Crystalfontz

INTELLIGENT LCD MODULE SPECIFICATIONS

Datasheet Release 2021-09-20
for
CFA533-xxx-KS

Hardware Version: v1.4
Firmware Version: s1v2

Crystalfontz America, Inc.
12412 East Saltese Avenue
Spokane Valley, WA 99216-0357
Phone: 888-206-9720
Fax: 509-892-1203
Email: support@crystalfontz.com
URL: www.crystalfontz.com

Table of Contents

1. General Information

Datasheet Revision History

Datasheet Version: **2021-09-20**
Hardware Version: **v1.4**
Firmware Version: **s1v2**

This datasheet has been updated to reflect hardware version v1.4, firmware s1v2 for the CFA533-xxx-KS LCD module.

For information about firmware and hardware revisions, see the Part Change Notifications (PCNs) under "News" in our website's navigation bar.

Previous datasheet Version: **2016-12-16**

For reference, previous datasheets may be downloaded by clicking the "Show Previous Versions of Datasheet" link under the "Datasheets and Files" tab of the product web page.

Product Change Notifications

To check for or subscribe to "Part Change Notices" for this display module, see the Product Notices webpage.

Variations

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

Volatility

This display module has volatile and non-volatile memory.
Non-volatile memory capability (EEPROM): Set the "power on" display screen, plus 16-bytes

Disclaimer

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.

All specifications in datasheets on our website are, to the best of our knowledge, accurate but not guaranteed. Corrections to specifications are made as any inaccuracies are discovered.

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

2. Introduction

The CFA533-xxx-KS is an intelligent LCD module with a 6-button keypad, configured for communication using RS-232.

2.1. Main Features

- 16 characters x 2 lines LCD with keypad and high-level interface.
- Fits in a 1U rack mount case (35 mm overall height). A drive bay bracket is available to add on during customization for a sleek fit in a 1U rack.
- A single 3.3-5v supply is needed for micro-controller, backlight, and LCD.
- "Live Display" shows up to eight temperature readings without host intervention, allowing temperatures to be shown immediately at boot, before the host operating system is loaded.
- Adjustable, long-life backlight driven from the 5v supply at constant current. The brightness is independent of power supply variations.
- Bi-directional RS232 interface
- Robust packet-based communications protocol with 16-bit CRC.
- 6 o'clock viewing direction.
- Integrated, LED-backlit, 6-button keypad with four directional arrows, Enter, and Cancel.
- **TFH** modules are edge-lit by a white LED backlight with positive FSTN light gray transflective LCD. Displays dark characters on a light gray background. Sunlight readable.
- **TMI** modules are edge-lit by a blue LED backlight with negative STN blue transmissive mode LCD. Displays light characters on a deep blue background.
- **YYH** modules are edge-lit by a yellow-green LED backlight with positive STN yellow-green transflective mode LCD. Displays dark characters on yellow-green background.
- Viewable in normal office lighting and in dark areas. Sunlight readability depends on module color.
- Non-volatile memory capability (EEPROM): Set the "power on" display screen, plus 16-bytes for storing IP, netmask, system serial number, or other data.

2.2. Module Classification Information

CFA	533	x	x	x	KS
1	2	3	4	5	6

1	**Brand**	Crystalfontz America, Inc.
2	**Model Identifier**	533
3	**Backlight Type & Color**	T – LED, white Y – LED, yellow-green
4	**Fluid Type, Image (positive or negative), & LCD Glass Color**	F – FSTN positive, light gray M – STN negative, blue Y – STN positive, yellow-green
5	**Polarizer Film Type, Temperature Range, & View Angle (O'clock)**	H – Transflective, Wide Temperature Range[1], 6:00 I – Transmissive, Wide Temperature Rage[1], 6:00
6	**Interface**	KU – USB KS – Full swing RS-232 KC – I2C KL – Logic-level serial

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

2.3. Build Configuration Options

CFA533-xxx-KS8 – populate J_DOW with header
CFA533-xxx-KS16 – populate J8 with 7-pin header
CFA533-xxx-KS24 – populate J8 with 7-pin header, populate J_DOW with header
CFA533-xxx-KS48 – populate J8 with 7-pin header, configure for ATX
CFA533-xxx-KS56 - populate J8 with 7-pin header, configure for ATX, populate J_DOW with header
DBBK – mount into drive bay bracket

- Populate J_DOW
  - Install a 3-pin Dallas 1-Wire (DOW) header to daisy chain up to 32 WR-DOW-Y17 DOW temperature sensor cables.
    - "Live Display" shows up to four temperature readings without host intervention. Temperatures can show at boot, before the host operating system is loaded.
    - RS-232 to DOW functionality allows control of other DOW compatible devices (Additional hardware required.)
- Populate J8
  - Install a 7-pin header to use J8 for power connections
- ATX Power Supply Control
  - Only available with J8 populated.
- ATX power supply control functionality allows the buttons on the CFA533-xxx-KS to replace the Power and Reset switches on the host system. The ATX functionality can also implement a hardware watchdog that resets host system on host software failure.

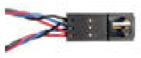
2.4. CFA533 Family

PART NUMBER	FLUID	LCD GLASS COLOR	IMAGE	POLARIZER FILM	BACKLIGHT COLOR/TYPE
CFA533-TFH-KC (I2C)	FSTN	light gray	positive	transflective	LCD: white edge LEDs Keypad: white LEDs
CFA533-TFH-KL ("logic-level" RS-232)					
CFA533-TFH-KS ("full swing" RS-232)					
CFA533-TFH- KU (USB)					
CFA533-TMI-KC (I2C)	STN	blue	negative	transmissive	LCD: white edge LEDs Keypad: blue LEDs
CFA533-TMI-KL ("logic-level" RS-232)					
CFA533-TMI-KS ("full swing" RS-232)					
CFA533-TMI-KU (USB)					
CFA533-YYH-KC (I2C)	STN	yellow-green	positive	transflective	LCD: yellow-green edge LEDs Keypad: yellow-green LEDs
CFA533-YYH-KL ("logic-level" RS-232)					
CFA533-YYH-KS ("full swing" RS-232)					
CFA533-YYH-KU (USB)					

2.5. Comparison to CFA633

The CFA533 series is mechanically similar to the CFA633 series. The CFA533 series command set is compatible with the CFA633 series. The CFA533 can be used as an economical "drop-in" replacement for most CFA633 series applications that do not need fan capabilities.

2.6. Accessories

Part Number	Image	Description All Cables Are RoHS Compliant
WR-PWR-Y12 ~13 inches		4-pin power splitter cable. Use this cable to plug a 4-pin "hard drive style" Molex power connector into the module's "floppy drive style" power connector, plus provides an additional 4-pin Molex connector.
WR-232-Y08 ~27 inches		Supply communications with this ribbon cable. Connect cable's 10-pin socket connector to the module's J_RS232 pin connector. Connect cable's RS232 DB9 9-pin socket connector to host's DB9 9-pin serial port. Default or alternate motherboard RS-232 pinouts can be accommodated by changing jumpers on the CFA533.
WR-232-Y22 ~26 inches		Supply communications with this cable. Connect one of the 10-pin socket connectors to the module's' J_RS232 10-pin connector. Connect cable's second 10-pin socket to host's motherboard 10-pin connector. This cable supports standard or alternate pinout motherboard RS-232 connections without changing jumpers on the module.
WR-PWR-Y14 ~24 inches WR-PWR-Y44		Use this ATX power cable to turn an ATX power supply on and off, or power cycle the host through the module. Connects to host's WOL connector. Connect the cable's 7-pin connector to the module's J8 connector.
WR-PWR-05		Use this ATX power cable to turn an ATX power supply on and off, or power cycle the host through the module. Connects to host's +5v Stand By. Connect the cable's 7-pin connector to the module's J8 connector.
WR-DOW-Y17 ~12 inches + ~12 inches between connectors		Connect ("daisy chain") up to 32 of these DOW (Dallas 1-Wire) DS18B20 temperature sensor cables. Requires optional DOW connector at J_DOW on module.
DB533-BK		Drive Bay Bracket for LCD module. For the module installed in the bracket, select the option after clicking "Customize and Add to Cart".

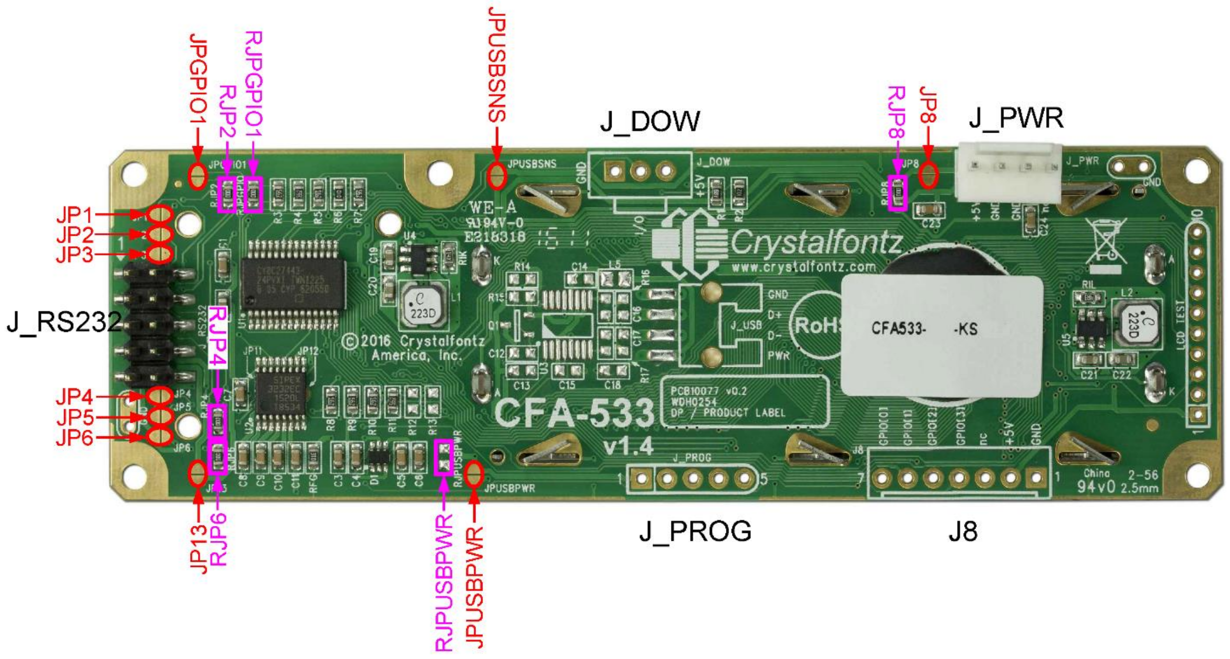
3. Mechanical Characteristics

3.1. Physical Characteristics

Item	Specification (mm)	Specification (inch, reference)
Module Overall Dimensions		
Width and Height	110.5 (W) x 35.0 (H)	4.35 (W) x 1.378 (H)
Depth without Keypad	16.60	0.65
Viewing Area	61.0 (W) x 15.8 (H)	2.402 (W) x 0.622 (H)
Active Area	56.20 (W) x 11.5 (H)	2.213 (W) x 0.453 (H)
Character Size	2.95 (W) x 5.55 (H)	0.116 (W) x 0.219 (H)
Dot Size	0.55 (W) x 0.65 (H)	0.022 (W) x 0.026 (H)
Keystroke Travel (approximate)	2.4	0.094
Weight	41 grams (typical)	1.45 ounces

3.2. Jumper Locations and Functions (All Interfaces)

All jumpers are configurable, but not all jumpers will affect a given interface. Close the jumpers by melting a ball of solder across their gap. Reopen the jumpers by removing the solder with a solder wick. Where applicable, the corresponding resistor must also be removed to open a jumper (RJP2 must be removed for JP2 to be open, for example).



CFA533 HW v1.4 Jumper Locations and Functions

JUMPER	FUNCTION	-KS
JP1	Alternate RS232 Configuration	Open
JP2	Standard RS232 Configuration	Closed (0Ω RJP2)
JP3	Alternate RS232 Configuration	Open
JP4	Standard RS232 Configuration	Closed (0Ω RJP4)
JP5	Alternate RS232 Configuration	Open
JP6	Standard RS232 Configuration	Closed (0Ω RJP6)
JP8	Connects the display's +5v to +5v on J_PWR. Conflicts with JPUSBSNS	Closed (0Ω RJP8)
JP13	Connects the display's +5v to Pin 4 on JRS232	Open
JPUSBPWR	Connects the display's +5v to PWR on JUSB	Open
JPUSBSNS	Connects the display's ATX SENSE to PWR on J-USB. Conflicts with JP8	Open
JPGPIO1	Bypasses R3 when closed. R3 is a 5.6KΩ resistor in series with GPIO1	Closed (0Ω RJPGPIO)

4. Optical Characteristics

The CFA533 has a **6 o'clock** viewing direction.

Module	Symbol	Typical	Conditions	
TFH	Φ_{Right}	50	Θ=0	CR≥2 Ta=25°
	Φ_{Left}	30	Θ=180	
	Φ_{Up}	30	Θ=90	
	Φ_{Down}	30	Θ=270	
TMI and YYH	Φ_{Right}	45	Θ=0	CR≥2 Ta=25°
	Φ_{Left}	25	Θ=180	
	Φ_{Up}	30	Θ=90	
	Φ_{Down}	30	Θ=270	

Item	Symbol	Condition	Minimum	Typical	Maximum
Contrast Ratio[1]	CR		-	10	15
LCD Response Time[2]	T_{rise}	Ta=25°C	-	80 ms	160 ms
	T_{fall}	Ta=25°C	-	100 ms	200 ms

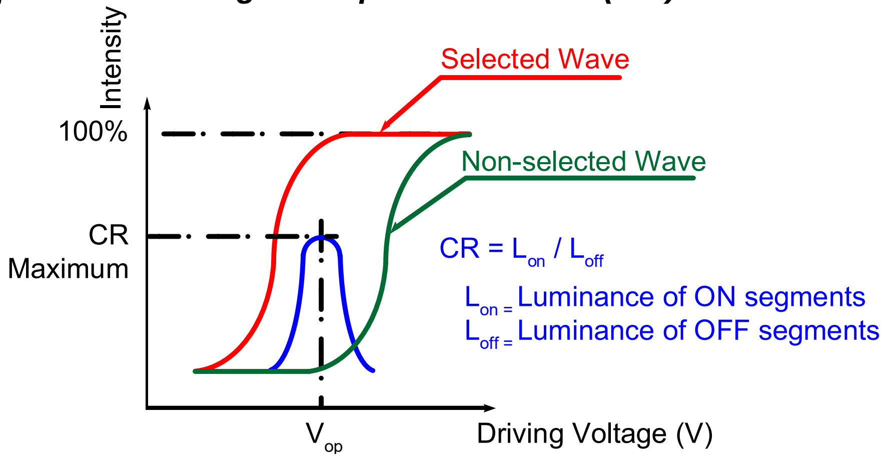
[1]Contrast Ratio = (brightness with pixels light)/ (brightness with pixels dark)
[2]Response Time = The amount of time it takes a liquid crystal cell to go from active to inactive or back again.

4.1. Test Conditions and Definitions for Optical Characteristics

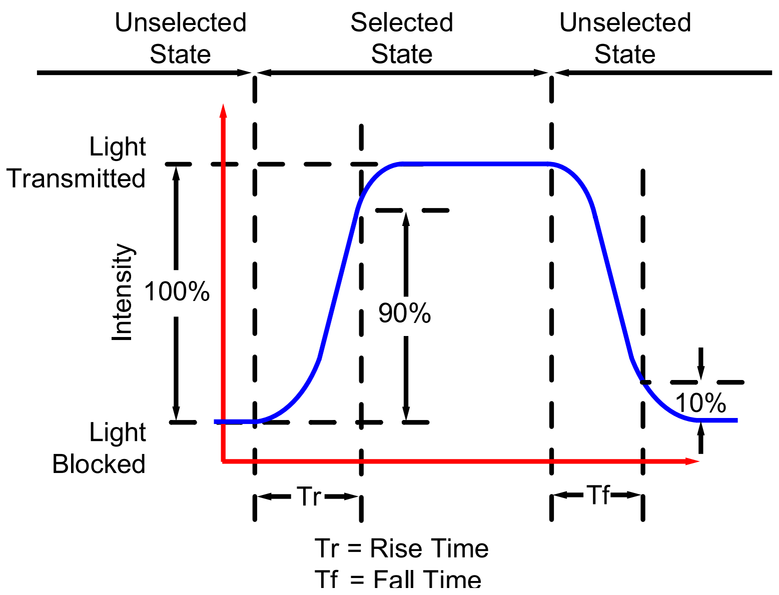
- Viewing Angle
  - Vertical (V)θ: 0°
  - Horizontal (H)φ: 0°
- Frame Frequency: 64 Hz
- Driving Waveform: 1/16 Duty, 1/5 Bias
- Ambient Temperature (Ta): 25°C

4.2. Optical Definitions for Negative Image Modules - CFA533-TMI-KS

4.2.1. Operational Voltage for Optimal Contrast (V_{OP})

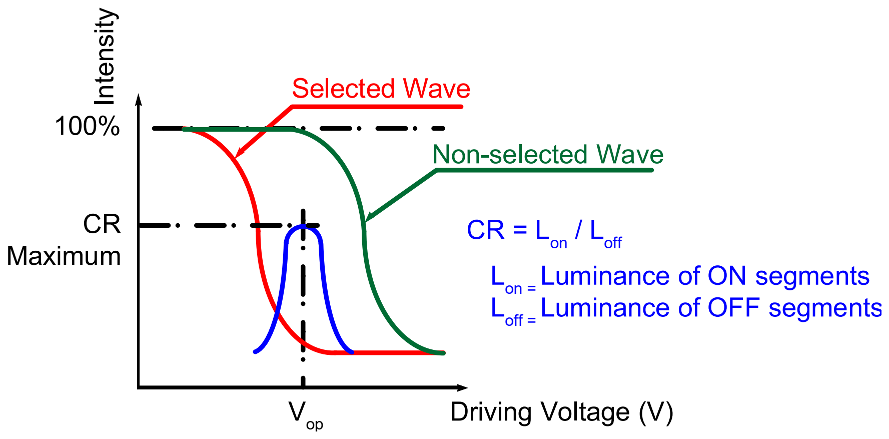

Intensity
100%
CR
Maximum
Selected Wave
Non-selected Wave
CR = L_{on} / L_{off}
L on = Luminance of ON segments
L off = Luminance of OFF segments
V op
Driving Voltage (V)

4.2.2. Response Time

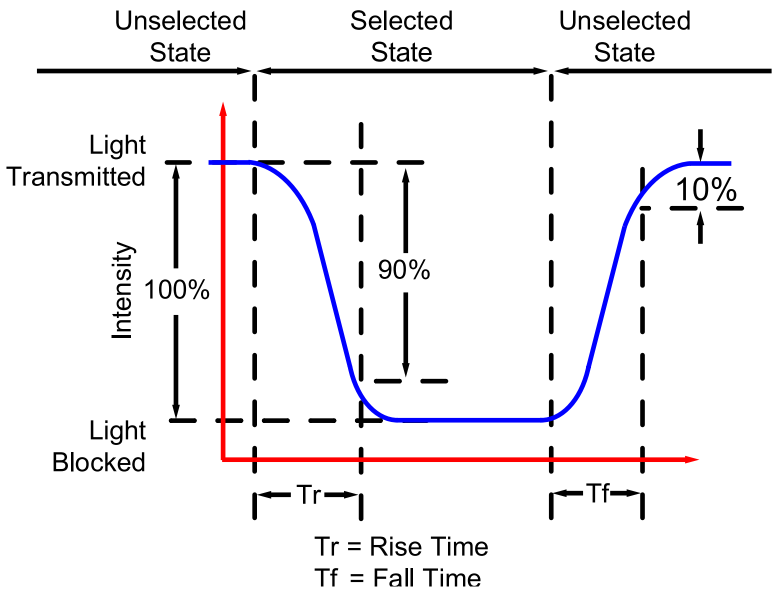


4.3. Optical Definitions for Positive Image Modules - CFA533-TFH-KS and CFA533-YYH-KS

4.3.1. Operational Voltage for Optimal Contrast (V_{OP})

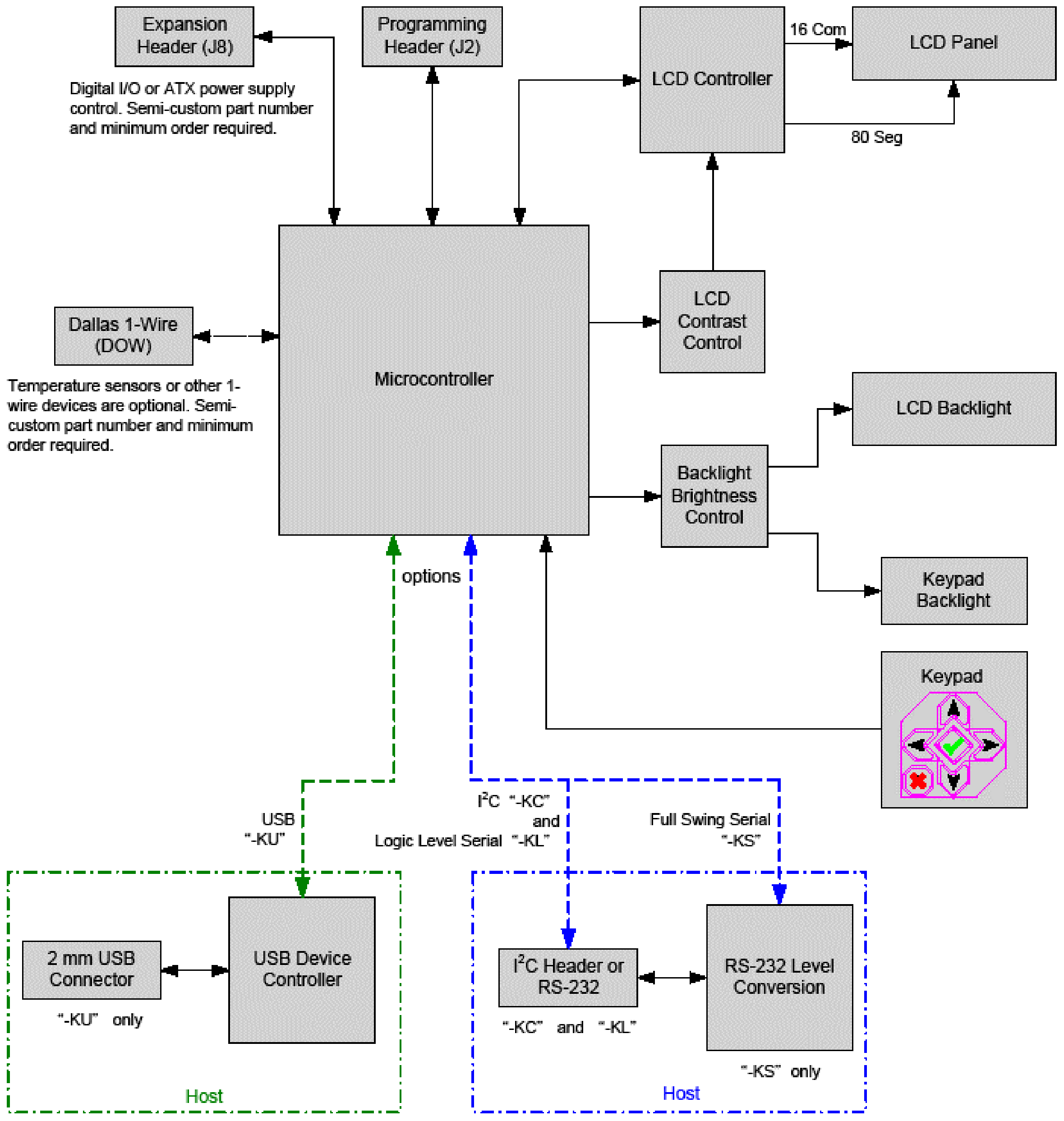

Intensity
100%
CR
Maximum
Selected Wave
Non-selected Wave
CR = L_{on} / L_{off}
L_{on} = Luminance of ON segments
L_{off} = Luminance of OFF segments
V_{op}
Driving Voltage (V)

4.3.2. Response Time



5. Electrical Specifications

5.1. System Block Diagram



System Block Diagram

6. Supply Voltages and Current

6.1. Absolute Maximum Ratings

Absolute Maximum Ratings	Symbol	Minimum	Maximum
Operating Temperature	T_{OP}	-20°C	+70°C
Storage Temperature	T_{ST}	-30°C	+80°C
Humidity Range (Non-condensing)	RH	10%	90%
Please note that these are stress ratings only. Extended exposure to the absolute maximum ratings listed above may affect device reliability or cause permanent damage. Functional operation of the module at these conditions beyond those listed under DC Characteristics is not implied.			

5v Typical Current Consumption	Specification
+5v (LCD, microcontroller, with backlight **off, 0%**)	< 20mA
+5v (LCD, microcontroller, with **white** backlight **on, 100%**)	< 100 mA
+5v (LCD, microcontroller, with yellow backlight **on, 100%**)	< 120 mA

6.2. DC Characteristics

The CFA533 has 5 GPIO (General-Purpose Input/Output) pins available. These pins connect to the processor's CMOS GPIO pins. They may be set to input or output. Some pins have special purpose functions. When they are set as GPIO outputs, the average voltage can be controlled by PWM. Refer to 34 (0x22): Set or Set and Configure GPIO Pins and 35 (0x23): Read GPIO Pin Levels and Configuration State for more information.

	DC Characteristics	Test Conditions	Symbol	Minimum	Typical	Maximum
Controller and Board	Supply Voltage for Logic	T_{OP} =-30°C to +70°C	V_{DD} - GND	3.2v	3.3v or 5.0v	5.25v
	Input High Voltage	V_{DD} = +5v	V_{IH}	2.2v	-	V_{DD}
	Input Low Voltage		V_{IL}	-0.3v	-	+0.6v
	Output High Voltage		V_{OH}	2.4v	-	-
	Output Low Voltage		V_{OL}	-	-	+0.4v
	Supply Current (including backlight)	V_{DD}=5.0v	I_{DD}	-	105mA	-

GPIO[0] through GPIO[4] Current Limits	Specification
Sink	25 mA
Source	10 mA

6.3. RS-232 Characteristics

Signals on Header J_RS232				
Specification	**Symbol**	**Minimum**	**Typical**	**Maximum**
RS232 Input Voltage Range		-15v		+15v
RS232 Input High Voltage	$V_{IH-RS232}$	+2.4v		
RS232 Input Low Voltage	$V_{IL-RS232}$			+0.8v
RS232 Output Voltage Swing		±5.0v	±5.4v	

ESD (Electro-Static Discharge) Specifications for Tx and Rx pins of connector J_RS232 only:

+15 kV Human Body Model

+15 kV IEC1000-4-2 Air Discharge

+8 kV IEC1000-4-2 Contact Discharge

The remainder of this circuitry is industry standard CMOS logic and susceptible to ESD damage. Use industry standard antistatic precautions such as those used for other static sensitive devices, e.g., expansion cards, motherboards, integrated circuits. Ground personnel, work surfaces, and equipment.

6.4. LED Backlight Information

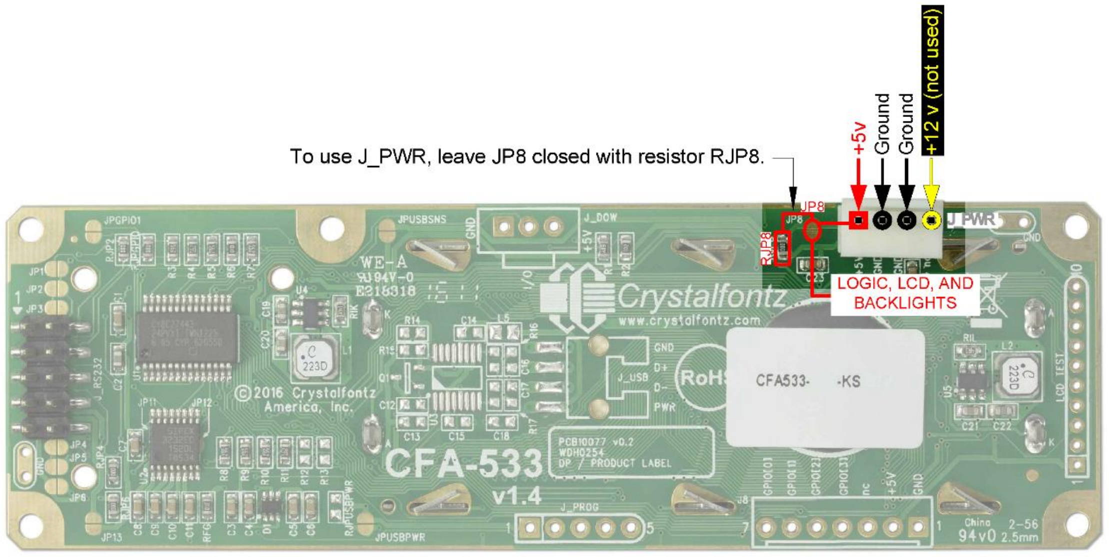
The backlights used in the CFA533 are designed for very long life, but their lifetime is finite. To conserve the LED lifetime, dim or turn off the backlights during periods of inactivity.

7. Connection Information

This section describes methods to connect power and host power sense to the display module. The host power supply can power the CFA533-xxx-KS in one of two ways:

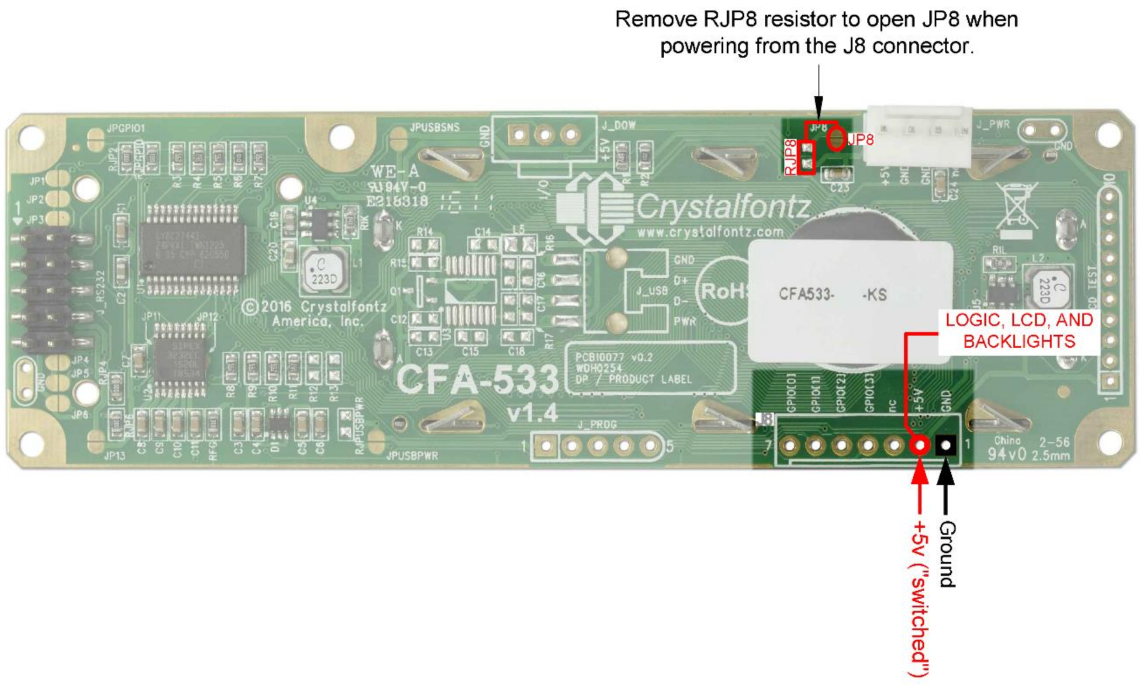
1. *Standard:* This is the basic method to supply power to the display module ("non-ATX").
2. *ATX:* This method supplies power to the display module and has power on, power off, and reset functionality to the host. When using ATX functionality, the module is powered from the host's standby power (VSB signal) which is always on.

7.1. Connection through J_PWR Connector (Standard)


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

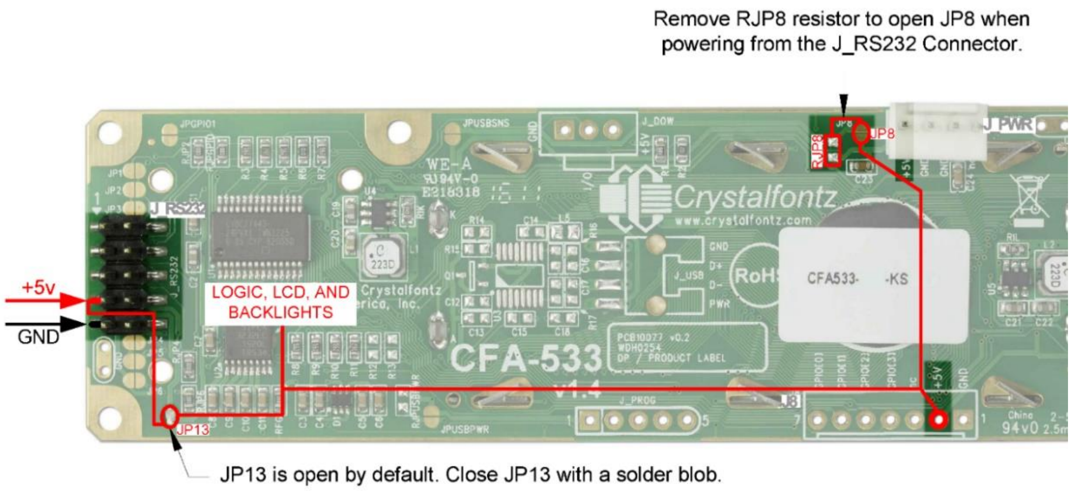
1. JP8 is closed with the RJP8 0KΩ resistor by default. Leave JP8 closed so that the J8 connector is electrically connected to J_PWR. The connector loaded at J_PWR is (Tyco Electronics / Amp part number 4-171825-4, Mouser Electronics part number 571-4-171825-4).
2. Supply +5v to pin 1 and Ground to pin 2 or 3 on the J_PWR connector. Use the Crystalfontz cable WR-PWR-Y12 (or equivalent) to connect from the host's 4-pin power supply cable connector to the display's J_PWR connector.

7.2. Connection through J8 Connector (Standard)


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

1. Open JP8 by removing the RJP8 resistor. This enables powering from the J8 connector, rather than the default, powering through J_PWR.
2. Supply +5v to pin2 and ground to pin 1 on the J8 header.
3. The module can be shipped with the J8 header installed. Select "Configure and Add to Cart" and add the J8 header option.

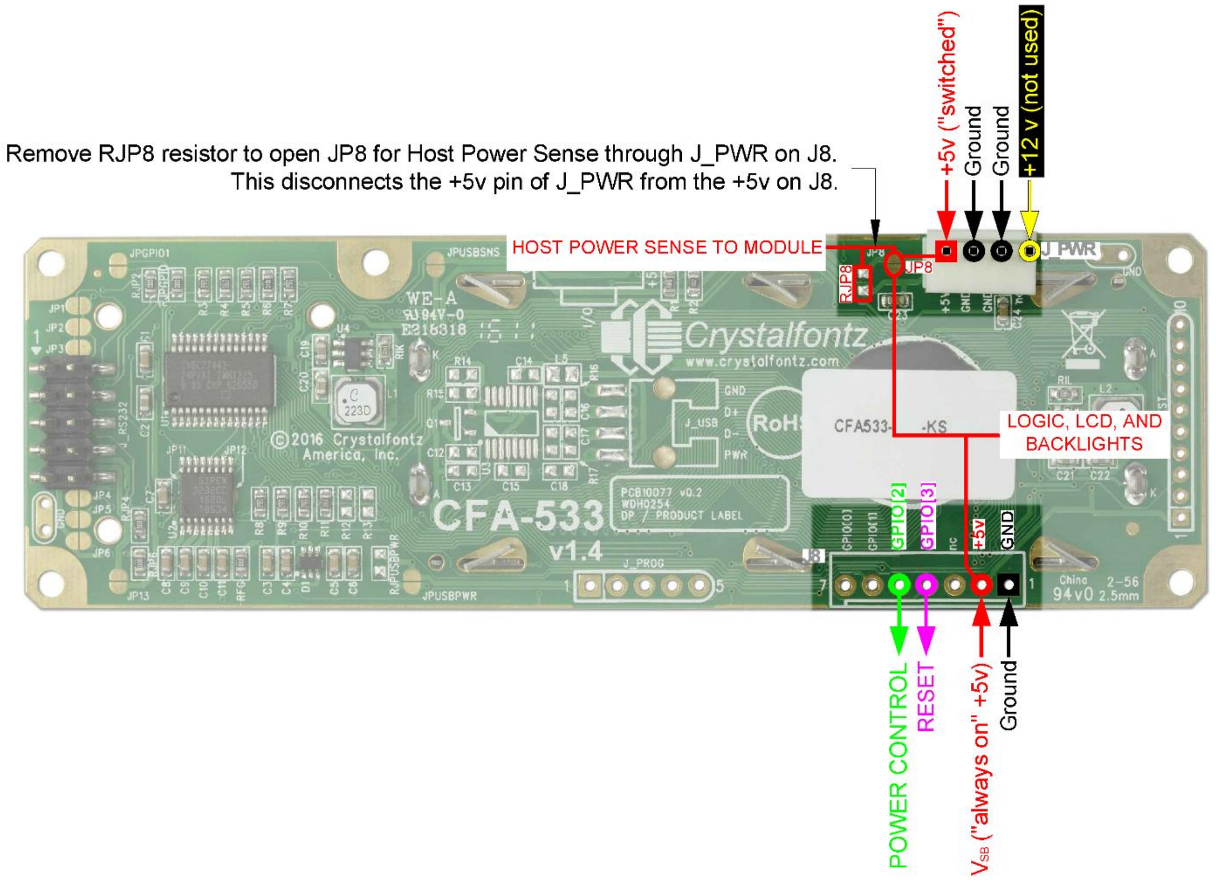
7.3. Connection through J_RS232 Connector (Standard)


Remove RJP8 resistor to open JP8 when powering from the J_RS232 Connector.
JP8
RJP8
J_PWR
J_RS232
+5v
GND
LOGIC, LCD, AND BACKLIGHTS
JP13
J8
JP13 is open by default. Close JP13 with a solder blob.

The V_{DD} power can be supplied through the J-RS232 connector, allowing a single cable for both power and data connections

1. Close JP13 with a solder blob.
2. Open JP8 by removing the RJP8 resistor.
3. By default, the five necessary connections are all on the outside column of pins on J_RS232, and a single 0.1-inch pitch 5-conductor cable can be connected between the module and host.

7.4. ATX Host Power Sense through +5v on J_PWR


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

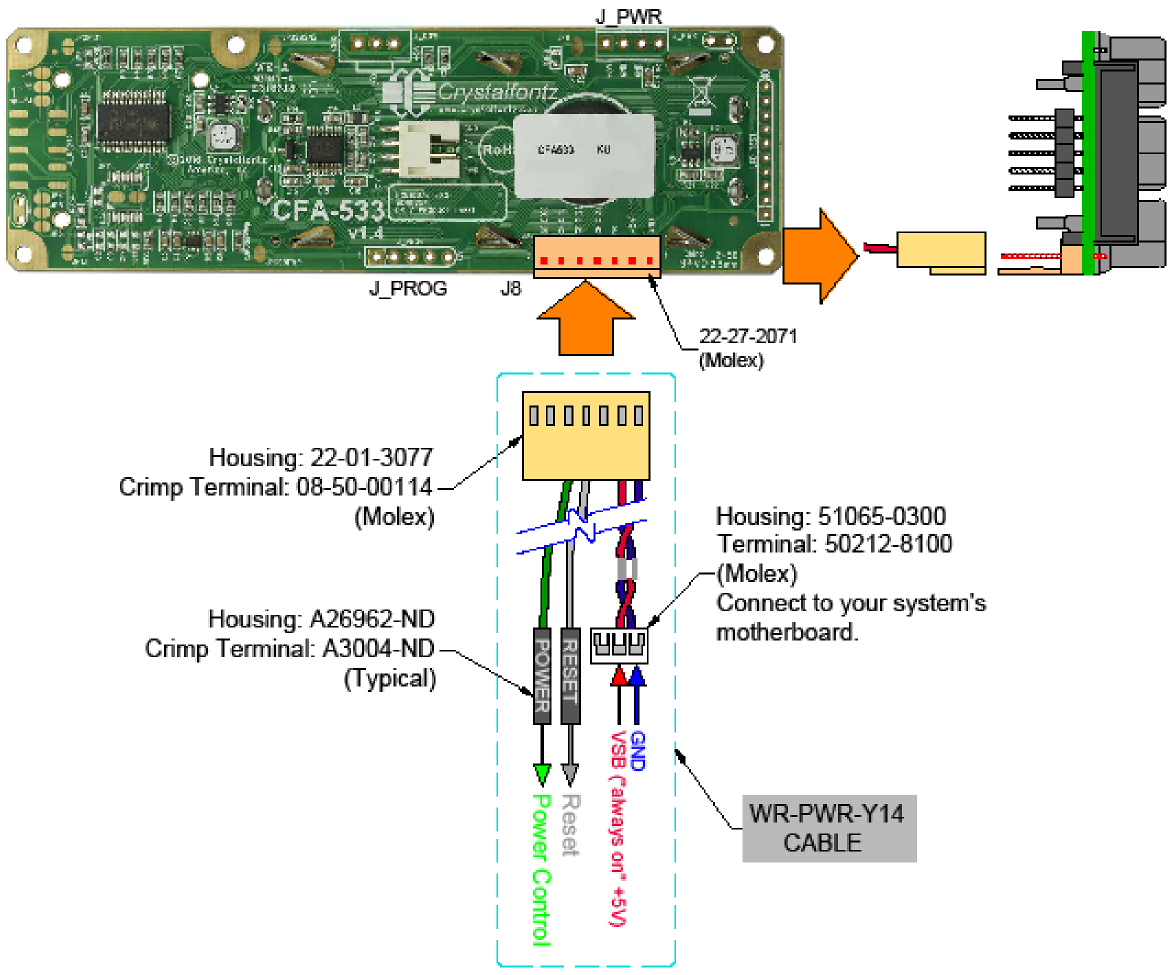
By default, the pin labeled +5v on the J_PWR connector is electrically connected to the +5v pin on the J8 connector through the normally closed JP8. To use the CFA533 for ATX power supply control, open jumper JP8 by removing the RJP8 resistor. This disconnects the +5v pin of the J_PWR connector from the +5v of the J8 connector. The +5v pin of the J_PWR connector will then function as the “Host Power Sense”. The +5v pin of the J8 connector will function as VSB power to the module.

Connect the motherboard's power switch input to pin 5 (labeled as GPIO2) of the CFA533’s J8 connector. This pin functions as POWER CONTROL. The POWER CONTROL pin is a high--impedance input until the display module signals 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.

Connect the motherboard's reset switch input to pin 4 (labeled as GPIO3) of the CFA533’s J8 121--connector. This pin functions as RESET. The RESET pin is configured as a high-impedance input until the display module resets the host. Then it will change momentarily to low-impedance output, driving either low or high.

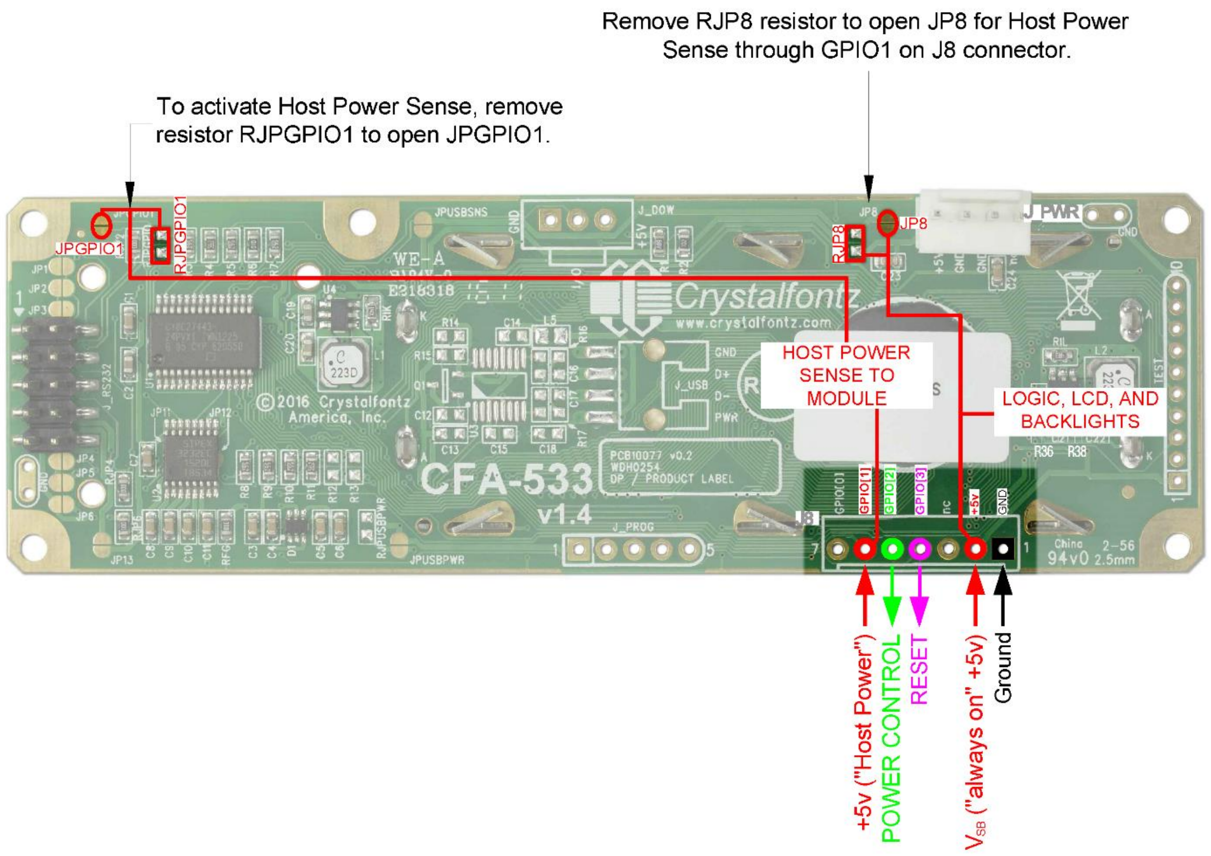
For more about ATX control, see command 28 (0x1C): Ser ATX Switch Functionality.

To simplify making these connections, the WR-PWR-Y14 and WR-PWR-Y44 connect to J8 and includes appropriate connectors.



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

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


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

The CFA533 can be configured to sense host power through GPIO[1] on connector J8. In addition to +5VSB, Ground, Power Control (GPIO[2]), and Reset Control (GPIO[3]) connections, connect the host's +5v power to GPIO[1].

JP8 is closed by default. To properly function, JP8 must be opened by removing RJP8 resistor. To activate Host Power Sense, remove resistor RJPGPIO1 to open JPGPIO1.

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

For more about ATX control, see command 28 (0x1C): Ser ATX Switch Functionality.

7.6. GPIO Connections

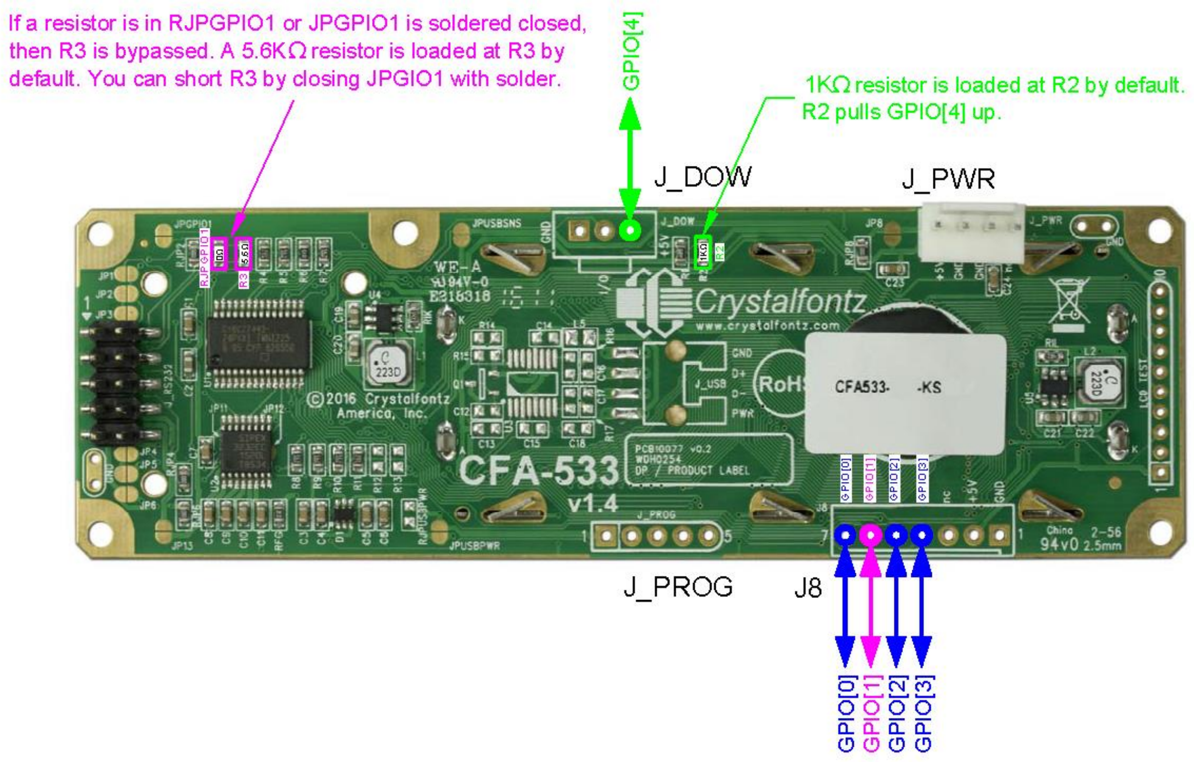
The CFA533 has five General Purpose Input/Output (GPIO) pins. The GPIO are port pins from the CFA533-xxx-KS's micro-controller brought out to connectors. As an output, a GPIO can be used to turn on an LED, or perhaps drive a relay. As an input, a GPIO can be used to read a switch or a button. Most of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

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

GPIO[0], GPIO[2] and GPIO[3] are connected directly from the micro-controller port pin to the connector pin. GPIO[1] has a series 5.6KΩ resistor in R3.

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

Please refer to commands 34 (0x22): Set/Configure GPIO (Pg. 52) and 35 (0x23): Read GPIO Pin Levels and Configuration State (Pg. 53) for additional details concerning the GPIO operation.


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

8. Host Communications

8.1. RS-232 Connections

The CFA533-xxx-KS 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.

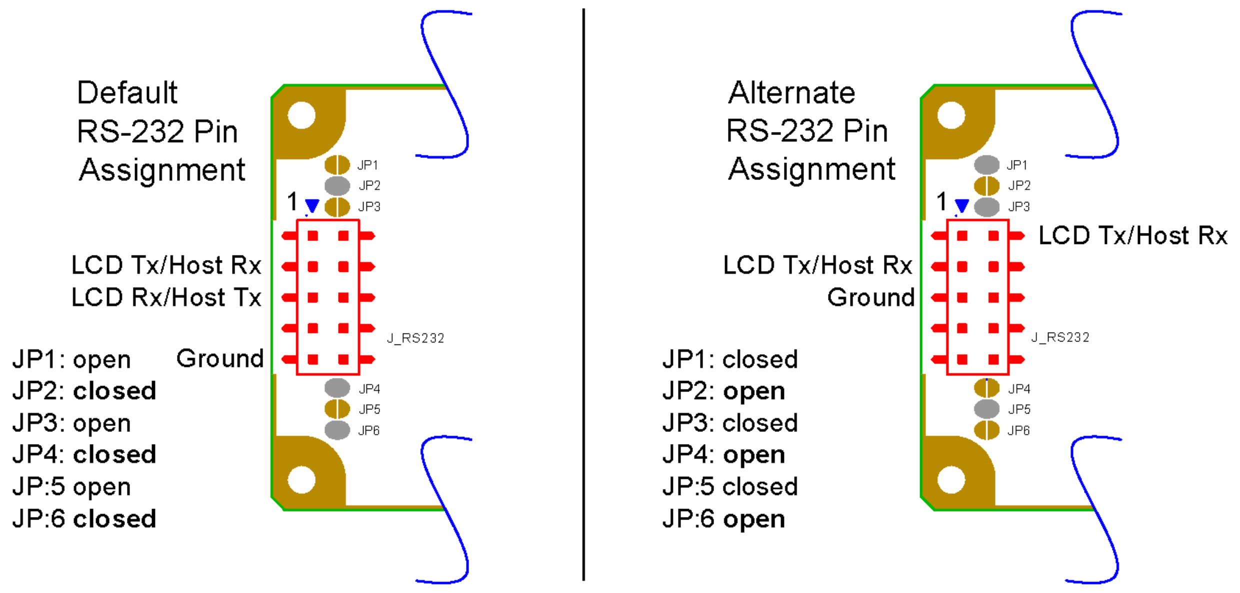
By default, with JP2, JP4, and JP6 all closed, the default RS232 pin assignments are selected. This selection allows a low-cost ribbon cable, such as the WR-232-08, to connect to a DB9 COM port.

To connect to a host system that uses a 10-pin 0.1-inch connector instead of the standard RS232 DB9, the alternate pin assignment may be needed.

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

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

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



8.2. Packet Structure

Communication between the CFA533 and the host takes place in the form of a simple and robust CRC-checked packet. The packet format enables very reliable communications between the CFA533 and the host without the traditional problems that occur in a stream-based serial communication.

Reconciling packets is recommended rather than using delays when communicating with the module. To reconcile packets, ensure the acknowledgement packet from the packet most recently sent is received before sending additional packets to the LCD module. This protects against dropped packets or missed communication with the LCD module.

The packets follow this structure: <type><data_length><data><CRC>

Alternately, it may be useful to think of the packet as follows:

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

8.2.1. <type>

<type> is one byte that identifies the type and function of the packet. The first two bits indicate the type of packet (command, response, report, error) and the last six bits encode the details.

```
TTcc cccc
|||| ||||--Command, response, error or report code 0-63₁₀
||---------Type:
00 = normal command from host to CFA533
01 = normal response from CFA533 to ost
10 = normal report from CFA533 to host (not in direct 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)
```

8.2.2. <data_length>

<data_length> is one byte that specifies the number of bytes that will follow in the **<data>** field. The valid range for **<data_length>** is 0-18$_{10}$.

8.2.3. <data>

<data> is the payload of the packet. Each command is associated with a **<data_length>**. The data field contains up to 18 bytes of information related to the command specified in the **<type>**.

8.2.4. <crc>

<crc> is a standard 16-bit CRC (cyclic redundancy check) which verifies all the information in the packet, excluding the **<crc>** itself. The **<crc>** immediately follows the last used element of **<data>**, and is sent LSB first.
See Appendix A for several examples of how to calculate the CRC in a variety of programming languages.

Additionally, Crystalfontz supplies a demonstration and test program, cfTest, along with its C source code. Included in the cfTest source is a CRC algorithm and an algorithm that detects packets. The algorithm automatically re-synchronizes to the next valid packet in the event of communications errors. Follow the algorithm in the sample code closely to realize the benefits of using packet communications.

8.3. 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.

Note that some operating systems 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.

8.4. 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:

8.4.1. 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.

```
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
```

8.4.2. 0x81: Reserved

8.4.3. 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.

```
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  MSB  of  Temperature_Sensor_Counts
data[2]  is  the  LSB  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,"B
AD 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);
  }
}
```

8.5. 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 host sends a packet include ng:

```
type: 0x00 = 0000 0000₂ = 0₁₀
data_length: between 0 and 16
data[0-(data_length-1)]: any data can be sent
```

The return packet is identical to the packet sent by the host, except the first two bits of the type now indicate the packet is a normal response to the host:

```
type: 0x40 | 0x00 = 0x40 = 0100 0000₂ = 64₁₀
data_length: between 0 and 16
data[0-(data_length-1)]: same data as sent by host
```

1 (0x01): Get Hardware & Firmware Version

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

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

The return packet will be:

```
type: 0x40 | 0x01 = 0x41 = 65₁₀
data_length: 16
data[] = "CFA533:XhX, YsY”

XhX is the hardware revision, "1h4" for example
YsY is the firmware version, "1s2" for example
```

NOTE: Hardware version v1.4 is printed on back of PCB.

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 a serial number, IP address, gateway address, netmask, or any other data required. All 16 bytes must be supplied.

```
type: 0x02 = 2₁₀
data_length: 16
data[] = 16 bytes of arbitrary user data to store 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
data_length: 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 Crystalfontz boot screen when power is applied. This command can be used to customize the boot screen, along with the following settings:

- The contents of the DDRAM (characters shown on LCD), affected by:
  - Command 6 (0x06): Clear LCD Screen.
  - Command 7 (0x07): Set LCD Contents, Line 1. (Deprecated)
  - Command 8 (0x08): Set LCD Contents, Line 2. (Deprecated)
  - Command 31 (0x1F): Send Data to LCD.
- Command 9 (0x09): Set LCD Special Character Data
- Command 11 (0x0B): Set LCD Cursor Position
- Command 12 (0x0C): Set LCD Cursor Style
- Command 13 (0x0D): Set LCD Contrast
- Command 14 (0x0E): Set LCD & Keypad Backlight
- Command 21 (0x15): Set Up Live Temperature Display*
- Command 23 (0x17): Configure Key Reporting
- Command 28 (0x1C): Set ATX Switch Functionality**
- Command 33 (0x21): Set Baud Rate
- Command 34 (0x22): Set/Configure GPIO

*Temperature reporting cannot be stored, though the live display of temperatures can be saved.
** The host watchdog cannot be stored. The host software should enable this item once the system is initialized and ready to receive the data.

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

The return packet will be:

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

Note:
If the current state and the boot state do not match after saving, the module will return an error instead of an ACK. In this unlikely error case, the boot state will be undefined.

Saving the boot state may not work properly at voltages lower the +5v. It is recommended to only save the boot state when operating at +5v logic. Saving the boot state at a +3.3v logic level may cause corrupted characters to appear on the display module.

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 GPIO pins used for ATX control must not be configured as user GPIO, and must be configured in default drive mode for the ATX functions to work correctly. These may be changed by the user, see command 34 (0x22): Set or Set and Configure GPIO Pins.

Rebooting the CFA533 may be useful when testing the boot configuration. It may also be useful to re-enumerate the devices (WR-DOW-Y17 temperature sensors), on the 1-Wire bus. To reboot the CFA533, send the following packet:

```
type: 0x05 = 5₁₀
data_length: 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 Power Switch Functionality, send the following packet:

```
type: 0x05 = 5₁₀
data_length: 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 Power Switch Functionality, send the following packet:

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

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

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

6 (0x06): Clear LCD Screen

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

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

The return packet will be:

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

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

7 (0x07): Set LCD Contents, Line 1 and 8 (0x08): Set LCD Contents, Line 2

These commands set the 16 characters displayed on either the top (command 7) or bottom (command 8) line of the display. These commands have been replaced by Command 31: Send Data to LCD. The commands are still supported for backwards compatibility in legacy systems. These commands affect the DDRAM which is saved by Command 4: Store Current State as Boot State.

9 (0x09): Set LCD Special Character Data

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

```
type: 0x09 = 9₁₀
data_length: 9
data[0] = index of special character 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 the character. Any value between 0 and 31 is valid. 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 stored by command 4 (0x04): Store Current State as Boot State.

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

This command returns the contents of the LCD's DDRAM or CGRAM. This command is intended for debugging.

```
type: 0x0A = 10₁₀
data_length: 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 (\143) for DDRAM, line 1
0xC0 (\192) to 0xD3 (\207) for DDRAM, line 2
```

The return packet will be:

```
type: 0x40 | 0x0A = 0x4A = 74₁₀
data_length: 9
data[0]: address code.
data[1-8]: data read from the LCD controller's memory.
```

11 (0x0B): Set LCD Cursor Position

This command places the cursor at the given location on the CFA533's LCD screen. Cursor visibility is set by Command 12 (0x0C): Set LCD Cursor Style.

```
type: 0x0B = 11₁₀
data_length: 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 stored by the command 4 (0x04): Store Current State as Boot State.

12 (0x0C): Set LCD Cursor Style

This command selects among four hardware generated cursor options.

```
type: 0x0C = 12₁₀
data_length: 1
data[0] = cursor style (0-3 valid)
        0 = no cursor.
        1 = blinking block cursor.
        2 = static underscore cursor.
        3 = blinking underscore cursor.
```

The return packet will be:

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

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

13 (0x0D): Set LCD Contrast

This command sets the contrast of the display.

CFA533 Enhanced

Using two bytes to set the contrast takes advantage of the CFA533's native enhanced contrast resolution (compared to the CFA633). The first byte, data[0], simply indicates the enhanced version, any value from 0 to 254 is accepted. The second byte, data[1], controls the CFA533 contrast resolution.

```
type: 0x0D = 13₁₀
data_length: 2
data[0]: required but ignored
data[1]: contrast setting (0-200 valid)
      0-99 = lighter
      100 = no correction
      101-200 = darker
```

CFA633 Compatible

The CFA633 compatible version allows the contrast to be set using only 1 byte.

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

The return packet for either method is:

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

Set LCD Contrast is stored by Command 4 (0x04): Store Current State as Boot State.

14 (0x0E): Set LCD & Keypad Backlight

This command sets the brightness of the LCD and keypad backlights.

CFA533 Enhanced

Using two bytes allows the LCD and keypad brightness to be separately set. The LCD brightness is set by the first byte and the keypad by the second byte.

```
type: 0x0E = 14₁₀
data_length: 2
data[0]: LCD backlight power setting (0-100 valid)
      0 = off
      1-99 = variable brightness
      100 = on
data[1]: keypad backlight power setting (0-100 valid)
      0 = off
      1-99 = variable brightness
      100 = on
```

CFA633 Compatible

Using one byte sets both the keypad and LCD backlights to the same brightness. This method is CFA633 compatible.

```
type: 0x0E = 14₁₀
data_length: 1
data[0]: keypad and LCD backlight power setting (0-100 valid)
     0 = off
     1-99 = variable brightness
     100 = on
```

The return packet for either method is:

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

Set LCD & Keypad Backlight is stored by Command 4 (0x04): Store Current State as Boot State.

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

18 (0x12): Read DOW Device Information

This command provides device information about devices connected to the 1-Wire (DOW) bus. On power-up, the CFA533 detects any devices connected to the bus and stores the device information. The first byte returned is the "family code" of the 1-Wire device. A list of the possible 1-Wire device family codes is available on the Maxim website.

```
type: 0x12 = 18₁₀
data_length: 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
```

Note: In order for the DOW subsystem to operate correctly, GPIO[4] must be configured in the default drive mode, as follows:

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

This can be achieved by sending the following command and saving the boot state (Command 4):

```
type: 34
data_length: 3
data[0]: 4
data[1]: 100
data[2]: 7
```

19 (0x13): Set Up Temperature Reporting

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

```
type: 0x13 = 19₁₀
data_length: 4
data[0-3] = 32-bit bitmask indicating which temperature sensors are enabled
to report (0-255 valid in each location)
data[0]
08 07 06 05   04 03   02 01  Enable Reporting of sensor with
|  |  |  |    |  |    |  |   device index of:
|  |  |  |    |  |    |  |--  0: 1 = enable, 0 = disable
|  |  |  |    |  |    |-----  1: 1 = enable, 0 = disable
|  |  |  |    |  |---------  2: 1 = enable, 0 = disable
|  |  |  |    |------------  3: 1 = enable, 0 = disable
|  |  |  |-----------------  4: 1 = enable, 0 = disable
|  |  |--------------------  5: 1 = enable, 0 = disable
|  |-----------------------  6: 1 = enable, 0 = disable
|--------------------------  7: 1 = enable, 0 = disable
```

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

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

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

The return packet will be:

type: 0x40 | 0x13 = 0x53 = 83_{10}
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 is sufficient for many devices, but some devices require larger transactions and cannot be fully used with the CFA533.

This command allows 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 one of write_phase or read_phase must be sent
<read_phase>   //optional, but one of write_phase or read_phase must be sent

type: 0x14 = 20₁₀
data_length: 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

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, so the CFA533 can immediately display system temperatures as soon as power is applied.

The live display is based on a concept of display slots. There are 8 slots, and each of the 8 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
data_length: 7 or 2 (for turning a slot off)
data[0]: display slot (0-3)
data[1]: type of item to display in this slot
       0 = nothing (data_length then must be 2)
       1 = (invalid)
       2 = temperature (data_length then must be 7)
```

```
data[2]: index of the sensor to display in this slot:
       0-31 are valid (and the temperature device must be attached)
data[3]: number of digits
       3 digits (-XX or XXX)
       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 HD44780 compatible. Generally, low-level access to the LCD controller is unnecessary, but some arcane functions of the HD44780 are not exposed by the CFA533's command set. This command allows access to the CFA533's LCD controller directly.

IMPORTANT: 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.

```
#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 stored by the command 4 (0x04): Store Current State as Boot State.

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. 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 pressed since the last poll
data[2] = bit mask showing the keys released since the last poll
```

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

28 (0x1C): Set ATX Power 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 stored by the command 4 (0x04): Store Current State as 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.

REGARDING 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 default but may be changed by the user. See command 34 (0x22): Set or Set and Configure GPIO Pins. These settings must be saved as the boot state.

To ensure 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 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 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.

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.

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

The return packet will be:

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

29 (0x1D): Enable/Disable and Reset the Watchdog

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 reset the CFA533's watchdog timer, the CFA533 will reset the host system.

> 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. See the note under command 28 (0x1C): Set ATX Power Switch Functionality or command 34 (0x22): Set or Set and Configure GPIO Pins.

```
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 avoid a watchdog reset.
```

To turn the watchdog off, simply set the timeout to 0. If the command is not re-issued within timeout seconds, the CFA533 will reset the host (see command 28). As the watchdog is off by default when the CFA533 powers up, the CFA533 will not issue a host reset until the host has re-enabled the watchdog.

The return packet will be:

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

30 (0x1E) Read Reporting & Status

This command verifies the items configured to report to the host, and other status information. The information returned by the CFA533 differs from the information returned by similar Crystalfontz displays.

```
type = 0x1E = 30₁₀
data_length: 0
```

The return packet will be:

```
type = 0x40 | 0x1E = 0x5E = 94₁₀
data_length: 15

data[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[0] (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])
```

NOTE: Previous and 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 stored by the command 4 (0x04): Store Current State as Boot State.

32 (0x20): Reserved

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 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 for 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 or Set and Configure GPIO Pins

This command configures the five user-definable general-purpose input / output (GPIO) pins. These pins are shared with the DOW and ATX functions. DOW and ATX functions require specific GPIO settings and changing the related GPIO pin settings may cause undesired behavior from the DOW and ATX systems.

The architecture of the CFA533 allows flexibility in the configuring the GPIOs. 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. In output mode using PWM (and a current limiting resistor), an LED may be turned on, off, or dimmed under host software control. With external circuitry, the GPIOs can 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 keeps track rising and falling edges between host queries (subject to the resolution of the 32 Hz sampling) so the host is not forced to poll quickly in order to detect short events.

The algorithm used by the CFA533 to read the inputs is inherently “debounced”.

The GPIOs also have “pull-up” and “pull-down” modes. These modes can be useful when using the GPIO as an input connected to a switch since no external pull-up or pull-down resistor is needed. For instance, the GPIO can be set to pull-up. 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 a 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. R3, a 5.6kΩ resistor is in series with GPIO[1] to limit the possibility of latchup.

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

```
type: 0x22 = 34₁₀
data_length: 2 bytes to change value only
             3 bytes to change value and configure function and drive mode
data[0]: index of GPIO to modify
       0 = GPIO[0] = J8, Pin 7
       1 = GPIO[1] = J8, Pin 6 (may be ATX Host Power Sense, as configured
       by command 28, data[2])
       2 = GPIO[2] = J8, Pin 5 (default is ATX Host Power Control)
       3 = GPIO[3] = J8, Pin 4 (default is ATX Host Reset Control)
```

```
        4 = GPIO[4] = J_DOW, Pin 2 (default is DOW I/O -- has 1kΩ resistor
        hardware pull-up: R2)
        5-255 = reserved
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)
        ---- 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
```

For DOW on GPIO[4], data[2] should be 7 (`111: 1=Hi-Z, 0=Slow, Strong Drive Down`).
For ATX POWER CONTROL and RESET, data[2] should be 2 (`010: Hi-Z, use for input`). If using GPIO[1] for HOST POWER SENSE, data[2] should be 3 (`011: 1=Resistive Pull Up, 0=Fast, Strong Drive Down`).

The return packet will be:

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

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

See command 34 (0x22): Set or Set and Configure GPIO Pins 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 (default is ATX Host Power Sense)
     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--always has 1KΩ hardware
        pull-up on SCAB.)
     5-255 = reserved
```

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.

The return packet will be:

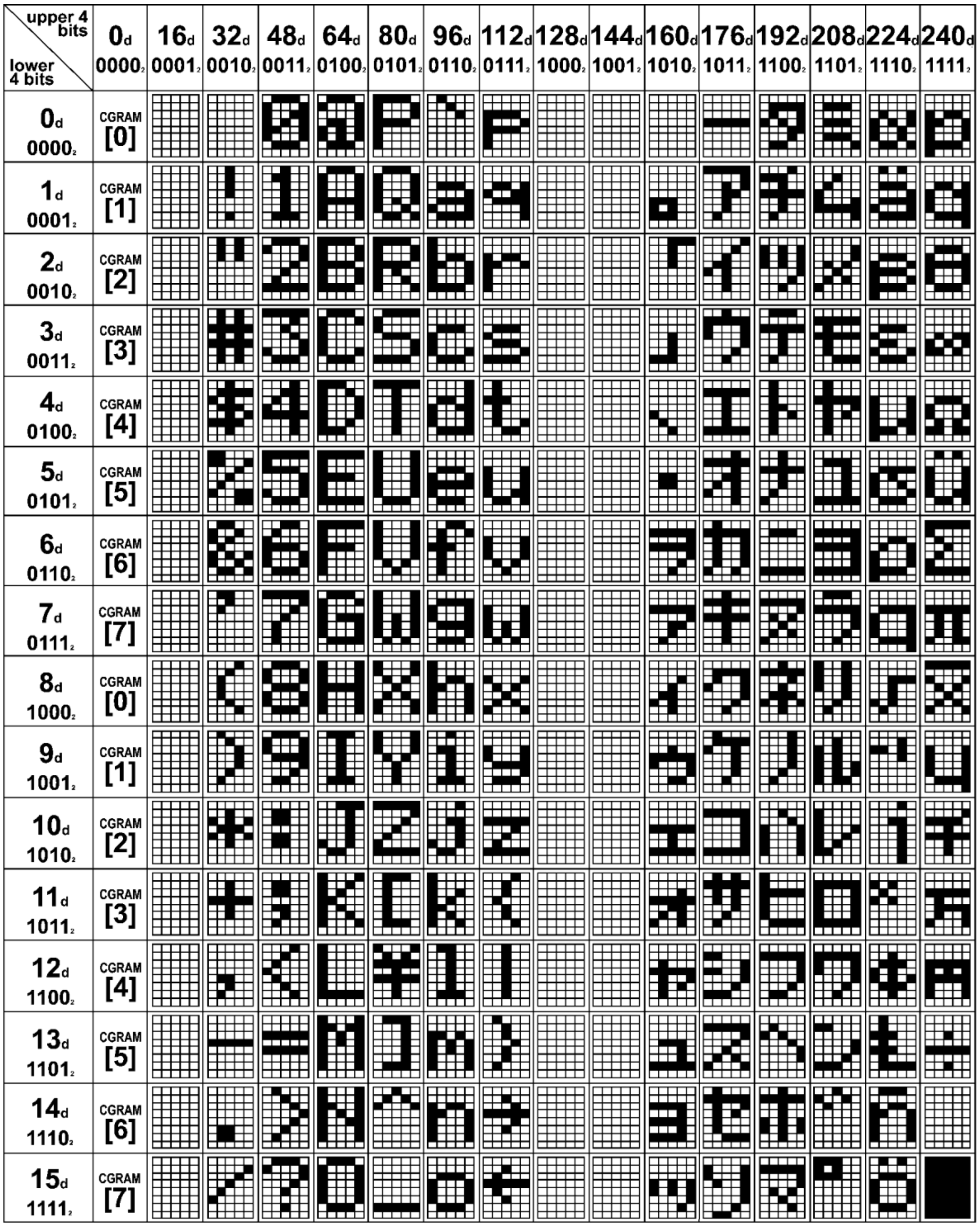
```
type = 0x40 | 0x23 = 0x63 = 99₁₀
data_length: 4
returns:
  data[0] = index of GPIO read
  data[1] = Pin state & changes since last poll

    ---- -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
```

9. Character Generator ROM (CGROM)

To find the code for a given character, add the two numbers 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”. Add 224 + 2 to get 226. When a byte with the value of 226 is sent to the display, the Greek letter "β" will be shown.


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

Character Generator ROM (CGROM)

10. LCD Module Reliability and Longevity

We work to continuously improve our products, including backlights that are brighter and last longer. Slight color variations from module to module and batch to batch are normal. ***For modules with consistent color, please ask for a custom order.***

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

10.1. Module Longevity (EOL / Replacement Policy)

Crystalfontz is committed to making all of our LCD modules available for as long as possible. For each module that 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 Datasheet specifications and tolerances of the discontinued module, changes may require modification to your circuit and/or firmware. Possible changes include:

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

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

11. Care and Handling Precautions

For optimum operation of the module and to prolong its life, follow the precautions described below.

11.1. ESD (Electrostatic Discharge)

This circuitry is industry standard CMOS logic and susceptible to ESD damage. 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.

11.2. Design and Mounting

- The exposed surface of the “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. 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 module, avoid touching the polarizer. Finger oils are difficult to remove.
- Module without Crystalfontz overlay: Protect the soft plastic polarizer from damage, with a transparent plate (for example, acrylic, polycarbonate or glass) in front of the module, leaving a small gap between the plate and the display surface.
- Do not disassemble or modify the module.
- Do not modify the six tabs of the metal bezel or make connections to them.
- Do not reverse polarity to the power supply connections. This will immediately ruin the module.

11.3. Avoid Shock, Impact, Torque, or Tension

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

11.4. 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.

11.5. Cleaning

- The polarizer (laminated to the glass) is soft plastic. The soft plastic is easily scratched or damaged. Be very careful when cleaning 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 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, carefully blow it off with clean, dry, oil-free compressed air.
- Module 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.

11.6. Operation

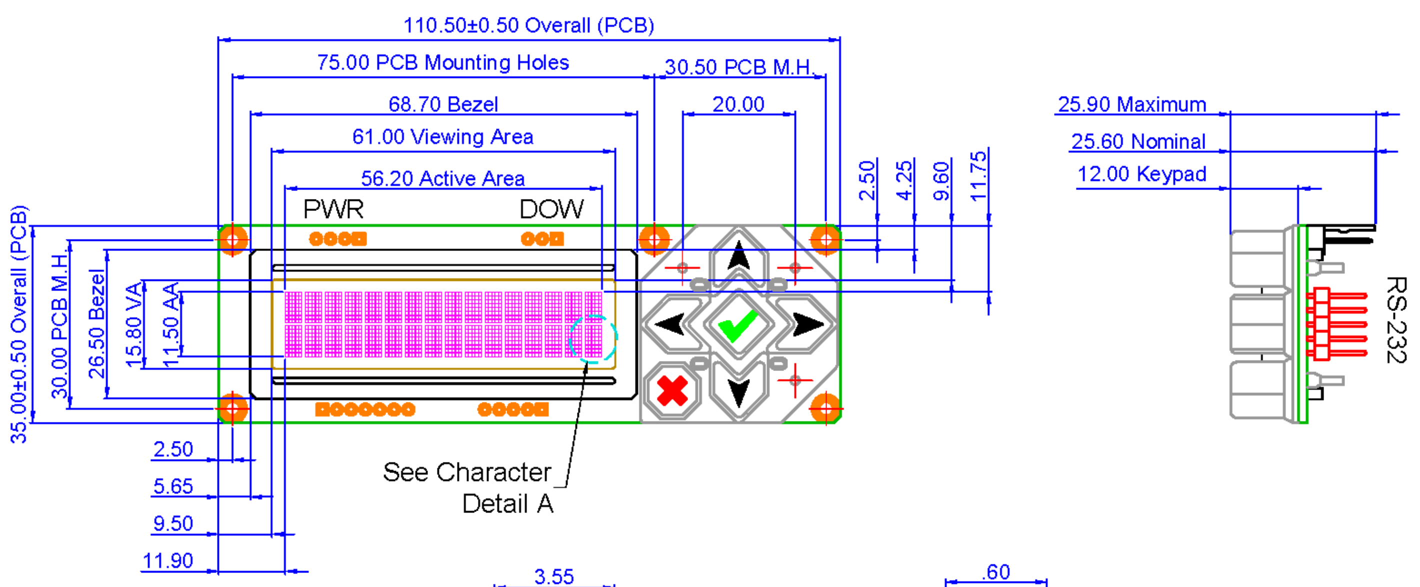
- Protect the module from ESD and power supply transients.
- Observe the operating temperature limitations: a minimum of -20°C to a maximum of +70°C with minimal fluctuation. Operation outside of these limits may shorten life and/or harm display.
- At lower temperatures of this range, response time is delayed.
- At higher temperatures of this range, display becomes dark and the contrast will need adjusting.
- Operate away from dust, moisture, and direct sunlight.
- Adjust backlight brightness so the display is readable but not too bright. Dim or turn off the backlight during periods of inactivity to conserve the white LED backlight lifetime.

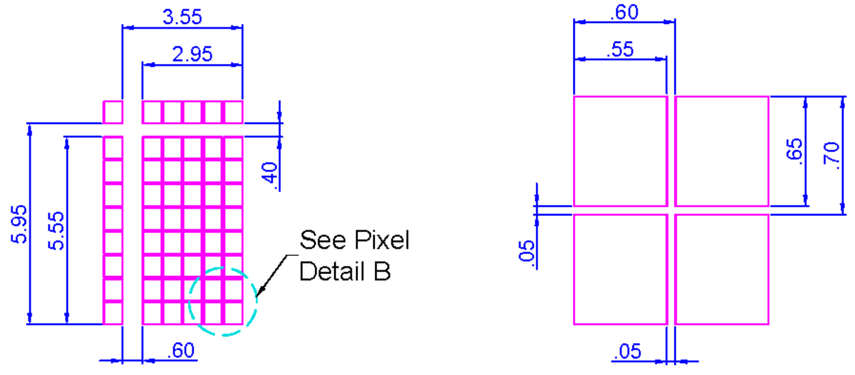
11.7. Storage and Recycling

- Store in an ESD-approved container away from dust, moisture, and direct sunlight.
- Observe the storage temperature limitations: -30°C minimum, +80°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 modules while they are in storage.
- Please recycle outdated Crystalfontz modules at an approved facility.

12. Mechanical Drawings

CFA533 Module Outline Drawing (1 of 2)





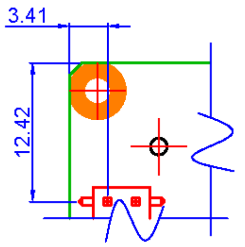
Character Detail A

Pixel Detail B

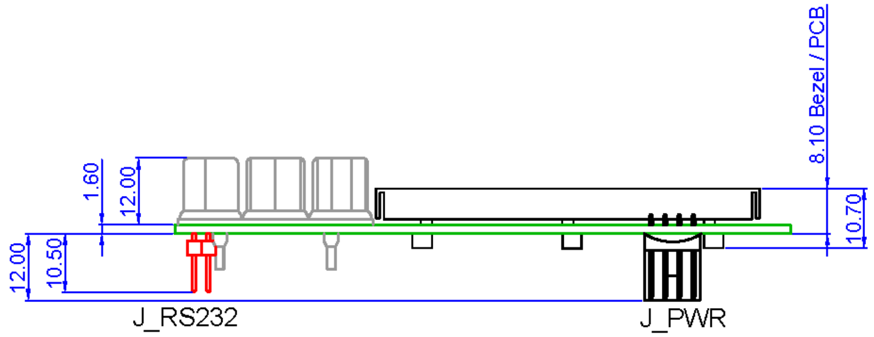
Illustration is deemed accurate but not guaranteed..

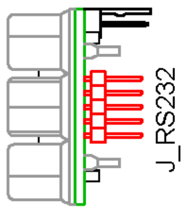
Part No.(s): CFA533-TFH-KS CFA533-TMI-KS CFA533-YYH-KS	Scale: Not to scale	Drawing Number: CFA533_master	Hardware Rev.: v1.4
	Units: Millimeters	Date: 2016-07-28	Sheet: 1 of 2

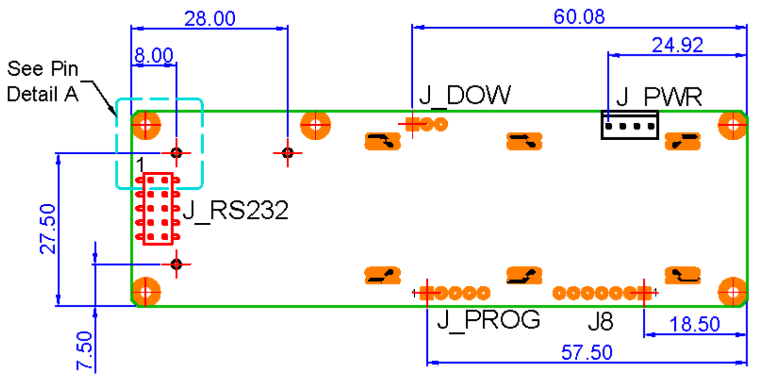
CFA533 Module Outline Drawing (2 of 2)



Pin Detail A





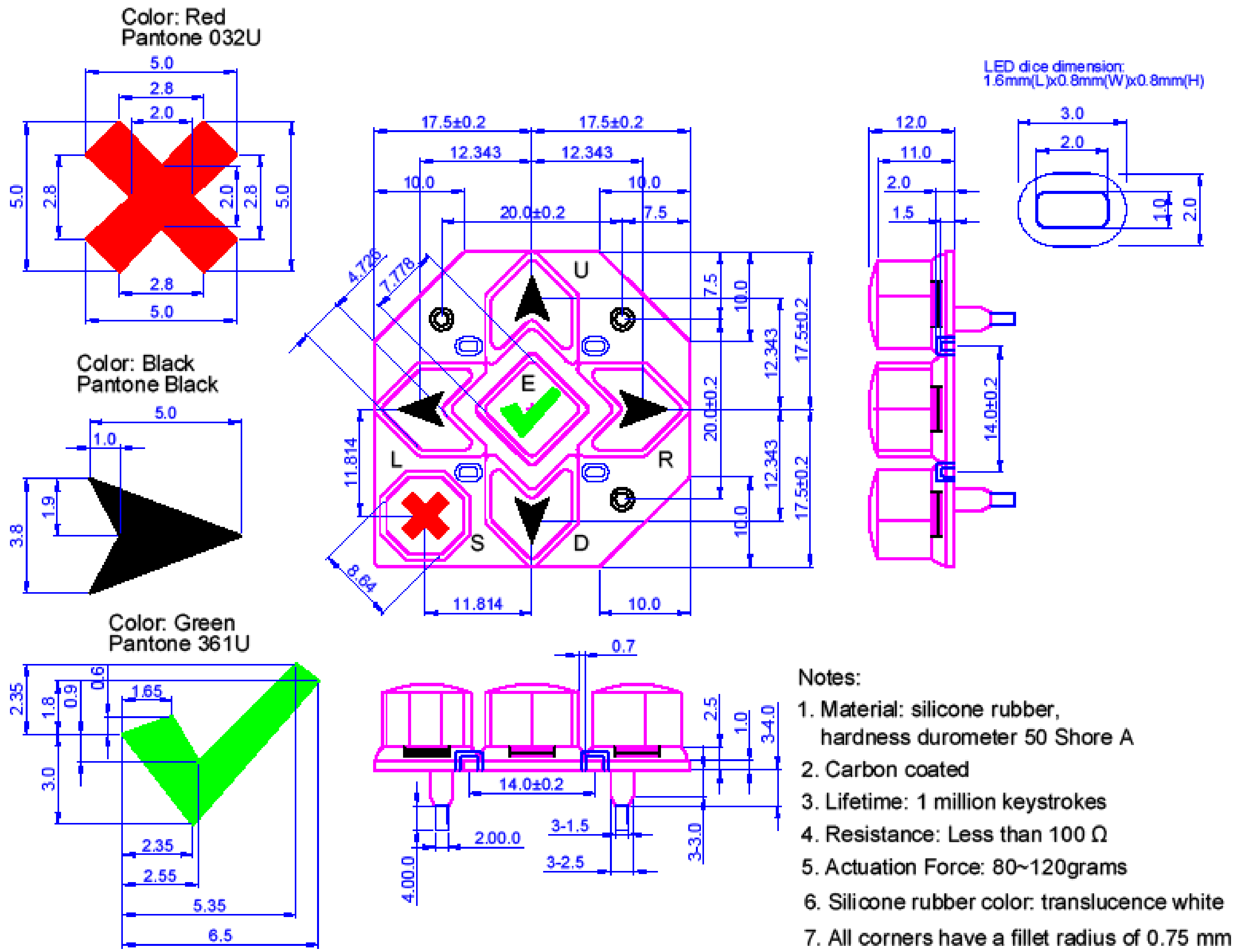


Back View

Illustration is deemed accurate but not guraanteed.

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

Keypad Detail Drawing

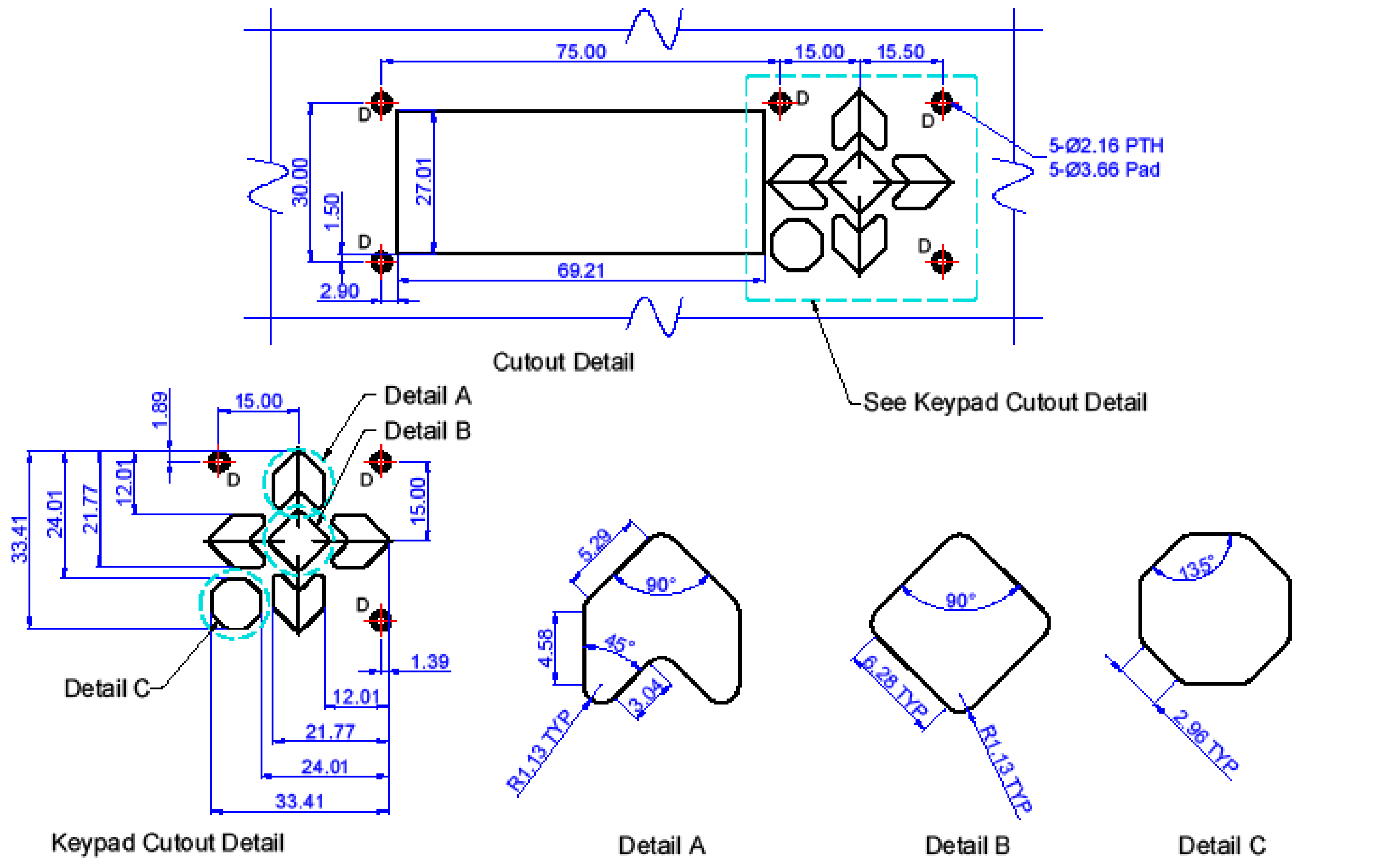


Notes:

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

Copyright © 2017 by Crystalfontz America, Inc. www.crystalfontz.com/products/	Part No.(s): CFA533 Series Keypad Detail	Scale: Not to scale	Drawing Number: CFA533_master	Hardware Rev.: v1.4
		Units: Millimeters	Date: 2017-03-14	Sheet: 1 of 1

Panel Mounting Application Cutout Drawing



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

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

Copyright © 2017 by Crystalfontz America, Inc. www.crystalfontz.com/products/	*Part No.(s):* CFA533 Panel Mounting Application Detail	*Scale:* Not to scale	*Drawing Number:* Panel_master	*Hardware Rev.:* v1.4
		Units: Millimeters	*Date:* 2017-03-14	*Sheet:* 1 of 1

13. Appendix A: Demonstration Software and Sample Code

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.
  https://www.crystalfontz.com/product/cftest
- Linux compatible command-line demonstration program with C source code. 8K.
  https://www.crystalfontz.com/product/linuxexamplecode
- Supported by CrystalControl freeware.
  https://www.crystalfontz.com/product/CrystalControl2.html

In addition, see http://lcdproc.org/index.php3 for Linux LCD drivers. LCDproc is an open source project that supports many of the Crystalfontz displays.

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 CFA631 and CFA635. The CRC used in the CFA533 is the same one that is used in IrDA, which came from PPP, which 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,0x0AC42,0x09ED9,0x08F50,0x0FBEF,0x0EA66,0x0D8FD,0x0C974,
     0x04204,0x0538D,0x06116,0x0709F,0x00420,0x015A9,0x02732,0x036BB,
     0x0CE4C,0x0DFC5,0x0ED5E,0x0FCD7,0x08868,0x099E1,0x0AB7A,0x0BAF3,
     0x05285,0x0430C,0x07197,0x0601E,0x014A1,0x00528,0x037B3,0x0263A,
     0x0DECD,0x0CF44,0x0FDDF,0x0EC56,0x098E9,0x08960,0x0BBFB,0x0AA72,
     0x06306,0x0728F,0x04014,0x0519D,0x02522,0x034AB,0x00630,0x017B9,
     0x0EF4E,0x0FEC7,0x0CC5C,0x0DDD5,0x0A96A,0x0B8E3,0x08A78,0x09BF1,
   0x07387,0x0620E,0x05095,0x0411C,0x035A3,0x0242A,0x016B1,0x00738,
   0x0FFCF,0x0EE46,0x0DCDD,0x0CD54,0x0B9EB,0x0A862,0x09AF9,0x08B70,
   0x08408,0x09581,0x0A71A,0x0B693,0x0C22C,0x0D3A5,0x0E13E,0x0F0B7,
   0x00840,0x019C9,0x02B52,0x03ADB,0x04E64,0x05FED,0x06D76,0x07CFF,
   0x09489,0x08500,0x0B79B,0x0A612,0x0D2AD,0x0C324,0x0F1BF,0x0E036,
   0x018C1,0x00948,0x03BD3,0x02A5A,0x05EE5,0x04F6C,0x07DF7,0x06C7E,
   0x0A50A,0x0B483,0x08618,0x09791,0x0E32E,0x0F2A7,0x0C03C,0x0D1B5,
   0x02942,0x038CB,0x00A50,0x01BD9,0x06F66,0x07EEF,0x04C74,0x05DFD,
   0x0B58B,0x0A402,0x09699,0x08710,0x0F3AF,0x0E226,0x0D0BD,0x0C134,
   0x039C3,0x0284A,0x01AD1,0x00B58,0x07FE7,0x06E6E,0x05CF5,0x04D7C,
   0x0C60C,0x0D785,0x0E51E,0x0F497,0x08028,0x091A1,0x0A33A,0x0B2B3,
   0x04A44,0x05BCD,0x06956,0x078DF,0x00C60,0x01DE9,0x02F72,0x03EFB,
   0x0D68D,0x0C704,0x0F59F,0x0E416,0x090A9,0x08120,0x0B3BB,0x0A232,
   0x05AC5,0x04B4C,0x079D7,0x0685E,0x01CE1,0x00D68,0x03FF3,0x02E7A,
   0x0E70E,0x0F687,0x0C41C,0x0D595,0x0A12A,0x0B0A3,0x08238,0x093B1,
   0x06B46,0x07ACF,0x04854,0x059DD,0x02D62,0x03CEB,0x00E70,0x01FF9,
   0x0F78F,0x0E606,0x0D49D,0x0C514,0x0B1AB,0x0A022,0x092B9,0x08330,
   0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78};

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

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

Algorithm 2: "C" Bit Shift Implementation

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

```
typedef unsigned char ubyte;
typedef unsigned short word;
word get_crc(ubyte *bufptr,word len)
  {
  register unsigned int
    newCRC;
  //Put the current byte in here.
  ubyte
     data;
  int
     bit_count;
  //This seed makes the output of this shift based algorithm match
  //the table based algorithm. The center 16 bits of the 32-bit
  //"newCRC" are used for the CRC. The MSb of the lower byte is used
  //to see what bit was shifted out of the center 16 bit CRC
  //accumulator ("carry flag analog");
  newCRC=0x00F32100;
  while(len--)
    {
    //Get the next byte in the stream.
    data=*bufptr++;

    //Push this byte's bits through a software
    //implementation of a hardware shift & xor.
    for(bit_count=0;bit_count<=7;bit_count++)
      {
      //Shift the CRC accumulator
      newCRC>>=1;
```

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

        //If the low bit of the current CRC accumulator was set
        //before the shift, then we need to XOR the accumulator
        //with the polynomial (center 16 bits of 0x00840800)
        if(newCRC&0x00000080)
          newCRC^=0x00840800;
        //Shift the data byte to put the next bit of the stream
        //into position 0.
        data>>=1;
        }
      }

    //All the data has been done. Do 16 more bits of 0 data.
    for(bit_count=0;bit_count<=15;bit_count++)
      {
      //Shift the CRC accumulator
      newCRC>>=1;

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

Algorithm 2B: “C” Improved Bit Shift Implementation

This is a simplified algorithm that implements the CRC.

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

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

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

Algorithm 3: “PIC Assembly” Bit Shift Implementation

This routine was graciously donated by one of our customers.

```
;====================================================================
; Crystalfontz CFA533 PIC CRC Calculation Example
;
; This example calculates the CRC for the hard coded example provided in the
documentation.
;
; It uses "This is a test. " as input and calculates the proper CRC of 0x93FA.
;====================================================================
#include "p16f877.inc"
;====================================================================
; CRC16 equates and storage
;--------------------------------------------------------------------

accuml      equ         40h         ; BYTE - CRC result register high byte
accumh      equ         41h         ; BYTE - CRC result register high low byte
datareg           equ         42h         ; BYTE - data register for shift
j           equ         43h         ; BYTE - bit counter for CRC 16 routine
Zero        equ         44h         ; BYTE - storage for string memory read
index       equ         45h         ;BYTE - index for string memory read
savchr      equ         46h         ;BYTE - temp storage for CRC routine
;
seedlo      equ         021h        ;initial seed for CRC reg lo byte
seedhi      equ         0F3h        ;initial seed for CRC reg hi byte
;
polyL       equ         008h        ;polynomial low byte
polyH       equ         084h        ;polynomial high byte
;====================================================================
;     CRC Test Program
;--------------------------------------------------------------------
            org         0           ; reset vector = 0000H
;
            clrf        PCLATH      ; ensure upper bits of PC are cleared
            clrf        STATUS      ; ensure page bits are cleared
            goto        main        ; jump to start of program
;
; ISR Vector
;
            org         4           ; start of ISR
            goto        $           ; jump to ISR when coded
;
            org         20          ; start of main program
main
            movlw       seedhi      ; setup intial CRC seed value.
            movwf       accumh      ; This must be done prior to
            movlw       seedlo      ; sending string to CRC routine.
            movwf       accuml      ;
            clrf        index       ; clear string read variables
;
main1
```

```
            movlw       HIGH InputStr       ; point to LCD test string
            movwf       PCLATH        ; latch into PCL
            movfw       index         ; get index
            call        InputStr      ; get character
            movwf       Zero          ; setup for terminator test
            movf        Zero,f        ; see if terminator
            btfsc       STATUS,Z      ; skip if not terminator
            goto        main2         ; else terminator reached, jump out of loop
            call        CRC16         ; calculate new    crc
            call        SENDUART      ; send data to LCD
            incf        index,f       ; bump index
            goto        main1         ; loop
;
main2
            movlw       00h           ; shift accumulator 16 more bits.
            call        CRC16         ; This must be done after sending
            movlw       00h           ; string to CRC routine.
            call        CRC16         ;
;
            comf        accumh,f      ; invert result
            comf        accuml,f      ;
;
            movfw       accuml        ; get CRC low byte
            call        SENDUART      ; send to LCD
            movfw       accumh        ; get CRC hi byte
            call        SENDUART      ; send to LCD
;
stop        goto        stop          ; word result of 0x93FA is in accumh/accuml
;====================================================================
; calculate CRC of input byte
;--------------------------------------------------------------------
CRC16
             movwf      savchr        ; save the input character
            movwf       datareg       ; load data register
            movlw .     8             ; setup number of bits to test
            movwf       j             ; save to incrementor
_loop
            clrc                      ; clear carry for CRC register shift
            rrf         datareg,f     ; perform shift of data into CRC register
            rrf         accumh,f      ;
            rrf         accuml,f      ;
            btfss       STATUS,C      ; skip jump if if carry
            goto        _notset       ; otherwise goto next bit
            movlw       polyL         ; XOR poly mask with CRC register
            xorwf       accuml,F      ;
            movlw       polyH         ;
            xorwf       accumh,F      ;
_notset
            decfsz      j,F           ; decrement bit counter
            goto        _loop         ; loop if not complete
            movfw       savchr        ; restore the input character
            return                    ; return to calling routine
;====================================================================
; USER SUPPLIED Serial port transmit routine
;--------------------------------------------------------------------
SENDUART
            return                    ; put serial xmit routine here
;====================================================================
; test string storage
```

```
;-------------------------------------------------------------------
             org         0100h
;
InputStr
             addwf       PCL,f
             dt          7h,10h,"This is a test. ",0
;
;===================================================================
             end
```

Algorithm 4: “Visual Basic” Table Implementation

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

```
'Written by Crystalfontz America, Inc. 2004 http://www.crystalfontz.com

'Free code, not copyright copy left 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:
'http://www.crystalfontz.com/products/633/633_WinTest.zip

'Full zip of the project is available in our forum:
'https://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.
 Sub Initialize_CRC_Lookup_Table()
   crcLookupTable(0).Lo = &H0
   crcLookupTable(0).Hi = &H0
   . . .
 'For purposes of brevity in this Datasheet, I have removed 251 entries of this
 table, the 'full source is available in our forum:
 'https://www.crystalfontz.com/forum/showthread.php?postid=9921#post9921
   . . .
   crcLookupTable(255).Lo = &H78
   crcLookupTable(255).Hi = &HF
 End Sub
```

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

Private Sub Send_Packet(ByRef packet As PACKET_STRUCT)
  Dim Index As Integer
  'Need to put the whole packet into a linear array
  'since you can't do type overrides. VB, gotta love it.
  Dim linear_array(26) As Byte
  linear_array(0) = packet.command
  linear_array(1) = packet.data_length
  For Index = 0 To packet.data_length - 1
    linear_array(Index + 2) = packet.data(Index)
  Next Index
  packet.crc = Get_Crc(linear_array, packet.data_length + 2)
  'Might as well move the CRC into the linear array too
  linear_array(packet.data_length + 2) = packet.crc.Lo
  linear_array(packet.data_length + 3) = packet.crc.Hi
  'Now a simple loop can dump it out the port.
  For Index = 0 To packet.data_length + 3
    MSComm.Output = Chr(linear_array(Index))
  Next Index
End Sub
```

Algorithm 5: "Java" Table Implementation

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

```
public class CRC16 extends Object
  {
  public static void main(String[] args)

    {
    byte[] data = new byte[2];
    // hw - fw
    data[0] = 0x01;
    data[1] = 0x00;
    System.out.println("hw -fw req");
    System.out.println(Integer.toHexString(compute(data)));

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

    // reboot
    data[0] = 0x05;
    data[1] = 0x00;
    System.out.println("reboot");
    System.out.println(Integer.toHexString(compute(data)));
    // clear lcd
    data[0] = 0x06;
    data[1] = 0x00;
    System.out.println("clear lcd");
    System.out.println(Integer.toHexString(compute(data)));

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

Algorithm 6: "Perl" Table Implementation

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

```
    #!/usr/bin/perl

    use strict;

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

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

    my $type = '80';
    my $length = '01';
    my $data = '05';

    my $packet = chr(hex $type) .chr(hex $length) .chr(hex $data);

    my $valid_crc = '5584' ;

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

    my $crc = 0xFFFF ;

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

    foreach my $char (split //, $packet)
      {
      # newCrc = (newCrc >> 8) ^ crcLookupTable[(newCrc ^ *bufptr++) & 0xff];
      # & is bitwise AND
      # ^ is bitwise XOR
      # >> bitwise shift right
```

```
        $crc = ($crc >> 8) ^ $CRC_LOOKUP[($crc ^ ord($char) ) & 0xFF] ;
        # print out the running crc at each byte
        printf("%x\n", $crc);
        }

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

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

Algorithm 7: For PIC18F8722 or PIC18F2685

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

```
; CRC Algorithm for CrystalFontz CFA635 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 the byte 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                ; XOR poly mask with CRC register
      xorwf        dsplyCRC,F
_notset:
      decfsz       dsplyCRCCount,F     ; decrement bit counter
      bra    cloop                     ; loop if not complete
      return
;====================================================================
; example to clear screen
dsplyFSR1_TEMP     equ    0x83A ;      ; 16-bit save for FSR1 for display
                                       ; message handler
dsplyCRC           equ    0x83C        ; 16-bit CRC (H/L)
dsplyLPCNT         equ    0x83E        ; 8-bit save for display message
                                       ; length - CRC
dsplyCRCData       equ    0x83F        ; 8-bit CRC data for display use
dsplyCRCCount      equ    0x840        ; 8-bit CRC count for display use
```

```
SendCount           equ    0x841        ; 8-bit byte count for sending to
                                        ; display
RXBUF2              equ    0x8C0        ; 32-byte receive buffer for
                                        ; Display
TXBUF2              equ    0x8E0        ; 32-byte transmit buffer for
                                        ; Display
;---------------------------------------------------------------
ClearScreen:
      movlb        RAM8
      movlw        .0
      movwf        SendCount
      movlw        0xF3
      movwf        dsplyCRC     ; seed ho for CRC calculation
      movlw        0x21
      movwf        dsplyCRC+1         ; seen lo for CRC calculation
      call         ClaimFSR1
      movlw        0x06
      movwf        TXBUF2
      LFSR         FSR1,TXBUF2
      movf         SendCount,w
      movwf        TXBUF2+1           ; message data length
      call         BMD1
      goto         SendMsg
;=====================================================================
; send message via interrupt routine. The code is made complex due
; to the limited FSR registers and extended memory space used
;
; example of sending a string to column 0, row 0
;---------------------------------------------------------------
SignOnL1:
      call         ClaimFSR1
      lfsr         FSR1,TXBUF2+4      ; set data string position
      SHOW         C0R0,BusName       ; move string to TXBUF2
      movlw        .2                 ;
      addwf        SendCount          ;
      movff        SendCount,TXBUF2+1
                                      ; insert message data length
      call         BuildMsgDSPLY
      call         SendMsg
      return
;=====================================================================
; BuildMsgDSPLY used to send a string to LCD
;---------------------------------------------------------------
 BuildMsgDSPLY:
      movlw        0xF3
      movwf        dsplyCRC           ; seed hi for CRC calculation
      movlw        0x21
      movwf        dsplyCRC+1         ; seed lo for CRC calculation
      LFSR         FSR1,TXBUF2        ; point at transmit buffer
      movlw        0x1F               ; command to send data to LCD
      movwf        TXBUF2             ; insert command byte from us to
                                      ; CFA635
      BMD1         movlw .2
      ddwf         SendCount,w        ; + overhead
      call         ComputeCRC2        ; compute CRC of transmit message
      movf         dsplyCRC+1,w
      movwf        POSTINC1           ; append CRC byte
      movf         dsplyCRC,w
      movwf        POSTINC1           ; append CRC byte
      return
;=====================================================================
SendMsg:
      call         ReleaseFSR1
      LFSR         FSR0,TXBUF2
      movff        FSR0H,irptFSR0
      movff        FSR0L,irptFSR0+1
                                      ; save interrupt use of FSR0
      movff        SendCount,TXBUSY2
      bsf          PIE2,TX2IE
                                      ; set transmit interrupt enable
                                      ; (bit 4)
      return
;===================================================================
; macro to move string to transmit buffer
SHOW macro    src, stringname
      call         src
      MOVLF        upper stringname, TBLPTRU
      MOVLF        high stringname, TBLPTRH
      MOVLF        low stringname, TBLPTRL
      call         MOVE_STR
      endm
;===================================================================
MOVE_STR:
      tblrd        *+
      movf         TABLAT,w
      bz           ms1b
      movwf        POSTINC1
      incf         SendCount
      goto         MOVE_STR

ms1b:
      return
;===================================================================
```