INTRODUCTION

Greenliant’s 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 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 a Greenliant based target application to a standard RS232 port on a personal computer (PC). The PC then communicates with the target device firmware.

Greenliant 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 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.

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.
TABLE 1: Pin Description

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

1. See Figure 2

FIGURE 2: Greenliant SCI Module Schematic

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

1. Connect the personal computer serial port to the SCI Module serial connector using an RS-232 cable.
2. Once connected, set the serial communication protocol to the appropriate value. See Table 2.
3. Connect the SCIDOUT, SCIDIN, and SCICLK pins of the SCI Module to the SCIDIN, SCIDOUT, and SCICLK pin on the NAND Controller or NANDrive. See Figure 1.
4. 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. See “SOFTWARE” on page 3.

<table>
<thead>
<tr>
<th>Product</th>
<th>Values²</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLS55LD019x</td>
<td>9600 baud</td>
</tr>
<tr>
<td>GLS55VD020</td>
<td>115K baud</td>
</tr>
<tr>
<td>GLS55VD031</td>
<td>115K baud</td>
</tr>
<tr>
<td>GLS85LDxxxx-120-x</td>
<td>9600 baud</td>
</tr>
<tr>
<td>GLS85LDxxxx-60-x</td>
<td>15200 baud</td>
</tr>
</tbody>
</table>

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

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
  Port Restore 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.

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.
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 target device. Contact your Greenliant representative for detailed information for specific devices.

Source Code Listing

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

/*================================================================--------------
©2010 Greenliant Systems, Ltd. All Rights Reserved.
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_1STOPBIT 0x00
#define LINECTRL_2STOPBITS 0x04
#define LINECTRL_PARITYENABLE 0x08
#define LINECTRL_PARITYDISABLE 0x00
#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 SCI Module
void sciEnable(void)
{
    outp(RegModemCtl, 0x1); // bit 0 of port 0x2fc or 0x3fc
}

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

/*----------------------------------------------------------------------------
Name: ReceiveChar
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);
        #define RECEIVECHAR
        #define SUCCESS 0
        #define FAIL 1
        #define YES 1
        #define NO 0
        }
        
unsigned int TemporaryChar;
    
    while(LineStatus & 0x01) // receive a character
        
    CharData[0] = TemporaryChar;
    
    if((RecTimeOut-- == 0))
        
    return FAIL;
    
    return SUCCESS;
}
Application Note

```c
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
-----------------------------------------*/

```c
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
-----------------------------------------*/

```c
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);
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);
}