INTELLIGENT LCD MODULE SPECIFICATIONS

CFA735-TFK-KR
CFA735-TML-KR

CFA735-TFK-KT
CFA735-TML-KT

Hardware Version: v1.3
Firmware Version: v1.2

Datasheet Release: 2021-01-14

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

Table of Figures

1. General Information

Datasheet Revision History
Hardware Version: v1.3 Firmware Version: v1.5 Datasheet Release: 2021-01-14 For information about firmware and hardware revisions, see the Part Change Notifications (PCN) under "News" in our website's navigation bar. 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
You can check for or subscribe to Part Change Notices for this display module on our website.

Variations
Slight variations between lots are normal (e.g., contrast, color, or intensity).

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 © 2021 by Crystalfontz America, Inc.,12412 East Saltese Avenue, Spokane Valley, WA 99216 U.S.A.

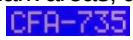
2. Introduction

The CFA735 family of modules has two interface choices:

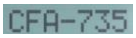
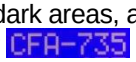
- CFA735-xxx-KR ("Logic Level, Inverted" Serial and USB)
- CFA735-xxx-KT ("Full Swing" CFA-RS232 Serial and USB)

This datasheet has information for the following interface modules:

"Logic Level, Inverted" Serial and USB:

- CFA735-TFK-KR  (dark letters on a light background; this display can be read in normal office lighting, in dark areas, and in bright sunlight)
- CFA735-TML-KR  (light letters on a blue background; this display can be read in normal office lighting and in dark areas)

"Full Swing" CFA-RS232 Serial and USB:

- CFA735-TFK-KT  (dark letters on a light background; this display can be read in normal office lighting, in dark areas, and in bright sunlight)
- CFA735-TML-KT  (light letters on a blue background; this display can be read in normal office lighting and in dark areas)

All variants can simultaneously use a USB and a Serial interface ("Logic Level, Inverted" Serial or "Full Swing" CFA-RS232 Serial). Modules with "Full Swing" CFA-RS232 serial have a mounted CFA-RS232 level translator board.

When the information in this datasheet applies to both interfaces the term "CFA735" is used.

2.1. 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).
- Six-button translucent silicone keypad with screened legend is backlit with white LEDs. Fully decoded keypad: any key combination is valid and unique.
- Only a single supply is needed. Wide power supply voltage range (V_{DD} = +3.3v to +5.5v), so perfect for embedded systems.
- May be installed in a standard half-height 51/4 drive bay by using an optional drive bay mounting bracket or an optional SLED bracket. The SLED holds the CFA735 display module, an optional CFA-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.
- Adjustable contrast. The default contrast value for the module will be acceptable for most applications. If necessary, the contrast can be adjusted 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 that allows for 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.
- DAC (Digital-to-Analog Converter) controls the constant current LED driver.
- Powered by an ST-Micro STM32F103 series 32-bit ARM-based microcontroller and Sitronix ST7529 driver/controller.
- 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.
- Optional ATX Power Supply control functionality allows the CFA735's buttons to replace the Power and Reset switches in a system, simplifying front panel design. Optional Crystalfontz WR-PWR-Y25 or WR-PWR-Y38 cables simplify connection to the host's motherboard.
- Hardware watchdog can reset host on host software failure.

- The CFA735-xxx-KR may be used with an optional CFA-FBSCAB (System Cooling Accessory Board). The combination of the CFA735-xxx-KR with the optional CFA-FBSCAB (CFA735+CFA-FBSCAB) allows:
  - Four fan connections with RPM monitoring and variable PWM (Pulse Width Modulation), fan power control.,
  - Fail-safe fan power settings to safely control the fans based on temperature,
  - 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 CFA-FBSCAB, buy the WR-PWR-Y25 or the WR-PWR-Y38 ATX power cable,
  - For more information, see ATX Power Supply Control and the CFA-FBSCAB Datasheet on crystalfontz.com.
- Crystalfontz America, Inc. is ISO 9001:2015 certified.
- A Declaration for Conformity, RoHS, and REACH:SVHC are available under the Datasheets & Files tab on display web pages.

2.2. Difference Between the Two Serial Interfaces

Both of the serial interfaces use firmware that brings the two UART pins (Tx & Rx) of the CFA735's microcontroller to the CFA735's H1 connector.

“Logic Level, Inverted” Serial (CFA735-xxx-KR)

The CFA735-xxx-KR exposes the UART Tx & Rx (“logic level, inverted”, 0v to 3.3v nominal), signals on pin 1 and pin 2 of the CFA735's connector H1. When in close proximity, the UART Rx and Tx pins can be cabled directly to the CFA735-xxx-KR's Tx and Rx pins. No RS232 level translators are required on either end.

“Full Swing” RS232 Serial (CFA735-xxx-KT)

The CFA735-xxx-KT is a CFA735-xxx-KR with a CFA-RS232 level translator board attached. The CFA735-xxx-KT is the correct choice for an embedded controller or host system has a “real” RS232 serial port (-5v to +5v “Full Swing” serial interface) available.

2.3. Comparison of CFA735 Family and CFA635 Family

The CFA735 family is mechanically compatible with and uses the same command format as the CFA635 family. In most applications, a CFA735 module can be used in place of a CFA635 module with little or no change to the host system's software required.

Please note that the USB driver used on the host PC is different for the CFA735 than for the CFA635.

All CFA735 modules support USB and serial logic level interfaces simultaneously. The CFA635 modules are USB only or Serial only. The CFA735 emulates all the CFA635 functions, with the exception of command 22 (0x16): Send Command Directly to the LCD Controller.

Unlike the CFA635+CFA-FBSCAB, the CFA735+CFA-FBSCAB has no ATX functionality provided through the CFA-FBSCAB. However, ATX control is available using the H1 connector on the CFA735.

Key Differences	CFA735+CFA-FBSCAB	CFA635+CFA-FBSCAB or CFA631+SCAB
WR-DOW-Y17 Temperature Sensors	Supports up to 16 sensors.	Supports up to 32 sensors.
Number of fans that can be used with a SCAB or CFA-FBSCAB	Up to 4 fans.	Up to 4 fans for USB interface. Up to 3 fans for Serial interface.
Fan speed when module is disconnected from the SCAB or CFA-FBSCAB	Default fan speed in nonvolatile memory of CFA-FBSCAB, set by CFA735 command.	100% power.
GPIOs	H1 connector on CFA735.	H1 connector on CFA631 and CFA635 passes through to J8
ATX for module with SCAB or CFA-FBSCAB	H1 connector on CFA735 using WR-PWR-Y25 cable.	J8 connector on SCAB using WR-PWR-Y05 or WR-PWR-Y14 cable.
ATX for module without SCAB or CFA-FBSCAB		H1 connector on CFA631 and CFA635 using WR-PWR-Y25 cable.

2.4. Firmware

How to Identify Revision Numbers

There are three ways to identity the revision number on the display:

- Before applying power to the module, press the right arrow key on the keypad. Apply power, keeping the right arrow key depressed until the firmware revision displays. As long as the keypad is depressed, this information is displayed. When the right arrow key is released, the display clears after five seconds.
- When coming out of reset, keep the right arrow key depressed until the firmware revision displays. As long as the keypad is depressed, this information is displayed. When the right arrow key is released, the display clears after five seconds.
- Using command 1 (0x01): Get Hardware & Firmware Version.

CFA635 Emulation Code (Shipped by Default from Factory)

Each CFA735 module has the current CFA635 emulation firmware revision installed at the time it is shipped. Firmware updates are announced through a Part Change Notice (PCN).

Use Custom Code

To use custom code instead of the pre-installed CFA635 emulation firmware, clone the cfa_735_simple git repository. Follow the ***readme.txt*** instructions to compile and install the firmware on the CFA735-xxx-KR (JTAG programmer/ debugger required). User code is community supported in our forum. Crystalfontz has no phone or email support for user code.

IMPORTANT: Installation of custom firmware on a CFA735 will remove firmware supported by Crystalfontz. There is no method to reinstall the supported firmware on CFA735 without returning the module to Crystalfontz.

NOTE: When not in use, always verify that the microSD card socket is in the ***closed and locked*** position.

2.5. Module Classification Information

CFA ❶ 735 ❷ - x ❸ x ❹ x ❺ - K ❻ R ❼ | CFA ❶ 735 ❷ - x ❸ x ❹ x ❺ - K ❻ T ❼

❶	**Brand**	Crystalfontz America, Incorporated
❷	**Model Identifier**	735
❸	**Backlight Type & Color**	T – LED, white Y – LED, yellow-green
❹	**Fluid Type, Image (positive or negative), & LCD Glass Color**	F – FSTN, positive M – STN, negative, blue Y – STN, positive, yellow-green
❺	**Polarizer Film Type, Temperature Range, & View Angle (O 'Clock)**	K – Transflective, Wide Temperature -20°C to +70°C, 12:00 L – Transmissive, Wide Temperature -20°C to +70°C, 12:00
❻	**Special Code**	K – Manufacturer's code
❼	**Interface**	**R** – "Logic Level, Inverted" Serial and USB Bidirectional 9600 / 19200 / 115200 baud logic-level asynchronous serial interface is suitable to connect directly to microcontroller UART pins. Full-speed USB interface is available simultaneously.
		T – "Full Swing" CFA-RS232 Serial and USB Bidirectional 9600 / 19200 / 115200 baud ESD protected CFA-RS232 serial interface is provided by the included serial conversion board (named CFA-RS232 Level Translator), when connected with the appropriate cables.

NOTE: When ordering a CFA735 online, choices of configurations (including accessories) are offered through the "Customize and Add to Cart" feature.

2.6. Ordering Information

Part Number	Fluid	LCD Glass Color	Image	Polarizer Film	Backlight Color/Type
CFA735-TFK-Kx	FSTN	neutral	positive	transflective	Backlight: white Keypad: white 
CFA735-TML-Kx	STN	blue	negative	transmissive	Backlight: white Keypad: blue 

2.7. Accessories

Modules with Cables

KR Series

These CFA735-xxx-KR color variants are sold separately or as a part of a kit that includes three cables.

Kit Choices	Serial Cable in Kit
CFA735-TFK-KR1 CFA735-TML-KR1	Includes a WR-USB-Y27, a Micro-B to USB-A to 6-foot cable.
CFA735-TFK-KR2 CFA735-TML-KR2	Includes WR-USB-Y34, a Micro-B USB to 4-pin 0.1” connector for USB pins on your motherboard.

KT Series

These CFA735-xxx-KT color variants are sold separately or as a part of a kit that includes three cables.

Kit Choices	Serial Cables in Kit
CFA735-TFK-KT1	WR-232-Y08: RS232 DB9 female to 0.1" 2x5 female, 27" ribbon cable for a PC’s 9-pin serial port. WR-232-Y22: 0.1" 2x5 (female) to 0.1" 2x5 (female) x2 standard/alternate pinout. 10-pin to 10-pin for a 10-pin header. WR-PWR-Y24: PC power supply to 16-pin connector. Connect power directly from the ATX power supply.
CFA735-TML-KT1	

NOTE: Individual cables for the CFA735 module are available for purchase on our website; please see Cables.

2.8. Kit Configurations

In addition to modules sold with the cables listed above, we offer module kits that include an assortment of cables with the following accessories:

- BRACKET: a 5.25-inch half-height drive bay black plastic mounting bracket.

or

- SLED: a chassis that fits in 5.25-inch half-height drive bay. The SLED can hold a CFA735-xxx-Kx module, a CFA-FBSCAB, and a 3.5-inch hard disk drive (hard drive is not included).

and

- OVERLAY: an overlay for the front of the module with a display window of thick hard-coated polycarbonate. Overlays are sold with the bracket and SLED.

2.9. Display Mounts

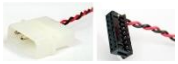
On the web page for the CFA735 Series, after you click the *"Customize and Add to Cart"* button, you will see a list of options for different cables, connectors, drive bay bracket, and SLED.

Figure 1. CFA735 Drive Bay Bracket

Figure 2. CFA735 SLED

2.10. Cables

Below is a list of some of the cables we offer to make it easy to integrate the CFA735 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-232-Y08 ~27 inches		**KT Series:** Use this cable to supply communications to the CFA735-xxx-KT through the mounted CFA-RS232 converter. Connect cable's 10-pin connector to CFA-RS232's J1 connector. Connect cable's CFA-RS232 DB9 9-pin connector to host's external 9-pin serial port.
WR-232-Y22 ~26 inches		**KT Series:** Use this cable to supply communications to the CFA735-xxx-KT. Connect one of the 10-pin connectors to the module's' J_CFA-RS232 10-pin connector. Connect cable's second 10-pin connector to host's motherboard 10-pin connector. This cable supports standard or alternate pinout motherboard CFA-RS232 connections without changing jumpers on the module.
WR-USB-Y27 ~6 feet		Connect cable's Micro-B USB connector to CFA735's Micro-B USB connector. Connect cable's USB-A connector to host's USB-A connector.
WR-USB-Y34 ~27.5 inches		Connect cable's Micro-B USB connector to CFA735's Micro-B USB connector. Connect cable's single piece 4-pin 0.1" connector to USB pins on host's motherboard. For correct orientation, note the +5v location on the 4-pin connector.
WR-PWR-Y24 ~26 inches		This cable supplies power to the CFA735 directly from a PC power supply's "hard-drive" connector, rather than the normal USB power.
WR-PWR-Y25 ~11 inches		This cable simplifies the connections for using ATX power and reset control. One end plugs into the CFA735 H1 connector. The other end has connections for power control, reset control, always on power, switched power, and ground.
WR-PWR-Y12 ~13 inches		Cable enables plugging a 4-pin "hard drive style" Molex power connector into the module's "floppy drive style" power connector, plus provides an additional female 4-pin Molex connector.
WR-EXT-Y37 ~18 inches		**For use with CFA-FBSCAB:** Used to connect the CFA735 to the CFA-FBSCAB.
WR-PWR-Y38 ~2 ft. 11 inches		Longer version of the WR-PWR-Y25 (described in row above).
WR-FAN-X01 ~16 inches		**For use with CFA-FBSCAB:** Fan extension cable for standard 3-pin fans.
WR-DOW-Y17 ~12 inches + ~12 inches between connectors		**For use with CFA-FBSCAB:** Connect ("daisy chain") up to 16 of these DOW (Dallas 1-Wire) DS18B20 temperature sensor cables to the CFA-FBSCAB.
Note that J8 of the CFA-FBSCAB is unused. ATX functions are supported by H1 on the CFA735.		

3. Mechanical Characteristics

3.1. Physical Characteristics

Item	Specification (mm)	Specification (inch, reference)
Overall Width and Height	142.0 (W) x 37.0 (H)	5.59 (W) x 1.46 (H)
Viewing Area / Bezel Opening	83.0 (W) x 27.5 (H)	3.27 (W) x 1.08 (H)
Active Area	79.27 (W) x 23.78 (H)	3.121 (W) x 0.936 (H)
5x7 Standard Character	3.225 (W) x 4.875 (H)	0.126 (W) x 0.191 (H)
6x8 Character Matrix	3.90 (W) x 5.60 (H)	0.154 (W) x 0.220 (H)
Pixel Size	0.300 (W) x 0.325 (H)	0.012 (W) x 0.013 (H)
Pixel Pitch	0.325 (W) x 0.350 (H)	0.013 (W) x 0.014 (H)
Depth with Keypad, with Connectors	21.1 (KR Series)	0.831 (KR Series)
Depth with CFA-RS232 mounted	34.2 (KT Series)	1.346 (KT Series)
Keystroke Travel (approximate)	~2.4	~0.09
Weight (typical)	62 grams (KR Series) 57 grams (KT Series)	2.19 ounces (KR Series) 2.01 ounces (KT Series)

3.2. Optical Characteristics CFA735-TFK-Kx

Item	Symbol	Condition	Min	Typ	Max	Direction
Viewing Angle (12 o'clock is the preferred direction for this module)	θ	CR≧2	45°	—	—	above, 12 o'clock
	θ	CR≧2	35°	—	—	below, 6 o'clock
	θ	CR≧2	40°	—	—	right, 3 o'clock
	θ	CR≧2	40°	—	—	left, 9 o'clock
Contrast Ratio	CR	—	4.8	6.8	—	—
Response Time	Trise	Ta=25°C	—	120	180	ms
	Tfall	Ta=25°C	—	260	390	ms

3.3. Optical Characteristics CFA735-TML-Kx

Item	Symbol	Condition	Min	Typ	Max	Direction
Viewing Angle (12 o'clock is the preferred direction for this module)	θ	CR≧2	40°	—	—	above, 12 o'clock
	θ	CR≧2	30°	—	—	below, 6 o'clock
	θ	CR≧2	35°	—	—	right, 3 o'clock
	θ	CR≧2	35°	—	—	left, 9 o'clock
Contrast Ratio	CR	—	3.5	5	—	—
Response Time	T_{rise}	Ta=25°C	—	220	330	ms
	T_{fall}	Ta=25°C	—	140	210	ms

3.4. LED Backlight Information

Backlight control is by DAC (Digital-to-Analog Converter), controlling the constant current LED driver. The LCD and keypad backlights are independently controlled.

The backlights used in the CFA735 are designed for very long life, but their lifetime is finite. To conserve the LED lifetime and reduce power consumption dim or turn off the backlights during periods of inactivity.

Item	Symbol	Condition	Min	Typ	Max	Units
Supply Current	I_{LED}	V=16.8		10.7*2		mA
Supply Voltage	V		15.6	16.8	18.0	v
Reverse Voltage	V_R				5	v
Chromaticity	x		0.27	0.29	0.31	
	y		0.29	0.31	0.33	
Luminance			1080	1350		Cd/m^2
LED Lifetime				50K		hours

4. Electrical Specifications

4.1. System Block Diagram –KR Series

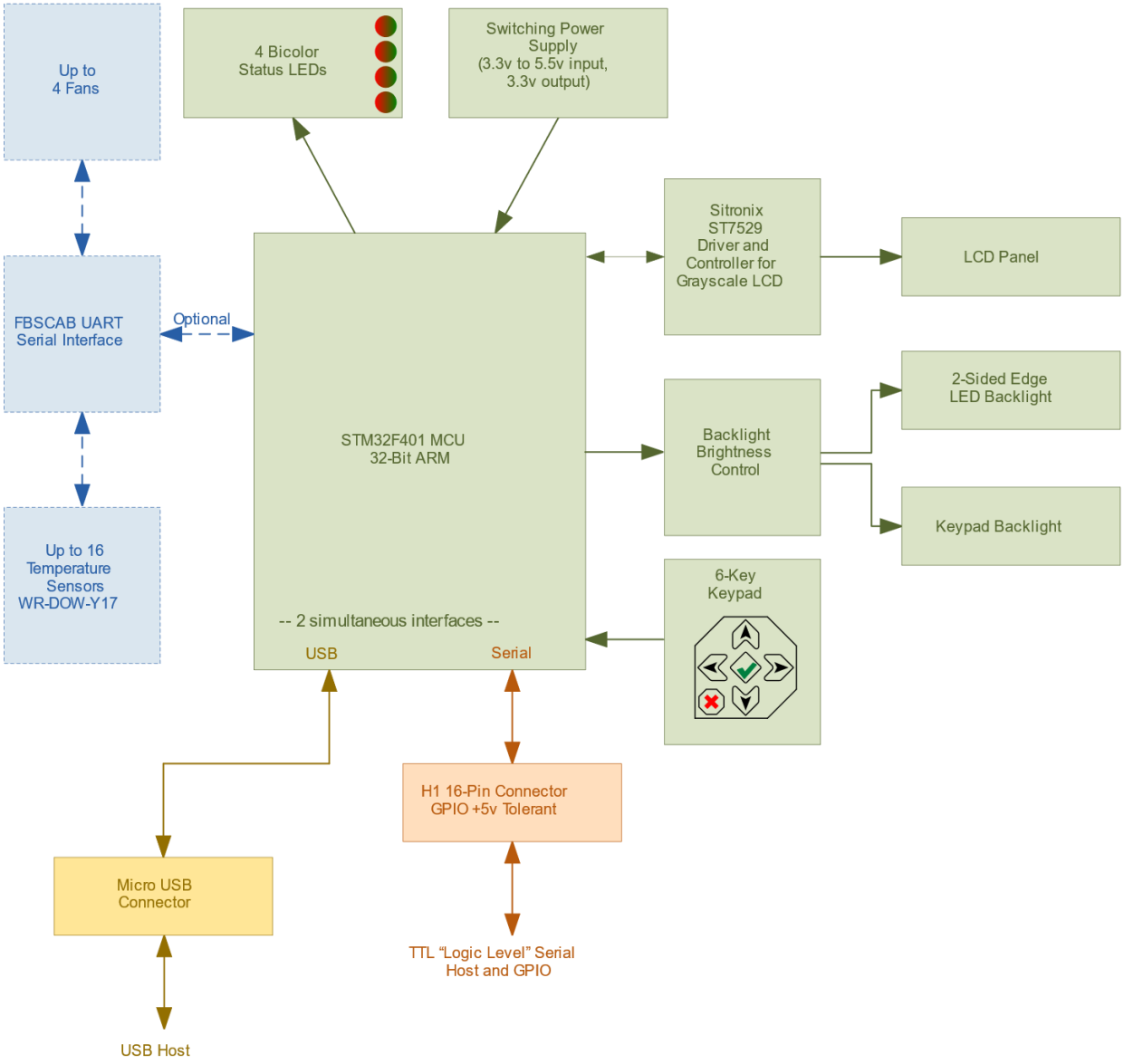


Figure 3. System Block Diagram – CFA735-xxx-KR

4.2. System Block Diagram –KT Series

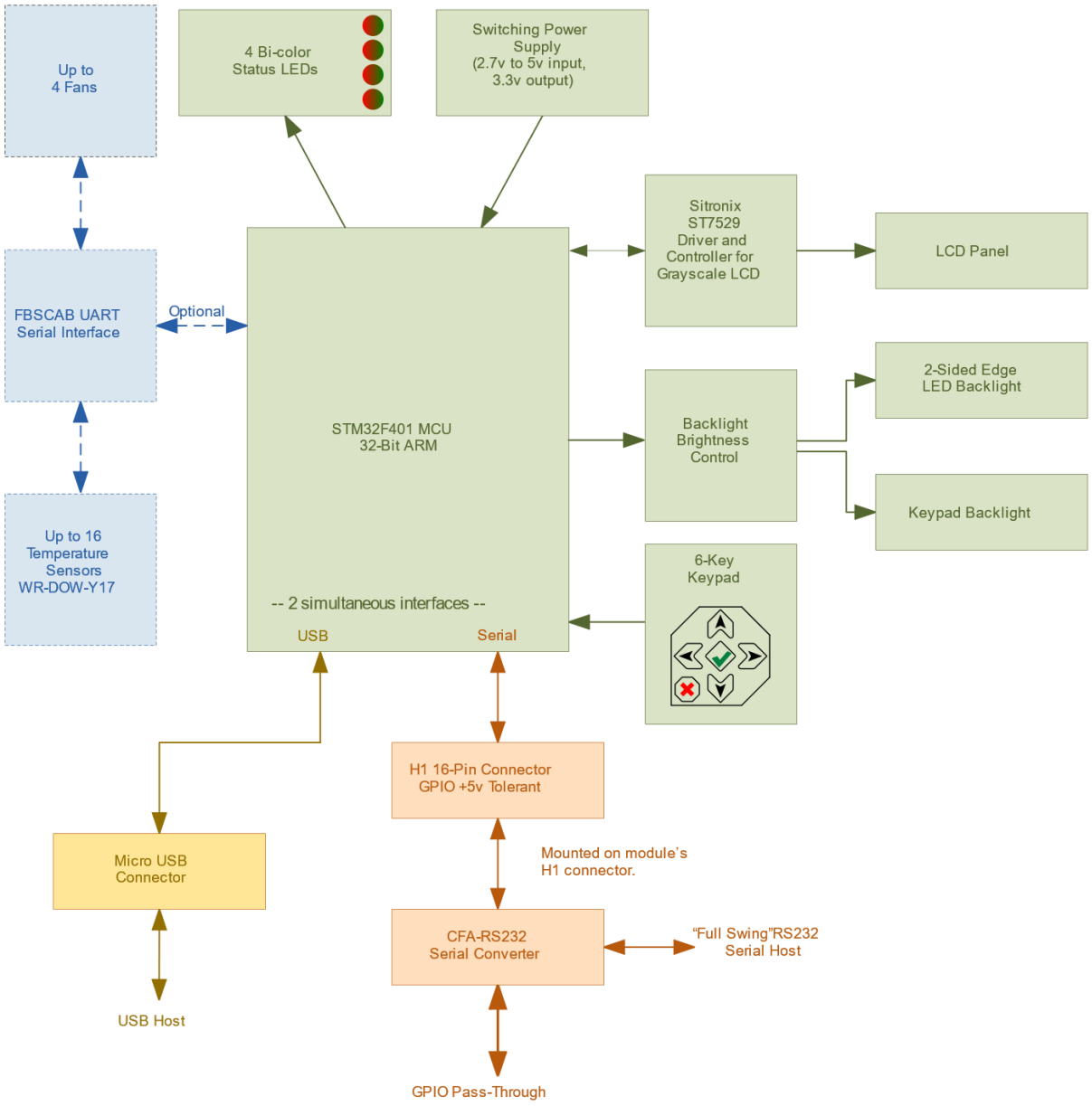


Figure 4. System Block Diagram – CFA735-xxx-KT

5. Supply Voltages and Current

5.1. 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}	-0.5v	+4.0v
Power Supply Voltage			
KT Series: Input and Output Pins for CFA-RS232 Serial			
CFA-RS232 Input Pin	VRX	-25v	+25v
CFA-RS232 Output Pin	VTX	-13v	+13v
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.			

5.2. GPIO Current Limits

Typical GPIO Current Limits	Specification
Sink	8 mA
Source	8 mA

5.3. Logic Level GPIO +5V Tolerant Pins

DC Characteristics	Symbol	Minimum	Maximum
GPIO Input High Voltage	V_{IH}	0.42*(V_{DD}-2v) +1v If V_{DD}=+3.3v =+1.55v	+5.5v
GPIO Input Low Voltage	V_{IL}	-0.3v	0.32*(V_{DD}-2v) +0.75v If V_{DD}=+3.3v =+1.17v
GPIO Output High Voltage	V_{OH}	+2.4v	+3.3v
GPIO Output Low Voltage	V_{OL}	+0.4v	+1.3v

5.4. 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 CFA-FBSCAB is attached to the module.

CFA735-TFK-Kx  **(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	-	-	45 mA	35 mA
X	X	-	150 mA	215 mA
X	-	X	180 mA	125 mA
X	X	X	355 mA	235 mA

CFA735-TML-Kx  **(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	-	-	45 mA	35 mA
X	X	-	150 mA	215 mA
X	-	X	180 mA	125 mA
X	X	X	355 mA	235 mA

6. Connection Information

6.1. Location of Connectors – KR Series

The module has three connectors on the back of the PCB: H1, USB, and CFA-FBSCAB.

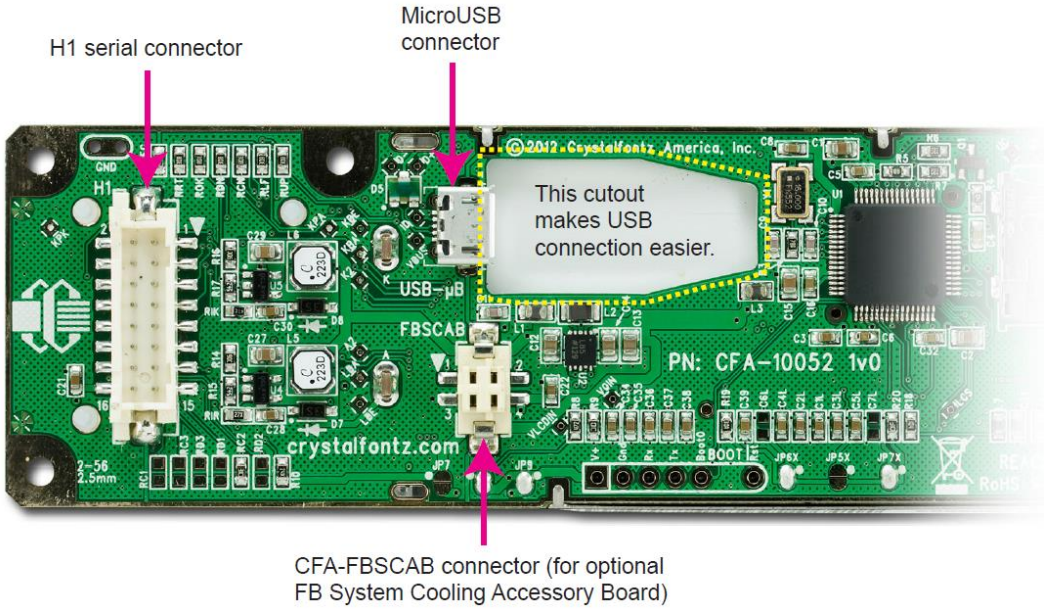


Figure 5. Location of Connectors – KR Series

6.2. Location of Connectors – KT Series

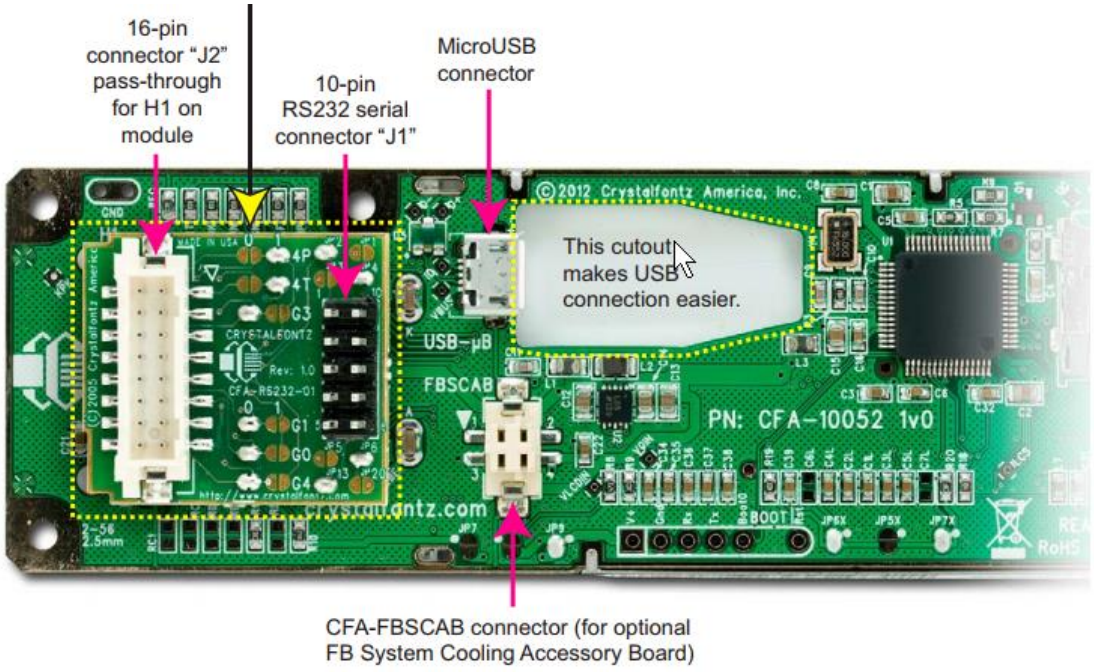


Figure 6. Location of Connectors – KT Series

NOTE: The PCB pads labeled BOOT and the EXPANSION pads on both the KR and KT series are reserved for factory testing and programming.

6.3. USB Connector

The USB connector is a micro USB 5-pin (F) B type. The module may be connected to one host using a USB interface while at the same time using a serial interface to a second host. For details, see Connection 5V Power through USB.

6.4. CFA-FBSCAB Connector

The CFA735 has an optional system cooling accessory board CFA-FBSCAB. Fans and up to 16 temperature sensors may be connected to the CFA-FBSCAB. For more information, please see Connect Optional WR-DOW-Y17 Temperature Sensors to CFA-FBSCAB.

6.5. Use WR-EXT-Y37 Cable

Use the WR-EXT-Y37 cable (~18 inches) to connect the CFA-FBSCAB to the CFA735.

6.6. Make a Cable

To make a custom cable, typical connector parts are listed below, manufactured by Hirose and most are available through Digi-Key:

Connection at CFA-FBSCAB

- Socket housing on cable: Hirose DF11-4DS-2C.
- Socket crimp terminal in housing: Hirose DF11-2428SC.
- For reference, mating header on CFA-FBSCAB: Hirose DF11GZ-4DP-2V(20).

Connection at CFA735

- In-line plug on cable: Hirose DF11-4DEP-2C.
- Crimp terminal pin in housing: Hirose DF11-EP2428PC.
- For reference, mating receptacle connector on CFA735: Hirose DF11Z-4DS-2V(20).

Pre-crimped wires are also available from Digi-Key. For example these 12", 24ga, pin-to-socket in blue: Hirose H3ABT-10112-L4-ND.

6.7. H1 Connector for Serial Interface and GPIO/ATX Functionality

This 16-pin connector can be used for TTL “logic level, inverted” serial connection to one host while at the same time using a USB interface to a second host.

The following parts may be used to make a cable to connect to the modules’ H1 connector:

- 16-position housing: Hirose DF11-16DS-2C / Digi-Key H2025-ND.
- Crimping contact (tape & reel): Hirose DF11-2428SCF / Digi-Key H1504TR-ND.
- Crimping contact (loose): Hirose DF11-2428SC / Digi-Key H1504-ND.
- Pre-terminated interconnect wire: Hirose / Digi-Key H3BBT-10112-B4-ND (typical).

6.8. H1 Connector for CFA-RS232, “Full Swing” RS232 Serial Interface

A CFA-RS232 level translator is mounted to the CFA735-xxx-KT’s H1 connector. For more information, please see CFA-RS232 Level Translator.

6.9. Connecting 5v Power Through USB

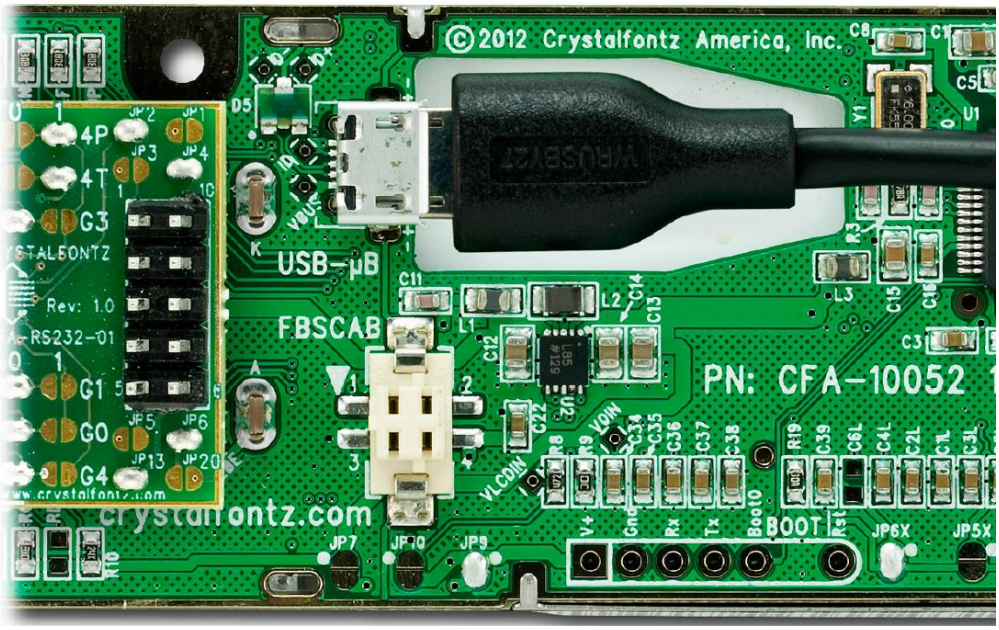


Figure 7. Connecting 5v Power Through USB

IMPORTANT: Keep the Micro-B USB cable connector parallel to the CFA735-xxx-KR when plugging or unplugging the cable. Do not lift or pull up on the cable. Too much pressure may permanently damage the CFA735's Micro-B USB connector.

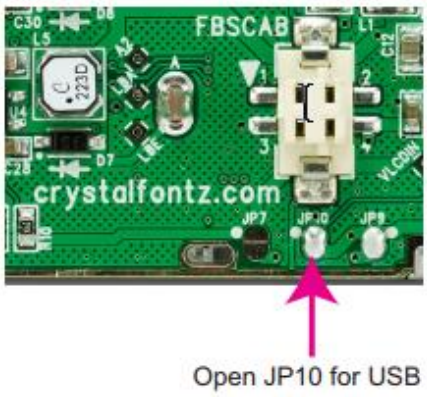
By using the Micro-B USB connector, the CFA735 requires only this one connection to the host for both data communications and power supply.

Figure 8. USB Connection Detail

6.10. When Using USB Interface While Supplying Power Through H1

6.11. KR Series

JP10 on the CFA735-xxx-KR is closed by factory default. To use USB interface while supplying power through H1, **open JP10** to prevent back-powering the USB.



6.12. KT Series

JP10 on the CFA735-xxx-KT is open by factory default. To use USB interface while supplying power through H1, **leave JP10 open** to prevent back-powering the USB.

7. ATX Power Supply Control

7.1. Introduction

ATX power supply control functionality allows the buttons on the CFA735 to replace the power and reset button in a system, simplifying front panel design. This ATX power supply control functionality can be accomplished with the optional WR-PWR-Y25 or WR-PWR-Y38 cables, or equivalent 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. See command 34 (0x22): Set or Set and Configure GPIO Pins.

GPIO[1] ATX Host Power Sense

Since the CFA735 must act differently depending on whether the host's power supply is on or off, 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, 5kΩ 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 Power Supply Control	Pins on CFA-RS232, J1 Connector
$V_{SB}^{(+5v)}$	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

7.2. ATX Connection with WR-PWR-Y25 or WR-PWR-Y38 Cable

KR Series

The illustration below shows how the Crystalfontz WR-PWR-Y25 or WR-PWR-Y38 ATX cables connect to the CFA735's connector H1 and your system's host and ATX Power Supply:

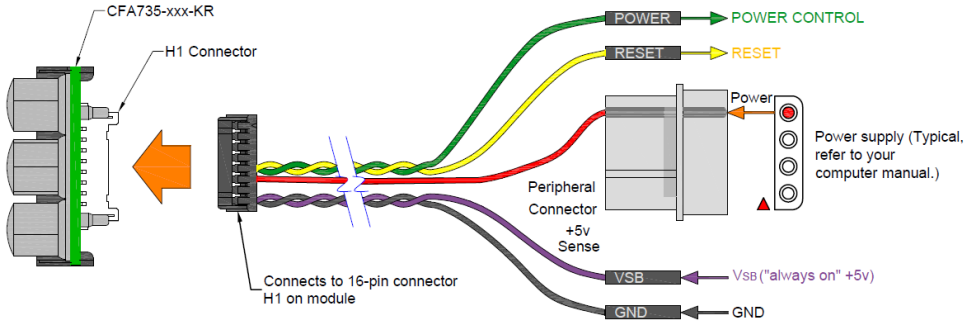


Figure 9. CFA735-xxx-KR ATX Connection to H1 with WR-PWR-Y25 or WR-PWR-Y38 Cable

KT Series

The illustration below shows how the Crystalfontz WR-PWR-Y25 or WR-PWR-Y38 ATX cables connect to the CFA735's connector H1 and your system's host and ATX power supply:

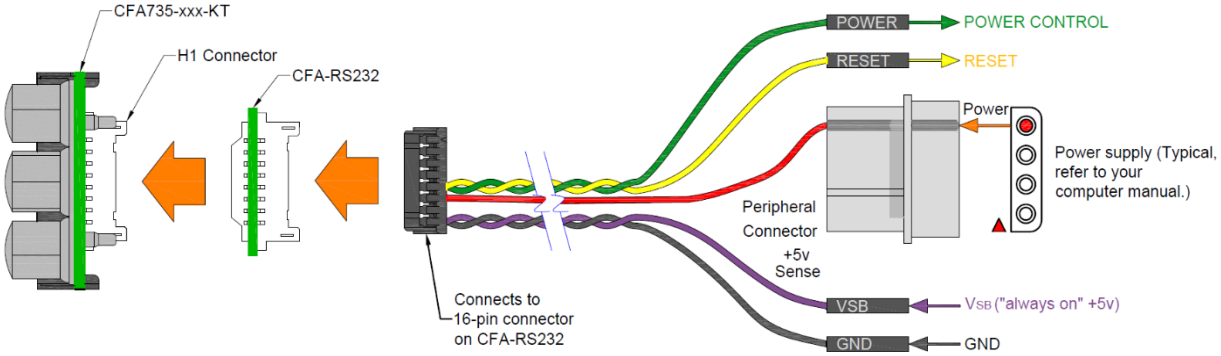


Figure 10. CFA735-xxx-KT ATX Connection to H1 with WR-PWR-Y25 or WR-PWR-Y38 Cable

7.3. KR Series: TTL "Logic Level, Inverted" Serial H1 Pin Assignments

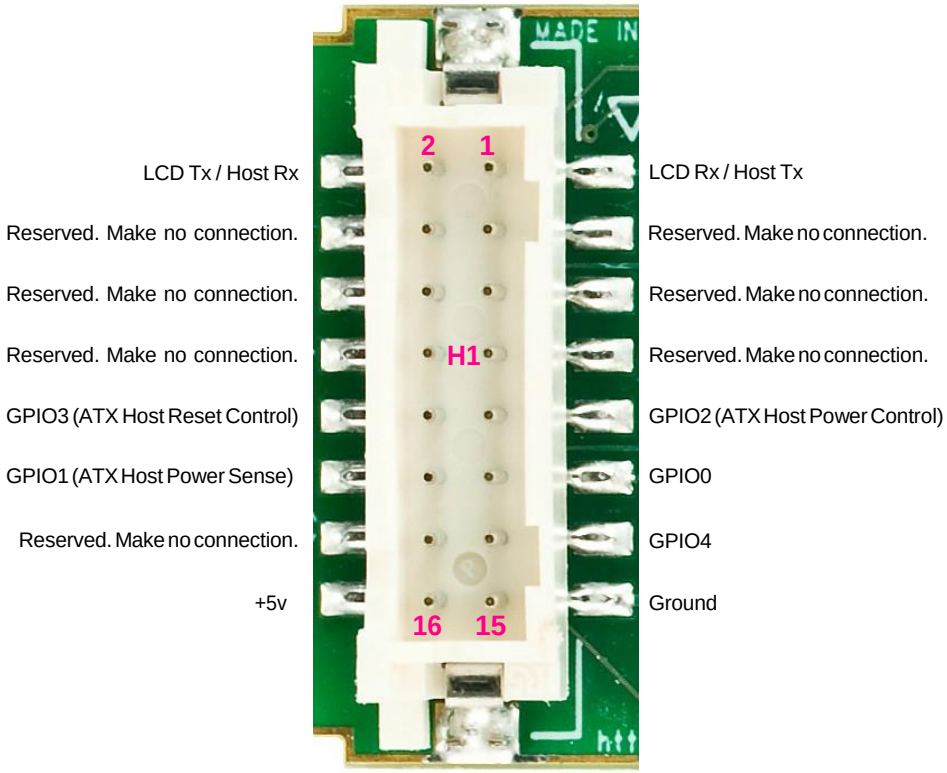


Figure 11. Pin Assignments on CFA735-xxx-KR's H1 Connector (Includes GPIOs)

7.4. KT Series: CFA-RS232 Level Translator for "Full Swing" RS232 Serial

The "Full Swing" CFA-RS232 serial interface on the CFA735-xxx-KT consists of two parts:

- CFA735-xxx-KR Serial and USB LCD Module
- CFA-RS232 Level Translator

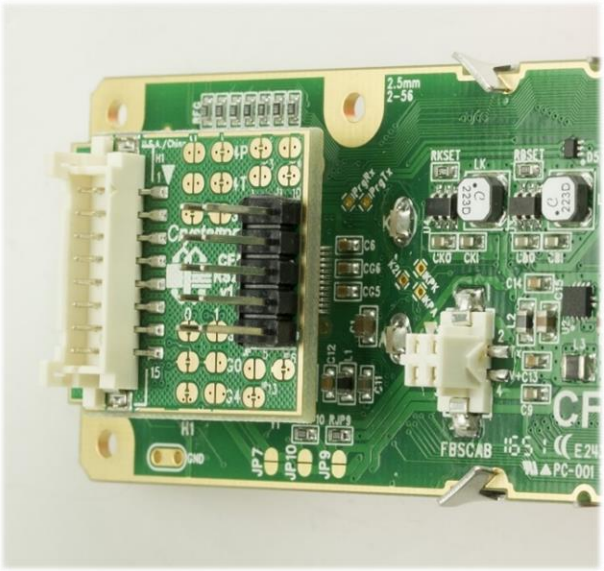


Figure 12. Angled View of CFA-RS232 Level Translator Mounted on CFA735-xxx-KR

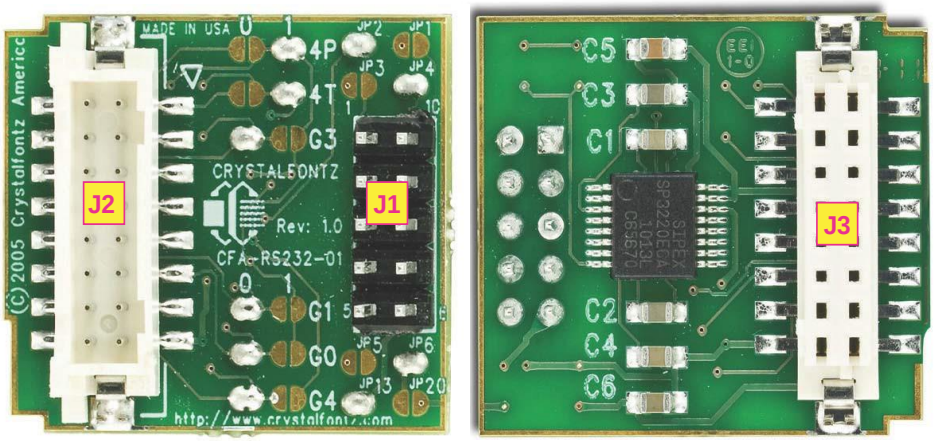
Figure 13. Side View of CFA-RS232 Level Translator Mounted on CFA735-xxx-KR

The CFA-RS232 Level Translator is a small printed circuit assembly mounted on the CFA735-xxx-KR module. It has a 16-pin female connector, J3, that mates with the male 16-pin connector, H1, on the back of the CFA735-xxx-KT module. The CFA-RS232 level translator converts the 0v to +5v (logic level, inverted), Rx and Tx signals from the module's microcontroller to RS232 levels.

7.5. Connectors on the CFA-RS232 Level Translator

The top side of the CFA-RS232 has the Crystalfontz logo silk-screened and has two male connectors. The bottom side of the CFA-RS232 does not have the Crystalfontz logo and has one female connector.

The J1 and J2 connectors are on the top side of the mounted CFA-RS232, facing away from the module. The J3 connector is on the bottom of the mounted CFA-RS232, facing toward the module.



Top Bottom (connects to H1 on CFA735-xxx-KR)

Figure 14. CFA-RS232 Top and Bottom View of Connectors

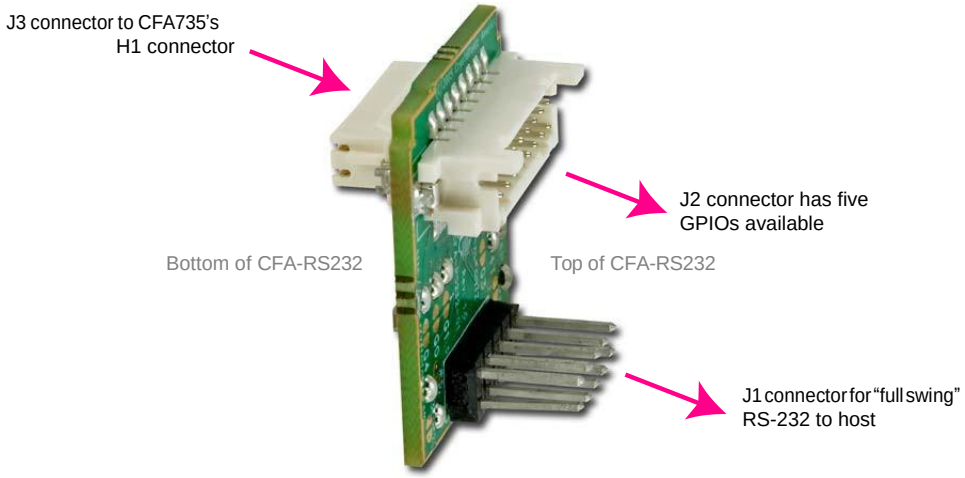


Figure 15. CFA-RS232 Side View of Connectors

J1 is the male 10-pin (0.1" center) RS232 host communication connector on the top side. For pin assignments, please see CFA-RS232 J1 Connector Pin Assignments.

J2 is the male 16-pin 2mm "pass through" connector on the top side, passing through to the J3 female 16-pin 2mm connector on the bottom side of the board. For pin assignments, please see CFA-RS232 J2 Connector Pin Assignments (Includes GPIO Connections).

J3 is the female 16-pin 2mm connector on the bottom side that mates with H1 male 16-pin 2mm connector on the CFA735 serial module.

CFA-RS232 J1 Connector Pin Assignments (Default and Alternate)

The pin order of the motherboard's header will determine if the CFA735-xxx-KT 's pin assignments (“Full Swing” RS232 Serial) needs to be “Default” or “Alternate”, as described below.

NOTE: The WR-232-Y22 cable, when connected to the J1 of the CFA-RS232 Level Translator, provides two connectors on its opposite end. The connector a few inches from the end has a “Default” pin assignment and the connector at the very end has an “Alternate” pin assignment. By using the WR-232-Y22 cable, changing jumpers on the CFA-RS232 Level Translator can be avoided.

On the CFA-RS232 Level Translator, the jumpers JP2, JP4, and JP6 are closed by default, selecting the J1 connector “Default RS232 Pin Assignments”. This default pin assignment allows a low-cost ribbon cable, WR-232-Y08, to connect the CFA735-xxx-KT (“Full Swing” RS232 Serial) to a PC's DB9 COM port.

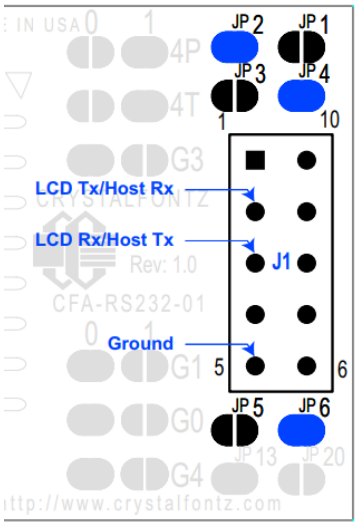


Figure 16. CFA-RS232 J1 Connector Default RS232 Pin Assignments

Opening jumpers JP2, JP4, and JP6 and closing JP1, JP3, and JP5 selects the “Alternate RS232 Pin Assignments”.

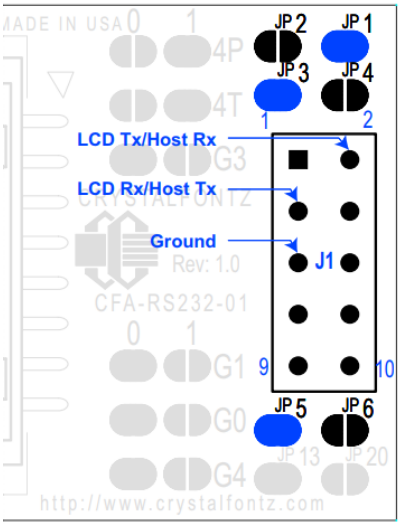


Figure 17. CFA-RS232 J1 Connector Alternate RS232 Pin Assignments

If there is a matching 0.1-inch center, 10-pin RS232 connector on the system's motherboard, then in most cases a simple straight-through ribbon cable such as the Crystalfontz WR-232-Y22 cable can be used to connect from the CFA735-xxx-KT (“Full Swing” RS232 Serial) to the motherboard's 10-pin header.

7.6. CFA-RS232 J2 Connector Pin Assignments

The CFA735-xxx-KT module has five pins that can be used for General Purpose Input or Output (GPIO). The passthrough header J2 on the CFA-RS232 Level Translator. These GPIOs can be accessed directly through J2.

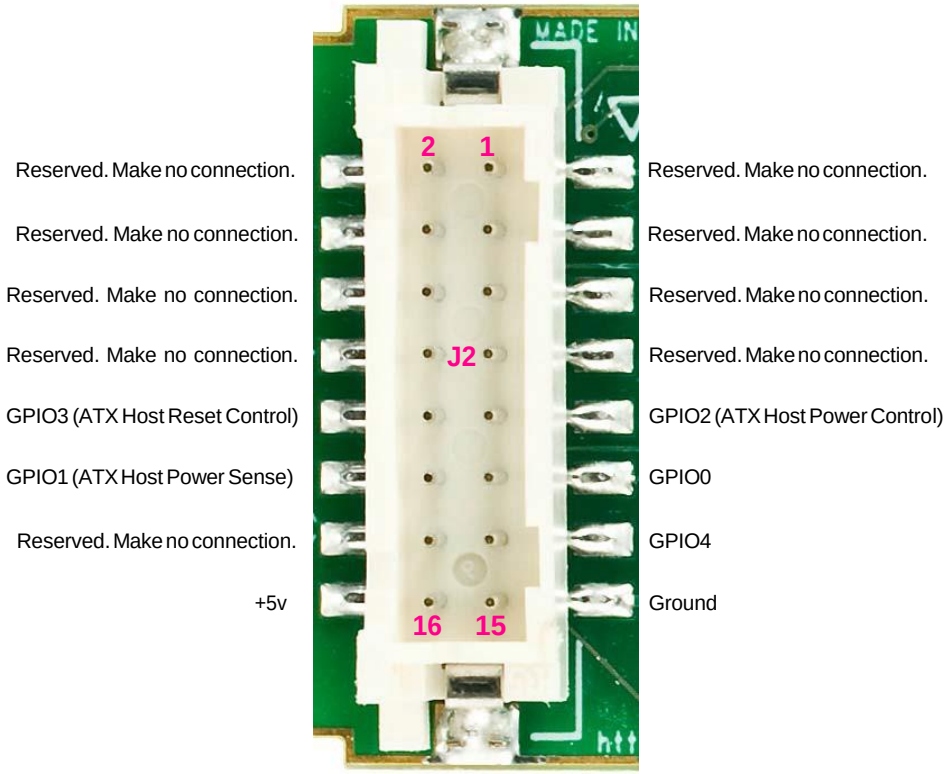


Figure 18. CFA-RS232 J2 Connector Pin Assignments

7.7. How to Connect the Optional FBSCAB

The optional FBSCAB is designed to connect to the CFA735's FBSCAB connector.

The photo below shows the CFA735 connected to the optional FBSCAB using the WR-EXT-Y37 cable:

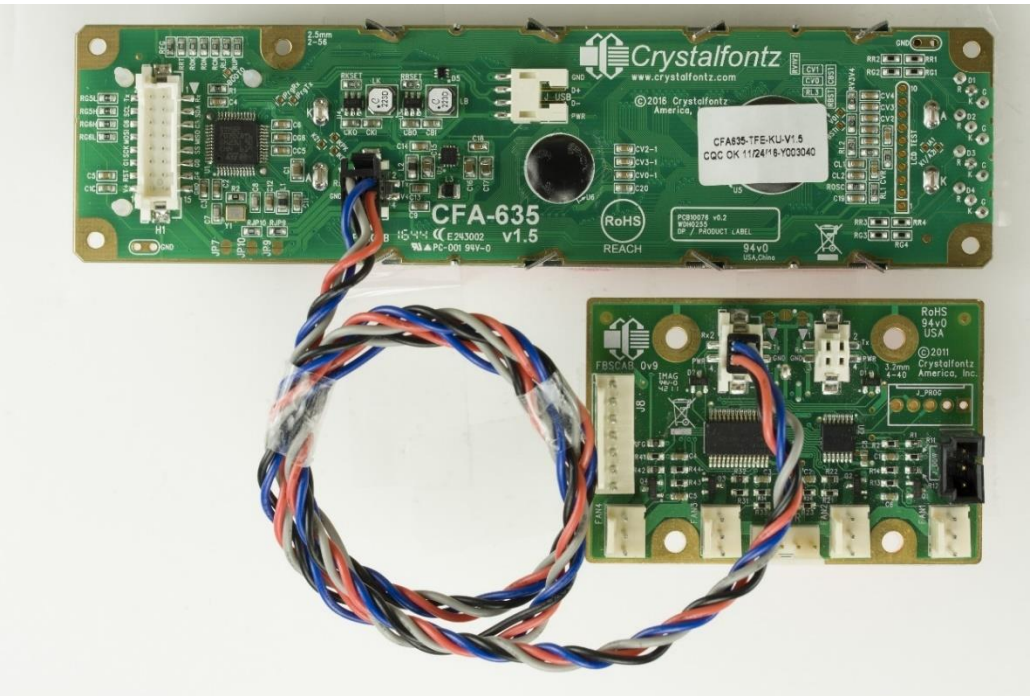


Figure 19. CFA735 Connected to Optional FBSCAB Using WR-EXT-Y37 Cable

The CFA735 does not supply power to the CFA-FBSCAB. The CFA-FBSCAB requires external power, typically supplied by a 4-pin, 3.5-inch floppy drive power connector. For more information, see the CFA-FBSCAB Datasheet.

The CFA735+CFA-FBSCAB provides no ATX functionality through the CFA-FBSCAB; however, ATX functionality is available using the H1 connector on the CFA735.

7.8. Connect Optional WR-DOW-Y17 Temperature Sensors to CFA-FBSCAB

The Crystalfontz WR-DOW-Y17 cable has a DS18B20 Dallas Programmable Resolution 1-Wire (DOW) temperature sensor attached to a "daisy chainable" cable. ("Daisy chain" means several devices connected in a linear series.) Connect the WR-DOW-Y17 to the connector labeled J_DOW on the CFA-FBSCAB. If desired, connect the cable's 3-pin male connector to an additional temperature sensor. Up to 16 temperature sensors can be connected. ("daisy chained").

The DS18B20 on the WR-DOW-Y17 has 0.5°C absolute accuracy. You can make a temperature sensor cable using a DS1822 Dallas Econo 1-Wire Digital Thermometer with +2°C accuracy.

For more information, please see the CFA-FBSCAB Datasheet.

8. Host Communications

8.1. Through USB

The easiest and most common way for the host software to access the USB is through the Crystalfontz virtual COM port (VCP) drivers. A link to VCP drivers download and installation instructions can be found on the Crystalfontz website at USB LCD Drivers. Using these drivers makes it appear to the host system as if there is an additional serial port (the VCP) on the host system when the CFA735 is connected. When communicating over USB, the VCP settings are accepted for compatibility reasons. The virtual COM port settings such as baud rate (speed), stop bits, etc., are ignored as the communications occur as pure USB data.

NOTE: Because there is no difference in communications and commands for serial and USB interfaces, this datasheet section uses the term “CFA735” for the entire CFA735 family of modules.

For CFA635 Customers – the CFA635 requires different firmware for USB interface. The CFA735 supports serial and USB interface with the same firmware. The CFA735 USB driver is not the same as the CFA635 USB driver. If you used the CFA635 USB driver, you will need to replace it with the CFA735 USB driver.

8.2. Through “Logic Level, Inverted” Serial

Modules are shipped with port settings 115200 baud, 8 data bits, no parity, 1 stop bit. Baud rate can be changed to 19200 baud. Please see command 33 (0x21): Set Baud Rate.

8.3. Through “Full Swing” CFA-RS232 Serial

The CFA735 communicates with its host using the CFA-RS232 interface. Modules are shipped with port settings 115200 baud, 8 data bits, no parity, 1 stop bit. Baud rate can be changed to 19200 baud. Please see command 33 (0x21): Set Baud Rate.

8.4. Packet Structure

All communication between the CFA735 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 CFA735 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 LCD module. To reconcile your packets, please ensure that you have received the acknowledgement packet from the packet most recently sent before sending any additional packets to the LCD module. This practice will guarantee that you will not have any dropped packets or missed communication with the LCD module.

All packets have the following structure:

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

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

```
TTcc cccc
|||| ||||--Command, response, error or report code 0-63
||---------Type:
             00 = normal command from host to CFA735
             01 = normal response from CFA735 to host
             10 = normal report from CFA735 to host (not in
                  direct response to a command from the host)
             11 = error response from CFA735 to host (a packet
                  with valid structure but illegal content
                  was received by the CFA735)
```

data_length specifies the number of bytes that will follow in the data field. See individual commands for valid packet lengths.

data is the payload of the packet. Each typeof 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, along with its C source code. Included in the cfTest source is a CRC algorithm and an algorithm that detects packets. The algorithm will automatically resynchronize 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.

8.5. About Handshaking

The nature of CFA735's packets makes it unnecessary to implement traditional hardware or software handshaking.

The host should wait for a corresponding acknowledge packet from the CFA735 before sending the next command packet. The CFA735 will respond to all packets within 250 mS. The host software should stop waiting and retry the packet if the CFA735 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 (e.g., a disconnected cable).

Please note that some operating systems may introduce delays between when the data arrives at the physical port from the CFA735 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 CFA735 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 CFA735. For any modern PC using reasonably efficient software, this requirement will not pose a challenge.

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

8.6. Report Codes

The CFA735 can be configured to report three items. The CFA735 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, and (3) 0x82: Temperature Sensor Report. Details are below.

0x80: Key Activity

If a key is pressed or released, the CFA735 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] = 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 four fans connected to CFA735+FBSCAB is configured to report its speed information to the host, the CFA735 will send Fan Speed Reports for each selected fan every 1/2 second, please see command 16 (0x10): Set Up Fan Reporting. Fan reporting is off when the module is powered on.

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

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 = 0;
   int 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 PPR
      int pulses_per_revolution = 2;
      int 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 CFA735+FBSCAB will send Temperature Sensor Reports for each selected sensor every second, please see the command 19 (0x13): Set Up Temperature Reporting. Temperature reporting is off when the module is powered on.

```
type = 0x82
data_length = 4
data[0] = index of the temperature sensor being reported:
  0 = temperature sensor 1
  1 = temperature sensor 2
  . . .
  15 = temperature sensor 16
data[1] = MSB of Temperature_Sensor_Counts
data[2] = LSB of Temperature_Sensor_Counts
data[3] = DOW_crc_status
```

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

```
void OnReceivedTempReport(COMMAND_PACKET *packet, char *output)
{
   //First check the DOW CRC return code from the CFA635
   if(packet->data[3]==0)
      strcpy(output,"BAD CRC");
   else
   {
      double degc = (*(short*)&(packet->data[1]))/16.0;
      double degf = (degc*9.0)/5.0+32.0;
      sprintf(output,"%9.4f°C =%9.4f°F", degc, degf);
   }
}
```

8.7. Command Codes

Below is a list of valid commands for the CFA735. 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 CFA735 returns the Ping Command to the host.

Request packet format:

```
type: 0x00 = 0₁₀
data_length = 0 to 255
data[] = up to 255 bytes of arbitrary data
```

Successful return packet format:

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

1 (0x01): Get Hardware & Firmware Version

The CFA735 returns the hardware and firmware revision information to the host.

Request packet format:

```
type = 0x01 = 1₁₀
data_length = 0 or 1
data[0] = module information to return:
   0 = Hardware and Firmware Revision
   1 = Module Serial Number
```

Successful return packet format:

```
type = 0x40 | 0x01 = 0x41 = 65₁₀
data_length = 16
data[] = "CFA735:vX.X,sY.Y"
   vX.X is the hardware revision
   sY.Y is the firmware revision

or for data[0] = 1 from the host
type = 0x40 | 0x01 = 0x41 = 65₁₀
data_length = 20
data[] = Module Serial Number
```

2 (0x02): Write User Flash Area

The CFA735 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.

Request packet format:

```
type: 0x02 = 2₁₀
data_length = 16
data[] = 16 bytes of arbitrary user data to be stored in the CFA735's
nonvolatile memory
```

Successful return packet format:

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

3 (0x03): Read User Flash Area

This command reads the User Flash Area and returns the data to the host.

Request packet format:

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

Successful return packet format:

```
type: 0x40 | 0x03 = 0x43 = 67₁₀
data_length = 16
data[] = 16 bytes user data recalled from the CFA735's nonvolatile memory
```

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

The CFA735 loads its power-up configuration from nonvolatile memory when power is applied. The CFA735 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 that 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 Functionality).
- Baud rate (Command 33 (0x21): Set Baud Rate).
- GPIO settings (Command 34 (0x22): Set or Set and Configure GPIO Pins).
- The front panel LED/SPO settings (Command 34 (0x22): Set or Set and Configure GPIO Pins).

Fan or temperature reporting, the fan fail-safe or host watchdog cannot be stored. The host software should enable these items once the system is initialized and it is ready to receive the data.

Request packet format:

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

Successful return packet format:

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

5 (0x05): Reset Functions

Depending on the parameters provided, this command provides five reset functions: (1) Reload Boot Settings, (2) Reset Host, (3) Power Off Host, (4) CFA735 Soft Reboot, or (5) CFA735 Soft Reboot and Settings Reset.

Note: When using both the USB and serial interface simultaneously, performing a reset from one interface may impact the other interface.

- **Reload Boot Settings**

  Rebooting the CFA735 may be useful when testing the boot configuration. It may also be useful to re-enumerate connected devices on the 1-Wire bus (e.g., WR-DOW-Y17 temperature sensors). The CFA735 will return the acknowledge packet immediately; then reboot itself.

  At boot-up, there is up to a 500-millisecond delay between power-on and turning on the fans. By default, all fans are set to “on” at 100%. If not using a fan, set power to 0% (command 17 (0x11): Set Fan Power), and save this setting as the default boot state (command 4 (0x04): Store Current State as Boot State).

  Request packet format:

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

  The CFA735 will return the acknowledge packet immediately, then reload it settings. The module will respond to new commands immediately. This may reconfigure the interface to its boot state. Part of this delay is the intentional staggered sequencing of turning on power to the fans. If fans are not used, the boot process can be sped up by setting the fan power to 0, command 17 (0x11): Set Fan Power (FBSCAB Required) and saving this as the default boot state, command 4 (0x04): Store Current State as Boot State.

- **Reset Host**

  This command instructs the CFA735+WR-PWR-Y25 cable 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 allows certain host operating system configuration changes to complete. The ability to turn off the host's power under software control may be useful in systems that do not have Advanced Configuration and Power Interface (ACPI) compatible BIOS.

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

  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:

  Request packet format:

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

The CFA735 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 fans are not used, the boot process can be sped up by setting the fan power to 0, command 17 (0x11): Set Fan Power (FBSCAB Required) 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.

- **Power Off Host**

  This command instructs the CFA735+WR-PWR-Y25 cable to simulate a power-on restart of itself, reset the host, or turn the host's power off. The ability to reset the host may be useful to allow certain host operating system configuration changes to complete. The ability to turn the host's power off under software control may be useful in systems that do not have ACPI compatible BIOS.

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

  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.

  Request packet format:

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

The CFA735 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 fans are not used, the boot process can be sped up by setting the fan power to 0, command 17 (0x11): Set Fan Power (FBSCAB Required) 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.

- **CFA735 Soft Reboot**

  Performs a soft reboot of the CFA735 module. If used as a USB device, CFA735 soft reboot will cause the module to disconnect and then reconnect (re-enumerate).

  NOTE: The CFA735 will return the acknowledge packet immediately, then reboot itself. The module will not respond to new command packets for up to 3 seconds.

  Request packet format:

  ```
  type = 0x05 = 5₁₀
  data_length =  3
  data[0] = 8
  data[1] = 25
  data[2] = 48
  ```

- **CFA735 Soft Reboot and Settings Reset**

  Resets the system boot state to that of an “un-customized” CFA735 and then performs a CFA735 soft reboot. If used as a USB device, CFA735 soft reboot will cause the module to disconnect and then reconnect (re-enumerate).

  NOTE: The CFA735 will return the acknowledge packet immediately, then reboot itself. The module will not respond to new command packets for up to 3 seconds.

  Request packet format:

  ```
  type = 0x05 = 5₁₀
  data_length = 3
  data[0] = 10
  data[1] = 8
  data[2] = 98
  ```

In any of the above cases, successful return packet format:

```
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.

Request packet format:

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

Successful return packet format:

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

9 (0x09): Set LCD Special Character Data

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

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

Request packet format:

```
type: 0x09 = 9₁₀
data_length = 9
data[0] = index of special character to modify (0-7 valid)
data[1-8] = bitmap of the new font for this character
  data[1]is at the top of the cell.
  data[8]is at the bottom of the cell.
  any value is valid between 0 and 63.
  the msb is at the left of the character cell
  lsb is at the right of the character cell.
```

Successful return packet format:

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

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.

Request packet format:

```
type: 0x0A = 10₁₀
data_length = 1
data[0] = address code of desired data
data[1] = address code native to the LCD controller:
   0x40 ( 64) to 0x7F (127) for CGRAM
   0x80 (128) to 0x93 (147) for DDRAM, line 0
   0xA0 (160) to 0xB3 (179) for DDRAM, line 1
   0xC0 (192) to 0xD3 (211) for DDRAM, line 2
   0xE0 (224) to 0xF3 (243) for DDRAM, line 3
```

Successful return packet format:

```
type: 0x40 | 0x0A = 0x4A = 74₁₀
data_length = 9
data[0] requested address code.
data[1-8] requested 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 CFA735's LCD screen. For the cursor to be visible, also send a command 12 (0x0C): Set LCD Cursor Style.

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

Request packet format:

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

Successful return packet format:

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

12 (0x0C): Set LCD Cursor Style

This command selects among four hardware generated cursor options.

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

Request packet format:

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

Successful return packet format:

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

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 CFA735.

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

Request packet format:

```
type: 0x0D = 13₁₀
data_length = 1
data[0] = contrast setting (0-255 valid)
   0-65 = very light
   66 = light
   85 = about right
   125 = dark
   126-254 = very dark (may be useful at cold temperatures)
```

Successful return packet format:

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

14 (0x0E): Set Display & Keypad Backlight

This command sets the brightness of the LCD and keypad backlights. If one byte is supplied, both the keypad and LCD backlights are set to that brightness (compatible to the CFA735).

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

Request packet format:

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

Successful return packet format:

```
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.

Request packet format:

```
type: 0x0E = 14₁₀
data_length = 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
```

Successful return packet format:

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

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

This command configures the CFA735+FBSCAB to report the fan speed information to the host every 500 mS. Fan reporting is off when the module is powered on.

Request packet format:

```
type = 0x10 = 16₁₀
data_length = 1
data[0] = bitmask indicating which fans are enabled to
   report (0 to 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
```

Successful return packet format:

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

If `data[0]` is not 0, then the CFA735+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, see Fan Connections for a detailed description.

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

This command will configure the power for the fan connectors. The fan power setting is one of the items stored by the command 4 (0x04): Store Current State as Boot State.

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

Request packet format:

```
type = 0x11 = 17₁₀
data_length = 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)
```

Successful return packet format:

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

18 (0x12): Read WR-DOW-Y17 Temperature Sensors (FBSCAB Required)

When power is applied to the CFA735+FBSCAB+WR-DOW-Y17, 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.

Request packet format:

```
type: 0x12 = 18(10)
data_length =  1
data[0] = device index (0-15 valid)
```

Successful return packet format:

```
type: 0x40 | 0x12 = 0x52 = 82(10)
data_length = 9
data[0] = device index (0-15 valid)
data[1-8] = ROM ID of the device
```

If `data[1]` is 0x22 (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 CFA735+FBSCAB+WR-DOW-Y17 to report the temperature information to the host every second. Temperature reporting is off when the module is powered on.

Any sensor enabled must have been detected as a 0x28 (WR-DOW-Y17 temperature sensor) during DOW enumeration. This can be verified by using the command 18 (0x12): Read DOW Device Information.

Request packet format:

```
type: 0x13 = 19(10)
data_length = 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 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 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
```

Successful return packet format:

```
type: 0x40 | 0x13 = 0x53 = 83(10)
data_length = 0
```

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

The CFA735+FBSCAB can function as an CFA-RS232 to Dallas 1-Wire bridge. This command specifies 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
```

Request packet format:

```
type: 0x14 = 20(10)
data_length = 2 to 16
data[0] = device_index (0-32 valid)
data[1] = number_of_bytes_to_read (0-14 valid)
data[2-15] = data_to_be_written[data_length-2]
```

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

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

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

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

Successful return packet format:

```
type: 0x40 | 0x14 = 0x54 = 84(10)
data_length = 2 to 16
data[0] = device index (0-31 valid)
data[data_length-2] = data read from the 1-Wire bus
data[data_length-1] = 1-Wire CRC
```

23 (0x17): Configure Key Reporting

By default, the CFA735 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.

Bitmask options:

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

Request packet format:

```
type: 0x17 = 23(10)
data_length = 2
data[0] = press mask
data[1] = release mask (0 to 63 valid)
```

Successful return packet format:

```
type: 0x40 | 0x17 = 0x57 = 87(10)
data_length = 0
```

24 (0x18): Read Keypad, Polled Mode

In some situations, it may be convenient for the host to poll the CFA735 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.

Bitmask options:

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

Request packet format:

```
type: 0x18 = 24(10)
data_length = 0
```

Successful return packet format:

```
type: 0x40 | 0x18 = 0x58 = 88(10)
data_length = 3
data[0] = bit mask of keys currently pressed
data[1] = bit mask of keys pressed since the last poll
data[2] = bit mask of keys released since the last poll
```

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

The CFA735+FBSCAB can be used as part of an active cooling system. For instance, the fans in a system can be slowed to reduce noise when a system is idle or when the ambient temperature is low, and sped 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 CFA735'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.

Bitmask options:

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

Request packet format:

```
type = 0x19 = 25(10)
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
```

Successful return packet format:

```
type = 0x40 | 0x19 = 0x59 = 89(10)
data_length = 0
```

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

The combination of the CFA735+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 CFA735 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 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 sets a variable-length delay after the fan has been turned on before the CFA735 will recognize transitions on the tachometer line. The delay is specified in counts, each count being nominally 552.5 µS 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.

Typical:

(1) start at a delay count of 50 or 100

(2) reduce it until the problem reappears

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

Request packet format:

```
type = 0x1A = 26(10)
data_length = 4
data[0] = delay count of fan 1 (0 to 100 valid)
data[1] = delay count of fan 2 (0 to 100 valid)
data[2] = delay count of fan 3 (0 to 100 valid)
data[3] = delay count of fan 4 (0 to 100 valid)
```

Successful return packet format:

```
type = 0x40 | 0x1A = 0x5A = 90(10)
data_length = 0
```

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

The combination of the CFA735+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.

Bitmask options:

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

Request packet format:

```
type = 0x1B = 27(10)
data_length = 0
```

Successful return packet format:

```
type = 0x40 | 0x1B = 0x5B = 91(10)
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

The combination of the CFA735 with ATX 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.

The GPIO pins used for ATX control be configured to their default / unused 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 configure a pin to default/unused mode, use Command 34 to set the pin function bit to 0. For example, to set the ATX Sense pin (GPIO1) for ATX use, send the following command:

```
type: 0x33 = 34(10)
data_length: 3
data[0] = 1 (gpio pin number)
data[1] = 0 (output duty)
data[2] = 0 (function bit set to 0)
```

Note: the output duty and the lower 3 drive mode bits are disregarded when the function bit is set to 0.

The RESET (GPIO[3]) and POWER CONTROL (GPIO[2]) lines on the CFA735 are normally high-impedance. Electrically, they appear to be disconnected or floating. When the CFA735 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 CFA735 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 CFA735 will show "RESET", and then the CFA735 will reset itself, showing its boot state as if it had just powered on. Once the pulse has finished, the CFA735 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 CFA735 will show "POWER ON".

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 CFA735 will continue to drive the line for a maximum of 5 additional seconds. During this time, the CFA735 will show "POWER OFF".

Function 4: LCD_OFF_IF_HOST_IS_OFF

If LCD_OFF_IF_HOST_IS_OFF is set, the CFA735 will blank its screen and turn off its backlight to simulate its power being off any time POWER-ON SENSE is low. The CFA735 will still be active (since it is powered by VSB), monitoring the keypad for a power-on keystroke. If +12v remains active (which would not be expected, since the host is “off”), the fans will remain on at their

previous settings. Once POWER-ON SENSE (GPIO[1]) goes high, the CFA735 will reboot as if power had just been applied to it.

Bitmask options:

```
#define AUTO_POLARITY              0x01    //automatically detects polarity for
                                           //reset and power (default)
#define RESET_INVERT               0x02    //reset pin drives high instead of low
                                           //(ignored if AUTO_POLARITY)
#define POWER_INVERT               0x04    //power pin drives high instead of low
                                           //(ignored if AUTO_POLARITY is set)
#define LEDS_FOLLOW_MODULE_LOOK    0x08    // Turn off the LEDs also if the host is
                                           //off (ignored if LCD_OFF_IF_HOST_IS_OFF
                                           //is not set)
#define LCD_OFF_IF_HOST_IS_OFF     0x10    //Turn off LCD and backlight if host is
                                           //off
#define KEYPAD_RESET               0x20    //enabled keypad reset function
#define KEYPAD_POWER_ON            0x40    //enable keypad power on function
#define KEYPAD_POWER_OFF           0x80    //enable keypad power off function
```

Request packet format:

```
type: 0x1C = 28(10)
data_length = 1 or 2
data[0] = bit mask of enabled functions (see above)
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 (default)
data[2] = optional bit mask of extra power functions
   0 = none enabled (default)
   1 = power immediately (module is on whenever power is supplied)
```

Successful return packet format:

```
type: 0x40 | 0x1C = 0x5C = 92(10)
data_length: 0
```

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

Some systems use hardware watchdog timers to ensure that a software or hardware failure does not result in an extended system outage. Once the host system has booted, a system monitor program is started. The system monitor program would enable the watchdog timer on the CFA735 with ATX. If the system monitor program fails to reset the CFA735's watchdog timer, the CFA735 with ATX 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.

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.

Request packet format:

```
type: 0x1D = 29₁₀
data_length = 1
data[0] = enable/timeout
   0 = timeout/watchdog disabled
   1-255 = 1 to 255 seconds until watchdog reset occurs
```

Successful return packet format:

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

30 (0x1E): 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.

Request packet format:

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

Successful return packet format:

```
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 9-16 reporting status (as set by command 19)
data[3] = 0
data[4] = 0
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.

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

Request packet format:

```
type: 0x1F = 31₁₀
data_length = 3 to 22
data[0] = column 0 to 19
data[1] = row 0 to 3
data[2-21] = text to place on the LCD, variable from 1 to 20 characters
```

Successful return packet format:

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

33 (0x21): Set Baud Rate

The CFA735 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 CFA735 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 CFA735 to power up at the new baud rate.

The factory default baud rate is 115200.

Request packet format:

```
type: 0x21 = 33₁₀
data_length = 1
data[0] = baud rate:
   0 = 19200 baud
   1 = 115200 baud
   2 = 9600 baud
```

Successful return packet format:

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

34 (0x22): Set or Set and Configure GPIO Pins

The CFA735 has five pins for user-definable general-purpose input / output (GPIO). See Section 5.3 regarding acceptable input/output voltages. The GPIOs do not have under/over-voltage or over-current protection. These pins are shared with the ATX functions. Be careful when configuring the GPIO if you want to use the ATX at the same time.

The architecture of the CFA735 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 CFA735 continuously polls the GPIOs as inputs at 32 Hz. The present level can be queried by the host software at a lower rate. The CFA735 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 CFA735 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 40kΩ.

Typical GPIO current limits when sinking or sourcing all five GPIO pins simultaneously are 8 mA. If you need more information, please see the ST-Micro STM32F103 datasheet.

REGARDING SETTING AND CONFIGURING GPIO PINS

The GPIO pins may also be used for ATX control through header J8 and temperature sensing through the CFA735'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.

Unlike the CFA635+CFA-FBSCAB, the CFA735+CFA-FBSCAB has no ATX functionality provided through the CFA-FBSCAB. ATX control is available using the H1 connector on the CFA735.

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

Request packet format:

```
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:
   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
   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
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
   101-254 = invalid
data[2] = Pin function select and drive mode (optional, 0-15 valid except
for 6 and 14)
   ---- 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 - - error returned
   |||| |      111: 1=Hi-Z, 0=Slow, Strong Drive Down
   |||| |
   |||| |----- F = Function (only valid for GPIOs, index of 0-4)
   ||||        ========================================================
```

```
|||| 0: Port unused for GPIO. It will take on the default
||||    function such as ATX 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
```

Successful return packet format:

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

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

See command 34 (0x22): Set or Set and Configure GPIO Pins for details on the GPIO architecture.

Request packet format:

```
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
  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
```

Successful return packet format:

```
type = 0x40 | 0x23 = 0x63 = 99₁₀
data_length = 4
data[0] = index of GPIO read
data[1] = GPIO pin state & changes since last poll
  ---- -RFS
  |||| ||||-- S = state at the last reading
  |||| |||--- F = at least one falling edge has
  |||| ||        been detected since the last poll
  |||| ||---- R = at least one rising edge has
  |||| |         been detected since the last poll
  ||||-|----- reserved
data[2] = requested pin level/PWM percentage (0 to 100)
data[3] = pin function select and drive mode.
  ---- FDDD
  |||| ||||-- DDD = Drive Mode
  |||| |      ============================================================
  |||| |      000: 1=Fast, Strong Drive Up, 0=Resistive Pull Down
  |||| |      001: 1=Fast, Strong Drive Up, 0=Fast, Strong Drive Down
  |||| |      010: Hi-Z, use for input
  |||| |      011: 1=Resistive Pull Up,     0=Fast, Strong Drive Down
  |||| |      100: 1=Slow, Strong Drive Up, 0=Hi-Z
  |||| |      101: 1=Slow, Strong Drive Up, 0=Slow, Strong Drive Down
  |||| |      110: reserved
```

```
||||  |      111: 1=Hi-Z,                  0=Slow, Strong Drive Down
||||  |
||||  |----- F = Function
||||         ========================================================
||||         0: Port unused for GPIO. It will take on the default
||||            function such as ATX, DOW or unused. The user is
||||            responsible for setting the drive to the correct
||||            value in order for the default function to work
||||            correctly.
||||         1: Port used for GPIO under user control. The user is
||||            responsible for setting the drive to the correct
||||            value in order for the desired GPIO mode to work
||||            correctly.
||||-------- reserved, will return 0
```

9. Character Generator ROM (CGROM)

To find the code for a given character, add the two numbers that are shown in bold for its row and column. For example, the superscript "9" is in the column labeled “128d” and in the row labeled “9d”. Add 128 + 9 to get 137. When you send a byte with the value of 137 to the display, then a superscript "9" will be shown.

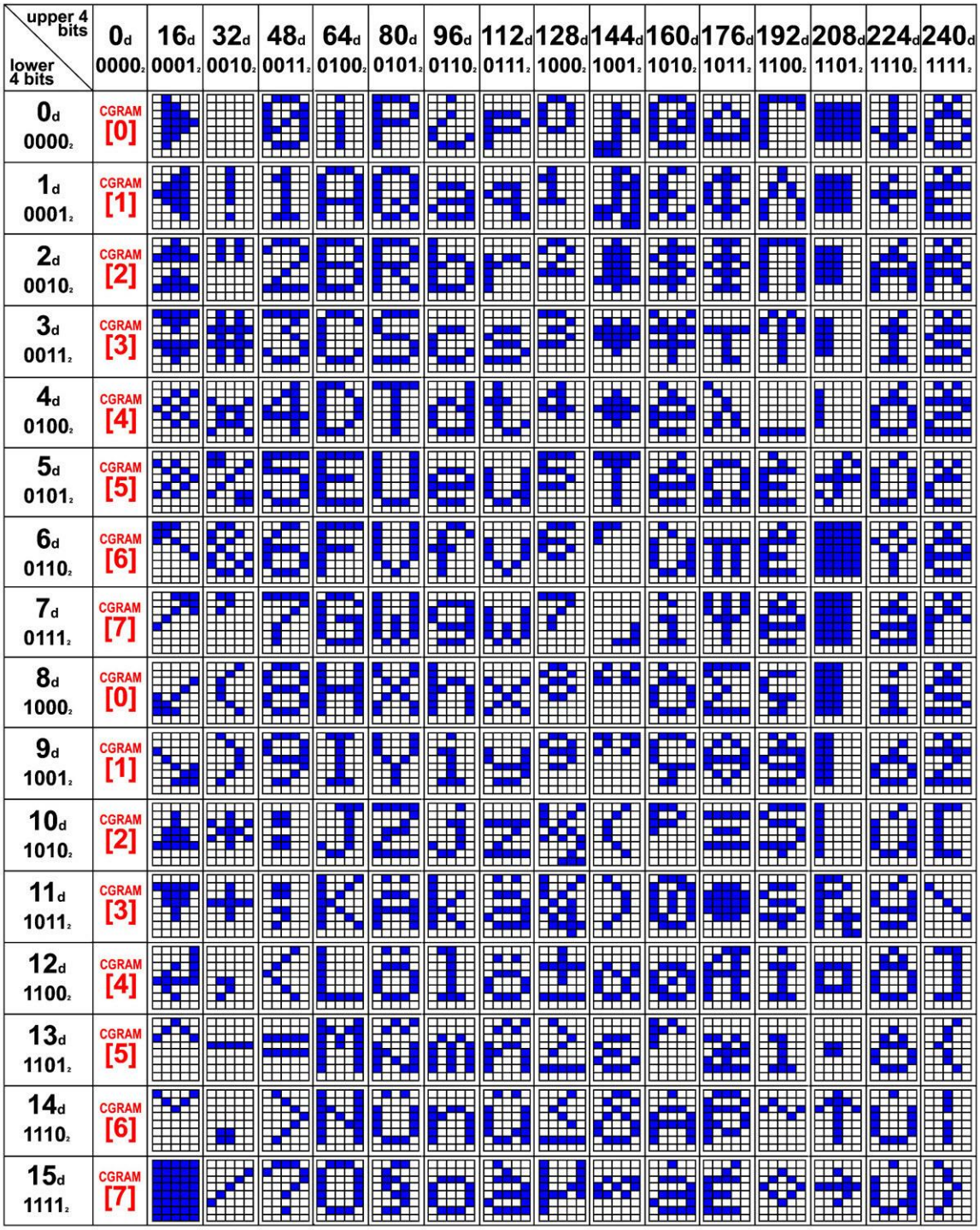


Figure 20. Character Generator ROM (CGROM)

10. LCD Module Reliability and Longevity

Crystalfontz works 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 modules with consistent color are required, 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 LED status lights	50,000 to 100,000 hours	
White, Blue, and Yellow-Green LED Display Keypad Backlights ***NOTE**: We recommend that the backlight of the white LED backlit modules be dimmed or turned off during periods of inactivity to conserve the white LED backlight lifetime.*	Power-On Hours	% of Initial Brightness
	<10,000	>90%
	<50,000	>50%

10.1. Module Longevity (EOL / Replacement Policy)

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

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

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

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

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

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

11. Care and Handling Precautions

For optimum operation of the CFA735 and to prolong its life, please follow the precautions described below.

11.1. ESD (Electrostatic Discharge)

The CFA-RS232, Tx and Rx lines have enhanced ESD protection following industry standard practice, please see Serial ESD Specifications. The USB D+ & D- lines have enhanced ESD protection following industry standard practice, please see USB ESD Characteristics.

The remainder of this 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.

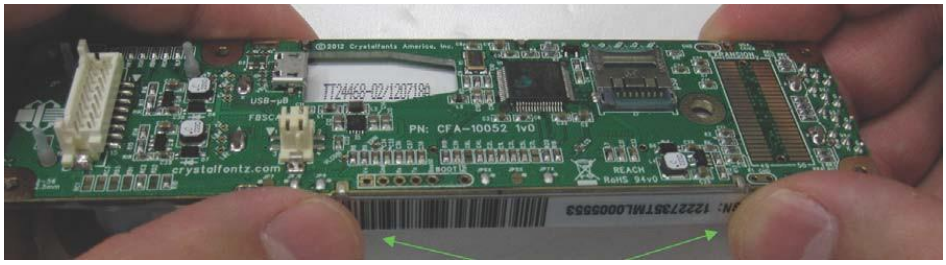
11.2. Design and Mounting

- The exposed surface of the “glass” is actually a polarizer laminated on top of the glass. To protect the soft plastic polarizer from damage, the module ships with a protective film over the polarizer. 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.

11.3. Avoid Damage to Flat Flex Cable

- The CFA735 has a flat flex cable that is covered by the serial number sticker. Force on this sticker may damage the flat flex cable resulting in a loss of function of the display.

- To avoid damaging the FFC, handle the module with care and avoid any pressure along the sticker.



11.4. Avoid Shock, Impact, Torque, or Tension

- Do not expose the CFA735 to strong mechanical shock, impact, torque, or tension.
- Do not drop, toss, bend, or twist the CFA735.
- Do not place weight or pressure on the CFA735.

11.5. 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.

11.6. 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.

11.7. Operation

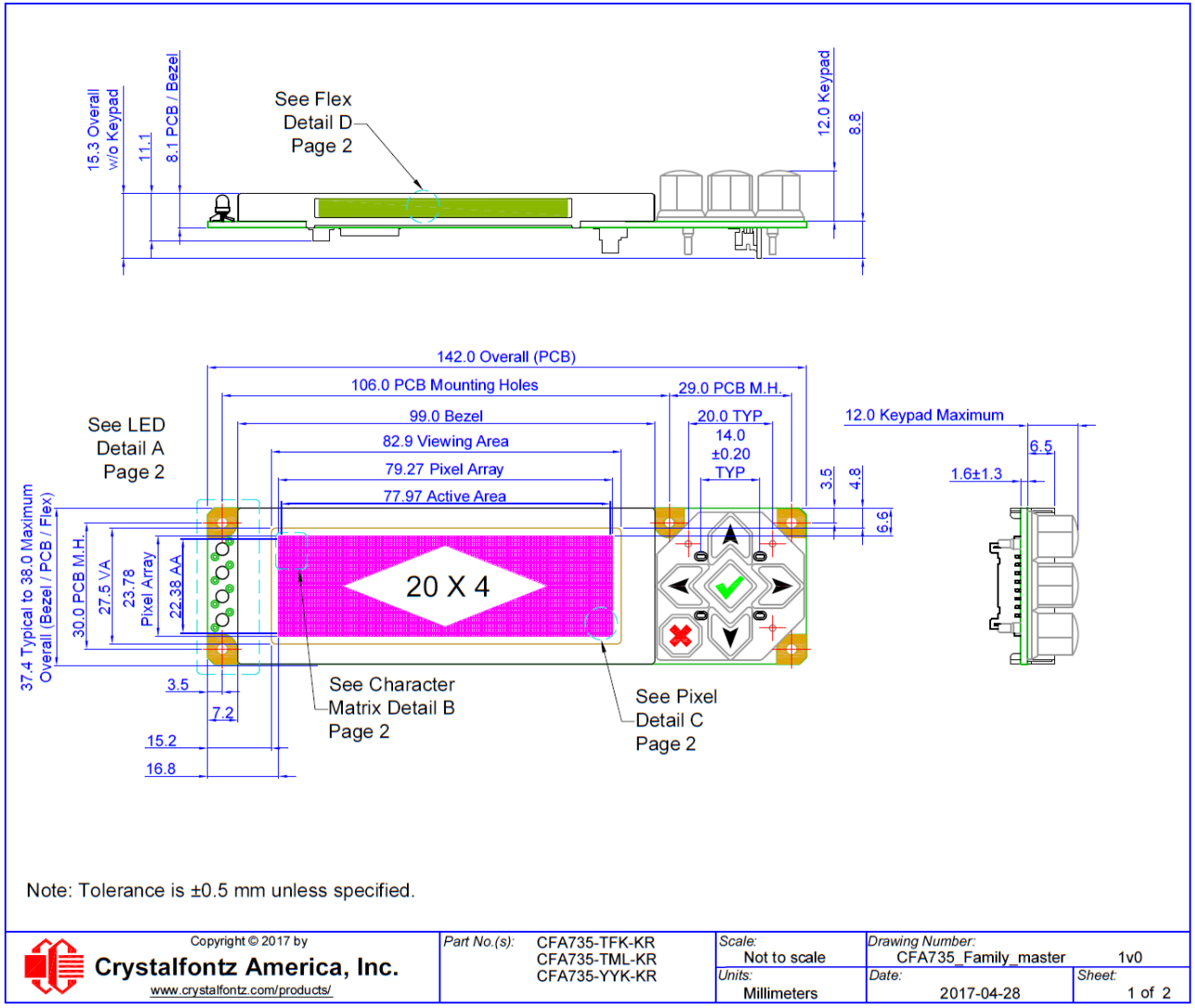
- Protect the CFA735 from ESD and power supply transients.
- Observe the operating temperature limitations: a minimum of -20°C to a maximum of +70°C with minimal fluctuation. Operation outside of these limits may shorten life and/or harm display.
  - At lower temperatures of this range, response time 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.

11.8. Storage and Recycling

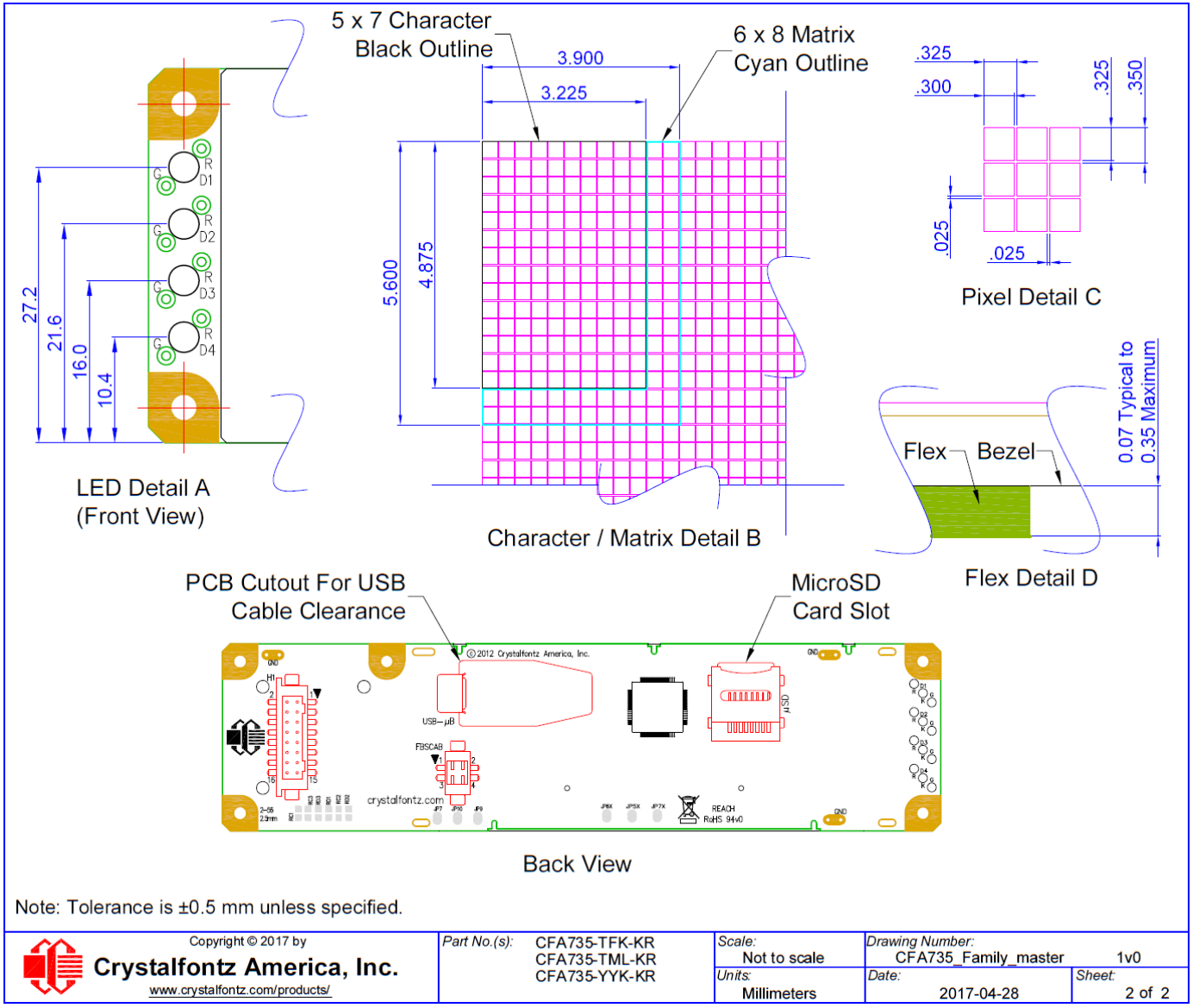
- Store in an ESD-approved container away from dust, moisture, and direct sunlight.
- Observe the storage temperature limitations: -30°C minimum, +80°C maximum with minimal fluctuation. Rapid temperature changes can cause moisture to form, resulting in permanent damage.
- Do not allow weight to be placed on the CFA735 while in storage.
- Please recycle your outdated Crystalfontz modules at an approved facility.

12. Mechanical Drawings

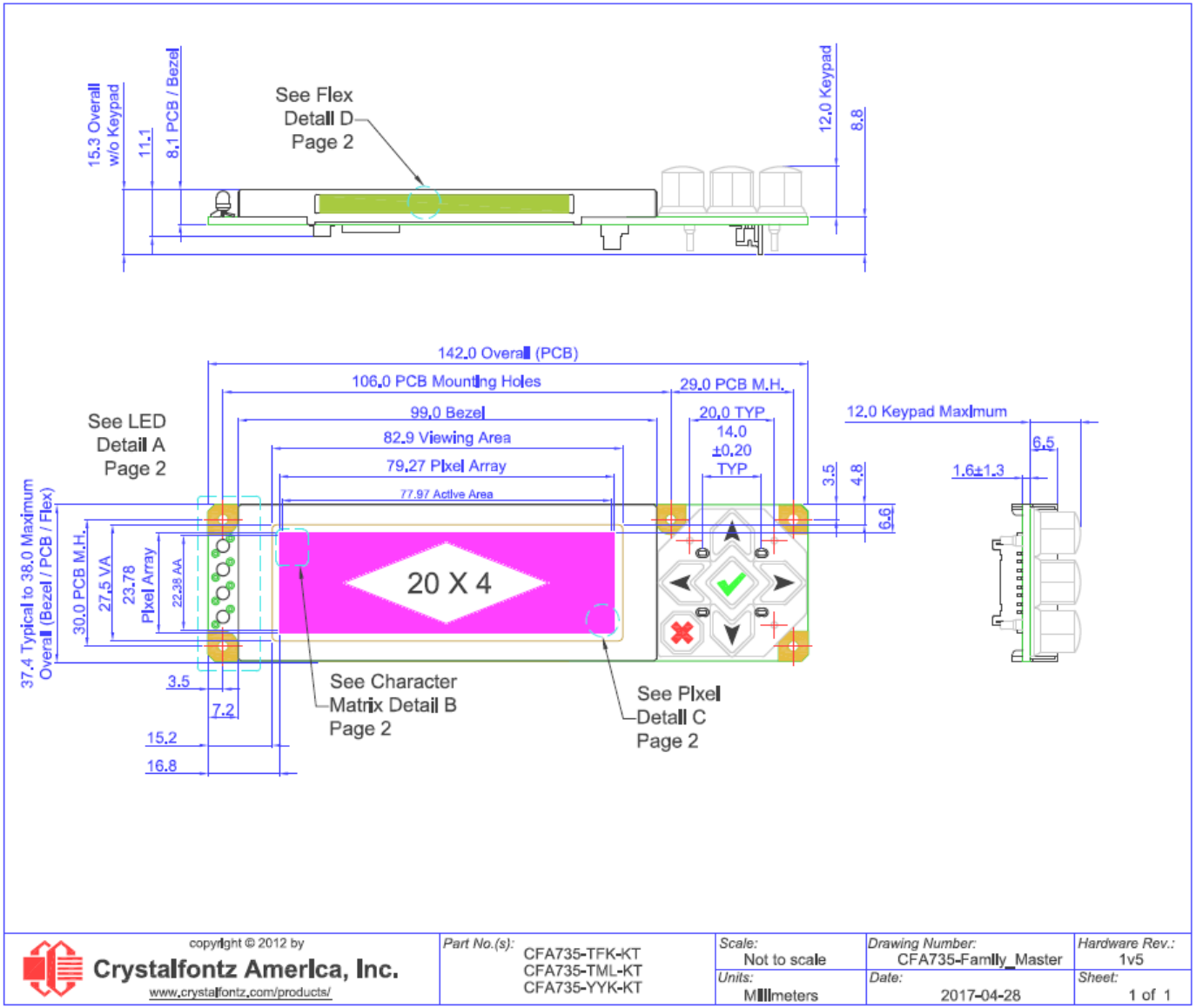
12.1. CFA735 Module Outline Drawing (KR Series, 1 of 2)



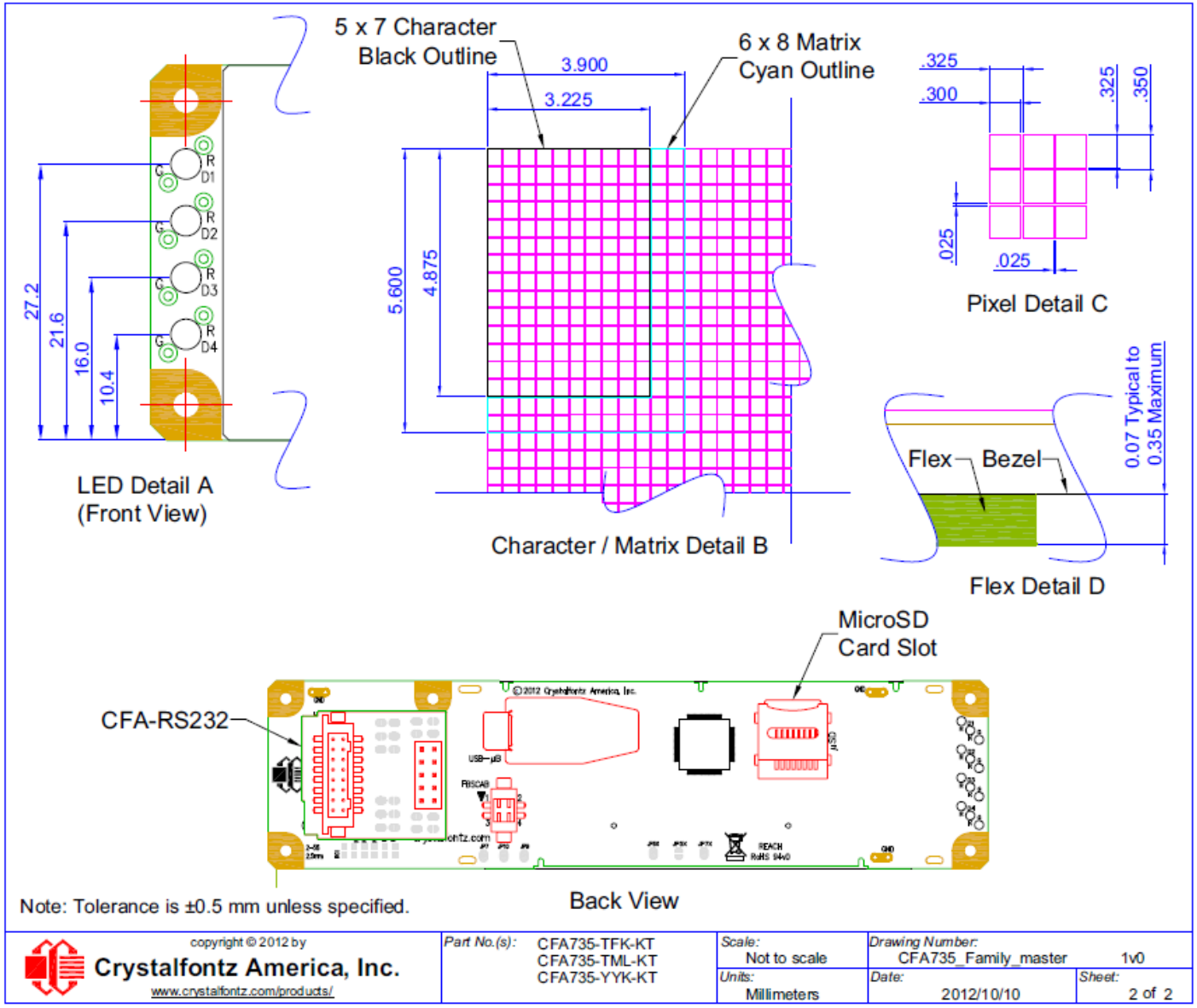
12.2. CFA735 Module Outline Drawing (KR Series, 2 of 2)



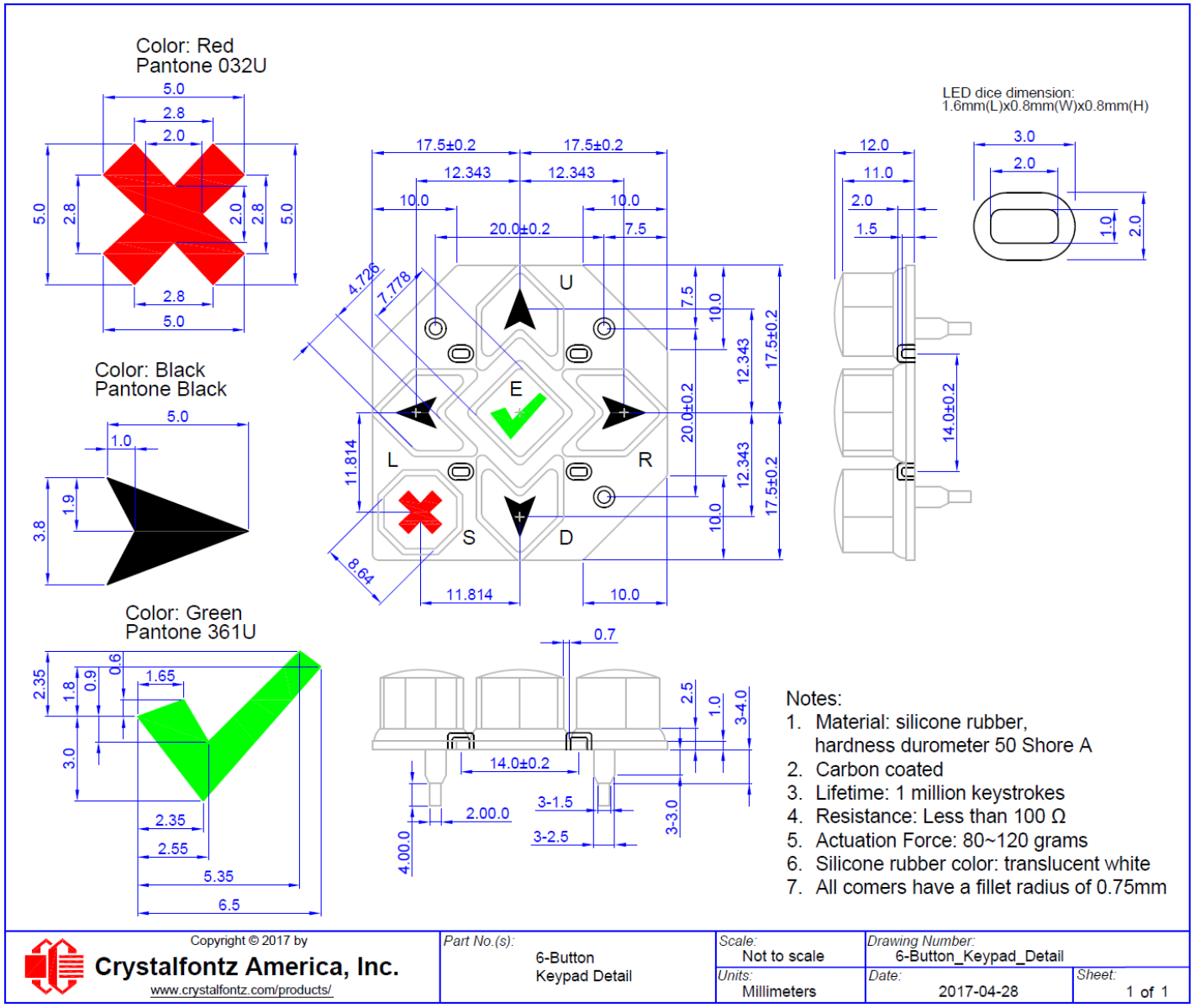
12.3. CFA735 Module Outline Drawing (KT Series, 1 of 2)



12.4. CFA735 Module Outline Drawing (KT Series, 2 of 2)



12.5. Keypad Detail Drawing (KR and KT Series)



13. Appendix A: Example Software and Sample Source Code

13.1. Example Software

- Crystalfontz cfTest (Windows compatible test/demonstration software): https://www.crystalfontz.com/product/cftest
- Crystalfontz WinTest (Windows compatible example program and source): https://www.crystalfontz.com/product/635wintest
- Linux compatible command-line demonstration and source: https://www.crystalfontz.com/product/linuxexamplecode
- Crystalfontz CrystalControl2 (Windows compatible LCD display software): https://www.crystalfontz.com/product/CrystalControl2.html
- LCDProc (Linux compatible open-source LCD display software): http://lcdproc.org/

13.2. 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 as that used in IrDA, which came from PPP, which seems to be related to a CCITT standard (ref: Network Working Group Request for Comments: 1171). 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/IrLAP
11_Plus_Errata.zip
//
//I doubt that there are any worries about the legality of this code,
//searching for the first line of the table below, it appears that
//the code is already included in the linux 2.6 kernel "Driver for
//ST5481 USB ISDN modem". This is an "industry standard" algorithm
//and I do not think there are ANY issues with it at all.

typedef unsigned char ubyte;
typedef unsigned short word;
word get_crc(ubyte *bufptr,word len)
{
   //CRC lookup table to avoid bit-shifting loops.
   static const word crcLookupTable[256] =
{0x00000,0x01189,0x02312,0x0329B,0x04624,0x057AD,0x06536,0x074BF,
0x08C48,0x09DC1,0x0AF5A,0x0BED3,0x0CA6C,0x0DBE5,0x0E97E,0x0F8F7,
0x01081,0x00108,0x03393,0x0221A,0x056A5,0x0472C,0x075B7,0x0643E,
0x09CC9,0x08D40,0x0BFDB,0x0AE52,0x0DAED,0x0CB64,0x0F9FF,0x0E876,
0x02102,0x0308B,0x00210,0x01399,0x06726,0x076AF,0x04434,0x055BD,
0x0AD4A,0x0BCC3,0x08E58,0x09FD1,0x0EB6E,0x0FAE7,0x0C87C,0x0D9F5,
0x03183,0x0200A,0x01291,0x00318,0x077A7,0x0662E,0x054B5,0x0453C,
0x0BDCB,0x0AC42,0x09ED9,0x08F50,0x0FBEF,0x0EA66,0x0D8FD,0x0C974,
0x04204,0x0538D,0x06116,0x0709F,0x00420,0x015A9,0x02732,0x036BB,
0x0CE4C,0x0DFC5,0x0ED5E,0x0FCD7,0x08868,0x099E1,0x0AB7A,0x0BAF3,
0x05285,0x0430C,0x07197,0x0601E,0x014A1,0x00528,0x037B3,0x0263A,
0x0DECD,0x0CF44,0x0FDDF,0x0EC56,0x098E9,0x08960,0x0BBFB,0x0AA72,
0x06306,0x0728F,0x04014,0x0519D,0x02522,0x034AB,0x00630,0x017B9,
0x0EF4E,0x0FEC7,0x0CC5C,0x0DDD5,0x0A96A,0x0B8E3,0x08A78,0x09BF1,
0x07387,0x0620E,0x05095,0x0411C,0x035A3,0x0242A,0x016B1,0x00738,
0x0FFCF,0x0EE46,0x0DCDD,0x0CD54,0x0B9EB,0x0A862,0x09AF9,0x08B70,
0x08408,0x09581,0x0A71A,0x0B693,0x0C22C,0x0D3A5,0x0E13E,0x0F0B7,
0x00840,0x019C9,0x02B52,0x03ADB,0x04E64,0x05FED,0x06D76,0x07CFF,
0x09489,0x08500,0x0B79B,0x0A612,0x0D2AD,0x0C324,0x0F1BF,0x0E036,
0x018C1,0x00948,0x03BD3,0x02A5A,0x05EE5,0x04F6C,0x07DF7,0x06C7E,
0x0A50A,0x0B483,0x08618,0x09791,0x0E32E,0x0F2A7,0x0C03C,0x0D1B5,
0x02942,0x038CB,0x00A50,0x01BD9,0x06F66,0x07EEF,0x04C74,0x05DFD,
0x0B58B,0x0A402,0x09699,0x08710,0x0F3AF,0x0E226,0x0D0BD,0x0C134,
0x039C3,0x0284A,0x01AD1,0x00B58,0x07FE7,0x06E6E,0x05CF5,0x04D7C,
0x0C60C,0x0D785,0x0E51E,0x0F497,0x08028,0x091A1,0x0A33A,0x0B2B3,
0x04A44,0x05BCD,0x06956,0x078DF,0x00C60,0x01DE9,0x02F72,0x03EFB,
0x0D68D,0x0C704,0x0F59F,0x0E416,0x090A9,0x08120,0x0B3BB,0x0A232,
0x05AC5,0x04B4C,0x079D7,0x0685E,0x01CE1,0x00D68,0x03FF3,0x02E7A,
0x0E70E,0x0F687,0x0C41C,0x0D595,0x0A12A,0x0B0A3,0x08238,0x093B1,
0x06B46,0x07ACF,0x04854,0x059DD,0x02D62,0x03CEB,0x00E70,0x01FF9,
0x0F78F,0x0E606,0x0D49D,0x0C514,0x0B1AB,0x0A022,0x092B9,0x08330,
0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78};

   register word newCrc = 0xFFFF;
   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 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;
   ubyte data;
   int bitcount;

   //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(bitcount = 0; bitcount <= 7; bitcount++)
      {
         //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(bitcount = 0; bitcount <= 15; bitcount++)
   {
      //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,0x0AC42,0x09ED9,0x08F50,0x0FBEF,0x0EA66,0x0D8FD,0x0C974,
0x04204,0x0538D,0x06116,0x0709F,0x00420,0x015A9,0x02732,0x036BB,
0x0CE4C,0x0DFC5,0x0ED5E,0x0FCD7,0x08868,0x099E1,0x0AB7A,0x0BAF3,
0x05285,0x0430C,0x07197,0x0601E,0x014A1,0x00528,0x037B3,0x0263A,
0x0DECD,0x0CF44,0x0FDDF,0x0EC56,0x098E9,0x08960,0x0BBFB,0x0AA72,
0x06306,0x0728F,0x04014,0x0519D,0x02522,0x034AB,0x00630,0x017B9,
0x0EF4E,0x0FEC7,0x0CC5C,0x0DDD5,0x0A96A,0x0B8E3,0x08A78,0x09BF1,
0x07387,0x0620E,0x05095,0x0411C,0x035A3,0x0242A,0x016B1,0x00738,
0x0FFCF,0x0EE46,0x0DCDD,0x0CD54,0x0B9EB,0x0A862,0x09AF9,0x08B70,
0x08408,0x09581,0x0A71A,0x0B693,0x0C22C,0x0D3A5,0x0E13E,0x0F0B7,
0x00840,0x019C9,0x02B52,0x03ADB,0x04E64,0x05FED,0x06D76,0x07CFF,
0x09489,0x08500,0x0B79B,0x0A612,0x0D2AD,0x0C324,0x0F1BF,0x0E036,
0x018C1,0x00948,0x03BD3,0x02A5A,0x05EE5,0x04F6C,0x07DF7,0x06C7E,
```

```
0x0A50A,0x0B483,0x08618,0x09791,0x0E32E,0x0F2A7,0x0C03C,0x0D1B5,
0x02942,0x038CB,0x00A50,0x01BD9,0x06F66,0x07EEF,0x04C74,0x05DFD,
0x0B58B,0x0A402,0x09699,0x08710,0x0F3AF,0x0E226,0x0D0BD,0x0C134,
0x039C3,0x0284A,0x01AD1,0x00B58,0x07FE7,0x06E6E,0x05CF5,0x04D7C,
0x0C60C,0x0D785,0x0E51E,0x0F497,0x08028,0x091A1,0x0A33A,0x0B2B3,
0x04A44,0x05BCD,0x06956,0x078DF,0x00C60,0x01DE9,0x02F72,0x03EFB,
0x0D68D,0x0C704,0x0F59F,0x0E416,0x090A9,0x08120,0x0B3BB,0x0A232,
0x05AC5,0x04B4C,0x079D7,0x0685E,0x01CE1,0x00D68,0x03FF3,0x02E7A,
0x0E70E,0x0F687,0x0C41C,0x0D595,0x0A12A,0x0B0A3,0x08238,0x093B1,
0x06B46,0x07ACF,0x04854,0x059DD,0x02D62,0x03CEB,0x00E70,0x01FF9,
0x0F78F,0x0E606,0x0D49D,0x0C514,0x0B1AB,0x0A022,0x092B9,0x08330,
0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78
   };
   public static int compute(byte[] data)
   {
      int newCrc = 0x0FFFF;
      for (int i = 0; i < data.length; i++ )
      {
         int lookup = crcLookupTable[(newCrc ^ data[i]) & 0xFF];
         newCrc = (newCrc >> 8) ^ lookup;
      }
      return(~newCrc);
   }
}
```

Algorithm 6: "Perl" Table Implementation

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

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

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

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

my $packet = chr(hex $type) .chr(hex $length) .chr(hex $data);
my $valid_crc = '5584' ;
print "A CRC of Packet ($packet) Should Equal($valid_crc)\n";
my $crc = 0xFFFF ;
printf("%x\n", $crc);

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

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

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

Algorithm 7: For PIC18F8722 or PIC18F2685

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

```
;CRC Algorithm for CrystalFontz CFA635 display (DB535)
; This code written for PIC18F8722 or PIC18F2685
;
; Your main focus here should be the ComputeCRC2 and
; CRC16_ routines
;
;=================================================================
ComputeCRC2:
      movlb       RAM8
      movwf       dsplyLPCNT          ;w has the byte count
nxt1_dsply:
      movf        POSTINC1            ;w
      call        CRC16
      decfsz      dsplyLPCNT
      goto        nxt1_dsply
      movlw       .0                  ;shift accumulator 16 more bits
      call        CRC16
      movlw       .0
      call        CRC16
      comf        dsplyCRC,F          ;invert result
      comf        dsplyCRC+1,F
      return
;=================================================================
CRC16 movwf:
      dsplyCRCData                    ;w has the byte crc
      movlw       .8
      movwf       dsplyCRCCount
_cloop:
      bcf         STATUS,C            ; clear carry for CRC register shift
      rrcf        dsplyCRCData,f      ; perform shift of data into CRC
                                      ; register
      rrcf        dsplyCRC,F
      rrcf        dsplyCRC+1,F
      btfss       STATUS,C            ; skip jump if carry
      goto    _   notset              ; otherwise goto next bit
      movlw       0x84                ; XOR poly mask with CRC register
      xorwf       dsplyCRC,F
_notset:
      decfsz      dsplyCRCCount,F     ; decrement bit counter
      bra    cloop                    ; loop if not complete
      return
;=================================================================
; example to clear screen
dsplyFSR1_TEMP     equ    0x83A  ;    ; 16-bit save for FSR1 for display
                                      ; message handler
dsplyCRC           equ    0x83C        ; 16-bit CRC (H/L)
dsplyLPCNT         equ    0x83E        ; 8-bit save for display message
                                      ; length - CRC
dsplyCRCData       equ    0x83F        ; 8-bit CRC data for display use
dsplyCRCCount      equ    0x840        ; 8-bit CRC count for display use
SendCount          equ    0x841        ; 8-bit byte count for sending to
                                      ; display
RXBUF2             equ    0x8C0        ; 32-byte receive buffer for
```

```
                                        ; Display
TXBUF2          equ    0x8E0            ; 32-byte transmit buffer for
                                        ; Display
;-------------------------------------------------------------
ClearScreen:
        movlb       RAM8
        movlw       .0
        movwf       SendCount
        movlw       0xF3
        movwf       dsplyCRC     ; seed ho for CRC calculation
        movlw       0x21
        movwf       dsplyCRC+1          ; seen lo for CRC calculation
        call        ClaimFSR1
        movlw       0x06
        movwf       TXBUF2
        LFSR        FSR1,TXBUF2
        movf        SendCount,w
        movwf       TXBUF2+1            ; message data length
        call        BMD1
        goto        SendMsg
;=============================================================================
; send message via interrupt routine. The code is made complex due
; to the limited FSR registers and extended memory space used
;
; example of sending a string to column 0, row 0
;-------------------------------------------------------------
SignOnL1:
        call        ClaimFSR1
        lfsr        FSR1,TXBUF2+4       ; set data string position
        SHOW        C0R0,BusName        ; move string to TXBUF2
        movlw       .2                  ;
        addwf       SendCount           ;
        movff       SendCount,TXBUF2+1
                                        ; insert message data length
        call        BuildMsgDSPLY
        call        SendMsg
        return
;=============================================================================
; BuildMsgDSPLY used to send a string to LCD
;-------------------------------------------------------------
 BuildMsgDSPLY:
        movlw       0xF3
        movwf       dsplyCRC            ; seed hi for CRC calculation
        movlw       0x21
        movwf       dsplyCRC+1          ; seed lo for CRC calculation
        LFSR        FSR1,TXBUF2         ; point at transmit buffer
        movlw       0x1F                ; command to send data to LCD
        movwf       TXBUF2              ; insert command byte from us to
                                        ; CFA635
        BMD1        movlw .2
        ddwf        SendCount,w         ; + overhead
        call        ComputeCRC2         ; compute CRC of transmit message
        movf        dsplyCRC+1,w
        movwf       POSTINC1            ; append CRC byte
        movf        dsplyCRC,w
        movwf       POSTINC1            ; append CRC byte
        return
;=============================================================================
SendMsg:
        call        ReleaseFSR1
        LFSR        FSR0,TXBUF2
        movff       FSR0H,irptFSR0
        movff       FSR0L,irptFSR0+1
                                        ; save interrupt use of FSR0
        movff       SendCount,TXBUSY2
        bsf         PIE2,TX2IE
```

```
                                        ; set transmit interrupt enable
                                        ; (bit 4)
      return
;=====================================================================
; macro to move string to transmit buffer
SHOW macro    src, stringname
      call          src
      MOVLF         upper stringname, TBLPTRU
      MOVLF         high stringname, TBLPTRH
      MOVLF         low stringname, TBLPTRL
      call          MOVE_STR
      endm
;=====================================================================
MOVE_STR:
      tblrd         *+
      movf          TABLAT,w
      bz            ms1b
      movwf         POSTINC1
      incf          SendCount
      goto          MOVE_STR

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

14. Appendix B: CRYSTALFONTZ USB MODULE FIRMWARE UPDATE INSTRUCTIONS

These instructions apply to:

- **CFA10052 hardware version v1.0 and above, including CFA735 and CFA835 of hardware version v1.0 and above.**
- **CFA635 hardware version v1.4 and above.**

There are three methods for updating the firmware:

1 - Using a USB or Serial connection to a Windows PC (keypad reset)
2 - Using a USB or Serial connection to a Windows PC (software reset)
3 - Using a microSD card

Method 1 - Using a USB or Serial connection to a Windows PC (keypad reset)

1. Make sure the appropriate Crystalfontz Windows USB drivers are installed (available from the Crystalfontz website).

2. While holding the UP & DOWN keys on the module, power-on the module by plugging it into a USB port, or supplying it power (if using serial connection). The module should display a firmware update screen. If not, try this step again.
   Note: if this step is difficult due to physical module installation, please see update Method 2.

3. On the PC, run "fw_send.exe" (Crystalfontz Module Firmware Update Utility).

4. In the utility, select the new firmware file (BLF file extension).
   Firmware file version information should be shown in the "information" box.

5. In the communications box, select the module. It should be listed as "CFA10052-USB Bootloader" or “CFA635-USB Bootloader".
   If the module is listed as its normal type (i.e., “Crystalfontz CFA835-USB”), then it is not in bootloader mode. Repeat Step 2, or try one of the other update methods.

6. Click the "Update Firmware" button.
   Note: When updating to a previous version, or to a special version of the firmware, the "forced update" checkbox may need to be selected before clicking the update button.

7. Both the status box on the PC, and the screen on the module will show updating progress.

8. When complete, the module will reset itself.

Method 2 - Using a USB or Serial connection to a Windows PC (software reset)

1. Make sure the appropriate Crystalfontz Windows USB drivers are installed (available from the Crystalfontz website).

2. Make sure the module is plugged into the PC, powered on, and no other software is currently using the display.

3. On the PC, run "fw_send.exe" (Crystalfontz Module Firmware Update Utility).

4. In the utility, select the new firmware file (BLF file extension).
   Firmware version information should be shown in the "information" box.

5. In the communications box, select the module to update.

6. Click the "Rest Module into Bootloader Mode".
   After a few seconds, the module should reboot itself and display the firmware update screen.

7. In the communications box, re-select the module. It should now be listed as "CFA10052-USB Bootloader" or "CFA635-USB Bootloader".

9. Click the "Update Firmware" button.
   Note: When updating to a previous version, or to a special version of the firmware, the "forced update" checkbox may need to be selected before clicking the update button.

8. Both the status box on the PC, and the screen on the module will show updating progress.

9. When complete, the module will reset itself.

Method 3 - Using a microSD card

1. Prepare the microSD card by formatting the microSD card to the FAT32 filesystem on a Windows PC.

2. Copy the firmware file (BLF file extension) on to the microSD card.

3. Rename the BLF file to match the module type, i.e., "cfa635.blf", "cfa735.blf", or "cfa835.blf".

4. With the module turned off (USB cable disconnected, or un-powered), insert the microSD card into the back of the module.

5. While holding the UP & DOWN keys on the module, power-on the module by plugging it into a USB port, or supplying it power (if using serial connection).

6. The firmware updater should now be displayed on the module, and ask if you wish to flash the new firmware. To confirm, press the TICK (center green) button.

7. The module will now update its firmware, and reboot itself when complete.
