Application Note
April 2011

Serial Command Interface

Rev 01.201

Greenliant Systems
Proprietary and Confidential
04/07/2011

NDA Required

Application Note
April 2011

Disclaimer

By downloading and using these files, you agree:

01.. The information delivered hereunder (“Information”) is confidential to Greenliant Systems. You may use the Information only to generate or produce a reference PCB (“Product”), or a Modified Product, as defined below. The Information is delivered as is. Greenliant Systems makes no warranty express or implied, with respect to the use, merchantability, or fitness for purpose of the Information, the Product or the Modified Product. Further, Greenliant Systems disclaims any patent infringement liability arising out of, resulting from, or in connection with the use of the Information, the Product, or the Modified Product, or the use of the Information, the Product or the Modified Product, in combination with any other component, products or process. In no event shall Greenliant Systems be liable for any incidental or consequential damages with regard to the Information, the Product, or the Modified Product.

2. You will not sell or otherwise commercially exploit the Product. You will have the right to use the Information to generate or produce a reference PCB in which at least all references to Greenliant Systems markings, logos and part numbers are removed (“Modified Product”), and have the right to commercially exploit the Modified Product.

3. The unauthorized use, disclosure or duplication of the Information or the Product will result in irreparable harm to Greenliant Systems for which legal damages would be an inadequate remedy. Accordingly, in the event of unauthorized use, disclosure or duplication, Greenliant Systems will be entitled to injunctive relief in addition to any other rights or remedies it may have at law or equity.

4. All Intellectual Property rights residing or subsisting in the Information or the Product belongs to Greenliant Systems.

Table of Contents

Application Note
April 2011

Revision History

Revision	Description	Date
01.00	Initial Release	06/14/ 2010
01.10	Changed the OSC frequency for specifying the product dependency	06/16/2010
01.200	Updated the format Corrected the typos	04/07/2011
01.201	Correct the type	04/08/2011

Serial Communication Interface

Application Note
April 2011

1. INTRODUCTION

Greenliant Systems' NAND controllers and NANDrive products are well suited for embedded storage applications and provide the industry with versatile building blocks for silicon based storage subsystem products. Each Greenliant Systems NAND product features a Serial Communication Interface (SCI) which, when connected to an SCI Module, is used for product development, customization, and debugging. The SCI Module is a complementary tool to assist in product design and development.

The SCI uses the industry standard RS232 protocol which allows two-wire data transfer capable of full duplex channel communication. Using the SCI Module described in this application note, connect the SCI port of an Greenliant Systems based target application to a standard RS232 port on a personal computer (PC). The PC then communicates with the target device firmware.

Greenliant Systems recommends using the SCI Module for all product designs to aid in development and debugging, and to reduce time-to-market. For NAND Controller or NANDrive devices, Greenliant Systems recommends that all designs have the SCI port signals brought out to an easily accessible connector, test points, or pads. This application note describes a complete system design example of the SCI.

2. SCI SYSTEM INTERFACE

The SCI Module provides appropriate voltage conversion and connection between the PC serial port and the SCI input/output pins of the NAND controller or NANDrive. SCI Module power is provided by the target application board. See Figure 1 for the system block diagram.

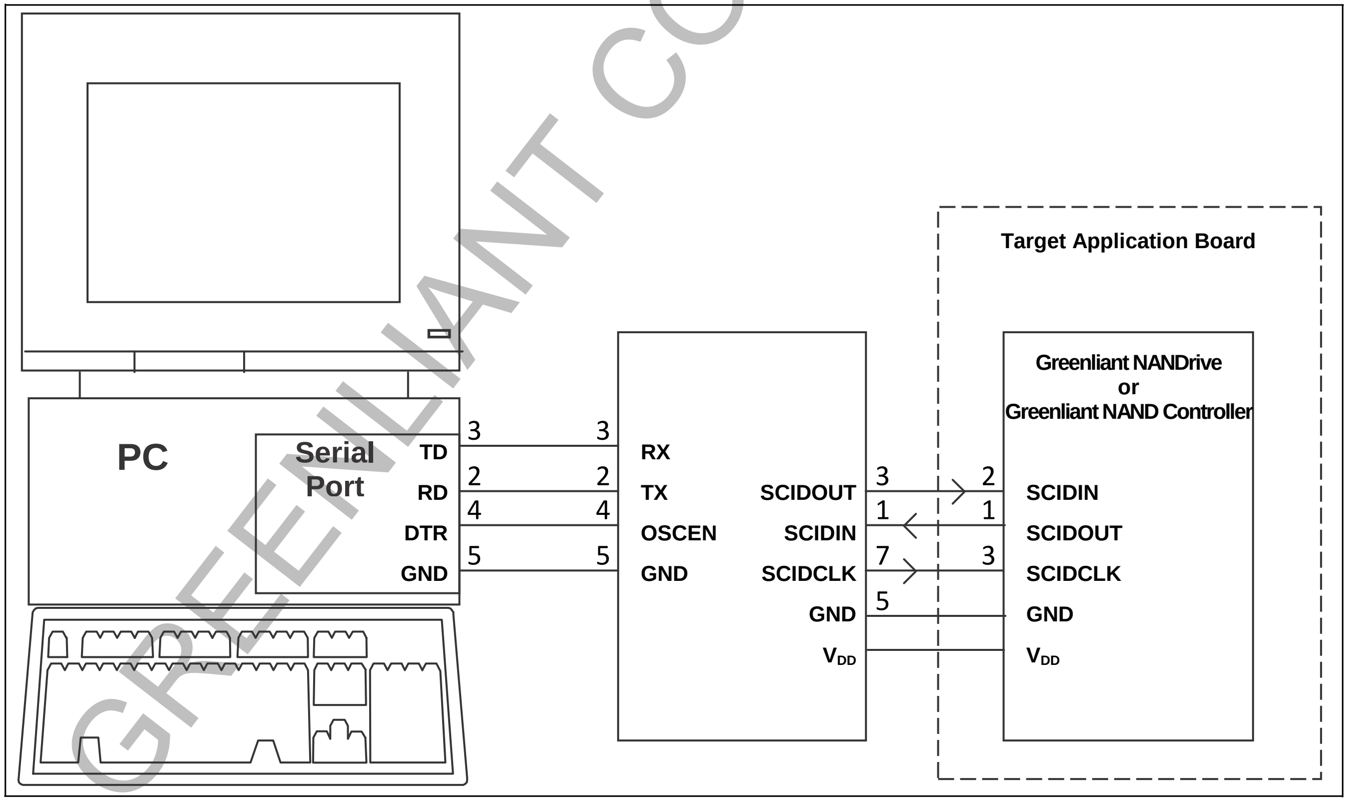

Figure 1: System Block Diagram

Table 1: Pin Description (Please refer Figure 2)

Symbol	Description	Functions
J1	DB9 Connector	Connects the RS232 cable between the PC's serial port and the SCI Module
U1	MAX3232	Contains two pairs of transmitters and receivers. The internal dual charge-pump voltage converter makes these pairs capable of operating on two voltage levels.
J2	Power Connector	Brings 3.3V from target board, same as VDD of NAND Controller
U3	Oscillator	Provides the driving clock for the SCI clock interface to the ATA Flash Disk Controller
U4A/U4B	Inverters	Inverts the bus signal
U4C-U4F	Inverters	Reserved for future use

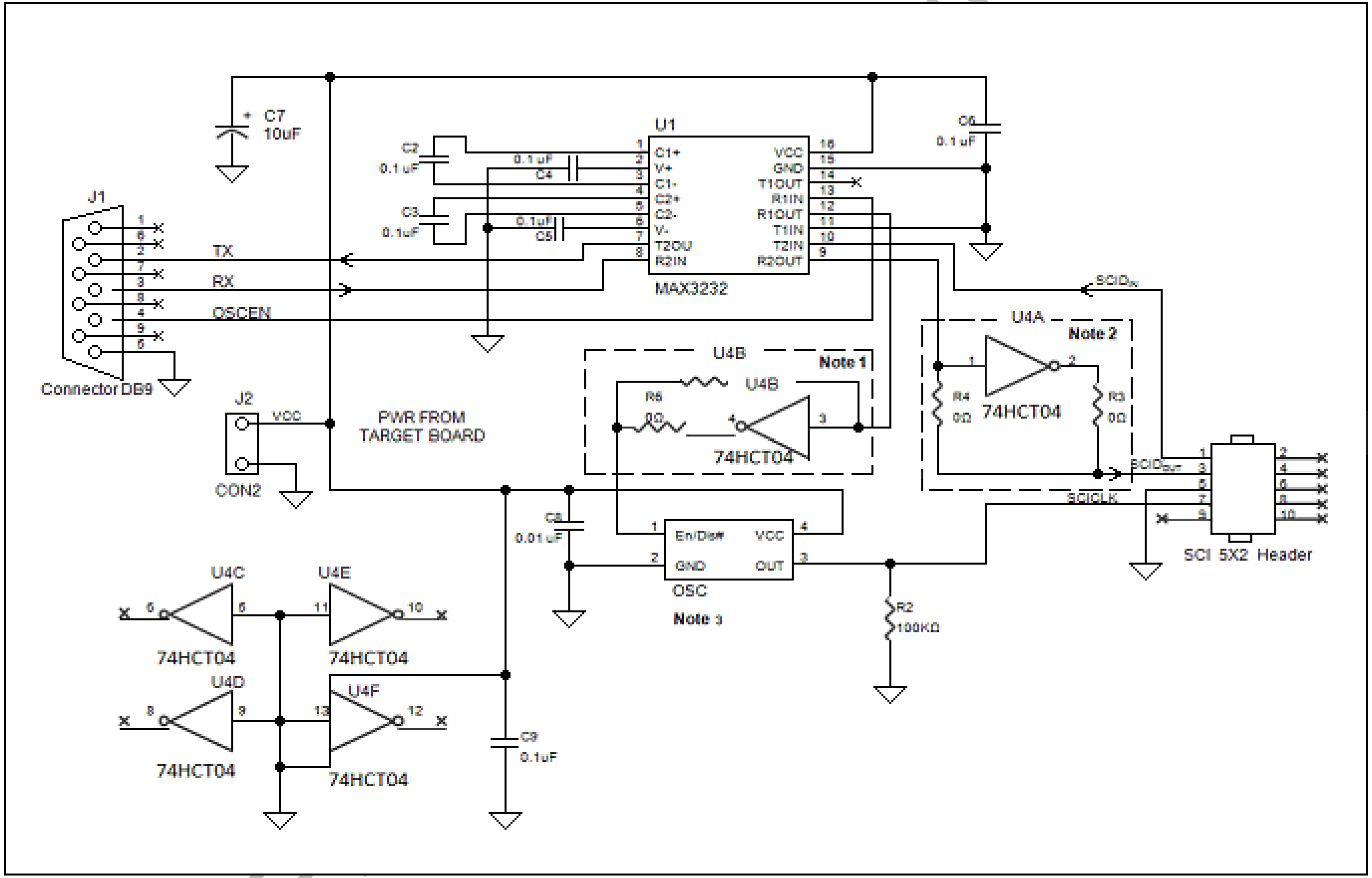

Figure 2: Greenliant Systems SCI Module Schematic

1) Connect either R5 or R6. Choosing R6 will reverse the logic state of OSCEN signal.
2) Connect either R3 or R4. Choosing R3 will reverse the logic state of SCIDOUT signal.
3) OSC frequency is 4.915MHz for 55LD019x, 55VD 020, 55VD031 and 85LDxxxx products; OSC frequency is 11.059MHz for 85LPxxx and 85LSxxx products.

2.1. SCI Configuration

Four steps are required to configure the NAND Controller or NANDrive SCI port.

a) Connect the personal computer serial port to the SCI Module serial connector using an RS-232 cable.

b) Once connected, set the serial communication protocol to the appropriate value (Please refer Table 2).

c) Connect the $SCID_{OUT,}$ $SCID_{IN,}$ and SCICLK pins of the SCI Module to the $SCID_{IN,}$ $SCID_{OUT,}$ and SCICLK pin on the NAND Controller or NANDrive (Please refer Figure 1).

d) Attach 3.3V power and ground wires to both the SCI Module and the target application board.

After completing the setup, the PC host program communicates directly with the NAND Controller to perform multiple tasks (Please refer 2.2. SOFTWARE).

Table 2: Serial Communication Protocol Values[4)]

Product	Values[5)]
55LD019x	9600 baud
55VD020	115K baud
55VD031	115K baud
85LDxxxx-120-x	9600 baud
85LDxxxx-60-x	115200 baud
85LPxxxx	115200 baud
85LSxxxx	115200 baud

4) The typical communication setting is 8 bits, no parity, 1 stop bit (8N1). Other possible settings are 8O1 and 8E1.
5) The correct baud rate value depends on the internal controller specific to each NAND Controller product.

2.2. SOFTWARE

The SCI source code includes these subroutines:

InitComPort	Save the current port settings, set the baud rate, odd parity, one stop bit, and set the port to selectable (3F8 or 2F8)
SendChar	Send one data byte
ReceiveChar	Receive one data byte
RestoreCom	PortRestore the port settings
sciEnable	Enable the clock on the ATA controller SCI Module
sciDisable	Disable the clock on the ATA controller SCI Module

The examples provided in this application note are compiled using a General ANSI C compiler for the IBM AT/XT PC platform, and are provided only as reference for system designers.

2.3. Initialization Procedure

Here is the procedure to use the SCI port on the host computer:

Step 1: Initialize the port.
Step 2: Enable the clock if using the ATA controller SCI Module.
Step 3: Send or receive data.
Step 4: Disable clocks and restore port settings before program ends

2.4. Data Send/Receive

While the SCI can both send or receive data, the primary function of this interface is to receive data from the target device and display status, configuration, function, and statistical information. The specific data received varies by the particular Greenliant Systems target device. Contact your Greenliant Systems representative for detailed information for specific Greenliant Systems devices.

2.5. Source Code Listing

The example source code listed in this Section is provided for user reference only—although validated in a Greenliant Systems test platform, the provided source code may require modification for other platforms. For additional information, please contact Greenliant Systems technical support at TechSupport@Greenliant Systems.com.

```
/*---------------------------------------------------------------------------
Greenliant Systems, Inc.
This communication program provide functions of send/receive data byte to/from serial port.
The communication port, baud rate, parity, stop bits, acknowledgement enable/disable and time-out wait set are assumed to be already
initialized.
functions:
---------------------
ReceiveChar         Receive one byte of data
SendChar            Send one byte of data
InitComPort         Communication port initialization
RestorComPort       Restore previous communication port's setting
---------------------------------------------------------------------------*/

    #include <stdio.h>
    #include <conio.h>
    #include <bios.h>
    #include <dos.h>
    #include <string.h>
    #include <stdlib.h>

    //Serial
    #define RegIntEnable                      (AuxBase + 1)
    #define RegIntIdentification              (AuxBase + 2)
    #define RegLineCtrl                       (AuxBase + 3)
    #define RegModemCtl                       (AuxBase + 4)
    #define RegLineStatus                     (AuxBase + 5)
    #define RegModemStatus                    (AuxBase + 6)
    #define RegDivisorLo                      (AuxBase)
    #define RegDivisorHo                      (AuxBase + 1)

    #define ODD_PARITY                          0
    #define EVEN_PARITY                         1
    #define NO_PARITY                           2

    #define LINESTATUS_DATARCVD               0x01
    #define LINESTATUS_OVERRUNERROR           0x02
    #define LINESTATUS_PARITYERROR            0x04
    #define LINESTATUS_FRAMINGERROR           0x08
    #define LINESTATUS_BREAKERROR             0x10
    #define LINESTATUS_HOLDREADY              0x20
    #define LINESTATUS_SHIFTREADY             0x40

    #define LINECTRL_5BITS                    0x00
    #define LINECTRL_6BITS                    0x01
    #define LINECTRL_7BITS                    0x02
    #define LINECTRL_8BITS                    0x03

    #define LINECTRL_EVENPARITY               0x10
    #define LINECTRL_ODDPARITY                0x00
    #define LINECTRL_BREAKENABLE              0x40
    #define LINECTRL_BREAKDISABLE             0x00
    #define LINECTRL_DIVISORACCESS            0x80
    //Communication error codes

    #define OVERRUN_ERROR                 0x10
    #define FRAMING_ERROR                 0x11
    #define BREAK_ERROR                   0x12
    #define PARITY_ERROR                  0x13
    #define USER_INTERRUPT                0x14
    #define OUTOFTIME_ERROR               0x15
    #define ACK_ERROR                     0x16

    #define FALSE                         0
    #define TRUE                          1
    #define SUCCESS                       0
    #define FAIL                          1
    #define YES                           1
    #define NO                            0

    unsigned char OldBaudHigh, OldBaudLow, OldLineCtrl, OldIntEnable;

    unsigned int AuxBase, AckEnable;
    unsigned int RecTimeOut=1000, SendTimeOut=100; //in ms

    //To be used with Greenliant Systems SCI Module void sciEnable(void)
    {
    outp(RegModemCtl, 0x1); //bit 0 of port 0x2fc or 0x3fc
    }

    void sciDisable(void)
    {
    outp(RegModemCtl, 0x0);
    }

/"----------------------------------------------------------------------------
Name:              Receive Char
Function:          Receive one character from serial port; check time-out
Parameters:        CharData = Address to store the data received
Return:            Error code
----------------------------------------------------------------------------*/
    unsigned int ReceiveChar(unsigned char *CharData)
    {
    unsigned int LineStatus;
    unsigned long i=0;

    do{
        LineStatus = inp(RegLineStatus);
    i++;
    if(i>=RecTimeOut) return OUTOFTIME_ERROR;
    }while(!(LineStatus & LINESTATUS_DATARCVD));

    *CharData = inp(AuxBase);

    return SUCCESS;
    }
```

```
/*------------------------------------------------------------------------------
Name:               SendChar
Function:           Send one character to the serial port
Parameters:         CharData   = Data to be send
Return:             error code
------------------------------------------------------------------------------*/
   unsigned int SendChar(unsigned char CharData)
   {
   unsigned int LineStatus;
   unsigned long i, j;

   i = 0;
   do{
      LineStatus = inp(RegLineStatus);
   delay(1);
   i++;

   if(i>=SendTimeOut) return OUTOFTIME_ERROR;
      }while((LineStatus & (LINESTATUS_HOLDREADY | LINESTATUS_SHIFTREADY))!=0x60);
   delay(1);
   outp(AuxBase, CharData);

   i = 0;
   do{
      LineStatus = inp(RegLineStatus);
   delay(1);
   i++;

   if(i>=SendTimeOut) return OUTOFTIME_ERROR;
      }while((LineStatus & (LINESTATUS_HOLDREADY | LINESTATUS_SHIFTREADY))!=0x60);

   return SUCCESS;
   }

/*------------------------------------------------------------------------------
Name:               InitComPort
Function:           Initialize the communication port
Parameters:         ComPort = Communication port number (1 or 2)
BaudRate = Baud rate
Parity = Parity type
Return:             None
------------------------------------------------------------------------------*/
   void InitComPort(unsigned int ComPort)
   {
   unsigned int BaudSet;
   unsigned int ParityStatus;
   unsigned int Parity    = 2;                    //no parity
   unsigned int StopBits  = 1;                    //1 stop bit
   unsigned long BaudRate = 15200;

   ParityStatus = (Parity == NO_PARITY) ?
      LINECTRL_PARITYDISABLE : LINECTRL_PARITYENABLE;

   Parity = (Parity  == EVEN_PARITY) ?
      LINECTRL_EVENPARITY : LINECTRL_ODDPARITY;

    StopBits = (StopBits == 1) ?
      LINECTRL_1STOPBIT : LINECTRL_2STOPBITS; BaudSet = (unsigned int)(((unsigned long)115200) / BaudRate);

   AuxBase = (ComPort == 1)? 0x3f8 : 0x2f8;

   // Save line control register
      OldLineCtrl = inp(RegLineCtrl);

   // Set divisor latch access bit
      outp(RegLineCtrl, LINECTRL_DIVISORACCESS);

   // Save baud rate low
      OldBaudLow = inp(RegDivisorLo);

   // Set baud rate low
```

```
        outp(RegDivisorLo, (int)BaudSet);
    // Save baud rate high
        OldBaudHigh = inp(RegDivisorHo);

    // Set baud rate high
        outp(RegDivisorHo, 0);

    // 1 stop bit,8 bits
        outp(RegLineCtrl, ParityStatus|
                Parity |
                LINECTRL_8BITS|
                StopBits);

    // Clear line status register
         inp (RegLineStatus);

    // Clear receive buffer
        inp (AuxBase);

    // Save interrupt enable
        OldIntEnable = inp(RegIntEnable);

    // disable interrupt
        outp(RegIntEnable, 0);
    }
```

```
/*-----------------------------------------------------------------------------
Name:           RestoreComPort
Function:       Restore communication port set
Parameters:     None
Return:         None
-----------------------------------------------------------------------------*/
    void RestoreComPort(void)
    {
    // Set divisor latch access bit
        outp(RegLineCtrl,  LINECTRL_DIVISORACCESS);

    // Baud rate low
        outp(RegDivisorLo, OldBaudLow);

    // Baud rate high
        outp(RegDivisorHo, OldBaudHigh);

    // Line control register
        outp(RegLineCtrl,  OldLineCtrl);

    // Interrupt enable register
        outp(RegIntEnable, OldIntEnable);
    }
```