



INTELLIGENT LCD MODULE SPECIFICATIONS



Datasheet Release 2017-03-30

for

CFA635-TFK-KL

CFA635-TML-KL

CFA635-YYK-KL

Hardware Version: h1v5

Firmware Version: s2v1

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	5
2. Introduction	6
2.1. <i>Difference Between the Two Serial Interfaces.....</i>	6
2.2. <i>Main Features</i>	6
2.3. <i>Module Classification Information</i>	8
2.4. <i>Ordering Information</i>	8
2.5. <i>Display Mounts.....</i>	9
2.6. <i>Cables</i>	10
3. Mechanical Characteristics.....	11
3.1. <i>Physical Characteristics.....</i>	11
3.2. <i>Jumper Locations and Functions (All Interfaces).....</i>	11
3.3. <i>Valid Jumper Settings</i>	12
4. Optical Characteristics	13
4.1. <i>CFA635 Series.....</i>	13
4.2. <i>LED Backlight Information.....</i>	13
5. Electrical Specifications.....	14
5.1. <i>System Block Diagram.....</i>	14
6. Supply Voltages and Current.....	15
6.1. <i>Absolute Maximum Ratings</i>	15
6.2. <i>DC Characteristics</i>	15
6.3. <i>Typical Current Consumption</i>	16
6.4. <i>GPIO Current Limits.....</i>	17
6.5. <i>Fan Criteria – Using Optional FBSCAB.....</i>	17
6.6. <i>UART Interface Characteristics</i>	17
7. Connection Information.....	18
7.1. <i>H1 Connector Pin Assignments.....</i>	18
8. ATX Power Supply Control	19
8.1. <i>How to Connect the Optional FBSCAB.....</i>	20
9. Host Communications	21
9.1. <i>Packet Structure.....</i>	21
9.2. <i>About Handshaking.....</i>	22
9.3. <i>Report Codes</i>	22
9.4. <i>Command Codes</i>	24
10. Character Generator ROM (CGROM)	46
11. LCD Module Reliability and Longevity.....	47
11.1. <i>Module Longevity (EOL / Replacement Policy).....</i>	47
12. Care and Handling Precautions.....	48

12.1. ESD (Electrostatic Discharge).....	48
12.2. Design and Mounting.....	48
12.3. Avoid Shock, Impact, Torque, or Tension.....	48
12.4. If LCD Panel Breaks.....	48
12.5. Cleaning.....	48
12.6. Operation.....	48
12.7. Storage and Recycling.....	49
13. Mechanical Drawings.....	50
Module Outline Drawing Front, Back, and Side Views.....	50
Optional Mounting Bracket Drawing.....	51
Keypad Detail Drawing.....	52
Panel Mounting Application Cutout Drawing.....	53
14. Appendix A: Demonstration Software and Sample Code.....	54
Sample Code.....	54
Algorithms to Calculate the CRC.....	54

Table of Figures

Figure 1. CFA635 Drive Bay Bracket.....	9
Figure 2. CFA635 SLED	9
Figure 3. XES635-TFK-KU in Steel Case	9
Figure 4: CFA635 Jumper Locations and Functions	11
Figure 5. System Block Diagram.....	14
Figure 6. Pin Assignments on the CFA635-xxx-KL H1 Connector	18
Figure 7. CFA635 Connected to Optional FBSCAB Using WR-EXT-Y37 Cable.....	20
Figure 8: Character Generator ROM (CGROM)	46

1. General Information

Datasheet Revision History

Datasheet Version: **2017-03-30**
Hardware Version: **v1.5**
Firmware Version: **s2v1**

This datasheet has been updated to reflect hardware version v1.5, firmware s2v1 for the CFA635-TFK-KL, CFA635-TML-KL, and the CFA635-YYK-KL LCD modules.

For information about firmware and hardware revisions, see the Part Change Notifications (PCNs) under “News” in our website’s navigation bar. To see the most recent PCN for the CFA635 family at the time of this datasheet release, see [PCN #10402](#).

Previous datasheet Version: **2014-03-18**

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](#) tab on the product’s webpage.

Variations

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

Volatility

This display module has volatile memory.

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.

Copyright © 2017 by Crystalfontz America, Inc., 12412 East Saltese Avenue, Spokane Valley, WA 99216 U.S.A.

2. Introduction

The CFA635 family of modules has four interface choices:

- CFA635-xxx-KL (logic-level serial / UART)
- CFA635-xxx-KS (RS232)
- CFA635-xxx-KU (USB)
- XES635BK-xxx-KU (enclosed USB)

This datasheet has information for these interface modules:

- CFA635-TFK-KL **CFA-635** (USB, dark letters on a light background; this display can be read in normal office lighting, in dark areas, and in bright sunlight.)
- CFA635-TML-KL **CFA-635** (USB, light letters on a blue background; this display can be read in normal office lighting and in dark areas.)
- CFA635-YYK-KL **CFA-635** (USB, dark letters on a yellow background; this display can be read in normal office lighting, in dark areas, and in bright sunlight.)

When information in this datasheet applies to all three colors the term “CFA635-xxx-KL” or “CFA635” is used.

2.1. Difference Between the Two Serial Interfaces

“Logic Level, Inverted” Serial (CFA635-xxx-KL)

The CFA635-xxx-KL exposes the UART Tx & Rx (logic level, inverted, 0v to 3.3v nominal) signals on pin 1 and pin 2 of the CFA635's expansion connector H1. If your embedded processor is in close physical proximity to the CFA635, you can cable its UART Rx and Tx pins directly to the CFA635-xxx-KL's Tx and Rx pins. No RS232 level translators are required on either end.

“Full Swing” RS232 Serial (CFA635-xxx-KS)

The CFA635-xxx-KS is a CFA635-xxx-KL with a CFA-RS232 level translator board attached. The CFA635-xxx-KS is the correct choice if your embedded controller or host system has a “real” RS232 serial port (-5v to +5v “full swing” serial interface) available.

NOTE: The USB connector on the CFA635-xxx-KL and CFA635-xxx-KS is not supported in the firmware and cannot be used.

2.2. Main Features

- Large, easy-to-read, 20 character x 4-line LCD in a compact overall size.
- Fits nicely in a 1U rack mount case (37 mm overall height).
- May be installed in a standard half-height 5 1/4 drive bay by using our optional mounting bracket or our optional SLED bracket. The SLED holds the CFA-635 display module, an optional FBSCAB, and has mounting points for a standard 3.5-inch hard disk drive.
- The LCD has a wide viewing angle, with a 12 o'clock preferred viewing direction.
- Serial interface “logic level, inverted”, 0v to +5v nominal.
- Six-button, LED backlit, translucent silicone keypad with screened legend. Fully decoded keypad: any key combination is valid and unique.
- LCD is edge-lit with 8 long-life, high performance, LEDs (4 per side).
- Adjustable contrast. The default contrast value for the module will be acceptable for most applications. If necessary, you can adjust the contrast using command [13 \(0x0D\): Set LCD Contrast](#).
- The front of the display has four bicolor (red + green), LED status lights. The LEDs' brightness can be set by the host software which allows smoothly adjusting the LEDs to produce other colors (for example, yellow and orange).
- Robust, packet-based protocol with 16-bit CRC ensures error-free communications.



- Optional ATX Power Supply control functionality allows the CFA635's buttons to replace the Power and Reset switches on your system, simplifying front panel design. Optional Crystalfontz WR-PWR-Y25 or WR-PWR-Y38 cables may be used to simplify connection to the host's motherboard.
- Nonvolatile memory capability (EEPROM):
 - Customize the "power-on" display settings (backlight brightness, boot screen, LED settings).
 - 16-byte "scratch" register for storing IP address, netmask, system serial number.
- Hardware watchdog functionality can reset host on host software failure.
- The CFA635-xxx-KL may be used with our optional FBSCAB (System Cooling Accessory Board). The combination of the CFA635-xxx-KL with the optional FBSCAB (CFA635+FBSCAB) allows:
 - Four fan connections with RPM monitoring and variable PWM (Pulse Width Modulation), fan power control.
 - Fail-safe fan power settings allows host to safely control the fans based on temperature.
 - Add up to 16 Crystalfontz WR-DOW-Y17 cables with DOW (Dallas 1-Wire) DS18B20 temperature sensors. Monitor temperatures up to 0.5°C absolute accuracy.
 - For ATX Power Supply control functionality when connected to a FBSCAB, buy the WR-PWR-Y25 or the WR-PWR-Y38 ATX power cable.
 - For more information, see [ATX Power Supply Control](#) and the [FBSCAB Datasheet](#) on our website.
- Crystalfontz America, Inc. is ISO 9001:2008 certified.
- A Declaration for Conformity, RoHS, and REACH:SVHC are available under the Datasheets & Files tab on display web pages.

2.3. Module Classification Information

CFA	635	-	x	x	x	-	K	L	
1	2		3	4	5		6	7	8

1	Brand	Crystalfontz America, Incorporated
2	Model Identifier	635
3	Backlight Type & Color	x = T – LED, white Y – LED, yellow-green
4	Fluid Type, Image (positive or negative), & LCD Glass Color	x = F – FSTN, positive, neutral M – STN, negative blue Y – STN, positive, yellow-green
5	Polarizer Film Type, Temperature Range, & View Angle (O’Clock)	x = K – Transflective, WT, 12:00 L – Transmissive, WT, 12:00
6	Special Code	K – Manufacturer’s code
7	Interface	L – Serial “logic level, inverted” 0v to +5v nominal
8	Customize Configuration Codes¹	x = 1 or more characters

¹When you order a CFA635 through our website, you may be offered a choice of configurations (including accessories), to add to your order through our “Customize and Add to Cart” feature.

2.4. Ordering Information

Part Number	Fluid	LCD Glass Color	Image	Polarizer Film	Backlight Color/Type
CFA635-TFK-KL	FSTN	neutral	positive	transflective	Backlight: white Keypad: white
CFA635-TML-KL	STN	blue	negative	transmissive	Backlight: white Keypad: blue
CFA635-YYK-KL	STN	yellow-green	positive	transflective	Backlight: yellow-green Keypad: yellow-green

Additional modules in the CFA635 family are:

- A serial interface using the CFA-RS232 level translator board. Part numbers end in “-KS”. Suitable for embedded controller or host system that has a “real” RS232 serial port (-5v to +5v “full swing” serial interface).
- A USB interface module. Part numbers end in “-KU”.
- An external enclosure with a captive USB “A” cable connection. Please see <https://www.crystalfontz.com/family/XES635BK?family=XES635BK>.

2.5. Display Mounts

On the webpage for the [CFA635 Series](#), after you click the “*Customize and Add to Cart*” button, you will see a list of options for different cables, connectors, drive bay bracket, and SLED. The [XES635BK Series](#) comes enclosed in a steel case.



Figure 1. CFA635 Drive Bay Bracket






Figure 2. CFA635 SLED



Figure 3. XES635-TFK-KU in Steel Case

2.6. Cables

Below is a list of some of the cables we offer to make it easy to integrate the CFA635 into your system. Please note that cable lengths are approximate. Common configurations are described in [Connection Information](#).

Crystalfontz Cable	Image	Description All Cables Are RoHS Compliant
WR-EXT-Y37 ~18 inches		For use with FBSCAB: Used to connect the CFA635 to the FBSCAB.
WR-FAN-X01 ~16 inches		For use with FBSCAB: Fan extension cable for standard 3-pin fans.
WR-DOW-Y17 ~12 inches + ~12 inches between connectors		For use with FBSCAB: You may connect ("daisy chain") up to 16 of these DOW (Dallas 1-Wire) DS18B20 temperature sensor cables to the FBSCAB.
Please note that J8 of the FBSCAB is unused. ATX functions are supported by H1 on the CFA635.		

3. Mechanical Characteristics

3.1. Physical Characteristics

Item	Specification (mm)	Specification (inch, reference)
Overall Width and Height	142.00 (W) x 37.0 (H)	5.59 (W) x 1.46 (H)
Viewing Area / Bezel Opening	82.90 (W) x 27.50 (H)	3.26 (W) x 1.08 (H)
Active Area	77.95 (W) x 22.35 (H)	3.07 (W) x 0.88 (H)
5x7 Standard Character Size	3.20 (W) x 4.85 (H)	0.126 (W) x 0.190 (H)
Pixel Size	0.60 (W) x 0.65 (H)	0.024 (W) x 0.026 (H)
Pixel Pitch	0.65 (W) x 0.70 (H)	0.026 (W) x 0.028 (H)
Depth with Keypad, with Connectors	20.80 nominal 21.10 maximum	0.82 nominal 0.83 maximum
Keystroke Travel (approximate)	~.4	~0.1
Weight (typical)	62.1 grams	2.2 ounces

3.2. Jumper Locations and Functions (All Interfaces)

The CFA635-xxx-KL has three jumpers. To open a jumper, remove the solder ball **AND** de-populate the corresponding resistor. To close a jumper, melt a ball of solder across the gap **OR** populate the corresponding resistor location with an 0603 0-ohm resistor.

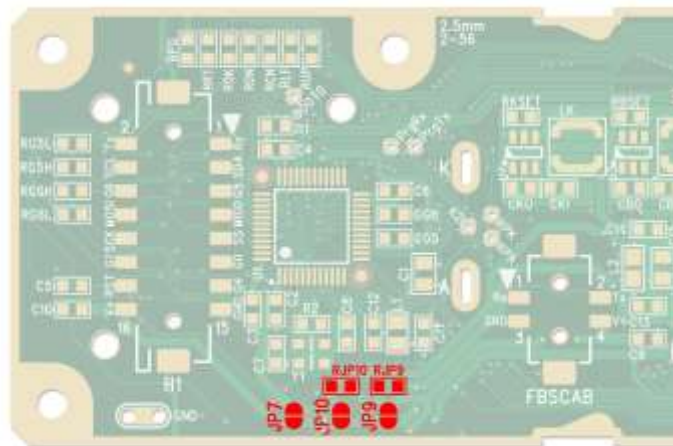

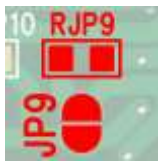



Figure 4: CFA635 Jumper Locations and Functions

Jumper Locations and Functions (All Interfaces)

JP7		open (default setting)	Output of 3.3v switcher is not connected to H1 or the FBSCAB connector.
		closed	Output of 3.3v switcher is connected to the power pin of H1 and the power pin of the FBSCAB connector. IMPORTANT: JP9/RJP9 must be open if JP7 is closed.
JP9/RJP9		open	H1 and the FBSCAB connector are not connected to the input of the 3.3v switcher.
		closed (default setting, RJP9 is loaded)	The power pin of H1 and the power pin of the FBSCAB connector are connected to the input of the 3.3v switcher. IMPORTANT: JP7 must be open if JP9/RJP9 is closed.
JP10/RJP10		open (default for KL & KS)	Power from the USB is not connected to the input of the 3.3v switcher.
		closed	Power from the USB is connected to the input of the 3.3v switcher.

3.3. Valid Jumper Settings

Function: USB interface, module powered by USB (not useful for -KL or -KS)

- JP7: Open
- JP9/RJP9: Closed
- JP10/RJP10: Closed

NOTE: Module is powered by USB, so the module will power up when the host system provides power to the USB connector. USB power is fed out the power pin of H1 and the power pin of the FBSCAB connector.

Function: USB interface, module powered by H1 (not useful for -KL or -KS)

- JP7: Open
- JP9/RJP9: Closed
- JP10/RJP10: Open

NOTE: Module is powered from the host supply by the power pin of H1. As soon as power is applied by the host to the power pin of H1 the module will boot. The host power fed into the power pin of H1 is fed out the power pin of the FBSCAB connector. The USB power is isolated from the power pin of H1, so that the host power fed into the power pin of H1 has no risk of "back powering" the host's USB port. This configuration will allow a message to be shown immediately when host power is applied ("System booting" or similar) without the delay of the operating system enabling the USB port.

Function: Serial interface, module powered by H1 (default setting for -KL & -KS)

- JP7: Open
- JP9/RJP9: Closed
- JP10/RJP10: Open

NOTE: Module is powered from the host supply by the power pin of H1. As soon as power is applied by the host to the power pin of H1 the module will boot. The host power fed into the power pin of H1 is fed out the power pin of the FBSCAB connector.

4. Optical Characteristics

4.1. CFA635 Series

Item	Symbol	Condition	Min	Typ	Max	Direction
Viewing Angle (12 o'clock is the preferred direction for this module)	θ	$CR \geq 2$	40°	—	—	above, 12 o'clock
	θ	$CR \geq 2$	20°	—	—	below, 6 o'clock
	θ	$CR \geq 2$	30°	—	—	right, 3 o'clock
	θ	$CR \geq 2$	30°	—	—	left, 9 o'clock
Contrast Ratio	CR	—	—	10	15	—
Response Time	T rise	—	—	80	160	ms
	T fall	—	—	100	200	ms

4.2. LED Backlight Information

The backlights used in the CFA635 are designed for very long life, but their lifetime is finite. To conserve the LED lifetime and reduce power consumption you can dim or turn off the backlights during periods of inactivity.

5. Electrical Specifications

5.1. System Block Diagram

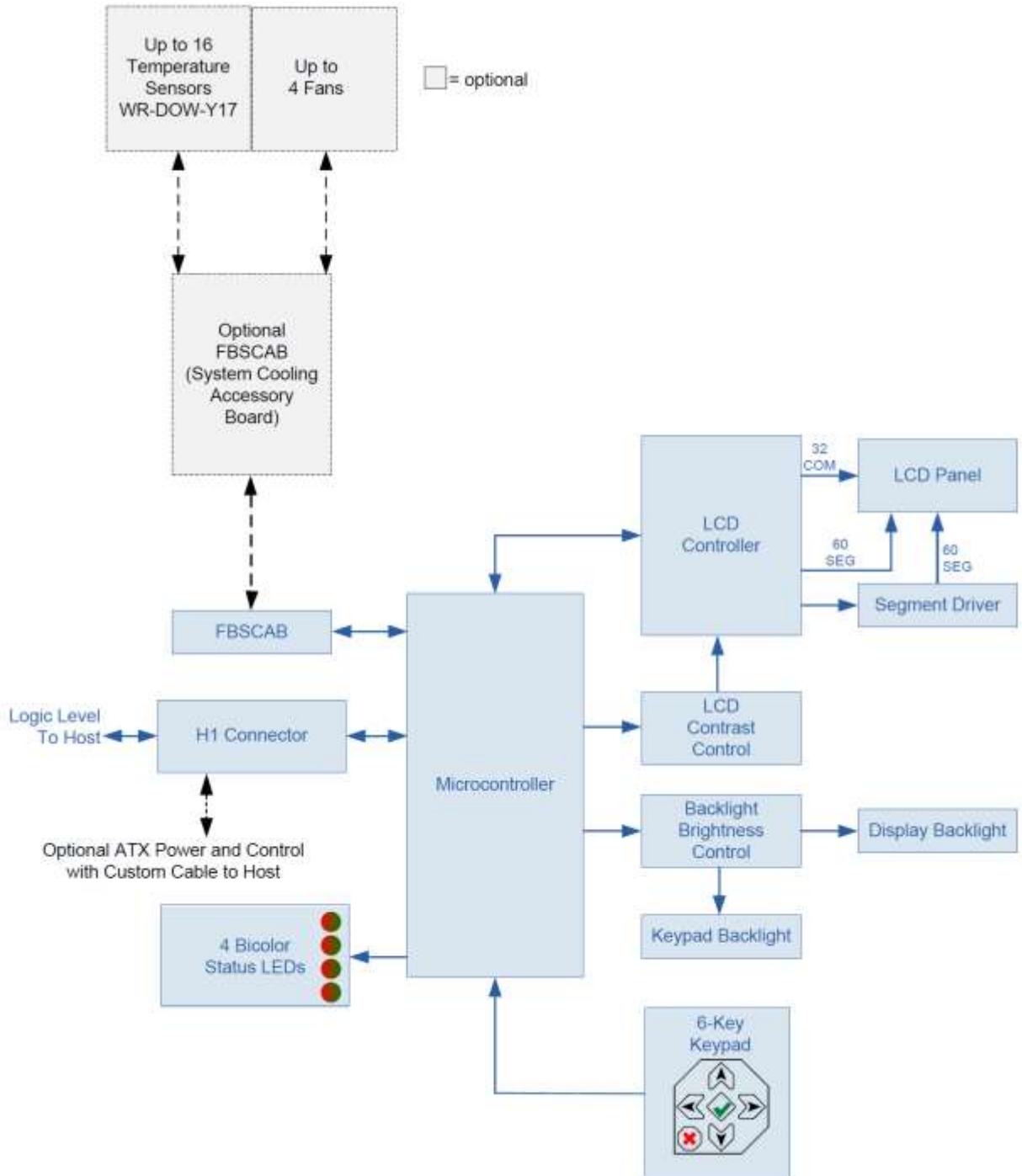


Figure 5. System Block Diagram

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%
Supply Voltage for Logic	V_{DD}	-0.3v	+5.5v

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. Changes in temperature can result in changes in contrast.

6.2. DC Characteristics

Specifications	Symbol	Minimum	Typical	Maximum
Supply Voltage	$V_{DD} - V_O$	+2.9v	+3.3v or +5.0v	+5.5v
GPIO Input High Voltage	V_{IH}	+1.85v		V_{DD}
GPIO Input Low Voltage	V_{IL}	V_{SS} (0v)		+1.3v
GPIO Output High Voltage (@ 8mA)	V_{OH}	+2.9v		
GPIO Input Low Voltage (@ 8mA)	V_{OL}			+0.4v

6.3. Typical Current Consumption

Variables that affect current consumption include the choice of color, interface type, brightness of backlights, brightness of the four status lights, power supply voltage, and if the optional [FBSCAB](#) is attached to the module.

CFA635-TFK-KL **CFA-635** (dark characters on a light background)

Items Enabled			Typical Current Consumption	
Logic	LCD and Keypad Backlights at 100%	All Status LEDs 4 Red + 4 Green at 100%	VDD=+3.3v	VDD=+5v
X	-	-	25 mA	18 mA
X	X	-	150 mA	100 mA
X	-	X	150 mA	100 mA
X	X	X	280 mA	180 mA

CFA635-TML-KL **CFA-635** (light characters on a deep blue background)

Items Enabled			Typical Current Consumption	
Logic	LCD and Keypad Backlights at 100%	All Status LEDs 4 Red + 4 Green at 100%	VDD=+3.3v	VDD=+5v
X	-	-	25 mA	18 mA
X	X	-	150 mA	100 mA
X	-	X	150 mA	100 mA
X	X	X	280 mA	180 mA

CFA635-YYK-KL **CFA-635** (dark characters on a yellow-green background)

Items Enabled			Typical Current Consumption	
Logic	LCD and Keypad Backlights at 100%	All Status LEDs 4 Red + 4 Green at 100%	VDD=+3.3v	VDD=+5v
X	-	-	25 mA	18 mA
X	X	-	180 mA	120 mA
X	-	X	150 mA	100 mA
X	X	X	310 mA	200 mA

6.4. GPIO Current Limits

Typical GPIO Current Limits	Specification
Sink	8 mA
Source	8 mA

6.5. Fan Criteria – Using Optional FBSCAB

Fan Criteria	Specification
Fan Tachometer Speed Range (assuming two PPR ¹)	600 RPM to 3,000,000 RPM
Fan Power Control PWM ² Frequency	18 Hz nominal
¹ PPR is Pulses Per Revolution, can also written as p/r. ² PWM is Pulse Width Modulation.	

6.6. UART Interface Characteristics

The Rx and Tx pins of the H1 connector conforms to the same voltage threshold levels as other GPIO pins, and are listed in table [DC Characteristics](#).

The CFA-635 circuitry is industry standard CMOS logic and susceptible to ESD damage. Please use industry standard antistatic precautions as you would for any other static sensitive devices such as expansion cards, motherboards, or integrated circuits. Make sure to ground your body, work surfaces, and equipment.

7. Connection Information

7.1. H1 Connector Pin Assignments

The H1 male 16-pin 2 mm connector on the CFA635-xxx-KL provides power, the logic-level serial communication and GPIO connections.

The following parts may be used to make a mating cable for H1:

- 16-position housing: Hirose DF11-16DS-2C / [Digi-Key H2025-ND](#).
- Terminal: Hirose DF11-2428SC / [Digi-Key H1504-ND](#).
- Pre-terminated interconnect wire: Hirose / [Digi-Key H3BBT-10112-B4-ND](#) (typical).

H1: Power Connections

- Connect your positive supply voltage to H1, Pin 16, labeled “V+”.
- Connect the ground to H1, Pin 15, labeled “GND”.

H1: Logic-Level / UART Serial Connections

- Connect your Tx (output) signal to H1, Pin 1 (input to the CFA635), labeled “Rx”.
- Connect your Rx (input) signal to H1, Pin 2 (output from the CFA635), labeled “Tx”.

H1: GPIO Connections

The CFA635-xxx-KL has five GPIOs available on the H1 connector.

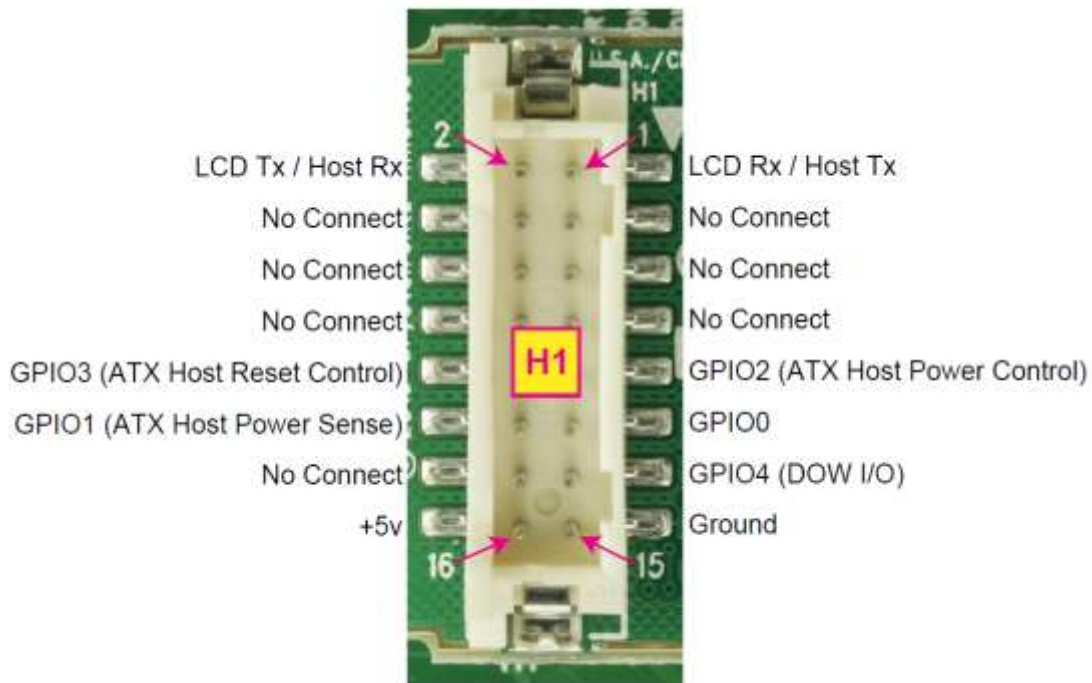


Figure 6. Pin Assignments on the CFA635-xxx-KL H1 Connector

Please see the commands [34 \(0x22\) Set or Set and Configure GPIO Pins](#) and [35 \(0x23\): Read GPIO and GPIO Pin Levels and Configuration State](#), for details on how to control the GPIOs.

8. ATX Power Supply Control

ATX Power Supply control functionality allows the buttons on the CFA635-xxx-KL to replace the power and reset buttons on your system, simplifying your front panel design. This ATX Power Supply control functionality can be accomplished with the optional [WR-PWR-Y25](#) or [WR-PWR-Y38](#) cables, or you may make your own cables.

NOTE: The GPIO pins used for ATX control must not be configured as user GPIO. The GPIO pins must be configured to their default drive mode in order for the ATX functions to work correctly. These settings are factory default, but may have been changed by the user. Please see command [34 \(0x22\) Set or Set and Configure GPIO Pins](#).

In ATX configuration, the CFA635-xxx-KL is powered from the PC's V_{SB} signal (the “stand-by” or “always-on” +5v of your ATX Power Supply), which is connected to pin 15 (V_{SB}) of H1. Ground is supplied to pin 16 of H1.

GPIO[1] ATX Host Power Sense

Since the CFA635-xxx-KL must act differently depending on whether the host's power supply is on or off, you must connect the host's “switched +5v” to GPIO[1]. This GPIO line functions as POWER SENSE. The POWER SENSE pin is configured as an input with a pull-down, 25k Ω nominal.

GPIO[2] ATX Host Power Control

The motherboard's power switch input is connected to GPIO[2]. This GPIO line functions as POWER CONTROL. The POWER CONTROL pin is configured as a high impedance input until the LCD module instructs the host to turn 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 Power Switch Functionality](#).

GPIO[3] ATX Host Reset Control

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

ATX Connections	Pins on Connector H1
V_{SB} +5v	Pin 16
Ground	Pin 15
GPIO[1] ATX Host Power Sense	Pin 12
GPIO[2] ATX Host Power Control	Pin 9
GPIO[3] ATX Host Reset Control	Pin 10

8.1. How to Connect the Optional FBSCAB

The optional FBSCAB is designed to connect to the CFA635's FBSCAB connector.

The photo below shows the CFA635 connected to the optional FBSCAB using the [WR-EXT-Y37](#) cable:

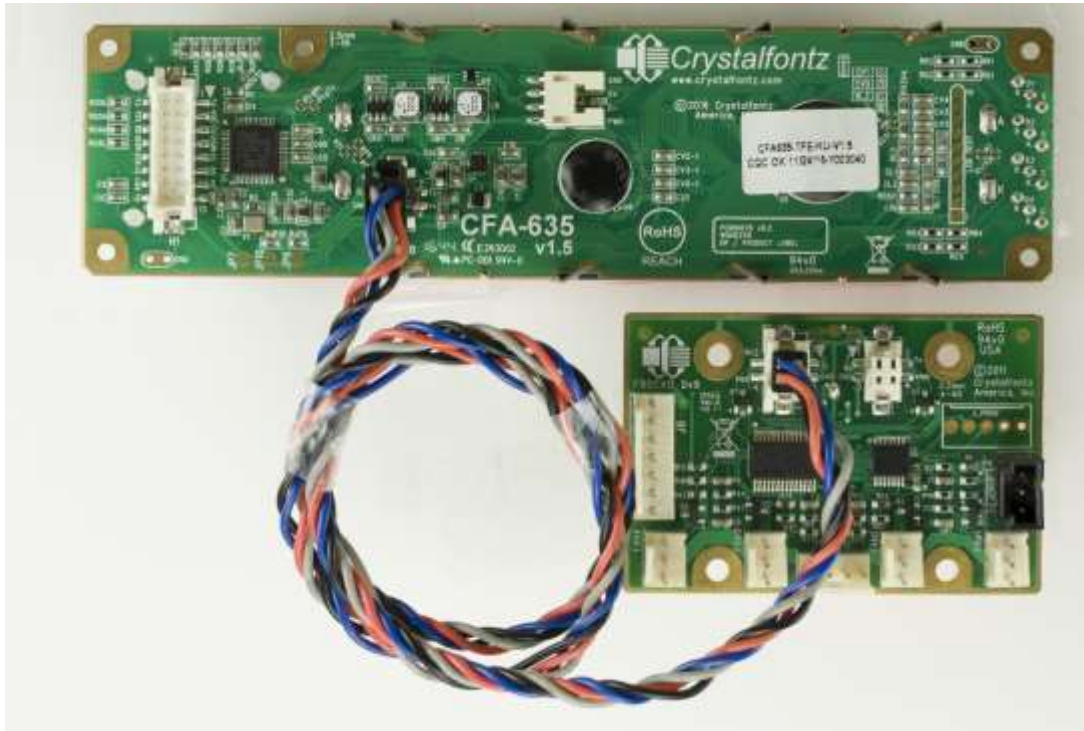


Figure 7. CFA635 Connected to Optional FBSCAB Using WR-EXT-Y37 Cable

9. Host Communications

The CFA635-xxx-KL communicates with its host using a “logic-level, or UART” interface. The port settings are 115200 baud, 8 data bits, no parity, 1 stop bit by default. The speed can be set to 19200 baud under software control, see command [33 \(0x21\): Set Baud Rate](#).

9.1. Packet Structure

All communication between the CFA635 and the host takes place in the form of a simple and robust CRC checked packet. The packet format allows for very reliable communications between the CFA635 and the host without the traditional problems that occur in a stream-based serial communication (such as having to send data in inefficient ASCII format, to “escape” certain “control characters”, or losing sync if a character is corrupted, missing, or inserted).

Reconciling packets is recommended rather than using delays when communicating with the module. To reconcile your packets, please ensure that you have received the acknowledgement packet before sending any additional packets.

All packets have the following structure:

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

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

```
TTcc cccc
```

```
|||| ||| |--Command, response, error or report code 0-63
```

```
||-----Type:
```

```
00 = normal command from host to CFA635
```

```
01 = normal response from CFA635 to host
```

```
10 = normal report from CFA635 to host (not in  
direct response to a command from the host)
```

```
11 = error response from CFA635 to host (a packet  
with valid structure but illegal content  
was received by the CFA635)
```

data_length specifies the number of bytes that will follow in the data field. The valid range of **data_length** is 0 to 22.

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

CRC is a standard 16-bit CRC of all the bytes in the packet except the CRC itself. The CRC is sent LSB first. At the port, the CRC immediately follows the last used element of data []. [See Appendix A: Demonstration Software and Sample Code](#) for details.

The following C definition may be useful for understanding the packet structure.

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

Crystalfontz supplies a demonstration and test program, [cfTest](#), that can be used to experiment with and test the CFA635's operation. We also offer [635WinTest](#), which is a simpler, open-source program. Included in the 635WinTest source is a CRC algorithm and an algorithm that detects and reconciles packets. The algorithm will automatically re-synchronize to the next valid packet in the event of any communications errors. Please follow the algorithm in the sample code closely in order to realize the benefits of using the packet communications.

9.2. About Handshaking

The nature of CFA635's packets makes it unnecessary to implement traditional hardware or software handshaking.

The host should wait for a corresponding acknowledge packet from the CFA635 before sending the next command packet. The CFA635 will respond to all packets within 250 mS. The host software should stop waiting and retry the packet if the CFA635 fails to respond within 250 mS. The host software should report an error if a packet is not acknowledged after several retries. This situation indicates a hardware problem – for example, a disconnected cable.

Please note that some operating systems may introduce delays between when the data arrives at the physical port from the CFA635 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 CFA635 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 CFA635. 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.

9.3. Report Codes

The CFA635 can be configured to report three items. The CFA635 sends reports automatically when the data becomes available. Reports are not sent in response to a particular packet received from the host. The three report types are (1) 0x80: Key Activity, (2) 0x81: Fan Speed Report (FBSCAB Required), and (3) 0x82: Temperature Sensor Report (FBSCAB Required). The three report types are below.

0x80: Key Activity

If a key is pressed or released, the CFA635 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
```

0x81: Fan Speed Report (FBSCAB Required)

If any of up to three fans connected to CFA635-xxx-KL+[FBSCAB](#) is configured to report its speed information to the host, the CFA635-xxx-KL will send Fan Speed Reports for each selected fan every 1/2 second, please see command [16 \(0x10\): Set Up Fan Reporting](#).

```
type = 0x81
data_length = 4
data[0] is the index of the fan being reported:
    0 = FAN 1
    1 = FAN 2
    2 = FAN 3
    3 = FAN 4
data[1] is number_of_fan_tach_cycles
data[2] is the MSB of Fan_Timer_Ticks
data[3] is the LSB of Fan_Timer_Ticks
```

If the FBSCAB module is disconnected, data[1], data[2], and data[3] will be returned with values of 0.

The following C function will decode the fan speed from a Fan Speed Report packet into RPM:

```
int OnReceivedFanReport(COMMAND_PACKET *packet, char * output)
{
    int
    return_value;
    return_value=0;

    int
    number_of_fan_tach_cycles;
    number_of_fan_tach_cycles=packet->data[1];

    if(number_of_fan_tach_cycles<3)
        sprintf(output, " STOP");
    else if(number_of_fan_tach_cycles<4)
        sprintf(output, " SLOW");
    else if(0xFF==number_of_fan_tach_cycles)
        sprintf(output, " ----");
    else
    {
        //Specific to each fan, most commonly 2
        int
        pulses_per_revolution;
        pulses_per_revolution=2;

        int
        Fan_Timer_Ticks;
        Fan_Timer_Ticks=*(unsigned short *)(&(packet->data[2]));

        return_value=((27692308L/pulses_per_revolution)*
            (unsigned long)(number_of_fan_tach_cycles-3))/
            (Fan_Timer_Ticks);
        sprintf(output, "%5d", return_value);
    }
    return(return_value);
}
```

0x82: Temperature Sensor Report (FBSCAB Required)

If any of the up to 16 temperature sensors is configured to report to the host, the CFA635+[FBSCAB](#) will send Temperature Sensor Reports for each selected sensor every second, please 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
```

If a DOW sensor error occurs, data[3] will be returned with a value of 0.

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 CFA635
if(packet->data[3]==0)
    strcpy(output, "BAD CRC");
else
    {
    double
    degc;
    degc=(*(short *)&(packet->data[1]))/16.0;

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

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

9.4. Command Codes

Below is a list of valid commands for the CFA635. 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 CFA635 will return the Ping Command to the host.

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


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

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

1 (0x01): Get Hardware & Firmware Version

The CFA635 will return the hardware and firmware version information to the host.

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

The return packet will be:

```
type: 0x40 | 0x01 = 0x41 = 6510
data_length = 16
data[] = "CFA635:h1.5,s2v1"

h1.5 is the hardware revision
s2v1 is the firmware version
```

2 (0x02): Write User Flash Area

The CFA635 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
valid data_length is 16
data[] = 16 bytes of arbitrary user data to be stored in the CFA635's non-volatile
memory
```

The return packet will be:

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

3 (0x03): Read User Flash Area

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

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

The return packet will be:

```
type: 0x40 | 0x03 = 0x43 = 6710
data_length = 16
data[] = 16 bytes user data recalled from the CFA635's non-volatile memory
```

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

The CFA635 loads its power-up configuration from nonvolatile memory when power is applied. The CFA635 is configured at the factory to display a “welcome” screen when power is applied. This command can be used to customize the “welcome” screen, as well as the following items:

- Characters shown on LCD, which are affected by:
 - Command 6 (0x06): Clear LCD Screen.
 - Command 31 (0x1F): Send Data to LCD.
- Special character font definitions (Command 9 (0x09): Set LCD Special Character Data).
- Cursor position (Command 11 (0x0B): Set LCD Cursor Position).
- Cursor style (Command 12 (0x0C): Set LCD Cursor Style).
- Contrast setting (Command 13 (0x0D): Set LCD Contrast).
- Backlight setting (Command 14 (0x0E): Set LCD & Keypad Backlight).
- Fan power settings (Command 17 (0x11): Set Fan Power).
- Key press and release masks (Command 23 (0x17): Configure Key Reporting).
- Fan glitch delay settings (Command 26 (0x1A): Set Fan Tachometer Glitch Filter).
- ATX function enable and pulse length settings (command 28 (0x1C): Set ATX Power Switch).
- Baud rate (Command 33 (0x21): Set Baud Rate).
- GPIO settings (Command 34 (0x22): Set or Set and Configure GPIO Pins).

You cannot store the fan or temperature reporting (although the live display of fans or temperatures can be saved). You cannot store the fan fail-safe or host watchdog. The host software should enable these items once the system is initialized and it is ready to receive the data.

The command packet is:

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

The return packet will be:

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

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.

5 (0x05): Reboot CFA635, Reset Host, or Power Off Host (ATX Required)

This command instructs the CFA635 with ATX to simulate a power-on restart of itself, reset the host, or turn the host's power off. The ability to reset the host may be useful to allow certain host operating system configuration changes to complete. The ability to turn the host's power off under software control may be useful in systems that do not have Advanced Configuration and Power Interface (ACPI), compatible BIOS. ACPI is an industry specification for the efficient handling of power consumption in desktop and mobile computers.

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

Rebooting the CFA635 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 CFA635, send the following packet:

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

The reboot command may take up to 3 seconds to return its acknowledge packet.

At boot-up, there is a 520-millisecond delay between power-on and turning on the fans. By default, all fans are set to “on” at 100%. If you are not using a fan, set power to 0% (command [17 \(0x11\): Set Fan Power](#)), and saving this setting as the default boot state (command [4 \(0x04\): Store Current State as Boot State](#)).

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  
valid data_length is 3  
data[0] = 12  
data[1] = 28  
data[2] = 97
```

The CFA635 will return the acknowledge packet immediately; then reset the host. After resetting the host (~1.5 seconds), the module will reboot itself. The module will not respond to new command packets for up to 3 seconds (~4.5 seconds overall) after its reboot. Part of this delay is the intentional staggered sequencing of turning on power to the fans. If you are not using fans, you can speed the boot process by setting the fan power to 0 (command [17 \(0x11\): Set Fan Power](#)), and saving this as the default boot state (command [4 \(0x04\): Store Current State as Boot State](#)). Normally, the host will be recovering from its own reset, so the boot delay of the module will not be of consequence.

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  
valid data_length is 3  
data[0] = 3  
data[1] = 11  
data[2] = 95
```

The CFA635 will return the acknowledge packet immediately, then power cycle the host. The power cycle length is dependent on the length of the power pulse (command [28 \(0x1C\): Set ATX Power Switch Functionality](#)). After power cycling the host, the module will reboot itself. The module will not respond to new command packets for up to 3 seconds after its reboot. Part of this delay is the intentional staggered sequencing of turning on power to the fans. If you are not using fans, you can speed the boot process by setting the fan power to 0 (command [17 \(0x11\): Set Fan Power](#)), and saving this as the default boot state (command [4 \(0x04\): Store Current State as Boot State](#)). Normally the host will be off or recovering from its own power cycle, so the boot delay of the module will not be of consequence.

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 and moves the cursor to the left-most column of the top line.

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

The return packet will be:

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

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

7 (0x07): Deprecated (See command 31 (0x1F): Send Data to LCD)

8 (0x08): Deprecated (See command 31 (0x1F): Send Data to LCD)

9 (0x09): Set LCD Special Character Data

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

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

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

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

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

The return packet will be:

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

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

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

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

NOTE: Firmware version prior to v1.9 did not return the address code.

```
type: 0x0A = 1010
valid data_length is 1
data[0] = address code of desired data
data[0] is the address code native to the LCD controller:

    0x40 (\064) to 0x7F (\127) for CGRAM
    0x80 (\128) to 0x93 (\143) for DDRAM, line 1
    0xC0 (\192) to 0xD3 (\207) for DDRAM, line
```

The return packet will be:

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

data[0] of the return packet will be the address code.

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

11 (0x0B): Set LCD Cursor Position

This command allows the cursor to be placed at the desired location on the CFA635's LCD screen. If you want the cursor to be visible, you may also need to send a command [12 \(0x0C\): Set LCD Cursor Style](#).

```
type: 0x0B = 1110
valid data_length is 2
data[0] = column (0-19 valid)
data[1] = row (0-3 valid)
```

The return packet will be:

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

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

12 (0x0C): Set LCD Cursor Style

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

```
type: 0x0C = 1210
valid data_length is 1
data[0] = cursor style (0-4 valid)
    0 = no cursor.
    1 = blinking block cursor.
    2 = static underscore cursor
```

3 = blinking underscore cursor
(Note for 3: (Behavior is different from previous CFA635 versions
firmware s/u1.6 and earlier.)

The return packet will be:

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

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

13 (0x0D): Set LCD Contrast

This command sets the contrast or vertical viewing angle of the display. Initiated by the host, responded to by the CFA635.

```
type: 0x0D = 1310  
valid data_length is 1  
data[0] = contrast setting (0-254 valid)  
    60 = light  
    120 = about right  
    150 = dark  
    151-254 = very dark (may be useful at cold temperatures)
```

The return packet will be:

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

Set LCD Contrast is one of the items stored by the command [4 \(0x04\): Store Current State as Boot State](#).

14 (0x0E): Set LCD & Keypad Backlights

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

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

The return packet will be:

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

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

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

```
0 = off
1-100 = variable brightness

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

The return packet will be:

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

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

15 (0x0F): (Deprecated)

16 (0x10): Set Up Fan Reporting (FBSCAB Required)

This command will configure the CFA635-xxx-KL+[FBSCAB](#) to report the fan speed information to the host every 500 mS.

```
type = 0x10 = 1610
valid data_length is 1
data[0] = bitmask indicating which fans are enabled to report (0-15 valid)
---- 8421 Enable Reporting of this Fan's Tach Input
|||| ||||-- Fan 1: 1 = enable, 0 = disable
|||| |||--- Fan 2: 1 = enable, 0 = disable
|||| ||---- Fan 3: 1 = enable, 0 = disable
|||| |----- Fan 4: 1 = enable, 0 = disable
```

The return packet will be:

```
type = 0x40 | 0x10 = 0x50 = 8010
data_length = 0
```

If `data[0]` is not 0, then the CFA635-xxx-KL+[FBSCAB](#) will start sending 0x81: Fan Speed Report packets for each enabled fan every 500 mS, please see [0x81: Fan Speed Report](#). Each of the report packets is staggered by 1/8 of a second.

Reporting a fan will override the fan power setting to 100% for up to 1/8 of a second every 1/2 second. Please see Fan Connections in the [FBSCAB Datasheet](#) for a detailed description.

17 (0x11): Set Fan Power (FBSCAB Required)

When power is applied to the CFA635+[FBSCAB](#)+[WR-DOW-Y17](#) temperature sensors, it detects any devices (WR-DOW-Y17) connected to the Dallas Semiconductor 1-Wire (DOW) bus and stores the device's information. This command will allow the host to read the device's information.

```
type = 0x11 = 1710
valid data_length is 4
data[0] = power level for FAN 1 (0-100 valid)
data[1] = power level for FAN 2 (0-100 valid)
data[2] = power level for FAN 3 (0-100 valid)
data[3] = power level for FAN 4 (0-100 valid)
```

The return packet will be:

```
type = 0x40 | 0x11 = 0x51 = 8110  
data_length = 0
```

Set Fan Power is one of the items stored by the command [4 \(0x04\): Store Current State as Boot State](#).

18 (0x12): Read DOW Device Information

When power is applied to the CFA635, it detects any devices ([WR-DOW-Y17](#) temperature sensors) connected to the Dallas Semiconductor 1-Wire (DOW) bus and stores the device's information. This command will allow the host to read the device's information.

The first byte returned is the Family Code of the Dallas 1-Wire device.

REGARDING COMMAND 18: READ DOW DEVICE INFORMATION

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

These settings are factory default but may be changed by the user, see command [34 \(0x22\): Set or Set and Configure GPIO Pins](#).

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

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

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

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

This setting must be saved as the boot state, so when the CFA633 reboots it will detect the DOW devices.

```
type: 0x12 = 1810  
valid data_length is 1  
data[0] = device index (0-15 valid)
```

The return packet will be:

```
type: 0x40 | 0x12 = 0x52 = 8210  
data_length = 9  
data[0] = device index (0-15 valid)  
data[1-8] = ROM ID of the device
```

If data[1] is 0x22 ([DS1822](#) Econo 1-Wire Digital Thermometer temperature sensor) or 0x28 ([DS18B20](#) High Precision 1-Wire Digital Thermometer temperature sensor used on our [WR-DOW-Y17](#)), then that device can be set up to automatically convert and report the temperature every second. See the command [19 \(0x13\): Set Up Temperature Reporting](#).

19 (0x13): Set Up Temperature Reporting (FBSCAB Required)

This command will configure the CFA635 to report the temperature information to the host every second.

```

type: 0x13 = 1910
valid data_length is 4
data[0-3] = 32-bit bitmask indicating which temperature sensors fans 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]= must be 0
data[3]= must be 0

```

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 = 8310
data_length = 0

```

20 (0x14): Arbitrary DOW Transaction (FBSCAB Required)

The CFA635-xxx-KL+FBSCAB can function as an RS-232 to Dallas 1-Wire bridge. This command allows you to specify arbitrary transactions on the 1-Wire bus. 1-Wire commands follow this basic layout:

```

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

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

```

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

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

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

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

The return packet will be:

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

21 (0x15): Not Accessible

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

The controller on the CFA635 is HD44780 compatible. Generally, you won't need low-level access to the LCD controller but some arcane functions of the HD44780 are not exposed by the CFA635's command set. This command allows you to access the CFA635's LCD controller directly.

IMPORTANT: It is possible to corrupt the CFA635 display using this command.

```
type: 0x16 = 2210
data_length: 2
data[0]: location code
        0 = "Data" register
        1 = "Control" register, RE=0
        2 = "Control" register, RE=1
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 CFA635 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
Valid values of the mask are \000-\063.
```

The return packet will be:

```
type: 0x40 | 0x17 = 0x57 = 8710
data_length = 0
```

Configure Key Reporting is one of the items stored by the command [4 \(0x04\): Store Current State as Boot State](#).

24 (0x18): Read Keypad, Polled Mode

In some situations, it may be convenient for the host to poll the CFA635 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 that have been pressed since the last poll
data[2] = bit mask showing the keys that have been released since the last poll
```

25 (0x19): Set Fan Power Fail-Safe (FBSCAB Required)

The combination of the CFA635-xxx-KL+FBSCAB+WR-FAN-X01 cable can be used as part of an active cooling system. The fans can be slowed down to reduce noise when a system is idle or when the ambient temperature is low. The fans speed up when the system is under heavy load or the ambient temperature is high.

Since there are a very large number of ways to control the speed of the fans (thresholds, thermostat, proportional, PID, multiple temperature sensors contributing to the speed of several fans, etc.), there was no way to foresee the particular requirements of your system and include an algorithm in the CFA635's firmware that would be an optimal fit for your application.

Varying fan speeds under host software control gives the ultimate flexibility in system design but would typically have a fatal flaw: a host software or hardware failure could cause the cooling system

to fail. If the fans were set at a slow speed when the host software failed, system components may be damaged due to inadequate cooling.

The fan power fail-safe command allows host control of the fans without compromising safety. When the fan control software activates, it should set the fans that are under its control to fail-safe mode with an appropriate timeout value. If for any reason the host fails to update the power of the fans before the timeout expires, the fans previously set to fail- safe mode will be forced to 100% power.

```
#define FAN_1      0x01
#define FAN_2      0x02
#define FAN_3      0x04
#define FAN_4      0x08

type = 0x19 = 2510
data_length = 2
data[0] = bit mask of fans set to fail-safe (1-15 valid)
data[1] = timeout value in 1/8 second ticks:
    1 = 1/8 second
    2 = 1/4 second
    255 = 31 7/8 seconds
```

The return packet will be:

```
type = 0x40 | 0x19 = 0x59 = 8910
data_length = 0
```

26 (0x1A): Set Fan Tachometer Glitch Filter (FBSCAB Required)

The combination of the CFA635-xxx-KL+[FBSCAB+WR-FAN-X01](#) cable controls fan speed by using PWM. Using PWM turns the power to a fan on and off quickly to change the average power delivered to the fan. The CFA635 uses approximately 18 Hz for the PWM repetition rate. The fan's tachometer output is only valid if power is applied to the fan. Most fans produce a valid tachometer output very quickly after the fan has been turned back on but some fans take time after being turned on before their tachometer output is valid.

This command allows you to set a variable-length delay after the fan has been turned on before the CFA635 will recognize transitions on the tachometer line. The delay is specified in counts, each count being nominally 552.5 μ S long (1/100 of one period of the 18 Hz PWM repetition rate).

In practice, most fans will not need the delay to be changed from the default length of 1 count. If a fan's tachometer output is not stable when its PWM setting is other than 100%, simply increase the delay until the reading is stable.

Typically, you would

- (1) start at a delay count of 50 or 100,
- (2) reduce it until the problem reappears, and then
- (3) slightly increase the delay count to give it some margin.

Setting the glitch delay to higher values will make the RPM monitoring slightly more intrusive at low power settings. Also, the higher values will increase the lowest speed that a fan with RPM reporting enabled will seek at 0% power setting.

The Fan Glitch Delay is one of the items stored by the command [4 \(0x04\): Store Current State as Boot State](#).

```
type = 0x1A = 2610
data_length = 4
```

```
data[0] = delay count of fan 1 (0-100 valid)
data[1] = delay count of fan 2 (0-100 valid)
data[2] = delay count of fan 3 (0-100 valid)
data[3] = delay count of fan 4 (0-100 valid)
```

The return packet will be:

```
type = 0x40 | 0x1A = 0x5A = 9010
data_length = 0
```

27 (0x1B): Query Fan Power & Fail-Safe Mask (FBSCAB Required)

The combination of the CFA635-xxx-KL+[FBSCAB](#)+[WR-FAN-X01](#) cable is required to use this command. This command can be used to verify the current fan power and verify which fans are set to fail-safe mode.

```
#define FAN_1      0x01
#define FAN_2      0x02
#define FAN_3      0x04
#define FAN_4      0x08

type = 0x1B = 2710
data_length = 0
```

The return packet will be:

```
type = 0x40 | 0x1B = 0x5B = 9110
data_length = 5
data[0] = fan 1 power
data[1] = fan 2 power
data[2] = fan 3 power
data[3] = fan 4 power
data[4] = bitmask of fans with fail-safe set
```

28 (0x1C): Set ATX Power Switch Functionality

For enable ATX power switch functionality a [WR-PWR-Y25](#) or [WR-PWR-Y38](#) cable is required.

The CFA635 with ATX power switch functionality enabled can be used to replace the function of the power and reset switches in a standard ATX-compatible system. The ATX Power Switch Functionality is one of the items stored by the command [4 \(0x04\): Store Current State as Boot State](#).

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 factory default but may be changed by the user. Please see command [34 \(0x22\): Set or Set and Configure GPIO Pins](#). These settings must be saved as the boot state.

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

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

This configuration can be assured by sending the following command:

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

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

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

This configuration can be assured by sending the following command:

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

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

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

This configuration can be assured by sending the following command:

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

These settings must be saved as the boot state.

The RESET (GPIO[3]) and POWER CONTROL (GPIO[2]) lines on the CFA635 are normally high-impedance. Electrically, they appear to be disconnected or floating. When the CFA635 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 CFA635 changes the lines back to high-impedance.

Four functions may be enabled by Command 28

Function 1: KEYPAD_RESET

If POWER-ON SENSE (GPIO[1]) is high, holding the green check key for 4 seconds will pulse RESET (GPIO[3]) pin for 1 second. During the 1-second pulse, the CFA635 will show "RESET", and then the CFA635 will reset itself, showing its boot state as if it had just powered on. Once the pulse has finished, the CFA635 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 CFA635 will show "POWER ON", then the CFA635 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 CFA635 will continue to drive the line for a maximum of 5 additional seconds. During this time, the CFA635 will show "POWER OFF".

Function 4: LCD_OFF_IF_HOST_IS_OFF

If LCD_OFF_IF_HOST_IS_OFF is set, the CFA635 will blank its screen and turn off its backlight to simulate its power being off any time POWER-ON SENSE is low.

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

```
type: 0x1C = 2810
data_length: 1 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 CFA635. If the system monitor program fails to reset the CFA635's watchdog timer, the CFA635 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. Please 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 once it has been enabled, simply set timeout to 0.

If the command is not re-issued within timeout seconds, then the CFA635 will reset the host (see command 28 for details). Since the watchdog is off by default when the CFA635 powers up, the CFA635 will not issue another host reset until the host has once again enabled the watchdog.

The return packet will be:

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

30 Read Reporting & Status

This command can be used to verify the current items configured to report to the host, as well as some other miscellaneous status information. The combination of CFA635-xxx-KL+[FBSCAB+WR-DOW-Y17](#) temperature sensors is required to report the temperature information. The combination of the CFA635-xxx-KL+[FBSCAB+WR-FAN-X01](#) cable is required to control fans.

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

The return packet will be:

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



```
data[ 0] = fan 1-4 reporting status (as set by command 16)
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] = fan RPM glitch delay[0] (as set by command 26)
data[10] = fan RPM glitch delay[1] (as set by command 26)
data[11] = fan RPM glitch delay[2] (as set by command 26)
data[12] = fan RPM glitch delay[3] (as set by command 26)
data[13] = contrast setting (as set by command 13)
data[14] = backlight setting (as set by command 14)
```

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 22
data[0]: col = x = 0 to 9
data[1]: row = y = 0 to 3
data[2-21]: text to place on the LCD, variable from 1 to 20 characters
```

The return packet will be:

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

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

32: Reserved for CFA631 Key Legends

33 (0x21): Set Baud Rate

This command will change the CFA635's baud rate. The CFA635 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 CFA635 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](#) if you want the CFA635 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 (FBSCAB Required)

The CFA635 (hardware versions v1.4 and up, firmware versions 1.9 and up) has five pins for user-definable general purpose input / output (GPIO). These pins are shared with the DOW and ATX functions. Be careful when you configure the GPIO if you want to use the ATX or DOW at the same time.

The architecture of the CFA635 allows great flexibility in the configuration of the GPIO pins. They can be set as input or output. They can output constant high or low signals or a variable duty cycle 100 Hz PWM signal.

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

The CFA635 continuously polls the GPIOs as inputs at 32 Hz. The present level can be queried by the host software at a lower rate. The CFA635 also keeps track of whether there were rising or falling edges since the last host query (subject to the resolution of the 32 Hz sampling). This means that the host is not forced to poll quickly in order to detect short events. The algorithm used by the CFA635 to read the inputs is inherently "bounce-free".

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

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

REGARDING SETTING AND CONFIGURING GPIO PINS

The GPIO pins may also be used for ATX control through header J8 and temperature sensing through the CFA633's DOW header. By factory default, the GPIO output setting, function, and drive mode are set correctly to enable operation of the ATX and DOW functions. **The GPIO output setting, function, and drive mode must be set to the correct values in order for the ATX and DOW functions to work. Improper use of this command can disable the ATX and DOW functions.** The [cfTest](#) may be used to easily check and reset the GPIO configuration to the default state so the ATX and DOW functions will work.

The GPIO configuration is one of the items stored by the command [4 \(0x04\): Store Current State as Boot State](#).

```
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/GPO to modify on optional FBSCAB's connector when  
using CFA635+FBSCAB+Custom cable. GPOs are for the 4 status LEDs on the  
module's front panel.
```

```

0 = GPIO[0] = H1, Pin 11
1 = GPIO[1] = H1, Pin 12 (default is ATX Host Power Sense)
2 = GPIO[2] = H1, Pin 9 (default is ATX Host Power Control)
3 = GPIO[3] = H1, Pin 10 (default is ATX Host Reset Control)
4 = GPIO[4] = H1, Pin 13 (default is DOW I/O--always has 1KΩ hardware
pull-up on FBSCAB.)
5 = GPO[ 5] = LED 3 (bottom) green die
6 = GPO[ 6] = LED 3 (bottom) red die
7 = GPO[ 7] = LED 2 green die
8 = GPO[ 8] = LED 2 red die
9 = GPO[ 9] = LED 1 green die
10 = GPO[10] = LED 1 red die
11 = GPO[11] = LED 0 (top) green die
12 = GPO[12] = LED 0 (top) red die
13-255 = not accessible

```

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

```

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

```

```

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

```

The return packet will be:

```

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

```

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

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] = H1, Pin 11
  1 = GPIO[1] = H1, Pin 12 (default is ATX Host Power Sense)
  2 = GPIO[2] = H1, Pin 9 (default is ATX Host Power Control)
  3 = GPIO[3] = H1, Pin 10 (default is ATX Host Reset Control)
  4 = GPIO[4] = H1, Pin 13 (default is DOW I/O--always has 1KΩ hardware
    pull-up on FBSCAB.)
  5 = GPO[ 5] = LED 3 (bottom) green die
  6 = GPO[ 6] = LED 3 (bottom) red die
  7 = GPO[ 7] = LED 2 green die
  8 = GPO[ 8] = LED 2 red die
  9 = GPO[ 9] = LED 1 green die
 10 = GPO[10] = LED 1 red die
 11 = GPO[11] = LED 0 (top) green die
 12 = GPO[12] = LED 0 (top) red die
 13-255 = not accessible

```

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 Enable Reporting of this Fan's Tach Input
|||| ||| |-- 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
```

10. Character Generator ROM (CGROM)

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

Character Generator ROM (CGROM) for CrystalFontz CFA-635

upper 4 bits lower 4 bits	0 _d 0000.	16 _d 0001.	32 _d 0010.	48 _d 0011.	64 _d 0100.	80 _d 0101.	96 _d 0110.	112 _d 0111.	128 _d 1000.	144 _d 1001.	160 _d 1010.	176 _d 1011.	192 _d 1100.	208 _d 1101.	224 _d 1110.	240 _d 1111.
0 _d 0000.	CGRAM [0]															
1 _d 0001.	CGRAM [1]															
2 _d 0010.	CGRAM [2]															
3 _d 0011.	CGRAM [3]															
4 _d 0100.	CGRAM [4]															
5 _d 0101.	CGRAM [5]															
6 _d 0110.	CGRAM [6]															
7 _d 0111.	CGRAM [7]															
8 _d 1000.	CGRAM [0]															
9 _d 1001.	CGRAM [1]															
10 _d 1010.	CGRAM [2]															
11 _d 1011.	CGRAM [3]															
12 _d 1100.	CGRAM [4]															
13 _d 1101.	CGRAM [5]															
14 _d 1110.	CGRAM [6]															
15 _d 1111.	CGRAM [7]															

Figure 8: Character Generator ROM (CGROM)

11. 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. ***If you need 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	
Bicolor status LEDs	50,000 to 100,000 hours	
Yellow-green LED Display and Keypad Backlight (CFA635-YYE-Kx)	50,000 to 100,000 hours	
White LED Display and Blue LED Keypad Backlights <i>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.</i>	Power-On Hours	% of Initial Brightness
	<10,000	>70%
	<50,000	>50%

11.1. 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](#).

12. Care and Handling Precautions

For optimum operation of the CFA635-XXX-KL and to prolong its life, please follow the precautions described below.

12.1. ESD (Electrostatic Discharge)

The CFA635-xxx-KL circuitry is industry standard CMOS logic and susceptible to ESD damage. Please use industry standard antistatic precautions as you would for any other static sensitive devices such as expansion cards, motherboards, or integrated circuits. Ground your body, work surfaces, and equipment.

12.2. 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. Please peel off the protective film slowly. Peeling off the protective film abruptly may generate static electricity.
- When handling the module, avoid touching the polarizer. Finger oils are difficult to remove.
- To protect the soft plastic polarizer from damage, place 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. Reversing polarity will immediately ruin the module.

12.3. Avoid Shock, Impact, Torque, or Tension

- Do not expose the CFA635 to strong mechanical shock, impact, torque, or tension.
- Do not drop, toss, bend, or twist the CFA635.
- Do not place weight or pressure on the CFA635.

12.4. 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 warm soapy water.

12.5. Cleaning

- The polarizer (laminated to the glass), is soft plastic that can easily be scratched or damaged, so use extra care when you clean it.
 - Do not clean the polarizer with liquids.
 - Do not wipe the polarizer with any type of cloth or swab (for example, Q-tips).
 - Use the removable protective film to remove smudges (for example, fingerprints), and any foreign matter. If you no longer have the protective film, use standard transparent office tape (for example, Scotch® brand “Crystal Clear Tape”).
 - If the polarizer becomes dusty, carefully blow it off with clean, dry, oil-free compressed air.
 - The polarizer will eventually become hazy if you do not use care when cleaning it.
 - Contact with moisture may permanently spot or stain the polarizer.

12.6. Operation

- Protect the CFA635 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. (You may need to adjust the contrast.)

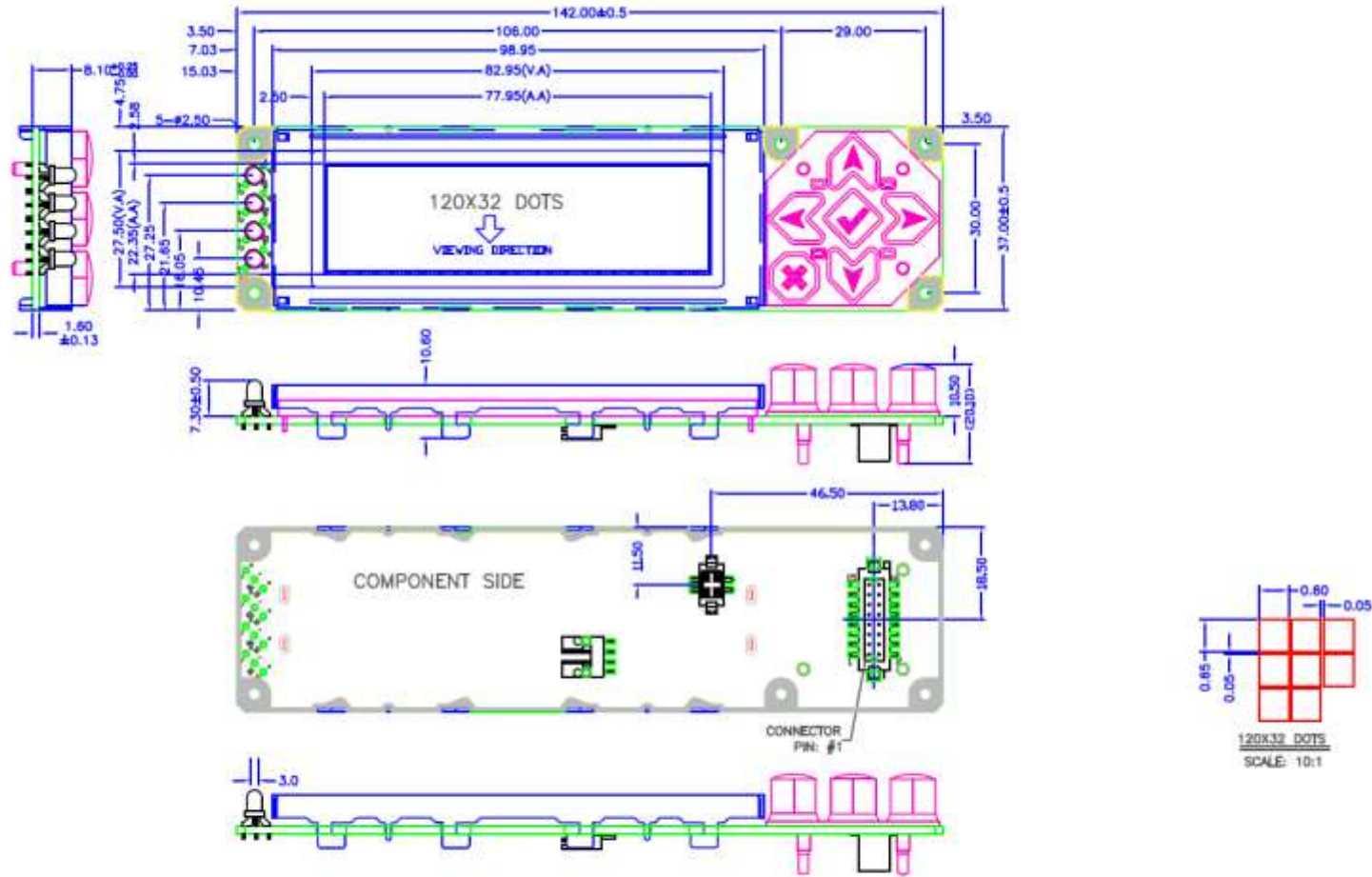
- Operate away from dust, moisture, and direct sunlight.
- Adjust backlight brightness so the display is readable, but not too bright.
- Dim or turn off the backlight during periods of inactivity to conserve the backlight lifetime.

12.7. 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 CFA635 while in storage.
- Please recycle your outdated Crystalfontz modules at an approved facility.

13. Mechanical Drawings

Module Outline Drawing Front, Back, and Side Views

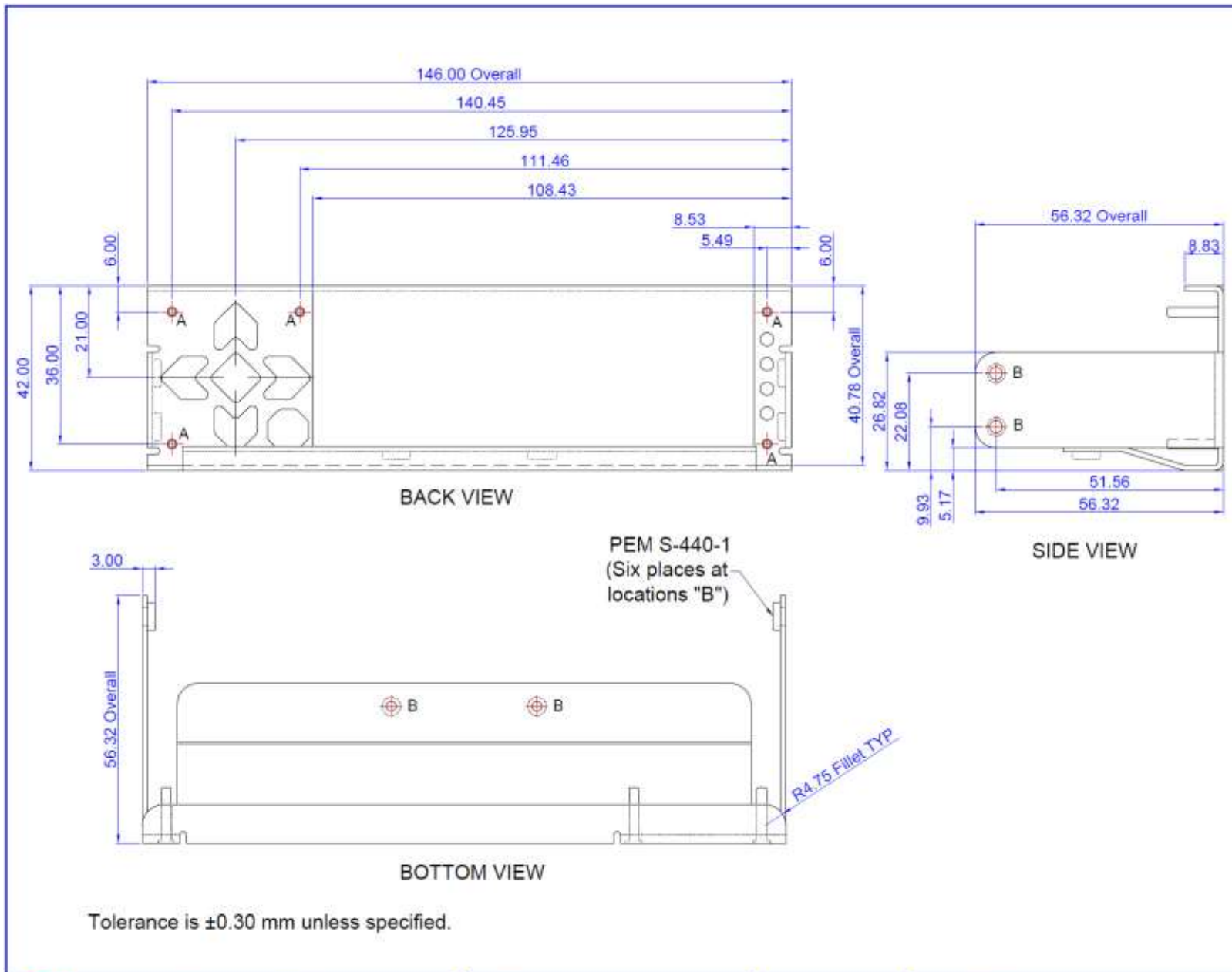



Tolerance is ±0.3mm unless specified.



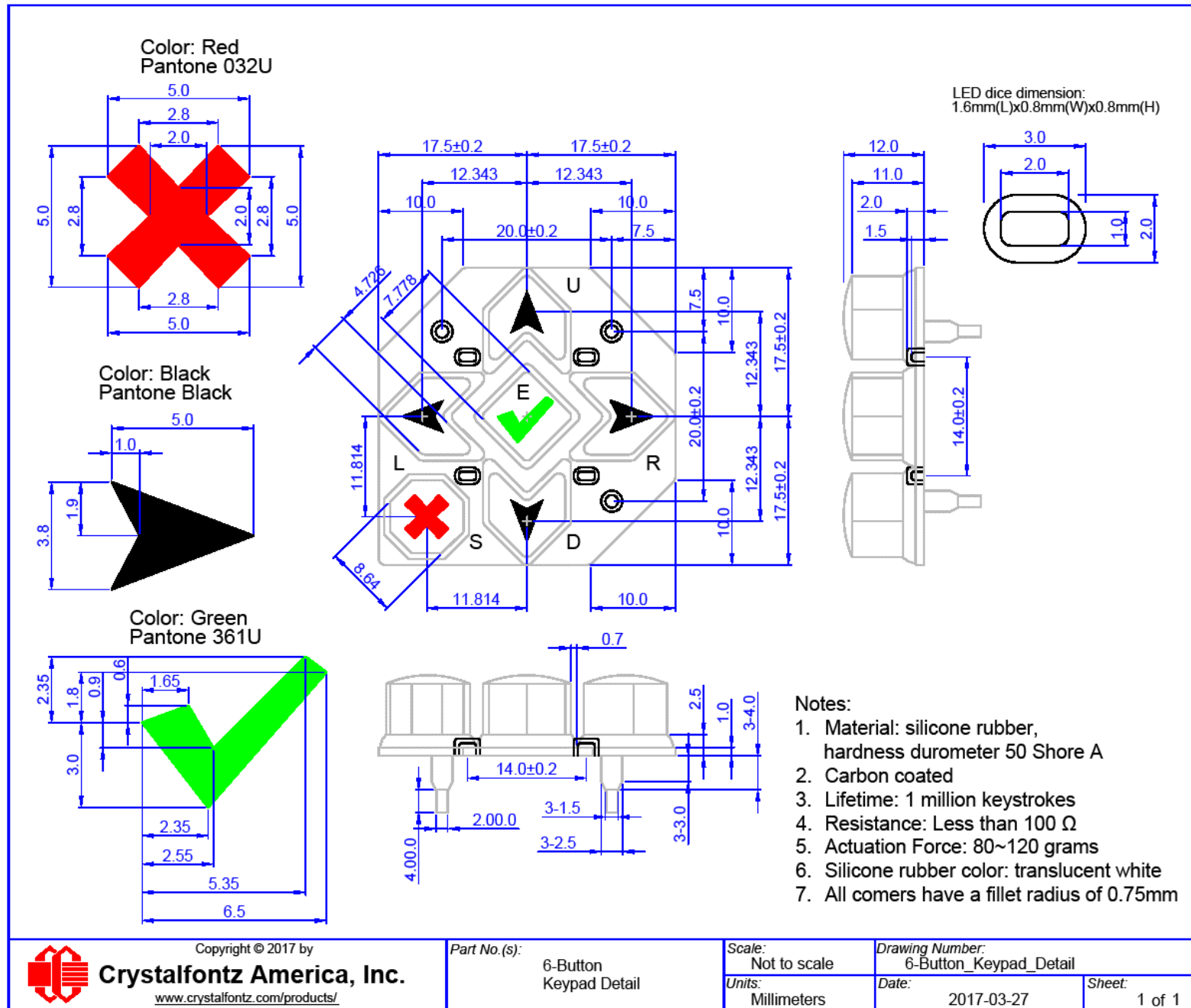
Scale: Not to scale	Part No.(s): CFA635-xxx-KL
Date: 2017-04-25	Hardware Rev.: h1.5v1
Units: Millimeters	Drawing No.: CFA635-Family-Master

Optional Mounting Bracket Drawing

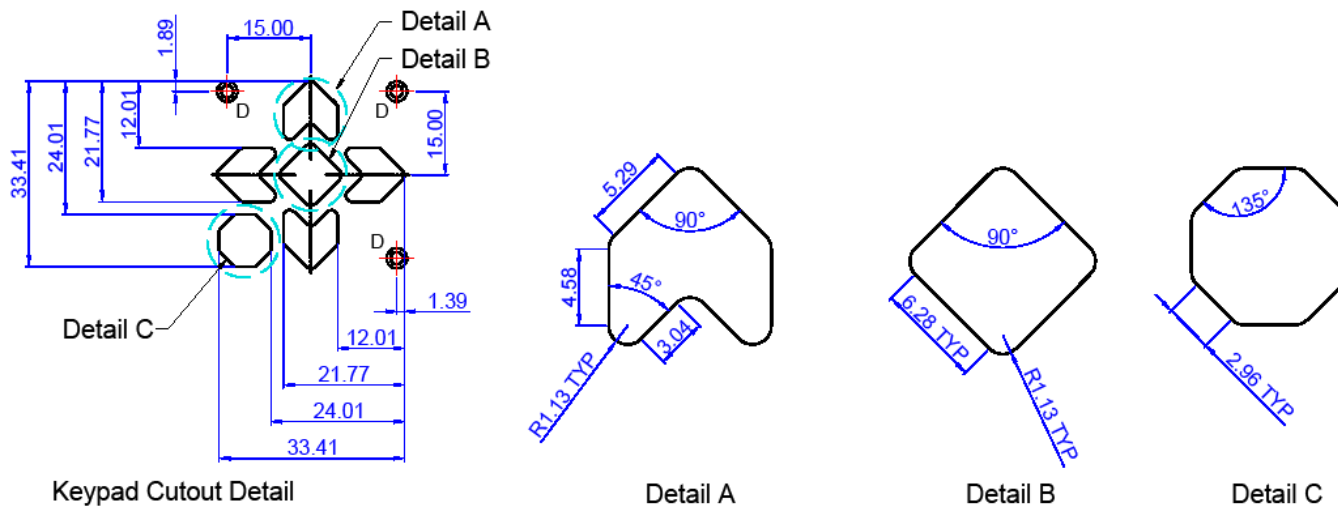


 Crystalfontz America, Inc. www.crystalfontz.com/products/	Copyright © 2017 by	Part No. (s):	Scale:	Drawing Number:	
		CFA635 Family Mounting Bracket	Not to scale	Bracket_master	
			Units:	Date:	Sheet:
			Millimeters	2017-03-27	1 of 1

Keypad Detail Drawing



Panel Mounting Application Cutout Drawing



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

- PEM FH-256-8
- Bivar Inc. 9913-5 mm spacer
- 2-56 "small profile" hex nut
- Use appropriate screen printed overlay to cover display bezel and mounting hardware, and to protect LCD from scratching.



Copyright © 2017 by

Crystalfontz America, Inc.

www.crystalfontz.com/products/

Part No.(s):

6-Button Keypad
Panel Mounting
Application Cutout

Scale:
Not to scale

Units:
Millimeters

Drawing Number:
6-Button_Panel_Cutout_Master

Date:
2017-03-27

Sheet:
1 of 1

14. 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: <https://www.crystalfontz.com/product/cftest>
- Windows compatible example program and source: <https://www.crystalfontz.com/product/635wintest>
- 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 CFA635 packet. Some of the algorithms were contributed by forum members and originally written for CFA631 and CFA635. The CRC used in the CFA635 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] =
{0x0000,0x01189,0x02312,0x0329B,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,0x0EA66,0x0DBFD,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 CFA635 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 CFA635 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 copyleft or anything else.
'Some visual basic concepts taken from:
'<http://www.planet-source code.com/vb/scripts/ShowCode.asp?txtCodeId=21434&lngWId=1>
'most of the algorithm is from functions in 633 WinTest:
'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,0x0AC42,0x09ED9,0x08F50,0x0FBEE,0x0EA66,0x0D8FD,0x0C974,
0x04204,0x0538D,0x06116,0x0709F,0x00420,0x015A9,0x02732,0x036BB,
0x0CE4C,0x0DFC5,0x0ED5E,0x0FCD7,0x08868,0x099E1,0x0AB7A,0x0BAF3,
0x05285,0x0430C,0x07197,0x0601E,0x014A1,0x00528,0x037B3,0x0263A,
0x0DECD,0x0CF44,0x0FDDF,0x0EC56,0x098E9,0x08960,0x0BBFB,0x0AA72,
0x06306,0x0728F,0x04014,0x0519D,0x02522,0x034AB,0x00630,0x017B9,
0x0EF4E,0x0FEC7,0x0CC5C,0x0DDD5,0x0A96A,0x0B8E3,0x08A78,0x09BF1,
0x07387,0x0620E,0x05095,0x0411C,0x035A3,0x0242A,0x016B1,0x00738,
0x0FFCF,0x0EE46,0x0DCDD,0x0CD54,0x0B9EB,0x0A862,0x09AF9,0x08B70,
0x08408,0x09581,0x0A71A,0x0B693,0x0C22C,0x0D3A5,0x0E13E,0x0F0B7,
0x00840,0x019C9,0x02B52,0x03ADB,0x04E64,0x05FED,0x06D76,0x07CFF,
0x09489,0x08500,0x0B79B,0x0A612,0x0D2AD,0x0C324,0x0F1BF,0x0E036,
0x018C1,0x00948,0x03BD3,0x02A5A,0x05EE5,0x04F6C,0x07DF7,0x06C7E,
0x0A50A,0x0B483,0x08618,0x09791,0x0E32E,0x0F2A7,0x0C03C,0x0D1B5,
0x02942,0x038CB,0x00A50,0x01BD9,0x06F66,0x07EEF,0x04C74,0x05DFD,
0x0B58B,0x0A402,0x09699,0x08710,0x0F3AF,0x0E226,0x0D0BD,0x0C134,
0x039C3,0x0284A,0x01AD1,0x00B58,0x07FE7,0x06E6E,0x05CF5,0x04D7C,
0x0C60C,0x0D785,0x0E51E,0x0F497,0x08028,0x091A1,0x0A33A,0x0B2B3,
0x04A44,0x05BCD,0x06956,0x078DF,0x00C60,0x01DE9,0x02F72,0x03EFB,
0x0D68D,0x0C704,0x0F59F,0x0E416,0x090A9,0x08120,0x0B3BB,0x0A232,
0x05AC5,0x04B4C,0x079D7,0x0685E,0x01CE1,0x00D68,0x03FF3,0x02E7A,
0x0E70E,0x0F687,0x0C41C,0x0D595,0x0A12A,0x0B0A3,0x08238,0x093B1,
0x06B46,0x07ACF,0x04854,0x059DD,0x02D62,0x03CEB,0x00E70,0x01FF9,
0x0F78F,0x0E606,0x0D49D,0x0C514,0x0B1AB,0x0A022,0x092B9,0x08330,
0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78
};
public static int compute(byte[] data)
{
int newCrc = 0xFFFF;
for (int i = 0; i < data.length; i++)
{
int lookup = crcLookupTable[(newCrc ^ data[i]) & 0xFF];
newCrc = (newCrc >> 8) ^ lookup;
}
return(~newCrc);
}
}
```

Algorithm 6: "Perl" Table Implementation

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

```
#!/usr/bin/perl

use strict;

my @CRC_LOOKUP =
(0x0000,0x01189,0x02312,0x0329B,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
_loop:
    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           _loop            ; 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

```

```

RXBUF2          equ    0x8C0          ; display
                                     ; 32-byte receive buffer for
TXBUF2          equ    0x8E0          ; Display
                                     ; 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
    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
    MOVLFS upper stringname, TBLPTRU
    MOVLFS high stringname, TBLPTRH
    MOVLFS low stringname, TBLPTRL
    call    MOVE_STR
endm
;=====
MOVE_STR:
    tblrd   *+
    movf    TABLAT, w
    bz     ms1b
    movwf   POSTINC1
    incf    SendCount
    goto   MOVE_STR

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