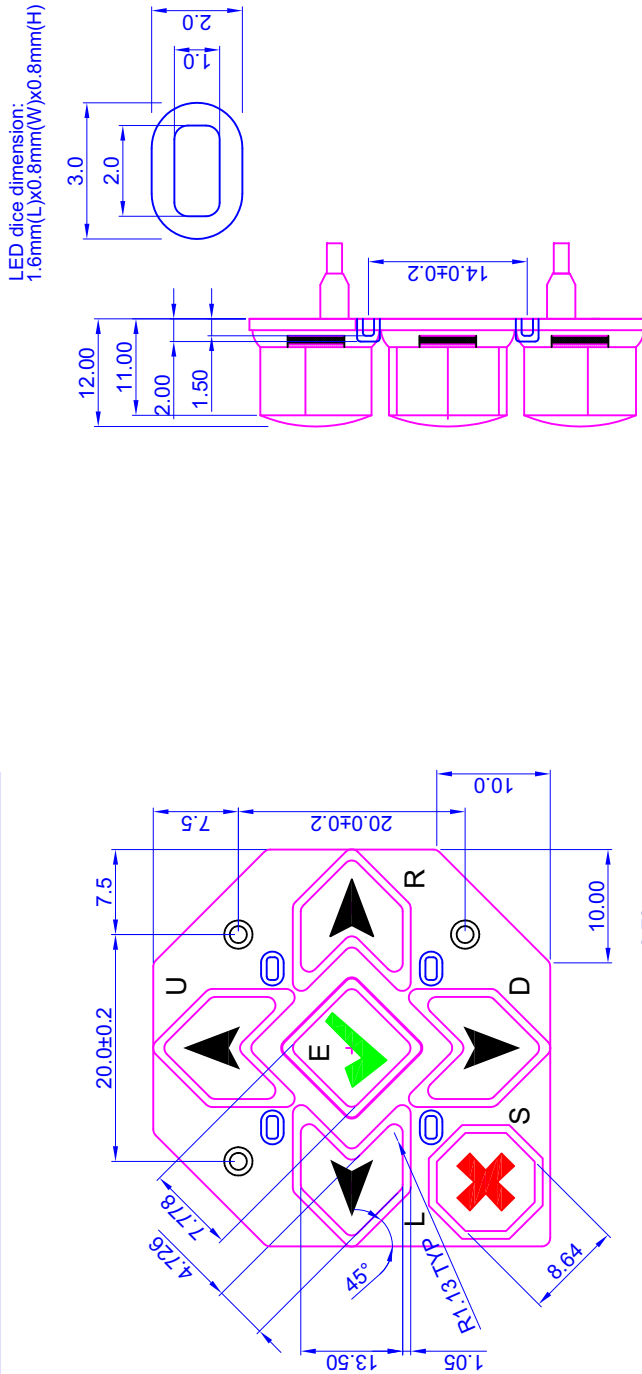
JUMPER	DEFAULT	FUNCTION	LOGIC LEVEL SERIAL		
			KL	KS	KU
JP1	open closed	J1 (RS232), Pin 10 is open LCD Tx/Host Rx to J1 (RS232), Pin 10	X	X	N/A
JP2	open closed	J1 (RS232), Pin 2 is open (see also JP3 and JP11) LCD Tx/Host Rx to J1(RS232), Pin 2	X	X	N/A
JP3	open closed	J1 (RS232), Pin 2 is open (see also JP2 and JP11) LCD Rx/Host Tx to J1 (RS232), Pin 2	X	X	N/A
JP4	open closed	J1 (RS232), Pin 3 is open (see also JP5 and JP12) LCD Rx/Host Tx to J1 (RS232), Pin 3	X	X	N/A
JP5	open closed	J1 (RS232), Pin 3 is open (see also JP4 and JP12) Ground to J1 (RS232), Pin 3	X	X	N/A
JP6	open closed	J1 (RS232), Pin 5 is open Ground to J1 (RS232), Pin 5	X	X	N/A
JP8	open closed	+5v pin from PWR connector used only for ATX SENSE +5v pin from PWR connector supplies power to module	X	X	X
JP13	open closed	J1 (RS232), Pin 4 is open +5v to J1 (RS232), Pin 4	X	X	N/A
JPUSBPWR	open closed	Module power independent of USB Module power supplied from USB	N/A	N/A	X
JPUSBSNSE	open closed	No function +5v from USB is fed to processor's ATX SENSE	N/A	N/A	X
RFG	1MΩ 0Ω open	Mounting holes and FG trace surrounding PCB are discharged slowly to LOGIC GND. Mounting holes and FG trace are connected to LOGIC GND Mounting holes and FG trace surrounding PCB are open	X	X	X
JP11, JP12	N/A	Factory built options. Do not change. Not visible on KL and KS series.	X	X	N/A

Figure 2. Jumper Locations and Functions



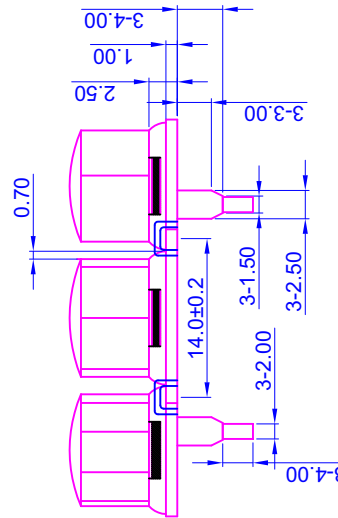
KEYPAD DETAIL DRAWING

June 2010, we are transitioning from a keypad 10.5 mm high to a keypad 12 mm high. PCN at <http://www.crystalfontz.com/products/pcn.phtml?id=10282>



Notes:

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



<p>copyright © 2010 by Crystalfontz America, Inc. www.crystalfontz.com/products/</p>	Part No.(s):	CFA533 Keypad Detail	Scale:	Not to scale	Drawing Number:	Keypad_master	Hardware Rev.:	V1.1
			Units:	Millimeters	Date:	2010/10/14	Sheet:	1 of 1

Figure 3. Keypad Detail Drawing



PANEL MOUNTING APPLICATION CUTOUT DRAWING

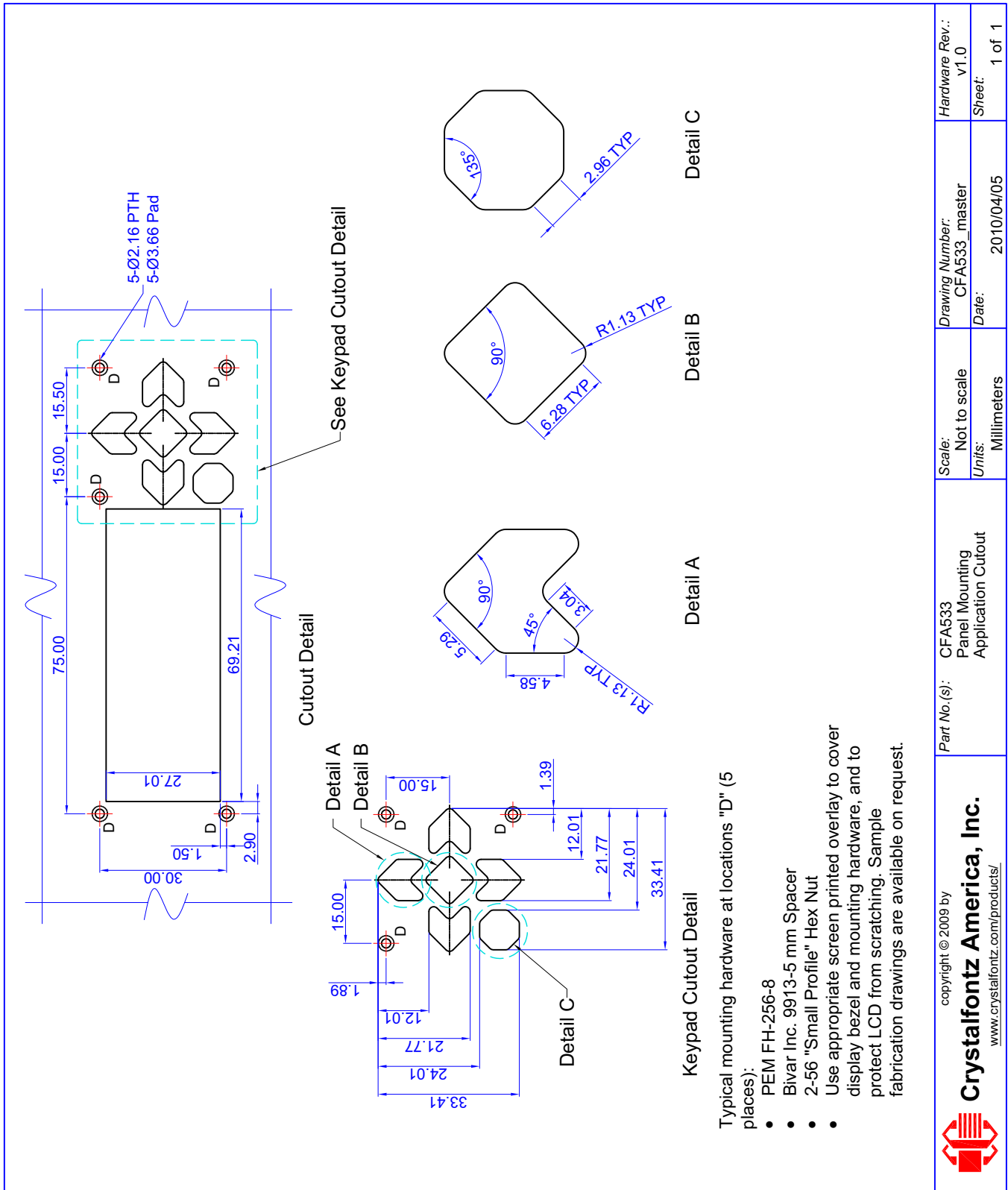


Figure 4. Panel Mounting Application Cutout Drawing



OPTICAL CHARACTERISTICS

Viewing Direction	6 o'clock
-------------------	-----------

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

This LCD module has a 6:00 o'clock viewing angle. A 6:00 o'clock viewing angle is a bottom viewing angle like what you would see when you look at a cell phone or calculator. A 12:00 o'clock viewing angle is a top viewing angle like what you would see when you look at the gauges in a golf cart or airplane.

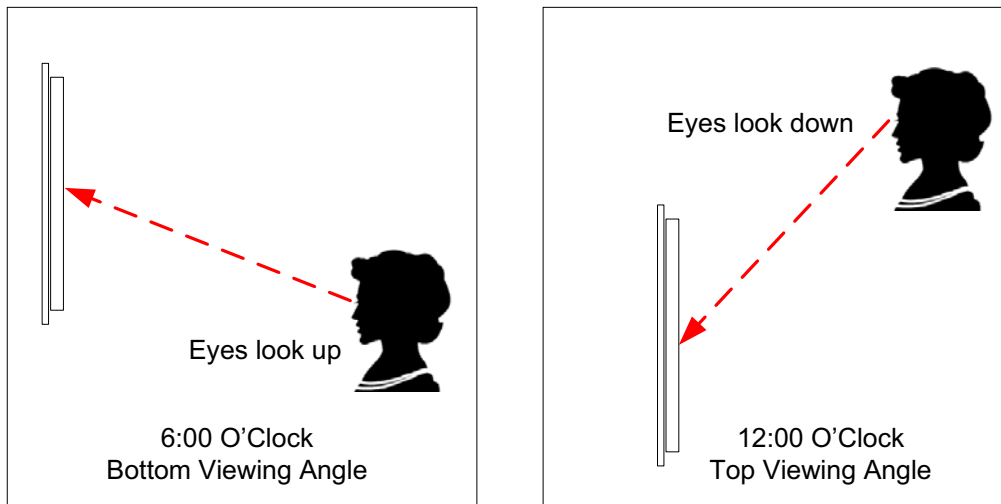


Figure 5. Definition of 6:00 O'clock and 12:00 O'clock Viewing Angles



ELECTRICAL SPECIFICATIONS

SYSTEM BLOCK DIAGRAM

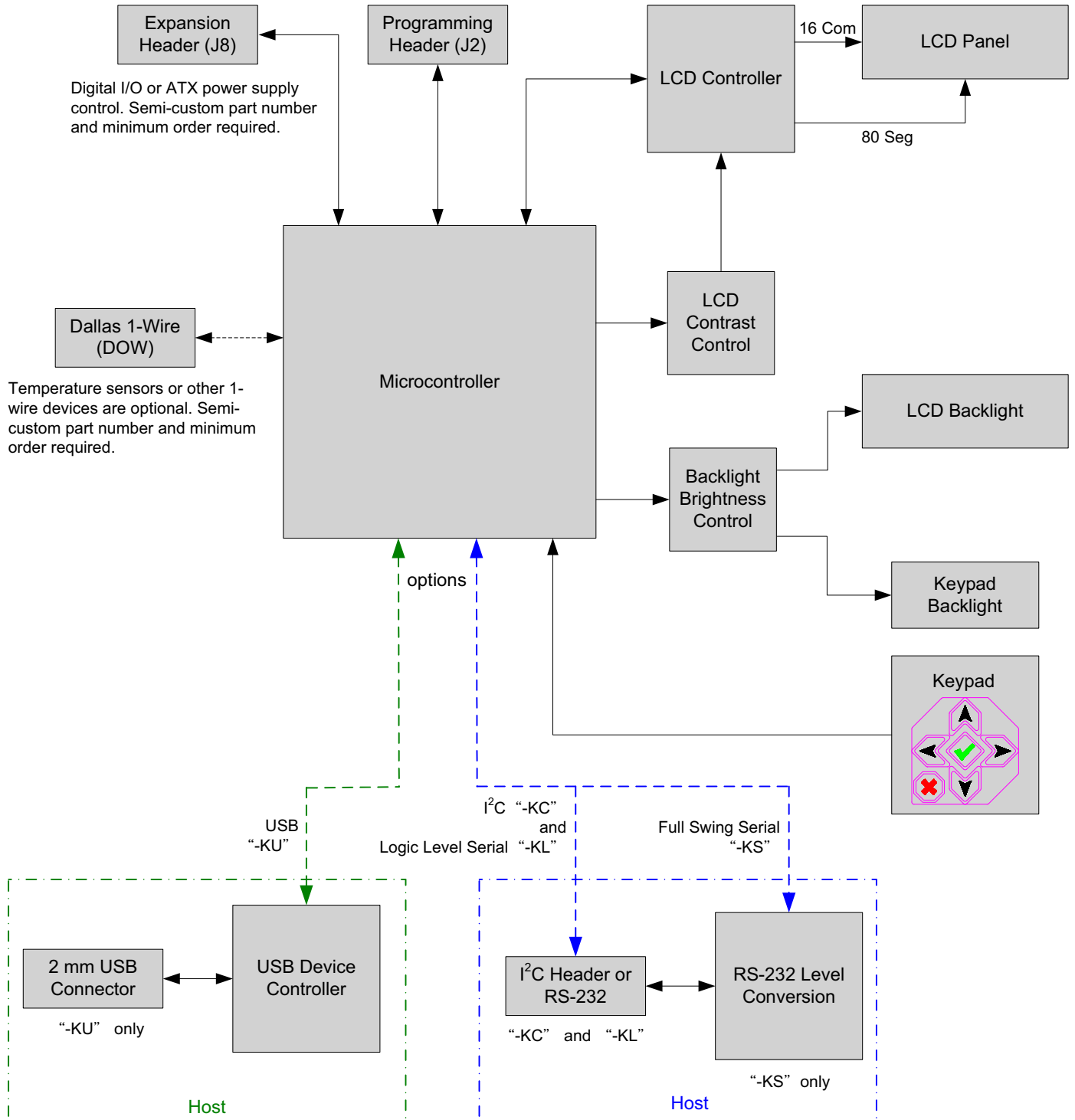


Figure 6. System Block Diagram



SUPPLY VOLTAGES AND CURRENT

DRIVING METHOD	SPECIFICATION
Duty	1/16

SUPPLY VOLTAGE	MINIMUM	MAXIMUM
Power Supply Voltage (V_{DD})	+3.3v	+5.5v
Pull-in Voltage		+3.2v
Drop-out Voltage		+3.0v

TYPICAL CURRENT CONSUMPTION	SPECIFICATION
+5v for logic (LCD + microcontroller)	< 20mA
+5v for logic (LCD + microcontroller) + YYH backlight	< 120mA

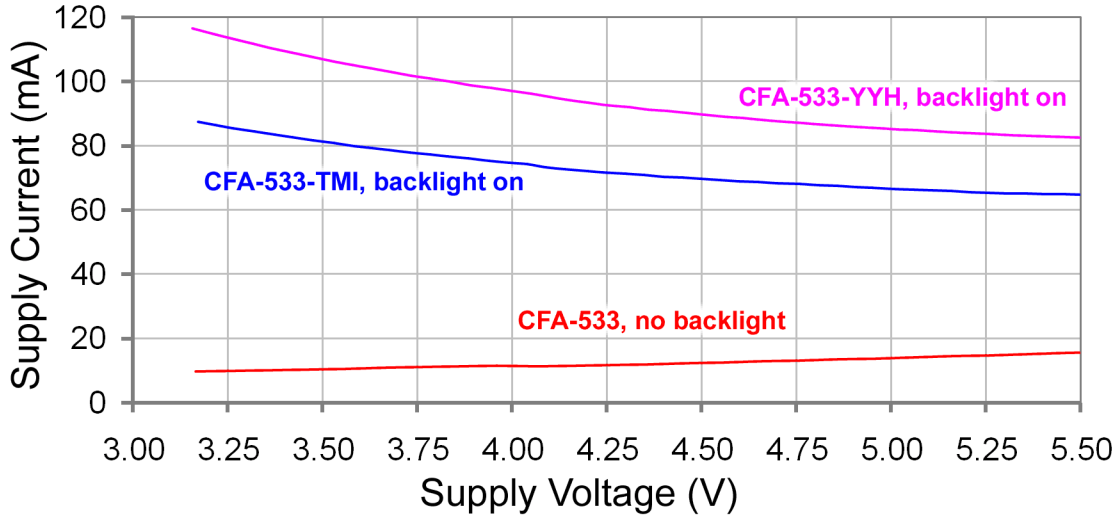
GPIO CURRENT LIMITS	SPECIFICATION
Sink	25 mA
Source	10 mA

TEMPERATURE AND HUMIDITY RATINGS

ABSOLUTE MAXIMUM RATINGS	SYMBOL	MINIMUM	MAXIMUM
Operating Temperature	T_{OP}	0°C	+50°C
Storage Temperature	T_{ST}	-10°C	+60°C
Humidity Range (noncondensing)	RH	10%	90%



CFA 533 supply current vs supply voltage (typical)



CFA 533-YYH supply current vs backlight setting (typical)

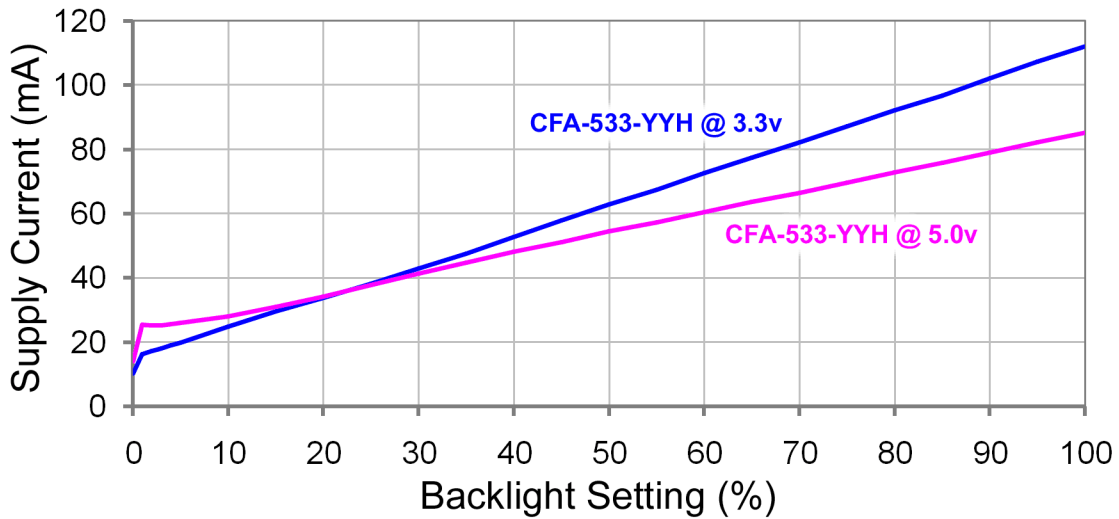


Figure 7. CFA533-YYH-KU Current Usage

ESD (ELECTRO-STATIC DISCHARGE) SPECIFICATIONS

The circuitry is industry standard CMOS logic and susceptible to ESD damage. Please use industry standard antistatic 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.



CONNECTION INFORMATION

OVERVIEW OF CONNECTION INFORMATION

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

The host power supply can power the CFA533-YYH-KU 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.

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

[Standard \(5+v\) and Data Communications Connections \(Pg. 21\)](#)

1. [Standard \(+5v\) Power Supply and USB Data Communications through USB Connector \(Pg. 22\)](#)
2. [Standard \(+5v\) Connection through J_PWR Connector \(Pg. 23\)](#)
3. [Standard \(+5v\) Connection through J8 Connector \(Pg. 24\)](#)

[ATX Power Supply and Control Connection for Host Power Sense \(Pg. 24\)](#)

[ATX Power Supply Connection \(Pg. 24\)](#)

1. [Host Power Sense through PWR on USB Connector via JPUSBSENSE \(Pg. 25\)](#)
 2. [Host Power Sense through +5v on J_PWR Connector \(Pg. 26\)](#)
 3. [Host Power Sense through GPIO\[1\] on J8 Connector \(Pg. 27\)](#)
- [ATX Keypad Control \(Pg. 29\)](#)

[GPIO Connections \(Pg. 30\)](#)

[Dallas Semiconductor 1-Wire Device Connections for Optional Accessories \(Pg. 31\)](#)

1. [Temperature Sensors \(Pg. 31\)](#)
2. [Other 1-Wire Devices \(Pg. 32\)](#)

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 [Jumper Locations and Functions \(Pg. 14\)](#). The table lists the open/close defaults for all jumpers.

STANDARD (5+V) AND DATA COMMUNICATIONS CONNECTIONS

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

1. Standard (+5v) Power Supply and Data Communications through USB Connector
2. Standard (+5v) Connection through J_PWR Connector
3. Standard (+5v) Connection through J8 Connector



1. Standard (+5v) Power Supply and USB Data Communications through USB Connector

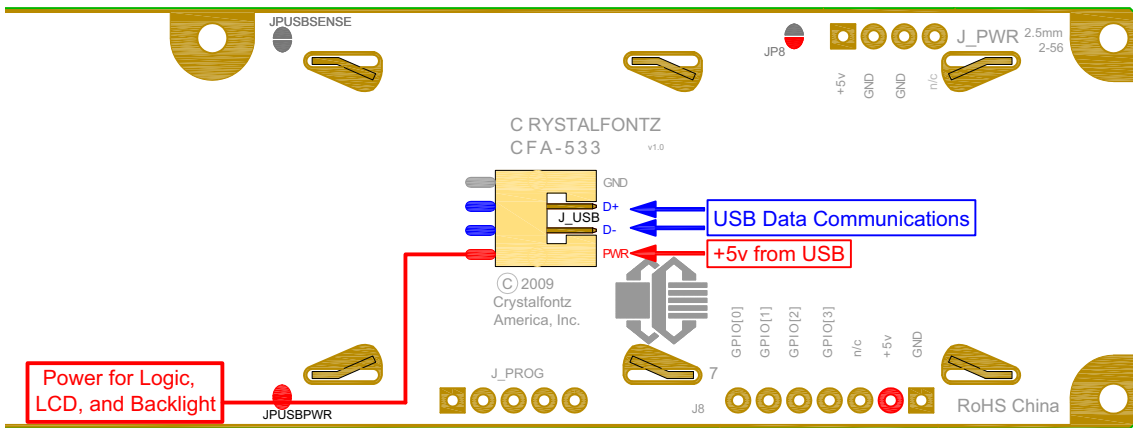


Figure 8. Standard (+5v) Power Supply and USB Data Communications through USB Connector

By using the USB connector, the CFA533-YYH-KU requires only this one connection to the host for both data communications and power supply.

JUSBPWR is closed by default. This allows the USB interface to control the power to the LCD module. **When using this connection method, the CFA533-YYH-KU will stay powered off through the host's boot cycle and will only power up after the host's USB subsystem is fully powered up. This is normal behavior.**

In order to keep the CFA533-YYH-KU as thin as possible, the CFA533-YYH-KU uses a very low profile 2 mm latching polarized connector for USB connection.

Crystalfontz Cables for USB Connection

Crystalfontz sells two cables that make the connection between the CFA533-YYH-KU and the host.

1. The [WR-USB-Y03](#) (6-foot) has the mating 2 mm connector on one end and a standard "USB A" on the other end.
2. The [WR-USB-Y11](#) (27-inch) has the mating 2 mm connector on one end and standard single pin connectors on the opposite end. These single pin connectors are suitable to plug directly onto the USB headers typically found on motherboards.

Make Your Own Cable for USB Connection

If you would like to make your own cable, the connector on the CFA533-YYH-KU is:

FCI/Berg 95000-004: SMT 2 mm connector, 4-position, polarized

The mating housing and terminals for the cable are:

FCI/Berg 90312-004: Housing, 2 mm connector, 4-position, polarized

FCI/Berg 77138-001: Terminal (4 pieces required)

If you want the CFA533-YYH-KU to power up instantly when the host powers up, you can modify the LCD modules or have Crystalfontz modify them for you.



2. Standard (+5v) Connection through J_PWR Connector

By configuring this way, the CFA533-YYH-KU will turn on as soon as +5v is supplied to J_PWR.

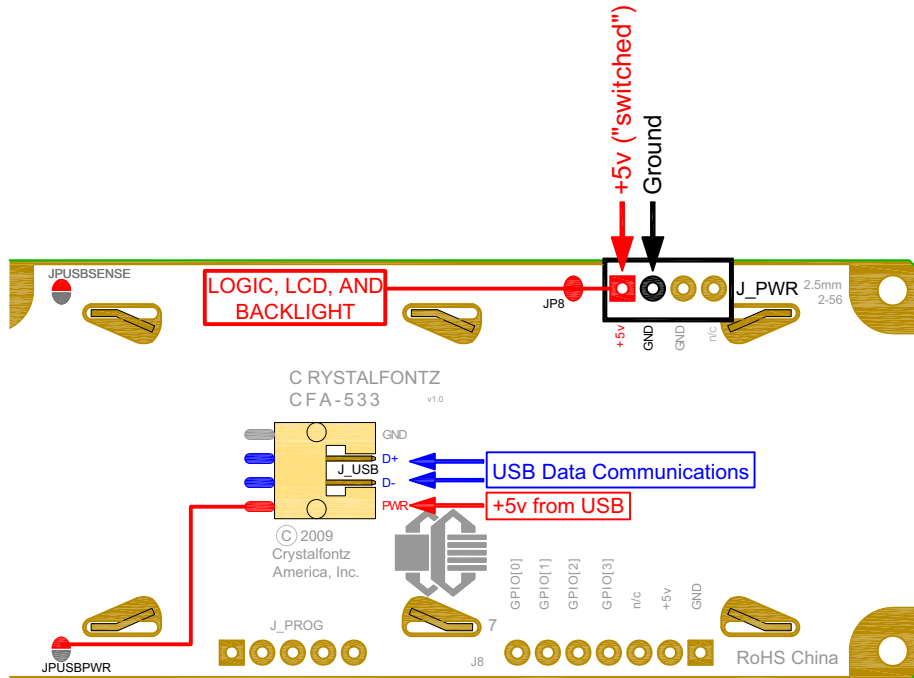


Figure 9. Standard +5v Connection through J_PWR Connector

1. Open JPUSBPWR. (JPUSBPWR is closed by default.)
2. Close JP8. (JP8 is open by default.)
3. Install the appropriate connector in J_PWR. (See [Tyco Electronics / Amp part number 4-171825-4](#), [Mouser Electronics part number 571-4-171825-4](#).)
4. Connect power from the host's power supply to the CFA533-YYH-KUs J_PWR connector. Use a Crystalfontz cable [WR-PWR-Y12](#) or equivalent.



3. Standard (+5v) Connection through J8 Connector

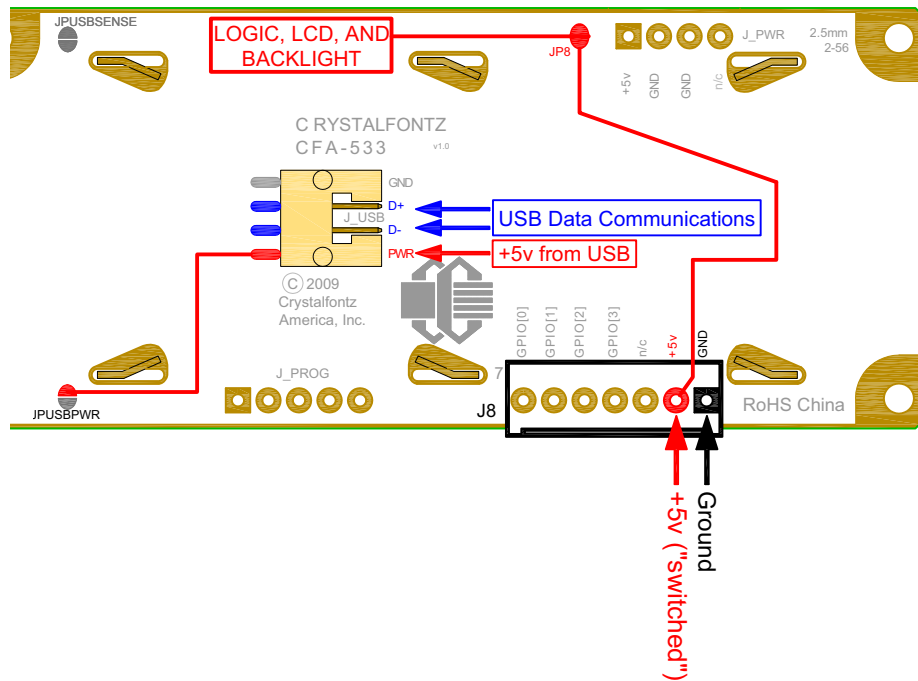


Figure 10. Standard (+5v) Connection through J8 Connector

1. Open JPUSBPWR. (JPUSBPWR is closed by default.)
2. Close JP8. (JP8 is open by default.)
3. Provide power through J8 connector on pin 1 (Ground) and pin 2 (+5v).

Modification by Crystalfontz

We can configure the LCD modules for you by opening JPUSBPWR, closing JP8, and loading your choice of header. The modules will be ready to use in your application without modification by you. 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-YYH-KU has the ability to control power on/off and reset functions of an ATX power supply.

NOTE

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



For this functionality, the CFA533-YYH-KU is powered from the host's V_{SB} signal (V_{SB} is the standby power which is always-on +5v ATX power supply output).

ATX Control Connections for Host Power Sense

For ATX control, choose one of these three connection methods described below:

1. Host Power Sense through PWR on USB Connector via JPUSBSSENSE
2. Host Power Sense through +5v on J_PWR Connector
3. Host Power Sense through GPIO[1] on J8 Connector

1. Host Power Sense through PWR on USB Connector via JPUSBSSENSE

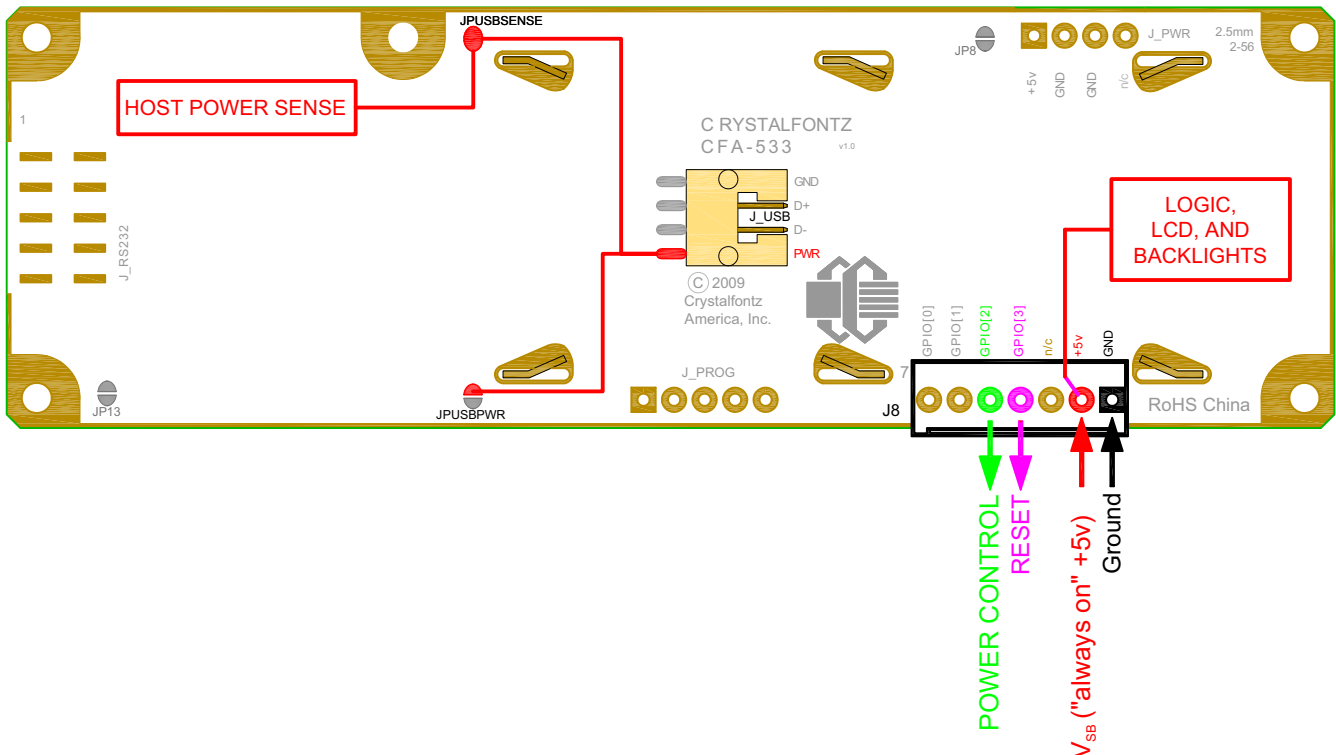


Figure 11. ATX Power Supply and Host Power Sense through PWR on USB Connector via JPUSBSSENSE

By default, the pin labeled PWR on the CFA533-YYH-KU's USB connector is electrically connected to +5v pin on the J8 connector through the normally closed JPUSBPWR. If you want to use the CFA533-YYH-KU to do ATX power supply control, open jumper JPUSBPWR, which disconnects the PWR on the USB connector from +5v of the J8 connector. The +5v pin of the J8 connector will function as V_{SB} (standby power which is always-on) to the LCD module. This completes the ATX power supply/host power sense configuration by providing +5v through the 5K Ω 0805 SMT resistor R3 to the microcontroller. See location of R3 in [Figure 15. on Pg. 31](#).

The motherboard's reset switch input is connected to Pin 4 of the CFA533-YYH-KU connector J8 (labeled as GPIO[3]). This pin functions as RESET. The RESET pin is configured as a high-impedance input until the LCD module wants to reset the host. Then it will change momentarily to low impedance output, driving either low or high depending on the setting of RESET_INVERT. (See command [28 \(0x1C\): Set ATX Switch Functionality \(Pg. 47\)](#).) This connection is also used for the hardware watchdog.



2. Host Power Sense through +5v on J_PWR Connector

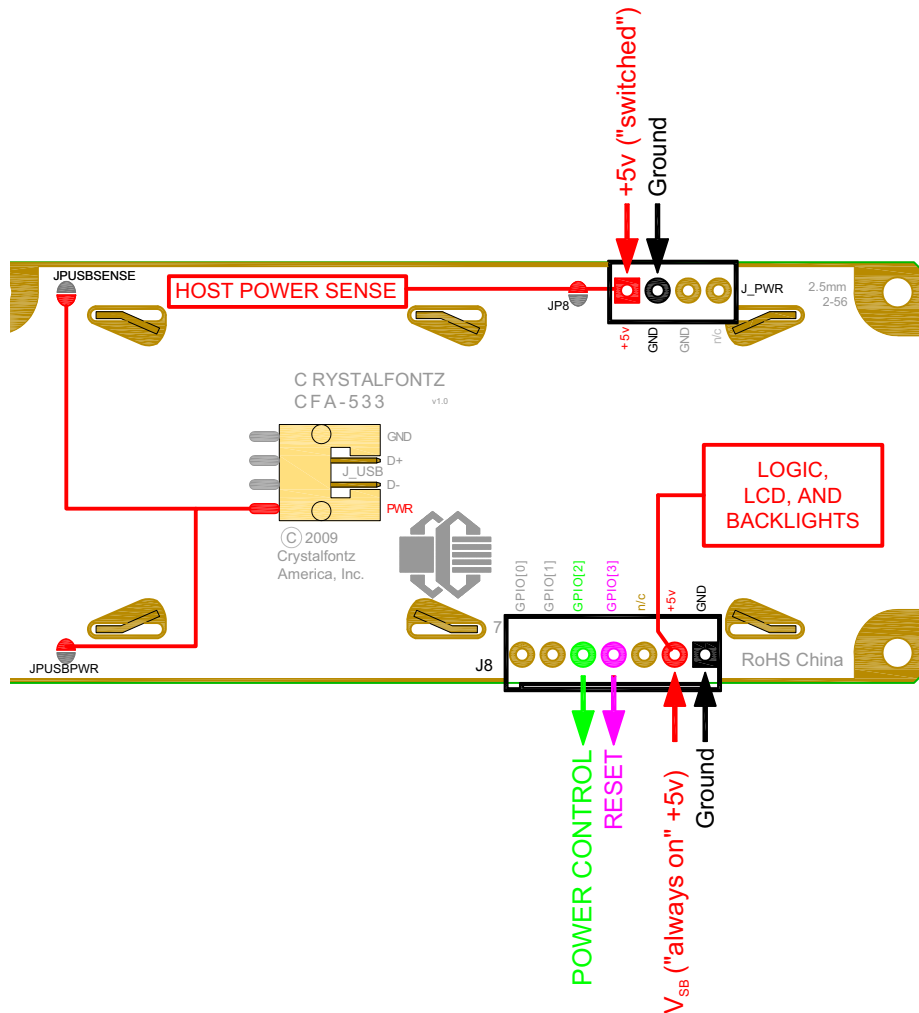


Figure 12. ATX Host Power Sense through +5v on J_PWR Connector

By default, the pin labeled +5v on the CFA533-YYH-KU's J_PWR connector is electrically connected to the +5v pin on the J8 connector through the normally open jumper JP8. If you want to use the CFA533-YYH-KU to do ATX power control, leave jumper JP8 open. The +5v pin of J_PWR will then function as the "Host Power Sense". The +5v pin of the J8 connector will function as V_{SB} (standby power which is always-on) to the LCD module. To complete the ATX power supply/host power sense configuration, open JPUSBPWR so that the LCD module is not powered from the host system's USB connection.



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

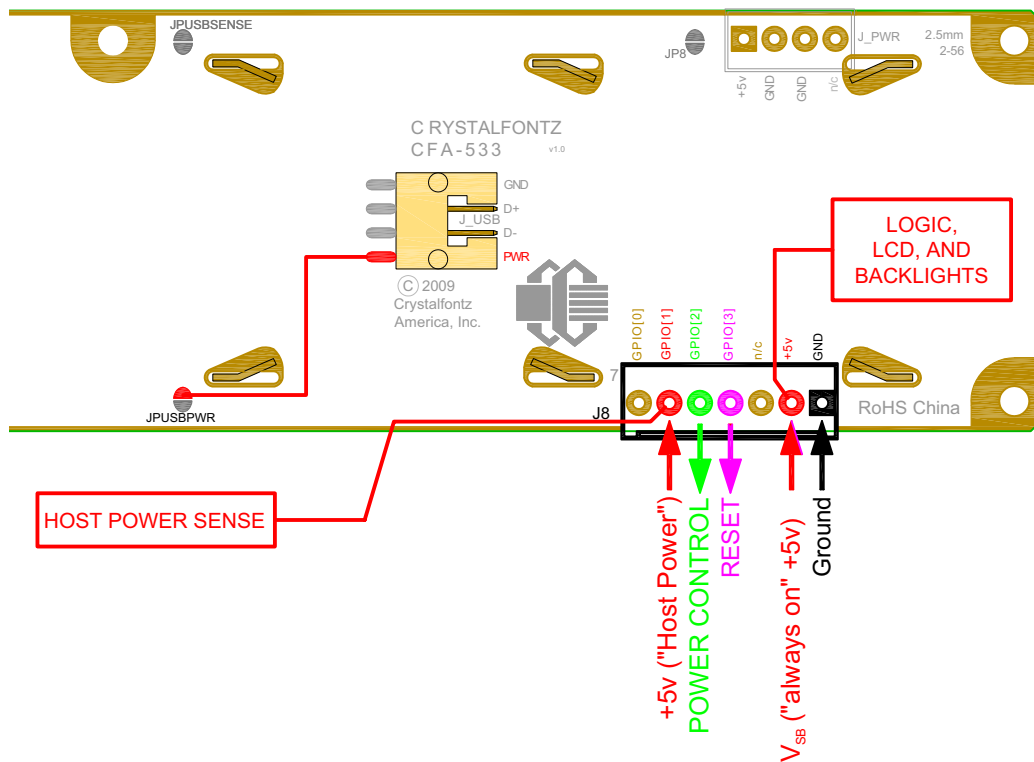


Figure 13. ATX Host Power Sense through GPIO[1] on J8 Connector

When configured, +5v senses on GPIO[1].

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. R21 is loaded with a 5kΩ 0805 SMT resistor for this functionality.

The motherboard's power switch input is connected to Pin 5 of the CFA533-YYH-KU'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 LCD 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 \(Pg. 47\)](#).) For location of R21, see [Figure 15. on Pg. 31](#).

JP8 from connector J_PWR is closed by default. When JP8 is closed, power is connected at J8. The Crystalfontz [WR-PWR-Y14](#) cable simplifies ATX power supply control connections. When using this cable, please open jumper JP8 in order to ensure correct operation.



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

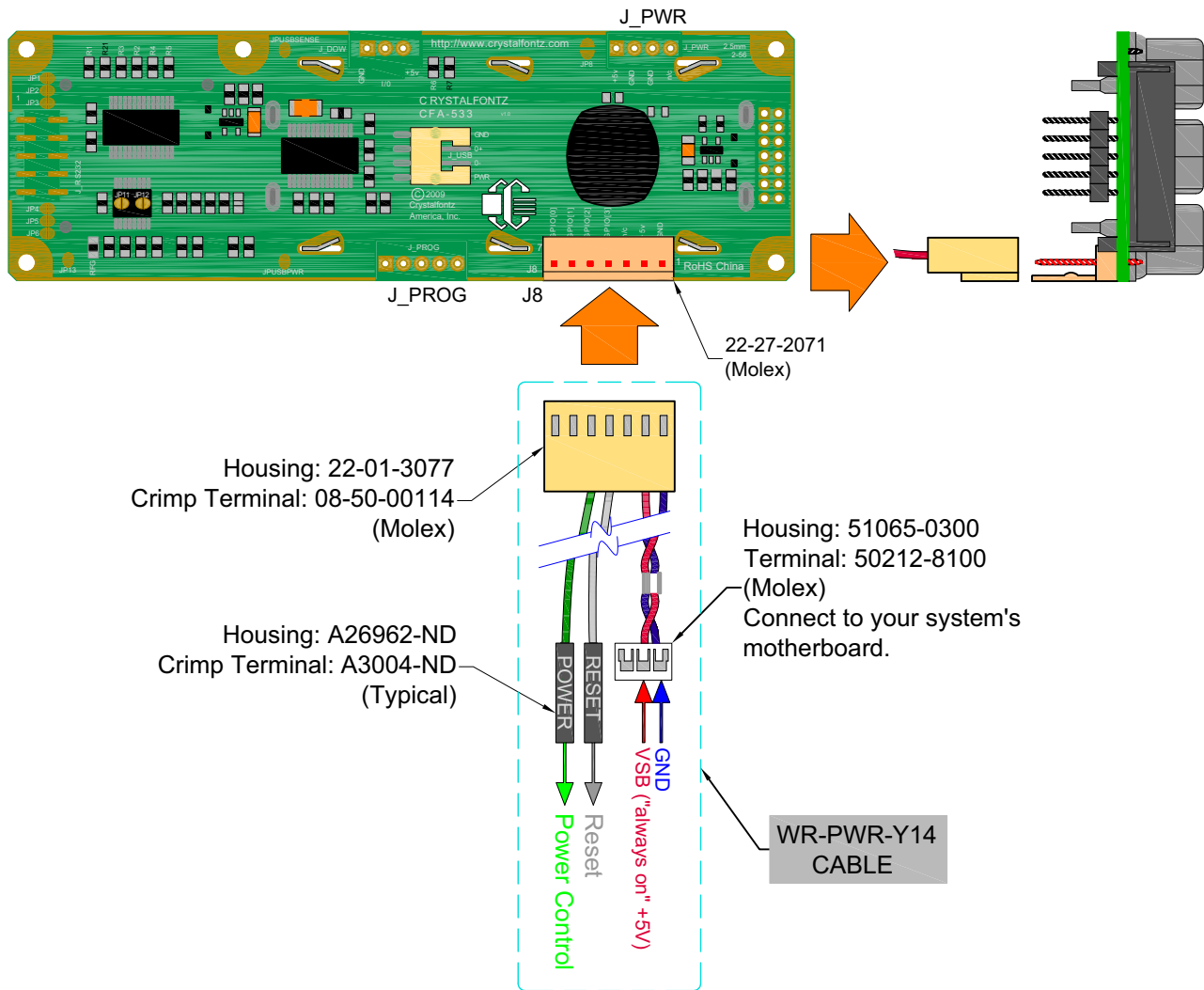


Figure 14. ATX Power Supply and Control Using Crystalfontz WR-PWR-Y14 Cable



NOTE

If the Crystalfontz [WR-PWR-Y14](#) cable is ordered at the same time as the LCD module, Crystalfontz will install the [WR-PWR-Y14](#) connector, open jumper JPUSBPWR, close JPUSBSENSE, and send the following software configuration commands (unless we are otherwise instructed). Please note that once these changes are made, if JP8 is not closed, power must be applied to +5v on the J_PWR connector and +5v on the J8 connector for the LCD module to power up. The module can not power up until the operating system allocates power to the LCD module, per the USB specification.

```
command = 28 // Set ATX Switch Functionality
length = 3
data[0] = 241 // Enable:
                // KEYPAD_POWER_OFF
                // KEYPAD_POWER_ON
                // KEYPAD_RESET
                // LCD_OFF_IF_HOST_IS_OFF
                // AUTO_POLARITY
data[1] = 16 // One half second power pulses
data[2] = 0 // sense ATX host power state on P0.7 (J_PWR, +5v)
command = 4 // Store current state as boot state
length = 0
```

ATX Keypad Control

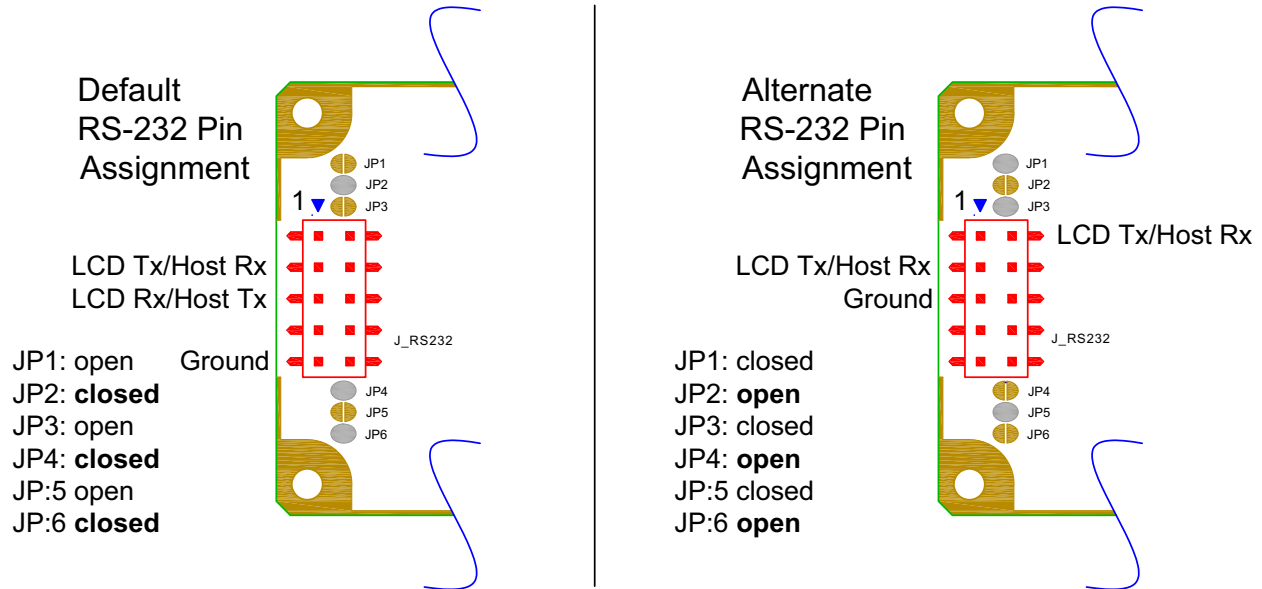
Once configured by the host software (see command [28 \(0x1C\): Set ATX Switch Functionality \(Pg. 47\)](#)), the following functions may be individually enabled:

- **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 \(Pg. 47\)](#) (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 \(Pg. 47\)](#) (1.0 seconds default). If the user continues to press the key, the CFA533-YYH-KU 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-YYH-KU will reboot itself immediately after resetting the host.

Since the computer and LCD module must look off if the computer's power is off, the CFA533-YYH-KU can be configured to monitor the POWER-ON SENSE line and blank its display any time the POWER-ON SENSE line is low.



GPIO CONNECTIONS



The CFA533-YYH-KU has five General-Purpose Input/Output (GPIO) pins. The GPIO are port pins from the CFA533-YYH-KU's microcontroller 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 LCD 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 R21 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--may have 1 kΩ hardware pull-up: R7)

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

GPIO[1] has a series 5kΩ resistor in R21.

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 R7 is loaded to pull GPIO[4] to V_{DD} (+5v power).



Please refer to commands [34 \(0x22\): Set/Configure GPIO \(Pg. 51\)](#) and [35 \(0x23\): Read GPIO Pin Levels and Configuration State \(Pg. 52\)](#) for additional details concerning the GPIO operation.

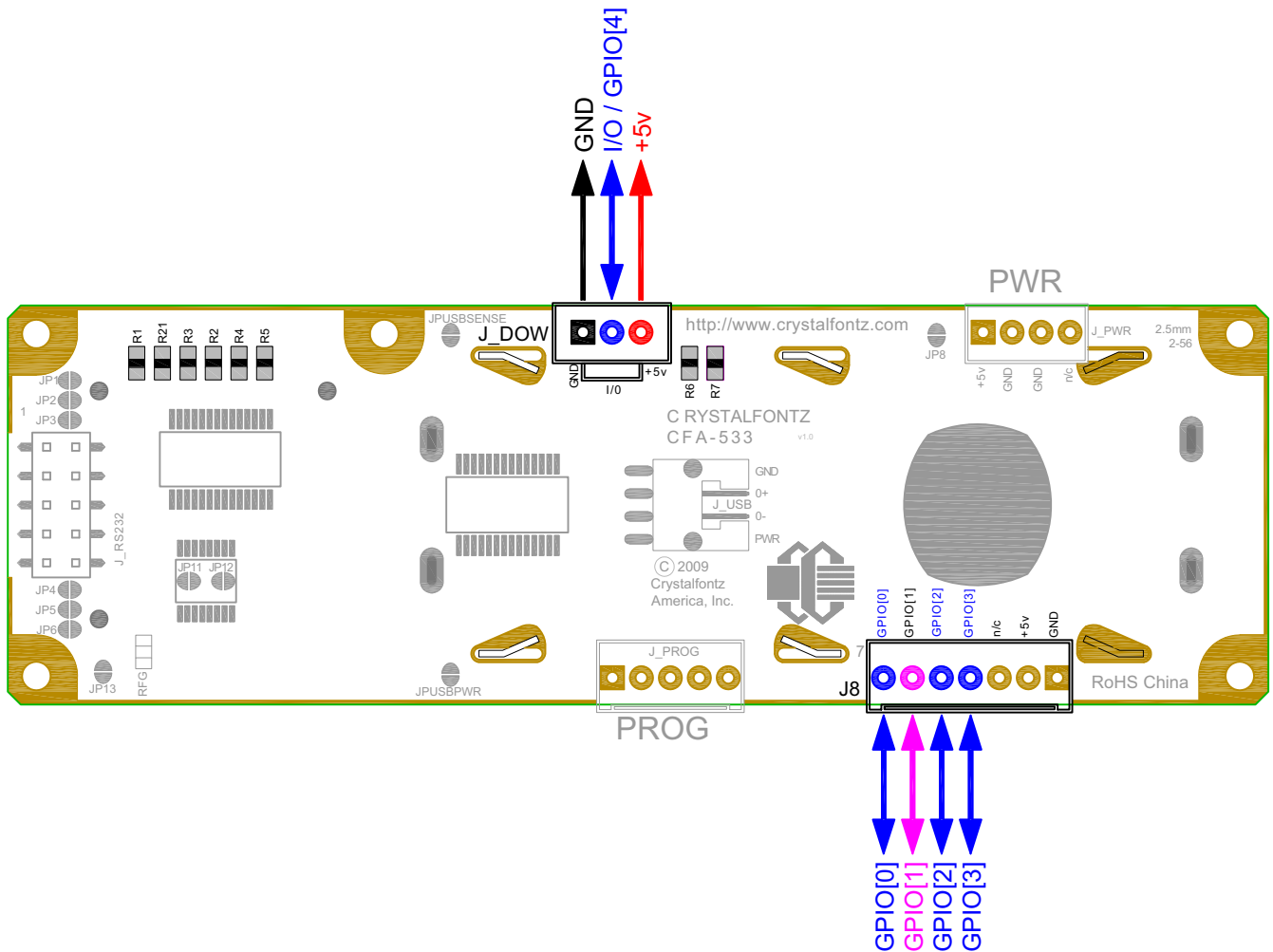


Figure 15. Location of GPIO Connections, Resistors, and J_DOW

DALLAS SEMICONDUCTOR 1-WIRE DEVICE CONNECTIONS FOR OPTIONAL ACCESSORIES

Temperature Sensors

The CFA533-YYH-KU supports Dallas Semiconductor 1-Wire (DOW) temperature sensors. See location of J_DOW in [Figure 15. on Pg. 31](#). By default, J_DOW is not loaded with a connector. If you request it, CrystalFontz can load a Molex 70543-0002. 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.

Any combination of up to 32 [DS1822](#) Econo 1-Wire Digital Thermometer (2°C absolute accuracy) or [DS18B20](#) High Precision 1-Wire Digital Thermometer (0.5°C absolute accuracy) temperature sensors or other DOW compatible devices are directly supported.



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 a CFA533-YYH-KU, Crystalfontz can load the WR-DOW-Y17's mating connector into the CFA533-YYH-KU's DOW position. For reference, the mating connector for the [WR-DOW-Y17](#) is [Molex 70543-0002](#) available from Digi-Key or other parts suppliers.

The temperature sensor can be configured to be automatically read and displayed on the CFA533-YYH-KU's LCD in °C or °F (see command [21 \(0x15\): Set Up Live Temperature Display \(Pg. 44\)](#)). 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 [Dallas Semiconductor 1-Wire devices](#) may be connected to the 1-Wire bus, with the CFA533-YYH-KU acting as a bridge between USB and the 1-Wire bus (see command [21 \(0x15\): Set Up Live Temperature Display \(Pg. 44\)](#)). The total number of 1-Wire devices supported is 32, including directly supported temperature sensors and any other user-provided 1-Wire devices. (See CFA533-YYH-KU's DOW connection location in [Figure 15. on Pg. 31.](#)) The LCD 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 module.

The CFA533-YYH-KU has a 1kΩ hardware pull-up on the DOW connector's I/O line.

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

HOST COMMUNICATIONS

NOTE

Because there is no difference in communications and commands for I²C 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 Data Sheet uses the shorter term “CFA533” instead of “CFA533-YYH-KU”.

The CFA533 series (includes CFA533-YYH-KU) 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 \(Pg. 50\)](#)).

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 LCD module. To reconcile your packets, please ensure that you have received the acknowledgement packet from the packet most recently sent before sending any additional packets to the LCD module. This practice will guarantee that you will not have any dropped packets or missed communication with the LCD 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 B: SAMPLE CODE AND CALCULATING THE CRC \(Pg. 64\)](#) 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 [633WinTest](#) along with its C source code. Both will work with the CFA533 modules. Included in the [633WinTest](#) 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 \(Pg. 45\)](#).

```
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
```

These codes are identical to the codes returned by the [CFA633](#), [CFA635](#), and [XES635](#). Please note that the [CFA631](#) will return codes 13 through 20. (See the [CFA631](#) Data Sheet on our website for more details).

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,yY.Y"
```

hX.X is the hardware revision, "1.0" for example
yY.Y is the firmware version, "u1.0" for example

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, which are affected by:
 - command [6 \(0x06\): Clear LCD Screen \(Pg. 38\)](#).
 - 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 \(Pg. 50\)](#).
- Special character font definitions (command [9 \(0x09\): Set LCD Special Character Data \(Pg. 39\)](#)).
- Cursor position (command [11 \(0x0B\): Set LCD Cursor Position \(Pg. 39\)](#)).
- Cursor style (command [12 \(0x0C\): Set LCD Cursor Style \(Pg. 40\)](#)).
- Contrast setting (command [13 \(0x0D\): Set LCD Contrast \(Pg. 40\)](#)).
- LCD backlight setting (command [14 \(0x0E\): Set LCD & Keypad Backlight \(Pg. 41\)](#)).
- Keypad backlight setting (command [14 \(0x0E\): Set LCD & Keypad Backlight \(Pg. 41\)](#)).
- Settings of any live displays (command [21 \(0x15\): Set Up Live Temperature Display \(Pg. 44\)](#)).



- Key press and release masks (command [23 \(0x17\): Configure Key Reporting \(Pg. 45\)](#)).
- ATX function enable and pulse length settings (command [28 \(0x1C\): Set ATX Switch Functionality \(Pg. 47\)](#)).
- Baud rate (command [33 \(0x21\): Set Baud Rate \(Pg. 50\)](#)).
- GPIO settings (command [34 \(0x22\): Set/Configure GPIO \(Pg. 51\)](#)).

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

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.

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 \(Pg. 51\)](#).

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 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 \(Pg. 47\)](#), 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 \(Pg. 47\)](#), 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 = 32₁₀ 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 center 16 characters displayed for the top line of LCD screen.

NOTE

Please use this command only if you need backwards compatibility with older [CFA633](#) units. For new applications, please use the more flexible command [31 \(0x1F\): Send Data to LCD \(Pg. 50\)](#) 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 center 16 characters displayed for the bottom line of LCD screen.

NOTE

Please use this command only if you need backwards compatibility with older [CFA633](#) units. For new applications, please use the more flexible command [31 \(0x1F\): Send Data to LCD \(Pg. 50\)](#) 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 (CGRAM).

```
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 CGRAM. 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 0x93 (\147) for DDRAM, line 1  
0xC0 (\192) to 0xD3 (\211) 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 \(Pg. 40\)](#).

```
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 Backlight

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 \(Pg. 36\)](#).

15-17 (0x0F-0x11): (reserved)

18 (0x12): Read DOW Device Information

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



The first byte returned is the “family code” of the Dallas 1-Wire / iButton device. There is a list of the possible Dallas 1-Wire / iButton device family codes available in [App Note 155: 1-Wire Software Resource Guide](#) on the Maxim/Dallas website.

NOTE ON COMMAND 18: READ DOW DEVICE INFORMATION

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 \(Pg. 51\)](#).

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 DOW devices.

```
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 ([DS1822](#) Econo 1-Wire Digital Thermometer temperature sensor) or 0x28 ([DS18B20](#) High Precision 1-Wire Digital Thermometer 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
  
```

(Continues on the next page.)

```

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 = disables
| | | | | | | | ----- 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 DOW Device Information \(Pg. 41\)](#).

The return packet will be:

```

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



20 (0x14): Arbitrary DOW Transaction

The CFA533 can function as an RS-232 to Dallas 1-Wire 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 and cannot be fully used with the CFA533.

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 C: CONNECTING A DS2450 1-WIRE QUAD A/D CONVERTER \(Pg. 76\)](#) 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 (0-7)
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 a Neotec [NT7070B](#) (HD44780 compatible). Generally you won't need low-level access to the LCD controller but some arcane functions of the [NT7070B](#) 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 = 2310
data_length: 2
data[0]: press mask
data[1]: release mask
```

The return packet will be:

```
type: 0x40 | 0x17 = 0x57 = 8710
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 \(Pg. 45\)](#). 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 = 2410
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
```

25-27 (0x19-0x1B): (reserved)



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 ON COMMAND 28: SET ATX SWITCH FUNCTIONALITY

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 \(Pg. 51\)](#). 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 R21 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
#define POWER_INVERT          0x04 //Power pin drives high instead of low
#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
          255 = 8 sec
data[2]: (optional) atx_sense_on_floppy (default setting)
          0: sense ATX host state on P2.1 (J8, pin 6 / GPIO [1] -- R21 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 high-availability 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 \(Pg. 47\)](#) or command [34 \(0x22\): Set/Configure GPIO \(Pg. 51\)](#).

```
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 \(Pg. 36\)](#).

32 (0x20): Reserved for [CFA631](#) Key Legends

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: 0
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 “bounce-free”.

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 5kΩ. 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 R21 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 R21 with a resistor suitable for your application. It is loaded with a 5kΩ resistor that is suitable for most applications.

NOTE ON SETTING AND CONFIGURING GPIO PINS

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 [633WinTest](#) 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] = J9, Pin 2 (default is DOW I/O--may have 1kΩ hardware pull-up)
  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 \(Pg. 51\)](#) 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] = J9, Pin 2 (default is DOW I/O--may have a 1kΩ hardware pull-up)
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	0_d 0000 ₂	16_d 0001 ₂	32_d 0010 ₂	48_d 0011 ₂	64_d 0100 ₂	80_d 0101 ₂	96_d 0110 ₂	112_d 0111 ₂	128_d 1000 ₂	144_d 1001 ₂	160_d 1010 ₂	176_d 1011 ₂	192_d 1100 ₂	208_d 1101 ₂	224_d 1110 ₂	240_d 1111 ₂
0_d 0000 ₂	CGRAM [0]															
1_d 0001 ₂	CGRAM [1]															
2_d 0010 ₂	CGRAM [2]															
3_d 0011 ₂	CGRAM [3]															
4_d 0100 ₂	CGRAM [4]															
5_d 0101 ₂	CGRAM [5]															
6_d 0110 ₂	CGRAM [6]															
7_d 0111 ₂	CGRAM [7]															
8_d 1000 ₂	CGRAM [0]															
9_d 1001 ₂	CGRAM [1]															
10_d 1010 ₂	CGRAM [2]															
11_d 1011 ₂	CGRAM [3]															
12_d 1100 ₂	CGRAM [4]															
13_d 1101 ₂	CGRAM [5]															
14_d 1110 ₂	CGRAM [6]															
15_d 1111 ₂	CGRAM [7]															

Figure 16. Character Generator ROM (CGROM)



LCD MODULE RELIABILITY AND LONGEVITY

MODULE RELIABILITY

ITEM	SPECIFICATION
LCD portion (excluding Keypad and Backlights)	50,000 to 100,000 hours (typical)
Keypad	1,000,000 keystrokes
LED Backlights	50,000 to 100,000 hours (typical)

MODULE LONGEVITY (EOL / REPLACEMENT POLICY)

CrystalFontz is committed to making all of our LCD modules available for as long as possible. For each module we introduce, we intend to offer it indefinitely. We do not preplan 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 Data Sheet 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.



CARE AND HANDLING PRECAUTIONS

For optimum operation of the CFA533-YYH-KU and to prolong its life, please follow the precautions described below.

ELECTROSTATIC DISCHARGE (ESD)

The circuitry is industry standard CMOS logic and susceptible to ESD damage. Please use industry standard antistatic 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.

DESIGN AND MOUNTING

- The exposed surface of the LCD “glass” is actually a polarizer laminated on top of the glass. To protect the soft plastic polarizer from damage, the module ships with a protective film over the polarizer. Please peel off the protective film slowly. Peeling off the protective film abruptly may generate static electricity.
- The polarizer is made out of soft plastic and is easily scratched or damaged. When handling the LCD module, avoid touching the polarizer. Finger oils are difficult to remove.
- CFA533-YYH-KU *without Crystalfontz overlay*: To protect the soft plastic polarizer from damage, place a transparent plate (for example, acrylic, polycarbonate, or glass) in front of the LCD module, leaving a small gap between the plate and the display surface. We use GE HP-92 Lexan, which is readily available and works well.
- Do not disassemble or modify the LCD module.
- Do not modify the six tabs of the metal bezel or make connections to them.
- Solder only to the I/O terminals. Use care when removing solder. It is possible to damage the PCB.
- Do not reverse polarity to the power supply connections. Reversing polarity will immediately ruin the LCD module.

AVOID SHOCK, IMPACT, TORQUE, OR TENSION

- Do not expose the LCD module to strong mechanical shock, impact, torque, or tension.
- Do not drop, toss, bend, or twist the LCD module.
- Do not place weight or pressure on the LCD module.

IF LCD PANEL BREAKS

- If the LCD panel breaks, be careful to not get the liquid crystal fluid in your mouth or eyes.
- If the liquid crystal fluid touches your skin, clothes, or work surface, wash it off immediately using soap and plenty of water.

CLEANING

- The polarizer (laminated to the glass) is soft plastic. The soft plastic is easily scratched or damaged. Be very careful when you clean the polarizer.
- Do not clean the polarizer with liquids. Do not wipe the polarizer with any type of cloth or swab (for example, Q-tips).
- Use the removable protective film to remove smudges (for example, fingerprints) and any foreign matter. If you no longer have the protective film, use standard transparent office tape (for example, Scotch® brand “Crystal Clear Tape”). If the polarizer is dusty, you may carefully blow it off with clean, dry, oil-free compressed air.



- CFA533-YYH-KU *without Crystalfontz overlay*: The exposed surface of the LCD “glass” is actually the front polarizer laminated to the glass. The polarizer is made out of a fairly soft plastic and is easily scratched or damaged. The polarizer will eventually become hazy if you do not take great care when cleaning it. Long contact with moisture (from condensation or cleaning) may permanently spot or stain the polarizer.

OPERATION

- Your circuit should be designed to protect the CFA533-YYH-KU from ESD and power supply transients.
- Observe the operating temperature limitations: a minimum of 0°C to a maximum of 50°C with minimal fluctuation. Operation outside of these limits may shorten life and/or harm display.
 - At lower temperatures of this range, response time is delayed.
 - At higher temperatures of this range, display becomes dark. (You may need to adjust the contrast.)
- Operate away from dust, moisture, and direct sunlight.

STORAGE AND RECYCLING

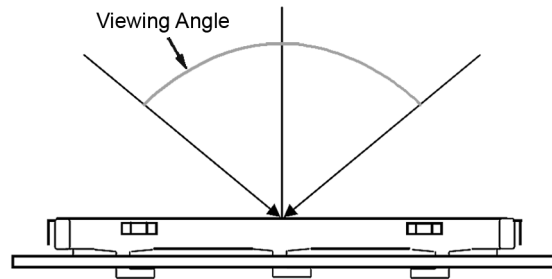
- Store in an ESD-approved container away from dust, moisture, and direct sunlight.
- Observe the storage temperature limitations: -10°C minimum, 60°C maximum with minimal fluctuation. Rapid temperature changes can cause moisture to form, resulting in permanent damage.
- Do not allow weight to be placed on the CFA533-YYH-KUs while they are in storage.
- Please recycle your outdated Crystalfontz modules at an approved facility.



APPENDIX A: QUALITY ASSURANCE STANDARDS

INSPECTION CONDITIONS

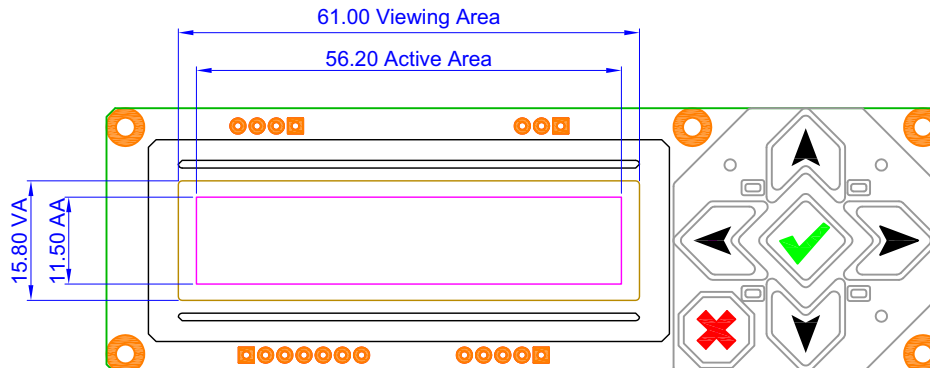
- Environment
 - Temperature: $25\pm 5^{\circ}\text{C}$
 - Humidity: 30~85% RH
- For visual inspection of active display area
 - Source lighting: two 20 Watt or one 40 Watt fluorescent light
 - Display adjusted for best contrast
 - Viewing distance: 30 ± 5 cm (about 12 inches)
 - Viewable angle: inspect at 45° angle of vertical line right and left, top and bottom



COLOR DEFINITIONS

We try to describe the appearance of our modules as accurately as possible. For the photos, we adjust for optimal appearance. Actual display appearance may vary due to (1) different operating conditions, (2) small variations of component tolerances, (3) inaccuracies of our camera, (4) color interpretation of the photos on your monitor, and/or (5) personal differences in the perception of color.

DEFINITION OF ACTIVE AREA AND VIEWABLE AREA





ACCEPTANCE SAMPLING

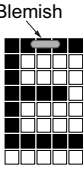
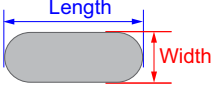
DEFECT TYPE	AQL*
Major	≤0.65%
Minor	≤1.00%
*Acceptable Quality Level: maximum allowable error rate or variation from standard	

DEFECTS CLASSIFICATION

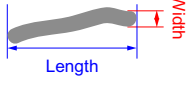
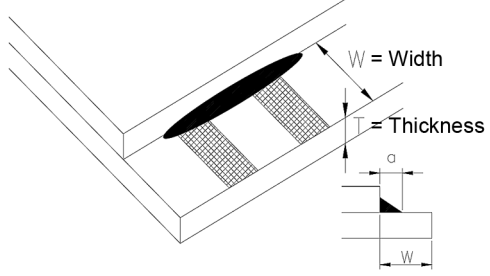
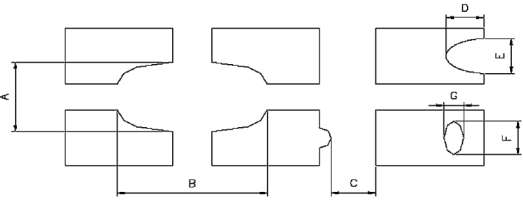
Defects are defined as:

- Major Defect: results in failure or substantially reduces usability of unit for its intended purpose
- Minor Defect: deviates from standards but is not likely to reduce usability for its intended purpose

ACCEPTANCE STANDARDS

#	DEFECT TYPE	CRITERIA		MAJOR / MINOR	
1	Electrical defects	1. No display, display malfunctions, or shorted segments. 2. Current consumption exceeds specifications.		Major	
2	Viewing area defect	Viewing area does not meet specifications. (See Inspection Conditions (Pg. 58)).		Major	
3	Contrast adjustment defect	Contrast adjustment fails or malfunctions.		Major	
4	Blemishes or foreign matter on display segments		<i>Defect Size (mm)</i>	<i>Acceptable Qty</i>	Minor
			≤0.3	3	
			≤2 defects within 10 mm of each other		
5	Other blemishes or foreign matter outside of display segments	Defect size = (A + B)/2 	<i>Defect Size (mm)</i>	<i>Acceptable Qty</i>	Minor
			≤0.15	Ignore	
			0.15 to 0.20	3	
			0.20 to 0.25	2	
			0.25 to 0.30	1	



#	DEFECT TYPE	CRITERIA			MAJOR / MINOR
6	Dark lines or scratches in display area 	<i>Defect Width (mm)</i>	<i>Defect Length (mm)</i>	<i>Acceptable Qty</i>	Minor
		≤0.03	≤3.0	3	
		0.03 to 0.05	≤2.0	2	
		0.05 to 0.08	≤2.0	1	
		0.08 to 0.10	≤3.0	0	
		≥0.10	>3.0	0	
7	Bubbles between polarizer film and glass	<i>Defect Size (mm)</i>	<i>Acceptable Qty</i>	Minor	
		≤0.20	Ignore		
		0.20 to 0.40	3		
		0.40 to 0.60	2		
		≥0.60	0		
8	Glass rest defect				Minor
9	Display pattern defect				Minor
		<i>Dot Size (mm)</i>	<i>Acceptable Qty</i>		
		$((A+B)/2) \leq 0.2$	≤3 total defects ≤2 pinholes per digit		
		C > 0			
		$((D+E)/2) \leq 0.25$			
		$((F+G)/2) \leq 0.25$			



#	DEFECT TYPE	CRITERIA	MAJOR / MINOR												
10	Chip in corner		Minor												
		<table border="1"> <thead> <tr> <th><i>a</i></th> <th><i>b</i></th> <th><i>c</i></th> <th>Acceptable Qty</th> </tr> </thead> <tbody> <tr> <td><4 mm</td> <td>$\leq W$</td> <td>$c \leq T$</td> <td>3</td> </tr> </tbody> </table>		<i>a</i>	<i>b</i>	<i>c</i>	Acceptable Qty	<4 mm	$\leq W$	$c \leq T$	3				
		<i>a</i>		<i>b</i>	<i>c</i>	Acceptable Qty									
<4 mm	$\leq W$	$c \leq T$	3												
11	Chip on "non-contact" edge of LCD		Minor												
		<table border="1"> <thead> <tr> <th><i>a</i></th> <th><i>b</i></th> <th><i>c</i></th> </tr> </thead> <tbody> <tr> <td>≤ 3 mm</td> <td>≤ 1 mm</td> <td>$\leq T$</td> </tr> <tr> <td>≤ 4 mm</td> <td>≤ 1.5 mm</td> <td>$\leq T$</td> </tr> </tbody> </table>		<i>a</i>	<i>b</i>	<i>c</i>	≤ 3 mm	≤ 1 mm	$\leq T$	≤ 4 mm	≤ 1.5 mm	$\leq T$			
		<i>a</i>		<i>b</i>	<i>c</i>										
≤ 3 mm	≤ 1 mm	$\leq T$													
≤ 4 mm	≤ 1.5 mm	$\leq T$													
12	Chip on "contact" edge of LCD, on the active side		Minor												
		<table border="1"> <thead> <tr> <th><i>a</i></th> <th><i>b</i></th> <th><i>c</i></th> <th>Acceptable Qty</th> </tr> </thead> <tbody> <tr> <td>≤ 2 mm</td> <td>$\leq W/4$</td> <td>$\leq T$</td> <td>Ignore</td> </tr> <tr> <td>≤ 3 mm</td> <td>$\leq W/4$</td> <td>$\leq T$</td> <td>3</td> </tr> </tbody> </table>		<i>a</i>	<i>b</i>	<i>c</i>	Acceptable Qty	≤ 2 mm	$\leq W/4$	$\leq T$	Ignore	≤ 3 mm	$\leq W/4$	$\leq T$	3
		<i>a</i>		<i>b</i>	<i>c</i>	Acceptable Qty									
≤ 2 mm	$\leq W/4$	$\leq T$	Ignore												
≤ 3 mm	$\leq W/4$	$\leq T$	3												



#	DEFECT TYPE	CRITERIA	MAJOR / MINOR												
13	Chip on "contact" edge of LCD, on the inactive side		Minor												
		<table border="1"> <thead> <tr> <th>a</th> <th>b</th> <th>c</th> <th>Acceptable Qty</th> </tr> </thead> <tbody> <tr> <td>≤3 mm</td> <td>≤1 mm</td> <td>≤T</td> <td>Ignore</td> </tr> <tr> <td>≤4 mm</td> <td>≤1.5 mm</td> <td>≤T</td> <td>3</td> </tr> </tbody> </table>		a	b	c	Acceptable Qty	≤3 mm	≤1 mm	≤T	Ignore	≤4 mm	≤1.5 mm	≤T	3
		a		b	c	Acceptable Qty									
		≤3 mm		≤1 mm	≤T	Ignore									
≤4 mm	≤1.5 mm	≤T	3												
<table border="1"> <thead> <tr> <th>a</th> <th>b</th> <th>c</th> <th>Acceptable Qty</th> </tr> </thead> <tbody> <tr> <td><3 mm</td> <td>≤1.5 mm</td> <td>≤1/2 T</td> <td>3</td> </tr> </tbody> </table>	a	b	c	Acceptable Qty	<3 mm	≤1.5 mm	≤1/2 T	3							
a	b	c	Acceptable Qty												
<3 mm	≤1.5 mm	≤1/2 T	3												
Unacceptable if c>50% of glass thickness or if the seal area is damaged.	Major														
14	Chip in seal area		Minor												
		<table border="1"> <thead> <tr> <th>a</th> <th>b</th> <th>c</th> <th>Acceptable Qty</th> </tr> </thead> <tbody> <tr> <td><3 mm</td> <td>≤1.5 mm</td> <td>≤1/2 T</td> <td>3</td> </tr> </tbody> </table>		a	b	c	Acceptable Qty	<3 mm	≤1.5 mm	≤1/2 T	3				
		a		b	c	Acceptable Qty									
<3 mm	≤1.5 mm	≤1/2 T	3												
Unacceptable if c>50% of glass thickness or if the seal area is damaged.	Major														
15	Backlight defects	<ol style="list-style-type: none"> 1. Light fails or flickers.* 2. Color and luminance do not correspond to specifications.* 3. Exceeds standards for display's blemishes or foreign matter (see test 5, Pg. 59), and dark lines or scratches (see test 6, Pg. 60). <p><i>*Minor if display functions correctly. Major if the display fails.</i></p>	Minor												
16	COB defects	<ol style="list-style-type: none"> 1. Pinholes >0.2 mm. 2. Seal surface has pinholes through to the IC. 3. More than 3 locations of sealant beyond 2 mm of the sealed areas. 	Minor												
17	PCB defects	<ol style="list-style-type: none"> 1. Oxidation or contamination on connectors.* 2. Wrong parts, missing parts, or parts not in specification.* 3. Jumpers set incorrectly. 4. Solder (if any) on bezel, LED pad, zebra pad, or screw hole pad is not smooth. <p><i>*Minor if display functions correctly. Major if the display fails.</i></p>	Minor												



#	DEFECT TYPE	CRITERIA	MAJOR / MINOR
18	Soldering defects	1. Unmelted solder paste. 2. Cold solder joints, missing solder connections, or oxidation.* 3. Solder bridges causing short circuits.* 4. Residue or solder balls. 5. Solder flux is black or brown. <i>*Minor if display functions correctly. Major if the display fails.</i>	Minor



APPENDIX B: SAMPLE CODE AND CALCULATING THE CRC

SAMPLE CODE

We encourage you to use the free sample code listed below. Please leave the original copyrights in the code.

- ❑ Windows compatible test/demonstration program and source. Supports CFA533 and CFA633.
<http://www.crystalfontz.com/product/633WinTest.html>
- ❑ Linux compatible command-line demonstration program with C source code. Supports CFA533, CFA631, CFA632, CFA633, CFA634, and CFA635.
http://www.crystalfontz.com/product/linux_cli_examples.html
- ❑ Supported by CrystalControl freeware.
<http://www.crystalfontz.com/product/CrystalControl2.html>

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_Errata.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, 0x0FBF7, 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
;
main      org      20         ; start of main program
          movlw   seedhi      ; setup initial 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     clrcc              ; 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_STRUCTURE)
  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,0x0AC42,0x09ED9,0x08F50,0x0FBF7,0x0EA66,0x0D8FD,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 = 0xFFFF;
  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

```




```
    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

;=====
```

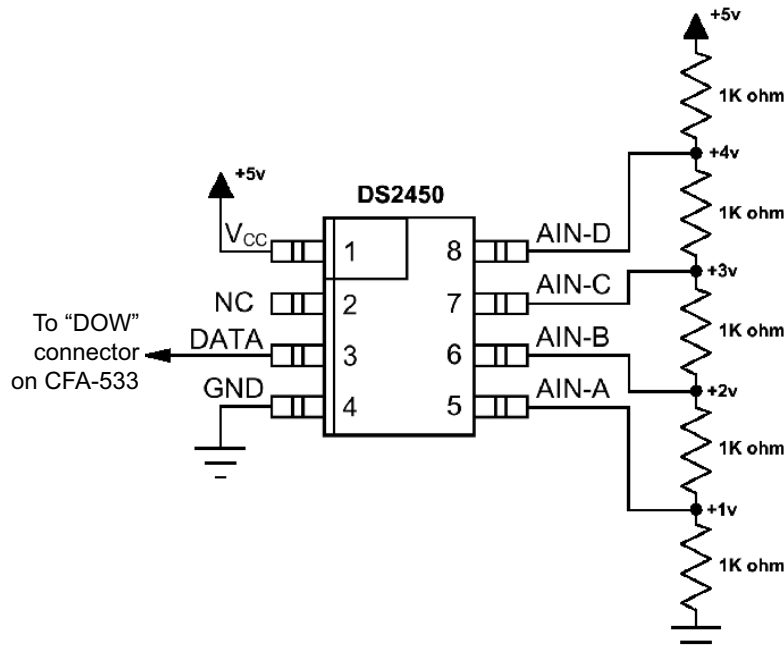


APPENDIX C: CONNECTING A DS2450 1-WIRE QUAD A/D CONVERTER

This appendix describes a simple test circuit that demonstrates how to connect a Dallas Semiconductor DS2450 4-channel ADC to the CFA-533's DOW (Dallas One Wire) 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 DOW Device Information \(Pg. 41\)](#) 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 DOW Transaction \(Pg. 44\)](#) more information.



Appendix C 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 = 130561.016015625 volts (channel A)
0x64DF = 258232.009541321 volts (channel B)
0x9684 = 385322.998553467 volts (channel C)
0xC86A = 513063.992623901 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 = 131631.024342346 volts (channel A)
0x64B2 = 257782.006039429 volts (channel B)
0x9697 = 385513.000032043 volts (channel C)
0xC842 = 512663.989511108 volts (channel D)
```