App Note AN-2025

Using the Finisar GBIC I²C Test/Diagnostics Port

INTRODUCTION

Finisar's new line of optical GBIC modules incorporates the real-time optical link control system that Finisar pioneered in its previous generation of fiber optic transceivers. Finisar has extended the capabilities of the module definition (4) 2-wire serial interface called for in the GBIC specification rev. 5.1 (see ref. 1). In addition to accessing the 96 bytes of information called for by the GBIC specification, the host system can monitor the transmitted optical power, received optical power, laser diode drive current, and module temperature.

All of the above quantities are measured and stored in each GBIC's memory at the time of manufacture. This data can be read by the host and compared to real-time values to look for variations or anomalies over time. With all of this information, it is possible to detect signs of laser aging or irregularity and perform preventative maintenance long before a link is disrupted. If the host is a network switch or hub, this preventative maintenance and early alarm capability can improve network uptime significantly.

This application note describes how to communicate with Finisar optical GBICs and take full advantage of their test and diagnostics capability. Relevant part numbers are FTR-1319, FTR-1320, FTR-8519, and FTR-8520.

SERIAL COMMUNICATION PORT DESCRIPTION

The GBIC specification calls for a "2-wire serial CMOS E²PROM protocol defined for the ATMEL AT24C01A/02/04 family of components." The two-wire interface used by ATMEL is essentially an I²C bus (see ref. 2), which is what Finisar has implemented. As per the GBIC specification, pin 5, or MOD_DEF(1), is the clock line, SCL, and pin 6, or MOD_DEF(2), is the data line, SDA.

The I²C bus is a master/slave arrangement. The entity wishing to initiate communication transmits a special START condition on the bus followed by the address of the device with which it would like to communicate. The device initiating communication becomes the bus master, and the addressed device becomes the slave. I²C supports multiple masters on a single bus through an arbitration process. It also supports multiple slaves with

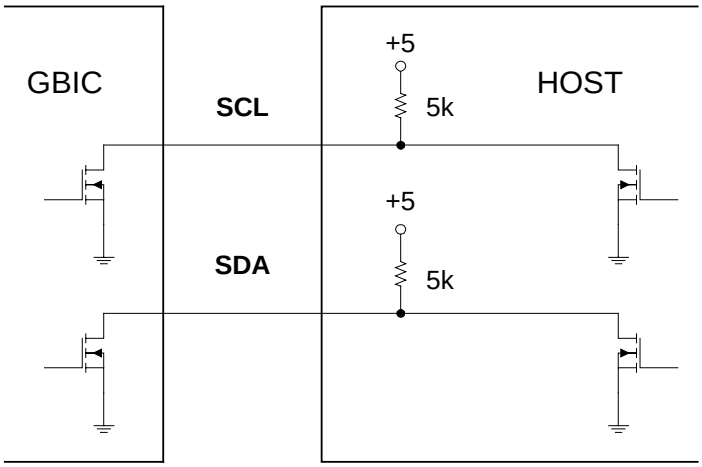
unique addresses. In the case of Finisar GBICs, they will always function as slaves, with the system host acting as the bus master. Since the GBIC specification requires that all GBICs have the same 7 bit I²C address (1010000, see below), only one configuration is possible. Each bus has one master, the host, and one slave with address 1010000.

I²C SPECIFICATIONS

Bus Hardware

Both bus lines are bi-directional, so all outputs must be open collector/open drain. Devices can actively pull bus lines low, but they must release them high. The bus lines, therefore, need to be pulled up (5k typical). Finisar GBIC modules do not have internal pull-ups, so the host must supply pull-up resistors.

Figure 1. I²C Bus Hardware.



Data Format

All data transmitted over the bus is sent one 8 bit byte at a time. All bytes are transmitted most significant bit first. This includes both data and address bytes. Transmission of each byte is always followed by an acknowledge as outlined below.

Device Addressing

Finisar GBICs implement the I²C 7 bit addressing format called for in the GBIC specification. The most significant 7 bits hold address information, and bit 0, the least significant bit (transmitted last) is a read/write bit. If bit 0 is low (write), the host will write the next byte to the GBIC. If it is high (read), the host will read the next byte from the GBIC.

The ATMEL AT24C01A/02/04 family of components have bits 7 - 4 fixed to 1010 (see ref. 3). Bits 3 - 1 are variable, so up to 8 devices may be addressed, however, the GBIC specification, at this time, requires that they be set to 0. Every GBIC thus has the address 0b1010000X, where X is the read/write bit.

Initiating/Terminating Communication

Changes on the data line (SDA), while the clock line (SCL) is high are invalid except in two special cases: the START and STOP conditions. A START condition consists of a falling edge on SDA while SCL is high. Conversely, a STOP condition consists of a rising edge (Figure 2). During normal data transmission, SDA must remain stable while SCL is high. Setup and hold times apply (Figure 3). The GBIC will interpret any change on SDA while SCL is high as either a STOP or a START condition.

Figure 2. I^2C START and STOP Conditions.

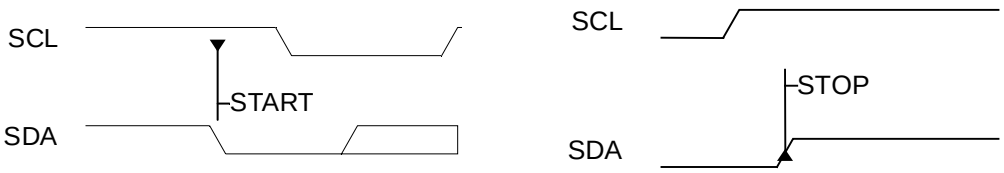
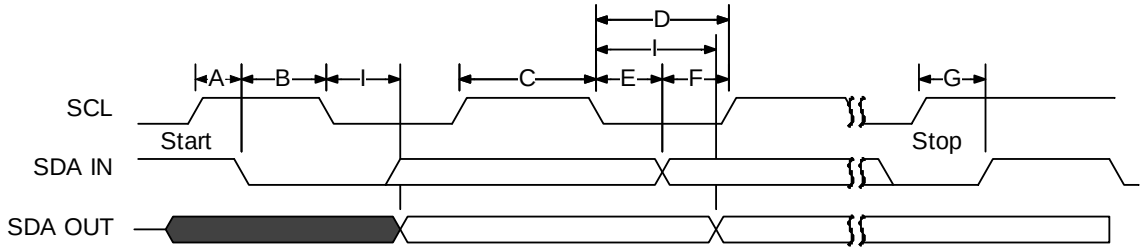


Figure 3. SDA, SCL Timing Requirements.

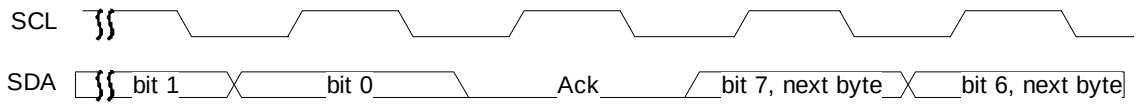


A: START setup time min. 4.7µsec
B: START hold time min. 4.0µsec
C: Clock high time min 4.0µsec
D: Clock low time min 4.7µsec
E: Data input hold time min 0 sec
F: Data input setup time min 250nsec
G: STOP condition setup time min 4.7µsec
I: Output valid from clock min 4.7µsec

Data Acknowledgement

The host, as master, always controls SCL, even when it is receiving data. For each byte of data transmitted, the host cycles the clock line 9 times. The first 8 cycles correspond to 8 data bits. The GBIC clocks data out on each falling edge, and it clocks data in on each rising edge depending on whether it is transmitting or receiving data. The 9th clock cycle is for either a slave or master receiver to acknowledge successful receipt of 8 bits. Prior to the ninth clock cycle, the transmitter of data (either master or slave), releases SDA high. On the ninth cycle the receiver (either master or slave) pulls SDA low to signal acknowledgment. Data transmission then continues on the next clock cycle (Figure 4).

Figure 4. Acknowledge Timing.



I²C Bus Speed

I²C has a feature allowing communication between a fast transmitter and a slow receiver. If the receiver needs time to process data it just received, it may, after receipt of a byte and acknowledgment, hold SCL low to force the transmitter into a wait state. Finisar GBICs implement this feature. Several commands that can be sent to the GBIC require some processing time, and the GBIC will hold SCL low while processing occurs. Hosts must thus monitor SCL every time they release it high to confirm that the GBIC is not holding the line low indicating a wait state. Hosts not implementing this feature (driving the bus blind) must run below 8kHz to allow for all GBIC processing delays. If the host does implement the I²C clock stretching feature, the maximum clock rate is 100kHz. To avoid tying up the GBIC processor indefinitely, an internal timer will time out and the GBIC will abort all data transfers if the clock rate is below 10Hz.

FINISAR GBIC COMMANDS

Below is a complete list of commands available to the host system. Commands 0–95 are simply the data addresses of bytes 0–95 of the serial identification information mandated by the GBIC specification rev. 5.1. The host can access this information by performing a "random read" as defined for the ATMEL AT24C01A/02/04 family of components. Note that Finisar optical GBICs currently do not support a "sequential read." Bytes must be read one at a time. The entire sequence is shown in Figure 5. Table 1 shows additional Finisar GBIC specific information stored in EEPROM.

Table 1. Finisar Test/Diagnostic System EEPROM Data.

EEPROM Address (Decimal)	*Stored Data Name*	*Description*
96	VERSION	Software version. Format: high nibble.low nibble.
97	TX_CAL	Laser output power calibration constant – high byte. (Long wavelength only)
98	TX_CAL	Laser output power calibration constant – low byte. (Long wavelength only)
99	TX_LOCK	Transmitter output power at manufacture – high byte. (Long wavelength only)
100	TX_LOCK	Transmitter output power at manufacture – low byte. (Long wavelength only)
101	RX_CAL	Received power calibration constant – high byte.
102	RX_CAL	Received power calibration constant – low byte.
103	LDI_CAL	Laser diode current at calibration point – high byte.
104	LDI_CAL	Laser diode current at calibration point – low byte.
105	LDI_LOCK	Laser diode operating current at manufacture – high byte.

EEPROM Address (Decimal)	*Stored Data Name*	*Description*
106	LDI_LOCK	Laser diode operating current at manufacture – low byte.
107	TEMP_LOCK	Operating temperature at manufacture – high byte.
108	TEMP_LOCK	Operating temperature at manufacture – low byte.
109	PREBIAS_CAL	Laser diode bias setting at calibration point. Integer.
110	PREBIAS_LOCK	Laser diode bias setting at operating point. Integer.
111	CAL_PWR	Optical power calibration value (-dBm). Optical power = -CAL_PWR (dBm)
112	OFC_STATUS	255 = off, 254 = OFC auto-sense high speed, 252 = OFC autosense low speed
113	AD1	AD1 A/D reference #1 at manufacture – high byte.
114	AD1	AD1 A/D reference #1 at manufacture – low byte.
115	AD2	AD2 A/D reference #2 at manufacture – high byte.
116	AD2	AD2 A/D reference #2 at manufacture – low byte.
117	AD3	AD3 A/D reference #3 at manufacture – high byte.
118	AD3	AD3 A/D reference #3 at manufacture – low byte.
119	LDI_RES_INT	Laser current measurement resistor integer. Format: LDI_RES_INT.LDI_RES_FRACT.
120	LDI_RES_FRACT	Laser current measurement resistor fraction. Format: LDI_RES_INT.LDI_RES_FRACT.

In addition to allowing access to data stored in EEPROM, Finisar GBICs have a command set that allows the host access to various system control functions as well as real time data such as module temperature. These commands mimic data addresses as outlined above. For example, the command, 140 (GET_TEMP), instructs the GBIC to transmit its current temperature over the I^2C bus. From the host's perspective, the data transaction proceeds just as if it were reading a two byte chunk of data from a virtual EEPROM at address 140. See Figure 6 for details.

Table 2. Commands to Access 10 Bit a/d Values.

Commands to read 10 bit ad_values (16 bit data with bits 15-10 = 0)
2 byte read described in Figure 6.

Command Number (Decimal)	*Name*	*Description*
140	GET_TEMP	Returns current operating temperature 10 bit ad value
141	GET_CURRENT	Returns laser diode bias current 10 bit ad value
142	GET_RX_PWR	Returns received optical power 10 bit ad value
143	GET_TX_PWR	Returns transmitted optical output power 10 bit ad value (Long wavelength only)
144	GET_AD2	Returns AD2 A/D reference # 2 10 bit ad value
145	GET_AD3	Returns AD3 A/D reference # 3 10 bit ad value
146	GET_AD1	Returns AD1 A/D reference # 1 10 bit ad value
147	GET_CAL1_HI	Returns CAL1 A/D calibration constant #1 high 16 bits (32 bit floating point)
148	GET_CAL1_LO	Returns CAL1 A/D calibration constant #1 low 16 bits (32 bit floating point)
149	GET_CAL2_HI	Returns CAL2 A/D calibration constant #2 high 16 bits (32 bit floating point)
150	GET_CAL2_LO	Returns CAL2 A/D calibration constant #2 low 16 bits (32 bit floating point)
151	GET_TEMP1_HI	Returns temperature sensor calibration value #1 high 16 bits (32 bit floating point)
152	GET_TEMP1_LO	Returns temperature sensor calibration value #1 low 16 bits (32 bit floating point)
153	GET_TEMP2_HI	Returns temperature sensor calibration value #2 high 16 bits (32 bit floating point)
154	GET_TEMP2_LO	Returns temperature sensor calibration value #2 low 16 bits (32 bit floating point)

Table 3. System Control Commands.

Single command write. Protocol described in Figure 7.

Command Number (Decimal)	*Name*	*Description*
160	RUN_AT_LOCK	Run the laser diode at its lock point (operating point).
161	RUN_AT_CAL	Run the laser diode at its calibration point.
162	RESET	Processor power on reset (~1sec delay)

Table 4. System Status Commands.

One byte read identical to the protocol described in Figure 5.

Command Number (Decimal)	*Name*	*Description*
203	GET_TX_STATUS	Returns system status byte. Bit 0 = TX_FAULT (1 = asserted), Bit 5 = OFC_ENABLE* (1 = OFC on), Bit 6 = OFC_SPEED (1 = fast, 0 = slow)

*OFC = Fibre Channel Open Fiber Control. Finisar has developed an “autosensing” OFC protocol that can interoperate with either OFC or non-OFC fiber optic modules. If a Finisar GBIC with the autosensing protocol enabled senses that its counterpart is running OFC then it runs OFC. Otherwise it operates as a standard non-OFC module.

Figure 5. GBIC One Byte Data Read.

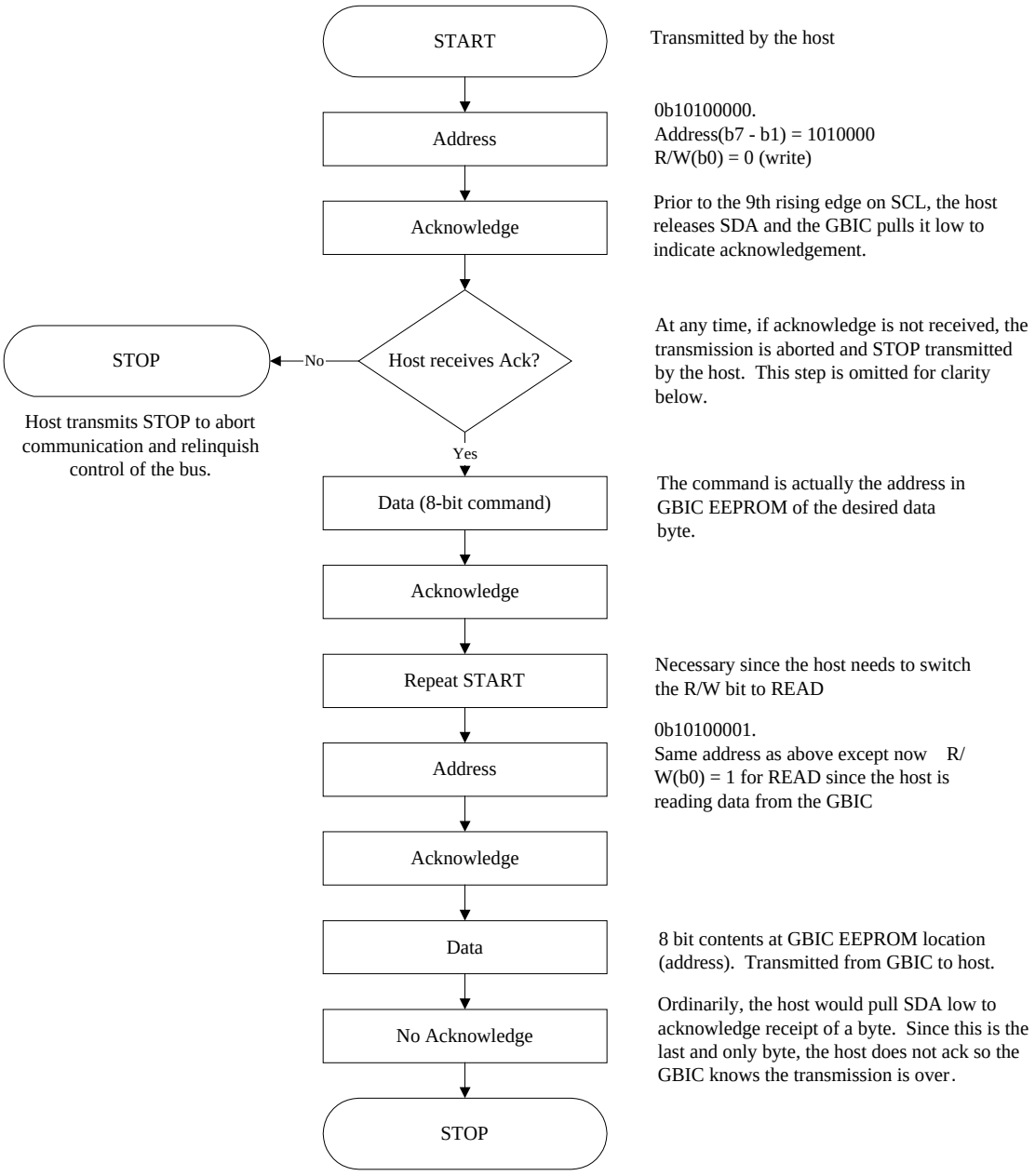
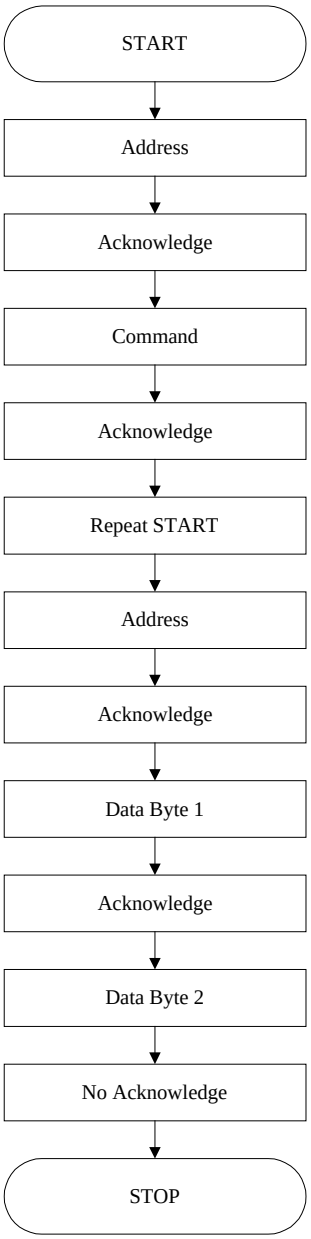


Figure 6. GBIC Two Byte Read.



Transmitted by the host

0b10100000.
Address(b7 - b1) = 1010000
R/W(b0) = 0 (write)

At any time, if acknowledge is not received, the transmission is aborted and STOP transmitted by the host. This step is omitted for clarity below.

Prior to the 9th rising edge on SCL, the host releases SDA and the GBIC pulls it low to indicate acknowledgement.

8-bit command to read a 2 byte a/d value (see Table 3)

Necessary since the host needs to switch the R/W bit to READ

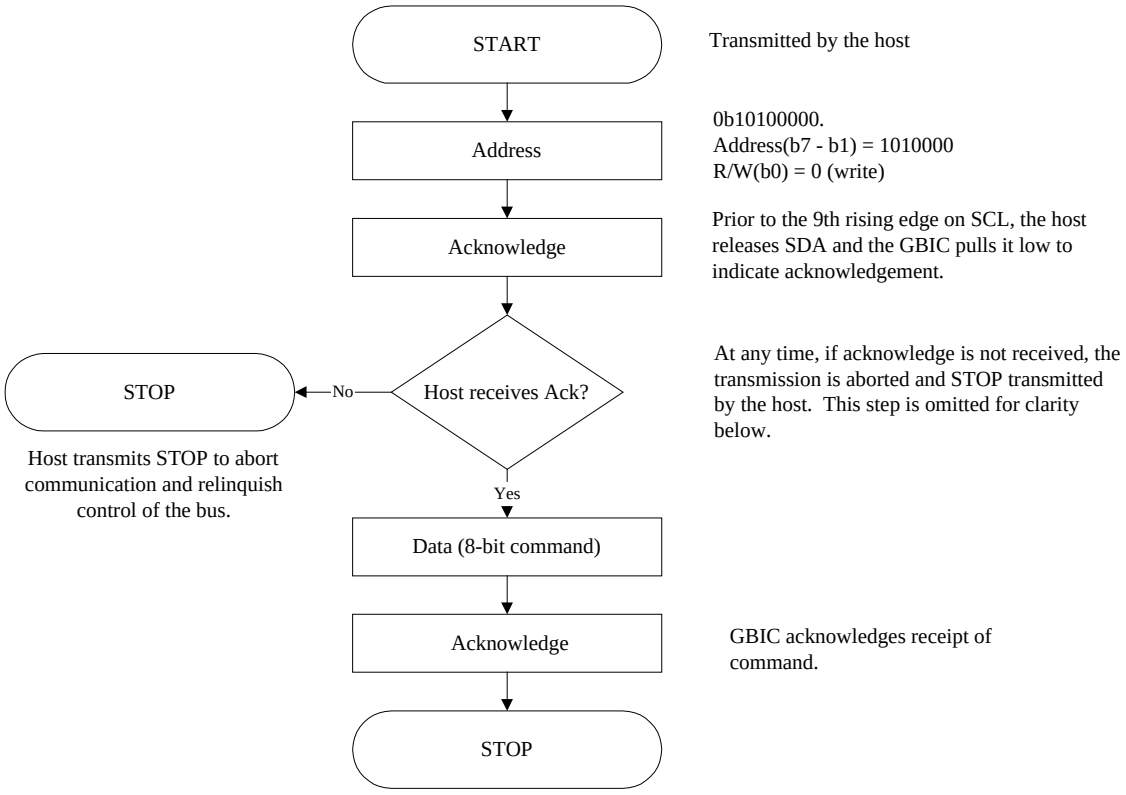
0b10100001.
Same address as above except now R/W(b0) = 1 for READ since the host is reading data from the GBIC

Most significant byte first for two byte quantities

Least signifiicant byte last

Ordinarily, the host would pull SDA low to acknowledge receipt of a byte. Since this is the last byte, the host does not ack so the GBIC knows the transmission is over.

Figure 7. GBIC System Command Communication Sequence.

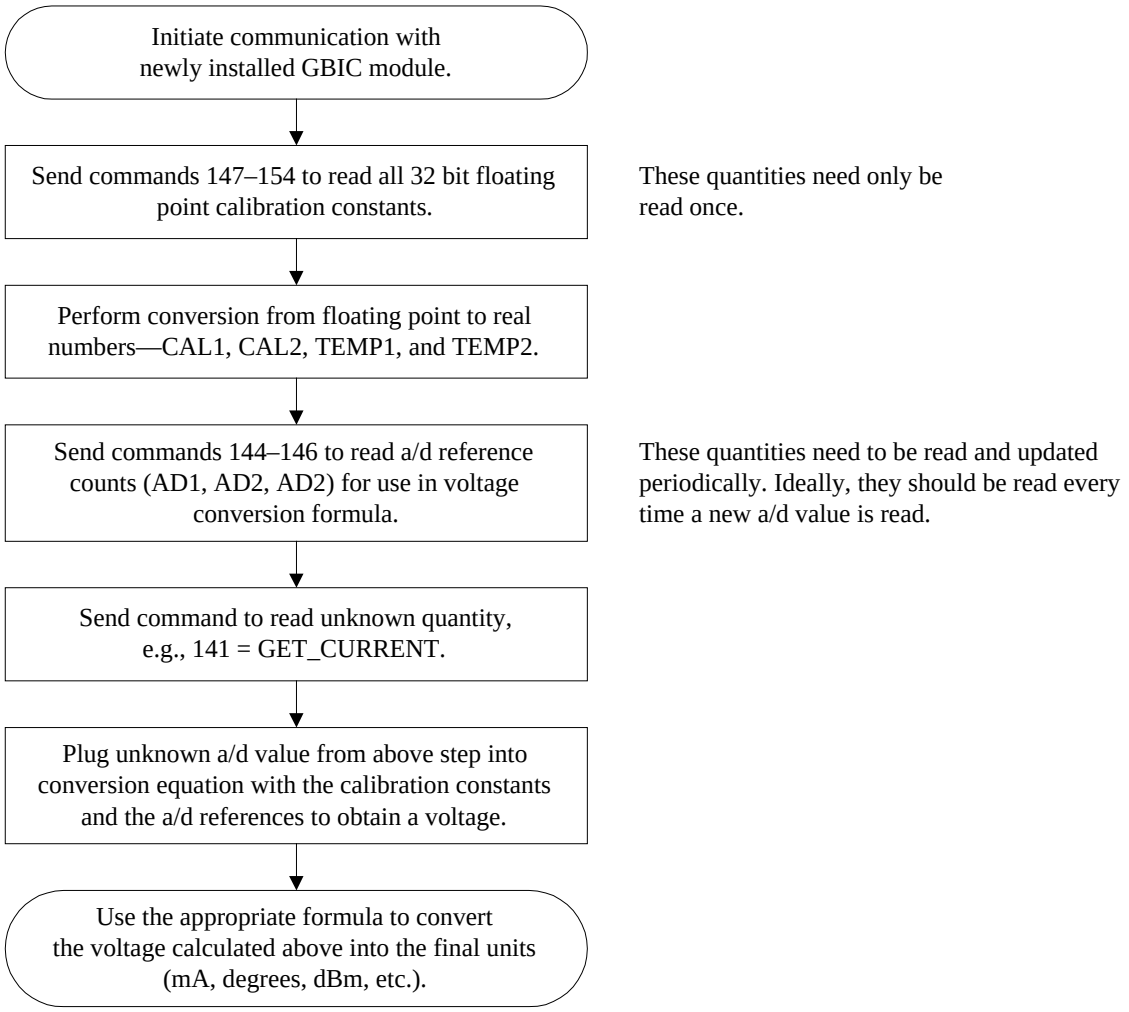


DATA CONVERSION AND INTERPRETATION

Introduction

All of the diagnostic information discussed above (received power, transmitted power, temperature, laser current) is read out as 10 bit a/d counts. In order to convert the quantities read over the I²C bus into real-world units like mA or degrees, several conversions are required. In order to perform the conversion, several calibration constants are required. These are stored as 32 bit floating point values and can be accessed with commands 149–154 (Table 2). The complete conversion process is summarized below.

Figure 8. Conversion Summary



Floating Point Conversions

The calibration constants are stored as 32 bit floating point numbers. Each 32 bit quantity is read out of the GBIC in two 16 bit segments. The most significant 16 bits are accessed with the lower command, and the least significant 16 bits are accessed with the higher command. For example, the high 16 bits of *CAL*2 are accessed with command 149 (GET_CAL2_HI), and the low 16 bits are accessed with command 150 (GET_CAL2_LO). Each 16 bit quantity is transmitted most significant byte first. The 32 bits are broken into four 8 bit segments: the exponent, low 8 mantissa bits, middle 8 mantissa bits, and high 8 mantissa bits. They are organized as follows:

16 bit segment	**high byte**	**low byte**
high 16 bits	mantissa low 8 bits	mantissa middle 8 bits
low 16 bits	mantissa high 8 bits	exponent

The high bit of the mantissa high byte is a sign bit. 1 = negative, and 0 = positive. If *D* is the decimal result, *M* is the mantissa, *S* is the sign bit, and *E* is the exponent, *D* is given by:

$$D = (-1)^S * M * (2)^E$$

where

$$E = \text{(exponent byte) - 127}$$

and

$$M = 1 + \sum_{p=0}^{22} 2^{(p-23)} * q$$

where *q* is a multiplier equal to the value of mantissa bit *p*.

Listed below are functions in ANSI C that perform the conversion given the 4 floating point bytes. Note that there will be subtle differences between results calculated on a calculator using the above formulas and those calculated using the code below due to differences in how floating point precision is handled by the respective machines.

For reference, here are typical values for each of the four 32 bit calibration constants.

$$CAL2 = 1.172 \text{ volts}$$
$$CAL1 = 0.127 \text{ (unitless)}$$
$$TEMP2 = 0.0036 \text{ volts/°C}$$
$$TEMP1 = 1.05 \text{ volts}$$

```
/*****************************************************************************
 *                                                                           *
 *          Functions to Convert 32 bit Floating Point                       *
 *          Representation to Decimal Format                                 *
 *                                                                           *
 *****************************************************************************/

#include <stdio.h>
#include <math.h>

double calculate_exponent(int eb)
 {
   double exponent = 0;
   exponent = eb - 127;
   return (exponent);
 }

int calculate_sign(int high_byte)
 {
   if(128 & high_byte)
     return (1);
   else
     return (0);
 }

double calculate_mantissa(int high_byte, int mid_byte, int low_byte)
 {
   int power = 0;
   int multiplier = 0;
   float mantissa = 0;
   long int mantissa_bits;
   long int index = 8388608;  /* 0b10000000000000000000000 */

   high_byte &= 127;   /* strip off sign bit. */
```

```
    mantissa_bits = (low_byte + (mid_byte * 256) + (high_byte * 65536) + 8388608);

    for (power = 0; power >= -23; power--)
      {
        if (mantissa_bits & index)
          multiplier = 1;
        else
          multiplier = 0;
        mantissa = mantissa + (pow(2,power) * multiplier);
        index = index/2; /* bit shift right */
      }

    return (mantissa);
  }

double ieee_to_decimal(int eb, int high_byte, int mid_byte, int low_byte)
  {
    double decimal_result = 0;
    double mantissa = 0;
    double exponent = 0;
    int sign        = 0;  /* 0 = positive, 1 = negative */

    mantissa = calculate_mantissa(high_byte,mid_byte,low_byte);
    exponent = calculate_exponent(eb);
    sign     = calculate_sign (high_byte);

    decimal_result =  ((pow(-1,sign))*(mantissa)*pow(2,exponent));
    return (decimal_result);
  }
```

A/D Conversion

Once the decimal calibration constants, *CAL*1 and *CAL*2, have been calculated (see above), the host can convert 10 bit a/d counts, N_{in}, into voltage values, V_{in}, using the following formulas together with *AD*1, *AD*2, and *AD*3 which can be read from the GBIC:

(1)

$$N_1 = AD3 - CAL1 * (AD2 - AD3)$$

(2)

$$Vin = \frac{(N_{in} - N_1)}{(AD1 - N_1)} * CAL2$$

Final Data Conversion

Temperature, received power, transmitted power, and laser current each require a different formula to convert the voltage or a/d count obtained from the GBIC to the appropriate units. Each of the four steps is outlined below.

1) Temperature conversion

The decimal quantities, *TEMP*1, and *TEMP*2 must first be read using commands 151–154, and then converted from floating point to decimal representation using the algorithm discussed above. V_{temp} (in volts) must also be calculated using the formulas above and the a/d count obtained using command 140. *T*, the temperature in degrees Celsius, can be calculated using the following:

$$T = 25 + \frac{(V_{temp} - TEMP1\ )}{TEMP2}$$

2) Received power conversion

The host must first obtain the 10 bit received power a/d count by sending the GBIC command number 142. It is also necessary to read *CAL_PWR* from EEPROM location 111, and the 10 bit *RX_CAL* constant from locations 101 and 102. First, it is necessary to convert the 8 bit *CAL_PWR* value into mW using the following formula:

$$cal\_power_{mW} = 10^{(\frac{-(CAL\_PWR)}{10})}$$

The unknown received power $P_{received}$ can then be calculated (in mW) with the following formula:

$$P_{received} = \frac{RX\_PWR_{count} * cal\_power_{mW}}{RX\_CAL_{count}}$$

3) Transmitted power conversion

Using the same cal_power_{mW} calculated above, TX_PWR_{count} read with command 143, and TX_CAL_{count} read from locations 97 and 98, together with the formula below, the host can calculate the transmitted power in mW (long wavelength modules only).

$$P_{transmitted} = \frac{TX\_PWR_{count} * cal\_power_{mW}}{TX\_CAL_{count}}$$

4) Current conversion

Using command 141, the host must first obtain the 10 bit current a/d count, N_{in}. This must then be converted to a voltage using the a/d conversion formulas above. Once $V_{current}$ (in volts) has been obtained, the laser drive current, *I* (mA) can be calculated using the following formulas:

(short wavelength)

$$I_{short} = \frac{V_{current} - 1.5155}{(\frac{R}{1000} + 0.0265)}$$

(long wavelength)

$$I_{long} = \frac{4.4 - (2 * V_{current})}{(\frac{R}{1000})}$$

where R is obtained by sending commands 119 and 120 and recording the quantities LDI_RES_INT and LDI_RES_FRACT and substituting into the form R = LDI_RES_INT.LDI_RES_FRACT.

SAMPLE TEMPERATURE CALCULATION

The following data were obtained from a sample GBIC:

Command	*Name*	*Value (dec)*	*high byte (hex)*	*low byte (hex)*	*high byte (binary)*	*Low byte (binary)*
140	GET_TEMP	184	0	B8	00000000	10111000
144	GET_AD2	189	0	BD	00000000	10111101
145	GET_AD3	25	0	19	00000000	00011001
146	GET_AD1	191	BF	BF	10111111	10111111
147	GET_CAL1_HI	49372	C0	DC	11000000	11011100
148	GET_CAL1_LO	124	0	7C	00000000	01111100
149	GET_CAL2_HI	58335	E3	DF	11100011	11011111
150	GET_CAL2_LO	6527	19	7F	00011001	01111111
151	GET_TEMP1_HI	47925	BB	35	10111011	00110101
152	GET_TEMP1_LO	3455	0D	7F	00001101	01111111
153	GET_TEMP2_HI	29198	72	0E	01110010	00001110
154	GET_TEMP2_LO	29814	74	76	01110100	01110110

Each of the 32 bit calibration constant floating point quantities is first broken up into high, mid, and low mantissa bytes and an exponent byte:

Quantity	*Mantissa High*	*Mantissa Middle*	*Mantissa Low*	*Exponent*
CAL2	11001	11011111	11100011	1111111
CAL1	0	11011100	11000000	1111100
TEMP2	1110100	1110	1110010	1110110
TEMP1	1101	110101	10111011	1111111

Decimal values of each floating point quantity are then calculated using the C program above. In this case, the results are:

CAL2 = 1.194
CAL1 = 0.1249
TEMP2 = 0.003724
TEMP1 = 1.103

1. Calculate N_1:

$$N_1 = (25) - (0.1249) * (189 - 25) = 4.5164$$

2. Calculate V_{temp}:

$$Vtemp = \frac{(184 - 4.5164)}{(191 - 4.5164)} * 1.194 = 1.149V$$

3. Calculate T:

$$T = 25 + \frac{(1.149 - 1.103)}{0.003724} = 37.4°C$$

REFERENCES

1. Sun Microsystems Computer Company, “Gigabit Interface Converter (GBIC) Revision 5.1” (Sun Microsystems, Inc., 1998).
2. Philips Semiconductor Corporation, “Data Handbook—I²C Peripherals for Microcontrollers” (Signetics Company, 1992), 2-22.
3. Atmel Corporation, “AT24C01A/02/04/08/16 2-Wire Serial CMOS E²PROM Data Sheet” (Atmel Corporation, 1998).

Finisar Corporation
1308 Moffett Park Drive
Sunnyvale, CA 94089-1133

(408) 548-1000
(408) 541-6138 FAX

sales@finisar.com
www.finisar.com

Rev. 1.1a
MIM0503a