INTELLIGENT LCD MODULE SPECIFICATIONS

Hardware Version: v1.3
Firmware Version: v1.1

Datasheet Release: 2019-10-28

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: v1.1 Datasheet Release: 2019-10-28 For information about firmware and hardware revisions, see the Part Change Notifications (PCN) under "News" in our website's navigation bar. To see the most recent PCN for the CFA835 family at the time of this datasheet release, see PCN #10959. Previous datasheet Version: 2019/10/21 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.

2. Introduction

The CFA835 family is available in two colors:

- CFA835-TFK 
  Dark letters on a light background; this display can be read in normal office lighting, in dark areas, and in bright sunlight
- CFA835-TML 
  Light letters on a blue background; this display can be read in normal office lighting and in dark areas

Both 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. The special "bridged" interface mode allows the use of an interface to communicate with other slave devices.

2.1. Main Features

- The CFA835 is a 244x68 graphic display module that can display 5-bit (32 shade) grayscale images from a host computer or a microSD card.
- Create your own custom-made Unicode compatible fonts to fit as many as 80 small characters by 8-lines in any language or use our standard set of characters. Please see Character Generator ROM (CGROM).
- Fits nicely in a 1U rack mount case (37 mm overall height).
- May be installed in a standard half-height 5 1/4 drive bay by using our optional drive bay mounting bracket or our optional SLED bracket. The SLED holds the CFA835 display module, an optional CFA-FBSCAB and has mounting points for a standard 3.5-inch hard disk drive.
- LCD is edge-lit with 12 long-life, high performance, LEDs (6 per side).
- The LCD has a wide viewing angle, with a 12 o'clock preferred viewing direction.
- Wide temperature operating range is -20°C to +70°C.
- Free downloadable utilities package includes the CFA835 Font Editor, CFA835 Video Encoder, and CFA835 Graphic Test. Please see CFA835 Utilities.
- The CFA835 is mechanically similar but not identical to the CFA635. Location of mounting holes, display, and keypad are the same for the CFA835 and CFA635. The CFA835 hardware is identical to the CFA735 hardware.
- The CFA835 command set supports most of the commands from the standard Crystalfontz Packet Command set. Certain commands have been expanded to support increased functionality, as well as the addition of many new useful commands, see Command Codes.
  - Free software cfTest can be downloaded to experiment with this command set.
  - Commands include tools to draw lines, rectangles, and circles. For example, create a chart to display temperature fluctuations over time.
- Adjustable contrast. The default contrast value for the module will be acceptable for most applications. If necessary, contrast can be adjusted using a command packet.
- Only a single supply is needed. Wide power supply voltage range (+3.3v to +5.5v) is perfect for most embedded systems.
- 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.
- The CFA835 is powered by an ST-Micro STM32F401 with a Sitronix ST7529 driver/controller.
- The CFA835 has a microSD card socket and is compatible with most FAT12/16/32 formatted SDHC microSD cards.
  - The microSD card can be used to display images, play videos, manipulate files, and update firmware.
- Nonvolatile memory capability (EEPROM):
  - Customize the "power-on" display settings (backlight brightness, boot screen, LED settings).
  - 124-byte "scratch" register for storing IP address, netmask, system serial number, etc.

- Optional ATX Power Supply control functionality allows the CFA835's buttons to replace the Power and Reset switches on your system, simplifying front panel design. Optional Crystalfontz WR-PWR-Y25 or WR-PWR-Y38 cables may be used to simplify connection to the host's motherboard.
- Five GPIOs can be used to control other devices or can be configured for optional ATX functionality.
- Hardware watchdog can reset host on host software failure.
- Crystalfontz America, Inc. is ISO 9001:2015 certified.
- A Declaration for Conformity with RoHS, and REACH:SVHC are available on the product's webpage.

2.2. Serial Interface Types / Options

Both of the serial interfaces use firmware that bring the two UART pins (Tx & Rx) of the CFA835's microcontroller to the CFA835's H1 connector.

"Logic Level, Inverted" Serial

Logic-level, inverted serial is the default interface for the CFA835. This interface exposes the UART Tx & Rx ("logic level", 0v to +3.3v Tx nominal, 0v to +5.0v Rx nominal) on pin 1 and pin 2 of the CFA835's 16-pin H1 connector. If your embedded processor is close to the CFA835, you can cable its UART Rx and Tx pins directly to the CFA835's Tx and Rx pins. No RS232 level translators are required on either end.

"Full Swing" RS232 Serial (Requires Optional CFA-RS232 Level Translator Board)

Bidirectional 9600 / 19200 / 115200 baud ESD protected RS232 is provided when you customize your web order with a mounted CFA-RS232 Level Translator. This interface is the correct choice if your embedded controller or host system has a "full swing" RS232 serial port implemented with a UART plus RS232 level converter.

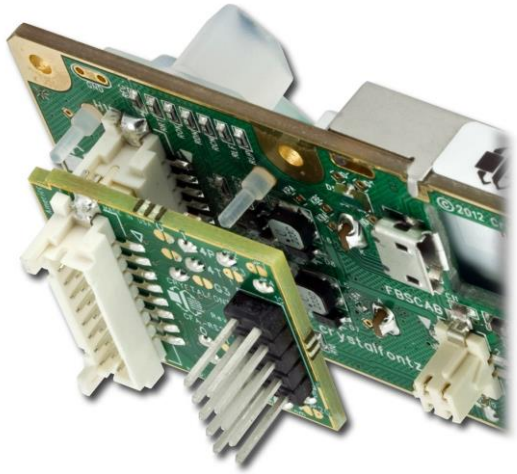


Figure 1. Angled View of CFA-RS232 Level Translator Mounted on CFA835

The CFA-RS232 Level Translator has a 16-pin female connector J3 that mates with the male 16-pin connector H1 on the back of the CFA835. The CFA-RS232 converts the 0v to +5v (logic level), Rx and Tx signals from the CFA835's microcontroller to RS232 levels.

For more information, please download the CFA-RS232 Datasheet from our website.

2.3. Additional Features When Used With CFA-FBSCAB

To use all of the command set described in command 37 (0x25): CFA-FBSCAB, at least one optional CFA-FBSCAB is required. Add up to 32 CFA-FBSCABs to your CFA835 order using the "Customize and Add to Cart" feature on our website. If you add optional CFA-FBSCABs, you will be prompted to add one WR-EXT-Y37 extension cable to your order.

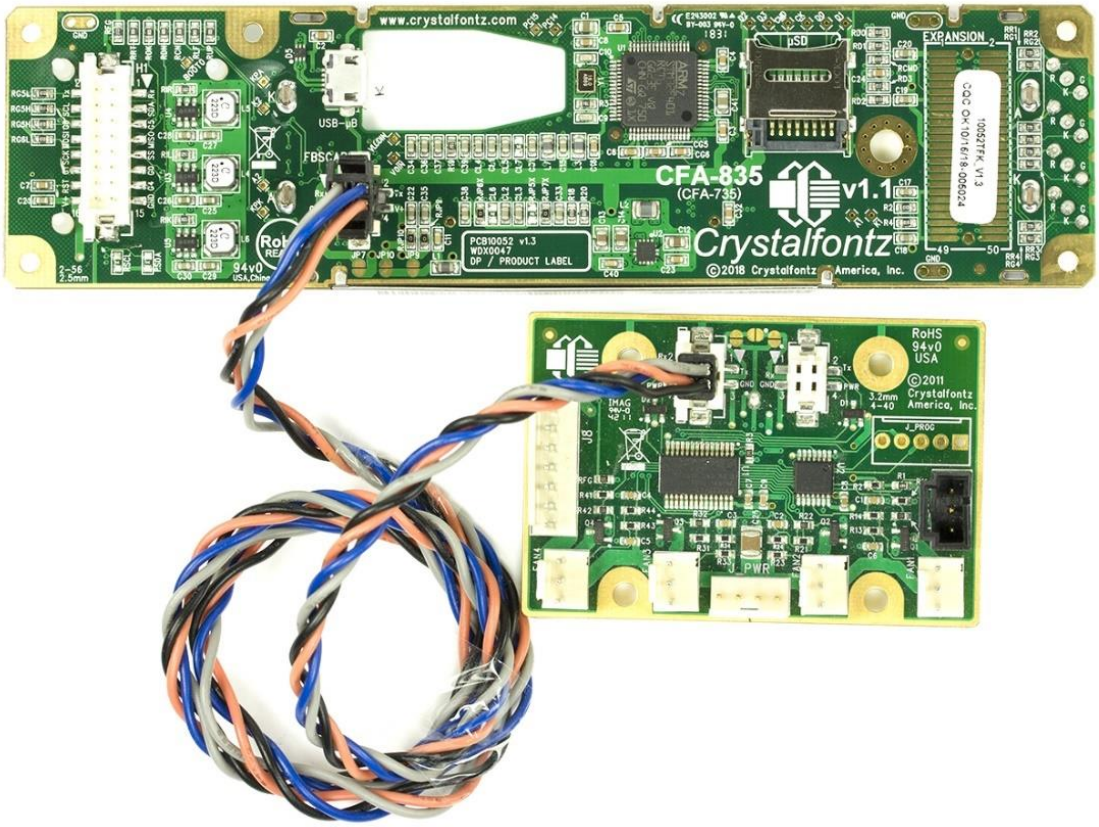


Figure 2. Optional CFA-FBSCAB Connected to the CFA835 with WR-EXT-Y37 Cable

A CFA835 + CFA-FBSCAB allows:

- Easy control of up to four fans with tachometer speed monitoring and variable PWM control per CFA-FBSCAB. Fail-safe fan power settings allow your host to safely control the fans based on temperature. Buy one 3-pin fan extension cable WR-FAN-X01 to connect each fan.
- Add up to 16 Crystalfontz WR-DOW-Y17 temperature sensor cables that have Maxim DS18B20 Programmable Resolution 1-Wire temperature sensors. The DS18B20 has an operating temperature range of -55°C to +125°C and is accurate to ±0.5°C over the range of -10°C to +85°C
- Up to 32 CFA-FBSCABs can be connected to each other ("daisy chained"), see image below. Maximum configuration will allow 128 fans and 512 temperature sensors.

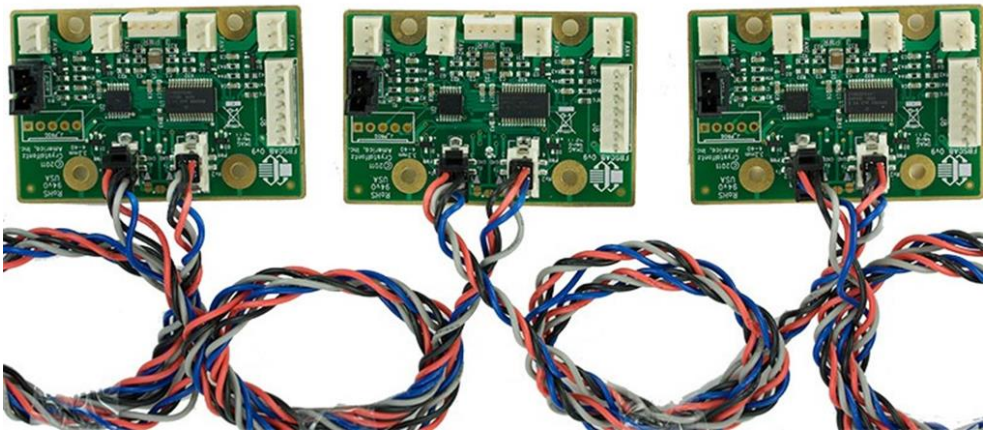


Figure 3. Example of CFA-FBSCABs Daisy-Chained Using the WR-EXT-Y37 Cable

NOTE: The CFA835+CFA-FBSCABs provide no ATX functionality through the CFA-FBSCABs. However, ATX control is still available using the H1 connector on the CFA835.

IMPORTANT: Remove power before connecting or disconnecting multiple CFA-FBSCABs. Connecting or disconnecting multiple CFA-FBSCABs while powered will cause addressing problems. For more information, please download the CFA-FBSCAB datasheet from our website.

2.4. Module Classification Information

CFA 835 - X X X
1 2 3 4 5

❶	**Brand**	Crystalfontz America, Inc.
❷	**Model Identifier**	835
❸	**Backlight Type & Color**	T – LED, White
❹	**Fluid Type, Image (Positive or Negative), & LCD Glass Color**	F – FSTN, Positive M – STN, Negative, Blue
❺	**Polarizer Film Type, Temperature Range, & Viewing Direction (O 'Clock)**	K – Transflective, Wide Temperature -20°C to +70°C, 12:00 L – Transmissive, Wide Temperature -20°C to +70°C, 12:00

2.5. Ordering Information

Part Number	Fluid	LCD Glass Color	Image	Polarizer Film	Backlight Color/Type
CFA835-TFK	FSTN	Neutral	Positive	Transflective	Backlight: White Keypad: White 
CFA835-TML	STN	Blue	Negative	Transmissive	Backlight: White Keypad: Blue 

2.6. Display Mounts

On the webpage for the CFA835, 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 4. CFA835 Drive Bay Bracket

Figure 5. CFA835 SLED

2.7. Cables

Below is a list of some of the cables we offer to make it easy to integrate the CFA835 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		**For use with CFA-RS232:** Use this ribbon cable to supply communications. Connect cable's 10-pin female connector to module's J_RS232 male connector. Connect cable's RS232 DB9 9-pin female connector to host's DB9 9-pin male serial port. Default or alternate motherboard RS-232 pinouts can be accommodated by changing jumpers on module.
WR-232-Y22 ~26 inches		**For use with CFA-RS232:** Use this cable to supply communications. Connect one of the 10-pin female connectors to the module's' J_RS232 10-pin male connector. Connect cable's second 10-pin female connector to host's motherboard 10-pin male connector. This cable supports standard or alternate pinout motherboard RS-232 connections without changing jumpers on module.
WR-232-Y23 ~26 inches		**For use with CFA-RS232:** Connect cable's 0.1" 2x5 female connector to the CFA-RS232's J1 10-pin connector. Connect cable's RS232 DB9 9-pin female connector to host's external 9-pin serial port. Choose standard or alternate pinout. **NOTE**: This cable is not listed on the CFA835 "Customize and Add to Cart" feature. Add the cable as a separate item to your order.
WR-USB-Y27 ~6 feet		**For use with USB:** Connect cable's Micro-B USB female connector to CFA835's Micro-B USB connector. Connect cable's USB-A female connector to host's USB-A connector.
WR-USB-Y34 ~27.5 inches		**For use with USB:** Connect cable's Micro-B USB female connector to CFA835's Micro-B USB connector. Connect cable's single piece 4-pin 0.1" female 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		**For use with ATX:** Use this cable to supply power to the CFA835 directly from a PC power supply's "hard-drive" connector, rather than the normal USB power.
WR-PWR-Y25 ~11 inches		**For use with ATX:** Use this cable to simplify the connections for using ATX power and reset control. One end plugs into the CFA835 H1 connector. The other end has connections for power control, reset control, always on power, switched power, and ground.
WR-PWR-Y12 ~13 inches		**For use with CFA-FBSCAB:** Use this cable to plug a 4-pin "hard drive style" Molex power connector into module's "floppy drive style" power connector, plus provides an additional female 4-pin Molex connector.

Crystalfontz Cable	Image	Description All Cables Are RoHS Compliant
WR-EXT-Y37 ~18 inches		**For use with CFA-FBSCAB:** Use this cable to connect the CFA835 to the CFA-FBSCAB.
WR-PWR-Y38 ~2 ft. 11 inches		**For use with CFA-RS232:** Longer version of the WR-PWR-Y25 (described 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 One Wire) DS18B20 temperature sensor cables to the CFA-FBSCAB.

3. Mechanical Characteristics

3.1. Physical Characteristics

Item	Specification (mm)	Specification (inch, reference)
Overall Width and Height	142.0 (W) x 37.4 (H)	5.591 (W) x 1.46 (H)
Viewing Area / Bezel Opening	82.9 (W) x 27.5 (H)	3.264 (W) x 1.083 (H)
Active Area	77.97 (W) x 22.38 (H)	3.069 (W) x 0.881 (H)
5x7 Standard Character Size	3.225 (W) x 4.875 (H)	0.127 (W) x 0.192 (H)
6x8 Character Matrix	3.900 (W) x 5.600 (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)
Module Depth with Keypad, with Connectors	20.80	0.819
Keystroke Travel (approximate)	~2.4	0.094
Weight (typical)	55 grams	1.94 ounces
Weight (typical) (with CFA-RS232 Level Translator mounted)	60 grams	2.12 ounces

3.2. Optical Characteristics CFA835-TFK

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 CFA835-TML

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 CFA835 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 Voltage	V		15.6	16.8	18.0	v
Reverse Voltage	V_R				5	v
Chromaticity	x		.27	.29	.31	
	y		.29	.31	.33	
Luminance			1080	1350		Cd/m^2
LED Lifetime				50K		hours

4. Electrical Specifications

4.1. System Block Diagram

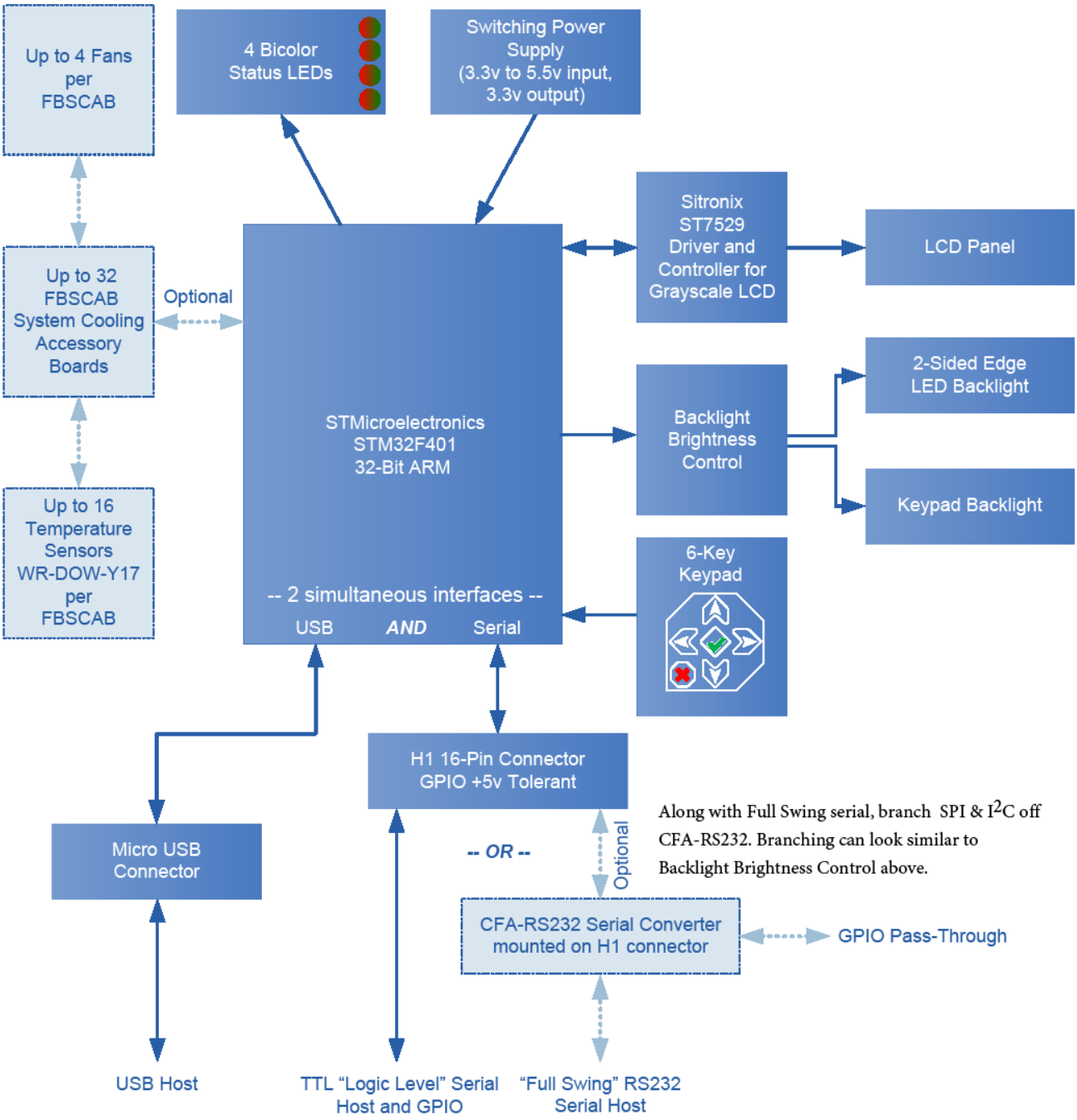


Figure 6. System Block Diagram

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}-V_{SS}	-0.5v	+4.0v
KT Series: Input and Output Pins for CFA-RS232 Serial			
CFA-RS232 Input Pin	V_{RX}	-25v	+25v
CFA-RS232 Output Pin	V_{TX}	-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.

CFA835-TFK (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

CFA835-TML (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

The module has three connectors on the back of the PCB: H1, USB, and FBSCAB. The H1 connector can be used for “logic level” serial interface and GPIO/ATX functionality. For “full swing” RS232 serial interface, the optional CFA-RS232 Serial Level Translator is mounted on H1.

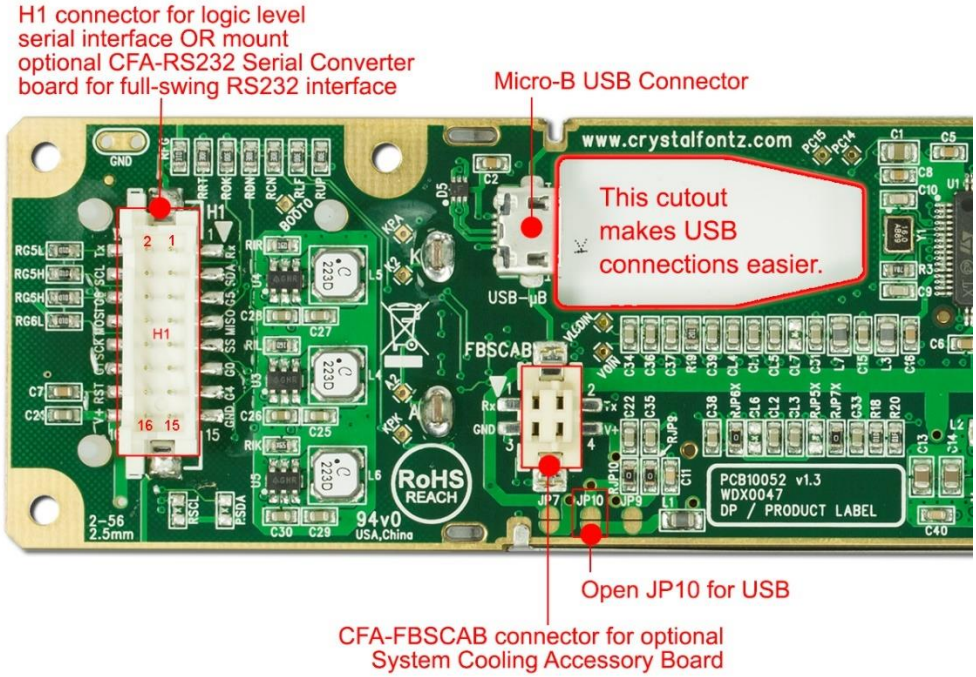


Figure 7. Location of CFA835 Connectors

6.2. H1 Connector Pin Assignments

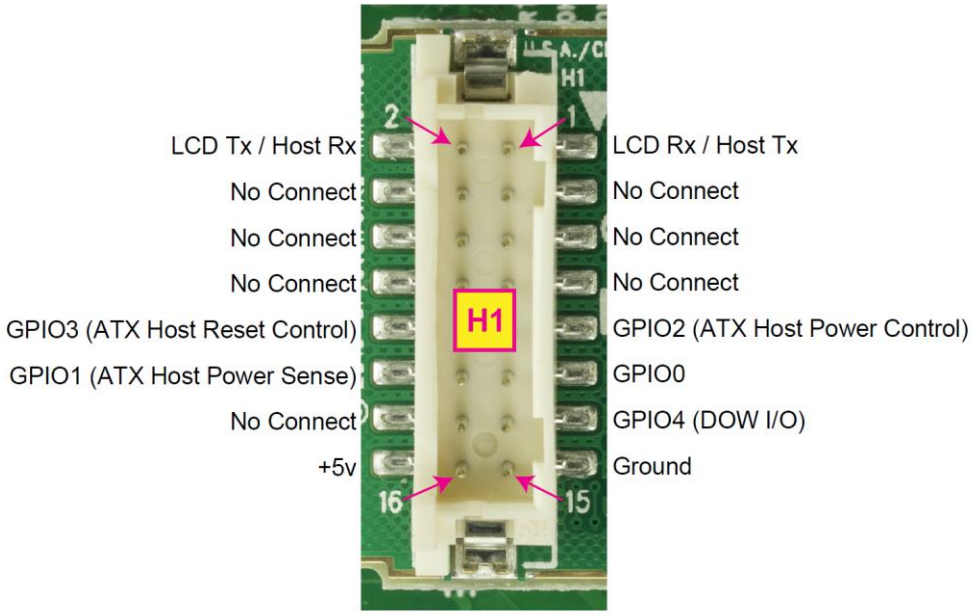


Figure 8. CFA835’s H1 Connector Pin Assignments

Make Your Own H1 Cable

The following parts may be used to make your own cable to connect to the CFA835's H1 connector:

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

6.3. Connecting 5v Power and Data Communications Through USB

The CFA835 has a USB peripheral, requiring only one connection to the host for both data communications and power supply.

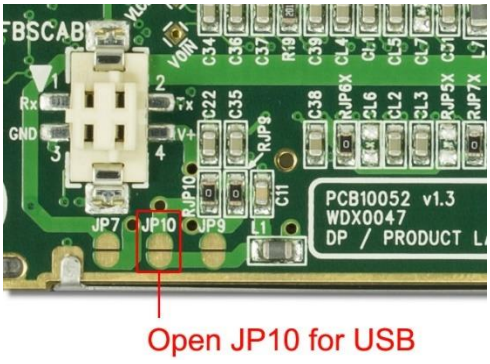
For host PCs using Microsoft Windows, the drivers will load automatically.

The Micro-B USB connector and the cutout in the PCB keeps the CFA835 profile as thin as possible. You can connect the CFA835 to one host using a USB interface while at the same time using a serial interface to a second host.

Figure 9. Connecting 5v Power Through USB

IMPORTANT: Too much pressure may permanently damage the CFA835's Micro-B USB connector. Keep the Micro-B USB cable connector parallel to the CFA835 when plugging or unplugging the cable. Do not lift or pull up on the cable.

Using USB Interface While Supplying Power Through H1



JP10 on the CFA835 is closed by factory default. If you are going to use USB interface while supplying power through H1, you must open JP10 to prevent back-powering the USB.

7. ATX Power Supply and Control Connections

ATX power supply control functionality allows the buttons on the CFA835 to replace the power and reset button on your system, simplifying front panel design.

NOTE: The CFA835+FBSCAB has no ATX functionality provided through the CFA-FBSCAB. However, ATX control is available using a WR-PWR-Y25 or WR-PWR-Y38 cable on the H1 connector of the CFA835.

IMPORTANT: 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 be changed by the user. Please see command 34 (0x22): Set or Set and Configure GPIO Pin.

GPIO[1] ATX Host Power Sense

Since the CFA835 must act differently depending on whether the host's power supply is on or off, you must connect the host's “switched +5v” to GPIO[1]. This GPIO line functions as POWER SENSE. The POWER SENSE pin is configured as an input with a pull-down, 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. Please see command 28 (0x1C): Set ATX Power Switch Functionality.

GPIO[3] ATX Host Restart Control

The motherboard's reset switch input is connected to GPIO[3]. This GPIO line functions as RESTART. The RESTART 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 RESTART_INVERT. Please see command 28 (0x1C): ATX Functionality. This connection is also used for the hardware watchdog.

ATX Power Supply & Control Connections	Pins on H1 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
*For “Full Swing” RS232 using the optional CFA-RS232 Level Translator Board, the H1 pins are passed through to the CFA-RS232’s J1 connector.	

7.1. ATX Connection to H1 Using WR-PWR-Y25/38 Cable

The illustration below shows how the Crystalfontz WR-PWR-Y25 and WR-PWR-Y38 ATX cables connect to the CFA835 H1 connector and your system's host and ATX Power Supply:

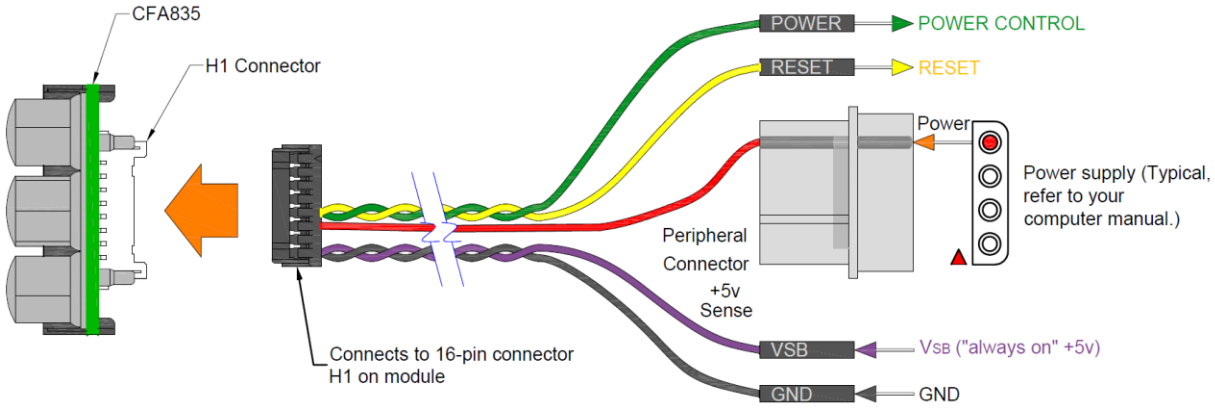


Figure 10. ATX Connection to H1 with WR-PWR-Y25 or WR-PWR-Y38 Cable

8. Firmware

8.1. How to Identify Firmware Revision Number

Before you apply power to the CFA835, 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 you release the right arrow key, the display clears after five seconds.

Or when coming out of restart, keep the right arrow key depressed until the firmware revision displays. As long as the keypad is depressed, this information is displayed. When you release the right arrow key, the display clears after five seconds.

An alternate method to identify revision number is by using command 1 (0x01): Get Module Information.

8.2. Possible Future Firmware Updates

The CFA835 display modules are shipped with preinstalled firmware that performs the command functions described in this Datasheet. We may make updates to the firmware in the future. Firmware updates are announced through our PCN (Part Change Notices).

Any firmware updates will be available as free a download in the “Files” section on the product’s webpage.

Updated firmware is downloaded onto the CFA835 by one of the three methods detailed in Appendix B.

8.3. Create Your Own Firmware or Use PC Update Software

The CFA835 uses an STMicroelectronics STM32F401 microcontroller. You can program the microcontroller by using a JTAG programmer interface.

IMPORTANT: If you load user-created firmware, you will overwrite the Crystalfontz firmware. Functions for the Command Codes described in this Datasheet will not work. There is no method to reinstall the supported firmware without returning the CFA835 to Crystalfontz. A reprogramming charge may apply.

NOTE: Crystalfontz has no phone or email support for user code.

9. Host Communications

To quickly get up and running, download our free demonstration cfTest. cfTest includes all the commands needed to communicate with the CFA835 display module and showcase its functionality.

9.1. Through USB

Windows Operating Systems

The easiest and most common way to communicate with the CFA835 is through USB. A link to VCP drivers download and installation instructions can be found on the Crystalfontz website. WHQL USB drivers are available under the "Datasheets & Files" section on the product's webpage. 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 CFA835 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.

Linux Operating Systems

The CFA835 will appear under Windows as a virtual COM port and under Linux as **`/dev/ttyACMx`**.

9.2. 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 or 9600 baud. Please see command 33 (0x21): Interface Options.

9.3. Multiple Port Communications

The CFA835 supports communication through two interfaces at the same time. Keypad report packets are sent to all available interfaces. All command reply packets are sent to the interface from where the command packet originated.
Packet Structure

All communication between the CFA835 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 CFA835 and the host without the traditional problems that occur in a stream-based serial communication such as having to send data in inefficient ASCII format, to "escape" certain "control characters", or losing sync if a character is corrupted, missing, or inserted.

NOTE: Reconciling packets is recommended rather than using delays when communicating with the LCD module. To reconcile packets, please ensure that the acknowledgement packet has been received from the most recently sent packet before sending any additional packets to the LCD module. This practice will avoid dropped packets or missed communication with the LCD module.

All packets have the following structure:

```
<type><data_length><data><CRC16>
```

`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 CFA835
        01 = normal response from CFA835 to host
        10 = normal report from CFA835 to host
        11 = error response from CFA835 to host
```

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

CRC is a standard 16-bit CRC of all the bytes in the packet except the CRC itself. The CRC is sent LSB first. At the port, the CRC immediately follows the last used element of data [ ]. See Appendix A: Demonstration Software and Sample Code for several examples of how to calculate the CRC in different programming languages.

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;
```

9.4. Packet Error Reporting

The CFA835 supports returning of error packets containing interface and error code information.

See Command 33 for information regarding configuring interfaces.

Error reply packet structure for a standard command is as follows:

```
type = 0xC0 | command-number
data length = 2
data[0] = originating command interface
  0 = serial
  1 = USB
data[1] = ID of extended error information (see table below)
```

Error reply packet structure for a sub-command is as follows:

```
type = 0xC0 | command-number
data length = 2
data[0] = sub-command number
data[1] = originating command interface
  0 = serial
  1 = USB
data[2] = ID of extended error information (see table below)
```

Error #	Description
1	Unknown Error
2	Unknown Command
3	Invalid Command Length/Options
4	Writing Flash Mem Failed
5	Reading Flash Mem Failed
6	CFA-FBSCAB Not Present At Index
7	CFA-FBSCAB Did Not Reply To Reg
8	Micro-SD Not Inserted Or Bad
9	Micro-SD Not Formatted
10	Micro-SD File Could Not Be Found/Opened
11	Micro-SD Unknown Error
12	Micro-SD File Could Not Be Read
13	Micro-SD Could Not Be Written
14	File Header Is Invalid
15	Micro-SD File Is Already Open
16	Micro-SD File Operation Failed
17	Micro-SD File Has Not Been Opened
18	GFX Stream Already Started
19	GFX Is Out Of LCD Bounds
20	Video Is Not Open In Slot
21	GFX Stream Has Timed Out
22	GPIO Not Set For ATX Use
23	Interface Not Enabled
24	Interface Not Available

Figure 11. CFA835 packet error codes table

9.5. About Handshaking

CFA835's packet structure makes traditional hardware or software handshaking unnecessary.

The host should wait for a corresponding acknowledge packet from the CFA835 before sending the next command packet. The CFA835 will respond to all packets within 500 mS. The host software should stop waiting and retry the packet if the CFA835 fails to respond within 500 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 CFA835 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 CFA835 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 CFA835. For any modern PC using reasonably efficient software, this requirement will not pose a challenge.

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

9.6. Command Codes

For your convenience, here is a list of command code links grouped by type. Below this list commands are listed numerically from 1 to 41.

Communications
Command 0 (0x00): Ping Command
Command 1 (0x01): Get Module Information
Command 5 (0x05): Restart includes: Reload Boot Settings Restart Host (WR-PWR-Y25 ATX Power Switch Cable Required) Power Off Host (WR-PWR-Y25 ATX Power Switch Cable Required) CFA835 Restart CFA835 Restore Default Settings
Command 28 (0x1C): ATX Functionality
Command 29 (0x1D): Watchdog
Command 33 (0x21): Interface Options
Command 36 (0x24): Interface Bridge
Display / LCD
Command 6 (0x06): Clear Display
Command 9 (0x09): Special Character Bitmaps
Command 11 (0x0B): Display Cursor Position
Command 12 (0x0C): Cursor Style
Command 13 (0x0D): Contrast
Command 14 (0x0E): Display and Keypad Backlights
Command 31 (0x1F): Write Text to the Display
Command 32 (0x20): Read Text from the Display
Command 38 (0x26): Custom Fonts includes: Subcommand 0: Load Custom Font Files from MicroSD Card Subcommand 1: Print Custom Font to Display
Command 40 (0x28): Display Graphic Options includes: Subcommand 0: Graphic Options Subcommand 1: Buffer Flush Subcommand 2: Send Image Data to Display from Host Subcommand 3: Display Image File from MicroSD Card on CFA835 Subcommand 4: Save Screenshot to MicroSD File Subcommand 5: Pixel Data Subcommand 6: Draw a Line Subcommand 7: Draw a Rectangle Subcommand 8: Draw a Circle
Command 41 (0x29): Video Playback Control includes: Subcommand 0: Load a Video from MicroSD Card Subcommand 1: Video Control

GPIOS and Keypad
Command 14 (0x0E): Display and Keypad Backlights
Command 23 (0x17): Keypad Reporting
Command 24 (0x18): Read Keypad, Polled Mode
Command 28 (0x1C): ATX Functionality includes: Function 1: KEYPAD_RESTART Function 2: KEYPAD_POWER_ON Function 3: KEYPAD_POWER_OFF
Command 34 (0x22): GPIO Pin Levels
Command 37(0x25) Subcommand 5: GPIO Pin Levels
Fan and Temperature Control / Monitoring
Command 37 (0x25): CFA-FBSCAB includes: Subcommand 0: Read CFA-FBSCAB Information Subcommand 1: Fan Settings includes Set Fan Power, Fail-Safe and Glitch information Subcommand 2: Read Fan Tachometers Subcommand 3: Read DOW Device Information Subcommand 4: Read WR-DOW-Y17 Temperature Subcommand 5: GPIO Pin Levels Subcommand 6: Reset and Search
Micro-SD Operations
Command 38 (0x26): Custom Fonts includes: Subcommand 0: Load Custom Font Files from MicroSD Card Subcommand 1: Print Custom Font to Display
Command 39 (0x27): MicroSD File Operations includes: Subcommand 0: Open/Close MicroSD File Subcommand 1: Position Seek Subcommand 2: Read File Data Subcommand 3: Write File Data Subcommand 4: Delete A File
Command 40 (0x28): Display Graphic Options includes: Subcommand 3: Display Image File from MicroSD Card on CFA835 Subcommand 4: Save Screenshot to MicroSD File
Command 41 (0x29), Subcommand 0: Load A Video from MicroSD Card
EEPROM Operations
Command 2 (0x02): Write User Flash Area
Command 3 (0x03): Read User Flash Area
Command 4 (0x04): Store Current State as Boot State

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.

You can experiment with these commands by using our free download of cfTest.

0 (0x00): Ping Command

Used to verify communication with the CFA835. The CFA835 will return the Ping Command to the host.

Command packet:

```
type = 0x00 = 0(10)
data_length = 0 to 124
data[] = any arbitrary data
```

Successful return packet:

```
type = 0x40 | 0x00 = 0x40 = 64(10)
data_length = (identical to received packet)
data[] = (identical to received packet)
```

1 (0x01): Get Module Information

The CFA835 will return the hardware and firmware reversion or serial number to the host.

Command packet:

```
type = 0x01 = 1(10)
data_length = 0 to 1
data[0] = module information to return (optional)
  0 = (optional) hardware and firmware version
  1 = CFA835 module serial number
```

Successful return packet (data_length=0 or data[0]=0):

```
type = 0x40 | 0x01 = 0x41 = 65(10)
data_length = 16
data[] = "CFA835:hX.X,fY.Y"
  hX.X is the hardware revision
  fY.Y is the firmware revision
```

Successful return packet (data[0]=1):

```
type = 0x40 | 0x01 = 0x41 = 65(10)
data_length = 17
data[] = "1134835TMI0000001"
```

2 (0x02): Write User Flash Area

The CFA835 reserves 124 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. This command requires approximately 400mS to complete. The reply packet is returned to the host when the command has completed.

Command packet:

```
type = 0x02 = 2(10)
data_length = 1 to 124
data[] = arbitrary user data to be stored in the CFA835's nonvolatile memory
```

Successful return packet:

```
type = 0x40 | 0x02 = 0x42 = 66(10)
data_length = 0
```

3 (0x03): Read User Flash Area

Command packet:

```
type = 0x03 = 3₁₀
data_length = 1
data[0] = number of bytes of data to be returned (1 to 124)
```

Successful return packet:

```
type = 0x40 | 0x03 = 0x43 = 67₁₀
data_length = number of bytes specified in command
data[] = user data recalled from the CFA835's flash memory
```

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

The CFA835 loads its power-up configuration from nonvolatile memory when power is applied. The CFA835 is configured at the factory to display a boot screen when power is applied. This command requires approximately 400mS to complete. The reply packet is returned to the host when the command has completed. This command can be used to customize the boot screen, as well as the following items:

- Characters shown on display, which are affected by:
- Command 6 (0x06): Clear Display
- Command 31 (0x1F): Write Text to The Display
- Command 38 (0x26), Subcommand 1: Print Custom Font to Display
- Command 9 (0x09): Special Character Bitmaps
- Command 11 (0x0B): Display Cursor Position
- Command 12 (0x0C): Cursor Style
- Command 13 (0x0D): Contrast
- Command 14 (0x0E): Display And Keypad Backlights
- Command 23 (0x17): Keypad Reporting
- Command 28 (0x1C): ATX Functionality
- Command 33 (0x21): Interface Options
- Command 34 (0x22): GPIO Pin Levels
- Command 37 (0x25): CFA-FBSCAB

All CFA-FBSCAB settings are also saved but are saved in the nonvolatile memory on the CFA-FBSCAB module itself.

Watchdog settings cannot be saved. The host software should enable these items once the system is initialized and ready to receive the data.

Command packet:

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

Successful return packet:

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

5 (0x05): Restart

Based on provided parameters, this command provides five reset functions: (1) Reload Boot Settings, (2) Restart Host, (3) Power Off Host, (4) CFA835 Restart, or (5) CFA835 Restore Default Settings.

When using both the USB and a serial interface simultaneously (logic level or “full swing” RS232 with mounted optional CFA-RS232 Serial Converter Board), you may notice that performing a restart from one interface will impact the other interface.

The ATX options to power down or restart the host using the CFA835 may be useful in many situations. These options rely on the GPIO pins used for ATX control to be configured in their default drive modes in order for the ATX functions to work correctly. Please see command 28 (0x1C): ATX Functionality.

(1) Reload Boot Settings

Reloads the settings stored using command 4 (0x04): Store Current State as Boot State. Reloading the boot settings may be useful when testing the boot configuration.

The CFA835 will return the acknowledgment packet immediately, then reload its settings.

Reloading of settings takes approximately 100mS. During this time, any data sent to the CFA835 will be disregarded.

Command packet:

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

(2) Restart Host (WR-PWR-Y25 ATX Power Switch Cable Required)

This option instructs the CFA835 to restart the host via the WR-PWR-Y25 ATX power switch cable and then restart itself. This command will also restart any attached CFA-FBSCAB modules to the state saved in their nonvolatile memory.

The CFA835 will return the acknowledge packet before carrying out the actions.

Command packet:

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

(3) Power Off Host (WR-PWR-Y25 ATX Power Switch Cable Required)

This option instructs the CFA835 to power down the host via the WR-PWR-Y25 ATX power switch cable and then restart itself. This command will also restart any attached CFA-FBSCAB modules to the state saved in their nonvolatile memory.

Command packet:

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

(4) CFA835 Restart

Performs a software restart of the CFA835 module. This command will also restart any attached CFA-FBSCAB modules to the state saved in their nonvolatile memory.

The CFA835 will return the acknowledge packet immediately, then restart itself. The CFA835 may not respond to new command packets for up to 3 seconds.

If used with the USB (virtual COM port) interface, this command will cause the CFA835 module to disconnect and then reconnect (re-enumerate). Software running on the host may need to close, and re-open the virtual COM port for communications to resume.

Command packet:

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

(5) CFA835 Restore Default Settings

Restarts the system boot state to that of a factory CFA835 and then performs a CFA835 restart. This command will also restart any attached CFA-FBSCAB to the state saved in their nonvolatile memory.

This option does not affect the user flash values set by command 2 (0x02): Write User Flash Area.

The CFA835 will return the acknowledge packet immediately, then restart itself. The CFA835 may not respond to new command packets for up to 3 seconds.

If used with the USB (virtual COM port) interface, this command will cause the CFA835 module to disconnect and then reconnect (re-enumerate). Software running on the host may need to close, and re-open the virtual COM port for communications to resume.

Command packet:

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

Successful return packet for all restart options:

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

6 (0x06): Clear Display

Clears the CFA835's display, graphical display buffer, and character row/column buffer. It also moves the cursor to the left-most column of the top line and stops any videos that are being played from a microSD card. Please see command 41 (0x3A): Video Playback Control.

Command packet:

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

Successful return packet:

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

9 (0x09): Special Character Bitmaps

Sets the bitmap for one of the special characters in the CGRAM to be used with command 31 (0x1F): Write Text to the Display.

NOTE: Special characters are not supported when using custom fonts. See command 38, Subcommand 0: Load Custom Font Files from MicroSD Card for details.

Command packet (Read):

```
type = 0x09 = 9₁₀
data_length = 1
data[0] = index of special character to read (0-7 valid)
```

Successful return packet (Read):

```
type = 0x40 | 0x09 = 0x49 = 73₁₀
data_length = 9
data[0] = index of special character data
data[1-8] = bitmap of this special character
```

Command packet (Write):

```
type = 0x09 = 9₁₀
data_length = 9
data[0] = index of special character to modify (0-7 valid)
data[1-8] = bitmap of this special character
```

Successful return packet (Write):

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

11 (0x0B): Display Cursor Position

This command allows the cursor to be placed at the desired location on the CFA835's LCD screen. If you want the cursor to be visible, you may also need to send a command 12 (0x0C): Cursor Style. The current cursor location can also be read using this command.

Command packet (Read):

```
type = 0x0B = 11₁₀
data_length = 0
```

Successful return packet (Read):

```
type = 0x40 | 0x0B = 0x4B = 75₁₀
data_length = 2
data[0] = column
data[1] = row
```

Command packet (Write):

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

Successful return packet (Write):

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

12 (0x0C): Cursor Style

This command allows you to either hide the cursor or select among four hardware generated cursor options. You can also read the current cursor style using this command.

Cursor Styles:

```
0 = no cursor
1 = blinking block cursor
2 = underscore cursor
3 = blinking block plus underscore
4 = inverting, blinking block
```

Command packet (Read):

```
type = 0x0C = 12₁₀
data_length = 0
```

Successful return packet (Read):

```
type = 0x40 | 0x0C = 0x4C = 76₁₀
data_length = 1
data[0] = cursor style
```

Command packet (Write):

```
type = 0x0C = 12₁₀
data_length = 1
data[0] = cursor style
```

Successful return packet (Write):

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

13 (0x0D): Contrast

This command sets the contrast of the display. This command can also be used to read the current display contrast.

Command packet (Read):

```
type = 0x0D = 13₁₀
data_length = 0
```

Successful return packet (Read):

```
type = 0x40 | 0x0D = 0x4D = 77₁₀
data_length = 1
data[0] = contrast setting (0-255 valid)
```

Command packet (Write):

```
type = 0x0D = 13₁₀
data_length = 1
data[0] = contrast setting (0-255 valid)
  0-111 = very light
  112 = light
  127 = about right
  168 = dark
  169-255 = very dark (may be useful at cold temperatures)
```

Successful return packet (Write):

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

14 (0x0E): Display and Keypad Backlights

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

If two bytes is supplied, the display is set to the brightness of the first byte, the keypad is set to the brightness of the second byte. This command can also be used to read the current brightness levels.

If one byte is supplied, both the keypad and display backlights are set to that brightness.

Command packet (Read):

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

Successful return packet (Read):

```
type = 0x40 | 0x0E = 0x4E = 78₁₀
data_length = 2
data[0] = current display brightness (0-100)
data[1] = current keypad brightness (0-100)
```

Command packet (Write):

```
type = 0x0E = 14(10)
data_length = 1 or 2
data[0] = display backlight brightness (0-100 valid)
  0 = off
  1-100 = variable brightness
data[1] = keypad backlight power (0-100 valid)
  0 = off
  1-100 = variable brightness
```

Successful return packet (Write):

```
type = 0x40 | 0x0E = 0x4E = 78(10)
data_length = 0
```

23 (0x17): Keypad Reporting

By default, the CFA835 reports any key event to the host. This command allows the key events to be enabled or disabled on an individual basis. This command can also be used to read the current key reporting masks.

Keypad Bitmasks:

```
bit0 - up key
bit1 - enter key
bit2 - cancel key
bit3 - left key
bit4 - right key
bit5 - down key
```

Command packet (Read):

```
type = 0x17 = 23(10)
data_length = 0
```

Successful return packet (Read):

```
type = 0x40 | 0x17 = 0x57 = 87(10)
data_length = 2
data[0] = current keypad press mask
data[1] = current keypad release mask
```

Command packet (Write):

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

Successful return packet (Write):

```
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 CFA835 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): 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.

Keypad Bitmasks:

```
bit0 - up key
bit1 - enter key
bit2 - cancel key
bit3 - left key
bit4 - right key
bit5 - down key
```

Command packet:

```
type = 0x18 = 24₁₀
data_length = 0
```

Successful return packet:

```
type = 0x40 | 0x18 = 0x58 = 88₁₀
data_length = 3
data[0] = bitmask indicating the keys currently pressed
data[1] = bitmask indicating the keys pressed since the last poll
data[2] = bitmask indicating the keys released since the last poll
```

28 (0x1C): ATX Functionality

The combination of the CFA835 with ATX can be used to replace the function of the power and restart 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.

NOTE: The GPIO pins used for ATX control must not be configured as user GPIO. The pins must be configured to their default drive mode in order for the ATX functions to work correctly. Please see ATX Power Supply and Control Connections.

The RESET (GPIO[3]) and POWER CONTROL (GPIO[2]) lines on the CFA835 are normally high-impedance. Electrically, they appear to be disconnected or floating. When the CFA835 asserts the RESTART or POWER CONTROL lines, they are momentarily driven high or low (as determined by the RESTART_INVERT and POWER_INVERT bits, detailed below). To end the power or restart pulse, the CFA835 changes the lines back to high-impedance.

FOUR FUNCTIONS MAY BE ENABLED BY COMMAND 28:

Function 1: KEYPAD_RESTART

If POWER-ON SENSE (GPIO[1]) is high, holding the green check key for 4 seconds will pulse RESTART (GPIO[3]) pin for 1 second. During the 1-second pulse, the CFA835 will show "RESTART", and then the CFA835 will reset itself, showing its boot state as if it had just powered on. Once the pulse has finished, the CFA835 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 CFA835 will show "POWER ON", then the CFA835 will reset itself.

Function 3: KEYPAD_POWER_OFF

If POWER-ON SENSE (GPIO[1]) is high, holding the red X key for 4 seconds will pulse POWER CONTROL (GPIO[2]) for the duration specified by in data[1] or the default of 1 second. If the user continues to hold the power key down, then the CFA835 will continue to drive the line for a maximum of 5 additional seconds. During this time, the CFA835 will show "POWER OFF".

Function 4: MODULE_MIMIC_HOST_POWER

If MODULE_MIMIC_HOST_POWER is set, the CFA835 will blank its screen and turn off its backlight to simulate its power being off any time POWER-ON SENSE (GPIO[1]) is low. The CFA835 will still be active (since it is powered by V_{SB}), 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 CFA835 will restart as if power had just been applied to it.

ATX Bitmasks:

```
bit0 - AUTO_POLARITY: Automatically detects polarity for restart and power
       (recommended)
bit1 - RESTART_INVERT: Restart pin drives high instead of low (ignored if
       AUTO_POLARITY is set)
bit2 - POWER_INVERT: Power pin drives high instead of low (ignored if
       AUTO_POLARITY is set)
bit3 - LEDS_MIMIC_HOST_POWER: Turn off the LEDs also if the host is off
       (ignored if MODULE_MIMIC_HOST_POWER is not set)
bit4 - MODULE_MIMIC_HOST_POWER: Turn off the display if the Host is off
bit5 - KEYPAD_RESTART
bit6 - KEYPAD_POWER_ON
bit7 - KEYPAD_POWER_OFF
```

Command packet (Read):

```
type = 0x1C = 28₁₀
data_length = 0
```

Successful return packet (Read):

```
type = 0x40 | 0x1C = 0x5C = 92₁₀
data_length = 2
data[0] = bitmask of enabled functions
data[1] = length of power on & off pulses in 1/32 second increments
```

Command packet (Write):

```
type = 0x1C = 28₁₀
data_length = 1 or 2
data[0] = bitmask of enabled functions
data[1] = length of power on & off pulses in 1/32 second increments (optional)
  1 = 1/32 second
  2 = 1/16 second
  16 = 1/2 second
  ...
  254 = 7.9 second
  255 = Hold until power sense change or 8 second, whichever is shorter
(default)
```

Successful return packet (Write):

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

29 (0x1D): 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 CFA835 with ATX (CFA835+WR-PWR-Y25 ATX power switch cable).

If the command is not reissued within the specified number of seconds, then the CFA835 with ATX will restart the host system (see command 28 (0x1C): ATX Functionality for details) and restart itself as if command 5 (0x05): Restart function was issued. Since the watchdog is off by default when it powers up, CFA835 with ATX will not issue another host restart until the host has once again enabled the watchdog.

To turn the watchdog off once it has been enabled, set data [0] = 0.

NOTE: The GPIO pins used for ATX control must not be configured as user GPIO. They must be configured to their default drive mode in order for the ATX functions to work correctly. These settings are factory default, but may be changed by the user. See the note under command 28 (0x1C): ATX Functionality or command 34 (0x22): GPIO Pin Levels.

Command packet (Read):

```
type = 0x1D = 29₁₀
data_length = 0
```

Successful return packet (Read):

```
type = 0x40 | 0x1D = 0x5D = 93₁₀
data_length = 1
data[0] = watchdog timeout in seconds (0=disabled)
```

Command packet (Write):

```
type = 0x1D = 29₁₀
data_length = 1
data[0] = enable counter
  0 = watchdog is disabled
  1-255 = timeout in seconds
```

Successful return packet (Write):

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

31 (0x1F): Write Text to the Display

This command allows text and special characters to be placed at any position on the display. The text is displayed in the default font, unless overridden by command 38, Subcommand 0: Load Custom Font Files from MicroSD Card. See default font standard set of characters at CHARACTER GENERATOR ROM (CGROM).

Command packet:

```
type = 0x1F = 31(10)
data_length = 3 to 22
data[0] = column position (x = 0 to 19)
data[1] = row position (y = 0 to 3)
data[2-21] = text to place on the LCD, variable from 1 to 20 characters
```

Successful return packet:

```
type = 0x40 | 0x1F = 0x5F = 95(10)
data_length = 0
```

32 (0x20): Read Text from the Display

The CFA835 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 CFA835 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 CFA835 to power up at the new baud rate.

This command allows the host to read back text that is displayed on the CFA835.

NOTE: This command will only read text displayed by command 31 (0x1F): Write Text to the Display. It cannot be used to read text written by custom font command 38, Subcommand 0: Load Custom Font Files from MicroSD.

Command packet:

```
type = 0x20 = 32(10)
data_length = 3
data[0] = column position (x = 0 to 19)
data[1] = row position (y = 0 to 3)
data[2] = length of text to read in characters (1 - 20)
```

Successful return packet:

```
type = 0x40 | 0x20 = 0x60 = 9610
data_length = 1 to 20
data[] = read text
```

33 (0x21): Interface Options

The CFA835 has a logic level serial interface located on pins 1 (Tx) and 2 (Rx) of the H1 connector. For “full swing” RS232 using the optional CFA-RS232 Serial Converter Board, the H1 pins are passed through to the CFA-RS232’s J1 connector.

After sending this command, the host should wait for a positive acknowledgment from the CFA835 at the old baud rate. The host can then begin communicating at the new baud rate.

The baud rate must be saved by command 4 (0x04): Store Current State as Boot State if you want the CFA835 to power-up/restart using the new baud rate. The factory default baud rate is 115200.

This command is also used to read the current interface options.

Option flags:

```
bit0 = enable interface
       NOTE: USB interface cannot be fully disabled
bit1 = command interpreter enabled
       NOTE: CFA835 will accept packets on this interface. interface must be
     enabled for interpreter on an interface to be enabled. normal reply packets
     are only sent to the originating interface. the following options are only
     available if the interpreter is enabled
bit2 = CFA835 will transmit report packets on this interface (reports 128)
bit3 = CFA835 will transmit errors from commands received on this interface
bit4 = CFA835 will transmit errors from commands received on either interface
```

Command packet (Read):

```
type = 0x21 = 33₁₀
data_length = 1
data[0] = interface
  0 = serial
  1 = USB
```

Successful return packet (Read):

```
type = 0x40 | 0x21 = 0x61 = 97₁₀

SERIAL INTERFACE:
data_length = 3
data[0] = 0 (serial)
data[1] = option flags
data[2] = baud rate
  0 = 19200
  1 = 115200
  2 = 9600

USB INTERFACE:
data_length = 2
data[0] = 1 (USB)
data[1] = option flags
```

Command packet (Write):

```
type = 0x21 = 33₁₀
data_length = 2 or 3
data[0] = interface
  0 = serial
  1 = USB
data[1] = option flags
data[2] = baud rate (serial only)
  0 = 19200
  1 = 115200
  2 = 9600
```

Successful return packet (Write):

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

34 (0x22): GPIO Pin Levels (including LEDs)

The CFA835 has five pins for user-definable general-purpose input / output (GPIO). These pins are shared with the ATX functions (see the note below).

This command also sets/configures the PWM duty used to control the four bi-color on-module LEDs.

The architecture of the CFA835 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.

The CFA835 continuously polls the GPIOs as inputs at 50 Hz. The present level can be queried by the host software at a lower rate. The CFA835 also keeps track of whether there were rising or falling edges since the last host query (subject to the resolution of the 50 Hz sampling). This means that the host is not forced to poll quickly in order to detect short events. The algorithm used by the CFA835 to read the inputs is inherently “debounced”.

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

Pull-up/pull-down resistance values are approximately 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 website for STM32F401 datasheets

NOTE: The GPIO pins may also be used for ATX control through the H1 connector using the WR-PWR-Y25 ATX power switch cable. By factory default, the GPIO output setting, function, and drive mode are set correctly to enable operation of the ATX function. **The GPIO output setting, function, and drive mode must be set to the correct values in order for the ATX function to function properly.**

Our free demonstration software cfTest may be used to easily check and restart the GPIO configuration to the default state so the ATX and DOW WR-DOW-Y17 temperature sensor cable functions will work.

Command packet (Read):

```
type = 0x22 = 34₁₀
data_length = 1
data[0] = index of GPIO/GPO to read (0-12 valid)
```

Successful return packet (Read):

```
type = 0x40 | 0x22 = 0x62 = 98₁₀
data_length = 4
data[0] = index of GPIO/GPO data[1] = pin output state
data[2] = pin PWM output value
data[3] = pin function select and drive mode
```

Command packet (Write):

```
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 (0-12 valid)
  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
data[1] = Pin output state (behavior depends on drive mode)(0-100 valid):
  0 = Output set to low
  1-99 = Output duty cycle percentage (100 Hz nominal)
  100 = Output set to high
data[2] = Pin function select and drive mode (optional)
  Only meaningful for GPIOs (index 0-4). GPIO (index of 5-12) will ignore.
  ---- FDDD
  |||| ||||-- DDD = Drive Mode (based on output state of 1 or 0)
  |||| |      ========================================================
  |||| |      000: 1=strong drive up, 0=resistive pull down
  |||| |      001: 1=strong drive up, 0=fast, strong drive down
  |||| |      010: hi-z, use for input
  |||| |      011: 1=resistive pull up, 0=strong drive down
  |||| |      100: 1=strong drive up, 0=hi-z
  |||| |      101: 1=strong drive up, 0=strong drive down
  |||| |      110: reserved, do not use - error returned
  |||| |      111: 1=hi-z, 0=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:

```
type = 0x40 | 0x22 = 0x62 = 98(10)
data_length = 0
```

36 (0x24): Interface Bridge

The CFA835 has two interfaces: USB and a serial interface (logic level or “full swing” RS232 with mounted optional CFA-RS232).

By default, all interfaces on the CFA835 have the command interpreter enabled and are used by the host (or hosts) to send/receive command packets to/from the CFA835. If the command interpreter is disabled for an interface using command 33 (0x21): Interface Options, that interface can be used to forward and receive raw data using this command.

For example, a host connected to the CFA835's USB interface could send raw data to the serial interface buffer. Incoming raw data on the serial interface is buffered and can be read from the buffer using the USB interface.

NOTE: This command will return an error if the interface being written to or read from has the command interpreter enabled.

Serial Interface

If the command interpreter is turned off, incoming bytes will be buffered in a circular buffer. If the buffer is allowed to wrap, it will overwrite the oldest data first. If the circular buffer does wrap, the next write/read command response will have the buffer overflow flag set. **`data[1]`** is treated as a timeout and the CFA835 will wait this long for the specified amount of data before aborting and throwing an error.

USB Interface

The USB host interface has flow control if the CFA835's incoming USB data buffer becomes full, the CFA835 will request the host not to send any more data. The overflow flag will never be set.

Command packet:

```
type = 0x24 = 36₁₀
data_length = 4 + write data length
data[0] = interface
  0 = serial
  1 = USB
data[1] = delay/timeout
  0 = no delay/timeout, only return data that is already in the buffer
  1 to 50 = time in milliseconds / 10 (up to a value of 500mS)
data[2] = clear receive buffer options
  0x0 = do not clear
  0x1 = clear before read
  0x2 = clear after read
  0x3 = clear before and after
data[3] = requested read bytes
data[4-123] = data to be written to specified interface
```

Successful return packet:

```
type = 0x40 | 0x24 = 0x64 = 100₁₀
data_length = 2 + read data length
data[0] = interface
data[1] = interface buffer status flags
   bit 0 = buffer overflow
   bit 1 = more data is available
data[2-123] = data read from interface buffer
```

NOTE: If there are fewer bytes available in the circular buffer than are requested, a smaller amount of data may be returned, as indicated by the read data length.

37 (0x25): CFA-FBSCAB Command Group

The CFA835 supports fans, temperature sensors, and additional GPIOs through the addition of one or more CFA-FBSCABs. This command group contains all of the subcommands necessary to interact with the attached CFA-FBSCABs including reading and writing from the CFA-FBSCAB's fans, temperature sensors, and GPIO pins. As many as 32 CFA-FBSCABs can be attached by daisy-chaining them with WR-EXT-Y37 communication cables.

The combination of the CFA835 + one or more CFA-FBSCABs can be used as part of an active cooling system. The fans can be slowed down to reduce noise when a system is idle or when the ambient temperature is low. The fans speed up when the system is under heavy load or the ambient temperature is high. The host system controls the attached fans power using sub-command 1.

See the sub-commands below for detailed information on FBSCAB operations.

Subcommand 0: Read CFA-FBSCAB Information

This subcommand returns the quantity of CFA-FBSCABs detected by the CFA835 or the serial number of a specified CFA-FBSCAB.

Command packet (Query Number Of CFA-FBSCABs):

```
type = 0x25 = 37₁₀
data_length = 1
data[0] = 0 (read FBSCAB information)
```

Successful return packet (Query Number Of CFA-FBSCABs):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 2
data[0] = 0 (read FBSCAB information)
data[1] = number of attached FBSCABs
```

Command packet (Query CFA-FBSCAB Serial Number):

```
type = 0x25 = 37₁₀
data_length = 2
data[0] = 0 (Read FBSCAB Information)
data[1] = FBSCAB index
```

Successful return packet (Query CFA-FBSCAB Serial Number):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 18
data[0] = 0 (read FBSCAB Information)
data[1] = index of queried FBSCAB
data[2-18] = serial number of specified FBSCAB module (text)
```

Subcommand 1: Fan Settings

This command will configure or read the power settings for the fan connectors on the specified CFA-FBSCAB module.

Fan power is controlled by PWM switching the fan's power supply at approximately 18Hz.

A fan power control fail-safe system is provided, and controlled by this sub-command. If the fail-safe bit for a fan is enabled, and the fan power level is not update by the host system using this sub-command within the time-out period, the fans with the fail-safe bit enabled will have the power level set to 100% until this sub-command packet is received.

This command also allows you to set a variable-length delay (glitch delay) after the fan has been turned on before the CFA835 will recognize transitions on the tachometer line. Some fans require a longer delay for the module to reliably read the tachometer output. 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.

Typically, you would

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

(2) reduce it until the problem reappears, and then

(3) slightly increase the delay count to give it some margin.

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

Command packet (Set Fan Power):

```
type = 0x25 = 37₁₀
data_length = 6
data[0] = 1 (Set/Read FBSCAB Fan Settings)
data[1] = FBSCAB module index
data[2] = power level for FAN 1 (0-100 valid)
data[3] = power level for FAN 2 (0-100 valid)
data[4] = power level for FAN 3 (0-100 valid)
data[5] = power level for FAN 4 (0-100 valid)
```

Successful return packet (Set Fan Power):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 1
data[0] = 1 (Set/Read FBSCAB Fan Settings)
```

Command packet (Set Fan Power and Fail-Safe):

```
type = 0x25 = 37₁₀
data_length = 8
data[0] = 1 (Set/Read FBSCAB Fan Settings)
data[1] = FBSCAB module index
data[2] = power level for FAN 1 (0-100 valid)
data[3] = power level for FAN 2 (0-100 valid)
data[4] = power level for FAN 3 (0-100 valid)
data[5] = power level for FAN 4 (0-100 valid)
data[6] = fail-safe enabled for these fans’ bitmask
data[7] = fan power update must happen within this many 1/8 second periods
```

Successful return packet (Set Fan Power and Fail-Safe):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 1
data[0] = 1 (Set/Read FBSCAB Fan Settings)
```

Command packet (Set Fan Power, Fail-Safe and Glitch):

```
type = 0x25 = 37₁₀
data_length = 12
data[0] = 1 (Set/Read FBSCAB Fan Settings)
data[1] = FBSCAB module index
data[2] = power level for FAN 1 (0-100 valid)
data[3] = power level for FAN 2 (0-100 valid)
data[4] = power level for FAN 3 (0-100 valid)
data[5] = power level for FAN 4 (0-100 valid)
data[6] = fail-safe enabled for these fans bitmask
data[7] = fan power update must happen within this many 1/8 second periods
data[8] = glitch delay for FAN 1 (1-100 valid)
data[9] = glitch delay for FAN 2 (1-100 valid)
data[10] = glitch delay for FAN 3 (1-100 valid)
data[11] = glitch delay for FAN 4 (1-100 valid)
```

Successful return packet (Set Fan Power, Fail-Safe and Glitch):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
```

```
data_length = 1
data[0] = 1 (Set/Read FBSCAB Fan Settings)
```

Command packet (Read Fan Settings):

```
type = 0x25 = 37₁₀
data_length = 2
data[0] = 1 (Set/Read FBSCAB Fan Settings)
data[1] = FBSCAB module index
```

Successful return packet (Read Fan Settings):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 12
data[0] = 1 (Set/Read FBSCAB Fan Settings)
data[1] = FBSCAB module index
data[2] = power level for FAN 1
data[3] = power level for FAN 2
data[4] = power level for FAN 3
data[5] = power level for FAN 4
data[6] = fail-safe enabled for these fans bitmask
data[7] = fan power update 1/8 second periods
data[8] = glitch delay for FAN 1
data[9] = glitch delay for FAN 2
data[10] = glitch delay for FAN 3
data[11] = glitch delay for FAN 4
```

Subcommand 2: Read Fan Tachometers

This command will read the last fan tachometer's information from the specified CFA-FBSCAB module. See Appendix A: Sample Code for RPM Calculation Information.

NOTE: If fan tachometer readings are unstable, or unreliable, see sub-command 1 information on setting the fan glitch-filter.

NOTE: This command must be executed every 60 seconds or less to read fan speed information from a CFA-FBSCAB module. If the command is not re-executed within 60 seconds, fan speed readings will be disabled by the CFA835 (to reduce fan noise) until the next "Read Fan Tachometers" subcommand is issued.

Command packet:

```
type = 0x25 = 37₁₀
data_length = 2
data[0] = 2 (read fan tachometer speed)
data[1] = FBSCAB module index
```

Successful return packet:

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 14
data[0]:2 (read fan tachometer speed)
data[1]:FBSCAB module index
data[2]:fan 1 number of fan tach cycles
data[3]:fan 1 LSB of fan timer ticks
data[4]:fan 1 MSB of fan timer ticks
data[5]:fan 2 number of fan tach cycles
data[6]:fan 2 LSB of fan timer ticks
```

```
data[7]:fan 2 MSB of fan timer ticks
data[8]:fan 3 number of fan tach cycles
data[9]:fan 3 LSB of fan timer ticks
data[10]:fan 3 MSB of fan timer ticks
data[11]:fan 4 number of fan tach cycles
data[12]:fan 4 LSB of fan timer ticks
data[13]:fan 4 MSB of fan timer ticks
```

Subcommand 3: Read DOW Device Information

This command returns the ROM ID of the specified DOW (Dallas one wire) device attached to the specified CFA-FBSCAB module.

Command packet:

```
type = 0x25 = 37[10]
data_length = 3
data[0] = 3 (read DOW device information)
data[1] = FBSCAB module index
data[2] = DOW device index (0-15)
```

Successful return packet:

```
type = 0x40 | 0x25 = 0x65 = 101[10]
data_length = 11
data[0] = 3 (read DOW device information)
data[1] = FBSCAB module index
data[2] = DOW device index
data[3-10] = DOW ROM ID
```

Subcommand 4: Read DOW Temperature Sensor Value

This command will return the temperature of the specified DOW (Dallas one wire) device on the specified CFA-FBSCAB module.

The specified DOW device must be of type 0x22 or 0x28 as read by command 37, sub-command 3.

Type 0x22 = Maxim DS18B20 sensor (as used by Crystalfontz WR-DOW-Y17)
Type 0x28 = Maxim DS1822 sensor

Command packet:

```
type = 0x25 = 37[10]
data_length = 3
data[0] = 4 (read WR-DOW-Y17 temperature)
data[1] = FBSCAB module index
data[2] = DOW device index (0-15)
```

Successful return packet:

```
type = 0x40 | 0x25 = 0x65 = 101[10]
data_length = 5
data[0] = 4 (read WR-DOW-Y17 temperature)
data[1] = FBSCAB module index
data[2] = DOW device index (0-15)
data[3] = LSB of temperature data
data[4] = MSB of temperature data
```

Temperature Data (MSB/LSB) Return Format:

```
cc ss s ttt tttt tttt
|| || | ||| |||| ||||-- 11 bit temperature value in degrees C * 16
|| || |----------------- Sign extension (2's complement)
||-------------------- DOW_CRC_status:
1 means CRC was checked and passed
2 means CRC was checked and failed
10 means no sensor detected in this slot
11 means valid sensor but no data yet
```

Subcommand 5: GPIO Pin Levels

The architecture of the CFA-FBSCABs 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 CFA-FBSCAB continuously polls the GPIOs as inputs. The present level can be queried by the host software at a lower rate. The CFA-FBSCAB also keeps track of whether there were rising or falling edges since the last host query (subject to the resolution of the 50 Hz sampling). This means that the host is not forced to poll quickly in order to detect short events. The algorithm used by the CFA-FBSCABs to read the inputs is inherently “debounced”.

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

Pull-up/pull-down resistance values are approximately 40kΩ. Typical GPIO current limits when sinking or sourcing all five GPIO pins simultaneously are 8 mA.

NOTE: Do not confuse FBSCAB GPIOs with the GPIOs available on the CFA835 module itself. This sub-command controls only the selected FBSCAB’s GPIOs. To use the CFA835 module GPIOs see command 34.

Command packet (Set Pin Value):

```
type = 0x25 = 37₁₀
data_length = 4
data[0] = 5 (Set/Read GPIO Pin Configuration & Value)
data[1] = FBSCAB module index
data[2] = index of GPIO to modify
  0 = GPIO[0] = J8, Pin 7
  1 = GPIO[1] = J8, Pin 6
  2 = GPIO[2] = J8, Pin 5
  3 = GPIO[3] = J8, Pin 4
  4 = GPIO[4] = J9, Pin 2 (DOW I/O, always has 1K hardware pull-up)
data[3] = pin output state (behavior depends on drive mode):
  0 = output set to low
  1-99 = output duty cycle percentage (100Hz nominal)
  100 = output set to high
  101-255 = invalid
```

Successful return packet (Set Pin Value):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 0
```

Command packet (Set Pin Value & Configuration):

```
type = 0x25 = 37₁₀
data_length = 5
data[0] = 5 (Set/Read GPIO Pin Configuration & Value)
data[1] = FBSCAB module index
data[2] = index of GPIO to modify
  0 = GPIO[0] = J8, pin 7
  1 = GPIO[1] = J8, pin 6
  2 = GPIO[2] = J8, pin 5
  3 = GPIO[3] = J8, pin 4
  4 = GPIO[4] = J9, pin 2 (DOW I/O, always has 1K hardware pull-up)
data[3] = pin output state (behavior depends on drive mode):
  0 = output set to low
  1-99 = output duty cycle percentage (100Hz nominal)
  100 = output set to high
  101-255 = invalid
data[4] = pin function select and drive mode
  ---- FDDD
  |||| ||||-- DDD = Drive Mode (based on output state of 1 or 0)
  |||| |      ========================================================
  |||| |      000: 1=strong drive up, 0=resistive pull down
  |||| |      001: 1=strong drive up, 0=fast, strong drive down
  |||| |      010: hi-z, use for input
  |||| |      011: 1=resistive pull up, 0=strong drive down
  |||| |      100: 1=strong drive up, 0=hi-z
  |||| |      101: 1=strong drive up, 0=strong drive down
  |||| |      110: reserved, do not use - error returned
  |||| |      111: 1=hi-z, 0=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 (Set Pin Value & Configuration):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 0
```

Command packet (Read Pin Value & Configuration):

```
type = 0x25 = 37₁₀
data_length = 3
data[0] = 5 (Set/Read GPIO Pin Configuration & Value)
data[1] = FBSCAB module index
```

```
data[2] = index of GPIO
```

Successful return packet (Read Pin Value & Configuration):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 6
data[0] = 5 (Set/Read GPIO Pin Configuration & Value)
data[1] = FBSCAB module index
data[2] = index of GPIO
data[3] = pin state & changes since last poll
  ---- -RFS
  |||| ||||-- S = state at the last reading
  |||| |||--- F = a falling edge has been detected since the last poll
  |||| ||---- R = a rising edge has been detected since the last poll
  |||| |----- reserved
data[4] = requested pin level/PWM level
data[5] = pin function select and drive mode
  ---- FDDD
  |||| ||||-- DDD = Drive Mode (based on output state of 1 or 0)
  |||| |      ========================================================
  |||| |      000: 1=strong drive up, 0=resistive pull down
  |||| |      001: 1=strong drive up, 0=fast, strong drive down
  |||| |      010: hi-z, use for input
  |||| |      011: 1=resistive pull up, 0=strong drive down
  |||| |      100: 1=strong drive up, 0=hi-z
  |||| |      101: 1=strong drive up, 0=strong drive down
  |||| |      110: reserved, do not use - error returned
  |||| |      111: 1=hi-z, 0=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

NOTE: The reported pin state is the actual pin state, which may or may not
agree with the pin setting, depending on drive mode and the load presented by
external circuitry. The pins are polled at approximately 32Hz asynchronously
with respect to this command. Transients that happen between polls will not be
detected.
```

Subcommand 6: Reset and Search

This command sends a reset instruction to all attached CFA-FBSCAB modules. This will revert the CFA-FBSCAB modules back to their saved power-on state. After the reset instructions have been sent, the CFA835 re-searches for attached CFA-FBSCAB modules.

NOTE: For one attached CFA-FBSCAB, this command takes approximately 400 mS to complete and return the response packet. If multiple CFA-FBSCABs are attached, searching may take longer, up to 2 additional seconds.

Command packet:

```
type = 0x25 = 37₁₀
data_length = 1
data[0] = 6 (Reset & Search)
```

Successful return packet:

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 1
data[0] = 6 (Reset & Search)
```

38 (0x26): Custom Fonts Command Group

The CFA835 is the first in our intelligent product line with a monochrome graphic LCD. It supports printing text using most any custom font in most any language. To support this exciting new functionality, we've developed a utility to convert fonts to the new CFA835 font structure. Using this utility, fonts can be created from scratch or imported from the Windows library and modified for export. Custom fonts can then be transferred to the CFA835 using the on-board microSD card. The CFA835 supports using up to 4 custom fonts simultaneously.

Subcommand 0: Load Custom Font Files from MicroSD Card

This command loads custom font files from the inserted microSD card. Your custom font files must be created using the CFA835 Font Editor. The loaded font is printed to the display using the subcommand immediately below, Subcommand 1: Print Custom Font to Display.

The CFA835 supports using up to 4 individual custom font files at a time (four “slots”).

User defined characters as set by command 9 (0x09): Special Character Bitmaps are not supported by this command or the subcommand immediately below, Subcommand 1: Print Custom Font to Display.

Command 31 (0x1F): Write Text to the Display supports a special replacement mode using a custom font. Replacement mode is activated by loading a custom font into slot 0 with **`data[2]`**:bit 1 set to 1.

To disable replacement mode, load a custom font into slot 0 with **`data[2]`**:bit 1 set to 0.

Replacement mode can only use a custom font in slot 0; attempting to set **`data[2]`**:bit 1 for a custom font loaded in any other slot will throw an error.

Command packet:

```
type = 0x26 = 38₁₀
data_length = 4 to 124
data[0] = 0 (Load Custom Font Files From MicroSD Card)
data[1] = font slot (0 to 3)
data[2] = option flags
  bit 0 = forced monospace (ignore proportional flag in font file header).
  bit 1 = use font for command 31 (utf-8 only, must be a monospace font or
forced monospace)
  bit 2 = 0=utf-8, 1=utf-16
data[3-123] = file name of the font file located on the microSD card
```

Successful return packet:

```
type = 0x40 | 0x26 = 0x46 = 102₁₀
data_length = 1
data[0] = 0 (Load Custom Font Files From MicroSD Card)
```

Subcommand 1: Print Custom Font to Display

This command prints the specified string to the display using the font slot set by the subcommand immediately above, Subcommand 0: Load Custom Font Files from MicroSD Card.

Command packet:

```
type = 0x26 = 38₁₀
data_length = 4 to 124
data[0] = 1 (Print Custom Font to Display)
data[1] = font slot (0 to 3)
data[2] = character placement style
  0 = char/row
  1 = pixel x/y
  column value only used if font is monospaced or forced monospaced. Pixel
  x/y is top left pixel of the first character
data[3] = column or x-pixel position of the top-left of first character
data[4] = row or y-pixel position of the top-left of first character
data[5-123] = utf-8 or utf-16 text string
```

Successful return packet:

```
type = 0x40 | 0x26 = 0x46 = 102₁₀
data_length = 2
data[0] = 1 (Print Custom Font to Display)
data[1] = length of the printed text in pixels
```

39 (0x27): MicroSD File Operations Command Group

This command group provides commands to perform operations a microSD card that may be inserted into the microSD slot on the back of the CFA835 module.
The microSD card must be of SDHC type, and be formatted to FAT12/16/32.

Subcommand 0: Open/Close MicroSD File

This command opens the specified file on the inserted microSD card for reading/writing. Only one file on the microSD card may be accessed at a time. The subcommands 1 through 4 operate on the opened file.

Command packet:

```
type = 0x27 = 39₁₀
data_length = 2 to 124
data[0] = 0 (Open/Close File)
data[1] = options
  0 = close currently opened file (file name does not need to be
specified)
  1 = open file for reading
  2 = open file for reading and writing (truncates existing file)
  3 = open file for reading and writing (appends to existing file)
data[2-123] = file name of the file located on the microSD card

options 1 and 2 will set the file pointer position to the start of the
file (position 0).
option 2 will set the file pointer position to the end of the file.
```

Successful return packet:

```
type = 0x40 | 0x27 = 0x67 = 103₁₀
data_length = 5
data[0] = 0 (Open/Close File)
data[1-4] = file size in bytes
```

Subcommand 1: Position Seek

This command seeks (sets the file pointer) to the location specified in the file opened with the subcommand immediately above, Subcommand 0: Open/Close MicroSD File.

Command packet:

```
type = 0x27 = 39₁₀
data_length = 5
data[0] = 1 (Position Seek)
data[1-4] = 32 bit location of byte position in the file (LSB first)
```

Successful return packet:

```
type = 0x40 | 0x27 = 0x67 = 103₁₀
data_length = 1
data[0] = 1 (Position Seek)
```

Subcommand 2: Read File Data

Read data from the file opened by command 39, Subcommand 0: Open/Close MicroSD File. Data is read from the current file pointer location. The file pointer position will be incremented by the amount of data read by this command. To read data from elsewhere in the file, use command immediately above, Subcommand 1: Position Seek first.

Command packet:

```
type = 0x27 = 39₁₀
data_length = 2
data[0] = 2 (Read File Data)
data[1] = number of bytes to read (1 to 124)
```

Successful return packet:

```
type = 0x40 | 0x27 = 0x67 = 103₁₀
data_length = 1 to 124
data[0] = 2 (Read File Data)
data[1-123] = data read from the file

If the returned length of data read from the file is less than requested,
the the end-of-file has been reached.
```

Subcommand 3: Write File Data

Writes data to the file opened by command 39, Subcommand 0: Open/Close MicroSD File. Data is written at the current file pointer location.

Command packet:

```
type = 0x2F = 47₁₀
data_length = 2 to 124
data[0] = 3 (Write File Data)
data[1-123] = data to write to the file
```

Successful return packet:

```
type = 0x40 | 0x27 = 0x67 = 103₁₀
data_length = 1
data[0] = 3 (Write File Data)
```

Subcommand 4: Delete A File

This command deletes the specified file from the microSD card. Attempting to delete a currently open file will result in an error.

Command packet:

```
type = 0x27 = 39₁₀
data_length = 2 to 124
data[0] = 4 (Delete a File)
data[1-123] = file name of the file located on the microSD card
```

Successful return packet:

```
type = 0x40 | 0x27 = 0x67 = 103₁₀
data_length = 1
data[0] = 4 (Delete a File)
```

40 (0x28): Display Graphic Options Command Group

The CFA835's front LCD is a 244 x 68 pixel monochrome / greyscale display. The sub-commands in this group manipulate this display.

The CFA835 supports the ability to update the display either directly or using a buffer that can be flushed manually. This option is enabled or disabled using subcommand 0

Valid ranges for all the subcommands in this command group are:

```
x pixels / width = 0 - 243
y pixels / height = 0 - 67
shade = 0 - 255
```

Subcommand 0: Graphic Options

This command controls two of the options related to the CFA835's graphical display capabilities:

- Buffer Flush

When enabled, display graphical commands (except command 31 (0x1F): Write Text to the Display) are buffered and only written to display when using sub-command 1.

- Gamma Correction

When enabled, graphics and fonts written to the display will have gamma correction applied. This option does not affect command 31 (0x1F): Write Text to the Display.

Command packet:

```
type = 0x28 = 40₁₀
data_length = 2
data[0] = 0 (Graphics Options)
data[1] = option flags
bit 0 = buffer flush (0 = automatic, 1 = manual)
bit 1 = gamma correction (1 = enabled, 0 = disabled)
```

Successful return packet:

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 0 (Graphics Options)
```

Subcommand 1: Buffer Flush

This command flushes the memory of the graphical buffer to the CFA835's display. This command has no effect unless sub command 0 buffer flush option is set to manual.

Command packet:

```
type = 0x28 = 40₁₀
data_length = 1
data[0] = 1 (Buffer Flush)
```

Successful return packet:

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 1 (Buffer Flush)
```

Subcommand 2: Send Image Data to Display from Host

This command supports a special “data streaming” mode unique to this command. After this packet has been sent to the CFA835, raw pixel data (not in normal packet format) is sent to the CFA835.

NOTE:

- As graphical data is not sent in packets, it is not CRC checked. Any data transmission errors will result in an incorrect image being displayed on the CFA835.
- A return acknowledge packet will not be sent by the CFA835 to the host until transmission of the graphical data is complete.
- If “manual buffer flush” is enabled (see command 40, Subcommand 0: Graphic Options), the image will not be drawn until the subcommand immediately above, Subcommand 1: Buffer Flush is executed.
- This command has no support for directly interpreting jpg/png/bmp/etc. file formats – only raw pixel data. cfTest includes functionality to convert an image (many different formats) into raw data which is then sent to the CFA835.

The raw pixel data transfer must be completed within 500 ms from the USB interface or 2 seconds from any other interface. Failure to do so will result in the CFA835 returning an error packet and ignoring any following raw data.

Raw pixel data is in the format of one byte per pixel. The display is capable of displaying 32 shades of grey (most significant 5 bits of the byte). The least significant 3 bits of shade is ignored. Pixel data is interpreted in order: left to right, top to bottom.

Optional RLE compression removes repetitive values. Here is an example:

RLE Compression Example (values in hexidecimal)											
Byte Order	0	1	2	3	4	5	6	7	8	9	10
Original Pixel Data	0x00	0xF8	0x30	0x30	0x30	0x30	0x30	0x30	0x30	0xF8	0x00
Sent RLE Data	0x00	0xF8	0x03	0x07	0x30	0xF8	0x00				
Displayed Pixel Data	0x00	0xF8	0x30	0x30	0x30	0x30	0x30	0x30	0x30	0xF8	0x00

Command packet:

```
type = 0x28 = 40₁₀
data_length = 6
data[0] = 2 (Send Image Data To Display From Host)
data[1] = option flags
  bit 0 = enable transparency (pixel value 0 is transparent)
  bit 1 = invert image color (will invert transparency value also)
  bit 2 = enable RLE compression (format: 0x03, length, value)
data[2] = x pixel location to start
data[3] = y pixel location to start
data[4] = width of image in pixels
data[5] = height of image in pixels
```

Successful return packet:

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 2 (Send Image Data To Display From Host)
```

Subcommand 3: Display Image File from MicroSD Card on CFA835

This command displays a BMP formatted image file located on the inserted microSD card. The BMP file must be grayscale, 8 bits/pixel, no compression, Microsoft Windows format only.

NOTE: If “manual buffer flush” is enabled (see command 40, Subcommand 0: Graphic Options), the image will not be drawn until command 40, Subcommand 1: Buffer Flush is executed.

Command packet:

```
type = 0x28 = 40₁₀
data_length = 6 to 124
data[0] = 3 (Display Image File From MicroSD Card On CFA835)
data[1] = option flags
  bit 0 = enable transparency (pixel value 0 is transparent)
  bit 1 = invert image shade (will invert transparency value also)
data[2] = x pixel location to start
data[3] = y pixel location to start
data[4-123] = name of the image file located on the microSD card
```

Successful return packet:

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 3 (Display Image File From MicroSD card on CFA835)
```

Subcommand 4: Save Screenshot to MicroSD File

This command saves a screenshot of the current image to a BMP file of the specified name on the microSD card. If a file with the specified name already exists, it will be overwritten. The BMP file will be saved in Microsoft format, 8bits/ pixel, greyscale, with no compression, and is 17KBytes in size.

NOTE: If “manual buffer flush” is enabled (see command 40, Subcommand 0: Graphic Options), the image stored will be the image currently in the buffer.

Command packet:

```
type = 0x28 = 40₁₀
data_length = 2 to 124
data[0] = 4 (Save Screenshot to MicroSD File)
data[1-123] = name of the file to create on the microSD card
```

Successful return packet:

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 4 (Save Screenshot to MicroSD File)
```

Subcommand 5: Pixel Data

This command sets or reads the value of the specified individual pixel on the display.

NOTE: If "manual buffer flush" is enabled by command 40, Subcommand 0: Graphic Options, the value returned is the pixel value in the buffer.

Command packet (Write):

```
type = 0x28 = 40₁₀
data_length = 4
data[0] = 5 (Pixel Data)
data[1] = x pixel location (0-243)
data[2] = y pixel location (0-67)
data[3] = new pixel shade
```

Successful return packet (Write):

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 5 (Pixel Data)
```

Command packet (Read):

```
type = 0x28 = 40₁₀
data_length = 3
data[0] = 5 (Pixel Data)
data[1] = x pixel location (0-243)
data[2] = y pixel location (0-67)
```

Successful return packet (Read):

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 2
data[0] = 5 (Pixel Data)
data[1] = pixel shade value
```

Subcommand 6: Draw a Line

This command draws a line of the specified shade from point A to point B.

NOTE: If "manual buffer flush" is enabled (see command 40, Subcommand 0: Graphic Options), the line will not be displayed onto the CFA835 until command 40, Subcommand 1: Buffer Flush is executed.

Command packet:

```
type = 0x28 = 40₁₀
data_length = 6
data[0] = 6 (Draw a Line)
data[1] = x pixel location to start
data[2] = y pixel location to start
data[3] = x pixel location to finish
data[4] = y pixel location to finish
data[5] = line shade value
```

Successful return packet:

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 6 (Draw a Line)
```

Subcommand 7: Draw a Rectangle

This command draws a rectangle to the CFA835's display.

NOTE: If “manual buffer flush” is enabled (see command 40, Subcommand 0: Graphic Options), the rectangle will not be displayed onto the CFA835 until command 40, Subcommand 1: Buffer Flush is executed.

Command packet:

```
type = 0x28 = 40₁₀
data_length = 7
data[0] = 7 (Draw a Rectangle)
data[1] = x pixel location (top-left)
data[2] = y pixel location (top-left)
data[3] = rectangle width
data[4] = rectangle height
data[5] = line shade
data[6] = fill shade (0 is transparent)
```

Successful return packet:

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 7 (Draw a Rectangle)
```

Subcommand 8: Draw a Circle

This command draws a circle of the specified radius using the specified x,y pair as its center point.

NOTE: If “manual buffer flush” is enabled (see command 40, Subcommand 0: Graphic Options), the circle will not be displayed onto the CFA835 until command 40, Subcommand 1: Buffer Flush is executed.

Command packet:

```
type = 0x28 = 40₁₀
data_length = 6
data[0] = 8 (Draw a Circle)
data[1] = x of circle
data[2] = y position center of circle
data[3] = circle radius
data[4] = line shade
data[5] = fill shade (0 is transparent)
```

Successful return packet:

```
type = 0x40 | 0x28 = 0x68 = 104₁₀
data_length = 1
data[0] = 8 (Draw a Circle)
```

41 (0x29): Video Playback Control Command Group

The CFA835 can play up to four independent video files (four “slots”) to the CFA835 at a time.

Video slots are drawn in order of slot number, so a video in slot 1 will be displayed over the top of a video in slot 0. Each video can be controlled independently using the sub-command immediately below, Subcommand 1: Video Control.

The video files must be encoded using the CFA835 Video Encoder utility.

NOTE: Playing a video directly on top of another video may result in flicker. We recommend against doing this. If your project solution depends on playing multiple videos layered over each other, compression must be disabled during encoding and the videos must have the same frame rate.

Subcommand 0: Load A Video from MicroSD Card

Command packet:

```
type = 0x29 = 41(10)
data_length = 3 to 124
data[0] = 0 (Load A Video From MicroSD Card)
data[1] = video slot number (0 to 3)
data[2-123] = name of the video file on the microSD card
```

Successful return packet:

```
type = 0x40 | 0x29 = 0x69 = 105(10)
data_length = 1
data[0] = 0 (Load A Video From MicroSD Card)
```

Subcommand 1: Video Control

This command controls the video(s) opened using the subcommand immediately above, Subcommand 0: Load A Video from MicroSD Card.

NOTE: Attempting to play a video outside of the display's graphical limits will result in an error being returned.

Command packet:

```
type = 0x29 = 41(10)
data_length = 3 or 6
data[0] = 1 (Video Control)
data[1] = video slot number (0 to 3)
data[2] = control option
  0 = play
  1 = stop (data[3-5] not required for this option)
  2 = toggle pause (data[3-5] not required for this option)
data[3] = play video X times in loop (up to 255) (0x00 = continuously)
data[4] = x pixel location
data[5] = y pixel location
```

Successful return packet:

```
type = 0x40 | 0x29 = 0x69 = 105(10)
data_length = 1
data[0] = 1 (Video Control)
```

Report Code 128 (0x80): Key Activity

The CFA835 can be configured to report information automatically when data becomes available. Reports are not sent in response to a particular packet received from the host, see details below.

If a key is pressed or released, the CFA835 sends a Key Activity report packet to the host. Key event reporting may be individually enabled or disabled by command 23 (0x17): Keypad Reporting.

Report packet:

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

10. Character Generator ROM (CGROM)

To find the code for a given character, add the two numbers that are shown in bold for its row and column. For instance, to display a superscript 9, add together the decimal column and row headers – 128_{10} and 9_{10} to get 137_{10} or combine the upper and lower 4 bits (1000 and 1001 become 1000 1001).

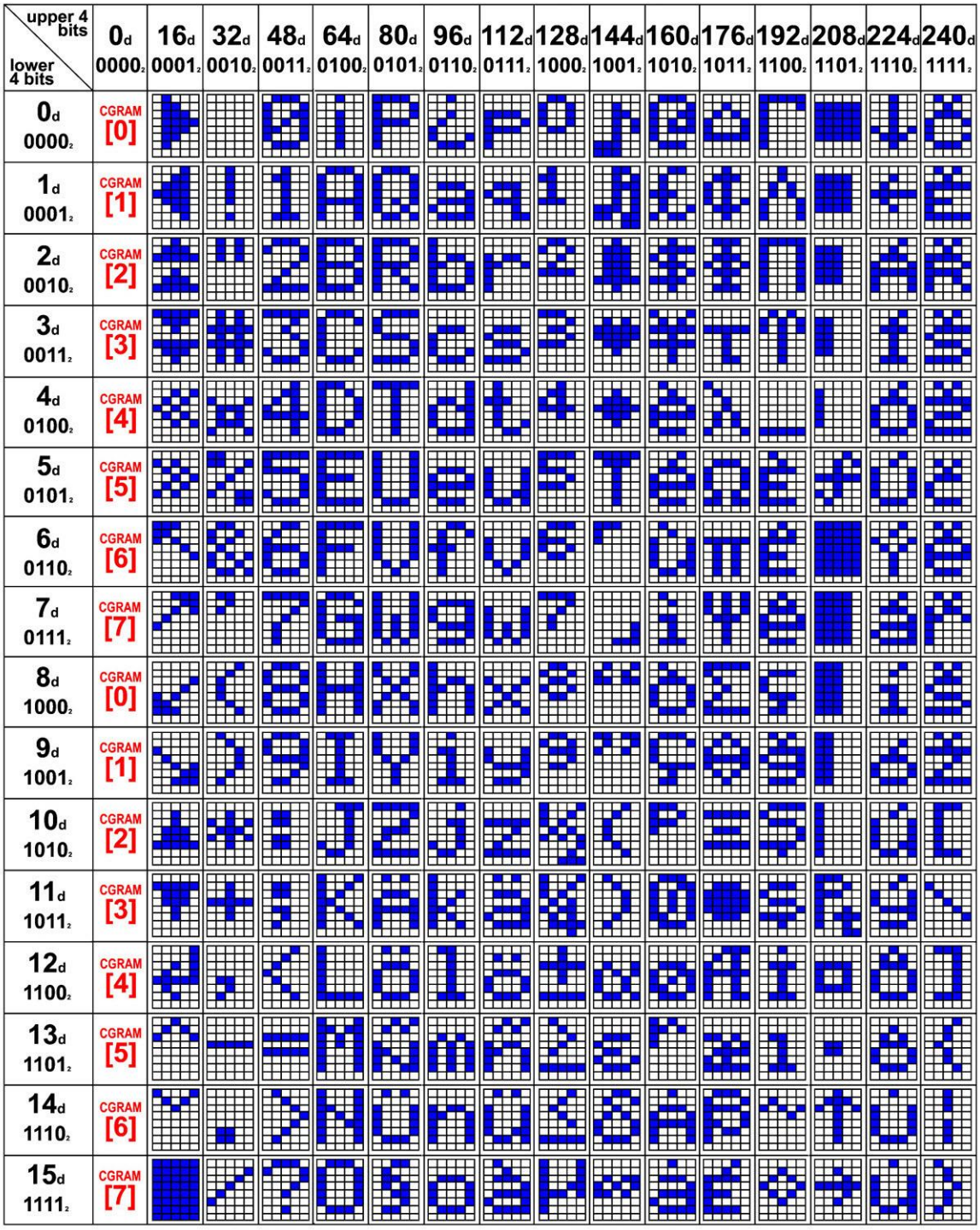


Figure 12. Character Generator (CGROM)

11. LCD Module Reliability and Longevity

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

ITEM	SPECIFICATION	
LCD portion (excluding Keypad and Backlights)	50,000 to 100,000 hours (typical)	
Keypad	1,000,000 keystrokes	
Bicolor LED status lights	50,000 to 100,000 hours	
White and Blue 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	>70%
	<50,000	>50%

11.1. Module Longevity (EOL / Replacement Policy)

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

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

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

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

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

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

12. Care and Handling Precautions

For optimum operation of the CFA835 and to prolong its life, please follow the precautions described below.

12.1. ESD (Electrostatic Discharge)

The USB, CFA-RS232, Tx and Rx lines have industry standard protection. 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.

12.2. Design and Mounting

- The exposed surface of the “glass” is actually a polarizer laminated on top of the glass. To protect the soft plastic polarizer from damage, the module ships with a protective film over the polarizer. Please peel off the protective film slowly. Peeling off the protective film abruptly may generate static electricity.
- When handling the module, avoid touching the polarizer. Finger oils are difficult to remove.
- To protect the soft plastic polarizer from damage, place a transparent plate (for example, acrylic, polycarbonate or glass), in front of the module, leaving a small gap between the plate and the display surface.
- Do not disassemble or modify the module.
- Do not modify the six tabs of the metal bezel or make connections to them.
- Do not reverse polarity to the power supply connections. Reversing polarity will immediately ruin the module.

12.3. Avoid Shock, Impact, Torque, or Tension

- Do not expose the CFA835 to strong mechanical shock, impact, torque, or tension.
- Do not drop, toss, bend, or twist the CFA835.
- Do not place weight or pressure on the CFA835.

12.4. If LCD Panel Breaks

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

12.5. Cleaning

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

12.6. Operation

- Protect the CFA835 from ESD and power supply transients.
- Observe the operating temperature limitations: a minimum of -20°C to a maximum of +70°C with minimal fluctuation. Operation outside of these limits may shorten life and/or harm display.
- At lower temperatures of this range, response time is delayed.
- At higher temperatures of this range, display becomes dark. (You may need to adjust the contrast.)
- Operate away from dust, moisture, and direct sunlight.

- Adjust backlight brightness so the display is readable, but not too bright.
- Dim or turn off the backlight during periods of inactivity to conserve the backlight lifetime.

12.7. Storage and Recycling

- Store in an ESD-approved container away from dust, moisture, and direct sunlight.
- Observe the storage temperature limitations: -30°C minimum, +80°C maximum with minimal fluctuation. Rapid temperature changes can cause moisture to form, resulting in permanent damage.
- Do not allow weight to be placed on the CFA835 while in storage.
- Please recycle your outdated Crystalfontz modules at an approved facility.

13. Mechanical Drawings

Figure 13. CFA835 Module Outline Drawing (1 of 2)

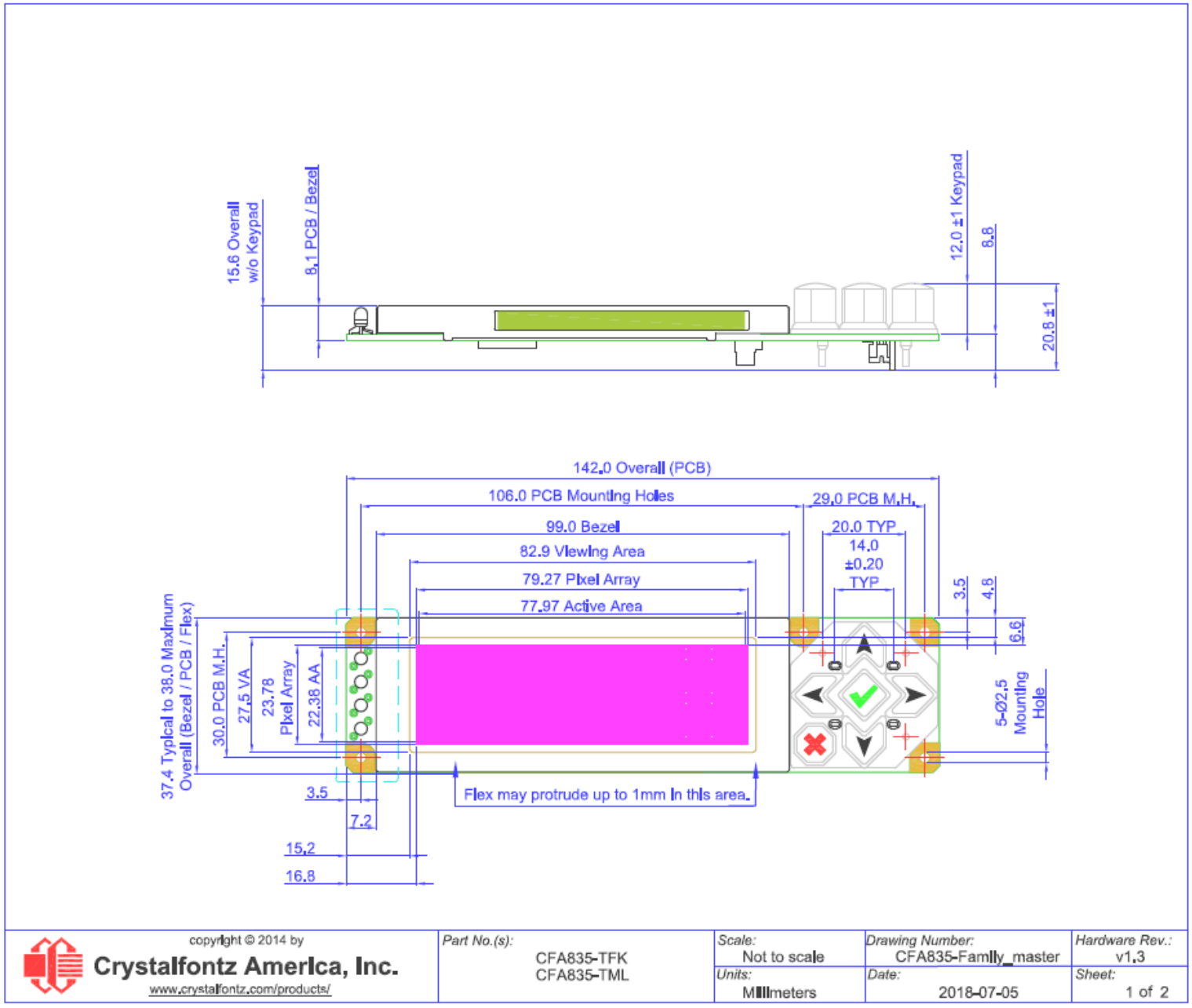
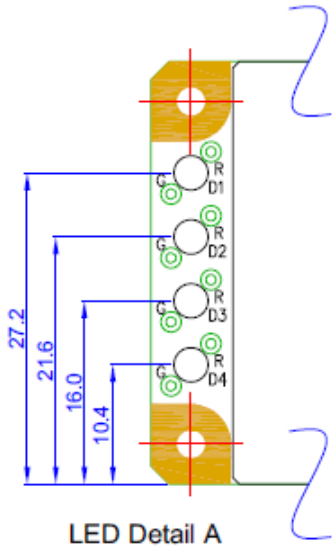
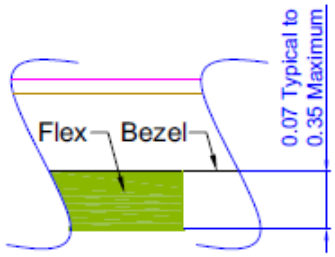


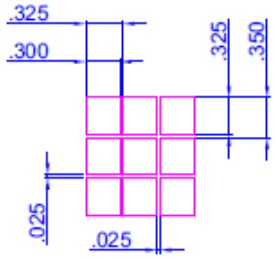
Figure 14. CFA835 Module Outline Drawing (2 of 2)



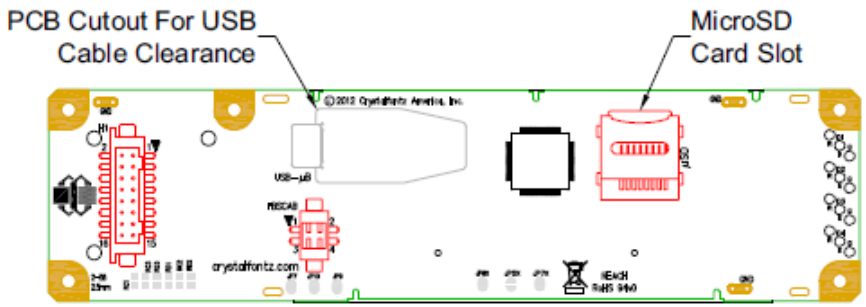
LED Detail A
(Front View)



Flex Detail B



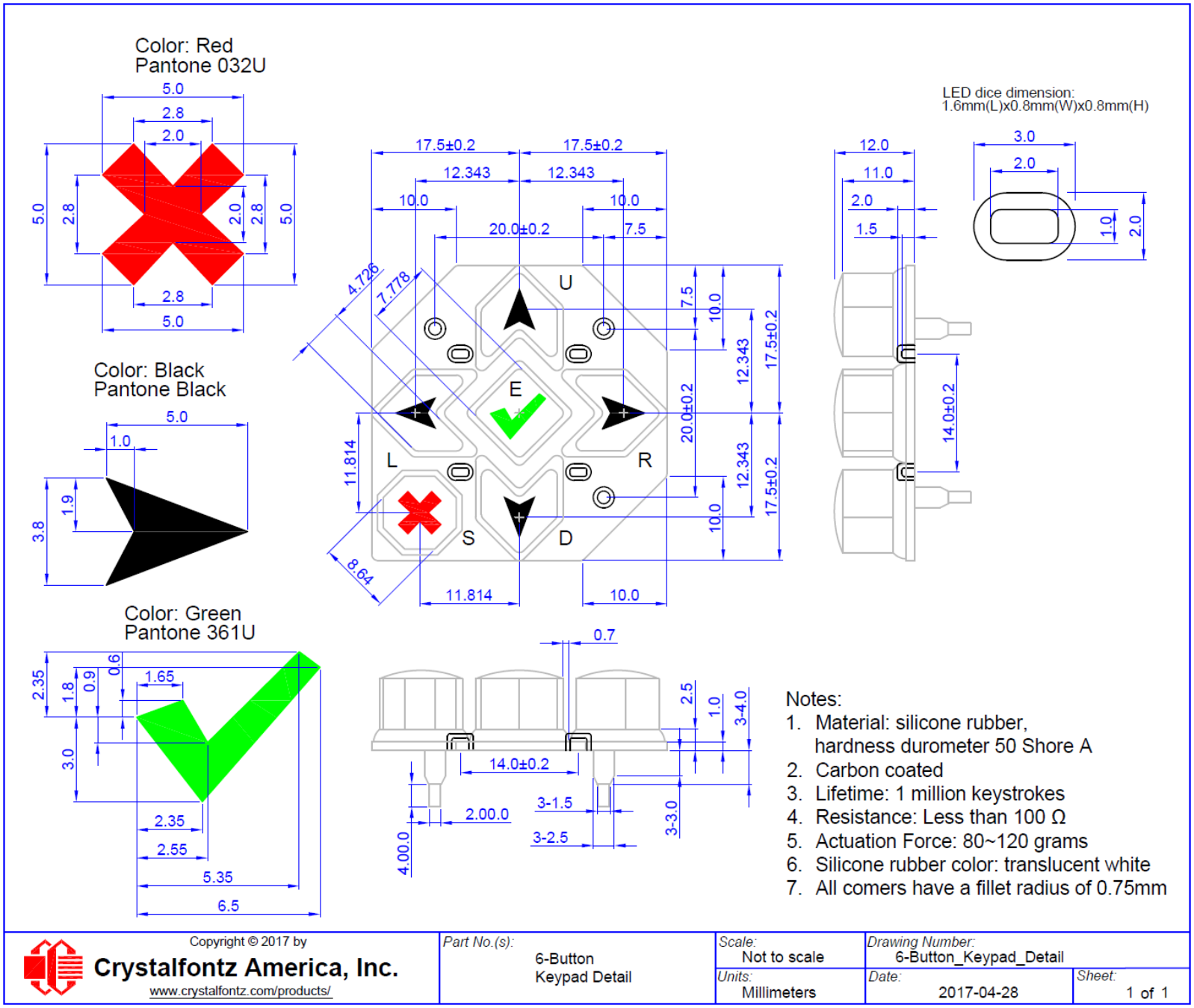
Pixel Detail C



Back View

copyright © 2017 by **Crystalfontz America, Inc.** www.crystalfontz.com/products/	*Part No.(s):* CFA835-TFK, CFA835-TML, CFA835-YYK	*Scale:* Not to scale	*Drawing Number:* CFA835-Family_master	v1.1
		Units: Millimeters	*Date:* 2018-01-19	*Sheet:* 2 of 2

Figure 15. Keypad Detail Drawing



14. Appendix A: Demonstration Software and Sample Code

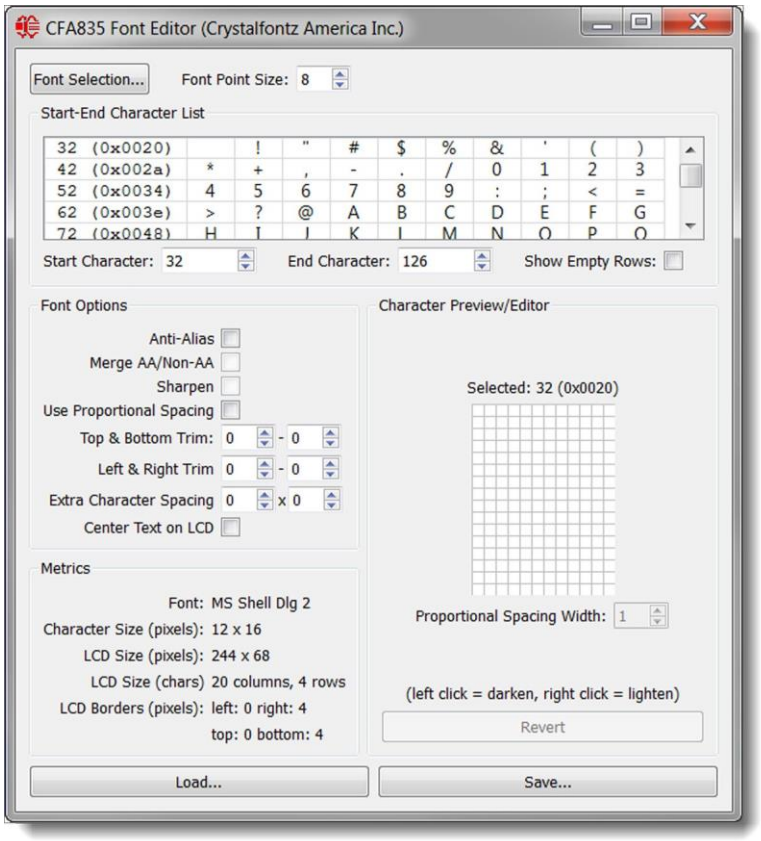
The CFA835 Window utilities described below are bundled together in a CFA835 Utilities Package.

14.1. Crystalfontz cfTest

cfTest for Windows is testing and configuration software that works on all Crystalfontz Intelligent LCD modules. This software allows you to experiment with the command set for all Crystalfontz Smart LCDs.

Streaming communication-based modules (CFA632, CFA634) and packet communication-based modules (CFA533, CFA631, CFA633, CFA635, CFA735, CFA835) are supported.

14.2. CFA835 Font Editor

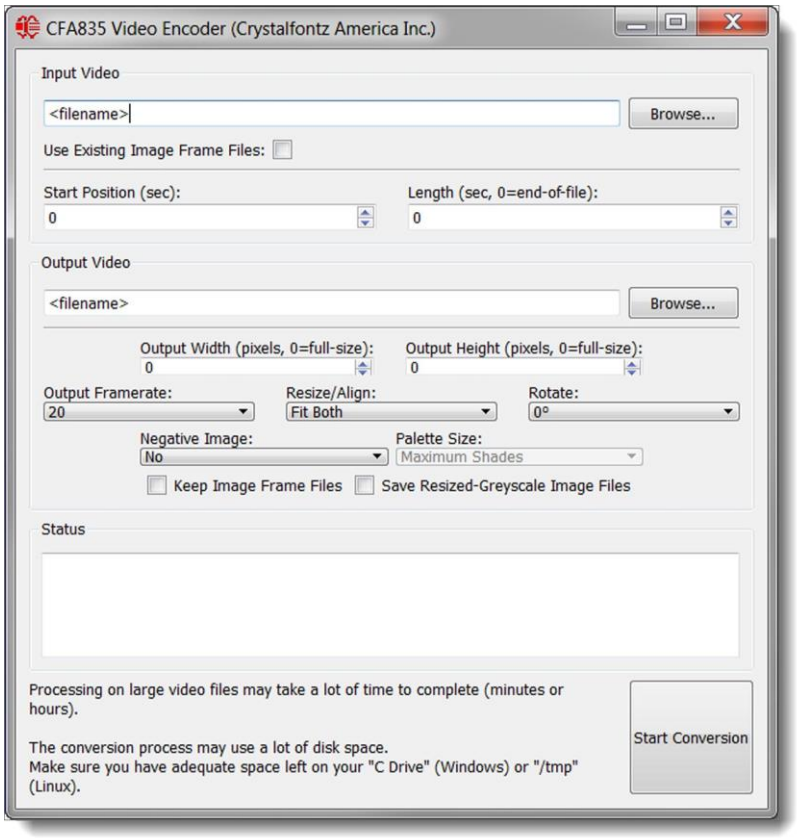


The CFA835 Font Editor converts any font into the CFA835 font format. The editor creates CFA835 compatible custom font files using fonts available on your PC. When the font file is loaded onto a microSD card inserted into the CFA835 card socket, the module can write custom font text to the display.

The font converter and CFA835 support UTF16 (Unicode) fonts, allowing non-English (for example, Cyrillic, Asian, symbolic, etc.) font files to be created and displayed. Many font size, type, spacing, and other options are available.

See CFA835 commands Subcommand 0: Load Custom Font Files from MicroSD Card and Subcommand 1: Print Custom Font to Display for details on font file use.

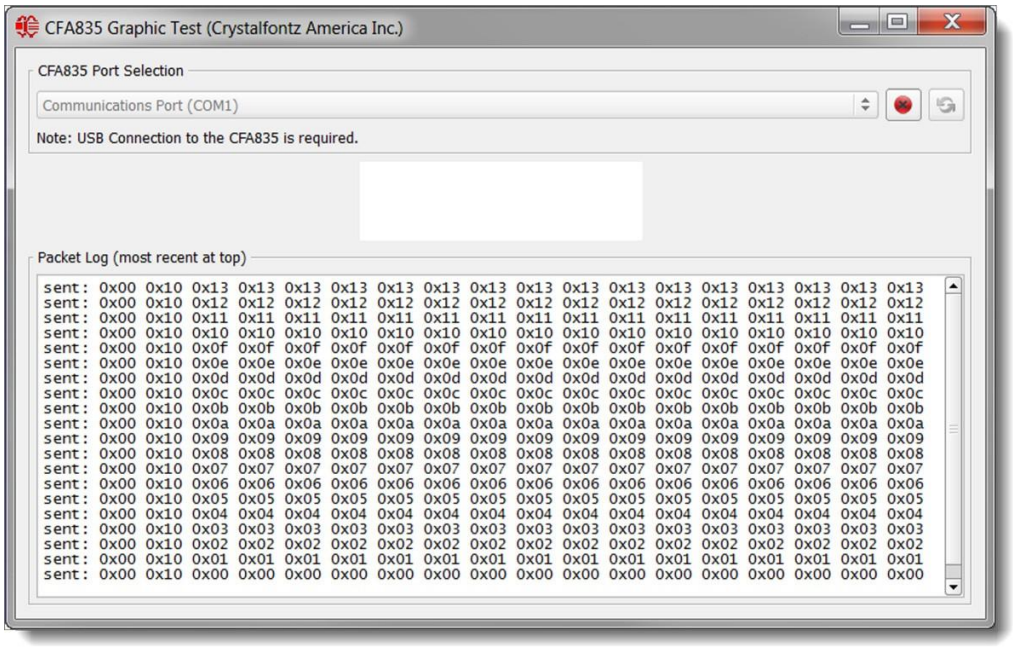
14.3. CFA835 Video Encoder



The Video Encoder converts common video format files into a video file that the CFA835 can play to the display. The video conversion uses MPlayer (a GNU-GPLv2 licensed open-source software) to create many single image files from the source video, and then reassembles the image files into a CFA835 video file. Processing time depends on the source video file.

See CFA835 commands 41 (0x3A): Video Playback Control for details on playing a video on the CFA835.

14.4. CFA835 Graphic Test



This demonstration shows some of the graphical capabilities of the CFA835 by rendering an animated logo, clock, histogram, and scrolling text. Source code (C++, Qt 4.8 and created in QtCreator 2.5) is included in the utilities package.

14.5. Linux CLI Examples

CLI Example Software is a Linux compatible command-line demonstration program with C source code. 8K.

NOTE: It will show as /dev/ttyACMx instead of /dev/ttyUSBx.

LCDproc is an open source project that supports many of the Crystalfontz displays. The CFA635 configuration should work with the CFA835.

14.6. Sample Code for RPM Calculation Information

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

```
bool HandleFanRPMReplyPacket (COMMAND_PACKET *packet, char *output)
{
   uint8_t fbscab_index;
   uint8_t fan_index;
   uint8_t cycles;
   uint8_t data_offset;
   uint8_t timer_lsb;
   uint8_t timer_msb;
   uint8_t pulses_per_revolution;
   uint16_t  timer_ticks;
   uint8_t output_offset;
   float     fan_rpm;

/*
   fan rpm query command response packet has the format of:
   type = 0x40 | 0x25 = 0x65 = 101
   data_length = 14
   data[0] = 2 (read fan tachometer speed)
   data[1] = FBSCAB module index
   data[2] = fan 1 number of fan tach cycles
   data[3] = fan 1 LSB of fan timer ticks
   data[4] = fan 1 MSB of fan timer ticks
   data[5] = fan 2 number of fan tach cycles
   data[6] = fan 2 LSB of fan timer ticks
   data[7] = fan 2 MSB of fan timer ticks
   data[8] = fan 3 number of fan tach cycles
   data[9] = fan 3 LSB of fan timer ticks
   data[10] = fan 3 MSB of fan timer ticks
   data[11] = fan 4 number of fan tach cycles
   data[12] = fan 4 LSB of fan timer ticks
   data[13] = fan 4 MSB of fan timer ticks
   */

   //check packet length
   if (packet->length != 14)
   {
          //unexpected packet length, should be 14 bytes
          return false;
   }
   //check the packets command number and type
      // 0x25 | 0x40 = FBSCAB Command Group | Reply Packet
      if (packet->command != (0x25 | 0x40))
   {
       //wrong packet command/type
       return false;
   }
   //check the packets sub-command type
// 2 = Read fan tachometer speed
if (packet->data[0] != 2)
   {
      //wrong packet sub-command value
      return false;
   }
   //get fbscab index from the packet
   fbscab_index = packet->data[1];

   //prepare output string
   output_offset = 0;
```

```
   output_offset += sprintf(&output[output_offset], "FBSCAB:%d - ",
fbscab_index);

   //process packet data for the 4 fans
   for (fan_index = 0; fan_index < 4; fan_index++)
   {
      //data offset for fan_index data in the packet
      data_offset = 2 + (fan_index * 3);
      //prepare output string
      output_offset += sprintf(&output[output_offset], "FAN%d: ",
fan_index);
      //get the fan data from the packet
      cycles = packet->data[data_offset];
      timer_lsb = packet->data[data_offset+1];
      timer_msb = packet->data[data_offset+2];
      timer_ticks = timer_lsb | (timer_msb << 8);
      //check fan cycles value
      if (cycles < 3)
      {
         //fan has stopped
         output_offset += sprintf(&output[output_offset], "STOPPED ");
         //next fan
         continue;
      }
      if (cycles < 4)
      {
         //fan is turning too slow to count RPM
         output_offset += sprintf(&output[output_offset], "SLOW ");
         //next fan
         continue;
      }
      if (cycles == 0xFF)
      {
         //unknown value
         output_offset += sprintf(&output[output_offset], "UNKNOWN ");
         //next fan
         continue;
      }

      //if we get to here, we have valid fan tach data
      //calculate fan RPM
      pulses_per_revolution = 2; //specific to each fan, most commonly 2
      fan_rpm = ((27692308L / pulses_per_revolution) * (cycles - 3)) /
(float)tim- er_ticks;

      //add RPM to output string
      output_offset += sprintf(&output[output_offset], "%5.2f ", fan_rpm);
      //done, next fan
      }
   //all done
   return true;
}
```

14.7. Sample Code for Temperature Sensor Report

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

```
bool HandleTempReplyPacket (COMMAND_PACKET *packet, char *output)
{
   uint8_t fbscab_index;
   uint8_t sensor_index;
   uint8_t temp lsb;
   uint8_t temp msb;
   uint16_t  temp raw;
   uint8_t crc status;
   float     deg c;
   float     deg f;

   /*
   temperature query command response packet has the format of:
   type = 0x40 | 0x25 = 0x65 = 101
   data_length = 5
   data[0] = 4 (read WR-DOW-Y17 temperature)
   data[1] = FBSCAB module index
   data[2] = DOW device index (0-15)
   data[3] = LSB of temperature data
   data[4] = MSB of temperature data
   */

   //check the packets command number and type
   // 0x25 | 0x40 = FBSCAB Command Group | Reply Packet
   if (packet->command != (0x25 | 0x40))
   {
      //wrong packet command/type
      return false;
   }
   //check the packets sub-command type
   // 4 = Read WR-DOW-Y17 temperature
   if (packet->data[0] != 4)
   {
   //wrong packet type
   return false;
   }

   //get fbscab & temp sensor index from the packet
   fbscab_index = packet->data[1];
   sensor index = packet->data[2];
   //get raw temperature data from the packet
   temp lsb = packet->data[3];
   temp msb = packet->data[4];
   temp raw = temp lsb | (temp msb << 8);

   //check temperature data CRC flags
   crc status = temp raw << 14;
   if (crc status = 1)
   {
      //CRC check failed
      return false;
   }
   if (crc_status == 2)
   {
   //no sensor in this location
   //this should never happen
   return false;
   }
   if (crc_status == 3)
```

```
   {
      //no valid data from this sensor yet
      return false;
   }
   //if we get to here, crc status==0, so temperature data is valid
   //calculate temperature
   deg_c = temp_raw / (float)16.0;
   deg_f = (deg_c * 9.0) / 5.0 + 32.0;
   //return text
   sprintf(output, "FBSCAB:%d SENSOR:%d TEMP_DEGC:%0.2f
TEMP_DEGF:%0.2f", fbscab_index, sensor_index, deg_c, deg_f);
   //done
   return true;
}
```

14.8. Sample Code for Font File Format

The following source code is C pseudo-code. It will need to be modified to fit your application. The structures are little- endian and are byte-aligned packed.

```
//font flags
#define FR_None         0x00
#define FR_AntiAliased 0x01
#define FR_Proportional       0x02
#define FR_MergeAA      0x04
#define FR_Sharpen      0x08
#define FR_CenterScreen       0x10

//char flags
#define FR_NoChar       0x00
#define FR_HasCharacter       0x01
#define FR_IsCustomChar       0x02

//version information
#define FR_FileID       "CFFF"
#define FR_FileVersion 105

typedef struct
{
char         ID[4];     //FR_FileID
uint16_t     Version;  //FR_FileVersion

//rendering data
uint8_t   DataWidth;   //character width in pixels
uint8_t   DataHeight;  //character height in pixels
uint16_t  StartChar;   //UTF16 character number of first character in font
file
uint16_t  EndChar;     //UTF16 character number of last character in font
file
uint8_t   CharSpaceRight;     //extra character spacing on the right
uint8_t   CharSpaceBelow;     //extra character spacing below
uint8_t   ScreenSpaceLeft; //offset character positions to the right by X
pixels
uint8_t   ScreenSpaceTop;     //offset character positions downwards by X
pixels
uint8_t   Flags;       //font flags

//font editor use only
//these values can be undefined, CFA835 module disregards these values
char      OrigFont[128];
uint8_t   TrimTop;
uint8_t   TrimBottom;
```

```
uint8_t   TrimLeft;
uint8_t   TrimRight;
} FR_FileHeader;

typedef struct
{
uint8_t   CharFlags; //character flags
uint8_t   CharWidth; //character width in pixels (for proportional fonts)
uint8_t   CharData[FR_FileHeader.DataWidth * FR_FileHeader.DataHeight];
} FR_Character;

typedef struct
{
FR_FileHeader     Header;
FR_Character      Characters[FR_FileHeader.EndChar -
FR_FileHeader.StartChar];
} FR_FontFile;
```

14.9. Sample Code

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

- Windows compatible test/demonstration program: https://www.crystalfontz.com/product/cftest
- Windows compatible example program and source: https://www.crystalfontz.com/product/635wintest
- Linux compatible command-line demonstration program with C source code. 8K. https://www.crystalfontz.com/product/linuxexamplecode
- Supported by CrystalControl freeware: https://www.crystalfontz.com/product/CrystalControl2.html

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

14.10. Algorithms to Calculate the CRC

Below are eight sample algorithms that will calculate the CRC of a CFA835 packet. Some of the algorithms were contributed by forum members and originally written for CFA631 and CFA835. The CRC used in the CFA835 is the same one that is used in IrDA, which came from PPP, which seems to be related to a CCITT (ref: Network Working Group Request for Comments: 1171) standard. At that point, the trail was getting a bit cold and diverged into several referenced articles and papers, dating back to 1983.

The polynomial used is $X^{16} + X^{12} + X^5 + X^0$ (0x8408)

The result is bit-wise inverted before being returned.

Algorithm 1: "C" Table Implementation

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

```
//This code is from the IRDA LAP documentation, which appears to
//have been copied from PPP.
//
//I doubt that there are any worries about the legality of this code,
//searching for the first line of the table below, it appears that
//the code is already included in the linux 2.6 kernel "Driver for
//ST5481 USB ISDN modem". This is an "industry standard" algorithm
//and I do not think there are ANY issues with it at all.

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

```
0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78};

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

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

Algorithm 2: "C" Bit Shift Implementation

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

```
typedef unsigned char ubyte;
typedef unsigned short word;
word get_crc(ubyte *bufptr,word len)
{
register unsigned int newCRC;

//Put the current byte in here.
ubyte data;
int bit_count;

//This seed makes the output of this shift based algorithm match
//the table based algorithm. The center 16 bits of the 32-bit
//"newCRC" are used for the CRC. The MSb of the lower byte is used
//to see what bit was shifted out of the center 16 bit CRC
//accumulator ("carry flag analog");
newCRC=0x00F32100;
while(len--)
{
//Get the next byte in the stream
data=*bufptr++;

//Push this byte's bits through a software
//implementation of a hardware shift & xor.
for(bit_count=0;bit_count<=7;bit_count++)
{
//Shift the CRC accumulator
newCRC>>=1;
//The new MSB of the CRC accumulator comes
//from the LSB of the current data byte.
if(data&0x01)
newCRC|=0x00800000;
//If the low bit of the current CRC accumulator was set
//before the shift, then we need to XOR the accumulator
//with the polynomial (center 16 bits of 0x00840800)
if(newCRC&0x00000080)
newCRC^=0x00840800;
//Shift the data byte to put the next bit of the stream into position 0.
data>>=1;
}
}

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

```
newCRC>>=1;

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

Algorithm 2B: "C" Improved Bit Shift Implementation

This is a simplified algorithm that implements the CRC.

```
unsigned short get_crc(unsigned char count,unsigned char *ptr)
{
unsigned short crc; //Calculated CRC
unsigned char i;    //Loop count bits in byte
unsigned char data; //Current byte being shifted
crc = 0xFFFF; // Preset to all 1's, prevent loss of leading zeros
while(count--)
{
   data = *ptr++; i = 8;
   do
   {
      if((crc ^ data) & 0x01)
      {
         crc >>= 1; crc ^= 0x8408;
      }
      else
         crc >>= 1;
      data >>= 1;
   } while(--i != 0);
   }

return (~crc);
   }
```

Algorithm 3: "PIC Assembly" Bit Shift Implementation

This routine was graciously donated by one of our customers.

```
;=====================================================================
; Crystalfontz CFA835 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 CFA835 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 735_WinTest:
'http://www.crystalfontz.com/products/735/735_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 CFA835 module.

```
; CRC Algorithm for CrystalFontz CFA835 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
                ; CFA835
   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
;=====================================================================
```

15. 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.
