Crystalfontz

# INTELLIGENT LCD MODULE SPECIFICATIONS


**Hardware Version: v1.7**
**Firmware Version: v1.5**

**Datasheet Release: 2020-12-01**


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

# Table of Contents

# 1. General Information

| Datasheet Revision History |
|---|
| Hardware Version: v1.7<br>Firmware Version: v1.5<br>Datasheet Release: 2020-12-01<br><br>For information about firmware and hardware revisions, see the Part Change Notifications (PCN) under "News" in our website's navigation bar<br><br>Previous datasheet Version: 2020/02/03<br><br>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 non-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.<br><br>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.<br><br>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.<br><br>Company and product names mentioned in this publication are trademarks or registered trademarks of their respective owners.<br><br>Copyright © 2020 by Crystalfontz America, Inc.,12412 East Saltese Avenue, Spokane Valley, WA 99216 U.S.A. |

# 2. Introduction

The CFA835 family of modules are intelligent graphic LCDs. These modules pack a lot in one package: a display with a stainless-steel bezel, six-button keypad, GPIOs, LEDs, and memory. These modules include an STM32F401 microcontroller and use a buck-boost switching supply to allow for a wide supply voltage range which makes these modules perfect for use in embedded systems. Use this display module to monitor temperatures and control fans using the optional FBSCAB, replace a system's power-on and power-off buttons, display system status information, or in a multitude of other ways.

## 2.1. Main Features

- Edge lit 244x68 16-shade greyscale LCD module with wide viewing angles.
- Backlit 6-key keypad, with presses reported to the host device.
- Four bi-color, host controllable, dimmable status LEDs
- Single power supply, with wide power supply voltage range (+3.3v to +5.5v).
- Five host interface options - USB 2.0, Logic-level Serial, RS232 Serial, I2C and SPI.
- Robust, packet-based communication protocol with 16-bit CRC for error-free communications.
- Slim form-factor fits nicely in a 1U rack mount case (37 mm overall height).
- Wide operating temperature range of -20°C to +70°C.
- Configurable 13 pin I/O interface for custom host-controlled monitoring/control applications.
- Support for one or more CFA-FBSCAB modules, providing temperature monitoring and fan power control abilities (including automatic fan control).
- Physically similar to CFA635 and CFA735 modules (same mounting, LCD panel, keypad, LCD locations).
- Optional half-height 5 ¼ inch PC drive bay mounting bracket is available.
- Nonvolatile memory capability (EEPROM):
  - Customize the "power-on" display settings (backlight brightness, boot screen, LED settings).
  - 124-byte "scratch" register for storing custom data, such as: IP address, netmask, system serial number, etc.
- Hardware watchdog can reset host on host software/hardware failure.
- Command set supports features such as:
  - Custom-made unicode compatible fonts (read from microSD card)
  - Images and video displayed from microSD card, or host device
  - Rendering of simple graphics objects
  - Manual and automatic fan control (when using a CFA-FBSCAB module)
  - Host PC ATX power control (on/off/reset) functions
  - microSD card file access.
- Field upgradeable firmware using a host PC, or microSD card.
- Crystalfontz cfTest PC (Window/Linux/Mac) utility can be used to setup & test the CFA835.
- Freely downloadable configuration utilities such as: CFA835 Font Editor, CFA835 Video Encoder, and CFA835 Graphic Test.
- Freely available source-code examples for PC or microcontroller interfacing.
- Crystalfontz America, Inc. is ISO 9001:2015 certified.
- A Declaration of Conformity with RoHS and REACH are available on the product's webpage.

## 2.2. Front Panel

The CFA835 family is available in two display colors:

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

The six key keypad is semi-transparent white on both color variants of the CFA835. The keypad is backlit with LEDs that match the LCD backlight color (white for TFK, blue for TML).

The status of the keys may be polled by the host or automatically reported by the module to the host upon key press and key release. See command 23 (0x17): Keypad Reporting for details.

The four 3mm LEDs located to the left of the LCD panel are bi-color (red & green) and are individually controllable / dimmable via the 34 (0x22): GPIO Pin Configuration command.

Front panel design and physical design information can be found in the Design and Mounting and Mechanical Drawings sections.

## 2.3. System Block Diagram

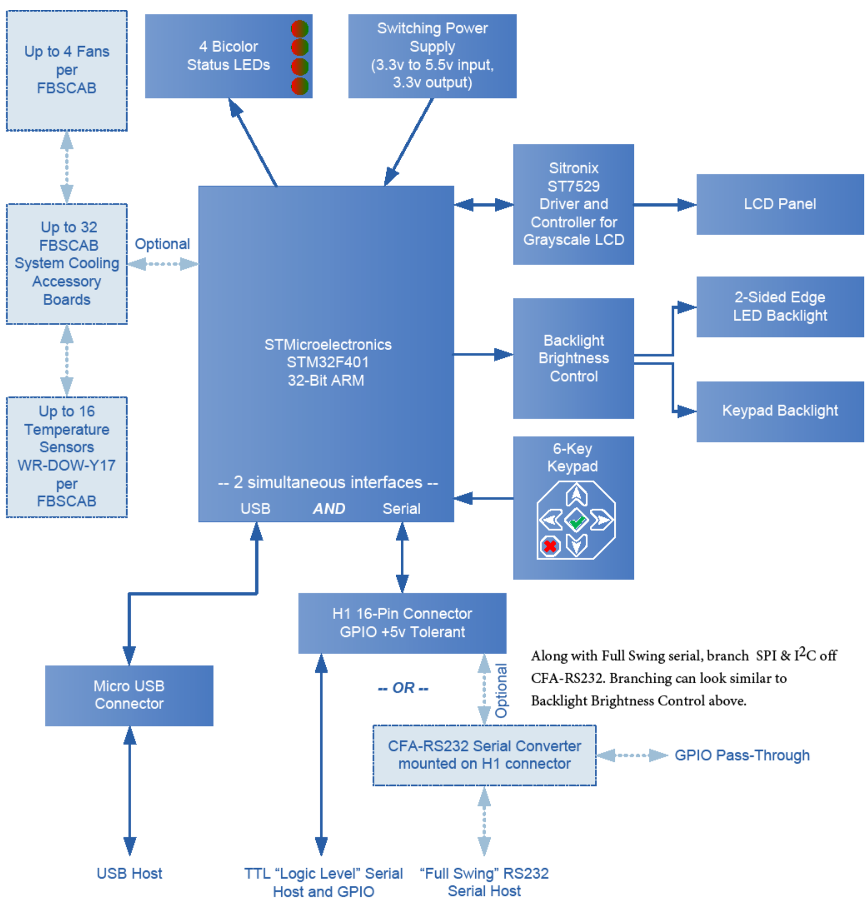



Figure 6. System Block Diagram

### 2.4. CFA-FBSCAB Features

The CFA-FBSCAB module (optional) can be connected to the CFA835 module to provide extra I/O functionality. The FBSCAB modules are added in a daisy-chain fashion. Up to 32 FBSCAB modules may be attached to a single CFA835.

The additional functions include:

- Manual or automatic PWM power control of up to four 12V fans (PC standard, 2- or 3-pin).
- RPM monitoring of up to four attached fans.
- Connection of up to 16 Dallas-one-wire (DOW) temperature sensors (WR-DOW-Y17).
- Five additional, host-controlled GPIO's
- The option to display live fan/temperature information to the CFA835 display without host interaction

Power steering jumpers are available on the FBSCAB module, so it may be configured to be supplied with 5V power from the attached CFA835 module, or by a PC "floppy-disk" style 5V/12V connector.
See the FBSCAB datasheet for more details.

When one or more FBSCAB modules are attached, use the CFA835 command 37 (0x25): CFA-FBSCAB to control/monitor them.

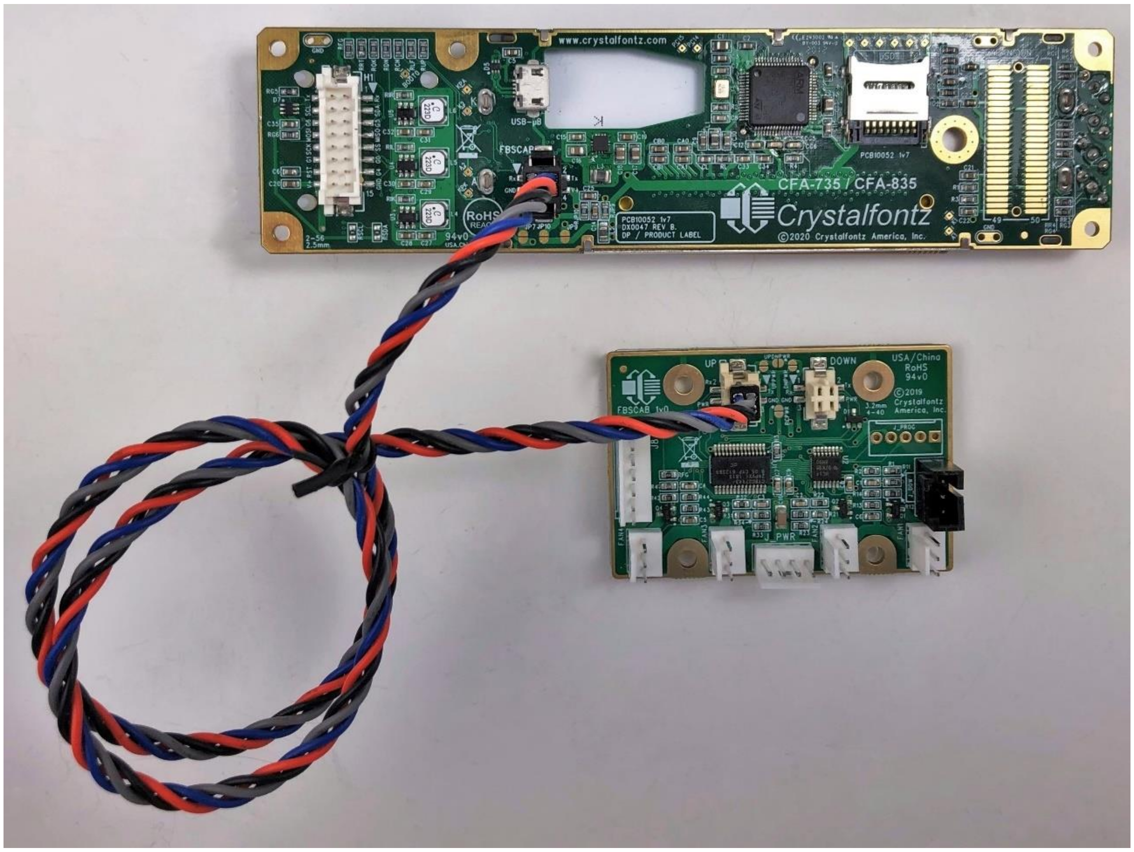



Figure 2. Optional CFA-FBSCAB Connected to the CFA835 with WR-EXT-Y37 Cable

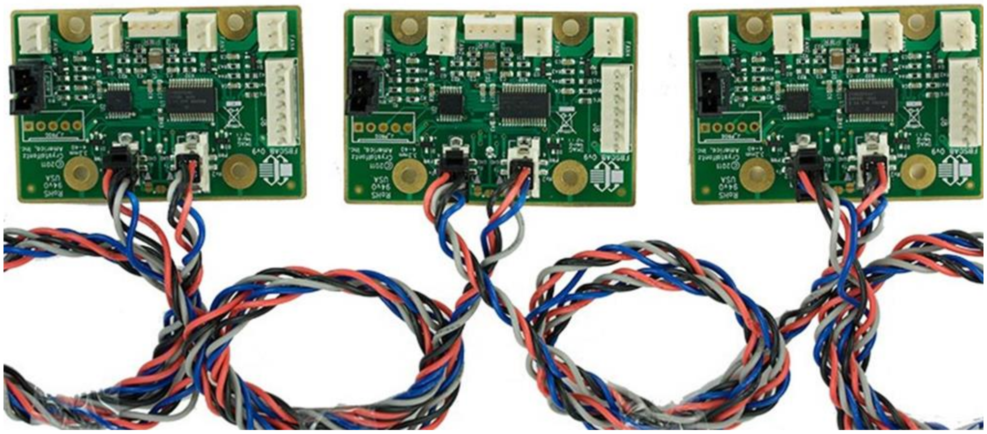


Figure 3. Example of CFA-FBSCABs Daisy-Chained Using the WR-EXT-Y37 Cable

**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.

### 2.5. Module Classification Information

CFA 835 - X X X
❶ ❷ ❸ ❹ ❺

| | | |
|---|---|---|
| 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<br>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<br>L – Transmissive, Wide Temperature -20°C to +70°C, 12:00 |

### 2.6. Ordering Information

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

For optional extras, click "*Customize and Add to Cart*" on the product page. Read more about Display Mounts , Cables and Serial Interface (Full-Swing RS232)

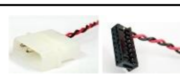
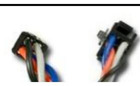
### 2.7. Display Mounts and Cables

The following display mounts are available after clicking "*Customize and Add to Cart*":

Figure 4. CFA835 Drive Bay Bracket


Figure 5. CFA835 SLED

The following cables are offered to simplify integrating the CFA835 into a system. Please note that cable lengths are approximate. Common configurations are described in Connection Information.

| Crystalfontz Cable | Image | Description<br>All Cables Are RoHS Compliant |
|---|---|---|
| WR-232-Y08<br>~27 inches |  | **For use with CFA-RS232:**<br>Connect cable's 10-pin socket connector to module's J_RS232 connector. Connect cable's RS232 DB9 9-pin socket connector to host's DB9 9-pin |
| WR-232-Y22<br>~26 inches |  | **For use with CFA-RS232:**<br>Connect one 10-pin socket connector to the module's J_RS232 10-pin connector. Connect cable's second 10-pin socket connector to host's motherboard 10-pin connector. Cable supports standard or alternate pinout motherboard RS-232 connections without changing module jumpers. |
| WR-232-Y23<br>~26 inches |  | **For use with CFA-RS232:**<br>Connect cable's 0.1" 2x5 socket connector to the CFA-RS232's J1 10-pin connector. Connect cable's RS232 DB9 9-pin socket connector to host's external 9-pin serial port. Choose standard or alternate pinout.<br>**NOTE**: Cable not listed on the CFA835 "*Customize and Add to Cart*" feature. Add the cable as a separate item to order. |
| WR-USB-Y27<br>~6 feet |  | **For use with USB:**<br>Connect cable's Micro-B USB connector to CFA835's Micro-B USB connector. Connect cable's USB-A connector to host's USB-A connector. |
| WR-USB-Y34<br>~27.5 inches |  | **For use with USB:**<br>Connect cable's Micro-B USB connector to CFA835's Micro-B USB connector. Connect cable's single piece 4-pin 0.1" socket connector to USB pins on host's motherboard. For correct orientation, note the +5v location on the 4-pin connector. |
| WR-PWR-Y24<br>~26 inches |  | **For use with ATX:**<br>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<br>~11 inches |  | **For use with ATX:**<br>Simplify 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<br>~13 inches |  | **For use with CFA-FBSCAB:**<br>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 4-pin receiver Molex connector. |
| WR-EXT-Y37<br>~18 inches |  | **For use with CFA-FBSCAB:**<br>Use this cable to connect the CFA835 to the CFA-FBSCAB. |
| WR-PWR-Y38<br>~2 ft. 11 inches |  | **For use with CFA-RS232:**<br>Longer version of the WR-PWR-Y25 (described above). |
| WR-FAN-X01<br>~16 inches |  | **For use with CFA-FBSCAB:**<br>Fan extension cable for standard 3-pin fans. |
| WR-DOW-Y17<br>~12 inches + ~12 inches between connectors |  | **For use with CFA-FBSCAB**:<br>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 37.0 (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 | 77.3 (W) x 23.8 (H) | 3.04 (W) x 0.94 (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)<br>(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<br>(12 o’clock is the preferred direction for this module) | θ | CR≧2 | 40° | 45° | — | above,<br>12 o’clock |
| | θ | CR≧2 | 35° | 40° | — | below,<br>6 o’clock |
| | θ | CR≧2 | 40° | 45° | — | right,<br>3 o’clock |
| | θ | CR≧2 | 35° | 40° | — | left,<br>9 o’clock |
| Contrast Ratio | CR | — | 3.5 | 4.5 | — | — |
| Response Time | Trise | Ta=25°C | — | 120 | 180 | ms |
| | Tfall | Ta=25°C | — | 220 | 300 | 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 | 35° | 40° | — | above, 12 o'clock |
| | θ | CR≧2 | 30° | 40° | — | below, 6 o'clock |
| | θ | CR≧2 | 35° | 55° | — | right, 3 o'clock |
| | θ | CR≧2 | 35° | 45° | — | left, 9 o'clock |
| Contrast Ratio | CR | — | 5 | 7 | — | — |
| Response Time | $T_{rise}$ | Ta=25°C | — | 120 | 180 | ms |
| | $T_{fall}$ | Ta=25°C | — | 200 | 300 | 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 | | 0.27 | 0.29 | 0.31 | |
| | y | | 0.29 | 0.31 | 0.33 | |
| Luminance | | | 1080 | 1350 | | $Cd/m^2$ |
| LED Lifetime | | | | 50K | | hours |

# 4. Electrical Specifications

## 4.1. 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 | $V_{DD}$-$V_{SS}$ | -0.3v | +4.0v |
| 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 beyond those listed under DC Characteristics is not implied. Changes in temperature can result in changes in contrast.* | | | |

## 4.2. H1 GPIO Current Limits

| Typical GPIO Current Limits | Specification |
|---|---|
| Sink | 8 mA |
| Source | 8 mA |

## 4.3. H1 GPIO Pins

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

### 4.4. H1 ADC Pins (pins 5 and 6)

| DC Characteristics | Symbol | Specification | Maximum |
|---|---|---|---|
| ADC Input High Voltage | $V_{IH}$ | +3.3v | +5.0v sustained<br>+8.0v for short-periods |
| ADC Input Low Voltage | $V_{IL}$ | 0.0v | -5.0v sustained<br>-8.0v for short-periods |

### 4.5. 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 whether 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 |

# 5. Connection Information

## 5.1. 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.

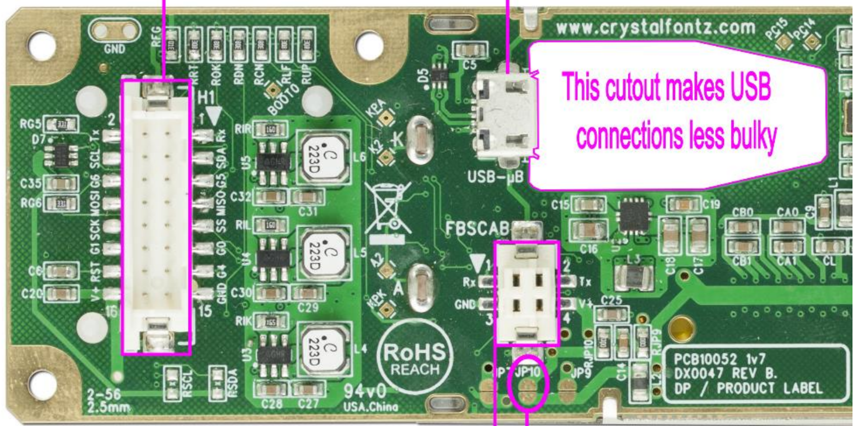



Figure 7. Location of CFA835 Connectors

## 5.2. H1 Connector Details

The H1 connector provides a simple method for controlling or monitoring external devices with the CFA835. Thirteen of the H1 pins may be configured separately as general-purpose inputs/outputs (GPIOs) or for specific control, communications, or ADC use (depending on the pin). The remaining three pins may be used for power-supply and external CFA835 reset control.

Pin functions are configured using a combination of the 33 (0x21): Interface Options, 34 (0x22): GPIO Pin Configuration and 28 (0x1C): ATX Functionality commands.

All pins are 5V tolerant, but as the microcontroller used on the CFA835 is 3.3V, outputs are limited to 3.3V high-level. See H1 GPIO Logic Level +5V Tolerant Pins for details.

H1 pins 5 and 6 are a special case as these are configured for ADC use. These two pins have an extra inline protection resistor, and power steering diodes. These pins can tolerate ±8V for a short amount of time. These pins also have a low-pass filter with a -3db roll-off at 27kHz. See H1 ADC Pins 5 and 6 for details.

#### 5.2.1. H1 Connector Pinout

| H1 Pin Number | Default Function | GPIO Number | Communications Function |
|---|---|---|---|
| 1 | Serial RX | GPIO[7] | Serial RX |
| 2 | Serial TX | GPIO[8] | Serial TX |
| 3 | GPIO[9] | GPIO[9] | I2C Slave SDA |
| 4 | GPIO[10] | GPIO[10] | I2C Slave SCL |
| 5 | ADC 0 | GPIO[5] | |
| 6 | ADC 1 | GPIO[6] | |
| 7 | GPIO[11] | GPIO[11] | SPI Slave MISO |
| 8 | GPIO[12] | GPIO[12] | SPI Slave MOSI |
| 9 | ATX Power Control | GPIO[2] | SPI Slave SS/CS |
| 10 | ATX Reset Control | GPIO[3] | SPI Slave SCK |
| 11 | GPIO[0] | GPIO[0] | |
| 12 | ATX Power Sense | GPIO[1] | SPI Slave INT (Data Ready) |
| 13 | GPIO[4] | GPIO[4] | I2C Slave INT (Data Ready) |
| 14 | CFA835 Reset | | |
| 15 | Power GND | | |
| 16 | Power +5V | | |

#### 5.2.2. Making a H1 Cable

The following parts may be used to make a 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).

### 5.3. Connecting Power and Data Communications through USB

The CFA835 has a USB peripheral, requiring only one connection to the host for both data communications and 5V power supply.

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

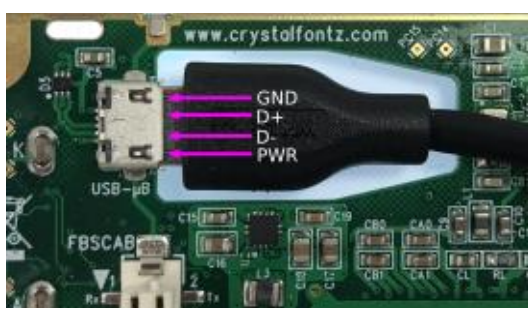
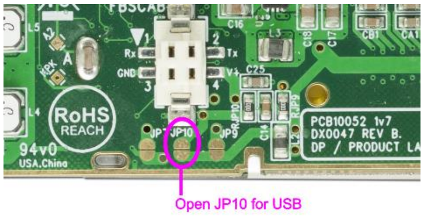



Figure 9. Connecting 5v Power Through USB

**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. To use USB interface while supplying power through H1, JP10 must be opened to prevent back-powering the USB.

# 6. 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 a system, simplifying front panel design.

**IMPORTANT**: The GPIO pins used for ATX control must not be configured as user GPIO, or be configured for use as a communications interface. 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. See commands 34 (0x22): Set or Set and Configure GPIO Pin and 33 (0x21): Interface Options.

### GPIO[1] ATX Host Power Sense

Since the CFA835 must act differently depending on whether the host's power supply is on or off, the host's "switched +5v" must be connected to GPIO[1]. This GPIO line functions as POWER SENSE. The POWER SENSE pin is configured as an input with a pull-down, 5kΩ nominal.

### GPIO[2] ATX Host Power Control

The motherboard's power switch input is connected to GPIO[2]. This GPIO line functions as POWER CONTROL. The POWER CONTROL pin is configured as a high impedance input until the LCD module instructs the host to turn on or off. Then it will change momentarily to low impedance output, driving either low or high depending on the setting of POWER INVERT. See command 28 (0x1C): Set ATX Power Switch Functionality.

### GPIO[3] ATX Host 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. 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. | |

**NOTE**: The CFA835 cannot control ATX functionality via connected FBSCAB's GPIO connector. ATX control must be performed via the CFA835's H1 connector.

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

The illustration below shows a Crystalfontz WR-PWR-Y25 or WR-PWR-Y38 ATX cable connected to the CFA835 H1 connector and a system's host and ATX Power Supply:

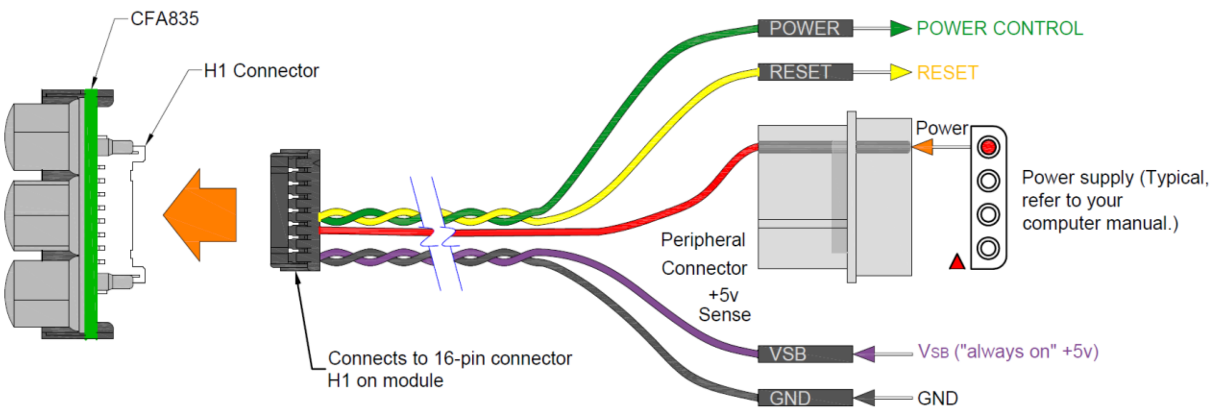



Figure 10. ATX Connection to H1 with WR-PWR-Y25 or WR-PWR-Y38 Cable

# 7. Firmware

## 7.1. How to Identify Firmware Revision Number

Before applying 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. The display clears five seconds after the arrow key is released.

Alternatively, 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. The display clears five seconds after the arrow key is released.

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

## 7.2. Possible Future Firmware Updates

CFA835 display modules are shipped with preinstalled firmware that performs the command functions described herein. Crystalfontz may make updates to the firmware in the future. Firmware updates are announced via PCN (Part Change Notices).

Any firmware updates will be available as a free download in the “Files” section on the product’s webpage. Updated firmware can be downloaded onto the CFA835 using one of the three methods detailed in Appendix B.

## 7.3. Custom Firmware

The CFA835 uses a STMicroelectronics STM32F401 microcontroller. The CFA10052 bootloader, CFA735, and CFA835 firmware are closed-source, and cannot be modified. However, the STM32F401 microcontroller may be completely erased and custom firmware programmed. This process requires the use of a STM32 compatible SWD programming interface. An open-source firmware project with some example LCD, keypad, etc. use can be found here.

**IMPORTANT**: If user-created firmware is loaded, the Crystalfontz firmware will be erased/overwritten. 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. Crystalfontz has no phone or email support for custom firmware.

## 7.4. Emergency Settings Reset

If the CFA835 cannot be recovered by a power off/on or reset command, attempt resetting the CFA835 back to firmware defaults using the following steps:

- Power off the CFA835 module.
- Press and hold the RIGHT and X keys.
- While holding the keys, power on the CFA835 module.

If this does not solve the problem, try following Appendix B: Firmware Update to update the CFA835 module’s firmware to the latest version.

# 8. Host Communications

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

## 8.1. USB Interface

### Windows Operating Systems

The easiest and most common way to communicate with the CFA835 is through USB. A link to Virtual COM Port (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 Linux as a Virtual COM port as `/dev/ttyACMx` (where x is the next available device number).

## 8.2. Serial Interface (Logic Level, Inverted)

A logic-level, inverted serial interface is available on the H1 connector – pins H1.1 (RX) and H1.2 (TX). Modules are shipped with the interface enabled with settings of 115200 baud, 8 data bits, no parity, 1 stop bit as default. If the interface is enabled, GPIO use of the same pins will be unavailable.
See command 33 (0x21): Interface Options or the Interface Configuration Screen for configuration options.

## 8.3. Serial Interface (Full-Swing RS232)

The CFA835 can be customized with a CFA-RS232 interface board to provide a "full-swing" industry standard, ESD protected, RS232 interface. When the CFA-RS232 board is fitted, the "Logic Level, Inverted" serial interface becomes inaccessible as both interfaces use the same H1.1 and H1.2 pins.

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

The CFA-RS232 Level Translator has a 16-pin socket connector J3 that mates with the 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.

## 8.4. I2C Slave Interface

The I2C slave interface is available on the H1 connector using pins H1.3 (SDA), H1.4 (SCL) and H1.13 (INT). The I2C interface is not enabled by default. To enable the interface, use command 33 (0x21):

Interface Options or the Interface Configuration Screen. If the I2C interface is enabled, GPIO use of the same pins will be unavailable.

Communication on the I2C interface is performed using the same packet structure as the other interfaces.

When the CFA835 has packet data available to read in its outgoing I2C buffer, the H1.13 data-ready/interrupt pin will be driven low. The H1.13 pin is open-drain so must be pulled-up externally. The host device should monitor the state of this pin and initiate a I2C data read sequence while it is low. The pin will return to a hi-z state when the outgoing buffer is empty.

Example Arduino source-code for communicating with the CFA835 on the I2C interface is available here.

### 8.5. SPI Slave Interface

The SPI slave interface is available on the H1 connector using pins H1.7 (MISO), H1.8 (MOSI), H1.9 (CS/SS), H1.10 (SCK) and H1.12 (INT). The SPI interface is not enabled by default. To enable the interface, use command 33 (0x21): Interface Options or the Interface Configuration Screen. If the interface is enabled, GPIO or other use of the same pins will be unavailable.

Communication on the SPI interface is performed using the same packet structure as the other interfaces (as described below).

The CS/SS (chip-select or slave-select) H1.9 pin is active low, and must be pulled-low before the host initiates SPI communications. It can be permanently tied low if the SPI bus is not shared with other devices.

When the CFA835 has packet data available to read in its outgoing SPI buffer, the H1.12 data-ready/interrupt pin will be driven low. The H1.12 pin is open-drain so must be pulled-up externally. The host device should monitor the state of this pin, and initiate a SPI data read sequence while it is low. The pin will return to a hi-z state when the outgoing buffer is empty.

Care needs to be taken when writing/reading from the SPI interface as there is no effective flow-control. Data may be dropped, or overflow the CFA835's incoming data buffer if sent too quickly.

Example Arduino source-code for communicating with the CFA835 on the SPI interface is available here.

### 8.6. Multiple Interface Communications

The CFA835 supports communication through all interfaces simultaneously unless otherwise noted. Keypad report packets are sent to all enabled interfaces. Command reply packets are sent to the interface from where the command packet originated.

See the interface option bits in command 33 (0x21): Interface Options for more details.

### 8.7. Interface Configuration Screen

A special interface configuration screen is available at all times. It can be accessed by pressing and holding the UP and RIGHT keys for 5 seconds. This screen allows the USB, Serial, I2C and SPI interfaces to be enabled/disabled and minimally configured.

The UP/DOWN keys select the item, the TICK key toggles enabling the interface and the LEFT/RIGHT keys change the interface configuration value (if available). When changes are complete the X key saves settings and reboots the CFA835.

**NOTE:** While the interface configurations screen is displayed, all other CFA835 functions cease, including all communications with the host device.

### 8.8. 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.

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;
```

### 8.9. Packet Error Reporting

The CFA835 supports returning 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

### 8.10. Handshaking / Flow Control

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

Reconciling packets is recommended rather than using delays when communicating with the LCD module. To reconcile packets, 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.

If very fast packet communications are required, more than one packet may be sent at a time. The CFA835 has a 1024-byte incoming data buffer for each interface, except for USB which has a 2048-byte buffer. As long as these buffers are not over-filled, all received packets will be processed, and replies sent, in order of reception.

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.

### 8.11. Command Codes

For convenience, command code links are grouped by type in the following list. The subsequent list has commands listed numerically from 1 to 41.

| Communications |
|---|
| Command 0 (0x00): Ping Command |
| Command 1 (0x01): Get Module Information |
| Command 5 (0x05): Restart includes:<br>Reload Boot Settings<br>Restart Host (WR-PWR-Y25 ATX Power Switch Cable Required)<br>Power Off Host (WR-PWR-Y25 ATX Power Switch Cable Required)<br>CFA835 Restart<br>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:<br>Subcommand 0: Load Custom Font Files from MicroSD Card<br>Subcommand 1: Print Custom Font to Display |
| Command 40 (0x28): Display Graphic Options includes:<br>Subcommand 0: Graphic Options<br>Subcommand 1: Buffer Flush<br>Subcommand 2: Send Image Data to Display from Host<br>Subcommand 3: Display Image File from MicroSD Card on CFA835<br>Subcommand 4: Save Screenshot to MicroSD File<br>Subcommand 5: Pixel Data<br>Subcommand 6: Draw a Line<br>Subcommand 7: Draw a Rectangle<br>Subcommand 8: Draw a Circle |
| Command 41 (0x29): Video Playback Control includes:<br>Subcommand 0: Load a Video from MicroSD Card<br>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:<br>Function 1: KEYPAD_RESTART<br>Function 2: KEYPAD_POWER_ON<br>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:<br>Subcommand 0: Read CFA-FBSCAB Information<br>Subcommand 1: Fan Settings includes Set Fan Power, Fail-Safe and Glitch information<br>Subcommand 2: Read Fan Tachometers<br>Subcommand 3: Read DOW Device Information<br>Subcommand 4: Read WR-DOW-Y17 Temperature<br>Subcommand 5: GPIO Pin Levels<br>Subcommand 6: Reset and Search<br>Subcommand 7: Live Fan or Temperature Display<br>Subcommand 8: Automatic Fan Control |

| Micro-SD Operations |
|---|
| Command 38 (0x26): Custom Fonts includes:<br>Subcommand 0: Load Custom Font Files from MicroSD Card<br>Subcommand 1: Print Custom Font to Display |
| Command 39 (0x27): MicroSD File Operations includes:<br>Subcommand 0: Open/Close MicroSD File<br>Subcommand 1: Position Seek<br>Subcommand 2: Read File Data<br>Subcommand 3: Write File Data<br>Subcommand 4: Delete A File |
| Command 40 (0x28): Display Graphic Options includes:<br>Subcommand 3: Display Image File from MicroSD Card on CFA835<br>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 are the same as the low 6-bits of the type field of the command packet being acknowledged.

Experiment with these commands using the 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 = 0₁₀
data_length = 0 to 124
data[] = any arbitrary data
```

Successful return packet:

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

### 1 (0x01): Get Module Information

The CFA835 returns the hardware and firmware version or serial number to the host.

Command packet:

```
type = 0x01 = 1₁₀
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 = 65₁₀
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 = 65₁₀
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 = 2₁₀
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 = 66₁₀
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 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), performing a restart from one interface may 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)**

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)**

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 also restarts 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. 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. For the cursor to be visible, also 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 either hides the cursor or selects among four hardware generated cursor options. The current cursor style can also be read using this command.

Cursor Styles:

```
0 = hidden (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 are supplied, the display is set to the brightness of the first byte and the keypad is set to the brightness of the second byte. If one byte is supplied, both the keypad and display backlights are set to that brightness. This command can also be used to read the current brightness levels.

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 = 14₁₀
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 = 78₁₀
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 = 23₁₀
data_length = 0
```

Successful return packet (Read):

```
type = 0x40 | 0x17 = 0x57 = 87₁₀
data_length = 2
data[0] = current keypad press mask
data[1] = current keypad release mask
```

Command packet (Write):

```
type = 0x17 = 23₁₀
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 = 87₁₀
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 replace the function of the power and restart switches in a standard ATX-compatible system. The ATX Power Switch Functionality is stored by the command 4 (0x04): Store Current State as Boot State.

**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. 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 = 31₁₀
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 = 95₁₀
data_length = 0
```

### 32 (0x20): Read Text from the Display

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 = 32₁₀
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 four host interfaces available for use; USB 2.0, Serial (logic level, or "full-swing" RS232 with attached sub-board option), I2C (slave) and SPI (slave). All interfaces may be used at any time (when enabled with this command) including being used simultaneously. See the Connection Information section for details on physical/electrical connections on the H1 connector.

Settings changed by this command must be saved by command 4 (0x04): Store Current State as Boot State for the CFA835 to power-up/restart using the settings.

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

Some pins on H1 used for Serial / I2C / SPI interfaces are shared with other CFA835 functions (for example, GPIO, ATX power control, ADC).
**When an interface is enabled, it will override any other H1 pin use.**

For example, if the SPI interface is enabled, ATX power control will no longer be available.

Option flags (applies to all interfaces):

```
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 send report packets on this interface (reports 128)
bit3 = CFA835 will send errors from commands received on this interface
bit4 = CFA835 will send errors from commands received on any interface
```

Command packet (Read Interface Options):

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

Successful return packet (Read Interface Options):

```
type = 0x40 | 0x21 = 0x61 = 97₁₀

SERIAL / RS232 INTERFACE:
data_length = 3
data[0] = 0 (serial/rs232)
data[1] = option flags (see above)
data[2] = baud rate
  0 = 19200
  1 = 115200
  2 = 9600

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

SPI INTERFACE:
data_length = 4
data[0] = 2 (SPI)
data[1] = option flags (see above)
data[2] = SPI mode settings
  bit0 = SPI CPOL (0 = 1st edge, 1 = 2nd edge)
  bit1 = SPI CPHA (0 = polarity low, 1 = polarity high)
  bit2 = Bit first (0 = MSB first, 1 = LSB first)
  bit3-7 = reserved
data[3] = reserved
```

```
I2C INTERFACE:
data_length = 4
data[0] = 3 (I2C)
data[1] = option flags (see above)
data[2] = I2C address (0x00 to 0x7F)
data[3] = I2C bus speed (0-1 valid)
 0 = 100Khz
 1 = 400Khz
```

Command packet (Write Interface Options):

```
type = 0x40 | 0x21 = 0x61 = 97₁₀

SERIAL / RS232 INTERFACE:
data_length = 3
data[0] = 0 (serial/rs232)
data[1] = option flags (see above)
data[2] = baud rate
  0 = 19200
  1 = 115200
  2 = 9600

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

SPI INTERFACE:
data_length = 4
data[0] = 2 (SPI)
data[1] = option flags (see above)
data[2] = SPI mode settings
  bit0 = SPI CPOL (0 = 1st edge, 1 = 2nd edge)
  bit1 = SPI CPHA (0 = polarity low, 1 = polarity high)
  bit2 = Bit first (0 = MSB first, 1 = LSB first)
  bit3-7 = reserved
data[3] = reserved (value is ignored)

I2C INTERFACE:
data_length = 4
data[0] = 3 (I2C)
data[1] = option flags (see above)
data[2] = I2C address (0x00 to 0x7F)
data[3] = I2C bus speed (0-1 valid)
 0 = 100Khz
 1 = 400Khz
```

Successful return packet (Write Interface Options):

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

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

### 34 (0x22): GPIO Pin Configuration (including on-board LEDs and ADC inputs)

This command configures the GPIO pins as well as the PWM duty used to control the four bi-color on-module LEDs.

The CFA835 has thirteen pins available on the H1 connector for user-definable general-purpose input / output (GPIO), communications interfaces, analog to digital converter (ADC), and ATX power control functions. See section Connection Information for more information on H1 pinout and pin functions.

The architecture of the CFA835 allows great flexibility in the configuration of the GPIO pins. When pins are not used for a communication interface, they can be set as an input or output. When configured as a GPIO output, they can output constant high or low signals or a variable duty cycle 100 Hz PWM signal. When configured as a GPIO input, the CFA835 continuously polls the pins at 50 Hz. The previously polled level can be queried by the host using this command. The CFA835 also keeps track of 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.

When an H1 pin is configured as GPIO, it may also have one of a few drive modes (strong drive up/down, resistive pull up/down and hi-z). 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. See the ST-Micro STM32F401 datasheets for additional information.

H1 pins H1.5 and H1.6 in default mode are configured as 0 to 3.3V analog-to-digital (ADC) inputs and are sampled continuously at 11kHz. The sampled ADC values are averaged between the host reading the values using this command. The averaged value is multiplied by 16 to increase value accuracy over long sample periods. The minimum and maximum ADC values are also tracked between the host reading values using this command.

The ADC has 12-bit resolution, and uses a 3.3V reference voltage (min=3.27v max=3.39v).
To calculate the approximate (uncalibrated) voltage at the H1 ADC pins:

Average voltage = (returned-average-value) / 16.0 / 4096 * 3.30.
Minimum voltage = (returned-minimum-value) / 4096 * 3.30.
Maximum voltage = (returned-maximum-value) / 4096 * 3.30.

**NOTE:** Communications interface settings override GPIO settings. If a communications interface using H1 pins is enabled, GPIO configuration of those same pins will be ignored.
The Serial interface is enabled by default. To use the H1 connector pins H1.1 and H1.2 as GPIOs the Serial interface must first be disabled. See command 33 (0x21): Interface Options.

**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 default, the GPIO output setting, function, and drive mode are set 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.**

**NOTE:** The GPIOs do not have under/over voltage or over current protection. See section 5.3: Logic Level GPIO +5V Tolerant Pins regarding acceptable input/output voltages.

Free demonstration software cfTest may be used to check and configure the GPIO pins.

H1 connector GPIO indexes:

| GPIO Index | GPIO / LED Name | H1 Pin | Default Function |
|---|---|---|---|
| 0 | GPIO[0] | Pin 11 | Unused (Hi-Z) |
| 1 | GPIO[1] | Pin 12 | ATX Host Power Sense |
| 2 | GPIO[2] | Pin 9 | ATX Host Power Control |
| 3 | GPIO[3] | Pin 10 | ATX Host Reset Control |
| 4 | GPIO[4] | Pin 13 | Unused (Hi-Z) |
| 5 | LED 3 Green | | LED Off |
| 6 | LED 3 Red | | LED Off |
| 7 | LED 2 Green | | LED Off |
| 8 | LED 2 Red | | LED Off |
| 9 | LED 1 Green | | LED Off |
| 10 | LED 1 Red | | LED Off |
| 11 | LED 0 Green | | LED 100% On |
| 12 | LED 0 Red | | LED Off |
| 13 | GPIO[5] | Pin 5 | ADC 0 Input |
| 14 | GPIO[6] | Pin 6 | ADC 1 Input |
| 15 | GPIO[7] | Pin 1 | Serial TX |
| 16 | GPIO[8] | Pin 2 | Serial RX |
| 17 | GPIO[9] | Pin 3 | Unused (Hi-Z) |
| 18 | GPIO[10] | Pin 4 | Unused (Hi-Z) |
| 19 | GPIO[11] | Pin 7 | Unused (Hi-Z) |
| 20 | GPIO[12] | Pin 8 | Unused (Hi-Z) |

Command packet (GPIO Read):

```
type = 0x22 = 34_10
data_length = 1
data[0] = index of GPIO/GPO to read (0-20 valid)
```

Successful return packet (GPIO Read):

```
type = 0x40 | 0x22 = 0x62 = 9810
data_length = 4
data[0] = index of GPIO (see table above)
data[1] = pin output state
   ---- -RFS
   |||| ||||-- S = state at the last reading
   |||| |||--- F = at least one falling edge has
   |||| ||         been detected since the last poll
   |||| ||---- R = at least one rising edge has
   |||| |          been detected since the last poll
   ||||-|----- reserved
data[2] = pin PWM output value
data[3] = 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=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
   ||||        =======================================================
   ||||        0: Port unused for GPIO. It will take on the default
   ||||           function such as ATX, DOW or unused. The user is
   ||||           responsible for setting the drive to the correct
   ||||           value in order for the default function to work
   ||||           correctly.
   ||||        1: Port used for GPIO under user control. The user is
   ||||           responsible for setting the drive to the correct
   ||||           value in order for the desired GPIO mode to work
   ||||           correctly.
   ||||------- reserved, will return 0
```

Command packet (ADC Read):

```
type = 0x22 = 34_10
data_length = 1
data[0] = index of GPIO ADC to read (13-14 valid)

NOTE: the pin must be in ADC (default) mode for the following return packet to
be sent by the CFA835. If the pin is not in ADC mode, the above “GPIO Read”
format packet will be returned.
```

Successful return packet (ADC Read):

```
type = 0x40 | 0x22 = 0x62 = 98_10
data_length = 7
data[0] = index of GPIO ADC
data[1] = Average of samples since last read * 16 (low-byte)
data[2] = Average of samples since last read * 16 (high-byte)
data[3] = Minimum sample value since last read (low-byte)
data[4] = Minimum sample value since last read (high-byte)
data[5] = Maximum sample value since last read (low-byte)
data[6] = Maximum sample value since last read (high-byte)
```

Command packet (GPIO Configure/Write):

```
type = 0x22 = 34_10
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 (0-20 valid, see table above)
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=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 ADC or unused.
   ||||        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
   ||||           as intended.
   ||||------- reserved, must be 0
```

Successful return packet (GPIO Configure/Write):

```
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 configures or reads the power settings of 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 updated 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 setting 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:

(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 = 37₁₀
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 = 101₁₀
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 = 37₁₀
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 = 101₁₀
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:
00 means CRC was checked and passed
```

```
01 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 rising and 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.
||||        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.
||||------- 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.**

### 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. For multiple CFA-FBSCABs, searching may take up to 2 additional seconds.

Command packet:

**type = 0x25 = $37_{10}$**
**data_length = 1**
**data[0] = 6 (Reset & Search)**

Successful return packet:

**type = 0x40 | 0x25 = 0x65 = $101_{10}$**
**data_length = 1**
**data[0] = 6 (Reset & Search)**

### Subcommand 7: Live Fan or Temperature Display

The CFA835 with one or more attached FBSCABs may be configured to update a portion of the LCD with a live fan RPM, fan power, or DOW temperature sensor display. Once configured using this command, the CFA835 will continue to display the live reading on the LCD without host intervention. The Live Fan or Temperature Display is stored by Store Current State As Boot State (command 4), so the CFA835 can display fan speeds, fan power or DOW temperatures as soon as power is applied.

The live display is based on a concept of display slots. There are 8 slots, and each of the 8 slots may be enabled or disabled independently.
Any slot may be requested to display any data that is available. For instance, slot-0 could display temperature sensor 3 from FBSCAB number 1 in °C, while slot-1 could simultaneously display fan power of FBSCAB number 2, fan number 1.

Any slot may be positioned at any location on the LCD, as long as all the digits of that slot fall fully within the display area. It is legal to have the display area of one slot overlap the display area of another slot, but senseless. This situation should be avoided in order to have meaningful information displayed.

Command packet (Set Live Display Slot):

```
type = 0x25 = 37
data_length = 3 or 9
data[0] = 7 (Live Fan of Temperature Display)
data[1]: display slot (0-7)
data[2]: type of item to display in this slot
   0 = nothing (data_length must be 3)
   1 = fan speed (RPM) (data_length must be 9)
   2 = temperature (data_length must be 9)
   3 = fan power % (data_length must be 9)
data[3]: index of FBSCAB module for the specified sensor (0-31 valid)
data[4]: index of the sensor to display in this slot:
   0-16 are valid for temperatures
   0-3 are valid for fan speed (RPM) and fan power %
data[5]: number of digits to display
   for a temperature: 3 digits (-XX or XXX)
   for a temperature: 5 digits (-XX.X or XXX.X)
   for a fan speed: 4 digits (XXXX)
   for a fan speed: 5 digits (XXXXX)
   for fan power %: must be 3 digits (XXX)
data[6]: display column
   0-17 valid for a 3-digit temperature
   0-15 valid for a 5-digit temperature
data[7]: display row (0-3 valid)
data[8]:
   for temperature: temperature unit (0 = deg C, 1 = deg F)
   for fan speed: fan RPM divisor
   for fan power %: not used, value ignored
```

Successful return packet (Set Live Display Slot):

```
type = 0x40 | 0x25 = 0x65 = 101
data_length = 1
data[0] = 7 (Live Fan of Temperature Display)
```

Command packet (Read Live Display Slot Settings):

```
type = 0x25 = 37
data_length = 2
data[0] = 7 (Live Fan of Temperature Display)
data[1]: display slot (0-7)
```

Successful return packet (Read Live Display Slot Settings):

```
type = 0x40 | 0x25 = 0x65 = 101₁₀
data_length = 9
data[0] = 7 (Live Fan of Temperature Display)
data[1]: display slot (0-7)
data[2]: type of item to displayed in this slot
   0 = nothing
   1 = fan speed (RPM)
   2 = temperature
   3 = fan power %
data[3]: index of FBSCAB module for the specified sensor
data[4]: index of the sensor to displayed in this slot
   data[5]: number of digits to displayed
   data[6]: display column
data[7]: display row
data[8]:
   for temperature: temperature unit (0 = deg C, 1 = deg F)
   for fan speed: fan RPM divisor
   for fan power %: not used, value ignored
```

### Subcommand 8: Automatic Fan Control

A CFA835 with one or more attached FBSCABs can be configured to automatically control fan power levels based upon the temperature of an attached DOW temperature sensor. The CFA835 will slow down or speed up the specified fan (attached to an FBSCAB) to attempt to maintain a set target temperature. Once configured, the CFA835+FBSCAB will continue to automatically control fan speed without host/user intervention. Automatic Fan Control is one of the items stored by Store Current State As Boot State (command 4), so the CFA835 can resume automatic fan control as soon as power is applied.

Fan control operation:

- If the specified temperature sensor's temperature is below the target value, the fan power will be gradually decreased at a rate determined by the responsiveness setting.
- If the specified temperature sensor's temperature is above the target value, the fan power will be gradually increased at a rate determined by the responsiveness setting.
- If the calculated fan power is below the specified minimum fan power value, the fan will either remain at that minimum value, or turn off depending on the "minimum fan power" option bits.
- If the calculated fan power is above the specified maximum fan power value, the fan will remain at the maximum fan power.
- If the specified temperature sensor does not exist (or there is a problem reading its value), the fan will be set to the specified maximum fan power value.

Each of the four fans attached to each attached FBSCAB module may be setup for automatic fan control. However, the temperature sensor used for a fan's power control must be attached to the same physical FBSCAB module as the fan.

When automatic fan control is enabled for a fan, manual fan speed control (as set by subcommand 1) will be unavailable (command will succeed, but setting will be ignored).

The power of the fan as set by automatic fan control may be read using subcommand 1 as per normal.

The Live Fan and Temperature command (command 37, subcommand 7) may be used display current automatic fan control fan power levels, along with fan RPM readings and temperatures on the display.

Command packet (Set Automatic Fan Control):

```
type = 0x25 = 37
data_length = 4 or 8
data[0] = 8 (Automatic Fan Control)
data[1] = FBSCAB module index (0-31 valid)
data[2] = controlled fan number (0-3 valid)
data[3] = option bits
   RRRR -MME
   |||| ||||-- E = automatic fan control enabled (0=disabled, 1=enabled)
   |||| |||--- MM = minimum fan power options
         00 = if power is under minimum value, the minimum value is used
         01 = if power is under minimum value, fan power is turned off
         10 = reserved
         11 = reserved
   |||| |
   |||| |----- reserved
   ||||------- RRRR = responsiveness value (0=slow, 15=fast)
data[4] = monitored temp sensor DOW index (0-15 valid)
data[5] = target temperature + 128 (degrees celcius)
   -40 degrees = -40 + 128 = 88 (minimum valid value)
   127 degrees = 127 + 128 = 255 (maximum valid value)
data[6] = minimum fan power value % (0-99 valid) (see MM option bits)
data[7] = maximum fan power value % (1-100 valid)
   - must be higher than minimum value (data[6])
- also used for initial startup power value, and if specified temp sensor
does not exist.
```

Successful return packet (Set Automatic Fan Control):

```
type = 0x25 = 37
data_length = 1
data[0] = 8 (Automatic Fan Control)
```

Command packet (Read Automatic Fan Control):

```
type = 0x25 = 37
data_length = 3
data[0] = 8 (Automatic Fan Control)
data[1] = FBSCAB module index (0-31 valid)
data[2] = controlled fan number (0-3 valid)
```

Successful return packet (Read Automatic Fan Control):

```
type = 0x25 = 37
data_length = 8
data[0] = 8 (Automatic Fan Control)
data[1] = FBSCAB module index
data[2] = controlled fan number
data[3] = option bits (see above "Set Automatic Fan Control")
data[4] = monitored temp sensor DOW index
data[5] = target temperature + 128 (degrees celcius)
data[6] = minimum fan power value %
data[7] = maximum fan power value %
```

### 38 (0x26): Custom Fonts Command Group

The CFA835 uses a monochrome graphic LCD. It supports printing text using most any custom font in most any language. To support this functionality, Crystalfontz offers a utility to convert fonts to the 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 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. Custom font files must be created using the CFA835 Font Editor. The loaded font is printed to the display using 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 by 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 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 with a microSD card 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

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

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

Reads 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 is incremented by the amount of data read by this command. To read data from elsewhere in the file, use 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 123)
```

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

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 LCD is a 244 x 68 pixel monochrome / greyscale display. The sub-commands in this group manipulate this display. The CFA835 supports updating the display directly or using a buffer that can be flushed manually. This option is toggled 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 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 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. Crystalfontz recommends against this. If the 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 = 41₁₀
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 = 105₁₀
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 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.

Command packet:

```
type = 0x29 = 41₁₀
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 = 105₁₀
data_length = 1
data[0] = 1 (Video Control)
```

### 62 (0x3E): Debugging

Reserved for internal CFA835 debugging functions.

### 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
```

# 9. Character Generator ROM (CGROM)

To find the code for a given character, add the two numbers that are shown in bold for its row and column. For 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).

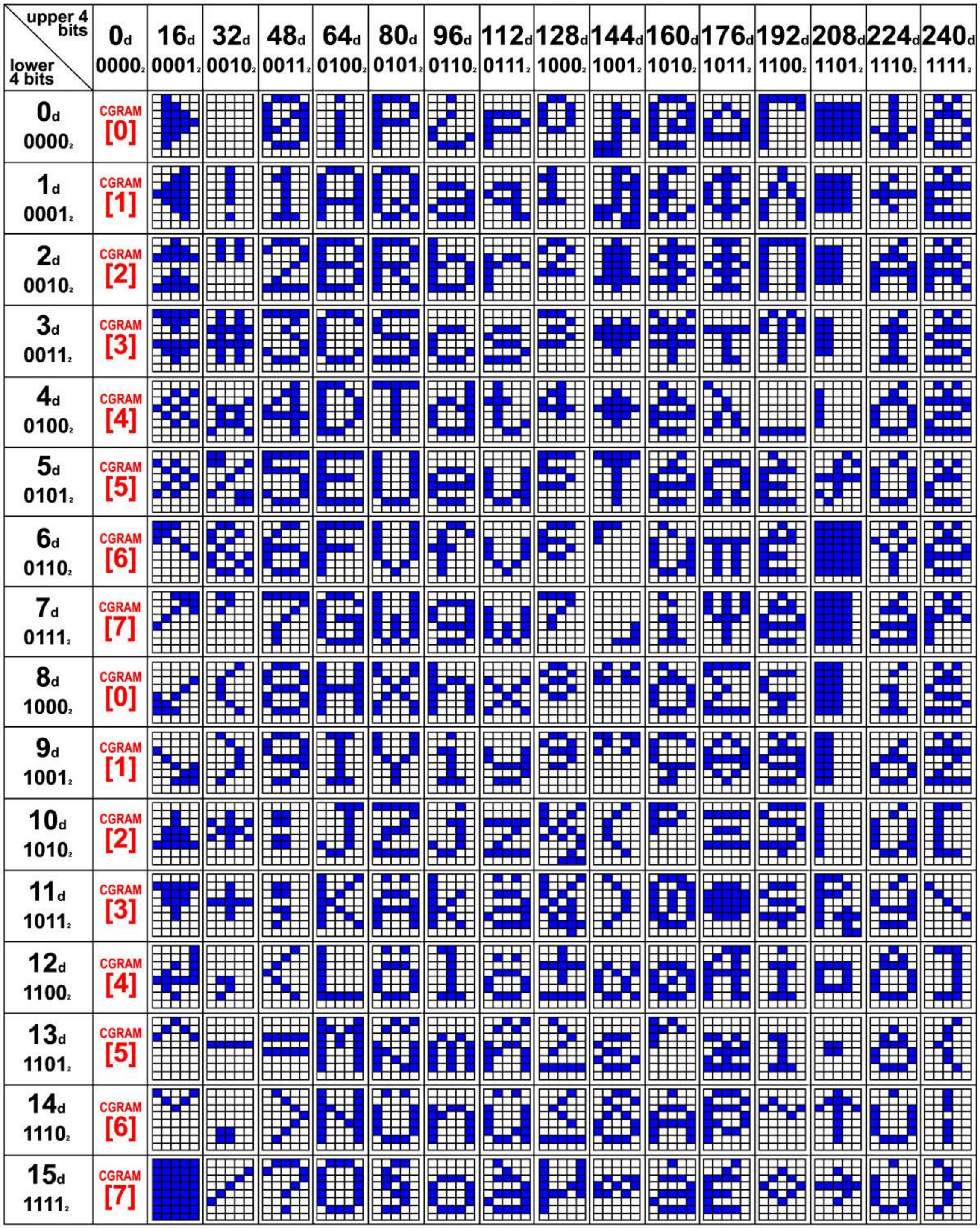



Figure 12. Character Generator (CGROM)

# 10. LCD Module Reliability and Longevity

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

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

## 10.1. Module Longevity (EOL / Replacement Policy)

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

We recognize that discontinuing a module may cause problems for some customers. However, rapidly changing technologies, component availability, or low customer order levels may force us to discontinue ("End of Life", EOL) a module. For example, we must occasionally discontinue a module when a supplier discontinues a component or a manufacturing process becomes obsolete. When we discontinue a module, we 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. However, sometimes a change in component or process for the replacement module results in a slight variation, often 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.

# 11. Care and Handling Precautions

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

## 11.1. 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.

## 11.2. Design and Mounting

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

## 11.3. Avoid 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.

## 11.4. If LCD Panel Breaks

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

## 11.5. 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.

## 11.6. 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.

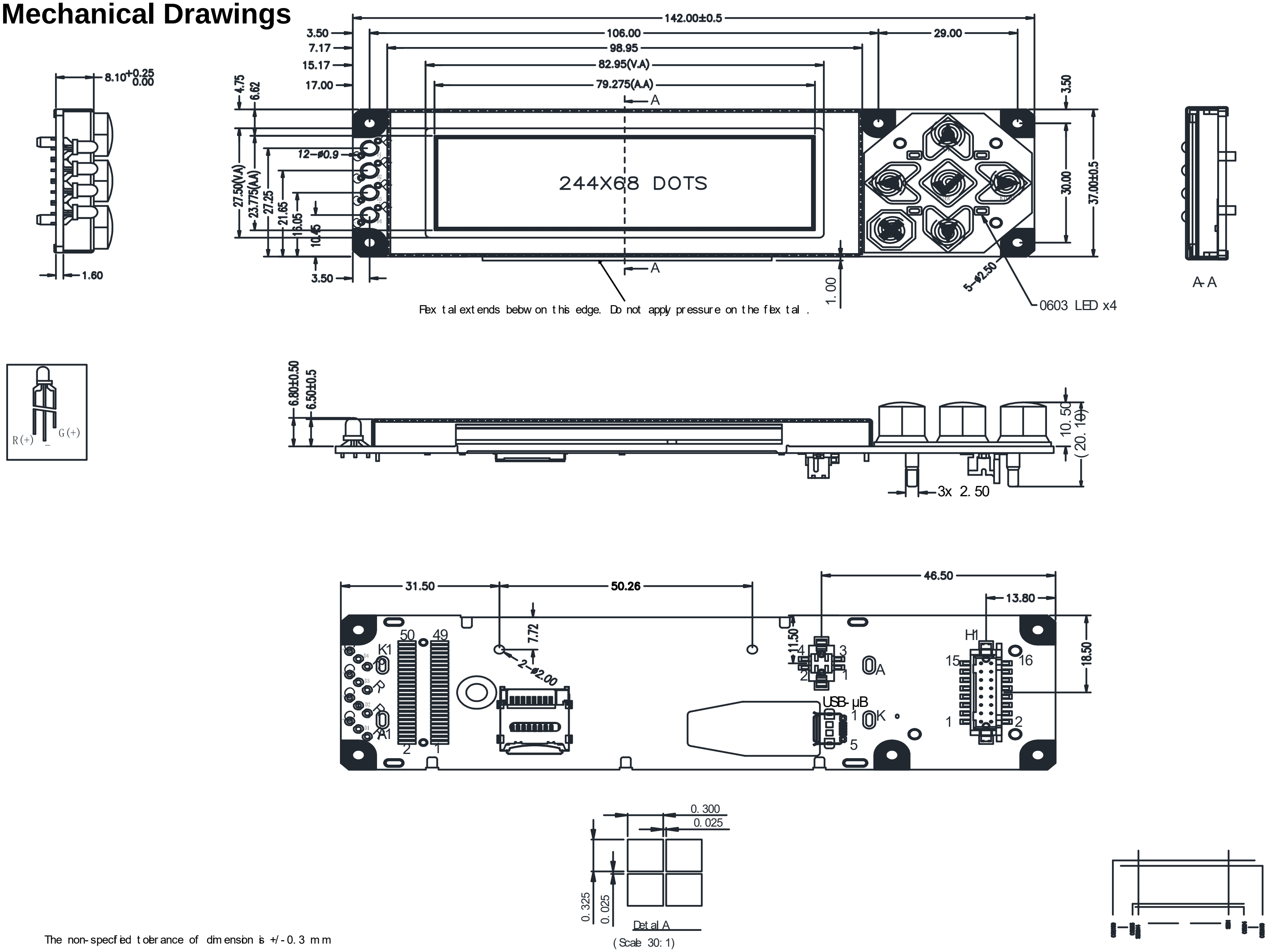
### 11.7. Storage and Recycling

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

### 11.8. Flat Flex Tail Care

- Damage to the flat flex tail can cause irreparable damage to the display. When handling the module, do not apply excessive pressure to the label covering the flex tail as doing so may cause tearing of the tail.

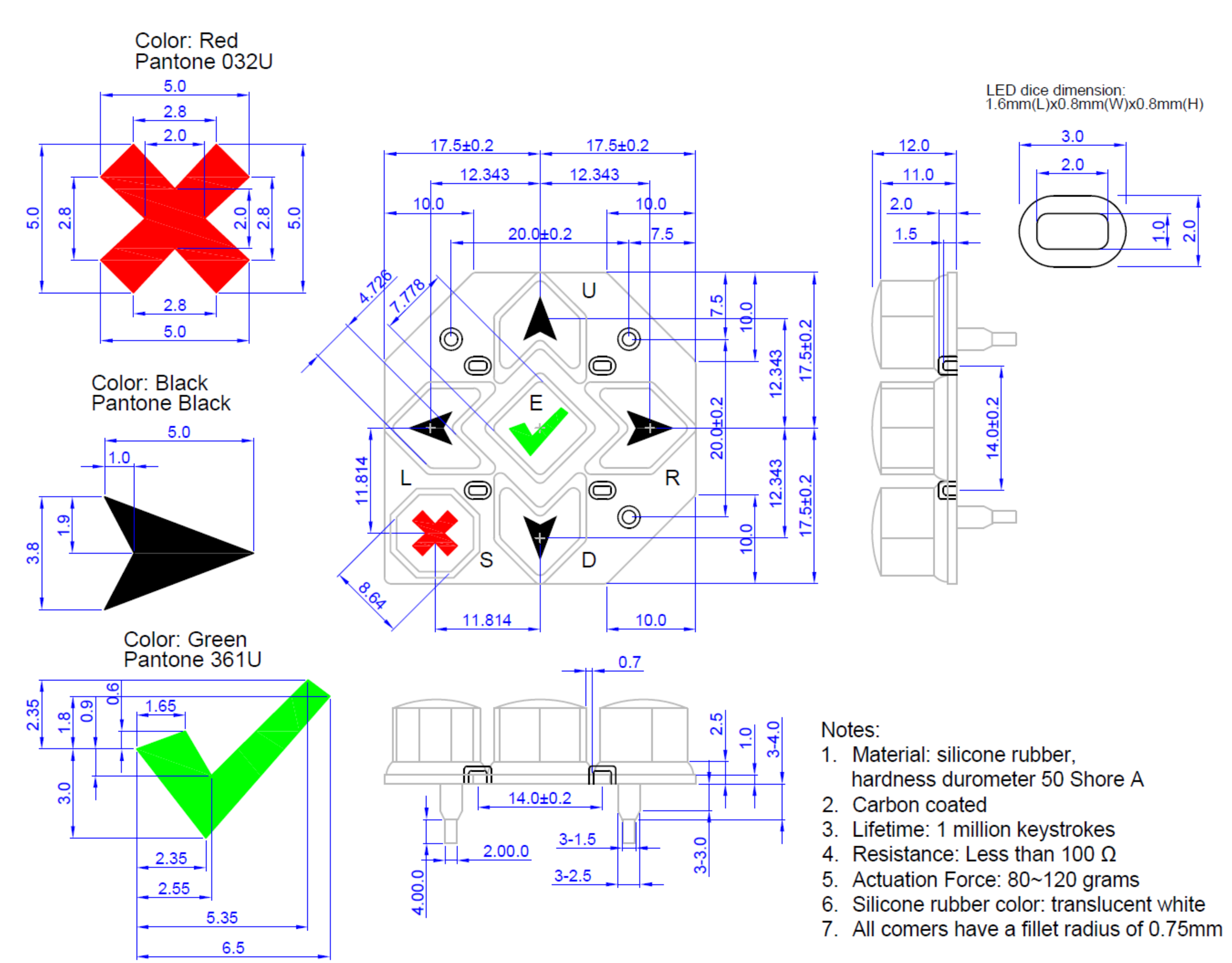
# 12. Mechanical Drawings




The non-specified tolerance of dimension is +/-0.3 mm

| Crystalfontz | Part Number: | Date: | Filename: | Revision: |
|---|---|---|---|---|
| | CFA835THK and CFA835TML | 2020-11-30 | CFA835 1v7 drawing | 1.0 |
| | | Web: | | Sheet: |
| | | www.crystalfontz.com/product/cfa835tml | | 1 of 1 |

Figure 15. Keypad Detail Drawing




| | | | | |
|---|---|---|---|---|
|  Crystalfontz America, Inc. www.crystalfontz.com/products/ | *Part No.(s):* 6-Button Keypad Detail | *Scale:* Not to scale | *Drawing Number:* 6-Button_Keypad_Detail | |
| | | *Units:* Millimeters | *Date:* 2017-04-28 | *Sheet:* 1 of 1 |

# 13. Appendix A: Demonstration Software and Sample Code

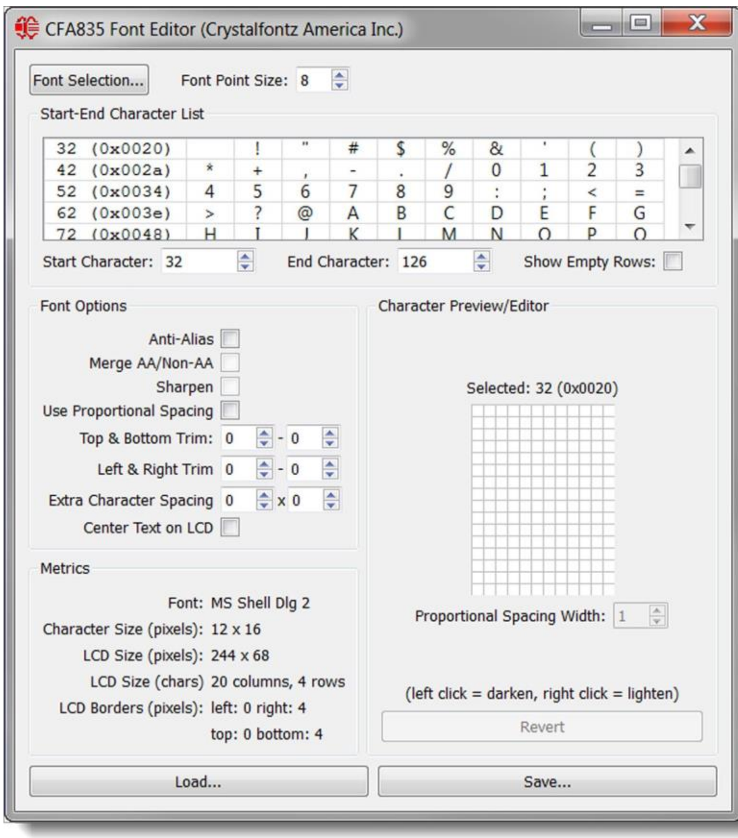
The CFA835 Window utilities described below are bundled together in a CFA835 Utilities Package.

## 13.1. Crystalfontz cfTest

cfTest for Windows is a testing and configuration software that works on all Crystalfontz Intelligent LCD modules. This software allows exploration of 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.

## 13.2. CFA835 Font Editor

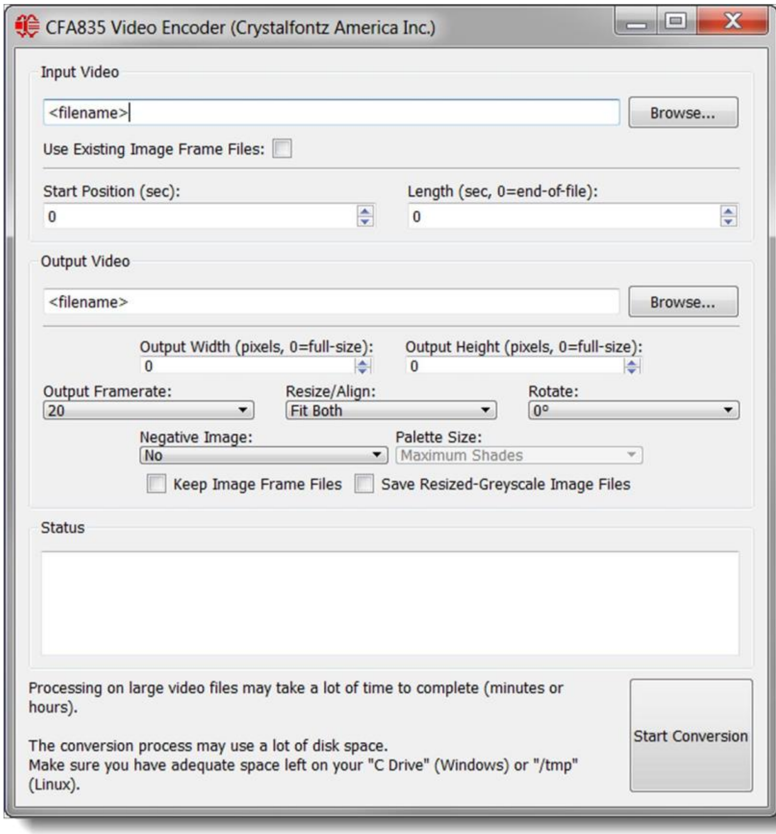



The CFA835 Font Editor converts any font into the CFA835 font format. The editor creates CFA835 compatible custom font files using fonts available on a PC. When the font file is loaded onto a microSD card inserted into the CFA835 card socket, the module can write custom font text to the display.

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.

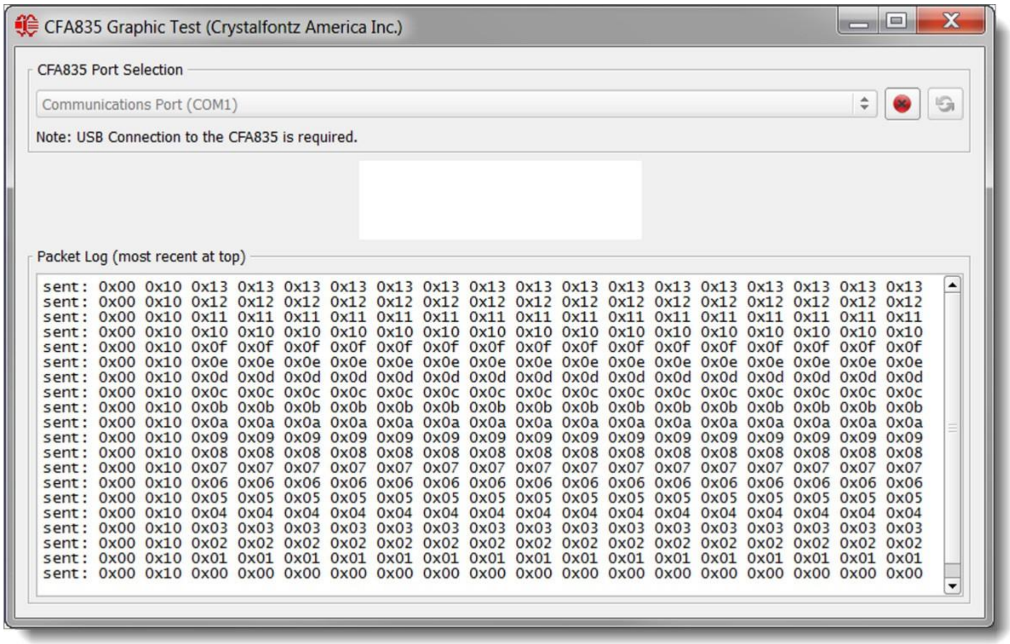
### 13.3. CFA835 Video Encoder




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.

### 13.4. 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.

### 13.5. 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.

### 13.6. 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;
}
```

### 13.7. 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;
  }
```

### 13.8. 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;
```

### 13.9. Sample Code

We encourage you to use the free sample code listed below. 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.

### 13.10. Algorithms to Calculate the CRC

Below are eight sample algorithms to 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.

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
;====================================================================
```

# 14. Appendix B: Firmware Update

**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.

---------------------------------------------------------------------