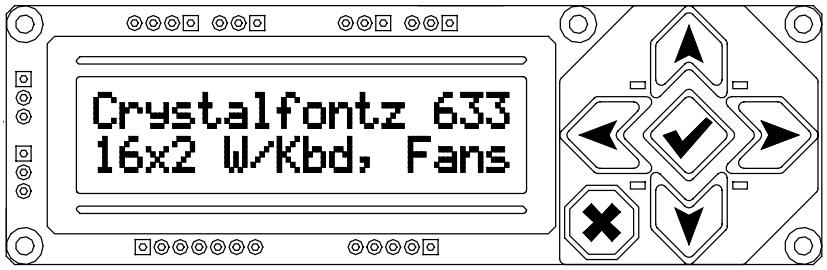
Crystalfontz America, Incorporated

LCD MODULE SPECIFICATIONS



Crystalfontz Model Number	**CFA-633**
Hardware Revision	**v1.5a**
Firmware Version	**k1.9**
Data Sheet Version	**k1.9a**
Product Page	**http://www.crystalfontz.com/products/633/**

Customer Name	
Customer Part Number	

Crystalfontz America, Inc.
12412 East Saltese Avenue
Spokane Valley, WA 99216-0357

Phone: (509) 892-1200
Fax: (509) 892-1203

e-mail: sales@crystalfontz.com
http://www.crystalfontz.com

Table of Contents

■ REVISION HISTORY

HARDWARE	
2005/19/19	Start Public Version Tracking. Current hardware revision: **v1.5a**

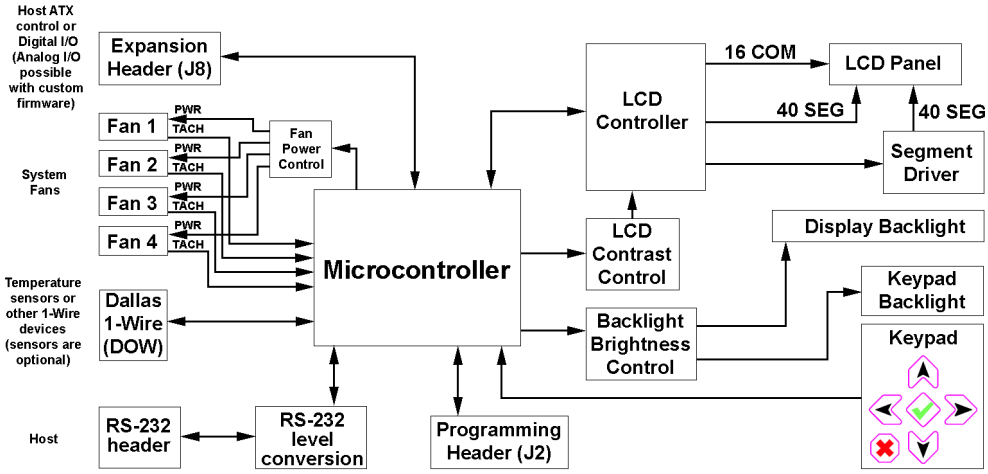
FIRMWARE	
2005/19/19	Start Public Version Tracking. Current firmware version: **k1.9** Command "1: Get Hardware & Firmware Version" returns: **"CFA633:h1.5,k1.9"**

DATA SHEET	
2005/19/19	Start Public Version Tracking. Current data sheet version: **k1.9a** *Changes from last released version (v1.9):* • Added "REVISION HISTORY" • Added "GPIO Current Limits" • Added "Appendix C: Calculating the CRC" • Added note about operating system delays • Added note regarding length of Command 30 reply

FEATURES

- New in hardware version v1.5a / firmware version v1.9:
  - "X,Y" LCD command to allow partial updates of the LCD
  - 115,200 baud supported
  - Advanced digital GPIO control with PWM output
- LED backlit STN 16x2 LCD
- LED backlit 6-button translucent silicon keypad with screened legend
- Several color choices:
  - Yellow-green backlit positive mode LCD, yellow/green keypad backlight ("YYB")
  - White backlit negative mode LCD, blue keypad backlight ("TMC")
  - Red backlit negative mode LCD, red keypad backlight ("RMC")
- Bi-directional 19200/115200 baud ESD protected RS-232 interface
- Robust packet-based communications protocol with 16-bit CRC
- Compact size: fits in a 1U rack mount case (35mm overall height)
- ATX power supply control functionality allows the buttons on the CFA-633 to replace the "power" and "reset" buttons on your system, simplifying your front panel design
- Built-in factory re-programmable microcontroller
- Four fan connectors with RPM monitoring and variable PWM fan power control
- "Fail-Safe" fan power setting allows safe host fan control based on temperature
- Temperature monitoring: up to 32 channels at up to 0.5 deg C absolute accuracy (Dallas 1-Wire sensors optional: Crystalfontz part number WRDOWY17)
- Hardware watchdog can reset host on host software failure
- "Live Display" shows temperature and fan readings without host intervention, allowing fans and temperatures to be checked immediately at boot, even before the host operating system is loaded.
- Non-volatile memory capability ("EEPROM"):
  Customize the "power-on" settings
  16-byte "scratch" register for storing IP, netmask, system serial number . . .
- RS-232 to Dallas Semiconductor 1-Wire bridge functionality allows control of other 1-Wire compatible devices (ADC, voltage monitoring, current monitoring, RTC, GPIO, counters, identification/encryption) (additional hardware required)
- Customizable firmware and configurable hardware can be modified at the factory to add specific features for your system needs. Minimum order and tooling fee may apply for custom firmware:
  - Analog I/O
  - Provide "dongle" functionality for system copy protection
  - Autonomous hardware monitoring
  - Customize the "power-on" settings
- 5.25" half-height drive-bay mounting bracket available (optional)

■ SYSTEM BLOCK DIAGRAM



■ MECHANICAL CHARACTERISTICS

Item	Description	Unit
PCB Outline Size	110.5(W) x 35.0(H)	mm
Viewing Area	61.0(W) x 15.8(H)	mm
Active Area	56.2(W) x 11.2(H)	mm
Character Size	2.95(W) x 5.55(H)	mm
Dot Size	0.55(W) x 0.65(H)	mm
Dot Pitch	0.60(W) x 0.70(H)	mm
Thickness:		
Without Keypad or Connectors	10.7	mm
With Keypad, without Connectors	12.1	mm
Without Keypad, with Connectors	20.1	mm
With Keypad, with Connectors	24.1	mm
Keystroke Travel (approximate)	2.4	mm

■ GENERAL SPECIFICATIONS

Operating Temperature: 0°C minimum, 50°C maximum
Storage Temperature: -10°C minimum, 60°C maximum
LCD Glass Type: STN, Yellow/Green Positive Mode or Blue Negative Mode
Viewing Direction: 6 O'clock
Polarizer Type: Transflective
Driving Method: 1/16 Duty, 1/5 Bias
Backlight: LED, Yellow/Green (568nm nominal), Red or White
Backlight PWM frequency: 320Hz nominal
Weight: 46 grams typical
Fan tachometer speed range (assuming 2 ppr): 600rpm to 300000rpm
Fan Power Control PWM Frequency: 18 Hz nominal

■ ELECTRICAL SPECIFICATIONS

Required voltage supplies:
+5v (logic): 4.75v minimum, 5.0v nominal, 5.25v maximum
+12v (backlights): 11v minimum, 12v nominal, 13v maximum
+12v (fans): 4.75v minimum, 12v nominal, 13v maximum *

*JP7 must be opened for the operating range of "+12v (fans)" to be different from "+12v (backlights)". If JP7 is closed (factory default) then the range of "+12v (backlights)" must be observed for both "+12v (backlights)" and "+12v (fans)" Please see the diagram under "■ POWER CONNECTIONS".

Current consumption:
+5v (logic): 13mA typical for LCD and microcontroller
+12v (backlights typical for 100% brightness):
Yellow: 45mA
White/Blue: 42mA
Red: 57mA
+12v (fans): Draw on +12v for fans will vary, depending on the user equipment connected to FAN1 through FAN4. Maximum continuous current draw must be no more than 1.5A per fan connector, no more than 4A total. Pulsed current may be up to 5A per connector, the pulse width must be less than 50mS. This pulse specification allows for the fan's start-up current spike.

GPIO Current Limits:
Sink: 25mA
Source: 10mA

ESD (Electro-Static Discharge) Specifications:
Tx and Rx pins of connector "RS232" only:
+15kV Human Body Model
+15kV IEC1000-4-2 Air Discharge
+8kV IEC1000-4-2 Contact Discharge

The remainder of the circuitry is industry standard CMOS logic, which is susceptible to ESD damage. Please exercise industry standard static precautions as you would for any other bare PCB such as expansion cards or motherboards.

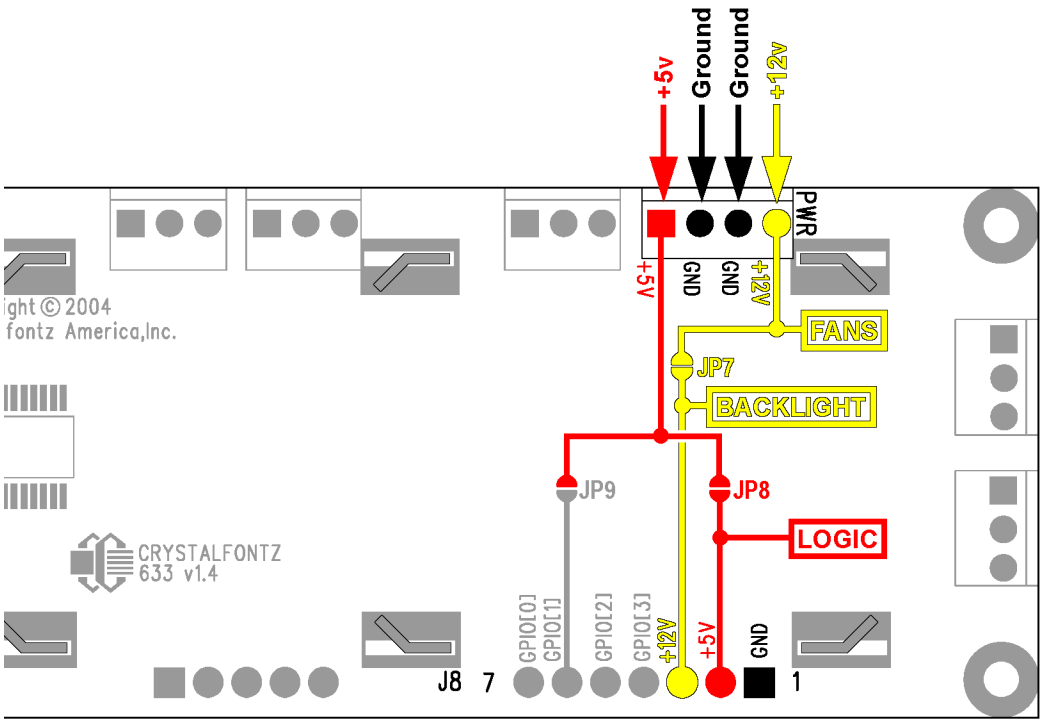
■ RELIABILITY

LCD portion (less the keypad & backlight):
50,000 to 100,000 hours.
Keypad:
1,000,000 keystrokes.
Yellow/Green and Red LED backlights:
50,000 to 100,000 hours.
White / Blue LED backlights:
10,000 hours to 70% of original brightness

■ POWER CONNECTIONS

JP7 and JP8 are closed at the factory, allowing a single standard "3 ½ inch floppy" power supply cable to provide power to the CFA-633. JP7 can be opened in order to supply power to the fans separate from the power to the backlight. See "■ JUMPER REFERENCE" for jumper positions.

Note: Do not connect high power fans to the module when +12v is supplied through J8, Pin 3 and JP7 is closed. Total fan current must be less than 500mA if +12v is supplied through J8, Pin 3 and JP7 is closed. When using high power fans, supply the +12v through the "PWR" connector.

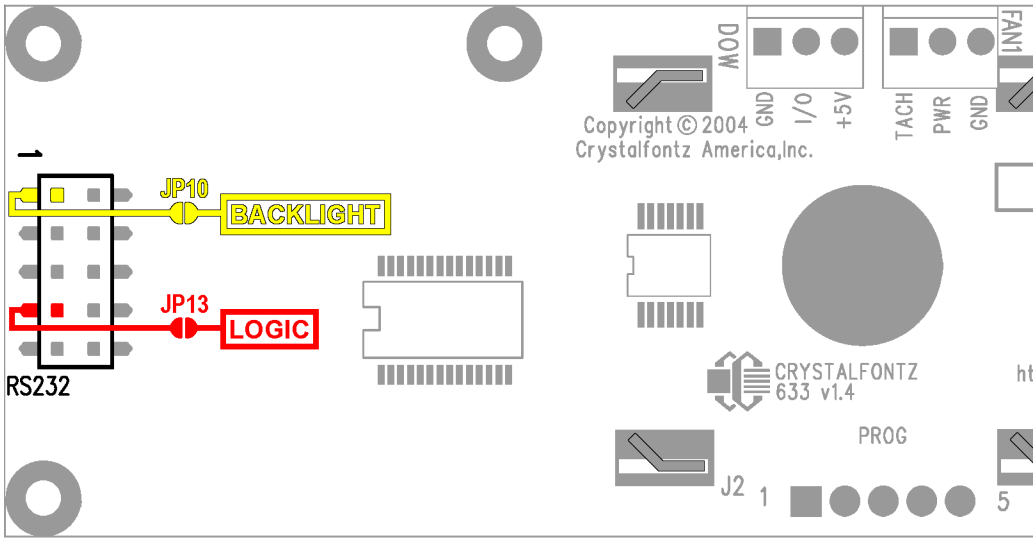


■ POWER CONNECTION THROUGH J1 (RS-232)

+5v and +12v power can be supplied through connector J1, allowing a single cable to contain both power and data connections. If the “Default RS-232 Pin Assignments” are selected, the five connections need to operate the module are all on a single column of pins on J1, which allows a single 0.1” spacing 5-conductor cable to connect between the 633 and your embedded system.

To enable +5v to be supplied through J1, close JP13. To allow +12v to be supplied through J1, close JP10. JP10 and JP13 are open by default from the factory. See “■ JUMPER REFERENCE” for jumper positions.

Note: Do not connect high power fans to the module when +12v is supplied through J1/JP10. Total fan current must be less than 500mA if +12v is supplied through J1/JP10. When using high power fans, supply the +12v through the “PWR” connector.



■ ATX POWER SUPPLY CONTROL CONNECTIONS

The CFA-633 has the ability to control power on/off and reset functions of a standard “ATX” PC.

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

Crystalfontz has a cable available (WRPWRY14) that simplifies ATX power control connections. When using this cable, please open jumper JP8 and close jumper JP9 in order to ensure correct operation. See “■ JUMPER REFERENCE” for jumper positions.

If the WRPWRY14 is ordered at the same time as the CFA-633, Crystalfontz will install the WRPWRY14, open jumper JP8 and close jumper JP9 unless otherwise instructed.

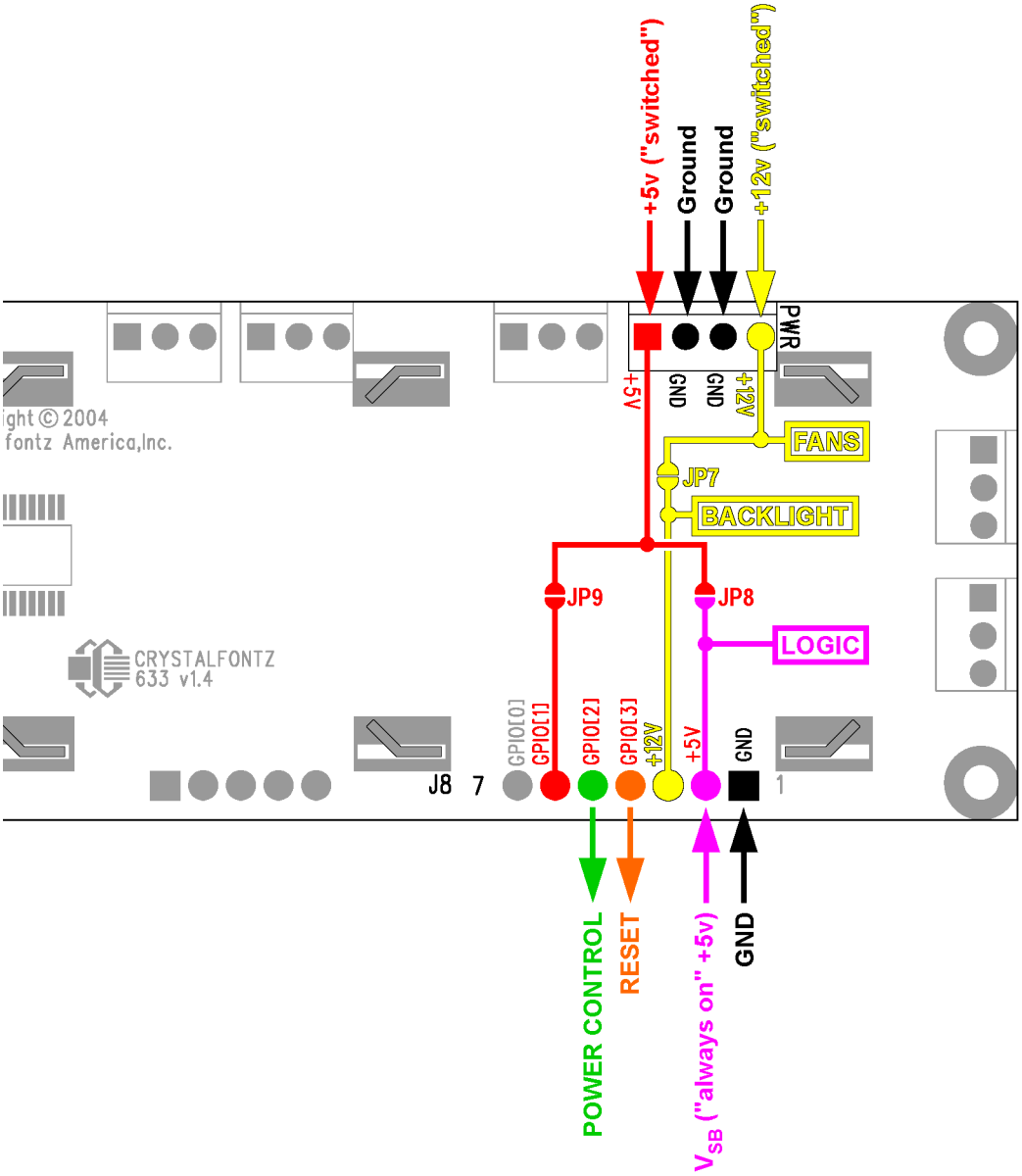
For this functionality, the CFA-633 is powered from the PC’s V_{SB} signal (the “stand-by” or “always-on” +5v ATX power supply output).

By default, the pin labeled “+5v” on the CFA-633’s connector J8 is electrically connected to the pin “+5v” on the CFA-633’s connector labeled “PWR”. If you are using the CFA-633 to do ATX power control, you will want to open jumper JP8, which will disconnect the “+5v” of the “PWR” connector from the “+5v” of connector J8.

Since the CFA-633 must act differently depending on if the host’s power supply is “on” or “off” you must also connect the host’s “switched +5v” to the pin 6 of the CFA-633’s connector J8 (labeled as “GPIO[1]”). This pin functions as “POWER-ON SENSE”. The “POWER-ON SENSE” pin is configured as an input with a pull-down (5KΩ nominal). To simplify this connection, JP9 is closed to make a connection between the “+5v” pin on the CFA-633’s “PWR” connector and the “POWER-ON SENSE” pin.

The motherboard’s “power switch” input is connected to pin 5 of the CFA-633’s connector J8 (labeled as “GPIO[2]”). This pin functions as “POWER CONTROL”. The “POWER CONTROL” pin is configured as a high-impedance input until the CFA-633 wants to turn the host on or off, then it will change momentarily to low impedance output, driving either low or high depending on the setting of POWER_INVERT (see command 28, Set ATX Power Switch Functionality).

The motherboard’s “reset switch” input is connected to pin 4 of the CFA-633’s connector J8 (labeled as “GPIO[3]”). This pin functions as “RESET”. The “RESET” pin is configured as a high-impedance input until the CFA-633 wants to reset the host, then it will change momentarily to low impedance output, driving either low or high depending on the setting of RESET_INVERT (see command 28, Set ATX Power Switch Functionality). This connection is also used for the hardware watchdog (see command 29: Enable/disable and reset the watchdog).



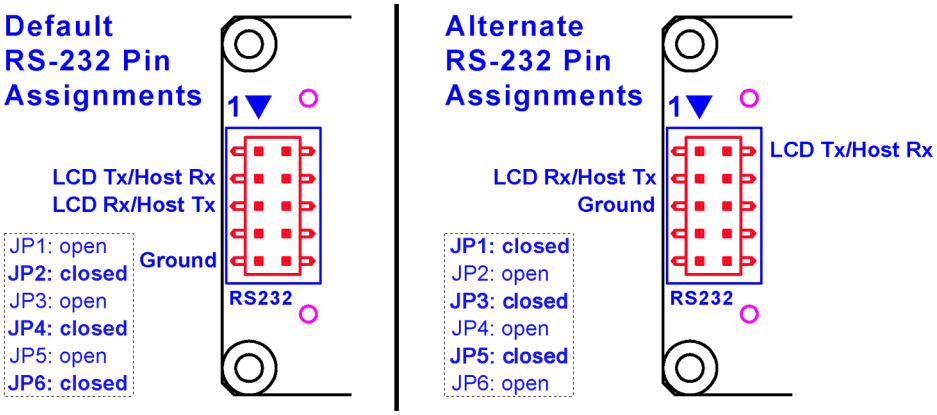
Once configured by the host software (see command 28, Set ATX Power Switch Functionality), the following functions may be individually enabled:

- System power on. If “POWER-ON SENSE” is low (0v), pressing the green check key for 0.25 seconds will turn the unit on by driving “POWER CONTROL” line for the pulse width set by command 28, Set ATX Power Switch Functionality (1.0 seconds default).
- System hard power off: If “POWER-ON SENSE” is high (+5v) pressing the red “X” key for 4 seconds will turn the system off by driving the “POWER CONTROL” line. The line will be driven for a minimum of the pulse width set by command 28, Set ATX Power Switch Functionality (1.0 seconds default). If the user continues to press the key, the CFA-633 will continue to drive the line for up to an additional 5 seconds.

- If “POWER-ON SENSE” is high (+5v) pressing the green check key for 4 seconds will reset the system off by driving the “RESET” line for 1 second. The CFA-633 will re-boot itself immediately after resetting the host.
- Since the computer and LCD must look “off” if the computer’s power is “off” the CFA-633 can be configured to monitor the POWER-ON SENSE line and blank its display any time the POWER-ON SENSE line is low. If +12v remains active (which would not be expected, since the host is “off”), the fans and backlight will remain on at their previous settings.

■ RS-232 CONNECTIONS

JP2, JP4 & JP6 are closed at the factory, selecting the “Default RS-232 Pin Assignments”. This connection allows a low-cost ribbon cable (Crystalfontz part number WR232Y08) to connect the CFA-633 to a PC’s DB-9 COM port. By opening JP2, JP4 & JP6 and closing JP1, JP3 & JP5 you can select the “Alternate RS-232 Pin Assignments”. See “■ JUMPER REFERENCE” for jumper positions.

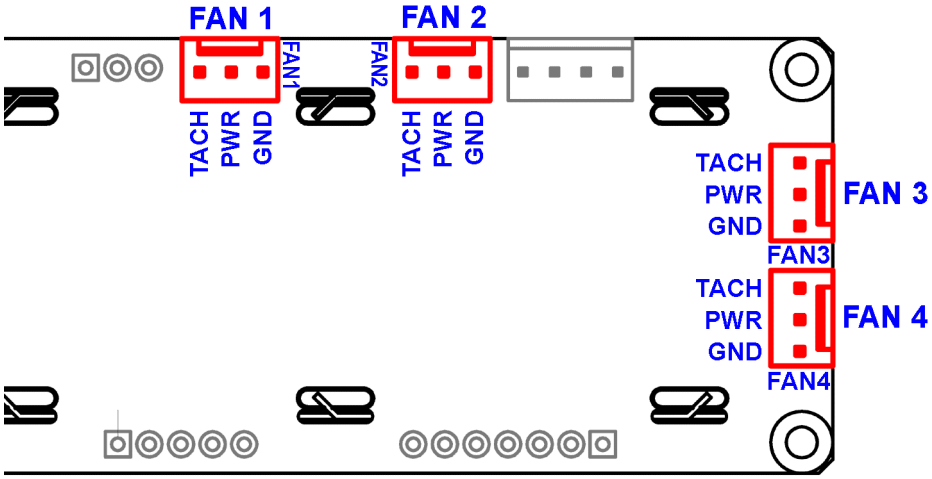


If there is a matching 0.1” center, 10-pin RS-232 connector on your system’s motherboard, then in most cases a simple straight-through ribbon cable (such as CW Industries C3AAG-1018G-ND) can be used to connect from the CFA-633 to the motherboard’s header. The pin order of your motherboard’s header will determine if the CFA-633’s pin assignments need to be “Default” or “Alternate”.

Please note that the CFA-633 can be powered through this header. Please refer to “■ POWER CONNECTION THROUGH J1 (RS-232)” above for details.

■ FAN CONNECTIONS

The CFA-633 supports up to 4 standard “3-pin” cooling fans. The fan connectors are compatible with industry standard “3-pin” fans.



The average power delivered to each fan may be set to any level between 0% and 100%. The power setting controls the PWM duty cycle of a high-performance open-drain FET connected between the system ground and the GND pin of each fan connector. The PWM frequency is nominally 18Hz.

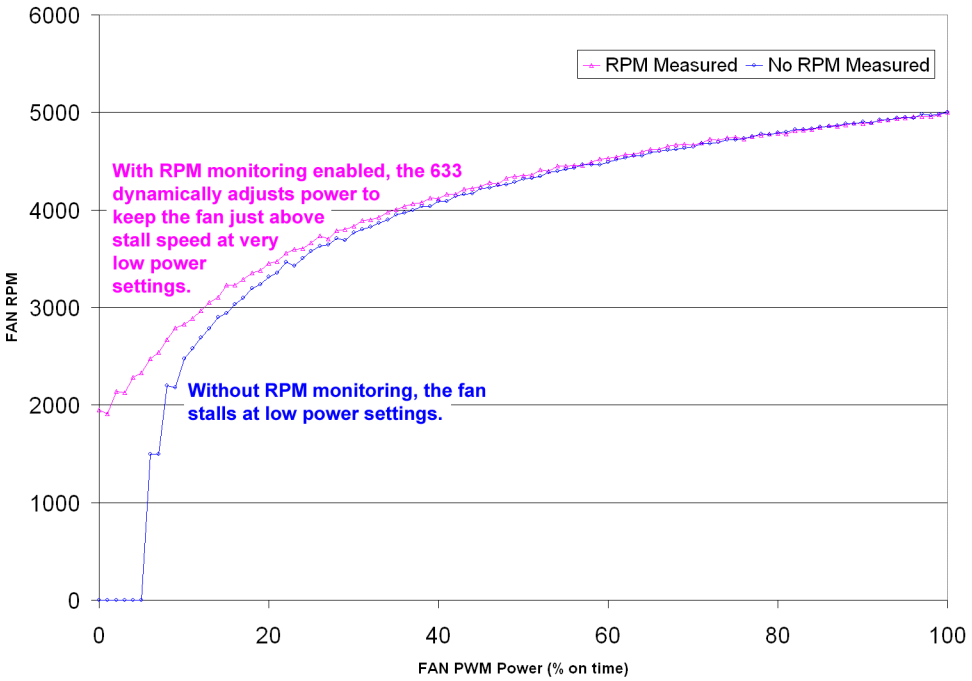
The CFA-633 can measure the frequency of the fan’s tachometer signal, and given the pulses-per-revolution, calculate the RPM and display it on the LCD or report the information needed to calculate the RPM to the host. If a fan’s power is set to 100%, then the average frequency of each fan’s tachometer signal is taken over a 1/8 second (125mS) period of time. Each fan is measured in sequence, so there is updated fan speed information available every 1/2 second (500mS) for each fan.

The power to a fan must be on in order for the fan’s tachometer signal to be valid. If a fan is configured to display its speed on the LCD or report its speed to the host, the power of the fan will be unconditionally set to 100% at the start of the 1/8 second period of time when the CFA-633 is measuring the frequency of the tachometer signal--overriding the PWM. The CFA-633 will leave the power to the fan on until the glitch delay (see command 26: Set Fan Tachometer Glitch Delay) has expired and two tachometer edges have been detected. The normal PWM cycle will then resume.

This technique allows the fan speed to be measured with a very minimal effect on the speed of the fan. If the fan power is set to 100% or if the speed of the fan and length of the PWM on time are such that the speed can be measured without stretching the PWM, then this override will not change the speed of the fan at all. If the fan power is set to some level other than 100%, and the PWM on time is short compared to the tachometer signal frequency, then the fan speed will “pulse” slightly every 1/2 second due to the stretching of

the PWM on time. During tachometer measuring, the maximum width of a stretched on pulse is 1/8 second. For some fans, the result is not very noticeable, and this technique will allow you to monitor the average speed of the fan while controlling the average power of the fan. For other fans (particularly high torque, high RPM models) the pulsing effect may be undesirable.

Since the on-time is dynamically stretched by the CFA-633 to force the fan to produce two tachometer edges, the result is that the fan will resist stalling as power is reduced towards 0% and the RPM is being measured. Here is a graph of fan RPM vs. the fan power setting for a typical high-performance 80mm fan (Delta FFB0812SHE):



Typically if the fan speed is not at 100%, then it is being controlled by the host software to drive a temperature sensor to a given reading in a closed-loop arrangement. In this case, the temperature, rather than the fan speed, would be monitored for out-of range conditions. If the temperature is within specification, you really do not care how fast the fan is turning. In an unattended system, it may be a good idea to set each fan to 100% for a few seconds during a test cycle—perhaps once a day or once a week—and log the steady-state RPM attained by the fan. If that steady state RPM were higher (this can be caused by a blocked airflow) or lower (perhaps the fan's bearings are failing) than expected, a maintenance warning would be generated by the host software.

For safety, enable the fan power fail-safe (see command 25: Set Fan Power Fail-Safe) on any fans involved in host-based speed control. By enabling the fail-safe on a fan that is being used in closed-loop control through host software, the CFA-633 will turn that fan to 100% if the host fails to update the power of the fans within a given time interval. For instance, if the communications cable were dislodged, the host operating system hangs, or cooling control process is terminated, the CFA-633 will automatically force those fans to 100%, preventing potential equipment damage due to lack of cooling.

We have tested the CFA-633 with a large range of fans and had good results. However, you are responsible for determining if the control and monitoring methods employed by the CFA-633 are acceptable for your application. In particular, if a fan's power is set too low, it may stall or fail to start, providing no cooling. Using a PWM to control fan speed is generally accepted, however we make no claims that it is compatible with any particular fan or that it does not affect the lifetime of the fans. Some higher torque fans (and especially the ball-bearing models) may click, buzz, or growl at low power settings due to the torque in the fan going from positive to negative in each PWM cycle. If you limit the power setting to 0% or 100% there should be no compatibility issues.

When power is applied to the CFA-633, the CFA-633 will set each fan's power to the factory default value of 100%, or to the value that is stored in the boot state. To minimize peak current loading on the +12v supply during start-up, the fans are started in sequence with a 0.5 second delay between any fans that are on.

■ TEMPERATURE SENSOR CONNECTIONS

The CFA-633 supports Dallas Semiconductor “1-Wire” temperature sensors (“DOW” is used in this data sheet as an abbreviation for “Dallas One Wire”). Any combination of up to 32 DS1822 (2°C absolute accuracy) or DS18B20 (0.5°C absolute accuracy) temperature sensors or other DOW compatible devices are directly supported.

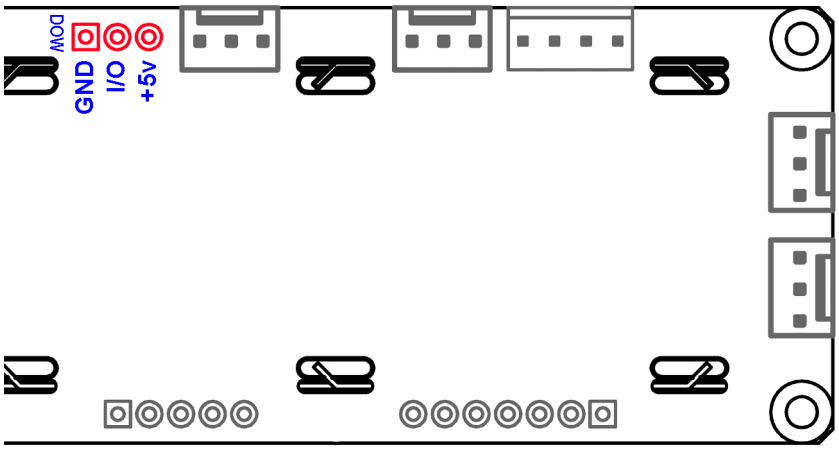
Crystalfontz can supply the WRDOWY17 temperature sensor assembly, which contains a DS18B20 attached to a “daisy chain” cable. If a WRDOWY17 is ordered at the same time as a CFA-633, Crystalfontz can load the WRDOWY17’s mating connector into the CFA-633’s “DOW” position. For reference, the mating connector for the WRDOWY17 is a Molex 70543-0002.

Any temperature sensor can be configured to be automatically read and displayed to the CFA-633’s LCD in °C or °F (see command 21: Set Up Live Fan or Temperature Display). Independently, any temperature sensor can be configured to report to the host (see command 19: Set Up Temperature Reporting). Any sensors configured to be displayed or reported are updated once each second.

Any other 1-Wire devices may be connected to the 1-Wire bus, with the CFA-633 acting as a bridge between RS-232 and the 1-Wire bus (see command 20: Arbitrary DOW Transaction). The total number of 1-Wire devices supported is 32, including directly supported temperature sensors and any other user-provided 1-Wire devices.

The CFA-633 has a 1KΩ hardware pull-up on the DOW connector’s I/O line.

Connect the 1-Wire sensors as detailed in the sensor’s data sheet.



■ HOST COMMUNICATIONS

The CFA-633 communicates with its host using an RS-232 interface. The port settings are 19200 baud, 8 data bits, no parity, 1 stop bit by factory default. The speed can be set to 115,200 baud under software control (see command 33: Set Baud Rate).

All communication between the CFA-633 and the host takes place in the form of a simple and robust CRC checked packet. The packet format allows for very reliable communications between the CFA-633 and the host without the traditional problems that occur in a stream based serial communication (such as having to send data in inefficient ASCII format, having to "escape" certain "control characters", or losing sync if a character is corrupted, missing or inserted).

PACKET STRUCTURE

All packets have the following structure:

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

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

```
TTcc cccc
|||| ||||--Command, response, error or report code 0-63
||---------Type:
             00 = normal command from host to CFA-633
             01 = normal response from CFA-633 to host
             10 = normal report from CFA-633 to host (not in
                  direct response to a command from the host)
             11 = error response from CFA-633 to host (a packet
                  with valid structure but illegal content
                  was received by the CFA-633)
```

data_length specifies the number of bytes that will follow in the data field. The valid range of **data_length** is 0 to 18.

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

CRC is a standard 16-bit **CRC** of all the bytes in the packet except the CRC itself. At the port, the CRC immediately follows the last used element of **data[]**. See "Appendix C: Calculating the CRC" for details.

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

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

Crystalfontz supplies a demonstration and test program, 633_WinTest, along with its C source code. Included in the 633_WinTest source is the CRC algorithm, and an algorithm that will detect packets. The algorithm will automatically re-synchronize to the next valid packet in the event of any communications errors.

HANDSHAKING

The packet nature of the CFA-633 makes it unnecessary to implement traditional hardware or software handshaking.

The host should wait for an acknowledge packet from the CFA-633 before sending the next command packet. The CFA-633 will respond to all packets within 250mS, so the host software should stop waiting and retry the packet if the CFA-633 fails to respond within 250mS. Please note that some operating systems may introduce delays between when the data arrives at the physical port from the CFA-633 until it is available to the user program. In this case, the host program may have to increase its timeout window to account for the additional overhead of the operating system. The host software should report an error if a packet is not acknowledged after several retries. This situation would indicate a hardware problem.

Since the CFA-633 can be configured to send several types of report packets, along with regular acknowledge packets, the host should be able to buffer several incoming packets and must guarantee that it can process and remove packets from its input buffer faster than the packets can arrive given the baud rate and the reporting configuration of the CFA-633. For any modern PC using reasonably efficient software, this requirement will not pose a challenge.

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

REPORT CODES

The CFA-633 can be configured to report the following items. These reports will be sent automatically by the CFA-633 when the data becomes available. They are not sent in response to a particular packet received from the host.

0x80: Key Activity

If a key is pressed or released, the CFA-633 will send a Key Activity report packet to the host. Key events may be individually enabled or disabled by command "23: Configure Key Reporting", below.

```
type = 0x80
data_length = 1
data[0] is the type of keyboard activity:
      KEY_UP_PRESS              1
      KEY_DOWN_PRESS            2
      KEY_LEFT_PRESS            3
      KEY_RIGHT_PRESS           4
      KEY_ENTER_PRESS           5
      KEY_EXIT_PRESS            6
      KEY_UP_RELEASE            7
      KEY_DOWN_RELEASE          8
      KEY_LEFT_RELEASE          9
      KEY_RIGHT_RELEASE        10
      KEY_ENTER_RELEASE        11
      KEY_EXIT_RELEASE         12
```

Please note that the CFA-631 will return codes 13 through 20. See the CFA-631 data sheet for more details.

0x81: Fan Speed Report

If any of the four fans is configured to report its speed information to the host, the CFA-633 will send Fan Speed Reports for each selected fan every 1/2 second. See command "16: Set Up Fan Reporting" below.

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

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

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

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

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

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

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

0x82: Temperature Sensor Report

If any of the 32 temperature sensors is configured to report to the host, the CFA-633 will send Temperature Sensor Reports for each selected sensor every second. See the command "19: Set Up Temperature Reporting" below.

```
type = 0x82
data_length = 4
data[0] is the index of the temperature sensor being reported:
      0 = temperature sensor 1
      1 = temperature sensor 2
      . . .
      31 = temperature sensor 32
data[1] is the MSB of Temperature_Sensor_Counts
data[2] is the LSB of Temperature_Sensor_Counts
data[3] is DOW_crc_status
```

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

```
void OnReceivedTempReport(COMMAND_PACKET *packet, char *output)
  {
  //First check the DOW CRC return code from the CFA-633
  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);
    }
  }
```

COMMAND CODES

This is a list of valid commands for the CFA-633. Each valid command packet will be answered by either a response packet or an error packet. The data_length must be less than or equal to 18 in order for a packet to be valid. The low 6 bits of the **type** field of the response or error packet will be the same as the low 6 bits of the **type** field of the command packet being acknowledged.

0: Ping Command

The CFA-633 will return the Ping Command to the host.

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

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

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

1: Get Hardware & Firmware Version

The CFA-633 will return the Hardware and Firmware version information to the host.

```
type = 1
valid data_length is 0
```

The return packet will be:

```
type = 0x40 | 1
data_length = 16
data[] = "CFA633:hX.X,fY.Y"

hX.X is the hardware revision, "h1.5" for example
sY.Y is the firmware version, "k1.9" for example
```

2: Write User Flash Area

The CFA-633 reserves 16 bytes of non-volatile memory for arbitrary use by the host. This memory could be used to store a serial number, IP address, gateway address, netmask, or any other data required. All 16 bytes must be supplied.

```
type = 2
valid data_length is 16
data[] = 16 bytes of arbitrary user data to be stored in
         the CFA-633's non-volatile memory
```

The return packet will be:

```
type = 0x40 | 2
data_length = 0
```

3: Read User Flash Area

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

```
type = 3
valid data_length is 0
```

The return packet will be:

```
type = 0x40 | 3
data_length = 16
data[] = 16 bytes user data recalled from the CFA-633's
         non-volatile memory
```

4: Store Current State As Boot State

The CFA-633 loads its power-up configuration from non-volatile memory when power is applied. The CFA-633 is configured at the factory to display a “welcome” screen when power is applied. This command can be used to customize that welcome screen, as well as many other settings.

The following items are stored by this command:

- Backlight setting (command 14)
- Contrast setting (command 13)
- Cursor position (command 11)
- Cursor style (command 12)
- The characters shown on the LCD (commands 6, 7, 8 & 31)
- The special character font definitions (command 9)
- The fan power settings (command 17)
- The fan glitch delay settings (command 26)
- The key press and release masks (command 23)
- The ATX function enable and pulse length settings (command 28)
- The baud rate (command 33)
- The GPIO settings (command 34)
- The settings of any “live” displays (command 21)

You cannot store the fan or temperature reporting, or the fan fail-safe or host watchdog. The host software should enable these items once the system is initialized.

```
type = 4
valid data_length is 0
```

The return packet will be:

```
type = 0x40 | 4
data_length = 0
```

5: Reboot CFA-633, Reset Host, or Power Off Host

This command instructs the CFA-633 to simulate a power-on restart of itself, reset the host, or turn the host's power off. The ability to reset the host may be useful to allow certain host operating system configuration changes to complete. The ability to turn the host's power off under software control may be useful in systems that do not have ACPI compatible BIOS.

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

Rebooting the CFA-633 may be useful when testing the boot configuration. It may also be useful to re-enumerate the devices on the 1-Wire bus. To reboot the CFA-633, send the following packet:

```
type = 5
valid data_length is 3
data[0] = 8
data[1] = 18
data[2] = 99
```

To reset the host (assuming the host's reset line is connected to GPIO[3] as described in "ATX POWER SUPPLY CONTROL CONNECTIONS" above), send the following packet:

```
type = 5
valid data_length is 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] as described in "ATX POWER SUPPLY CONTROL CONNECTIONS" above), send the following packet:

```
type = 5
valid data_length is 3
data[0] = 3
data[1] = 11
data[2] = 95
```

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

```
type = 0x40 | 5
data_length = 0
```

6: Clear LCD Screen

Sets the contents of the LCD screen DDRAM to ' ' = 0x20 = 32 and moves the cursor to the left-most column of the top line. The LCD contents are one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 6
valid data_length is 0
```

The return packet will be:

```
type = 0x40 | 6
data_length = 0
```

7: Set LCD Contents, Line 1

Sets the characters displayed for the top line of LCD screen. This command has been superseded by command 31: Send Data to LCD. Use this command only if you need backwards compatibility with older CFA-633 units. The LCD contents are one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 7
valid data_length is 16
data[] = top line's display content (must supply 16 bytes)
```

The return packet will be:

```
type = 0x40 | 7
data_length = 0
```

8: Set LCD Contents, Line 2

Sets the characters displayed for the bottom line of LCD screen. This command has been superseded by command 31: Send Data to LCD. Use this command only if you need backwards compatibility with older CFA-633 units. The LCD contents are one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 8
valid data_length is 16
data[] = top line's display content (must supply 16 bytes)
```

The return packet will be:

```
type = 0x40 | 8
data_length = 0
```

9: Set LCD Special Character Data

Sets the font definition for one of the special characters (CGRAM). The LCD CGRAM contents are one of the items stored by the "4: Store Current State As Boot State" command.

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

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

The return packet will be:

```
type = 0x40 | 9
data_length = 0
```

10: Read 8 Bytes of LCD Memory

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

Note: firmware versions prior to v1.9 did not return the address code.

```
type = 10
valid data_length is 1
data[0] = address code of desired data
```

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

```
0x40 ( 64) to 0x7F (127) for CGRAM
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 | 10
data_length = 9
```

`data[0]` of the return packet will be the address code
`data[1-8]` of the return packet will be the data read from the LCD controller's memory.

11: Set LCD Cursor Position

This command allows the cursor to be placed at the desired location on the CFA-633's LCD screen. The LCD cursor position is one of the items stored by the "4: Store Current State As Boot State" command. If you want the cursor to be visible, you may also need to send a "Set LCD Cursor Style" command.

```
type = 11
valid data_length is 2
data[0] = column (0-15 valid)
data[1] = row (0-1 valid)
```

The return packet will be:

```
type = 0x40 | 11
data_length = 0
```

12: Set LCD Cursor Style

This command allows you to determine the style of the hardware-generated cursor shown on the LCD. The LCD cursor style is one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 12
valid data_length is 1
data[0] = cursor style (0-3 valid)
      0 = no cursor
      1 = blinking block cursor
      2 = underscore cursor
      3 = blinking block plus underscore
```

The return packet will be:

```
type = 0x40 | 12
data_length = 0
```

13: Set LCD Contrast

This command sets the contrast or vertical viewing angle of the display. The LCD contrast is one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 13
valid data_length is 1
data[0] = contrast setting (0-50 valid)
       0 = light         (Vlcd=4v)
      16 = about right   (Vlcd=5v)
      29 = dark          (Vlcd=6v)
      30-50 = very dark  (Vlcd=6.0 to 6.7v)
                         (may be useful at cold temperatures)
```

The return packet will be:

```
type = 0x40 | 13
data_length = 0
```

14: Set LCD & Keypad Backlight

This command sets the brightness of the LCD and keypad backlights. The backlight brightness is one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 14
valid data_length is 1
data[0] = backlight power setting (0-100 valid)
      0 = off
      1-99 = variable brightness
      100 = on
```

The return packet will be:

```
type = 0x40 | 14
data_length = 0
```

15: reserved

16: Set Up Fan Reporting

This command will configure the CFA-633 to report the fan speed information to the host every 500mS.

```
type = 16
valid data_length is 1
data[0] = bitmask indicating which fans are enabled to
          report (0-15 valid)
      ---- 8421 Enable Reporting of this Fan's Tach Input
      |||| ||||-- Fan 1: 1 = enable, 0 = disable
      |||| |||--- Fan 2: 1 = enable, 0 = disable
      |||| ||---- Fan 3: 1 = enable, 0 = disable
      |||| |----- Fan 4: 1 = enable, 0 = disable
```

The return packet will be:

```
type = 0x40 | 16
data_length = 0
```

If `data[0]` is not 0, then the CFA-633 will start sending 0x81: Fan Speed Report packets for each enabled fan every 500mS. Each of the report packets is staggered by 1/8 of a second.

Reporting a fan will override the fan power setting to 100% for up to 1/8 of a second every 1/2 second. Please see "■ FAN CONNECTIONS" above for a detailed description.

17: Set Fan Power

This command will configure the power for the fan connectors. The fan power setting is one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 17
valid data_length is 4
data[0] = power level for FAN 1 (0-100 valid)
data[1] = power level for FAN 2 (0-100 valid)
data[2] = power level for FAN 3 (0-100 valid)
data[3] = power level for FAN 4 (0-100 valid)
```

The return packet will be:

```
type = 0x40 | 17
data_length = 0
```

18: Read DOW Device Information

When power is applied to the CFA-633, it detects any devices connected to the Dallas Semiconductor 1-Wire bus (DOW) and stores the device's information. This command will allow the host to read the device's information.

The first byte returned is the Family Code of the Dallas One Wire / iButton device. There is a list of the possible Dallas One Wire / iButton device family codes available in App Note 155: 1-Wire Software Resource Guide on the Dallas / Maxim web site.

Note: The GPIO pin used for DOW must not be configured as user GPIO, and must be configured to its default drive mode in order for the DOW functions to work correctly. These settings are factory default, but may be changed by the user. Please see command 34 Set or Set and Configure GPIO pin.

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

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

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

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

This setting must be saved as the boot state, so when the CFA-633 re-boots it will detect the DOW devices.

```
type = 18
valid data_length is 1
data[0] = device index (0-31 valid)
```

The return packet will be:

```
type = 0x40 | 18
data_length = 9
data[0] = device index (0-31 valid)
data[1-8] = ROM ID of the device
```

If `data[1]` is 0x22 (DS1822 temperature sensor) or 0x28 (DS18B20 temperature sensor), then that device can be set up to automatically convert and report the temperature every second. See the command "19: Set Up Temperature Reporting" below.

19: Set Up Temperature Reporting

This command will configure the CFA-633 to report the temperature information to the host every second.

```
type = 19
valid data_length is 4
data[0-3] = 32-bit bitmask indicating which temperature
            sensors fans are enabled to report (0-255
            valid in each location)

data[0]
      08 07 06 05   04 03  02 01  Enable Reporting of sensor with
      |  |  |  |    |  |   |  |   device index of:
      |  |  |  |    |  |   |  |--  0: 1 = enable, 0 = disable
      |  |  |  |    |  |   |-----  1: 1 = enable, 0 = disable
      |  |  |  |    |  |---------  2: 1 = enable, 0 = disable
      |  |  |  |    |------------  3: 1 = enable, 0 = disable
      |  |  |  |-----------------  4: 1 = enable, 0 = disable
      |  |  |--------------------  5: 1 = enable, 0 = disable
      |  |-----------------------  6: 1 = enable, 0 = disable
      |--------------------------  7: 1 = enable, 0 = disable

data[1]
      16 15 14 13   12 11  10 09  Enable Reporting of sensor with
      |  |  |  |    |  |   |  |   device index of:
      |  |  |  |    |  |   |  |--  8: 1 = enable, 0 = disable
      |  |  |  |    |  |   |-----  9: 1 = enable, 0 = disable
      |  |  |  |    |  |---------10: 1 = enable, 0 = disable
      |  |  |  |    |------------11: 1 = enable, 0 = disable
      |  |  |  |-----------------12: 1 = enable, 0 = disable
      |  |  |--------------------13: 1 = enable, 0 = disable
      |  |-----------------------14: 1 = enable, 0 = disable
      |--------------------------15: 1 = enable, 0 = disable
data[2]
      24 23 22 21   20 19  18 17  Enable Reporting of sensor with
      |  |  |  |    |  |   |  |   device index of:
      |  |  |  |    |  |   |  |-- 16: 1 = enable, 0 = disable
      |  |  |  |    |  |   |-----17: 1 = enable, 0 = disable
      |  |  |  |    |  |---------18: 1 = enable, 0 = disable
      |  |  |  |    |------------19: 1 = enable, 0 = disable
      |  |  |  |-----------------20: 1 = enable, 0 = disable
      |  |  |--------------------21: 1 = enable, 0 = disable
      |  |-----------------------22: 1 = enable, 0 = disable
      |--------------------------23: 1 = enable, 0 = disable
data[3]
      32 31 30 29   28 27  26 25  Enable Reporting of sensor with
      |  |  |  |    |  |   |  |   device index of:
      |  |  |  |    |  |   |  |-- 24: 1 = enable, 0 = disable
      |  |  |  |    |  |   |-----25: 1 = enable, 0 = disable
      |  |  |  |    |  |---------26: 1 = enable, 0 = disable
      |  |  |  |    |------------27: 1 = enable, 0 = disable
```

```
|   |   |   |----------------- 28: 1 = enable, 0 = disable
|   |   |--------------------- 29: 1 = enable, 0 = disable
|   |-------------------------- 30: 1 = enable, 0 = disable
|---------------------------- 31: 1 = enable, 0 = disable
```

Any sensor enabled must have been detected as a 0x22 (DS1822 temperature sensor) or 0x28 (DS18B20 temperature sensor) during DOW enumeration. This can be verified by using the "18: Read DOW Device Information" command.

The return packet will be:

```
type = 0x40 | 19
data_length = 0
```

20: Arbitrary DOW Transaction

The CFA-633 can function as a RS-232 to Dallas 1-Wire bridge. This command allows you to specify arbitrary transactions on the 1-Wire bus. 1-Wire commands follow this basic layout:

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

Please see “■ Appendix A: Connecting a DS2450 1-Wire™ Quad A/D Converter” for an example of using this command.

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

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

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

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

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

The return packet will be:

```
type = 0x40 | 20
data_length = 2 to 16
data[0] = device index (0-31 valid)
data[data_length-2] = Data read from the 1-Wire bus. This is the same
                      as number_of_bytes_to_read from the command.
data[data_length-1] = 1-Wire CRC
```

21: Set Up Live Fan or Temperature Display

You can configure the CFA-633 to automatically update a portion of the LCD with a "live" RPM or temperature reading. Once the display is configured using this command, the CFA-633 will continue to display the live reading on the LCD without host intervention. The live display setup is one of the items stored by the "4: Store Current State As Boot State" command, so you can configure the CFA-633 to immediately display fan speeds or system temperatures as soon as power is applied.

The live display is based on a concept of display slots. There are 8 slots, and each of the 8 slots may be enabled or disabled independently.

Any slot may be requested to display any data that is available. For instance, slot 0 could display temperature sensor 3 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. It is legal to have the display area of one slot overlap the display area of another slot, but senseless. This situation should be avoided in order to have meaningful information displayed.

```
type = 21
valid data_length is 7 or 2 (for turning a slot off)
data[0]: display slot (0-7)
data[1]: type of item to display in this slot
           0 = nothing (data_length then must be 2)
           1 = fan tachometer RPM (data_length then must be 7)
           2 = temperature (data_length then must be 7)
data[2]: index of the sensor to display in this slot:
           0-3 are valid for fans
           0-31 are valid for temperatures (and the temperature
                device must be attached)
data[3]: number of digits
           for a fan: 4 digits (0 to  9999) valid fan speed range
           for a fan: 5 digits (0 to 50000) valid fan speed range
           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
           0-12 valid for a 4-digit fan
           0-11 valid for a 5-digit fan or temperature
data[5]: display row (0-1 valid)
data[6]: pulses_per_revolution or temperature units
           for a fan: pulses per revolution for this fan (1 to 32)
           for a 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".

If the frequency of the tachometer signal is below the detectable range, the speed will be displayed as "SLOW" or "STOP".

Displaying a fan will override the fan power setting to 100% for up to 1/8 of a second every 1/2 second. Please see "■ FAN CONNECTIONS" above for a detailed description.

The return packet will be:

```
type = 0x40 | 21
data_length = 0
```

22: Send command directly to the LCD controller

The LCD controller on the CFA-633 is HD44780 compatible. Generally you would not need low-level access the LCD controller, but there are some arcane functions of the HD44780 that are not exposed by the CFA-633's command set. This command allows you to access the CFA-633's LCD controller directly. Please note that it is quite possible to corrupt the CFA-633's display using this command.

```
type = 22
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 | 22
```
data_length = 0

23: Configure Key Reporting

By default, the CFA-633 reports any key event to the host. This command allows the key events to be enabled or disabled on an individual basis. The key events set to report are one of the items stored by the "4: Store Current State As Boot State" command.

```
#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 = 23
data_length = 2
data[0]: press mask
data[1]: release mask
```

The return packet will be:

```
type = 0x40 | 23
```
data_length = 0

24: Read Keypad, Polled Mode

In some situations, it may be convenient for the host to poll the CFA-633 for key activity. This command allows the host to detect which keys are currently pressed, which keys have been pressed since the last poll, and which keys have been released since the last poll.

This command is independent of the key reporting masks set by command 23--all keys will always be visible to this command. Typically both masks of command 23: Configure Key Reporting would be set to 0 if the host is reading the keypad in polled mode.

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

type = 24
data_length = 0
```

The return packet will be:

```
type = 0x40 | 24
data_length = 3
data[0] = bit mask showing the keys currently pressed
data[1] = bit mask showing the keys that have been pressed since
          the last poll
data[2] = bit mask showing the keys that have been released since
          the last poll
```

25: Set Fan Power Fail-Safe

The CFA-633 can be used as part of an active cooling system. For instance, the fans in a system can be slowed down to reduce noise when a system is idle or when the ambient temperature is low, and sped up when the system is under heavy load or the ambient temperature is high.

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

Varying the fan speeds under host software control gives the ultimate flexibility in system design, but would typically have a fatal flaw: a host software or hardware failure could cause the cooling system to fail. If the fans were set at a slow speed when the host software failed, system components may be damaged due to inadequate cooling.

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

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

type = 25
data_length = 2
data[0] = bit mask of fans set to fail-safe
data[1] = timeout value in 1/8 second ticks:
          1 = 1/8 second
          2 = 1/4 second
          255 = 31 7/8 seconds
```

The return packet will be:

```
type = 0x40 | 25
data_length = 0
```

26: Set Fan Tachometer Glitch Delay

The CFA-633 controls fan speed by using PWM. Using PWM turns the power to a fan on and off quickly to change the average power delivered to the fan. The CFA-633 uses approximately 18Hz for the PWM repetition rate. The fan's tachometer output is only valid if power is applied to the fan. Most fans produce a valid tachometer output very quickly after the fan has been turned back on. But some fans take some amount of time after being turned on before their tachometer output is valid.

This command allows you to set a variable-length delay after the fan has been turned on before the CFA-633 will recognize transitions on the tachometer line. The delay is specified in counts, each count being nominally 552.5µS long (1/100 of one period of the 18Hz PWM repetition rate).

In practice, most fans will not need the delay to be changed from the default length of 1 count. If a fan's tachometer output is not stable when its PWM setting is something other than 100%, then simply increase the delay until the reading is stable. Typically you would start at a delay count of 50 or 100, then reduce it until the problem re-appears, then increase the delay count again slightly from that setting to give some margin.

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

The fan glitch delay is one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 26
data_length = 4
data[0] = delay count of fan 1
data[1] = delay count of fan 2
data[2] = delay count of fan 3
data[3] = delay count of fan 4
```

The return packet will be:

```
type = 0x40 | 25
data_length = 0
```

27: Query Fan Power & Fail-Safe Mask

This command can be used to verify the current fan power and which fans are set to "fail-safe" mode.

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

type = 27
data_length = 0
```

The return packet will be:

```
type = 0x40 | 25
data_length = 5
data[0] = fan 1 power
data[1] = fan 2 power
data[2] = fan 3 power
data[3] = fan 4 power
data[4] = bit mask of fans with fail-safe set
```

28: Set ATX Power Switch Functionality

This command combined with the optional WRPWRY14 cable allows the CFA-633 to replace power and reset switches in an "ATX" compatible system. The ATX Power Switch Functionality is one of the items stored by the "4: Store Current State As Boot State" command.

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

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

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

This configuration can be assured by sending the following command:

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

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

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

This configuration can be assured by sending the following command:

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

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

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

This configuration can be assured by sending the following command:

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

These settings must be saved as the boot state.

There are four functions that can be individually enabled by this command:

The RESET (GPIO[3]) and POWER CONTROL (GPIO[2]) lines on the CFA-633 are normally high-impedance—which means that electrically they appear to be disconnected or floating. When the CFA-633 asserts the RESET or POWER_CONTROL lines, they are momentarily driven high or low (as determined by the RESET_INVERT and POWER_INVERT bits detailed below). To end the power or reset pulse, the CFA-633 changes the lines back to high-impedance.

KEYPAD_RESET: If POWER-ON SENSE (GPIO[1]) is high, holding the green check key for 4 seconds will pulse RESET (GPIO[3]) pin for 1 second. During the 1-second pulse, the CFA-633 will show “RESET”, and then the CFA-633 will reset itself, showing its boot state as if it had just powered on. Once the pulse has finished, the CFA-633 will not respond to any commands until after it has reset the host and itself

KEYPAD_POWER_ON: If POWER-ON SENSE (GPIO[1]) is low, pressing the green check key for 0.25 seconds will pulse POWER CONTROL (GPIO[2]) for the duration specified by

in data[1] or the default of 1 second. During this time, the CFA-633 will show “POWER ON”, then the CFA-633 will reset itself.

KEYPAD_POWER_OFF: If POWER-ON SENSE (GPIO[1]) is high, holding the red “X” key for 4 seconds will pulse POWER CONTROL (GPIO[2]) for the duration specified by in data[1] or the default of 1 second. If the user continues to hold the power key down, then the CFA-633 will continue to drive the line, for a maximum of 5 additional seconds. During this time the CFA-633 will show “POWER OFF”. In order for the system to look “off”, the CFA-633 will then clear its screen. The fans will stop and the backlight will go off if the host drops the +12v supply, which would be expected. The CFA-633 will then “sleep”, waiting for POWER-ON SENSE (GPIO[1]) to go high or the user to press the green check key.

If you have set “LCD_OFF_IF_HOST_IS_OFF” The CFA-633 will then blank its screen to simulate its power being off any time POWER-ON SENSE (GPIO[1]) is low. The CFA-633 will still be active (since it is powered by V_{SB}), monitoring the keypad for a power-on keystroke. If +12v remains active (which would not be expected, since the host is “off”), the fans and backlight will remain on at their previous settings. Once POWER-ON SENSE (GPIO[1]) goes high, the CFA-633 will re-boot as if power had just been applied to it..

```
#define RESET_INVERT           0x02 //Reset pin drives high instead of low
#define POWER_INVERT           0x04 //Power pin drives high instead of low
#define LCD_OFF_IF_HOST_IS_OFF 0x10
#define KEYPAD_RESET           0x20
#define KEYPAD_POWER_ON        0x40
#define KEYPAD_POWER_OFF       0x80

type = 28
data_length = 1 or 2
data[0]: bit mask of enabled functions
data[1]: (optional) length of power on & off pulses in 1/32 second
           1 = 1/32 sec
           2 = 1/16 sec
           16 = 1/2 sec
           255 = 8 sec
```

The return packet will be:

```
type = 0x40 | 28
data_length = 0
```

29: Enable/Disable and Reset the Watchdog

Some high-availability systems use hardware watchdog timers to ensure that a software or hardware failure does not result in an extended system outage. Once the host system has booted, a system monitor program is started. The system monitor program would enable the watchdog timer on the CFA-633. If the system monitor program fails to reset the CFA-633's watchdog timer, the CFA-633 will reset the host system.

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

```
type = 29
data_length = 0
data[0] = enable/timeout

      If timeout is 0, the watchdog is disabled.

      If timeout is 1-255, then this command must be issued again within
      timeout seconds to avoid a watchdog reset.

      To turn the watchdog off once it has been enabled, simply set
      timeout to 0.

      If the command is not re-issued within timeout seconds, then the
      CFA-633 will reset the host (see command 28 for details). Since
      the watchdog is off by default when the CFA-633 powers up, the CFA-633
      will not issue another host reset until the host has once again enabled
      the watchdog.
```

The return packet will be:

```
type = 0x40 | 29
data_length = 0
```

30: Read Reporting & Status

This command can be used to verify the current items configured to report to the host, as well as some other miscellaneous status information.

```
type = 30
data_length = 0
```

The return packet will be:

```
type = 0x40 | 30
data_length = 10
data[0] = fan 1-4 reporting status (as set by command 16)
data[1] = temperatures 1-8 reporting status (as set by command 19)
data[2] = temperatures 9-15 reporting status (as set by command 19)
data[3] = temperatures 16-23 reporting status (as set by command 19)
data[4] = temperatures 24-32 reporting status (as set by command 19)
data[5] = key presses (as set by command 23)
data[6] = key releases (as set by command 23)
data[7] = ATX Power Switch Functionality (as set by command 28), and
          bit 0x08 will be set if the watchdog is active
data[8] = current watchdog counter (as set by command 29)
data[9] = fan RPM glitch delay[0] (as set by command 26)
data[10] = fan RPM glitch delay[1] (as set by command 26)
data[11] = fan RPM glitch delay[2] (as set by command 26)
data[12] = fan RPM glitch delay[3] (as set by command 26)
data[13] = contrast setting (as set by command 13)
data[14] = backlight setting (as set by command 14)
```

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

31: Send Data to LCD

This command allows any length of data to be placed at any position on the LCD. The LCD contents are one of the items stored by the "4: Store Current State As Boot State" command.

```
type = 31
data_length = 3 to 18
data[0]: col = x = 0 to 19
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 | 31
data_length = 0
```

32: reserved (CFA-631 Key Legends)

33: Set Baud Rate

This command will change the CFA-633's baud rate. The CFA-633 will send the acknowledge for this command and then change its baud rate to the new value. The host should send the baud rate command, wait for a positive acknowledge and then switch to the new baud rate itself. The baud rate must be saved by the "4: Store Current State As Boot State" command if you want the CFA-633 to power up at the new baud rate.

The factory default baud rate is 19,200.

```
type = 27
data_length = 1
data[1]: 0 = 19,200 baud
         1 = 115,200 baud
```

The return packet will be:

```
type = 0x40 | 27
data_length = 0
```

34: Set or Set and Configure GPIO pin

The CFA-633 (hardware version v1.4 and up, firmware version 1.9 and up) has five pins that can be used for user-definable general-purpose input / output (GPIO). These pins are also shared with the DOW and ATX functions. Care must be taken in configuring the GPIO if you want use the ATX or DOW at the same time.

The architecture of the CFA-633 allows great flexibility in the configuration of the GPIO pins. They can be set as input or output, they can output constant high or low signals or a variable duty cycle 100Hz PWM signal.

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

The CFA-633 continuously polls the GPIOs as inputs at 32Hz. The present level can be queried by the host software at a lower rate. The CFA-633 also keeps track of whether there were rising or falling edges since the last host query (subject to the resolution of the 32Hz sampling), so the host is not forced to poll quickly in order to detect short events. The algorithm used by the CFA-633 to read the inputs is inherently "bounce free".

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

Pull-up/pull-down resistance values are approximately 5KΩ. Do not exceed current of 20mA per GPIO.

Note: The GPIO pins may also be used for ATX control through header J8 and temperature sensing through the DOW header. By factory default, the GPIO output setting, function and drive mode are set correctly to enable operation of the ATX and DOW functions. The GPIO output setting, function and drive mode must be set to the correct values in order for the ATX and DOW functions to work—improper use of this command can disable the ATX and DOW functions. 633_WinTest may be used to easily check the GPIO configuration or reset the GPIO configuration to the default state so the ATX and DOW functions will work.

The GPIO configuration is one of the items stored by the "4: Store Current State As Boot State" command.

```
type: 34
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 (default is ATX Host Power Sense)
  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] = J9, Pin 2 (default is DOW I/O--always has 1KΩ
                           hardware pull-up)
  5-255: reserved

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

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

The return packet will be:

```
type = 0x40 | 34
data_length = 0
```

35: Read GPIO pin levels and configuration state

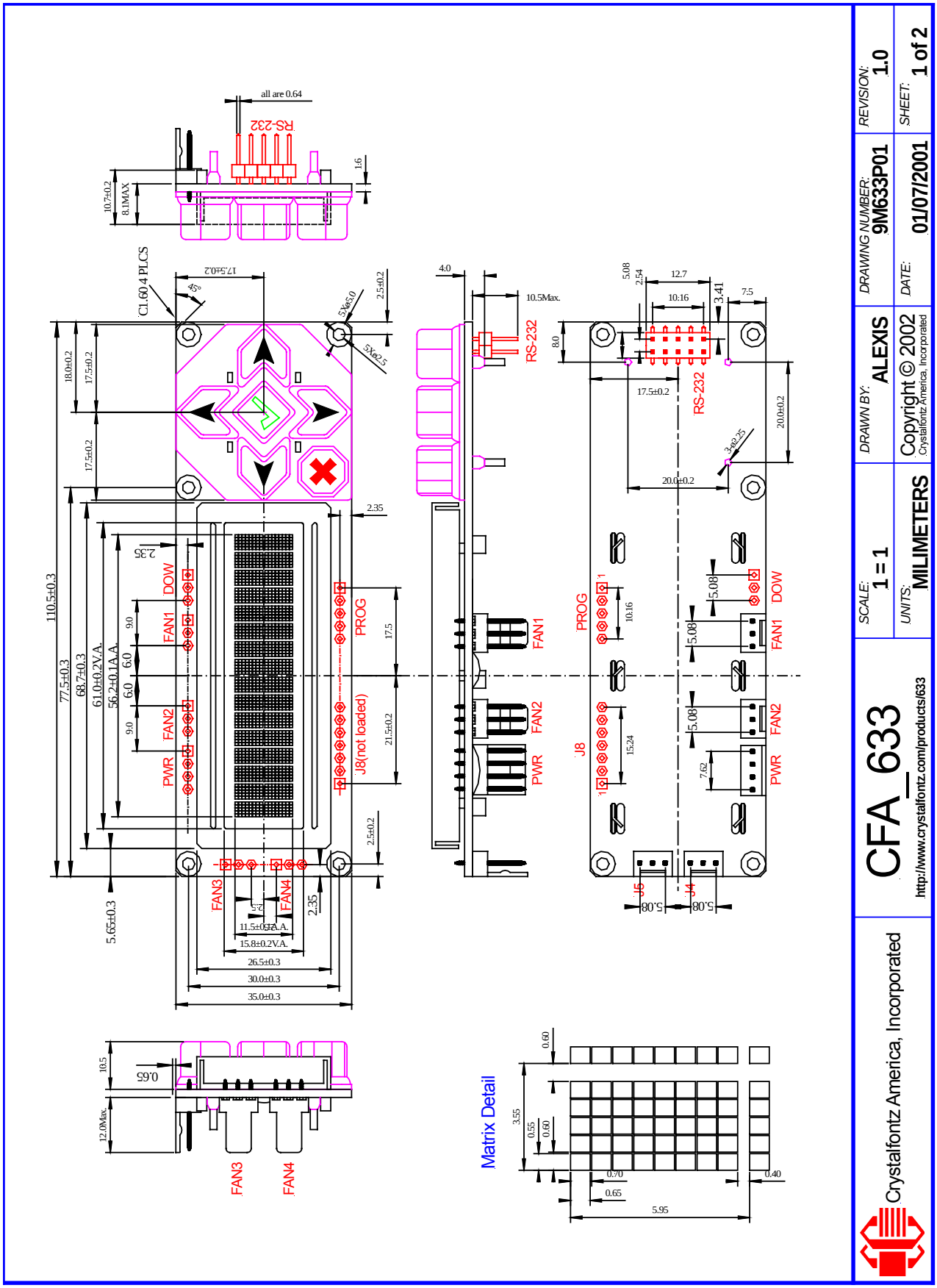
Please see "34: Set or Set and Configure GPIO pin", above for details on the GPIO architecture.

```
type: 35
data_length: 1

data[0]: index of GPIO to query
  0 = GPIO[0] = J8, Pin 7
  1 = GPIO[1] = J8, Pin 6 (default is ATX Host Power Sense)
  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] = J9, Pin 2 (default is DOW I/O--always has 1KΩ
                           hardware pull-up)
  5-255: reserved

  Please note: Future versions of this command on future
  hardware models may accept additional values for data[0],
  which would return the status of future additional GPIO
  pins

returns:
  data[0] = index of GPIO to read
  data[1] = Pin state & changes since last poll
    ---- -RFS Enable Reporting of this Fan's Tach Input
    |||| ||||-- S = state at the last reading
    |||| |||--- F = at least one falling edge has
    |||| ||         been detected since the last poll
    |||| ||---- R = at least one rising edge has
    |||| |          been detected since the last poll
    |||| |----- reserved
```

```
     (This reading is the actual pin state, which may
      or may not agree with the pin setting, depending
      on drive mode and the load presented by external
      circuitry. The pins are polled at approximately
      32Hz asynchronously with respect to this command.
      Transients that happen between polls will not be
      detected.)
data[2] = Requested Pin level/PWM level
  0-100: Output duty cycle percentage
    (This value is the requested PWM duty cycle. The
     actual pin may or may not be toggling in agreement
     with this value, depending on the drive mode and
     the load presented by external circuitry)
data[3] = Pin function select and drive mode
  ---- FDDD
  |||| ||||-- DDD = Drive Mode
  |||| |      ========================================================
  |||| |      000: 1=Fast, Strong Drive Up, 0=Resistive Pull Down
  |||| |      001: 1=Fast, Strong Drive Up, 0=Fast, Strong Drive Down
  |||| |      010: Hi-Z, use for input
  |||| |      011: 1=Resistive Pull Up,     0=Fast, Strong Drive Down
  |||| |      100: 1=Slow, Strong Drive Up, 0=Hi-Z
  |||| |      101: 1=Slow, Strong Drive Up, 0=Slow, Strong Drive Down
  |||| |      110: reserved
  |||| |      111: 1=Hi-Z,                  0=Slow, Strong Drive Down
  |||| |
  |||| |----- F = Function
  ||||         ========================================================
  ||||         0: Port unused for GPIO. It will take on the default
  ||||            function such as ATX, DOW or unused. The user is
  ||||            responsible for setting the drive to the correct
  ||||            value in order for the default function to work
  ||||            correctly.
  ||||         1: Port used for GPIO under user control. The user is
  ||||            responsible for setting the drive to the correct
  ||||            value in order for the desired GPIO mode to work
  ||||            correctly.
  ||||------- reserved, will return 0
```

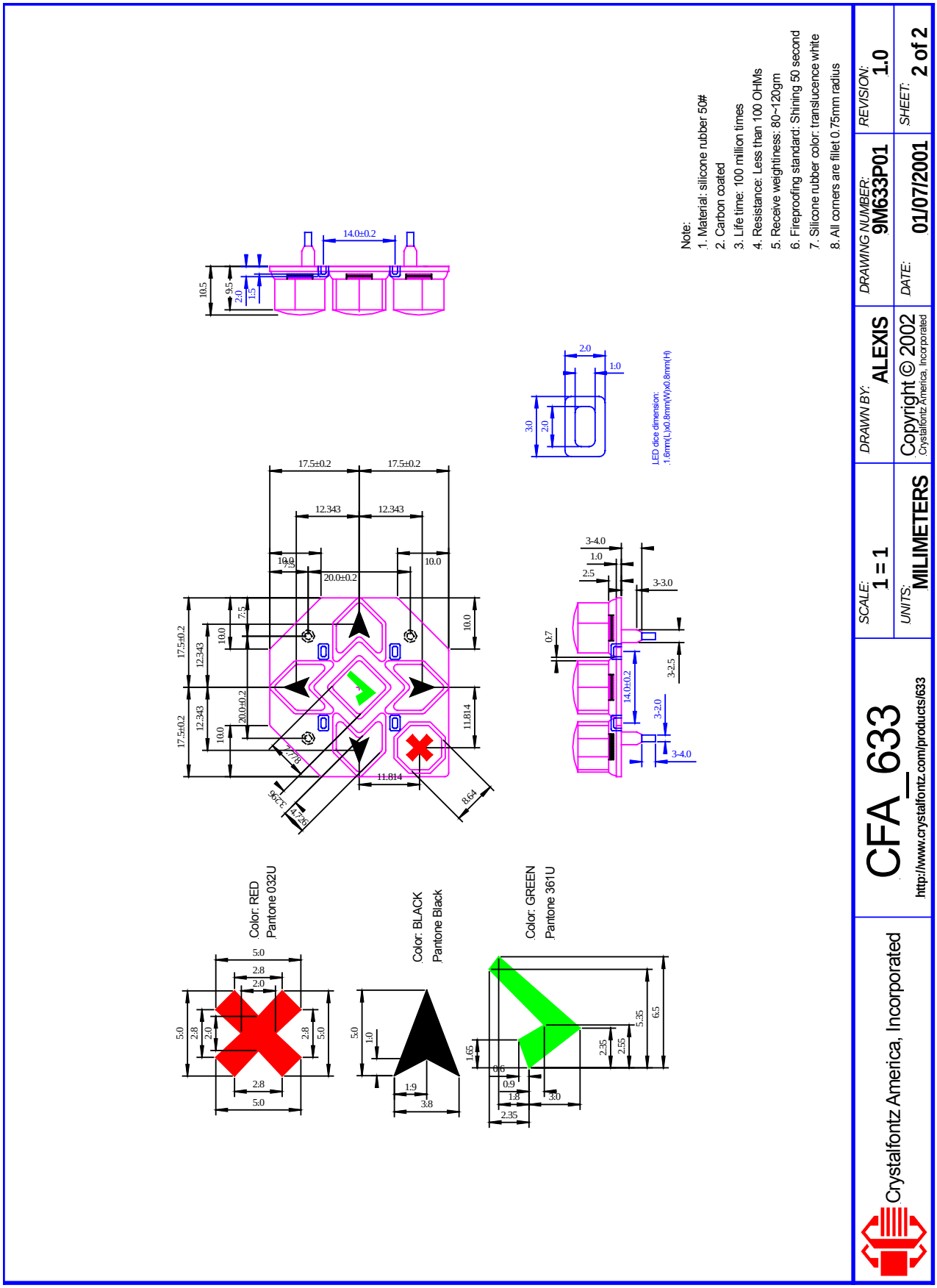
■ CHARACTER GENERATOR ROM (CGROM)

■ MODULE OUTLINE DRAWING

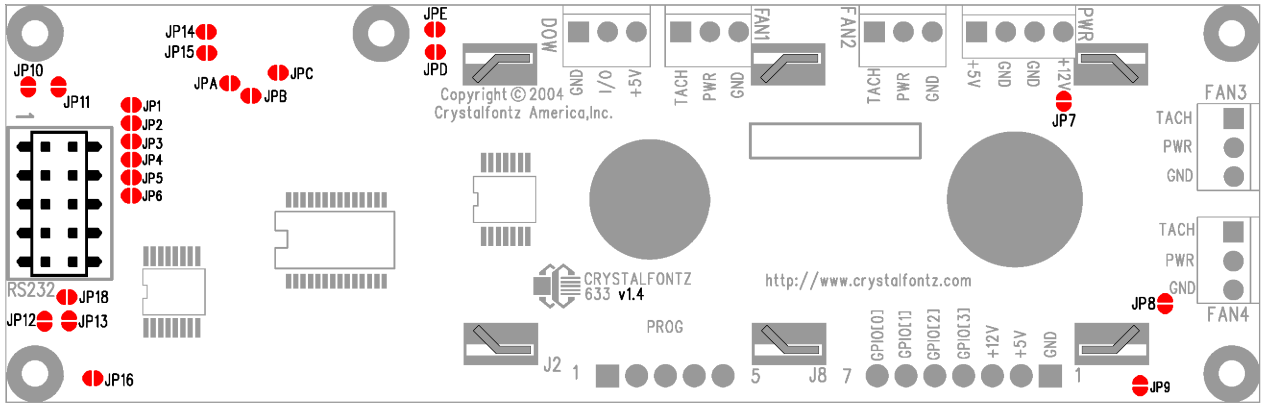


REVISION: 1.0			SHEET: 1 of 2
DRAWING NUMBER: 9M633P01			DATE: 01/07/2001
DRAWN BY: ALEXIS	Copyright © 2002 Crystalfontz America, Incorporated		
SCALE: 1 = 1	UNITS: MILLIMETERS		
CFA_633	http://www.crystalfontz.com/products/633		
Crystalfontz America, Incorporated			

■ KEYPAD OUTLINE DRAWING



CFA-633 Hardware v1.4 Jumper Locations and Functions



JP1	**open**	J1 (RS232), Pin 10 is open
	closed	LCD Tx/Host Rx to J1(RS232), Pin 10
JP2	open	J1(RS232), Pin 2 is open (see also JP3 and JP11)
	closed	LCD Tx/Host Rx to J1(RS232), Pin 2
JP3	**open**	J1 (RS232), Pin 2 is open (see also JP2 and JP11)
	closed	LCD Rx/Host Tx to J1(RS232), Pin 2
JP4	open	J1(RS232), Pin 3 is open (see also JP5 and JP12)
	closed	LCD Rx/Host Tx to J1(RS232), Pin 3
JP5	**open**	J1 (RS232), Pin 3 is open (see also JP4 and JP12)
	closed	Ground to J1(RS232), Pin 3
JP6	open	J1(RS232), Pin 5 is open
	closed	Ground to J1(RS232), Pin 5
JP7	open	+12v fans is isolated from +12v backlight
	closed	+12v fans is connected to +12v backlight
JP8	open	+5v pin from PWR connector is open (see also JP9)
	closed	+5v pin from PWR connector supplies power to module
JP9	**open**	+5v pin from PWR connector is open (see also JP8)
	closed	+5v pin from PWR connector connected to Pin 6 of J6 (GPIO[1] / POWER-ON SENSE)
JP10	**open**	J1 (RS232), Pin 1 is open
	closed	+12v backlight to J1 (RS232), Pin 1
JP11	**open**	J1 (RS232), Pin 2 is open (see also JP2 and JP3)
	closed	Logic-level (0-5v) LCD Tx/Host Rx to J1(RS232), Pin 2
JP12	**open**	J1 (RS232), Pin 3 is open (see also JP4 and JP5)
	closed	Logic-level (0-5v) LCD Rx/Host Tx to J1(RS232), Pin 3
JP13	**open**	J1 (RS232), Pin 4 is open
	closed	+5v to J1 (RS232), Pin 4
JP14, JP15, JPA, JPB, JPC, JPD, JPE: Factory build options, do not change		

Crystalfontz America, Incorporated	CFA_633 http://www.crystalfontz.com/products/633	SCALE: 1 = 1	DRAWN BY: ALEXIS	DRAWING NUMBER: 9M633P02	REVISION: v1.4
		UNITS: MILIMETERS	Copyright © 2004 Crystalfontz America, Incorporated	DATE: 06/03/2004	SHEET: 1 of 1

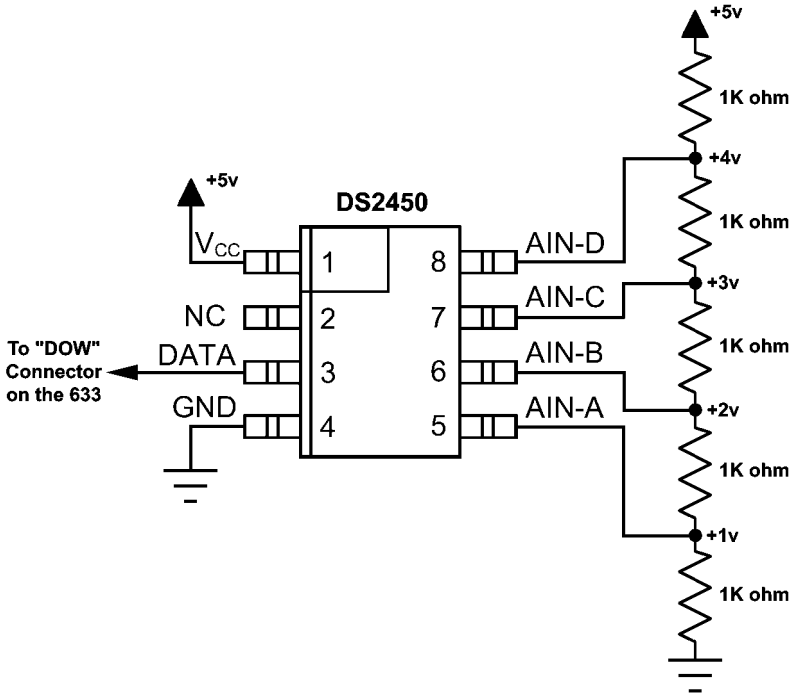
■ Appendix A: Connecting a DS2450 1-Wire™ Quad A/D Converter

This appendix describes a simple test circuit that demonstrates connecting a Dallas Semiconductor DS2450 4-channel ADC to the CFA-633's "DOW" (Dallas One Wire) connector. It also gives a sample command sequence to initialize and then read the ADC.

Up to 32 DOW devices can be connected to the CFA-633. In this example the DS2450 appears at device index 0. Your software should query the connected devices using command 18: Read DOW Device Information to verify the locations and types of DOW devices connected in your application.

Please refer to the DS2450 data sheet, and the description for command 20: Arbitrary DOW Transaction more information.

TEST CIRCUIT SCHEMATIC



Open the 633_WinTest Packet Debugger Window.

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)

■ Appendix B: Connecting a DS1963S SHA iButton

This appendix describes connecting a Dallas Semiconductor DS1963S Monetary iButton with SHA1 Challenge Response Algorithm and 4KB of non-volatile RAM to the CFA-633's DOW (Dallas One Wire) connector. It also gives a sample command sequence to read and write the DS1963S's scratch memory.

The DS1963S can be used as a secure dongle to protect your system's application software from being copied. Even if the communication channel is compromised or the host is not authentic, the SHA algorithm ensures that the data is still secure. Please see the following Dallas/Maxim application notes and white papers for more information:

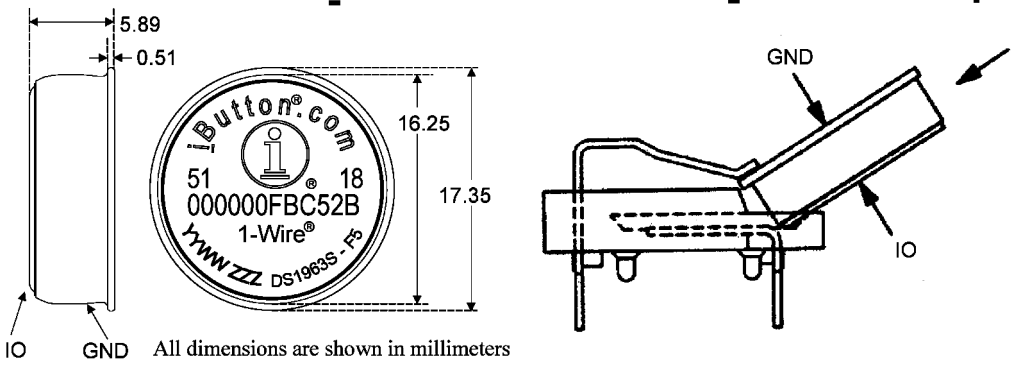
White Paper 1: SHA Devices Used in Small Cash Systems
White Paper 2: Using the 1-Wire Public-Domain Kit
White Paper 3: Why are 1-Wire SHA-1 Devices Secure?
White Paper 4: Glossary of 1-Wire SHA-1 Terms
White Paper 8: 1-Wire SHA-1 Overview
App Note 150: Small Message Encryption using SHA Devices
App Note 152: SHA iButton Secrets and Challenges
App Note 154: Passwords in SHA Authentication
App Note 156: DS1963S SHA 1-Wire API Users Guide
App Note 157: SHA iButton API Overview
App Note 190: Challenge and Response with 1-Wire SHA devices

Up to 32 DOW devices can be connected to the CFA-633. In this example the DS1963S appears at device index 0. Your software should query the connected devices using command 18: Read DOW Device Information to verify the locations and types of DOW devices connected in your application.

Please refer to the DS1963S data sheet, and the description for command 20: Arbitrary DOW Transaction more information.

To connect the DS1963S to the CFA-633, simply make one connection between the DS1963S's "GND" terminal and the CFA-633 DOW connector's GND pin, and a second connection between the DS1963S's "IO" pin and the CFA-633 DOW connector's I/O pin. By using a DS9094 iButton Clip, the connection is quite easy.

DS1963S SHA iButton **DS9094 iButton™ Clip**



To demonstrate reading and writing the scratch memory on DS1963S, open the 633_WinTest Packet Debugger Window and use it to experiment with the following commands.

To use the full power of the DS1963S, a program based on the Dallas/Maxim application notes listed above would be needed. The challenge/response sequence would be quite unwieldy to demonstrate using the 633_WinTest Packet Debugger.

First, we will read the address of the DS1963S as detected by the CFA-633 at boot. Since we know that there is only one device connected, we can get away with only querying index 0. In a production situation, you should query all 32 indices do get a complete picture of the devices available on the DOW bus.

```
Command:
  18 = Read DOW Device Information
Data sent:
  \000
Data received:
  C=82(d=0):18,CC,D2,19;00,00,00,9E
```

The first byte returned is the Family Code of the Dallas One Wire / iButton device. 0x18 indicates that this device is a DS1963. There is a list of the possible Dallas One Wire / iButton device family codes available in App Note 155: 1-Wire Software Resource Guide on the Dallas / Maxim web site.

Quoted from the DS1963S data sheet:

> *Erase Scratchpad [C3h]*
> *The purpose of this command is to clear the HIDE flag and to wipe out data that might have been left in the scratchpad from a previous operation. After having issued the command code the bus master transmits a target address, as with the write scratchpad command, but no data. Next the whole scratchpad will be automatically filled with FFh bytes, regardless of the target address. This process takes approximately 32 µs during which the master reads 1's. After this the master reads a pattern of alternating 0's and 1's indicating that the command has completed. The*

master must read at least 8 bits of this alternating pattern. Otherwise the device might not properly respond to a subsequent Reset Pulse.

```
Command:
  20 = Arbitrary DOW transaction
Data sent:
  \000\014\xC3\000\000
Data received:
  C=84(d=0):FF,AA,AA,AA,AA,AA,AA,AA,AA,AA,AA,AA,AA,9F
```

The "AA" bytes read are the "pattern of alternating 0's and 1's indicating that the command has completed".

Quoted from the DS1963S data sheet:

Read Scratchpad Command [AAh]
HIDE = 0:
The Read Scratchpad command allows verifying the target address, ending offset and the integrity of the scratchpad data. After issuing the command code the master begins reading. The first 2 bytes will be the target address. The next byte will be the ending offset/data status byte (E/S) followed by the scratchpad data beginning at the byte offset (T4: T0). The master may read data until the end of the scratchpad after which it will receive the inverted CRC generated by the DS1963S. If the master continues reading after the CRC all data will be logic 1's.

```
Command:
  20 = Arbitrary DOW transaction
Data sent:
  \000\014\xAA
Data received:
  C=84(d=0):00,00,1F,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,07
```

Since we just did an "Erase Scratchpad" as the previous command, the "Read Scratchpad" returns 0xFF bytes as expected.

Quoted from the DS1963S data sheet:

Write Scratchpad Command [0Fh]
HIDE = 0, Target Address range 0000h to 01FFh only
After issuing the write scratchpad command, the master must first provide the 2–byte target address, followed by the data to be written to the scratchpad. The data will be written to the scratchpad starting at the byte offset (T4:T0). The ending offset (E4: E0) will be the byte offset at which the master stops writing data. Only full data bytes are accepted. If the last data byte is incomplete its content will be ignored and the partial byte flag PF will be set.

When executing the Write Scratchpad command the CRC generator inside the DS1963S (see Figure 12) calculates a CRC of the entire data stream, starting at the

command code and ending at the last data byte sent by the master. This CRC is generated using the CRC16 polynomial by first clearing the CRC generator and then shifting in the command code (0FH) of the Write Scratchpad command, the Target Addresses TA1 and TA2 as supplied by the master and all the data bytes. The master may end the Write Scratchpad command at any time. However, if the ending offset is 11111b, the master may send 16 read time slots and will receive the CRC generated by the DS1963S.

We will now write 10 bytes of identifiable test data {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA} to the scratch pad in location 0:0

```
Command:
  20 = Arbitrary DOW transaction
Data sent:
  \000\000\x0F\x00\x00\x11\x22\x33\x44\x55\x66\x77\x88\x99\xAA
Data received:
  C=84(d=0):00
```

Now we will use the Read Scratchpad Command [AAh] to read back the data.

```
Command:
  20 = Arbitrary DOW transaction
Data sent:
  \000\013\xAA
Data received:
  C=84(d=0):00,00,09,11,22,33,44,55,66,77,88,99,AA,1E
```

Now write 10 bytes of identifiable test data {0x12, 0x23, 0x34, 0x45, 0x56, 0x67, 0x78, 0x89, 0x9A, 0xAB} to the scratch pad in location 0:0x0A

```
Command:
  20 = Arbitrary DOW transaction
Data sent:
  \000\000\x0F\x0A\x00\x12\x23\x34\x45\x56\x67\x78\x89\x9A\xAB
Data received:
  C=84(d=0):00
```

Use the Read Scratchpad Command [AAh] to read back the data.

```
Command:
  20 = Arbitrary DOW transaction
Data sent:
  \000\013\xAA
Data received:
  C=84(d=0):00,02,09,12,23,34,45,56,67,78,89,9A,AB,62
```

Reading and writing the scratch pad is the first necessary step required in communicating with the DS1863S. In order to fully use the DS1963S for in dongle application that securely

protects your software from copying, you would want to become familiar with the SHA algorithm as it applies to the SHA iButton by studying the application notes listed above, and then create a software application that would implement the secure challenge/response protocol as outlined in the application notes.

■ Appendix C: Calculating the CRC

This appendix gives three sample algorithms that will calculate the CRC of a CFA-633 packet. The CRC used in the CFA-633 is the same one that is used in IrDA, which came from PPP, which to at least some extent 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 diverging 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.

ALGORITHIM 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.
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 algorithim 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);
  }
```

ALGORITHIM 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.

```
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 bytes 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);
  }
```

ALGORITHIM 3: “PIC ASSEMBLY” BIT SHIFT IMPLEMENTATION

This routine was graciously donated by one of our customers.

```
;=====================================================================
; Crystalfontz CFA-633 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
```

ALGORITHIM 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 CFA-633 packet) data. This routine was adapted from the C table implementation. The complete project can be found in our forums.

```
'This program is brutally blunt. Just like VB. No apologies.
'Written by Crystalfontz America, Inc. 2004 http://www.crystalfontz.com
'Free code, not copyright copyleft or anything else.
'Some visual basic concepts taken from:
```

```
'http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=21434&lngWId=1
'most of the algorithm is from functions in 631_WinTest:
'http://www.crystalfontz.com/products/631/631_WinTest.zip
'Full zip of the project is available in our forum:
'http://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. Nice language. Fast processors, lots of memory, big disks,
'and we fill them up with this . . this . . this . . STUFF.
Sub Initialize_CRC_Lookup_Table()
  crcLookupTable(0).Lo = &H0
  crcLookupTable(0).Hi = &H0
  . . .
'For purposes of brevity in this data sheet, I have removed 251 entries of this table, the
'full source is available in our forum:
'http://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
```

ALGORITHIM 5: "JAVA" TABLE IMPLEMENTATION

This code was posted in our forum by "**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);
    }
  }
```

ALGORITHIM 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 = $type . $length . $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);
```