LCD MODULE WITH KEYPAD SPECIFICATIONS

Datasheet Release 2021-05-03
for
CFA533 I2C Modules
CFA533-TMI-KC
CFA533-TFH-KC
CFA533-YYH-KC

Hardware Version: v1.4
Firmware Version: c1v2

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

Table of Contents

1. General Information

Datasheet Revision History
Datasheet Version: **2021-05-03** Hardware Version: **v1.4** Firmware Version: c**1v2** This datasheet reflects hardware version v1.4, firmware c1v2 for the CFA533 family of LCD modules. For information about firmware and hardware revisions, see the Part Change Notifications (PCNs) at the bottom of the product page. Previous datasheet version: **2015-09-25** For reference, previous datasheets may be downloaded by clicking the “Show Previous Versions of Datasheet” link under the “Datasheets and Files” tab of the product web page.

Product Change Notifications
To check for or subscribe to “Part Change Notices” for this display module, see the Product Notices tab on the product’s webpage.

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

Volatility
This display module has non-volatile memory capability.

Disclaimer
Certain applications using Crystalfontz America, Inc. products may involve potential risks of death, personal injury, or severe property or environmental damage (“Critical Applications”). CRYSTALFONTZ AMERICA, INC. PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. Inclusion of Crystalfontz America, Inc. products in such applications is understood to be fully at the risk of the customer. In order to minimize risks associated with customer applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazard. Please contact us if you have any questions concerning potential risk applications. Crystalfontz America, Inc. assumes no liability for applications assistance, customer product design, software performance, or infringements of patents or services described herein. Nor does Crystalfontz America, Inc. warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of Crystalfontz America, Inc. covering or relating to any combination, machine, or process in which our products or services might be or are used. All specifications in datasheets on our website are, to the best of our knowledge, accurate but not guaranteed. Corrections to specifications are made as any inaccuracies are discovered. Company and product names mentioned in this publication are trademarks or registered trademarks of their respective owners.

2. Introduction

The CFA533-xxx-KC is an intelligent LCD module with a keypad. The "KC" series use an I2C interface, thus only four total lines are required for bi-directional communication: power, ground, SDA, and SCL.

Other interfaces are available at Crystalfontz.com.

2.1. Main Features

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

2.2. Module Classification Information

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

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

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

2.3. Comparison to CFA633

The CFA533 family of modules is mechanically similar to the CFA633 family. The CFA533 can be an economical drop-in replacement for most CFA633 applications that do not need fan capabilities. The CFA533 family is compatible with the CFA633.

The CFA533 family incudes further features such as a 3.3v to 5v operating range, the ability to adjust the keypad brightness separately from the LCD backlight, a stainless-steel bezel, and a single voltage supply for both logic and back lighting.

2.4. Build Configurations

Modifications of the CFA533KC are available after the module is added to the cart. These modifications include:

- **CFA533xxxKC8** which adds a Molex 70543-0002 header to J_DOW
- **DBBKCFA533xxxKC** the CFA533 is mounted into a drive bay bracket
- **DBBKCFA533xxxKC8** both the header and the drive bay bracket

2.5. CFA533 family

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

FSTN has better contrast than STN.
Positive Image: The display can be read in normal office lighting, in dark areas, and in bright sunlight.
Negative Image: Display can be read in normal office lighting and in dark areas. May be difficult to read in direct sunlight.
Slight color variations from module to module and batch to batch are normal. If modules with consistent color are required, request a custom order by emailing sales@crystalfontz.com.

2.6. CFA533KC Accessories

Crystalfontz Cable	Image	Description All Cables Are RoHS Compliant
WR-PWR-Y12 ~13 inches		4-pin power splitter cable. Use this cable to plug a 4-pin “hard drive style” Molex power connector into the module's “floppy drive style” power connector, plus provides an additional 4-pin Molex connector socket.
WR-PWR-Y14 ~24 inches		ATX power cable. Turn an ATX power supply on and off, or power cycle the host through the module. Connect the cable’s 7-pin connector to the module’s J8 socket connector, other end connects to WOL connector. (Requires optional 7-pin socket connector at J8 on module. Select J8 connector after clicking “Customize and Add to Cart”.)
WR-PWR-Y44 ~39 inches		ATX power cable. Longer version of the WR-PWR-Y14.
WR-PWR-Y05 ~25 inches		ATX power cable to turn an ATX power supply on and off, or power cycle the host through the module. Similar to WR-PWR-Y14, but uses 4 individual 0.1” connectors instead of WOL connector.
WR-DOW-Y17 ~12 inches + ~12 inches between connectors		Connect (“daisy chain”) up to 32 of these DOW (Dallas One-Wire) DS18B20 temperature sensor cables. Requires optional DOW connector at J_DOW on module.
DBBK		The drive bay mounting bracket is available for sale after clicking the “Customize and Add to Cart” button, along with a list of options for different cables and connectors.

3. Mechanical Characteristics

3.1. Physical Characteristics

Item	Specification (mm)	Specification (inch, reference)
Module Overall Dimensions		
Width and Height	110.5 (W) x 35.0 (H)	4.35 (W) x 1.378 (H)
Depth without Keypad	20.1	0.79
Depth with Keypad and Connectors	25.90 (max)	1.02
Viewing Area	61.0 (W) x 15.8 (H)	2.402 (W) x 0.622 (H)
Active Area	56.20 (W) x 11.5 0(H)	2.213 (W) x 0.453 (H)
Character Size	2.95 (W) x 5.55 (H)	0.116 (W) x 0.219 (H)
Dot Size	0.55 (W) x 0.65 (H)	0.022 (W) x 0.026 (H)
Keystroke Travel (approximate)	2.4	0.094
Weight	41 grams (typical)	1.48 ounces

3.2. Jumper Locations

All jumpers are configurable, but not all jumpers will affect your interface. Open jumpers by removing the corresponding resistors as shown below. Close the jumpers by melting a ball of solder across their gap. Reopen the jumpers by removing the solder with a solder wick.

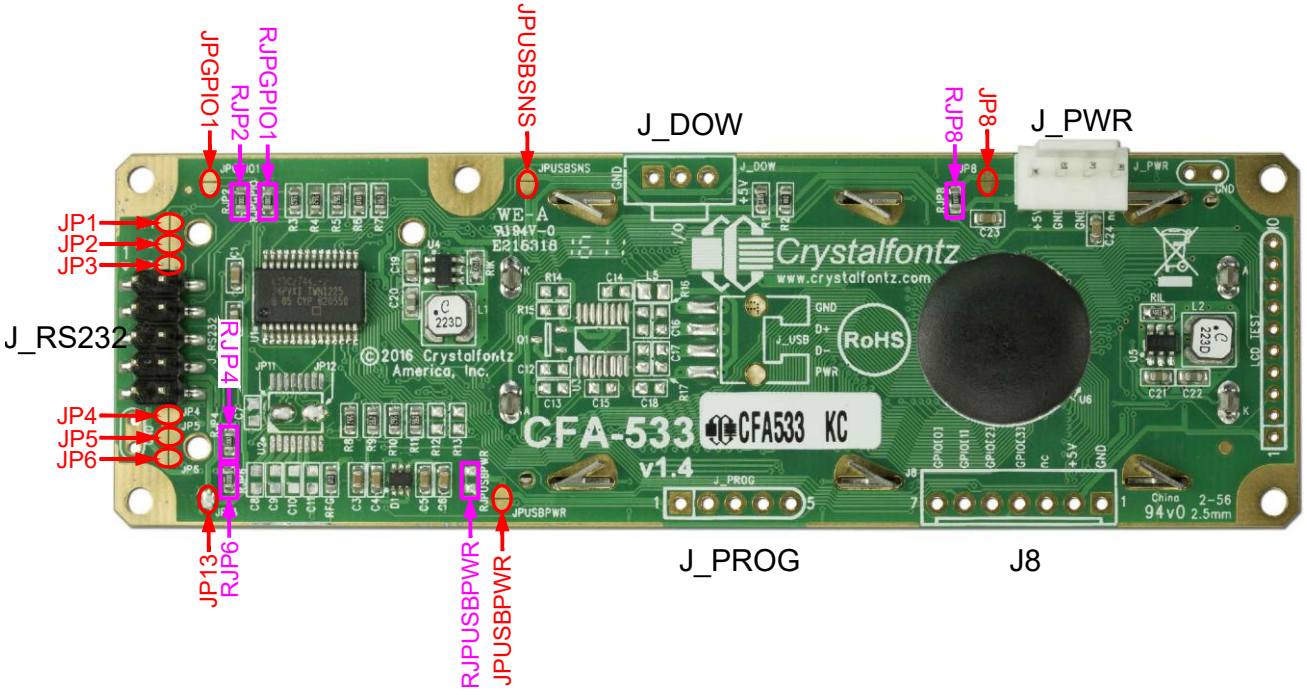
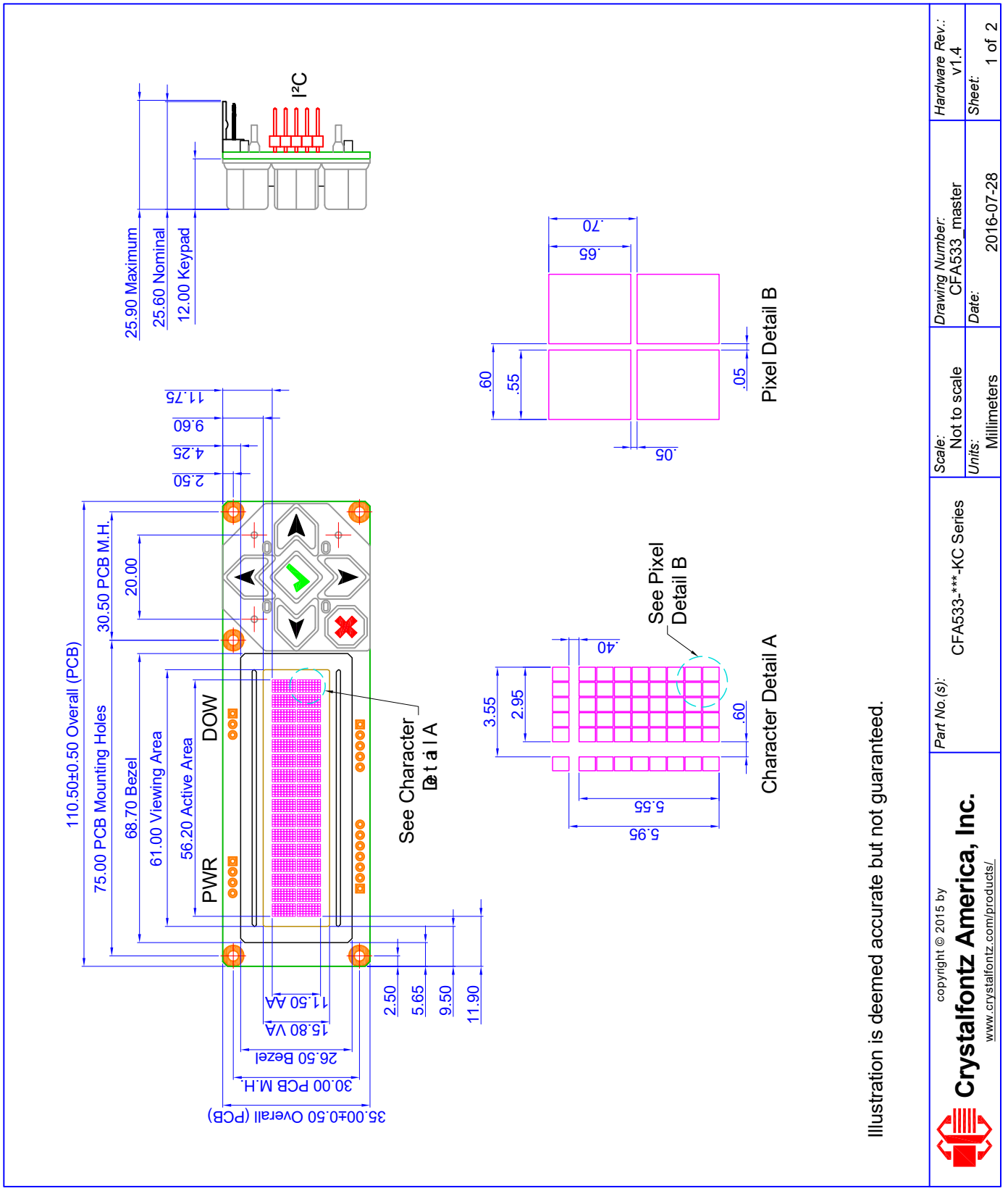
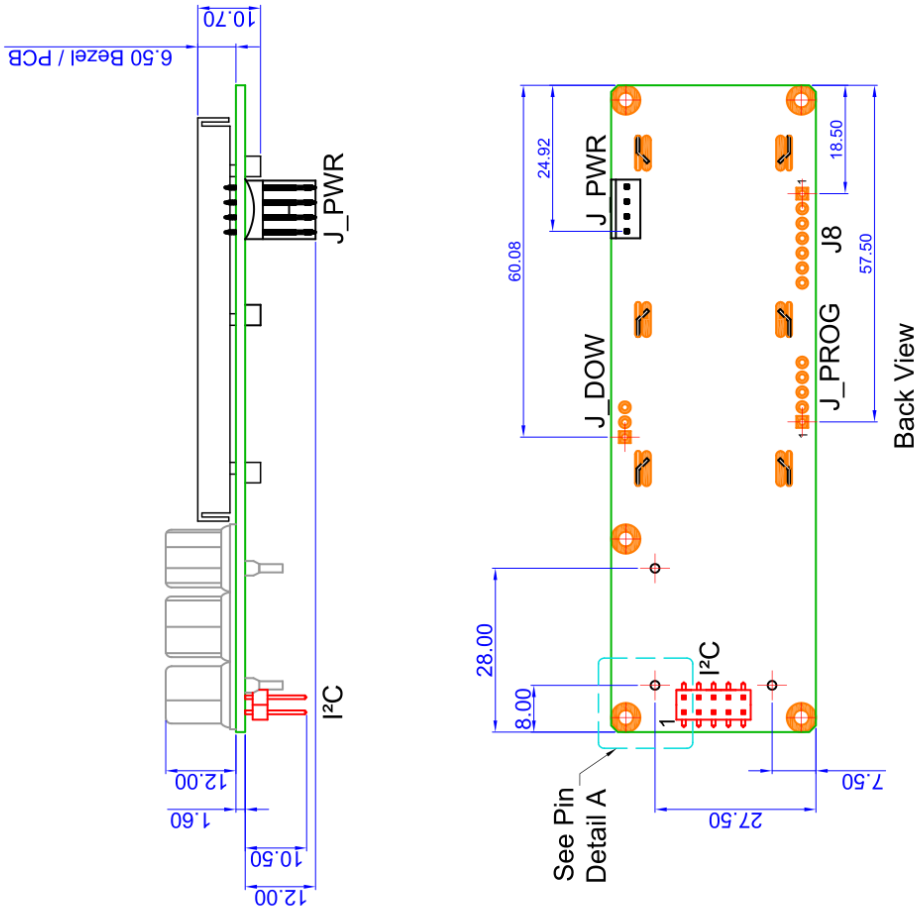


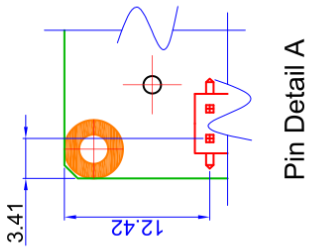
Figure 2. CFA533 HW v1.4 Jumper Locations

JUMPER	FUNCTION	-KC
JP1	Alternate RS232 Configuration	Open
JP2	Standard RS232 Configuration	Closed (0Ω RJP2)
JP3	Alternate RS232 Configuration	Open
JP4	Standard RS232 Configuration	Closed (0Ω RJP4)
JP5	Alternate RS232 Configuration	Open
JP6	Standard RS232 Configuration	Closed (0Ω RJP6)
JP8	**Connects the display's +5v to +5v on J_PWR. Conflicts with JPUSBSNS**	**Closed (0Ω RJP8)**
JP11	Connects the microprocessor's serial Tx line to JP1 and JP2	Closed
JP12	Connects the microprocessor's serial Rx line to JP3 and JP4	Closed
JP13	Connects the display's +5v to Pin 4 on JRS232	Open
JPUSBPWR	Connects the display's +5v to PWR on JUSB	Open
JPUSBSNS	Connects the display's ATX SENSE to PWR on J-USB. Conflicts with JP8	Open
JPGPIO1	Bypasses R3 when closed. R3 is a 5.6KΩ resistor in series with GPIO1	Closed (0Ω RJPGPIO)

3.3. Outline Drawings







Pin Detail A

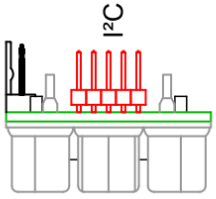
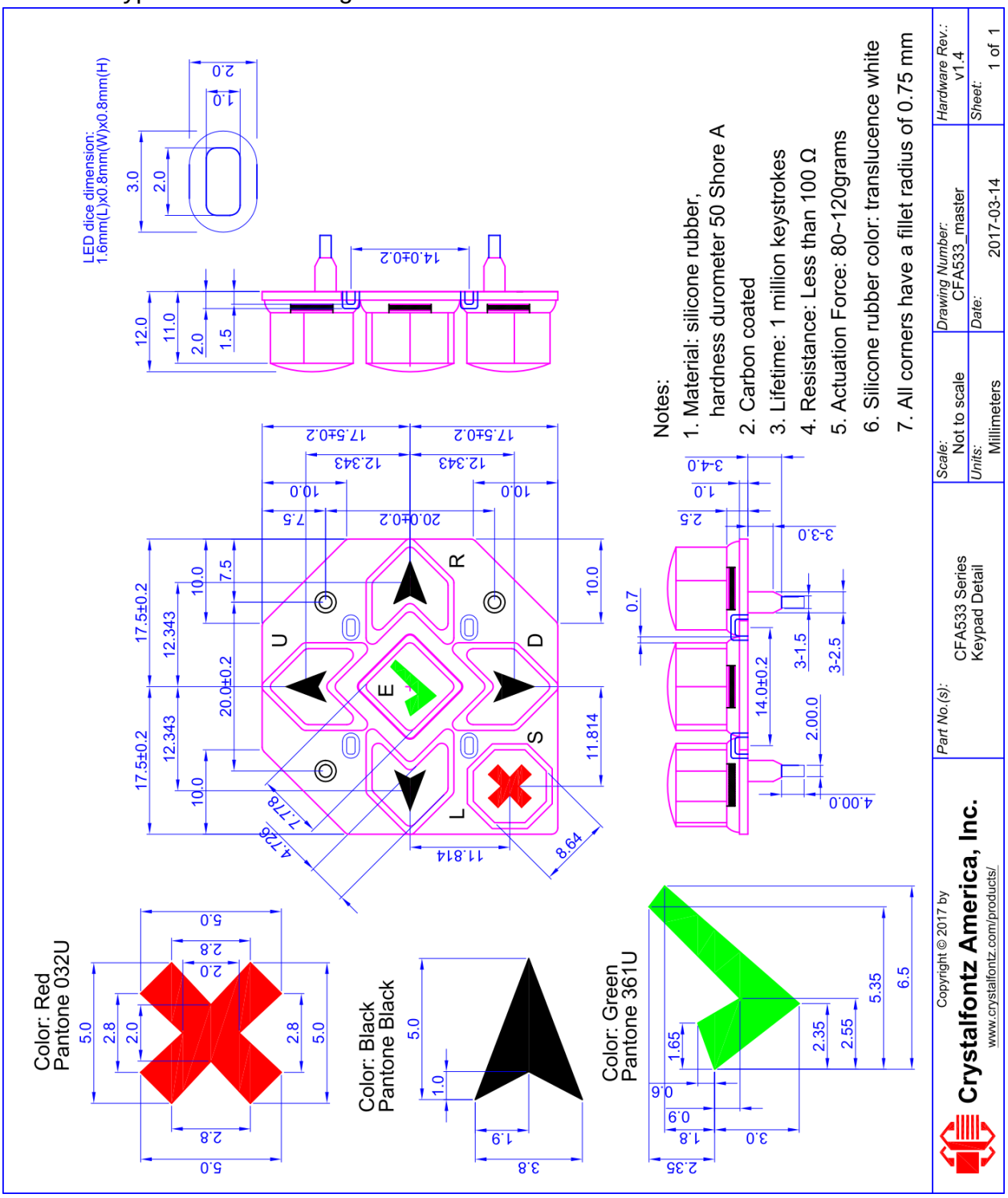


Illustration is deemed accurate but not guaranteed.

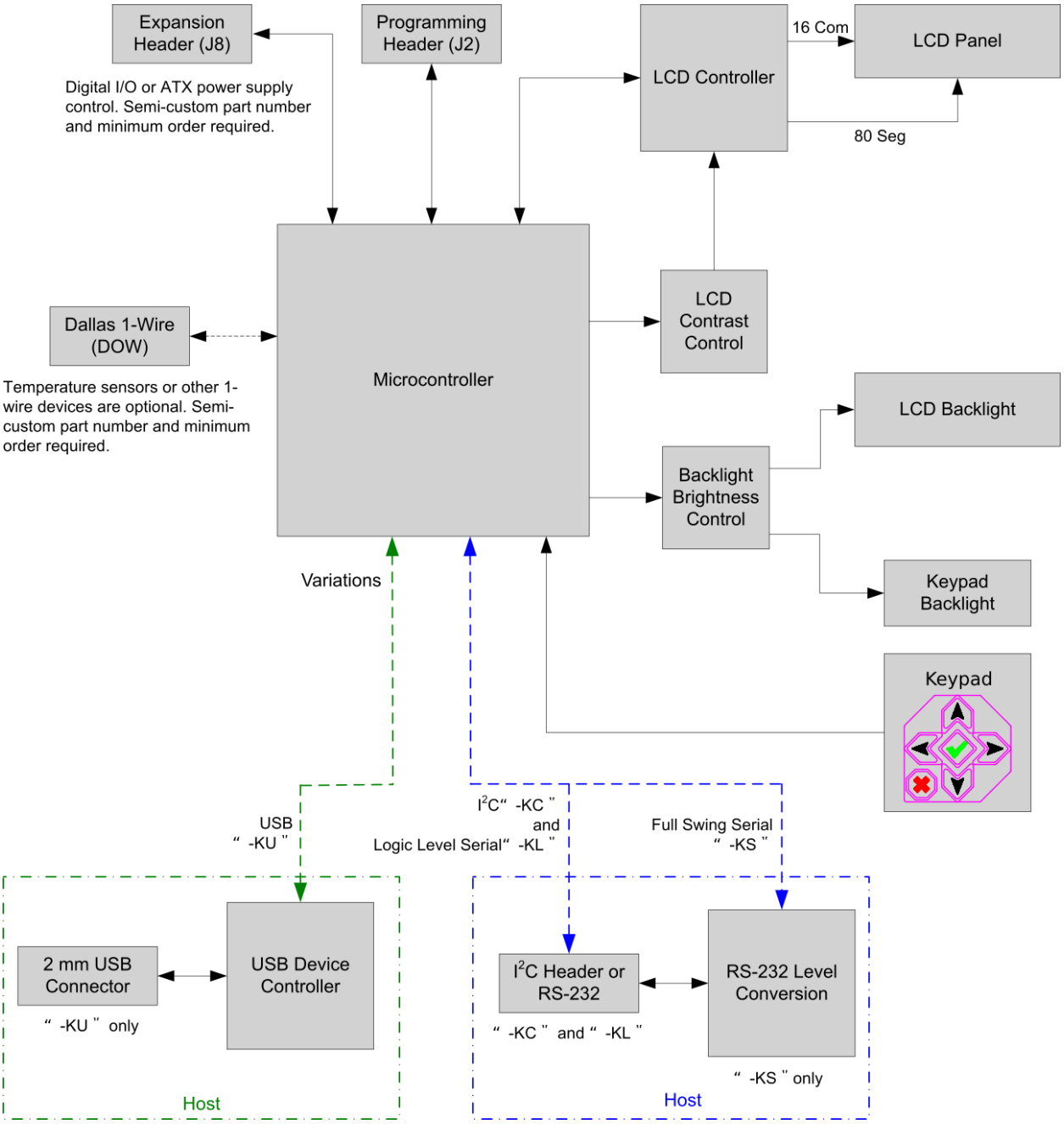
Hardware Rev.: v1.4			Sheet: 2 of 2
Drawing Number: CFA533_master			Date: 2016-07-28
Scale: Not to scale			Units: Millimeters
Part No.(s): CFA533-***-KC Series			

3.4. Keypad Detail Drawing



4. Electrical Characteristics

4.1. System Block Diagram



4.2. 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}	0v	+7.0v
These are stress ratings only. Extended exposure to the absolute maximum ratings listed above may affect device reliability or cause permanent damage. Functional operation of the module at these conditions beyond those listed under DC Characteristics is not implied.			

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

4.3. DC Characteristics

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

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

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

5. Optical Characteristics

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

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

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

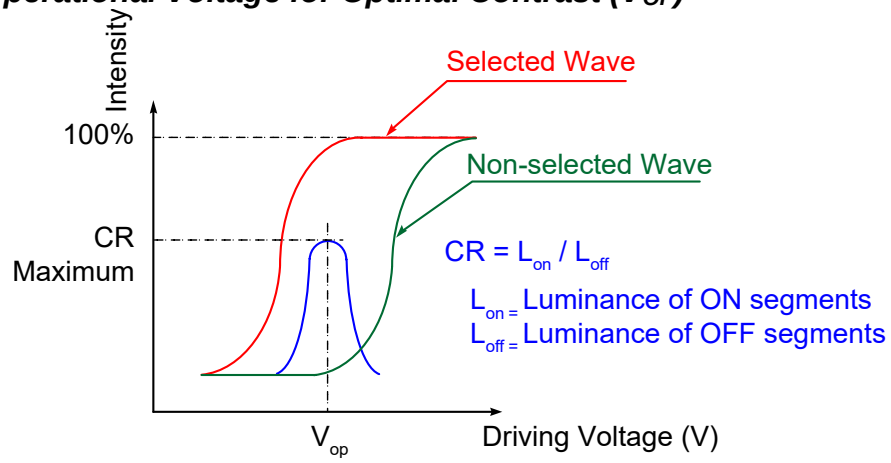
[1]Contrast Ratio = (brightness with pixels light)/ (brightness with pixels dark)
[2]Response Time = The amount of time it takes a liquid crystal cell to go from active to inactive or back again.

5.1. Test Conditions and Definitions for Optical Characteristics

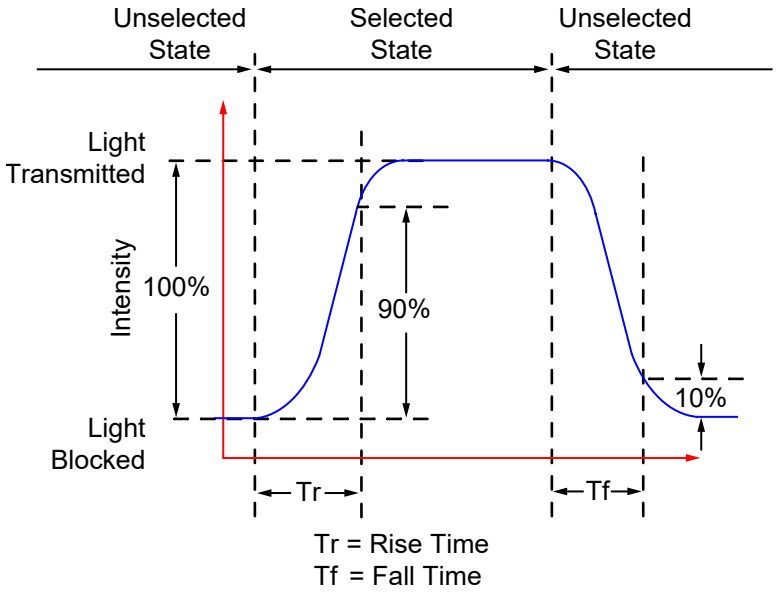
- Viewing Angle
  - Vertical (V)θ: 0°
  - Horizontal (H)φ: 0°
- Frame Frequency: 64 Hz
- Driving Waveform: 1/16 Duty, 1/5 Bias
- Ambient Temperature (Ta): 25°C

5.2. Optical Definitions for Negative Image Modules (CFA533-TMI-KC)

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

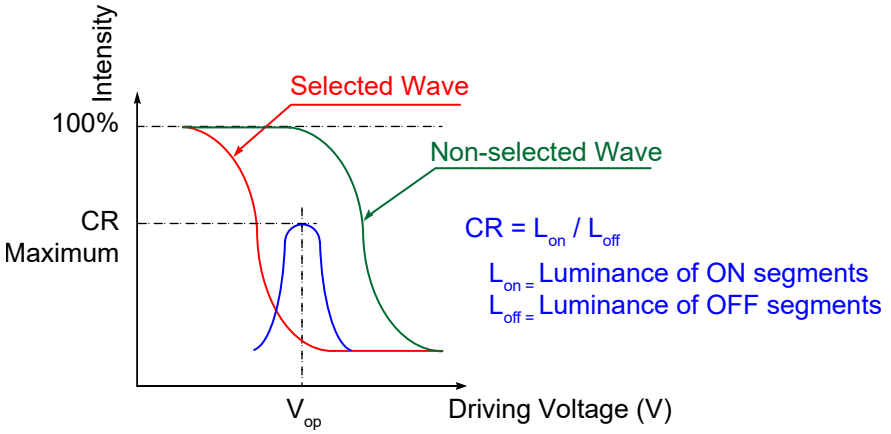


5.2.2. Response Time

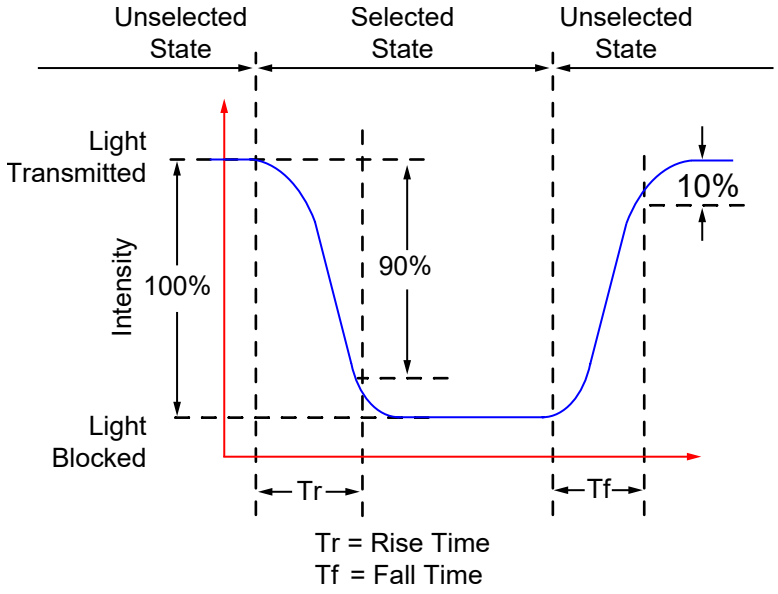


5.3. Optical Definitions for Positive Image Modules CFA533-TFH-KC and CFA533-YYH-KC

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



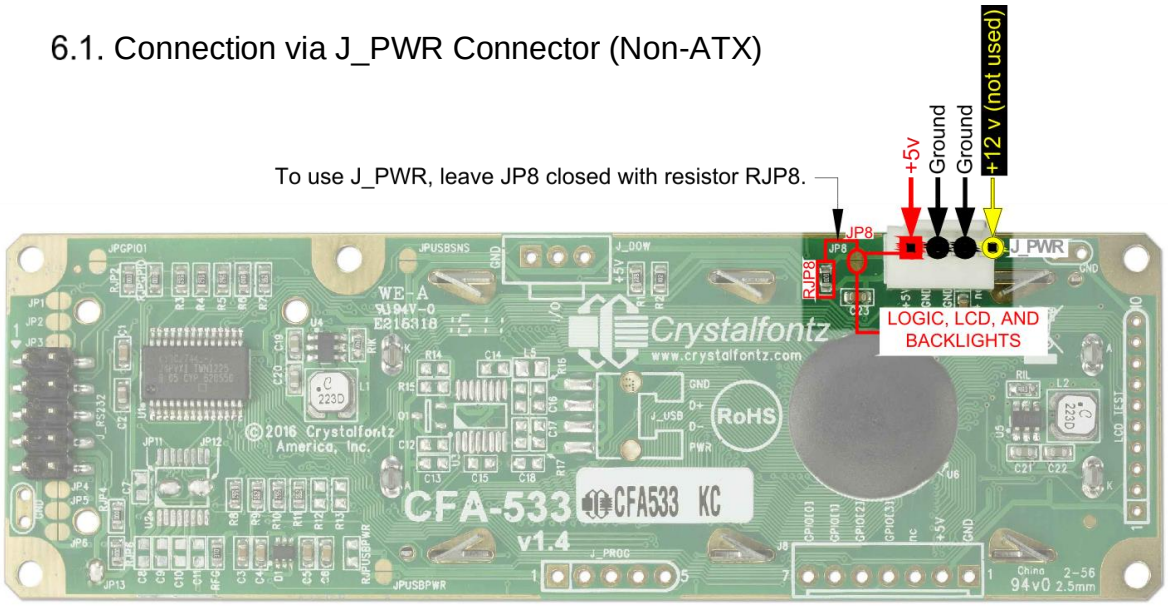
5.3.2. Response Time



6. Power Supply Connections

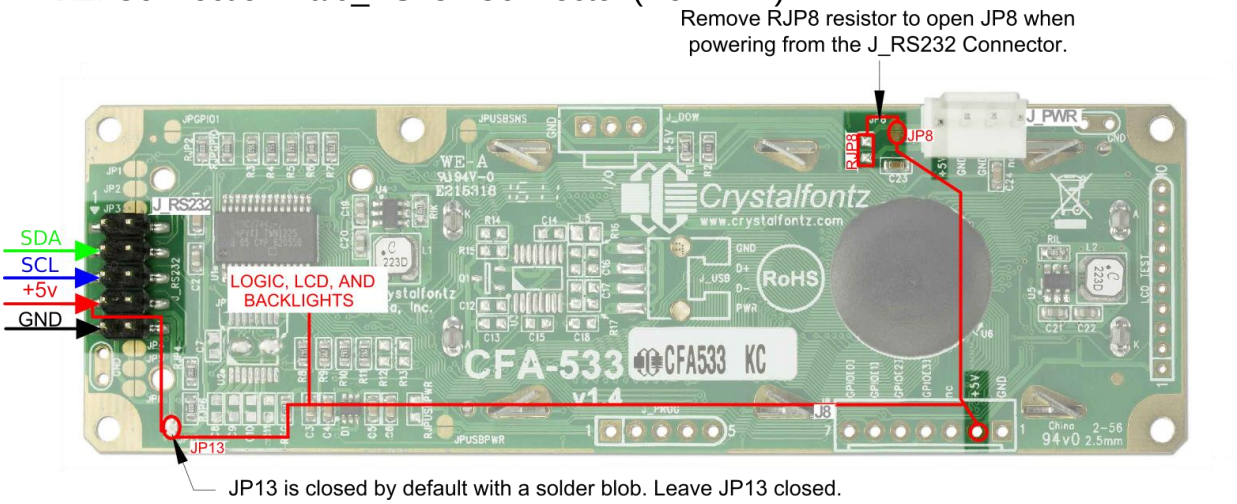
Power supply connections can be made with or without the ability to control power on/off and reset functions of an ATX power supply host. The connections described below are specified as ATX or non-ATX. CFA533KC standard parts do not ship with ATX ability. Contact support@crystalfontz.com to initiate a defined part for a CFA533KC with ATX capabilities. Otherwise, the modifications are described below to allow for this ability.

6.1. Connection via J_PWR Connector (Non-ATX)



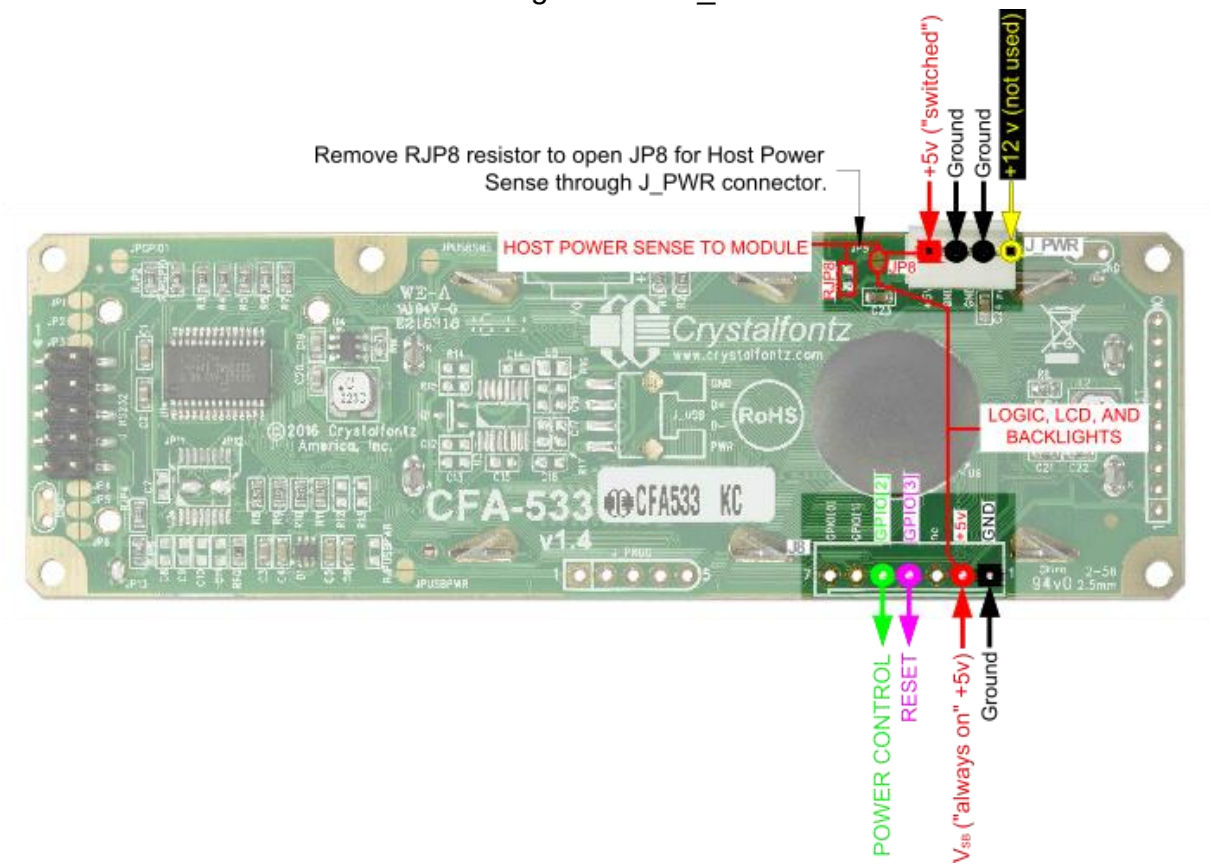
By default, JP8 is closed with the RJP8 0Ω resistor. To connect via the J_PWR connector, leave J8 closed.
Supply +5v to pin 1 and ground to either pin 2 or pin 3 on the J_PWR connector. This can be done with the WR-PWR-Y12.

6.2. Connection via J_RS232 Connector (Non-ATX)



The J_RS232 connector can be used as an I2C connector on CFA533KC modules. V_{DD} power can be supplied through the connector, allowing a single cable to bring in power and data connections. The five connections needed are all located on the same side of J_RS232, so a 0.1" 5-position cable can be used to connect the CFA533-KC to a main system, or an RS232 cable like the WR-232-Y22.
To connect via J_RS232, remove the RJP8 0Ω resistor to open JP8.

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



By default, J8 is closed by RJP8 and the +5v pin on the J_PWR connector is connected to the +5v net on the CFA533KC. In order to use J_PWR to sense the host power, **open JP8** to disconnect it the +5v pin on J_PWR from the +5v net. Then, the J_PWR +5v pin can be used at the "Host Power Sense."

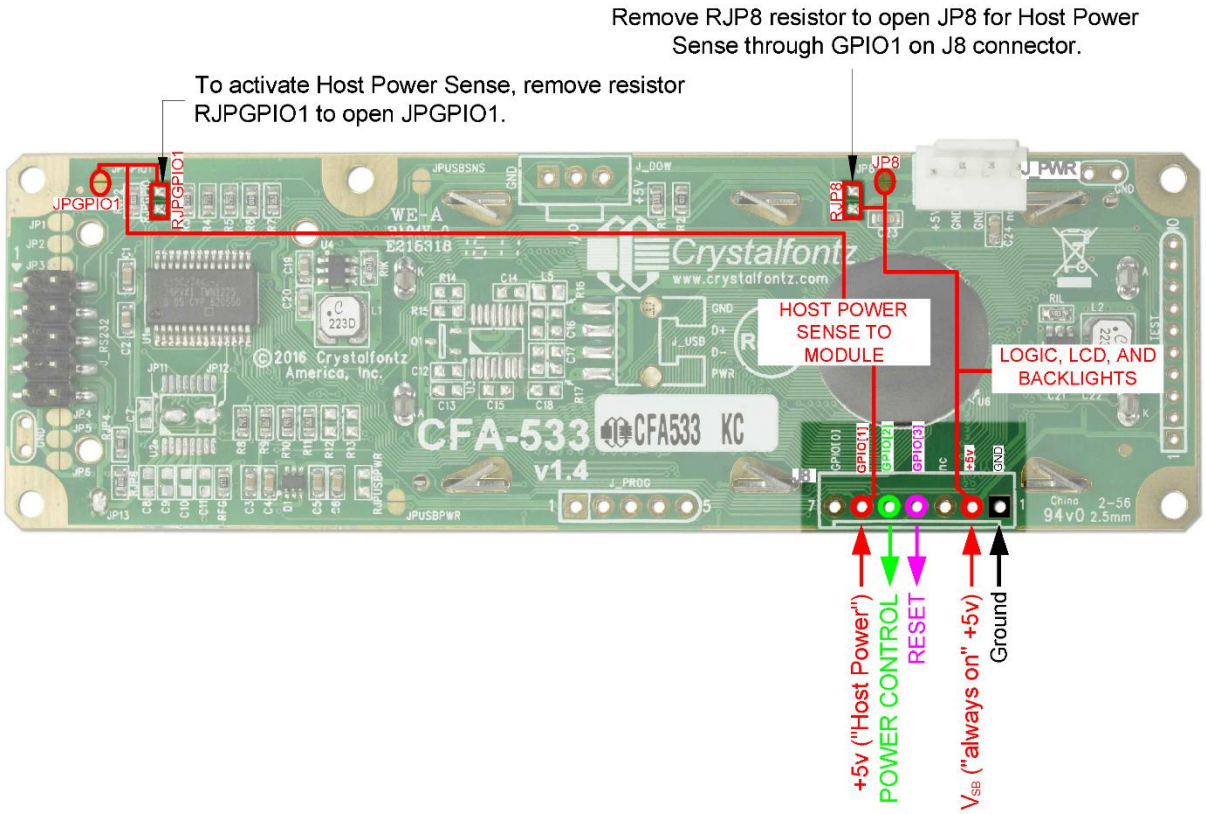
Connect the motherboard's power switch input to GPIO[2] (pin 5 on the J8 header). This will be the Power Control Pin, and is configured as a high-impedance input. When a host on or off command is sent, pin 5 will be changed to a low impedance output and drive either low or high depending on the setting of POWER_INVERT.

Connect the motherboard's reset switch input to GPIO[3] (pin 4 on the J8 header). This will be the Reset Pin, and is configured as a high-impedance input. When a Reset signal is sent to the host, pin 4 will be changed to a low impedance output, driving either low or high.

Crystalfontz offers two power cables to simplify ATX power supply control connections: the WR-PWR-Y14 and the WR-PWR-Y44. When using either of these cables, open JP8 by removing the RJP8 resistor.

Additional information about setting the ATX switch functionality can be found in Command 28.

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



For backwards compatibility with legacy CFA633 applications, the HOST POWER SENSE can alternately be provided through GPIO[1] (pin 6 of the J8 header). This configuration requires the +5v V_{SB}, Ground, Power Control (GPIO[2]), Reset Control (GPIO[3]) and +5v Host Power. In this method only, both RJP8 and RJPGPIO1 should be removed to open the corresponding jumpers.

6.5. ATX Keypad Control

Keypad control is configured using command 28 (0x1C): Set ATX Switch Functionality. The following functions may be individually enabled:

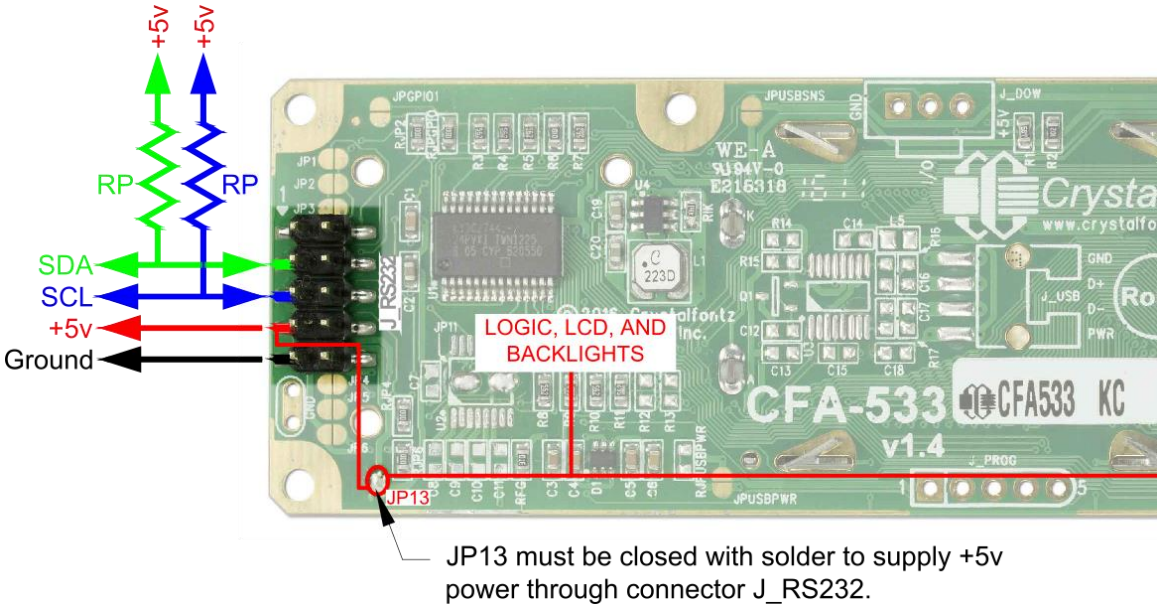
- System Power On. This function enables the use of the green check (enter) key to turn the host on when the HOST POWER SENSE is low. By default, pressing the enter key for 0.25 seconds will drive the POWER CONTROL line high for one pulse width as set in command 28 (1.0 seconds by default).
- System Hard Power Off. This function enables the use of the red X (cancel) key to turn the host off by driving the POWER CONTROL line down. This is initiated only when HOST POWER SENSE is high and the cancel key is pressed and held for 4 seconds. The line will be driven for a minimum of the pulse width set in command 28. Holding the cancel key longer than 4 seconds will cause the module to continue to drive the line for a maximum of an additional 5 seconds.
- System Hard Reset. This function enables forcing the host system to reset itself. If the green check (enter) key is pressed for 4 seconds when HOST POWER SENSE is high, the module will drive the RESET line for one pulse width. The module will reset the host, then reboot itself.

The module can also be configured to blank the display any time to HOST POWER SENSE line is low in order to appear off when the host is off.

7. Connections

7.1. I2C Connections

CFA533-KC modules communicate using I2C. The data (SDA) and clock (SCL) lines require external pull-up resistors (RP). The size of RP is determined by a combination of the supply voltage, clock speed, and bus capacitance. The minimum sink current for any I2C device should be no less than 3 mA at V_{OLMAX}= 0.4v for the output stage. This limits the minimum RP value for a 5-volt system to about 1.5kΩ. The maximum value for RP depends on bus capacitance and clock speed. For a 5-volt system with a bus capacitance of 150 pF, RP should be no larger than 6kΩ. For more information see the UM10204 I2C-bus specification and user manual on NXP Semiconductors' website.

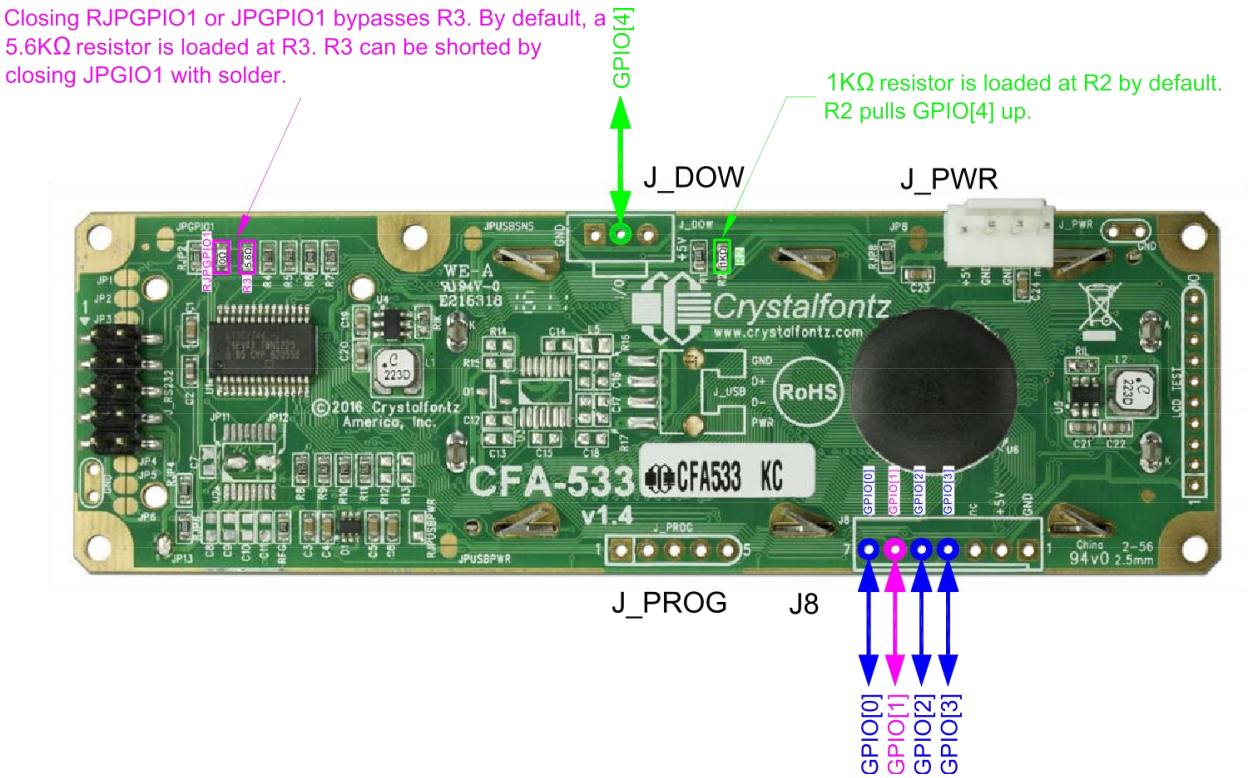


For more about communicating via I2C with the CFA533-KC, see section 8: I2C Communication.

Note: *Each command and related data bytes must be transmitted as a single packet to be processed correctly.*

7.2. GPIO Connections

The CFA533-KC modules have five general purpose input/output (GPIO) pins. These pins can be used to drive LEDs, relays, read switches or buttons and so on. Most of the CFA533 GPIOs have a default function to provide additional functionality. If ATX Host Power Sense or the 1-Wire bus are being used, do not reconfigure the related GPIO pins as user GPIO.




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

Refer to commands 34: Set/Configure GPIO and 35: Read GPIO Pin Levels and Configuration State for additional details.

7.3. Temperature Sensor 1-Wire Device (DOW) Connections

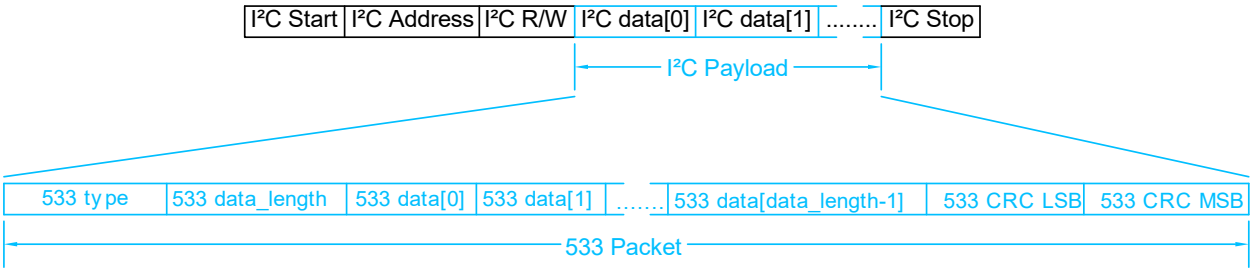
The CFA533-KC modules support Maxim 1-Wire (DOW) temperature sensors, which use the standard Dallas Semiconductor 1-Wire protocol for data transfers. These temperature sensors (WR-DOW-Y17) are an optional add-on to the CFA533.

The WR-DOW-Y17 has a DS18B20 sensor and a “daisy chain” cable. The mating connector for the WR-DOW-Y17 is the Molex 0705430002.

The CFA533-KC can be configured to automatically read and display the temperature in °C or °F using command 21: Set Up Live Temperature Display.

8. I2C Communication with Host

The CFA533-KC modules communicate with the host using a packet based I2C protocol. The I2C bus mater, the host, initiates all transactions. The host identifies the CFA533-KC based on its address, then writes the command and data to the input buffer of the CFA533-KC. The CFA533-KC executes the commands in its input buffer and acknowledges or write a response packet to its output buffer. The host then reads the output buffer of the CFA533-KC to verify the command or read the result of a query.



The I2C specification allows the I2C master (host) to run at clock speeds up to 100kHz.

For more information about I2C see the UM10204 I2C-bus specification and user manual on the NXP Semiconductors website.

Crystalfontz supplies 533 I2C WinTest, a demonstration and test program along with C source code. Included in the source code is a CRC algorithm and a packet validating algorithm. Use these algorithms to realize the full benefits of using packet-based communication.

8.1. I2C Address

The I2C address allows the host to identify an individual device (or group of devices). To communicate with a device on the I2C bus, the host begins a read or write transaction using a byte that contains the I2C address it would like to communicate with and a bit to indicate whether the operation is a read or write operation. The first 7 bits indicate the address of the device and the last bit indicates read or write.

Valid addresses are between 0 and 127_{10}. The least significant bit (LSB) of the byte contains the R/W bit. If this bit is 0, the address will be written to. If the LSB is 1, the address will have data read from it.

By default, the display module uses an I2C address of 0101010_2 (42_{10}, $0x2A_{16}$). To write to the display, the host device transmits the address left bit-shifted with a 0, which is 01010100_2 (84_{10}, $0x54_{16}$). To read from the display the address is instead left-shift with a 1 to get 01010101_2 (85_{10}, $0x55_{16}$).

The I2C address of the CFA533-KC can be set using command 33 (0x21): Set I2C Address. To make the change permanent, save it using command 4 (0x04): Store Current State as Boot State.

To display the current I2C address of the module on the LCD, press and hold both the up and down arrow keys for four seconds.

8.2. Packet Structure

Communication between the CFA533-KC and the host takes place in simple and robust CRC checked packets. The packet format enables reliable communication between the module and the host, avoiding problems that occur in stream-based serial communication.

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

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

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

8.2.1. <type>

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

```
TTcc cccc
|||| ||||--Command, response, error or report code 0-63₁₀
||---------Type:
00 = normal command from host to CFA533
01 = normal response from CFA533 to host
10 = normal report from CFA533 to host (not in direct response to a command
from the host)
11 = error response from CFA533 to host (a packet with valid structure but
illegal content was received by the CFA533)
```

8.2.2. <data_length>

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

8.2.3. <data>

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

8.2.4. <crc>

<crc> is a standard 16-bit CRC (cyclic redundancy check) which verifies all the information in the packet, excluding the **<crc>** itself. The **<crc>** immediately follows the last used element of **<data>**, and is sent LSB first.

See Appendix A for several examples of how to calculate the CRC in a variety of programming languages.

8.3. I2C Buffers

Reading and writing data to the CFA533-KC is accomplished using buffers. The buffers are called the input or write buffer for data written to the CFA533-KC by the host, and the output or read buffer for data to be read by the host.

During Read: The host, acting as the I2C master, reads data from the CFA533-KC output buffer. The host must read at least as many bytes as are included in the CFA533-KC's response packet. If the host attempts to read more data than the buffer contains, the last byte will be retransmitted until the host stops reading. The host must NAK the last byte that it reads so the CFA533-KC knows the read is terminated. The NAK should come just before the STOP.

During Write: The host writes data to the CFA533-KC input buffer. When the CFA533-KC input buffer memory is full, the CFA533-KC generates a NAK (negative acknowledgement). If the host continues to write data overflowing the buffer, the CFA533-KC will continue to NAK it. Any data written once the input buffer is full is not stored.

The CFA533-KC requires some time to execute a command after it has been written to its input buffer. For most commands 5mS is enough time for the CFA533-KC to have executed the command and have the result in the output buffer for the host to read.

Commands that take more time include:

- 2 (0x02): Write User Flash Area – 25mS
- 4 (0x04): Store Current State as Boot State – 50mS
- 5 (0x05): Reboot CFA533, Reset Host, or Power Off Host – up to 9S depending on function
- 14 (0x0E): Set LCD and Keypad Backlight – 50ms
- 20 (0x14): Arbitrary DOW Transaction – 50mS, though varies depending on transaction

Reconciling packets is recommended over using delays when communicating with the CFA533-KC. To reconcile packets, ensure the ACK for the most recent packet has been received before sending additional packets to the display module. This will prevent dropping packets or missing communication with the display module.

8.4. Command Codes

Below is a list of valid commands for the CFA533-KC. The CFA533-KC responds to each packet either by a response or error packet. As described in 8.2 Packet Structure, the first two bits of the response packet indicate the type (response or error) and the last six bits correspond to the type of command packet being acknowledged.

9. CFA533-KC Command Codes

0 (0x00): Ping Command

The host sends a packet include ng:

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

The CFA533-KC return packet is identical to the packet sent by the host, except the first two bits of the type now indicate the packet is a normal response to the host:

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

1 (0x01): Get Hardware and Firmware Version

The host sends a packet including:

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

The CFA533-KC return packet:

```
type: 0x40 | 0x01 = 0x41 = 0100 0001₂ = 65₁₀
data_length: 16
data[]: "CFA533:hX.X,yY.Y"

hX.X is the hardware revision e.g., "1.4"
yY.Y is the firmware version e.g., "c1.2"
```

(0x02): Write User Flash Area

The CFA533 reserves 16 bytes of nonvolatile memory as user accessible flash. This memory can be used to store data such as a serial number, IP address, gateway address, netmask, or any other data. All 16 bytes must be supplied. A delay of 25mS after the I2C write phase completes is required to guarantee the CFA533-KC will have the acknowledge or response packet ready to be read by the host.

```
type: 0x02 = 2₁₀
data_length: 16
data[]: 16 bytes of arbitrary user data to be stored in the CFA533's non-
volatile memory
```

The return packet will be:

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

3 (0x03): Read User Flash Area

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

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

The return packet will be:

```
type: 0x40 | 0x03 = 0x43 = 67₁₀
data_length: 16
data[]: 16 bytes user data recalled from the CFA533's non-volatile memory
```

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

The CFA533 loads its power-up configuration from nonvolatile memory when power is applied. The CFA533 is configured at the factory to display a Crystalfontz boot screen when power is applied. This command can be used to customize the boot screen, along with the following settings:

- The contents of the DDRAM (characters shown on LCD), affected by:
  - Command 6 (0x06): Clear LCD Screen
  - Command 31 (0x1F): Send Data to LCD
  - Depreciated commands 7 and 8
- Command 9 (0x09): Set LCD Special Character Data
- Command 11 (0x0B): Set LCD Cursor Position
- Command 12 (0x0C): Set LCD Cursor Style
- Command 13 (0x0D): Set LCD Contrast
- Command 14 (0x0E): Set LCD & Keypad Backlight
- Command 21 (0x15): Set Up Live Temperature Display*
- Command 28 (0x1C): Set ATX Switch Functionality**
- Command 33 (0x21): Set I2C Address
- Command 34 (0x22): Set/Configure GPIO

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

```
type: 0x04 = 4(10)
data_length: 0
```

This command may take longer to resolve. Wait 50mS after the I2C write phase completes to guarantee the CFA533-KC will have the acknowledge or response packet ready to be read by the host. The return packet will be:

```
type: 0x40 | 0x04 = 0x44 = 68(10)
data_length: 0
```

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

5 (0x05): Reboot CFA533, Reset Host, or Power Off Host

This command instructs the CFA533 to simulate a power-on restart of itself, reset the host, or turn the host's power off.

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

```
type: 0x05 = 5(10)
data_length: 3
data[0]: 8
data[1]: 18
data[2]: 99
```

To reset the host, when the host's reset line is connected to GPIO[3] in one of the ATX configurations, send the following packet:

```
type: 0x05 = 5(10)
data_length: 3
data[0]: 12
data[1]: 28
data[2]: 97
```

To turn the host's power off, assuming the host's power control line is connected to GPIO[2] in an ATX configuration send the following packet:

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

Wait after the write phase completes to guarantee the CFA533 will have the acknowledge or response packet ready to be read by the host. Execute times are up to:

500mS for parameters \008\018\099, Reboot CFA533

The CFA533 resets itself, then prepares the acknowledge packet. The host may read the acknowledge packet from the display module any time starting at 500mS after the command was sent.

2mS ~ 1500mS for parameters \012\028\097, Reset host

The CFA533 prepares the acknowledge packet immediately then waits 100mS for the host to read the acknowledge packet. After that, the CFA533 will be unavailable for ~1500mS (1000mS is the length of the host reset pulse, plus ~500mS for the CFA533 to reset itself). Typically, this variable delay will not be a concern, as the host system will be rebooting. When the CFA533 resets, it will display its boot screen, which can be configured using command 4.

2mS ~ 9S for parameters \003\011\095, Power off host

The CFA533 prepares the acknowledge packet immediately then waits 100mS for the host to read the acknowledge packet. After that, the CFA533 will be unavailable for a variable amount of time, depending on how long after the CFA533 asserts the power signal until the host power falls. The maximum time is ~9S (1000mS is the length of the host reset pulse, up to 7.5S for the host power to fall, plus allow ~500mS for the CFA533 to reset itself). Typically, this variable delay will not be a concern, as the host system will be powering down. The CFA533 can be set to blank its screen when the host power falls so the system appears powered off.

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

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

6 (0x06): Clear LCD Screen

This command empties the contents of the LCD's DDRAM (removes the characters from the display) and moves the cursor to the left-most column of the top line. The contents of the DDRAM are saved by Command 4: Store Current State as Boot State.

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

The return packet will be:

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

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

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

9 (0x09): Set LCD Special Character Data

This command sets the bitmap for the eight special characters in the CGROM.

```
type: 0x09 = 9₁₀
data_length: 9
data[0]: index of special character to modify, 0-7 are valid
data[1-8]: bitmap of the character
```

data[1-8] are the bitmap information for the character. Any value between 0 and 31 is valid. The MSB is at the left of the character cell of the row, and the LSB is at the right of the character cell. **data[1]** is at the top of the cell, **data[8]** is at the bottom of the cell.

The return packet will be:

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

The special characters can be saved using Command 4 (0x04): Store Current State as Boot State.

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

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

```
type: 0x0A = 10₁₀
data_length: 1
data[0]: address code of desired data
```

data[0] is the address code native to the LCD controller:

```
0x40 (\064) to 0x7F (\127) for CGROM
0x80 (\128) to 0x8F (\143) for DDRAM, line 1
0xC0 (\192) to 0xCF (\207) for DDRAM, line 2
```

The return packet will be:

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

11 (0x0B): Set LCD Cursor Position

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

```
type: 0x0B = 11₁₀
data_length: 2
data[0]: column (0-15 valid)
data[1]: row (0-1 valid)
```

The return packet will be:

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

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

12 (0x0C): Set LCD Cursor Style

This command selects among four hardware generated cursor options.

```
type: 0x0C = 12₁₀
data_length: 1
data[0]: cursor style (0-3 valid)
       0 = no cursor
       1 = blinking block cursor
       2 = underscore cursor
       3 = blinking underscore (Note: This behavior differs from the CFA633
       series which is: blinking block plus underscore.)
```

The return packet will be:

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

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

13 (0x0D): Set LCD Contrast

This command sets the contrast of the display.

CFA533 Enhanced

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

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

CFA633 Compatible

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

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

The return packet for either method is:

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

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

14 (0x0E): Set LCD & Keypad Backlight

This command sets the brightness of the LCD and keypad backlights. Wait 50mS after the write phase completes to guarantee the acknowledge or response packet is ready to be read by the host.

CFA533 Enhanced

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

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

CFA633 Compatible

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

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

The return packet for either method is:

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

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

15 (0x0F): Read Temperature

The command retrieves the most recent temperature sensor reading. Each temperature sensor is read once every second.

```
type: 0x0F = 1510
data_length: 1
data[0]: 0 to 31 DOW device index
```

The family code for the device at "device index" must be 0x22 (DS1822) or 0x28 (DS12B20). This can be verified with Command 18 (0x12): Read DOW Device Information.

The return packet will be:

```
type: 0x40 | 0x0F = 0x4F = 7910
data_length: 4
data[0]: index of the temperature sensor being reported:
       0 = temperature sensor 1
       1 = temperature sensor 2
       . . .
       31 = temperature sensor 32
data[1]: LSB of Temperature_Sensor_Counts
data[2]: MSB of Temperature_Sensor_Counts
data[3]: DOW_crc_status
```

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

```
void OnReceivedTempReport(COMMAND_PACKET *packet, char *output)
  {
  //First check the DOW CRC return code from the CFA533
  if(packet->data[3]==0)
    strcpy(output,"BAD CRC");
  else
    {
    double
      degc;
    degc=(*(short *)&(packet->data[1]))/16.0;

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

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

16 and 17: Reserved

18 (0x12): Read DOW Device Information

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

```
type: 0x12 = 18₁₀
data_length: 1
data[0] = device index (0-31 valid)
```

The return packet will be:

```
type: 0x40 | 0x12 = 0x52 = 82₁₀
data_length: 9
data[0]: device index (0-31 valid)
data[1-8]: ROM ID of the device
```

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

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

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

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

19 Reserved

20 (0x14): Arbitrary DOW Transaction

This command enables arbitrary transactions on the 1-Wire bus to be specified. The CFA533 can function as an I2C to 1-Wire bridge. The CFA533 can send up to 15 bytes and receive up to 14 bytes. Some devices require larger transactions and cannot be fully used with the CFA533. 1-Wire commands follow this basic layout:

```
<bus reset>     //Required
<address_phase> //Must be "Match ROM" or "Skip ROM"
<write_phase>   //At least one of write_phase or read_phase must be sent
<read_phase>    //At least one of write_phase or read_phase must be sent
```

See APPENDIX B: CONNECTING A DS2450 1-WIRE QUAD A/D CONVERTER for an example of using this command.

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

If **device_index** is 32, no address phase will be executed.
If **device_index** is in the range of 0 to 31 and a 1-Wire device was detected at that **device_index** at power-up, then the write cycle will be prefixed with a "Match ROM" command and the address information for that device.
If **data_length** is two, then no specific write phase will be executed (although address information may be written independently of **data_length** depending on the value of **device_index**).
If **data_length** is greater than two, then **data_length-2** bytes of **data_to_be_written** will be written to the 1-Wire bus immediately after the address phase.
If **number_of_bytes_to_read** is zero, then no read phase will be executed.
If **number_of_bytes_to_read** is not zero then **number_of_bytes_to_read** will be read from the bus and loaded into the response packet.

Wait 50mS after the write phase completes to guarantee the CFA533 has the acknowledge or response packet ready for the host.

The return packet will be:

```
type: 0x40 | 0x14 = 0x54 = 84(10)
data_length: 2 to 16
data[0]: device index (0-31 valid)
data[1-[data_length-2]]: Data read from the 1-Wire bus. The number of bytes
          is specified as number_of_bytes_to_read in the command.
data[data_length-1]: 1-Wire CRC
```

21 (0x15): Set Up Live Temperature Display

This command configures live temperature display. The CFA533 can automatically update portions of the LCD with a live temperature reading. Once configured, the CFA533 will continue to display the live reading without intervention. The settings from this command are stored by Command 4: Store Current State as Boot State. This allows the CFA533 to display system temperatures as soon as power is applied.

The live display uses display “slots”. There are 4 slots, and each of the 4 slots may be enabled or disabled independently. Any slot can display any data that is available. For instance, slot 0 could display temperature sensor 3 in °C, while slot 1 could simultaneously display temperature sensor 3 in °F. Any slot may be positioned at any location on the LCD, as long as all the digits of that slot fall fully within the display area. The display area of one slot can overlap the display area of another slot, but should be avoided in order to have meaningful information displayed.

```
type: 0x15 = 21(10)
data_length: 7 or 2 (for turning a slot off)
data[0]: display slot (0-3)
data[1]: type of item to display in this slot
       0 = nothing (data_length then must be 2)
       1 = (invalid)
       2 = temperature (data_length then must be 7)
data[2]: index of the sensor to display in this slot:
       0-31 are valid for temperatures (and the temperature device must be
       attached)
data[3]: number of digits
       for a temperature: 3 digits (-XX or XXX)
       for a temperature: 5 digits (-XX.X or XXX.X)
data[4]: display column
       0-13 valid for a 3-digit temperature
       0-11 valid for a 5-digit temperature
data[5]: display row (0-1 valid)
data[6]: temperature units(0 = deg C, 1 = deg F)
```

If a 1-Wire CRC error is detected, the temperature will be displayed as "ERR" or "ERROR".

The return packet will be:

```
type: 0x40 | 15 = 0x55 = 85(10)
data_length: 0
```

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

This command allows direct access to the LCD's controller. It is possible to corrupt the CFA533 display using this command. The controller on the CFA533 is the Neotec NT7070B (HD44780 compatible). Generally low-level access to the LCD controller is unnecessary. This command ensures full functionality of the display is accessible.

```
type: 0x16 = 22(10)
data_length: 2
data[0]: location code
       0 = "Data" register
       1 = "Control" register
data[1]: data to write to the selected register
```

The return packet will be:

```
type: 0x40 | 0x16 = 0x56 = 86(10)
data_length: 0
```

23 (0X17) Enable Key Ready Flag

This command allows GPIO[0] (J8's Pin 7) to act as a key ready flag. If enabled, it sets GPIO[0] high until read by the host using 24 (0x18): Read Keypad, Polled Mode.

```
type: 0x17 = 23(10)
data_length: 1
data[0]: Enable or disable Key Ready Flag
       0 = GPIO[0] is normal
       1 = Key Ready Flag enabled on GPIO[0]
       GPIO[0] is driven high if a key is ready and driven low if no keys
       are ready
```

The return packet will be

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

24 (0x18): Read Keypad, Polled Mode

This command allows the host to detect key presses including which keys are currently pressed, which keys have been pressed since the last poll, and which keys have been released since the last poll.

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

type: 0x18 = 24₁₀
data_length: 0
```

The return packet will be:

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

25-27 Reserved

28 (0x1C): Set ATX Switch Functionality

This command sets which of the four possible ATX functions are active. Activating ATX functionality, when used with an ATX-compatible system, allows the CFA533 to replace the power and reset switches in the system. These settings are stored by Command 4: Store Current State as Boot State.

By default there is an internal HOST POWER SENSE connected to the +5v pin of J_PWR, selected by setting data[2] to 1. Alternatively, GPIO[1] may be configured to act as HOST POWER SENSE. More information about ATX connections can be found in Section 6: Power Supply Connections.

FOUR FUNCTIONS CONTROLLED BY COMMAND 28

Function 1: KEYPAD_RESET

If HOST POWER SENSE is high, holding the green check key for 4 seconds will pulse RESET (GPIO[3]) pin for 1 second. During the 1-second pulse, the CFA533 will show "RESET". Then the CFA533 will reset itself, showing its boot state as if it had just powered on. After the pulse, the CFA533 will not respond to any commands until it has reset the host and itself.

Function 2: KEYPAD_POWER_ON

If HOST POWER SENSE is low, pressing the green check key for 0.25 seconds will pulse POWER CONTROL (GPIO[2]) for the duration specified by in data[1] or the default of 1 second. During this time the CFA533 will show "POWER ON", then the CFA533 will reset itself.

Function 3: KEYPAD_POWER_OFF

If HOST POWER SENSE is high, holding the red X key for 4 seconds will pulse POWER CONTROL (GPIO[2]) for the duration specified by in data[1] or the default of 1 second. If the user continues to hold the power key down, then the CFA533 will continue to drive the line for a maximum of 5 additional seconds. During this time the CFA533 will show "POWER OFF".

Function 4: LCD_OFF_IF_HOST_IS_OFF

If LCD_OFF_IF_HOST_IS_OFF is set, the CFA533 will blank its screen and turn off its backlight to simulate its power being off any time HOST POWER SENSE is low.

The RESET (GPIO[3]) and POWER CONTROL (GPIO[2]) lines on the CFA533 are normally high-impedance. Electrically, they appear to be disconnected or floating. When the CFA533 asserts the RESET or POWER CONTROL lines, they are momentarily driven high or low (as determined by the

AUTO_POLARITY, RESET_INVERT or POWER_INVERT bits). To end the power or reset pulse, the CFA533 changes the lines back to high impedance.

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

type: 0x1C = 2810
data_length: 1, 2 or 3
data[0]: bitmask of enabled functions
data[1]: (optional) length of power on & off pulses in 1/32 second
      1 = 1/32 sec
      2 = 1/16 sec
      16 = 1/2 sec
      255 = 8 sec
data[2]: (optional) atx_sense_on_floppy
      0: sense ATX host state on P2.1 (J8, pin 6 / GPIO [1] -- R3 must be
      loaded)
      1: sense ATX host state on P0.7 (JPWR,+5v -- recommended
      configuration))
```

The return packet will be:

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

GPIO Configuration for ATX

To use the ATX functionality, the relevant GPIO pins must be configured appropriately for ATX. This is the default configuration, but can be changed using Command 34: Set/Configure GPIO.

If using GPIO[1] as HOST POWER SENSE, GPIO[1] must be configured as:

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

This configuration can be assured by sending the following command:

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

For GPIO[2] to operate correctly as ATX POWER CONTROL, GPIO[2] must be configured as:

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

This configuration can be assured by sending the following command:

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

For GPIO[3] to operate correctly as ATX RESET, user GPIO[3] must be configured as:

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

This configuration can be assured by sending the following command:

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

These settings must be saved as the boot state using Command 4: Store Current State as Boot State.

29 (0x1D): Enable/Feed Host Watchdog Reset

This command sets a timeout length in seconds for systems which 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 enables the watchdog timer on the CFA533. If the system monitor program fails to feed the CFA533's watchdog timer, the CFA533 will reset the host system. To use this command ATX functionality must be enabled.

```
type: 0x1D = 29₁₀
data_length: 1
data[0]: enable/timeout
If timeout is 0, the watchdog is disabled.
If timeout is 1-255, then this command must be issued again within timeout
seconds to feed the watchdog and avoid a watchdog reset.
```

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

The return packet will be:

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

30 (0x1E): Read Reporting/ATX/Watchdog (debug)

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

```
type: 30
data_length: 0
```

The return packet will be:

```
type: 0x1E = 30₁₀
data_length: 15
data[0]-data[6]:  0
data[7]:  ATX Power Switch Functionality (as set by command 28)
data[8]:  Current watchdog counter (as set by command 29)
data[9]:  User Contrast Adjust (as set by command 13, data[1])
data[10]: Key backlight setting (as set by command 14, data[1])
data[11]: atx_sense_on_floppy (as set by command 28)
data[12]: 0
data[13]: CFA633-style contrast setting (as set by command 13, data[0])
data[14]: LCD backlight setting (as set by command 14, data[0])
```

Note: Previous and future firmware versions may return fewer or additional bytes.

31 (0x1F): Send Data to LCD

This command places data at a given position on the LCD. The command changes the contents of the DDRAM which is saved by Command 4: Store Current State as Boot State.

```
type: 0x1F = 31₁₀
data_length: 3 to 18
data[0]: col = x = 0 to 15
data[1]: row = y = 0 to 1
data[2-21]: text to place on the LCD, variable from 1 to 16 characters
```

The return packet will be:

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

32 Reserved

33 (0x21): Set I2C Address

This command sets the I2C address. An I2C address is a 7-bit number (0-127$_{10}$) that allows the host to read or write from a device specified by the address. The address and R/W bit are combined into a byte. The R/W bit is the LSB, thus the address is bit shifted left by one and combined with the R/W bit to make the actual I2C address byte. The default address of the CFA533-KC is 42$_{10}$ (84$_{10}$ writes, 85$_{10}$ reads), in hexadecimal 0x2A (0x54 writes, 0x55 reads). In binary, this looks like: 0101 010[R/W]

```
type: 0x21 = 33₁₀
data_length: 1
data[0]: 0 to 127
```

The return packet will be:

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

Debugging Tip: To display the I2C address of the display module on the LCD, hold both the up and the down arrows for 4 seconds.

34 (0x22): Set/Configure GPIO

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

The architecture of the CFA533 allows flexibility in the configuring the GPIOs. They can be set as input or output. They can output constant high or low signals or a variable duty cycle 100 Hz PWM signal. In output mode using PWM (and a current limiting resistor), an LED may be turned on, off, or dimmed under host software control. With external circuitry, the GPIOs can drive external logic or power transistors.

The CFA533 continuously polls the GPIOs as inputs at 32 Hz. The present level can be queried by the host software at a lower rate. The CFA533 keeps track rising and falling edges between host queries (subject to the resolution of the 32 Hz sampling) so the host is not forced to poll quickly in order to detect short events.

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

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

Pull-up/pull-down resistance values are approximately 5kΩ. Do not exceed a current of 25 mA per GPIO. GPIO[1] may be connected to the host's power in order to sense the host's power on/off state. R3, a 5.6kΩ resistor is in series with GPIO[1] to limit the possibility of latchup.

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

```
type: 0x22 = 34₁₀
data_length: 2 bytes to change value only
             3 bytes to change value and configure function and drive mode
data[0]: index of GPIO to modify
      0 = GPIO[0] = J8, Pin 7
      1 = GPIO[1] = J8, Pin 6 (may be ATX Host Power Sense, as configured
      by command 28, data[2])
      2 = GPIO[2] = J8, Pin 5 (default is ATX Host Power Control)
      3 = GPIO[3] = J8, Pin 4 (default is ATX Host Reset Control)
      4 = GPIO[4] = J_DOW, Pin 2 (default is DOW I/O -- has 1kΩ resistor
      hardware pull-up: R2)
      5-255 = reserved
data[1]: Pin output state (actual behavior depends on drive mode):
      0 = Output set to low
      1-99 = Output duty cycle percentage (100 Hz nominal)
      100 = Output set to high
      101-255 = invalid
data[2]: Pin function select and drive mode (optional)
      ---- FDDD
      |||| ||||-- DDD = Drive Mode (based on output state of 1 or 0)
      |||| | ========================================================
      |||| | 000: 1=Fast, Strong Drive Up, 0=Resistive Pull Down
      |||| | 001: 1=Fast, Strong Drive Up, 0=Fast, Strong Drive Down
      |||| | 010: Hi-Z, use for input
      |||| | 011: 1=Resistive Pull Up, 0=Fast, Strong Drive Down
      |||| | 100: 1=Slow, Strong Drive Up, 0=Hi-Z
      |||| | 101: 1=Slow, Strong Drive Up, 0=Slow, Strong Drive Down
      |||| | 110: reserved, do not use
      |||| | 111: 1=Hi-Z, 0=Slow, Strong Drive Down
      |||| |
      |||| |----- F = Function
      ||||         ============================================================
      ||||         0: Port unused for GPIO. It will take on the default
      ||||         function such as ATX, DOW or unused. The user is
      ||||         responsible for setting the drive to the correct
      ||||         value in order for the default function to work
      ||||         correctly.
      ||||         1: Port used for GPIO under user control. The user is
      ||||         responsible for setting the drive to the correct
      ||||         value in order for the desired GPIO mode to work
      ||||         correctly.
      ||||------- reserved, must be 0
```

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

The return packet will be:

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

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

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

```
type: 0x23 = 35₁₀
data_length: 1
data[0]: index of GPIO to query
      0 = GPIO[0] = J8, Pin 7
      1 = GPIO[1] = J8, Pin 6 (may be ATX Host Power Sense, as configured
            by command 28, data[2])
      2 = GPIO[2] = J8, Pin 5 (default is ATX Host Power Control)
      3 = GPIO[3] = J8, Pin 4 (default is ATX Host Reset Control)
      4 = GPIO[4] = J_DOW, Pin 2 (default is DOW I/O)
      5-255 = reserved
```

The return packet will be:

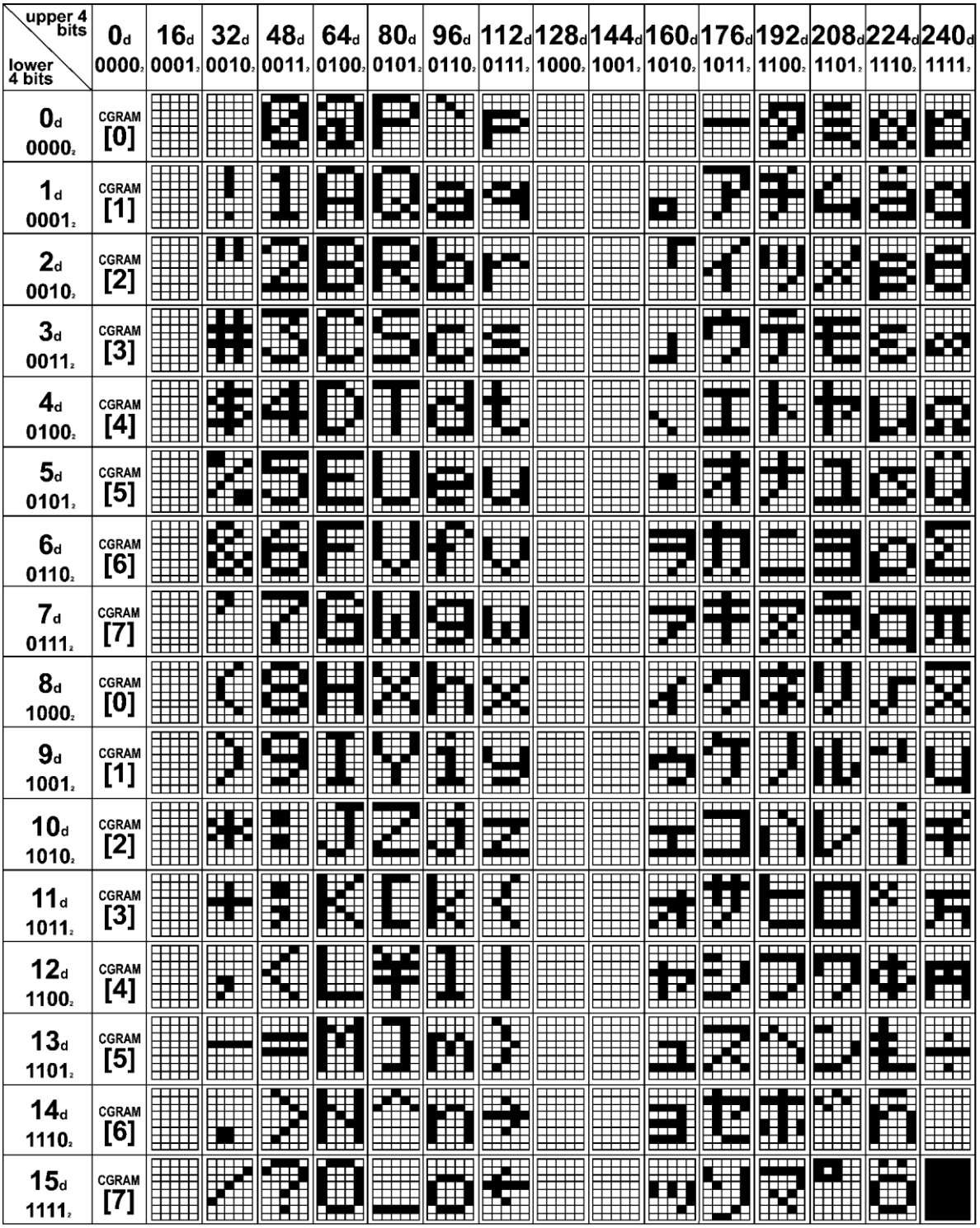
```
type: 0x23 = 35₁₀
data_length: 4
data[0]: index of GPIO read
data[1]: Pin state & changes since last poll
---- -RFS
|||| ||||-- S = state at the last reading
|||| |||--- F = at least one falling edge detected since last poll
|||| ||---- R = at least one rising edge detected since last poll
|||| |----- reserved
This reading is the actual pin state which may not agree with the pin
setting, depending on drive mode and load presented by external circuitry.
The pins are polled at approximately 32 Hz asynchronously with respect to
this command. Transients that happen between polls will not be detected.

data[2]: Requested Pin level/PWM level
      0-100 = Output duty cycle percentage
This value is the requested PWM duty cycle. The actual pin may not toggle
in agreement with this value, depending on the drive mode and the load
presented by external circuitry.

data[3]: Pin function select and drive mode
      ---- FDDD
      |||| ||||-- DDD = Drive Mode
      |||| | ========================================================
      |||| | 000: 1=Fast, Strong Drive Up, 0=Resistive Pull Down
      |||| | 001: 1=Fast, Strong Drive Up, 0=Fast, Strong Drive Down
      |||| | 010: Hi-Z, use for input
      |||| | 011: 1=Resistive Pull Up, 0=Fast, Strong Drive Down
      |||| | 100: 1=Slow, Strong Drive Up, 0=Hi-Z
      |||| | 101: 1=Slow, Strong Drive Up, 0=Slow, Strong Drive Down
      |||| | 110: reserved
      |||| | 111: 1=Hi-Z, 0=Slow, Strong Drive Down
      |||| |
      |||| |----- F = Function
      ||||         ========================================================
      ||||         0: Port unused for GPIO. It will take on the default
      ||||         function (ATX, DOW or unused). The user must set the
      ||||         drive to the appropriate value 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
      ||||         correctly.
      ||||------- reserved, will return 0
```

10. Character Generator ROM (CGROM)

The CFA533-KC includes a built in CGROM. To find the code for any given character, add the two base ten numbers of the row and column for the character. For example, to send an “A” from column 64$_d$ and row 1$_d$ add 64 and 1 to get 65. When a byte with a value of 65 is sent to the display an “A” will be shown. In binary, the columns represent the upper 4 bits of the byte, and the row the lower 4. So, the value for the letter is found by appending the two sets of bits. For the “A” the value sent is 0100 0001.



11. Module Reliability and Lifetime

11.1. Display Module Reliability

The following specifications are given for modules operated and stored according to the specifications in this datasheet, and in humidity non-condensing RH under 65% and with no direct exposure to sunlight. The values listed are approximate and represent typical values.

Part Number	Item	Specification	
CFA533-xxx-KC (all variants)	LCD Display Portion	50,000 – 100,000 hours	
	Keypad	1,000,000 keystrokes	
CFA533-TFH-KC CFA533-TMI-KC	White LED backlights and white or blue keypad backlights*	*% of initial Brightness*	*Power-On hours*
		>90%	10,000
		>50%	50,000
CFA533-YYH-KC	Yellow-green LED display and keypad backlights	50,000-100,000 hours	

11.2. Display Longevity and EOL/Replacement Policy

Crystalfontz is committed to making all of our display modules available for as long as possible. Each display module we introduce, we intend to offer indefinitely. We do not pre-plan a display module's obsolescence. The majority of modules we have introduced are still available.

We recognize that discontinuing a display 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 display module. For example, we must occasionally discontinue a display module when a supplier discontinues a component or a manufacturing process becomes obsolete. When we discontinue a display module, we will do our best to find an acceptable replacement display 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 display module to the discontinued display module it replaces. However, sometimes a change in component or process for the replacement display module results in a slight variation, perhaps an improvement, over the previous design. Although the replacement display module is still within the stated Datasheet specifications and tolerances of the discontinued display 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.* Display module components have manufacturing tolerances. In extreme cases, the tolerance stack can change the visual or operating characteristics.

We avoid changing a display module whenever possible; we only discontinue a display module if we have no other option. We will post Part Change Notices (PCN) on the product's web page as soon as possible. To be notified, subscribe to future part change notifications.

12. Appendix A: Demonstrations Software and Sample Code

We encourage you to use the free sample code listed below. Please leave the original copyrights in the code. Many examples are located in the Datasheets and Files section of the module web page.

- Connect CFA533-KC to an Arduino Uno using this example
  https://www.crystalfontz.com/products/document/3719/CFA533-I2C-Arduino-Example.zip
- Windows compatible test/demonstration program and source.
  https://www.crystalfontz.com/product/533i2cwintest with TotalPhase AardvarkI2C/SPI adapter
- 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
- http://lcdproc.org/index.php3 for Linux LCD drivers. LCDproc is an open-source project that supports many of the Crystalfontz displays.

12.1. Algorithms to Calculate the CRC

Below are eight sample algorithms that will calculate the CRC of a CFA533 packet. Some algorithms were contributed by forum members and originally written for CFA631 and CFA635. The CRC used in the CFA533 is the same one that is used in IrDA, which came from PPP, which seems to be related to a CCITT (ref: Network Working Group Request for Comments: 1171) standard. At that point, the trail was getting a bit cold and diverged into several referenced articles and papers, dating back to 1983.

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

The result is bit-wise inverted before being returned.

12.1.1. Algorithm 1: "C" Table Implementation

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

```
//This code is from the IRDA LAP documentation, which appears to
//have been copied from PPP:
//
//http://irda.affiniscape.com/associations/2494/files/Specifications/
//IrLAP11_Plus_Errata.zip
//
//I doubt that there are any worries about the legality of this code,
//searching for the first line of the table below, it appears that
//the code is already included in the linux 2.6 kernel "Driver for
//ST5481 USB ISDN modem". This is an "industry standard" algorithm
//and I do not think there are ANY issues with it at all.
typedef unsigned char ubyte;
typedef unsigned short word;
word get_crc(ubyte *bufptr,word len)
  {
  //CRC lookup table to avoid bit-shifting loops.  static
  const word crcLookupTable[256] =
    {0x00000,0x01189,0x02312,0x0329B,0x04624,0x057AD,0x06536,0x074BF,
     0x08C48,0x09DC1,0x0AF5A,0x0BED3,0x0CA6C,0x0DBE5,0x0E97E,0x0F8F7,
     0x01081,0x00108,0x03393,0x0221A,0x056A5,0x0472C,0x075B7,0x0643E,
     0x09CC9,0x08D40,0x0BFDB,0x0AE52,0x0DAED,0x0CB64,0x0F9FF,0x0E876,
     0x02102,0x0308B,0x00210,0x01399,0x06726,0x076AF,0x04434,0x055BD,
     0x0AD4A,0x0BCC3,0x08E58,0x09FD1,0x0EB6E,0x0FAE7,0x0C87C,0x0D9F5,
     0x03183,0x0200A,0x01291,0x00318,0x077A7,0x0662E,0x054B5,0x0453C,
     0x0BDCB,0x0AC42,0x09ED9,0x08F50,0x0FBEF,0x0EA66,0x0D8FD,0x0C974,
     0x04204,0x0538D,0x06116,0x0709F,0x00420,0x015A9,0x02732,0x036BB,
     0x0CE4C,0x0DFC5,0x0ED5E,0x0FCD7,0x08868,0x099E1,0x0AB7A,0x0BAF3,
     0x05285,0x0430C,0x07197,0x0601E,0x014A1,0x00528,0x037B3,0x0263A,
     0x0DECD,0x0CF44,0x0FDDF,0x0EC56,0x098E9,0x08960,0x0BBFB,0x0AA72,
     0x06306,0x0728F,0x04014,0x0519D,0x02522,0x034AB,0x00630,0x017B9,
     0x0EF4E,0x0FEC7,0x0CC5C,0x0DDD5,0x0A96A,0x0B8E3,0x08A78,0x09BF1,
     0x07387,0x0620E,0x05095,0x0411C,0x035A3,0x0242A,0x016B1,0x00738,
     0x0FFCF,0x0EE46,0x0DCDD,0x0CD54,0x0B9EB,0x0A862,0x09AF9,0x08B70,
     0x08408,0x09581,0x0A71A,0x0B693,0x0C22C,0x0D3A5,0x0E13E,0x0F0B7,
     0x00840,0x019C9,0x02B52,0x03ADB,0x04E64,0x05FED,0x06D76,0x07CFF,
```

```
    0x09489,0x08500,0x0B79B,0x0A612,0x0D2AD,0x0C324,0x0F1BF,0x0E036,
    0x018C1,0x00948,0x03BD3,0x02A5A,0x05EE5,0x04F6C,0x07DF7,0x06C7E,
    0x0A50A,0x0B483,0x08618,0x09791,0x0E32E,0x0F2A7,0x0C03C,0x0D1B5,
    0x02942,0x038CB,0x00A50,0x01BD9,0x06F66,0x07EEF,0x04C74,0x05DFD,
    0x0B58B,0x0A402,0x09699,0x08710,0x0F3AF,0x0E226,0x0D0BD,0x0C134,
    0x039C3,0x0284A,0x01AD1,0x00B58,0x07FE7,0x06E6E,0x05CF5,0x04D7C,
    0x0C60C,0x0D785,0x0E51E,0x0F497,0x08028,0x091A1,0x0A33A,0x0B2B3,
    0x04A44,0x05BCD,0x06956,0x078DF,0x00C60,0x01DE9,0x02F72,0x03EFB,
    0x0D68D,0x0C704,0x0F59F,0x0E416,0x090A9,0x08120,0x0B3BB,0x0A232,
    0x05AC5,0x04B4C,0x079D7,0x0685E,0x01CE1,0x00D68,0x03FF3,0x02E7A,
    0x0E70E,0x0F687,0x0C41C,0x0D595,0x0A12A,0x0B0A3,0x08238,0x093B1,
    0x06B46,0x07ACF,0x04854,0x059DD,0x02D62,0x03CEB,0x00E70,0x01FF9,
    0x0F78F,0x0E606,0x0D49D,0x0C514,0x0B1AB,0x0A022,0x092B9,0x08330,
    0x07BC7,0x06A4E,0x058D5,0x0495C,0x03DE3,0x02C6A,0x01EF1,0x00F78};

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

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

12.1.2. 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);
  }
```

12.1.3. 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);
}
```

12.1.4. Algorithm 3: "PIC Assembly" Bit Shift Implementation

This routine was graciously donated by one of our customers.

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

accuml          equ             40h             ; BYTE - CRC result register high byte
accumh          equ             41h             ; BYTE - CRC result register high low byte
datareg                 equ             42h             ; BYTE - data register for shift
j               equ             43h             ; BYTE - bit counter for CRC 16 routine
Zero            equ             44h             ; BYTE - storage for string memory read
index           equ             45h             ;BYTE - index for string memory read
savchr          equ             46h             ;BYTE - temp storage for CRC routine
;
seedlo          equ             021h            ;initial seed for CRC reg lo byte
seedhi          equ             0F3h            ;initial seed for CRC reg hi byte
;
polyL           equ             008h            ;polynomial low byte
polyH           equ             084h            ;polynomial high byte
;====================================================================
;       CRC Test Program
;--------------------------------------------------------------------
                org             0               ; reset vector = 0000H
;
                clrf            PCLATH          ; ensure upper bits of PC are cleared
                clrf            STATUS          ; ensure page bits are cleared
                goto            main            ; jump to start of program
;
; ISR Vector
;
                org             4               ; start of ISR
                goto            $               ; jump to ISR when coded
;
                org             20              ; start of main program
main
                movlw           seedhi          ; setup intial CRC seed value.
                movwf           accumh          ; This must be done prior to
                movlw           seedlo          ; sending string to CRC routine.
                movwf           accuml          ;
                clrf            index           ; clear string read variables
;
main1
                movlw           HIGH InputStr ; point to LCD test string
                movwf           PCLATH          ; latch into PCL
                movfw           index           ; get index
                call            InputStr        ; get character
                movwf           Zero            ; setup for terminator test
                movf            Zero,f          ; see if terminator
                btfsc           STATUS,Z        ; skip if not terminator
                goto            main2           ; else terminator reached, jump out of loop
                call            CRC16           ; calculate new      crc
                call            SENDUART        ; send data to LCD
```

```
            incf        index,f     ; bump index
            goto        main1       ; loop
;
main2
            movlw       00h         ; shift accumulator 16 more bits.
            call        CRC16       ; This must be done after sending
            movlw       00h         ; string to CRC routine.
            call        CRC16       ;
;
            comf        accumh,f    ; invert result
            comf        accuml,f    ;
;
            movfw       accuml      ; get CRC low byte
            call        SENDUART    ; send to LCD
            movfw       accumh      ; get CRC hi byte
            call        SENDUART    ; send to LCD
;
stop        goto        stop        ; word result of 0x93FA is in accumh/accuml
;=====================================================================
; calculate CRC of input byte
;---------------------------------------------------------------------
CRC16
             movwf      savchr      ; save the input character
            movwf       datareg     ; load data register
            movlw  .    8           ; setup number of bits to test
            movwf       j           ; save to incrementor
_loop
            clrc                    ; clear carry for CRC register shift
            rrf         datareg,f   ; perform shift of data into CRC register
            rrf         accumh,f    ;
            rrf         accuml,f    ;
            btfss       STATUS,C    ; skip jump if if carry
            goto        _notset     ; otherwise goto next bit
            movlw       polyL       ; XOR poly mask with CRC register
            xorwf       accuml,F    ;
            movlw       polyH       ;
            xorwf       accumh,F    ;
_notset
            decfsz      j,F         ; decrement bit counter
            goto        _loop       ; loop if not complete
            movfw       savchr      ; restore the input character
            return                  ; return to calling routine
;=====================================================================
; USER SUPPLIED Serial port transmit routine
;---------------------------------------------------------------------
SENDUART
            return                  ; put serial xmit routine here
;=====================================================================
; test string storage
;---------------------------------------------------------------------
            org         0100h
;
InputStr
            addwf       PCL,f
            dt          7h,10h,"This is a test. ",0
;
;=====================================================================
            end
```

12.1.5. Algorithm 4: "Visual Basic" Table Implementation

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

```
'Written by Crystalfontz America, Inc. 2004 http://www.crystalfontz.com
'Free code, not copyright copy left or anything else.
'Some visual basic concepts taken from:
'http://www.planet-source-
code.com/vb/scripts/ShowCode.asp?txtCodeId=21434&lngWId=1
'most of the algorithm is from functions in 633_WinTest:
'http://www.crystalfontz.com/products/633/633_WinTest.zip
'Full zip of the project is available in our forum:
'https://www.crystalfontz.com/forum/showthread.php?postid=9921#post9921

Private Type WORD
   Lo As Byte
   Hi As Byte
End Type

Private Type PACKET_STRUCT
   command As Byte
   data_length As Byte
   data(22) As Byte
   crc As WORD
End Type

Dim crcLookupTable(256) As WORD

Private Sub MSComm_OnComm()
'Leave this here
End Sub

'My understanding of visual basic is very limited--however it appears that there is
no way to initialize an array of structures.
Sub Initialize_CRC_Lookup_Table()
  crcLookupTable(0).Lo = &H0
  crcLookupTable(0).Hi = &H0
  . . .
'For purposes of brevity in this Datasheet, 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
```

12.1.6. 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);
    }
  }
```

12.1.7. 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);
```

12.1.8. Algorithm 7: For PIC18F8722 or PIC18F2685

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

```
; CRC Algorithm for CrystalFontz CFA635 display (DB535)
; This code written for PIC18F8722 or PIC18F2685
; Your main focus here should be the ComputeCRC2 and CRC16_ routines
;=================================================================
ComputeCRC2:
```

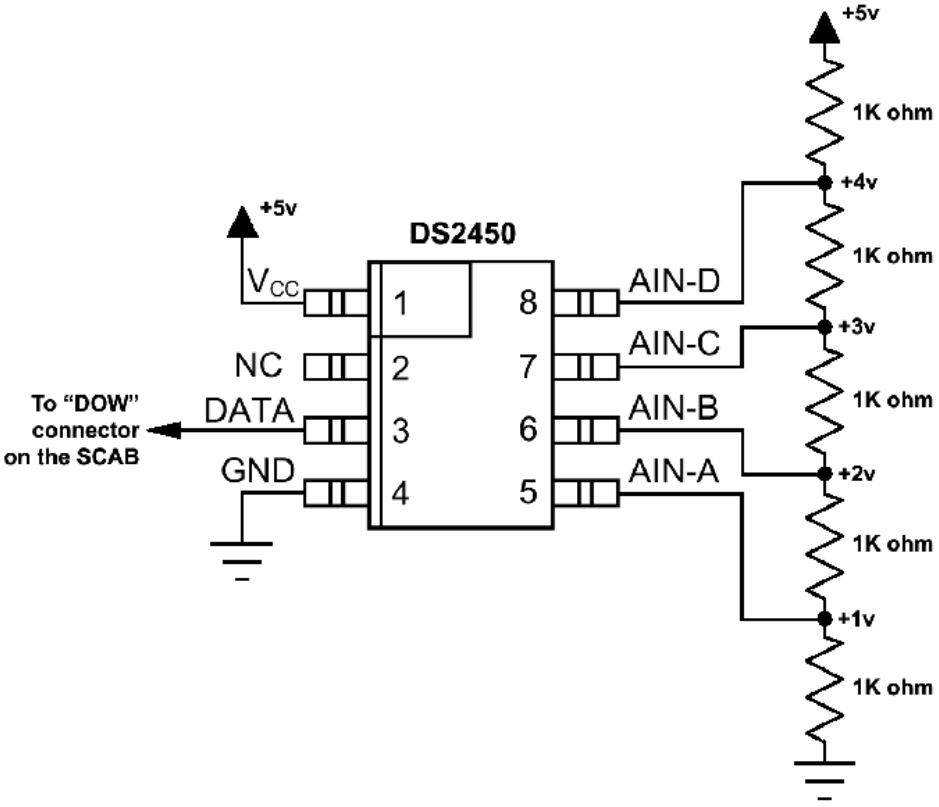
```
        movlb       RAM8
        movwf       dsplyLPCNT          ;w has the byte count
nxt1_dsply:
        movf        POSTINC1            ;w
        call        CRC16
        decfsz      dsplyLPCNT
        goto        nxt1_dsply
        movlw       .0                  ;shift accumulator 16 more bits
        call        CRC16
        movlw       .0
        call        CRC16
        comf        dsplyCRC,F          ;invert result
        comf        dsplyCRC+1,F
        return
;=====================================================================
CRC16   movwf:
        dsplyCRCData                    ;w has the byte crc
        movlw       .8
        movwf       dsplyCRCCount
_cloop:
        bcf         STATUS,C            ; clear carry for CRC register shift
        rrcf        dsplyCRCData,f      ; perform shift of data into CRC
                                        ; register
        rrcf        dsplyCRC,F
        rrcf        dsplyCRC+1,F
        btfss       STATUS,C            ; skip jump if carry
        goto    _   notset              ; otherwise goto next bit
        movlw       0x84                ; XOR poly mask with CRC register
        xorwf       dsplyCRC,F
_notset:
        decfsz      dsplyCRCCount,F     ; decrement bit counter
        bra    cloop                    ; loop if not complete
        return
;=====================================================================
; example to clear screen
dsplyFSR1_TEMP      equ    0x83A  ;     ; 16-bit save for FSR1 for display
                                        ; message handler
dsplyCRC            equ    0x83C        ; 16-bit CRC (H/L)
dsplyLPCNT          equ    0x83E        ; 8-bit save for display message
                                        ; length - CRC
dsplyCRCData        equ    0x83F        ; 8-bit CRC data for display use
dsplyCRCCount equ    0x840         ; 8-bit CRC count for display use
SendCount           equ    0x841        ; 8-bit byte count for sending to display
RXBUF2              equ    0x8C0        ; 32-byte receive buffer for Display
TXBUF2              equ    0x8E0        ; 32-byte transmit buffer for Display
;---------------------------------------------------------------
ClearScreen:
        movlb       RAM8
        movlw       .0
        movwf       SendCount
        movlw       0xF3
        movwf       dsplyCRC      ; seed ho for CRC calculation
        movlw       0x21
        movwf       dsplyCRC+1          ; seen lo for CRC calculation
        call        ClaimFSR1
        movlw       0x06
        movwf       TXBUF2
        LFSR        FSR1,TXBUF2
        movf        SendCount,w
        movwf       TXBUF2+1            ; message data length
        call        BMD1
        goto        SendMsg
;=====================================================================
;send message via interrupt routine. The code is made complex due
; to the limited FSR registers and extended memory space used
;
```

```
; example of sending a string to column 0, row 0
;---------------------------------------------------------------
SignOnL1:
       call          ClaimFSR1
       lfsr          FSR1,TXBUF2+4 ; set data string position
       SHOW          C0R0,BusName        ; move string to TXBUF2
       movlw         .2                  ;
       addwf         SendCount           ;
       movff         SendCount,TXBUF2+1
                                         ; insert message data length
       call          BuildMsgDSPLY
       call          SendMsg
       return
;===============================================================
; BuildMsgDSPLY used to send a string to LCD
;---------------------------------------------------------------
 BuildMsgDSPLY:
       movlw         0xF3
       movwf         dsplyCRC            ; seed hi for CRC calculation
       movlw         0x21
       movwf         dsplyCRC+1          ; seed lo for CRC calculation
       LFSR          FSR1,TXBUF2         ; point at transmit buffer
       movlw         0x1F                ; command to send data to LCD
       movwf         TXBUF2              ; insert command byte from us to
                                         ; CFA635
       BMD1          movlw .2
       ddwf          SendCount,w         ; + overhead
       call          ComputeCRC2         ; compute CRC of transmit message
       movf          dsplyCRC+1,w
       movwf         POSTINC1            ; append CRC byte
       movf          dsplyCRC,w
       movwf         POSTINC1            ; append CRC byte
       return
;===============================================================
SendMsg:
       call          ReleaseFSR1
       LFSR          FSR0,TXBUF2
       movff         FSR0H,irptFSR0
       movff         FSR0L,irptFSR0+1
                                         ; save interrupt use of FSR0
       movff         SendCount,TXBUSY2
       bsf           PIE2,TX2IE
                                         ; set transmit interrupt enable
                                         ; (bit 4)
       return
;===============================================================
; macro to move string to transmit buffer
SHOW macro     src,  stringname
       call          src
       MOVLF         upper stringname, TBLPTRU
       MOVLF         high stringname, TBLPTRH
       MOVLF         low stringname, TBLPTRL
       call          MOVE_STR
       endm
;===============================================================
MOVE_STR:
       tblrd         *+
       movf          TABLAT,w
       bz            ms1b
       movwf         POSTINC1
       incf          SendCount
       goto          MOVE_STR

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

13. APPENDIX B: CONNECTING A DS2450 1-WIRE QUAD A/D CONVERTER

This appendix describes a simple test circuit that demonstrates how to connect a DS2450 4-channel ADC to the CFA-533's DOW (Dallas One Wire - The DS2450 uses the standard Dallas Semiconductor 1-Wire protocol for data transfers) connector. It also gives a sample command sequence to initialize and read the ADC. Up to 32 DOW devices can be connected to the CFA-533. In this example the DS2450 appears at device index 0. Host software should query the connected devices using command 18 (0x12): Read DOW Device Information to verify the locations and types of DOW devices connected.



Refer to the DS2450 Datasheet and the description for command 20 (0x14): Arbitrary DOW Transaction for more information.

Start 533WinTest (works with CFA-533) and open the Packet Debugger dialog. Select Command 20 = Arbitrary DOW Transaction, then paste each string below into the data field and send the packet.

The response should be similar to what is shown.

```
//Write 0x40 (=64) to address 0x1C (=28) to leave analog circuitry on
//(see page 6 of the data sheet)
<command 20> \000\002\085\028\000\064
<response> C=84(d=0):2E,05,22 //16 bit "i-button" CRC + 8-bit "DOW" CRC
                              //Consult "i-button" docs to check 16-bit CRC
                              //DOW CRC is probably useless for this device.
//Write all 8 channels of control/status (16 bits, 5.10v range)
<command 20> \000\002\085\008\000\000 // address = 8, channel A low
<response> C=84(d=0):6F,F1,68 // 16-bits, output off

<command 20> \000\002\085\009\000\001 // address = 9, channel A high
<response> C=84(d=0):FF,F1,AB // no alarms, 5.1v

<command 20> \000\002\085\010\000\000 // address = 10, channel B low
<response> C=84(d=0):CE,31,88 // 16-bits, output off

<command 20> \000\002\085\011\000\001 // address = 11, channel B high
<response> C=84(d=0):5E,31,4B // no alarms, 5.1v

<command 20> \000\002\085\012\000\000 // address = 12, channel C low
<response> C=84(d=0):2E,30,A3 // 16-bits, output off

<command 20> \000\002\085\013\000\001 // address = 13, channel C high
<response> C=84(d=0):BE,30,60 // no alarms, 5.1v

<command 20> \000\002\085\014\000\000 // address = 14, channel D low
<response> C=84(d=0):8F,F0,43 // 16-bits, output off

<command 20> \000\002\085\015\000\001 // address = 15, channel D high
<response> C=84(d=0):1F,F0,80 // no alarms, 5.1v

//Read all 4 channels of control/status (check only)
<command 20> \000\010\170\008\000
<response> C=84(d=0):00,01,00,01,00,01,00,01,E0,CF,01

//Repeat next two commands for each conversion (two cycles shown)

//Start conversion on all channels
<command 20> \000\002\060\015\000
<response> C=84(d=0):3A,03,28

//Read all 8 channels
<command 20> \000\010\170\000\000
<response> C=84(d=0):00,33,DF,64,84,96,6A,C8,5A,6B,BE

//Decoded response:
0x3300 = 13056 1.016015625 volts (channel A)
0x64DF = 25823 2.009541321 volts (channel B)
0x9684 = 38532 2.998553467 volts (channel C)
0xC86A = 51306 3.992623901 volts (channel D)

//Start conversion on all channels
<command 20> \000\002\060\015\000
<response> C=84(d=0):3A,03,28

//Read all 8 channels
<command 20> \000\010\170\000\000
<response> C=84(d=0):6B,33,B2,64,97,96,42,C8,0F,C9,0A

//Decoded response:
0x336B = 13163 1.024342346 volts (channel A)
0x64B2 = 25778 2.006039429 volts (channel B)
0x9697 = 38551 3.000032043 volts (channel C)
0xC842 = 51266 3.989511108 volts (channel D)
```