Frank Henriquez **October 9, 1995**

NIRSPEC Electronics Design Note 05.00
Lakeshore Temperature Controller Software Interface

Abstract

Gemini and NIRSPEC both use the same type of Lakeshore Electronic Thermometers and Temperature Controllers to read and maintain the instrument temperatures. The controllers have built-in serial ports that allow them to be operatated from a remote computer. This document describes the communication software required and the limitations of the temperature controllers.

Results with the Lakeshore 208 Electronic Thermometer

The 208 Electronic Thermometer uses a simple 3 line (Read Data, Send Data and Ground) RS-232 interface, so there are no hardware flow control signals. Software flow control is not available in the controller so the computer must time its transactions with the 208 to avoid incorrect read back or confusing the device by sending commands before the previous command has been processed.

The Gemini Motor Controller has two RS-232 serial ports that were added to allow it to talk with the temperature sensors. After the hardware was added and some Occam test code written, we realized that because of the slow temperature controller readout rate, the software modifications required in both the main Gemini PC code and the Motor Controller Transputer software were beyond the time we had allocated to that project.

Since NIRSPEC, and the NIRSPEC Test Chamber, use the same type of temperature controllers as Gemini, the recent tests with the Test Chamber have given us the opportunity to fully characterize the temperature controllers. A small C program, (listed below) running on a PC reads out all eight channels of a Lakeshore 208 Electronic Thermometer every 15 minutes and writes the results to a text file, which can later be read into a spreadsheet program.

Commands to the 208 that do not require a reply from the controller must be allowed to settle for at least **3 seconds** before a new command can be sent. Selecting a channel and reading back data takes much longer - the computer must wait at least **7 seconds** after selecting a channel before sending the command to read back the temperature from the 208. These delays were determined by observing the temperature displayed by the controller vs. the temperature read in

by the computer. If the delay between the Select Channel command (**YCn** where n is the channel number) and the Read Channel Data command (**WS**) is less than 6 seconds, the 208 will occasionally return incomplete data (a display of "----") or even data from the previous channel. Delays of 7 seconds or more will always return good data.

Conclusions and Recommendations

Now that the delays and quirks of the Lakeshore controllers are understood, we should return to the Gemini motor controller and write a generic Occam routine to read in temperatures and return them to the controller. This will give us experience than can be transferred to the new NIRSPEC motor controllers. It will also allow the Gemini control software to read back the instrument temperatures.

Useful Lakeshore Commands

The 208 has several commands dealing with setpoints and alarms, but the following commands are all that are needed to read back the temperatures and control the readout units and scaning mode.

YS	Starts the channel scan
YH	Stops the channel scan
YCn	Selects channel n
FØx	Set the controller readout to format x. Where x is: F for Farenheit, C for Centigrade, K for Kelvin, V for voltage.
WS	Read back the current channel's temperature and alarm settings

The Lakeshore 208 Thermometer has the following serial communications specifications:

Baud Rate	300
Data Bits	7
Parity	Odd
Stop Bits	1
Duplex Mode	Half

Software Listing

This is the souce code for Tempest, a QuickC program that reads the 208 temperature sensor.

The program asks the user for a 1 - 15 minute delay between reads, reads all 8 channels of the 208 and saves the data to a file.
The data is formatted as a comma delimited string as follows:

TIME,	CHANNEL,	TEMPERATURE,	RANGE
HH:MM:SS,	xxx0x,	xxx.xx,	x

The channel number will often display as “Err01”, where Err indicates that the temperature sensor trip points and alarms have not been set. This does not affect the data. The Rage will usually be K (Kelvins) but can also be C (Centigrade), F (Farenheit) or V (volts).

```
/*************************************************************
*   Tempest.c                                                *
*************************************************************
*   Reads in temperatures from the Lakeshore 208 controller *
*   and saves them to a file.                                *
*   Based on code from serial.c and other IR lab code.       *
*   version history:                                         *
*                                                            *
*   0.4   940422 original C code.                            *
*   0.5   950929 first Lakeshore version                     *
*   0.6   951003 added file I/O  (from Microsoft C Bible)    *
*   0.7   951005 improved serial init , reads all channels   *
*   0.8   951020 added multi-minute delay between reads      *
*                                                            *
*   Frank Henriquez UCLA Astronomy Dept.                     *
*************************************************************/
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include "serial.h"


/********  Function prototypes  ********/
void          ScanTemp(void);
void          CvtTemp(void);
void          TempTime(void);
void          GetTemp(void);
void          Delay(clock_t);
int           InitSerial(unsigned,long,int,int,int);
unsigned char InSerial(void);
unsigned char OutSerial(char);
unsigned char LakeStat(void);
void          OutString(char *);
void          InString(char *);
/************************************/

int portbase;    /* COM port address */
int Done;
int err;
int chanum;
double  count, tused;
char ch, kh;
unsigned Serr;
time_t tnow, tstart, tstop;
```

```
struct tm *tmnow;


char filename[81], input[80] = "xx";
char tokensep[] = ",";
char *chanstr, *tempstr;
char timestr[9];
FILE *tempfile; /* file pointer to opened file */

main()
{
 Done = 0;
 err = InitSerial(COM1,300,EVEN_PARITY,7,1);
 if (err != 0)
 {

   printf("error opening serial port\n");
   Done = 1;
 }
 printf("Enter the number of minutes between readings (1 - 15): ");
 scanf("%le",&count);
 count = count * 60.0;
 printf("Enter a file name for the temp sensor data: ");
 scanf("%s",filename);
 if ((tempfile = fopen(filename,"a+")) == NULL)
    {
     printf("Doh! error opening file: %s\n",filename); /* Doh!*/
     exit(0);
     }

 time(&tnow);
 tmnow = localtime(&tnow);
 printf("Local time = %s\n",asctime(tmnow));
 printf("press a key to stop taking data...and wait for the scan to
finish\n");

 OutString("YH");
 Delay(3000);
 OutString("F0K");
 Delay(3000);
 ScanTemp();
  while(!kbhit())
  {
  time(&tstop);
  tused = difftime(tstop,tstart);
  if (tused >= count)
     ScanTemp();
  }

  rewind(tempfile);
  fclose(tempfile);
 } /* of main */

void ScanTemp(void)
{
  time(&tstart);
  OutString("YC1");
  GetTemp();
  OutString("YC2");
  GetTemp();
  OutString("YC3");
  GetTemp();
```

```
  OutString("YC4");
  GetTemp();
  OutString("YC5");
  GetTemp();
  OutString("YC6");
  GetTemp();
  OutString("YC7");
  GetTemp();
  OutString("YC8");
  GetTemp();
} /* ScanTemp */

/************************************************************
* TempTime gets the time the temperature was read.          *
************************************************************/
void TempTime(void)
{
 /*time(&tnow);
 tmnow = localtime(&tnow);

 strftime(timestr,6,"%H:%M,",tmnow); */
 _strtime(timestr);

} /* TempTime */

/************************************************************
* CvtTemp extracts the channel number and temp read.        *
************************************************************/
void CvtTemp(void)
{
 chanstr = strtok(input,tokensep);
 tempstr = strtok(NULL,tokensep);
} /* CvtTemp */

/************************************************************
* GetTemp asks the Lakeshore controller for the temperature, *
* asks the PC for the time and writes it out to the file     *
************************************************************/

void GetTemp(void)
  {
    Delay(7000);
    OutString("WS");
    InString(input);
    /* printf(input); */
    CvtTemp();
    TempTime();
    fprintf(tempfile,"%s,%s,%s\n",timestr,chanstr,tempstr);
    printf("%s,%s,%s\n",timestr,chanstr,tempstr);
  }

/************************************************************
* Delay takes the number of milliseconds to wait.           *
************************************************************/

void Delay( clock_t wait )
 {
    clock_t t1, t2;
    if( !wait )
      return;
    t1 = wait + clock();
    do
```

```
    {
      t2 = clock();
    } while( t2 < t1 );
 }

/*************************************************************
* InitSerial opens the selected COM port and sets it to     *
* 300,N,8,1. Returns a -1 if an error is detected.          *
*************************************************************/
int  InitSerial(unsigned port,long Baud, int Parity,int DataBits,int StopBits)
  {
   char         Current_Value;
   int          divisor;
   int          setting;

   switch (port)
    {
     case COM1: portbase = COM1BASE;
            break;
     case COM2: portbase = COM2BASE;
            break;
     case COM3: portbase = COM3BASE; /* for future expansion */
            break;
     case COM4: portbase = COM4BASE; /* for future expansion */
            break;
     default  : return(-1); /* Bruce! you broke it! */
    }
   /* set baud rate divisor for UART */
  divisor = (int)(115200L/Baud);
  Current_Value = inp(portbase + LCR); /* read in current UART settings */
  outp(portbase + LCR,(Current_Value | DLAB));
  outp(portbase + DLL,(divisor & 0x00FF)); /* load in baud rate divisor */
  outp(portbase + DLH,((divisor >> 8) & 0x00FF));
  outp(portbase + LCR,Current_Value);

  setting = DataBits - 5;
  setting |= ((StopBits == 1) ? 0x00 : 0x04);
  setting |= Parity;
  outp(portbase + LCR,setting); /* was 03 set parity, data bits and stop bits
*/
  outp(portbase + IER,0x00); /* no interrupts */
  return(0);
  }

/*************************************************************
* InSerial reads data from the port in portbase. It returns *
* a zero (0) if no character is available, or the received  *
* character.                                                *

*************************************************************/
unsigned char InSerial(void)
  {
  long dell = 0xFFFFFFFF;

    while((inp(portbase + LSR) & RCVRDY) == 0)
      {
       if (!(dell--))
       return(-1);
       }
   return(inp(portbase + RXR));
  }
```

```
/************************************************************
* OutSerial sends a character out through the serial port.  *
************************************************************/
unsigned char OutSerial(char data)
  {
  /* wait for transmitter to be ready */
   while((inp(portbase + LSR) & XMTRDY) == 0x00);
   outp(portbase + TXR, data);
  }

/************************************************************
* LakeStat returns the status of the serial port.           *
* It returns a 1 if busy and a 0 if not busy.               *
************************************************************/
unsigned char LakeStat(void)
 {
   if ((inp(portbase + MSR)) != 0x00)
      return(1);
   else
      return(0);
 }

/************************************************************
* OutString sends a character string out through the serial *
* port. It calls OutSerial repeatedly until it encounters   *
* the string termination character (0x00). This routine     *
* does not return any status info.                          *
************************************************************/
void OutString(char *str)
  {
   unsigned char SErr;
    int i = 0;
    while(str[i] != 0x00)
       SErr = OutSerial(str[i++]); /* ignore errors */
  }

/************************************************************
* InString reads data from the serial port until it sees a  *
* CR. The incoming data is collected into the string passed *
* to the routine, including the CR. The string is NULL      *
* terminated. The routine itself does not return any status *
* codes.                                                    *
************************************************************/
void InString(char *str)
  {
    int i = 0;
    char inch = 0x00;  /* NULL character */

     while(inch != 0x0d)  /* wait for a terminating CR */
      {
       inch =  InSerial();
      if (inch != 0x0a)
        str[i++] = inch;
      }
   str[i] = 0x00;
  }
```