OXIGRAF

REMOTE OPERATION GUIDE

OXIGRAF MO2i FAMILY OXYGEN SENSORS

Document Number 08-0478
Rev A4, March 10, 2011
File Name 08-0478A4 Mo2i Remote Operation.doc

By Eric Thorson
Design Engineer

Revision History

Rev A1 April 19, 2004
Document creation.

Rev A2 April 26, 2004
Added interface code example.

Rev A3 April 13, 2006
Added RS232 interface pinout.

Rev A4 March 9, 2011
Added Sample Pulse mode setting to P command.

1. Introduction

The Oxigraf MO2i oxygen sensor family can be controlled by commands received over the RS232 link from a host computer using an Oxigraf proprietary protocol. The communications link is active on power-up although the oxygen reading will not be valid until the oxygen analyzer has acquired the absorption line. During system warm-up, a thermoelectric cooler control loop ramps the laser diode temperature to the approximate operating point. The system then searches for and locks onto the particular absorption line used for the measurement. No oxygen absorption data is valid until line lock has been achieved.

2. Serial Communication

2.1 Interface

On the desktop analyzers the RS232 serial interface is accessed via a standard pinout 9-pin female "D" connector. The industrial analyzers a terminal strip is provided for connection to the three interface signals.

Signal	9-Pin D Connector Pin	Terminal Strip Label
Transmit Data from Analyzer (TXD)	2	TXD
Receive Data to Analyzer (RXD)	3	RXD
Interface Ground (GND)	5	GND

The default data format is 9600 baud, 8 bit, no parity, with alternate baud rates of 2400, 4800, 19200, and 38400 selected by interface commands.

2.2 Command Protocol

In RS232 mode the MO2i responds to ASCII commands received on the Rxd line. Data is returned to the host via the Txd line in ASCII or binary format. Handshaking using XON/XOFF or RTS/CTS is not supported or necessary due to the command/response characteristic of the protocol.

On power-up the Oxigraf analyzer restores the current configuration from EEPROM and places itself in command mode. In addition, the RS232 interface is reset to its default setting if a serial break character is detected. A break character is defined as any continuous marking condition lasting longer than 100 ms.

When the Oxigraf analyzer receives a command, it is executed immediately, with a response returned to the host on completion. The host should not send the next command until the response from the current command is received.

2.3 Command Format

The MO2i command format consists of an escape character (0x1b), followed by a one-character command designator, followed by optional parameters, and terminated by a semicolon. Any characters between the end of the previous command and the escape character of the next command are ignored. The command parameters are ASCII encoded decimal numbers, separated by commas.

Responses are transmitted to the host on completion of command execution. Responses can also be returned continuously at a periodic rate as selected by the Report Rate command. The host can suspend a periodic response by issuing any command. Continuous reporting will be suspended on receipt of the command escape character, and resume after the command is executed.

Response suspension will always occur at record boundaries, so the host receive buffer should be large enough for the largest expected response.

2.4 Response Format

2.4.1 ASCII Response Format

All commands are acknowledged with a response. The ASCII response format consists of the command character, a colon, an optional parameter field, and a terminating CR/LF (0x0d, 0x0a) sequence. The content of the parameter field is command specific. Numeric parameters are right justified in a seven-character field consisting of leading spaces, an optional "-" sign character, and the decimal numeric characters. The equivalent C language printf format is "%7d". String parameters are returned as ASCII characters.

2.4.2 Binary Response Format

The MO2i may be configured to use a binary response format in order to reduce the number of bytes returned per response. A binary response record has the following structure:

ACK Length Cmd Data Data ... Data Checksum
ACK = ASCII character 0x06
Length = Number of Cmd and Data bytes to follow
Cmd = Command that initiated this response
Data = Report data fields, 2 bytes per parameter,
msbyte first for numeric parameters; ASCII string for string parameters.
Checksum = Byte_sum of Cmd and Data MOD 2^16, 2 bytes, msbyte first.

2.4.3 ASCII Error Response Format

An error response will be generated if the oxygen sensor detects an error in parsing the command. The ASCII mode format for an error response consists of the command character, a colon, the string "ERROR", a numeric error code, and a CR/LF pair.

2.4.4 Binary Error Response Format

The following format is used to report errors in binary mode:

NAK Length Cmd Error_code Checksum
NAK = ASCII character 0x15
Length = 2 = number of Cmd and Error bytes to follow
Error_code = 1 byte numeric code
Checksum = Byte_sum of Cmd and Error bytes MOD 2^16, 2 bytes, msbyte first.

3. Command Set

The MO2i responds to the following ASCII command set. Note that spaces not part of the command are shown between the command components for clarity. Parameters required by some commands are described in detail in Section 4.

3.1 Report Parameters: Esc R p0,p1,...,pn;

Specify the order and identification of parameters reported to the host by the oxygen analyzer, where p0..pn are the decimal parameter identifiers. If the report rate is zero, then this command will result in the output of one report record, containing parameters p0...pn. If periodic reporting is enabled, the new parameter list will become active for the next report record. If a report with no parameters is requested, the parameter list for the last report will be assumed.

Example: Esc R 0,1;	Report status and oxygen
Response Data:	status, oxygen
Error Codes:	1 = parse error or invalid parameter
	2 = too many parameters

3.2 Report Period: Esc P n; or Esc P n,t;(Sample pulse option)

Set the report period to n in 10 ms units. If n is zero, reports are only transmitted in response to Esc R commands. If n is 2 or greater, the report specified by the Report Parameters command will be transmitted every n x 10 ms. If n equals 1 the report period is set to the modulation cycle period or 9.2 ms. Note that communications baud rate must be high enough to support the selected report rate.

If the Esc P n,t; format is uses the Sample time is set to t in 100 ms units and a sample is initiated. To return to normal Run Mode set t to 0.

Example: Esc P 2;	Set report period to 20 ms.
Example: Esc P 0,35;	Set sample time to 3.5 seconds(3500 ms)
Response Data:	none
Error Codes:	1 = parse error or invalid period value

3.3 Sample Average: Esc A n; or Esc A n1, n2; (CO2 Sensor Option)

Set the time response of the oxygen measurement. Parameter n, ranging from 0 to 13, selects the type of filtering applied to the measured oxygen. If the CO2 sensor option is installed the second parameter (n2) sets the CO2 time response filter.

n	*Step Response*
0	No filtering
1	20 ms running average
2	40 ms running average
3	80 ms running average
4	160 ms running average
5	200 ms exponential response
6	300 ms exponential response
7	500 ms exponential response
8	1 second exponential response
9	2 seconds exponential response
10	4 seconds exponential response
11	8 seconds exponential response
12	16 seconds exponential response
13	Prediction filter to enhance response time

Example: Esc A 8;	Set O2 step response time to 1 second.
Response Data:	None
Error Codes:	1 = parse error or invalid average value
Example: Esc A 5,7;	Set O2 step response to 200 ms, CO2 response to 500 ms.
Response Data:	None
Error Codes:	1 = parse error or invalid average value

3.4 Calibrate: Esc C p1, p2;

Calibrate the Oxygen analyzer, assuming oxygen concentration p1 is in the sample chamber. The parameter p1 is a number from 0 to 10000, representing oxygen in 0.01% units or CO2 in 0.001% units. Two gas samples, preferably widely separated in concentration, are necessary to do a complete calibration. The high cal O2 sample (typically 100% O2) may be indicated by a negative number or by parameter p2 equal to 1.

The low cal cycle with a positive parameter (typically 20.8% O2), or parameter p2 equal to 0, updates the cal factors for linear interpolation between the high cal sample and low cal sample. Subsequent cal cycles with positive parameters will continue to use the same high cal point established with the negative parameter. The cal cycle takes up to 5 seconds to complete, at which time a response is generated.

An O2 span cal cycle, specified with p2 = 2, calibrates the absorption amplifier analog signal gain prior to performing a high cal cycle. A low cal cycle must also be performed to complete the span calibration.

The optional CO2 sensor is calibrated using high and low CO2 gas samples in a manner similar to O2 sensor calibration. Parameter p2 indicates low point calibration (p2 = 3) or high point calibration (p2 = 4), while p1 specifies the CO2 cal gas concentration in 0.001% units. For example, p1 set to 10000 would indicate that 10% CO2 cal gas was being used.

Cal cycle summary:

Parameter p2	**Function**
0 or absent:	Low or high cal as function of p1 sign, as described above.
1:	High cal cycle, p1 need not be negative.
2:	Span cal cycle with absorption amplifier analog gain calibration. This function adjusts the absorption signal analog gain prior to performing a normal high cal cycle.
3:	Optional CO2 sensor low cal cycle.
4:	Optional CO2 sensor high cal cycle

Calibration Examples:

Example: Esc C -10000;	High cal at 100% O2.
Esc C 10000,1;	Same as above
Esc C 10000,2;	Span cal at 100% O2
Esc C 2060;	Low cal at 20.6% O2
Esc C 2080;	Recalibrate at 20.8% O2
Esc C 10000,4;	CO2 sensor high cal at 10% CO2
Esc C 0,3;	CO2 sensor low cal at 0% CO2
Response Data:	None
Error Codes:	1 = parse error or invalid calibration value, cal aborted.
	2 = cal value too close to reference value, cal aborted.
	3 = calibration error, cal factors out of range.
	4 = oxygen sensor not in line lock, cal aborted
	5 = oxygen reading not stable.

3.5 Save Configuration: Esc S;

Save the current system calibration in nonvolatile memory.

Example: Esc S;	Save system calibration data
Response Data:	none
Error Codes:	1 = parse error
	2 = failed to store in EEPROM

3.6 Version Report: Esc V;

Return the current firmware version string including the firmware version number, EEPROM configuration version, and cell configuration version.

Example: Esc V;	Report Version
Response Data:	Oxigraf MO2iA V1.07.00400.00400
Error Codes:	1 = parse error

3.7 Identification Report: Esc W;

Return the controller serial number, cell serial number, and cell operating time in hours. Note that repeated execution of the Identification command may cause excess measurement noise due to access of the cell EEPROM.

Example: Esc W;	Report Identification
Response Data:	123, 245, 301;
Error Codes:	1 = parse error

3.8 Read Clock Calendar: Esc H;

Read the current time and date from the optional battery backed up real time clock. Seven parameters are returned: year, month, date, day of week, hour (24 hr format), minute, and second. This command will not return valid data if the clock/calendar option is not installed.

Example: Esc H;	Return time and date
Response Data:	2003, 10, 8, 4, 11, 6, 35;
Error Codes:	1 = parse error

3.9 Set Clock Calendar: Esc H y, m, d, w, h, m, s;

Set the specified time and date into the optional battery backed up real time clock. Seven parameters are required: year, month, date, day of week, hour (24 hr format), minute, and second. This command will not store valid data if the clock/calendar option is not installed.

Example: Esc H 2003, 10, 4, 8, 4, 11, 6, 40;	Set time and date
Response Data:	none
Error Codes:	1 = parse error 2 = invalid or not enough parameters

3.10 Binary Response Format: Esc F n;

Enable the binary report format if n is nonzero, disable binary report format if n is zero or absent. The response is returned in the format in use before the command execution.

Example: Esc F 1;	Set binary format
Response Data:	none
Error Codes:	1 = parse error

3.11 Set Communications Baud Rate: Esc B n;

Set the communications baud rate to the value selected from the following table. The response is returned at the baud rate in use before the command is executed.

n	Baud Rate
0	38400
1	19200
2	9600 (default)
3	4800
4	2400
5	1200

Example: Esc B 1;	Set baud rate to 19200.
Response Data:	none
Error Codes:	1 = parse error 2 = baud rate index out of range.

3.12 Initialize: Esc I;

Reset all parameters to the power-up default state. This command will reset the report format to ASCII and the communications baud rate to 9600 baud. It will also restart the line search process. The command response will be returned prior to re-acquisition of the O2 line.

Example: Esc I;	Initialize
Response Data:	none
Error Codes:	1 = parse error

3.13 Initialize Communications Interface: Break Character

Reset the serial interface to 9600 baud and ASCII report format. This command also terminates continuous reporting. No response is generated. If the break character is followed by a valid Oxigraf protocol command then the MO2i will remain in RS232 mode. Otherwise the interface will be configured for RS485 Modbus operation if the Modbus option is installed (19200 baud, 8 bit, even parity).

3.14 Set Standby Option: Esc Z n

Set the oxygen sensor standby mode as specified by parameter n. The cell heater, laser TEC cooler, and laser may be selectively turned off to save power and extend laser lifetime. Update of the cell hour meter (ET parameter) is suspended when the laser is turned off in one of the standby modes. Bit 0 of the Status Word (parameter 0) is set for any non-zero standby mode.

n	Standby Mode
0	Exit all standby modes, start line search and return to normal operation
1	Cell heater off
2	Laser off
3	Laser and Cell heater off
4	Laser and TE cooler off
5	Laser, TE cooler, and cell heater off
6	Laser and TE cooler off (same as mode 4)
7	Laser, TE cooler, and cell heater off (same as mode 5)

3.15 Self-Test: Esc T n;

Execute the self-test routine indexed by opcode n. The results are placed in the Status Register, which can be examined by the Report Parameters command. The test opcode n selects the test routine executed as follows:

Opcode	Function
0	Normal power-up self test
1	No operation
2	No operation
3	Pressure lookup calibration

The normal self-test routine (n = 0) performs a system integrity check, with the results reported in the system status word. See Section 3.7 for details on the status format.

The pressure lookup calibration option is a factory test function used to calibrate the pressure compensation curve for the peak absorption mode. It is not intended to be used in customer applications.

Example: Esc T;	Execute normal self test
Response Data:	none (results in Test Register)
Error Codes:	1 = parse error

3.16 Get Parameter: Esc L n;

Read a single parameter identified by n without affecting the report specification set up by the 'R' command.

Example: Esc L 1;	(read O2 concentration)
Response Data:	parameter value (-32768 to 32767)
Error Codes:	1 = parse error

4. MO2i Parameters

This section describes the format of MO2i parameters accessible via the RS232 communication protocol.

0 -- System Status (S)

S is a 16-bit binary system status byte. The bit allocation, from lsb to msb is:

Bit 0:	Set to 1 for non-zero standby status
Bit 1:	Line lock acquired (system ready).
Bit 2:	Laser enable bit, off until laser temperature stabilized.
Bit 3:	Reserved
Bit 4:	Uncalibrated warning bit, set if current calibration factors are invalid. Reset with the low point calibration command.
Bit 5:	Test fault warning bit, disables laser and inhibits temperature control if set due to an internal self test error.
Bit 6:	Cell warm-up status, set while cell warming up to operating temperature
Bit 7:	Pressure calibration mode flag, set while pressure calibration is active.

Bits 8-15 contain the results of the most recent self-test command.

Self Test (test opcode 0):

Bit 8:	Microcontroller memory checksum failure.
Bit 9:	Configuration EEPROM signature failure.
Bit 10:	Watchdog time out.
Bit 11:	Invalid O2 computation error.
Bit 12:	Low reference signal level.
Bit 13:	Cell null balance failure.
Bit 14:	Laser temperature control failure.
Bit 15:	Reserved

Pressure Calibrate (test opcode 2):

Bits 8-15	Current pressure compensation table index.

1 – OxyVal: Oxygen Concentration

OxyVal ranges from 200 - 10000, representing oxygen from 2 to 100% in 0.01% increments. Percent oxygen is computed from the measured absorption by averaging and linear interpolation through the high and low calibration points. The number of samples averaged to compute O2 is set by the Sample Average command. Compensation for sample pressure and temperature is also performed to maintain accuracy over the environmental operating range. OxyVal is set to zero if the oxygen measurement is invalid.

2 – CellPres: Sample Cell Pressure

CellPres, ranging from 0 to 12000, is the sample cell absolute pressure in 0.1 mB units.

3 – CellTemp: Sample Cell Temperature

CellTemp, ranging from -2030 to 7031, is the sample cell gas temperature in 0.01 degree C units. After warm up, CellTemp will be regulated to approximately 4500 (45 deg C).

4 – SamFlow: Sample Flow Rate

SamFlow is the measured gas sample flow rate in ml/min.

5 -- TimeStamp: Time Stamp Counter

TimeStamp is the value of an unsigned 16 bit timer that increments once every modulation cycle (9.2 ms)

6 – Alarms: Alarm Status Word

The alarm status word is a 16 bit binary value representing the state of the various MO2i alarm detectors. Specific bits in Alarms are set if the respective alarm state is active.

Bit 0 (LSB)	Low O2 alarm A
Bit 1	High O2 alarm A
Bit 2	Low O2 alarm B
Bit 3	High O2 alarm B
Bit 4	Low CO2 alarm A
Bit 5	High CO2 alarm A
Bit 6	Low CO2 alarm B
Bit 7	High CO2 alarm B
Bit 8	Low supply voltage alarm
Bit 9	Low sample flow alarm
Bit 10	High sample flow alarm
Bit 11	Low sample cell pressure alarm
Bit 12	High sample cell pressure alarm
Bit 13	Reserved
Bit 14	Critical self test failure alarm
Bit 15	Reserved

7 -- CO2: CO2 Concentration

CO2 ranging from 0 - 1500, represents carbon dioxide from 0 to 15% in 0.01% increments. CO2 is computed from the measured absorption by averaging and linear interpolation through the high and low calibration points. The number of samples averaged to compute CO2 is set by the Sample Average command. Compensation for sample pressure and temperature is also performed to maintain accuracy over the environmental operating range. This parameter reads zero if the optional CO2 sensor is not installed.

8 – Co2Pres: CO2 Sample Cell Pressure

Co2Pres is the CO2 sample cell absolute pressure in 0.1 mm Hg units. Co2Pres is used for compensation of the CO2 absorption measurement. Note that the CO2 cell pressure is measured at the cell, and will probably differ from the pressure measured at the sample point as a result of sample tubing pressure drop. This parameter reads zero if the optional CO2 sensor is not installed.

9 – Co2Temp: CO2 Sample Cell Temperature

Co2Temp is the CO2 sample cell gas temperature in 0.01 degree C units. Co2Temp is used for temperature compensation of the CO2 absorption measurement. This parameter reads zero if the optional CO2 sensor is not installed.

5. Calibration

The Calibration command is used to calibrate the oxygen or CO2 sensors using a known oxygen or CO2 concentration. To fully calibrate either sensor, two gas samples are required. For the oxygen sensor typical calibration gases near 100% and 21% and certified to 0.03% accuracy are used. Room air may be used for O2 cal but is diluted by water vapor and may vary from 20.5 to 20.9% if not dried. For the CO2 sensor, 0 and 10% are appropriate, noting that room air may contain 0.1% CO2 if not scrubbed. An oxygen cal gas may also be used for the CO2 zero point as well as one of the O2 cal points.

All calibration modes except span cal are incremental, i.e., calibration at one point (high or low) does not affect the calibration at the other point. Consequently, high or low point calibration can be repeated or done in any order. Span calibration optimizes the absorption amplifier front-end gain for measurement up to the high point gas concentration. Since front-end gain affects both cal points the low point calibration factors are invalidated by a span cal, necessitating a follow-up low point calibration. The span is optimized to 100% O2 at the factory.

To perform a two-point oxygen cal use the following procedure. The CO2 sensor is calibrated in a similar manner.

a) Pass 100% oxygen through the instrument. When the measured value has stabilized, execute the Calibrate command with p1 equal to 10000 and p2 equal to 1. The cal factors and oxygen reading will change as a result of this cal cycle. Analog span calibration is enabled by calibration parameter p2 equal to 2. Uncalibrated status is set by an analog span calibration cycle, indicating that low point calibration is necessary to finish calibration of the sensor.

b) Repeat step (a) for the second sample value, setting p1 to the low point O2 concentration and setting parameter p2 equal to 0. Widely separated sample concentrations, such as 100% and 20.9%, will provide the most accurate calibration. The cal factors and oxygen reading will change as a result of this cal cycle. Low point calibration clears the uncalibrated status bit set by a analog span cal command.

c) Save the cal factors in non-volatile memory with the Save command. The calibration results will be lost on the next power cycle if they are not saved.

Appendix – Code Example

```
// Sample.c
// Oxigraf Interface Code Sample
//

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "com.h"

// Global variables
#define OXISZ 20
#define ACK 0x06
#define NAK 0x15

char oxibuf[OXISZ+2];  // oxigraf serial receive buffer
int  oxitimer;    // oxigraf watchdog timer
int  caltimer;    // calibration settling delay timer

char OxiFlags;    // global status flags
#define OXI_ENABLE 0x01
#define OXI_FAIL   0x02

// init oxigraf serial channel, reset sensor com settings
// call from foreground process
#define OXI_TIMEOUT 10  // number of ChkStr tries
void InitOxi(void)
{
   oxitimer = OXI_TIMEOUT;  // force init in RunOxi
}

//
// Oxigraf sensor I/O task, called every 10 ms from timer interrupt
// void OxiRun(void)
//
void RunOxi(void)
{
   if(oxitimer < OXI_TIMEOUT) {
      oxitimer++;
      while(oxibuf[0] == 0) {  // sync up to ACK character
         if(GetSc(COM1, oxibuf+1)) return;  // return if no character in fifo
         if(oxibuf[1] == ACK) oxibuf[0] = 1;
      }
      ChkNstr(COM1, oxibuf, OXISZ);  // get all of UART fifo
      if(oxibuf[0] >= 9) {
         OxiO2 =   *((int*)(oxibuf+4));
         OxiPres = *((int*)(oxibuf+6));
         OxiTemp = *((int*)(oxibuf+8));
         oxibuf[0] = 0;  // clear receive buffer character count
         oxitimer = 0;
         OxiFlags &= ~OXI_FAIL;
         PutStrf(COM1, "\x1b" "R;");  // request data report
      }
      return;
   }

   switch(oxitimer++) {  // initialization state sequence

   case OXI_TIMEOUT:  // after 10 tries initialize
      OxiFlags |= OXI_FAIL;
      SerialEnable(COM1, 9600, COM_8N1);  // oxigraf serial channel
      SetBrk(COM1);
      break;

   case OXI_TIMEOUT+20:  //  200 ms break time
      ClrBrk(COM1);
      break;

   case OXI_TIMEOUT+25:  //  50 ms delay
      PutStrf(COM1, "\x1b" "A0;");  // fast time response
      break;

   case OXI_TIMEOUT+30:  //  50 ms delay
      PutStrf(COM1, "\x1b" "F1;");  // binary mode
      break;

   case OXI_TIMEOUT+35:  //  50 ms delay
      PutStrf(COM1, "\x1b" "B1;");  // set 19200 baud
```

```
      break;

   case OXI_TIMEOUT+40:  //  50 ms delay
      SerialEnable(COM1, 19200, COM_8N1);  // 19200 baud, clr uart fifo
      oxibuf[0] = 0;  // clear receive buffer character count
      oxitimer = 0;
      PutStrf(COM1, "\x1b" "R1,2,3;");  // request data report
      break;
   }
   return;
}

// Calibrate Oxigraf O2 sensor
// int CalOxi(int opcode, int o2, int caltc)
//    opcode:  0 = low, 1 = high, 2 = span
//    o2 = oxygen in 0.01% units
//    caltc = number of 10ms ticks to wait for completion
//
int CalOxi(int opcode, int o2, int caltc)
{
   char cmdstr[12];
   char ecode;

   OxiFlags &= ~OXI_ENABLE;  // kill oxigraf interrupt routine
   Delayms(100);  // wait for unfinished response
   SerialEnable(COM1, 19200, COM_8N1);  // clear uart fifo
   sprintf(cmdstr, "\x1b" "C%d,%d;", o2, opcode);  // format command string
   PutStrf(COM1, cmdstr);  // request O2 calibration
   ecode = -1;  // default to timeout error
   caltimer = caltc;
   oxibuf[0] = 0;  // clear receive buffer character count

   while(caltimer) {
      if(oxibuf[0] == 0) {  // sync up to ACK character
         if(GetSc(COM1, oxibuf+1)) continue;
         if((oxibuf[1] == ACK) || (oxibuf[1] == NAK)) oxibuf[0] = 1;
         continue;
      }
      if(oxibuf[0] == 1) {  // got ACK or NAK
         ChkNstr(COM1, oxibuf, 2);
         continue;
      }
      if(ChkNstr(COM1, oxibuf, oxibuf[2]+2)) {
         oxibuf[0] = 0;  // clear receive buffer character count
         if(oxibuf[3] == 'C') {  // if calibrate response
            if(oxibuf[1] == NAK) {  // if error
               ecode = oxibuf[4];
               break;  // break out of while loop
            } else {
               PutStrf(COM1, "\x1b" "S;");  // save cal results
               ecode = 0;
            }
            continue;
         }
         if(oxibuf[1] == 'S') {  // if save command response
            break;  // break out of while loop
         }
      }
   }
   oxibuf[0] = 0;  // clear receive buffer character count
   PutStrf(COM1, "\x1b" "R1,2,3;");  // request data report
   OxiFlags |= OXI_ENABLE;  // start up oxigraf interrupt
   return ecode;  // return error code
}
```

// com.c

```
// 16C654 Quad UART Polled I/O routines

#include "com.h"

/*********************************************************/
/* Enable the serial port, reset FIFOs */
void SerialEnable (int Port, int Speed, int LineCtrl)
{
   Comreg(Port, IER) = 0x00;  // no interrupts
   Comreg(Port, MCR) = 0x00;  // default dtr/rts
   Comreg(Port, FCR) = 0x07;  // fifo on
   SetSpeed(Port, Speed);
   Comreg(Port, LCR) = LineCtrl;
}

/*********************************************************/
/* Disable the serial port */
void SerialDisable (int Port)
{
   Comreg(Port, FCR) = 0x07;  // reset fifo
}

/*********************************************************/
/* This routine gets a char from the uart receive fifo.
/*   Input:  char_ptr = int pointer to store char.
     Output: Return = 0  = char stored at pointer location.
                    = -1 = buffer is empty. no char stored.
*/

int GetSc (int Port, char *char_ptr)
{
   if ((Comreg(Port, LSR) & RCVRDY) == 0) return(-1);  // not ready
   *char_ptr = (char) Comreg(Port, RXR);      // get char
   return (0);                                          // return ok
}

/*********************************************************/
/* This routine gets multiple characters from the fifo.
   It honors a user requested timeout if no characters are
   available.
   Input: GetScs (Port, *char_ptr, num, timeout)
          char_ptr = pointer at which to buffer chars
          num = number of chars to get
          timeout = seconds to wait if no chars available
   Return:
          0 = characters have been buffered.
          -1 = timeout occurred.
*/

int GetScs (int Port, char *char_ptr, int num, int timeout)
{
int      tcnt;
int      numc;
char     c;
char     *cptr;

/* initialize working variables */
numc = num;
cptr = char_ptr;

while (numc >0)                          /* if any bytes left */
   {
   tcnt = timeout;
   while (GetSc(Port, &c) == -1)        /* get char from buffer */
      {
      Delayms (1);                       /* Delay */
      tcnt -= 1;                         /* decrement timeout count */
      if (tcnt == 0)                     /* if we have timed out */
         return (-1);                    /* return with error */
      }
   *cptr++ = c;                          /* buffer the character */
   numc -= 1;                            /* decrement numb bytes */
   }
return (0);                              /* return ok */
}

/*********************************************************/
/* This routine gets a char string from the fifo.
   It honors a user requested timeout if no characters are
```

```
   available.
   Input: get_str (Port, *char_ptr, delchar, timeout)
          char_ptr = pointer at which to buffer chars.
                     end of string char buffered also.
          delchar = end of string character.
          timeout = seconds to wait if no chars available
   Return:
          0 = characters have been buffered.
          -1 = timeout occurred.
*/

int GetStr (int Port, char *char_ptr, int delchar, int timeout)
{
int      tcnt;
char     c;
char     *cptr;

/* initialize working variables */
cptr = char_ptr;
for (;;)                                  /* forever loop */
  {
  tcnt = timeout;                         /* init timeout count */
  while (GetSc(Port, &c) == -1)          /* get char from buffer */
    {
    Delayms (1);                          /* Delay one millisec */
    tcnt -= 1;                            /* decrement timeout count */
    if (tcnt == 0)                        /* if we have timed out */
      return (-1);                        /* return with error */
    }
  *cptr++ = c;                            /* buffer the character */
  if (c == delchar) {                     /* if this is the delimeter */
     *cptr = 0;                           /* terminate with null */
     return (0);                          /* exit */
     }
  }
}

/*********************************************************/
/* This routine gets a string from the uart if available.
   It returns the number of characters in the string, zero
   if not available.  Initial call should have zero in first
   buffer position.
   Input: ChkStr (Port, *char_ptr, maxlen, delchar)
          buffer = pointer at which to buffer chars starting
                   at buffer+1.  Char count stored at buffer+0,
                   end of string char buffered also.
          maxlen = maximum size of string
          delchar = end of string character.
   Return: 0 -> not available, n -> string size
*/

int ChkStr(int Port, char *buffer, int maxlen, int delchar)
{
  char *cp;

  cp = buffer + (*buffer);
  while(GetSc(Port, cp+1) == 0) {
     if(*buffer < maxlen) {
        *buffer += 1;  // bump count
        cp++;  // bump pointer
     }
     if(*cp == delchar) return(*buffer);  // number of char
  }
  return(0);
}

/*********************************************************/
/* This routine gets n characters from the uart if available.
   It returns the number of characters in the string, zero
   if not available.  Initial call should have zero in first
   buffer position.
   Input: ChkNstr (Port, *char_ptr, n)
          buffer = pointer at which to buffer chars starting
                   at buffer+1.  Char count stored at buffer+0,
          n = size of string to get
   Return: 0 -> not available, n -> string size
*/

int ChkNstr(int Port, char *buffer, int n)
{
```

```
  char *cp;

  cp = buffer + (*buffer);
  while(GetSc(Port, cp+1) == 0) {
     cp++;  // bump pointer
     if((++(*buffer)) == n) return(n);  // return count when done
  }
  return(0);
}

// Output a character to the serial port when txfifo empty
// Input:  c = character for output.

void PutSc (int Port, char c)
{
   while (((Comreg(Port, LSR)) & XMTRDY) == 0);
   Comreg(Port, TXR) = c;  // output character
   return;
}

/*********************************************************/
/* Output a null terminated string to the serial port */
/* Input:  string = points to char string.
*/
void PutStr (int Port, char *string)
{
   char   *cptr;

/* initialize working variables */
   cptr = string;

/* Loop through buffer outputting chars */
   while(*cptr) PutSc(Port, *(cptr++));
   return;
}

/*********************************************************/
// Output a null terminated string to the tx fifo, size must be < 64 characters.
// Input:  string address, points to char string.
// Return:  0 if fifo not ready, number of characters transfered if fifo empty when called.

int PutStrf (int Port, char *string)
{
   char   *cptr;

// initialize working variables
   cptr = string;

// check for fifo empty
   if(((Comreg(Port, LSR)) & XMTRDY) == 0) return 0;

// Loop through buffer outputting chars
   while(*cptr) Comreg(Port, TXR) = *(cptr++);
   return (string - cptr);
}

/*********************************************************/
void SetBrk(int Port)
{
   Comreg(Port, LCR) |= 0x40;
}

void ClrBrk(int Port)
{
   Comreg(Port, LCR) &= 0xbf;
}

/* Send 200 ms break */
void TxBrk(int Port)
{
   Comreg(Port, LCR) |= 0x40;
   Delayms(200);
   Comreg(Port, LCR) &= 0xbf;
}

/*********************************************************/
/* This routine sets the speed; will accept funny baud rates. */
/* Setting the speed requires that the DLAB be set on         */

void SetSpeed (int Port, int Speed)
```

```
{
   int          divisor;

   if(Speed == 0) return;
   divisor = (int) (691200/Speed);

   Comreg(Port, LCR) |= 0x80;  // set dlab
   Comreg(Port, DLL) = divisor & 0x00ff;
   Comreg(Port, DLH) = (divisor >> 8) & 0x00ff;
   Comreg(Port, LCR) &= 0x7f;  // reset dlab
}
```

// com.h

```
// 16C654 Quad UART include file

extern void Delayms();
extern void SerialEnable();
extern void SerialDisable();
extern int  GetSc();
extern int  GetScs();
extern int  GetStr();
extern int  ChkStr();
extern int  ChkNstr();
extern void PutSc (int Port, char c);
extern void PutStr();
extern int  PutStrf();
extern void SetBrk();
extern void ClrBrk();
extern void TxBrk();
extern void SetSpeed();


#define  ESC        0x1B        /* ASCII Escape char    */
#define  ASCII      0x007f      /* Mask ASCII Char      */
#define  xoff       19
#define  xon        17

#define  COMBASE    0x200       // Quart base address

#define  COM1       COMBASE
#define  COM2       COMBASE+0x08
#define  COM3       COMBASE+0x10
#define  COM4       COMBASE+0x18

#define  Comreg(base, reg)    *(unsigned char volatile *)((base) + (reg))

/*   The 16C654 QUART has 17 registers accessible through 7 port addresses
     for each com port.  Here are their addresses relative to COMnBASE.  Note
     that the bause rate registers, (DLL) and (DLH) are active ony when
     the Divisor_Latch Access_Bit (DLAB) is on.  The (DLAB) is bit 7 of
     the (LCR).

    . TXR Output data to the serial port.
    . RXR Input data from the serial prot.
    . LCR Initialize the serial port
    . IER Controls interrupt generation.
    . IIR Identifies interrupts.
    . MCR Send control signals to the modem.
    . LSR Monitor the status of the serial port
    . MSR Receive status of the modem.
    . DLL Low byte of baud rate divisor.
    . DHH High byte of baud rate divisor.

*/
#define  TXR        0   /* Transmit register (WRITE) */
#define  RXR        0   /* Receive register (READ) */
#define  IER        1   /* Interrupt Enable        */
#define  IIR        2   /* Interrupt ID            */
#define  FCR        2   /* Fifo control            */
#define  LCR        3   /* Line control            */
#define  MCR        4   /* Modem control           */
#define  LSR        5   /* Line status             */
#define  MSR        6   /* Modem Status            */
#define  DLL        0   /* Divisor Latch Low       */
#define  DLH        1   /* Divisor Latch High      */
#define  EFR        2   /* Enhanced function       */

/*************************************************************
   Bit values held in the line control register (LCR).
       bit            meaning
       ___            _______
       0-1        00=5 bits, 01=6 bits, 10=7bits, 11=8bits.
       2          stop bits
       3          0=parity off, 1=parity on
       4          0=parity odd, 1=parity even
       5          sticky parity
       6          set break
       7          toggle port addresses
*************************************************************/
#define NO_PARITY       0x00
#define EVEN_PARITY     0x18
#define ODD_PARITY      0x08
```

```
#define SET_PARITY      0x28
#define CLR_PARITY      0x38
#define STOP_2          0x04
#define DATA_8          0x03
#define DATA_7          0x02
#define DATA_6          0x01
#define DATA_5          0x00
#define COM_8N1         NO_PARITY+DATA_8

/*************************************************************
   Bit values held in the line status register (LSR)
       bit             meaning
       ___             _______
       0           data ready
       1           overrun error - data register overwritten
       2           parity error - bad transmission
       3           framing error - no stop bit was found
       4           break detect - end to transmission requested
       5           transmitter holding register is empty
       6           transmitter shift register is empty
       7               time out - off line
*************************************************************/
#define RCVRDY      0x01
#define OVRERR      0x02
#define PRTYERR     0x04
#define FRMERR      0x08
#define BRKERR      0x10
#define XMTRDY      0x20
#define TIMEOUT     0x80

/*************************************************************
   Bit values held in the Modem output control register (MCR)
       bit             meaning
       ___             _______
       0           data terminal ready.  Computer ready to go
       1           request to send.  Computer wants to send data
       2           auxiliary output #1
       3           auxiliary output #2 (Note: this bit must be
              set to allow the communications card to send
              interrupts to the system)
       4           UART output looped back as input
       5-7             not used.

*************************************************************/
#define  DTR        0x01
#define  RTS        0x02
#define  MC_INT     0x08

/*************************************************************
   Bit values held in the Modem input status register (MSR)
       bit             meaning
       ___             _______
       0           delta clear to send
       1           delta data set ready
       2           delta ring indicator
       3           delta data carrier detect
       4           clear to send
       5           data set ready
       6           ring indicator
       7           data carrier detect

*************************************************************/
#define  CTS        0x10
#define  DSR        0x20

/*************************************************************
   Bit values held in the Interrupt Enable Register (IER)
       bit             meaning
       ___             _______
       0           interrupt when data received
       1           interrupt when transmitter holding reg. empty
       2           interrupt when data reception error
       3           interrupt when change in modem status register
       4-7         not used

*************************************************************/
#define  RX_INT     0x01

/*************************************************************
   Bit values held in the Interrupt identification register  (IIR)
```

```
       bit            meaning
       ___            _______
       0          interrupt pending
       1-2        interrupt ID code
              00=change in modem status register,
              01= transmitter holding register empty,
              10=data received,
              11=reception error, or break encountered
       3-7            not used

*********************************************************/
#define  RX_ID      0x04
#define  RX_MASK    0x07
```