Fall 1998
Computer Science 483

Scott Mauer
Glen L. Longley
Heather Jessel
Wayne Beyar

By

A Senior Design Project

Generic Device Interface for the Serial Bus
Table of Contents
SCSI hard drive, and a simple AD converter with software that provides basic read and write functions. We planned to demonstrate this functionality by implementing read and write to a personal computer.

By creating a generic interface, we allow users to connect any personal computer. This allows asynchronous communication interface adapter to interface between the FPGA and the software running on a personal computer. The serial communication uses Xilinx module allows for simple users of Xilinx to have send and receive this in a device-on style.

This project was to develop a generic device interface for the serial bus. A Xilinx

1. Abstract
<table>
<thead>
<tr>
<th>Stop Bit</th>
<th>One or two stop bits can be used to inform the receiver that transmission is complete.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity Bit</td>
<td>A parity bit can be set for error detection in situations where data is prone to be errored.</td>
</tr>
<tr>
<td>Data Bit</td>
<td>The data bits are the actual data that is found on the line. When sending binary information, 8 bits are used.</td>
</tr>
<tr>
<td>Start Bit</td>
<td>The start bit is used to alert the receiver that information is coming. If changes the data line from high to low to begin.</td>
</tr>
</tbody>
</table>

Table showing breakdown of serial communication.

---

**A Serial Background**

II. Serial to Xilinx
3. ACIA and UARTs

In order to do anything useful with our serial transmitted data, we must convert it to a useful form. One example of the useful form is an example circuit using a MAX-232 chip. The circuit is shown in the appendix. It consists of two chips: an ACIA chip, which allows the chip to communicate with the host, and a UART, which allows the chip to communicate with the radio.

2. AX232 Line Driver

The MAX-232 chip is connected to the AX232 line driver, which provides the necessary power to the chip. The output of the chip is a TTL signal, and the AX232 line driver converts this signal to a +5V level signal, which is then used to drive the radio.

1. RS-232 Level Converters

The AX232 line driver is connected to the RS-232 level converters, which convert the +5V signal back to a TTL level signal.

In summary, the AX232 line driver is used to drive the radio, and the RS-232 level converters are used to convert the +5V signal back to a TTL level signal.
<table>
<thead>
<tr>
<th>Division Ratio</th>
<th>1</th>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Division Ratio - 16</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Control Register Bit Assignments:

RS = 0, R/W = 0 can be set according to the following table:

Use the RS and R/W inputs to the 6850 as shown above. The control register.

<table>
<thead>
<tr>
<th>Receive Data Register</th>
<th>Read</th>
<th>Write</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit Data Register</td>
<td>Read</td>
<td>Write</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status Register</td>
<td>Read</td>
<td>Write</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Register</td>
<td>Read</td>
<td>Write</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register Address Scheme:

Need to Vcc, leaving just C2S to win the chip on or off.

The three chip select pins can be used in decoding circuits. CS0 and C31 normally are
are used to select the bus free registers. The Chip Select pins (CSA, CS1, and C31) are
are used to select from the control registers. The control register (RS) and read-write (R/W)
are used to select when it is ready to send data. The receive register (R0) and receive when it
is recorded in memory when it has recorded.

How important that is used to show it is ready to accept data. The interrupt request (IR0) in
low priority will interrupt the microprocessor when it is ready. As in the interrupt
acquire-low output that signifies that the 6850 is ready to transmit data. CS is an active
in active-low output. The status pins are used to handshake signals.

Requests to send (RTS*) and clear to send (CTS*) are used as handshake signals.

Register: Status Register, Control Register, and Receive Data Registers.

This and a data controller detect line. The 6850 contains four registers: a transmission data
transmission, a register select, a receive data, and a receive when ready to send
the receive when ready to send.

The basic structure of the 6850 uses an 8-bit bi-directional data bus, an enable
9600bps.

4. MC6850 Asynchronous Communications Interface Adapter
<table>
<thead>
<tr>
<th>Status Register Bit Description</th>
<th>Description of the Individual Status Bits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTS (Clear to Send)</td>
<td>Set if incoming data is ready, clear if no incoming data is ready.</td>
</tr>
<tr>
<td>DTR (Data Terminal Ready)</td>
<td>If the DTR line is set, the modem is ready.</td>
</tr>
<tr>
<td>RI (Ring Indicator)</td>
<td>If the RI line is set, the modem is ringing.</td>
</tr>
<tr>
<td>RI5 (High)</td>
<td>If the RI5 line is set, the modem is in high-speed mode.</td>
</tr>
<tr>
<td>RI5 (Low)</td>
<td>If the RI5 line is low, the modem is in low-speed mode.</td>
</tr>
<tr>
<td>RI5 (Disabled)</td>
<td>If the RI5 line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RXD (Receive Data)</td>
<td>If the RXD line is set, the modem is receiving data.</td>
</tr>
<tr>
<td>TXD (Transmit Data)</td>
<td>If the TXD line is set, the modem is transmitting data.</td>
</tr>
<tr>
<td>XON (XON)</td>
<td>If the XON line is set, the modem is in XON mode.</td>
</tr>
<tr>
<td>XOFF (XOFF)</td>
<td>If the XOFF line is set, the modem is in XOFF mode.</td>
</tr>
<tr>
<td>XON (Disabled)</td>
<td>If the XON line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>XOFF (Disabled)</td>
<td>If the XOFF line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>MMR (Modem Mode Register)</td>
<td>Set if the modem is in the modem mode.</td>
</tr>
<tr>
<td>DSR (Data Set Ready)</td>
<td>If the DSR line is set, the modem is ready.</td>
</tr>
<tr>
<td>DSR (Disabled)</td>
<td>If the DSR line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>RTS (Request to Send)</td>
<td>If the RTS line is set, the modem is ready to send data.</td>
</tr>
<tr>
<td>RTS (Disabled)</td>
<td>If the RTS line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>DCD (Data Carrier Detect)</td>
<td>If the DCD line is set, the modem is detecting data.</td>
</tr>
<tr>
<td>DCD (Disabled)</td>
<td>If the DCD line is disabled, the modem is in disabled mode.</td>
</tr>
<tr>
<td>Line 10 Indicate in intent.</td>
<td>Interim Request</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Party checked. If party is unable this contains the results of the</td>
<td>60 Party Error</td>
</tr>
<tr>
<td>prior to the last received are lost. All plates received before the first plate was read. All plates received than the more than one plate was</td>
<td>(ORN) Owner Receiver</td>
</tr>
<tr>
<td>absence of stop block. It is changed for each plate</td>
<td>(FPE) Framing Error</td>
</tr>
</tbody>
</table>
The two 10-bit counters are connected to provide the clock for the counter and a reset


2. Xilinx Module

- The Xilinx controller and the FPGA, DCD* and CSL are not used in the prototype.
- The Xilinx controller is used on the Xilinx to provide for bidirectional communication between the FPGA and the MAX-232 ports. The Xilinx is connected to the 10-bit counters. The data bus is connected to the FPGA and the CSOL. The Xilinx is also used to connect the FPGA to the MAX-232 ports. The MAX-232 ports are used to connect the FPGA to the Xilinx controller.

1. MAX-232 and 6850

C. Design

The 10-bit counters are connected to provide the clock for the counter and a reset.
because the readable basic connectivity was never established through the 6850 between Xilinx and the PC. This was never implemented in hardware and the 6850 output from the FIFO. This would then communicate read and writes from the second design in Xilinx utilizing a state machine to poll the SR on the 6850 was not the expected results, considering a stream of multiplex data, the data changed when a new character was sent, but attempting to send "HELLO" across the 6850 to a terminal program on the PC sent a that was being sent across the bus was not proper. Using a fast enable clock the results of output from the Xilinx was correct, a proper baud rate was being generated, but the data polled by the state machine was sent to the 6850 the data in the ROM.

3. Results

the LED for debugging, changed the RS and R/W values to enable a read from the receive register and wrote to the PC polled the SR from the 6850 and checked for the receive register full bit. If then
III. Small Computer System Interface (SCSI)

A. SCSI Background

1. Basic SCSI Principles

B. SCSI Bus Configurations

2. Multiple SCSI Disks

C. SCSI Physical Characteristics

3. SCSI Cables and Interconnections

D. Connection of Devices

E. SCSI Terminology

F. SCSI Bus Operations

G. SCSI Protocol

H. SCSI Enhancements

IV. SCSI Implementations

A. SCSI Front-Ends

1. SCSI Host Adapters

B. SCSI Abstraction Layers

C. SCSI Target Adapters

V. Advanced SCSI Concepts

A. RAID (Redundant Array of Independent Disks)

1. RAID Levels

B. SCSI Over Ethernet

C. SAN (Storage Area Network)

D. iSCSI (Internet Small Computer System Interface)
is indicated when this signal is asserted, active or not.

The CD signal is driven by the target and indicates the type of

select phase and by the initiator to select an initiator during the reselection phase.

The select signal is used by the initiator to select a target during the

is in use.

and information/selector action. When active, this signal indicates that the SCSI bus
begins an information/selector action, or by the target during an
device which is gain access to the bus. This signal is driven by an initiator to

The BUSY signal is an OR-logic signal that can be driven active by any

depending upon the system configuration. The parity bit is not valid during the

use of the DBP parity signal is optional and it can be enabled or disabled

DB0 the lower

DB0 through DB7 and DBP signals form the DATA BUS. After for

specification defines the SCSI bus signals as follows:

when the initiator controls the other. All 18 signals are active low. The ANSIT SCSI-I
control the flow and direction of the data bus. The initiator controls some of the signals
are used to indicate the various phases of the bus protocol and to

data signals are 2-bit bidirectional data bus, which includes 8 data plus 1 parity bit.

There are nine data signals and nine control signals on the SCSI bus. The nine

Bus Signals
The initialization routine of both may drive the SCSI signals. The table below shows

7. Signal Sources

Some devices can be driven using either OR-Red, open collector drivers, or these-

simultaneously driven above by several devices. Signals other than the BSY and

the mandatory OR-Red condition recognition allow these signals to be

The BSY and OR-Red signals are the only signals required to be OR-Red.

Device on the bus and indicates a REQEST condition.

The ATN signal is OR-Red and can be driven active by any

(ATTENTION)

Acknowledgment of the REQ/ACK data transfer handshake to the larger

The ACK signal is driven active by the initiator and indicates an

(REQ/ACKNOWLEDGE)

Acknowledged with a REQ/ACK handshake.

The data transfer request is

initiated for data transfer over the data bus. The data transfer request is

The REQ signal is driven active by the target and indicates a request to the

(REQ REQUEST)

phase is continuing.

The MSG signal is driven active by the target and indicates that a message

(MESSAGE)

Selective a target, when it is asserted, a target is rescanning an initiator.

Operation is being placed. When the I/O is negated or inactive, an initiator is

The I/O signal is also used to indicate if a selection or rescanning

bus and the target is receiving

An inactive signal indicates that the initiator is writing to the

following into the initiator. In the initiator is reading from the data bus and

data, over the data bus. An active, asserted or high signal indicates that data

The I/O signal is driven by the target and indicates the direction of data
establish a physical path and complete a previously requested operation. The

data

result of the SCSI bus. The select phase is when an Initiator is writing to select an Initiator in a

specific target or a SCSI device. The select phase is when and a physical path is established between the two

SCSI consists of a series of bus phases: initialization, selection, selection, and

data transfer. The initialization phase is where multiple SCSI devices are notified to join

SCSI bus phases:

1. Message Out
2. Status
3. Data Out
4. Data In
5. Command
6. Re-select
7. Bus Free

<table>
<thead>
<tr>
<th>Notes</th>
<th>Message Out</th>
<th>Status</th>
<th>Data Out</th>
<th>Data In</th>
<th>Command</th>
<th>Re-select</th>
<th>Bus Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Notes 1, 4, 6)</td>
<td>(Notes 7, 8, 9, 10)</td>
<td>(Notes 16)</td>
<td>ACK/ATN</td>
<td>CB/IO, MO</td>
<td>SET</td>
<td>RE</td>
<td>BS</td>
</tr>
</tbody>
</table>

signaling
Transfer Phase

When it encounters unusual time-consuming events, the RESOLUTION phase allows a target to disconnect from the bus.

RESOLUTION Phase

Perform a SCSI operation, the SELECTION phase allows an initiator to select a target device to simultaneously contest for ownership for the SCSI bus.

SELECTION Phase

The ARBITRATION phase allows multiple SCSI devices to is available for use by the SCSI devices.

ARBITRATION Phase

The BUS FREE phase indicates when the SCSI bus is in an idle state and

BUS FREE Phase

Message Phases. Phases five through eight are part of the data transfer phases described above. The SCSI bus can operate in up to four different phases (1) Bus FREE, (2) Data Phase, (3) Selection, (4) Reselection, (5) Command, (6) Datas, and (7) Staus and (8)

Bus Phase Descriptions
The DATA BUS.

The TARGET requests the INITIATOR to send a number of messages across

**MESSAGE OUT** phase

DATA BUS.

The TARGET requests the INITIATOR to send status information across the

**STATUS** phase

The TARGET requests commands from the INITIATOR

**COMMAND** phase

The TARGET requests the INITIATOR to receive bytes over the DATA BUS.

**DATA IN** phase

The TARGET requests the INITIATOR to send bytes over the DATA BUS.

**DATA OUT** phase

**Notes:**

- **0** = False; **1** = True
- **** = Reserved for future standardization

<table>
<thead>
<tr>
<th>Message Phase</th>
<th>Direction of Transfer</th>
<th>State Name</th>
<th>CMD</th>
<th>MSG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Out</td>
<td>INITIATOR to TARGET</td>
<td>MESSAGE OUT</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MESSAGE OUT</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MESSAGE OUT</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MESSAGE OUT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MESSAGE OUT</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MESSAGE OUT</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Message Phase**

- Message Out

**Direction of Transfer**

- INITIATOR to TARGET

**State Name**

- MESSAGE OUT

**CMD**

- **1**

**MSG**

- **1**
The TARGET requests to send data bytes over the DATA BUS in the
MESSAGE IN Phase.

INITIATOR
<table>
<thead>
<tr>
<th>EVEN PIN NUMBERS</th>
<th>SIGNAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>TERMPWR</td>
</tr>
<tr>
<td>24</td>
<td>GROUND</td>
</tr>
<tr>
<td>22</td>
<td>GROUND</td>
</tr>
<tr>
<td>20</td>
<td>GROUND</td>
</tr>
<tr>
<td>18</td>
<td>-DB (P)</td>
</tr>
<tr>
<td>16</td>
<td>-DB (7)</td>
</tr>
<tr>
<td>14</td>
<td>-DB (6)</td>
</tr>
<tr>
<td>12</td>
<td>-DB (5)</td>
</tr>
<tr>
<td>10</td>
<td>-DB (4)</td>
</tr>
<tr>
<td>8</td>
<td>-DB (3)</td>
</tr>
<tr>
<td>6</td>
<td>-DB (2)</td>
</tr>
<tr>
<td>4</td>
<td>-DB (1)</td>
</tr>
<tr>
<td>2</td>
<td>-DB (0)</td>
</tr>
</tbody>
</table>

I, Configuration

Appendix

I designed that I used in my design. For detailed schematics and data machines, look in the

<table>
<thead>
<tr>
<th>TRACK TO SCSI</th>
<th>FROM SCISC</th>
<th>EXTERNSITY</th>
<th>I will discuss the configuration and block</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSI GROUND</td>
<td>GROUND</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B. SCSI Design
2. Hardware Design

The minus sign next to the signal designates active low.

<table>
<thead>
<tr>
<th>28</th>
<th>30</th>
<th>32</th>
<th>36</th>
<th>38</th>
<th>42</th>
<th>44</th>
<th>46</th>
</tr>
</thead>
<tbody>
<tr>
<td>-AVN</td>
<td>GROUND</td>
<td>-ATN</td>
<td>GROUND</td>
<td>30</td>
<td>32</td>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td>-RS1</td>
<td>-MSG</td>
<td>-SET</td>
<td>-CD</td>
<td>-RD</td>
<td>46</td>
<td>44</td>
<td>42</td>
</tr>
<tr>
<td>-I/O</td>
<td>-FEP</td>
<td>-FEP</td>
<td>-FEP</td>
<td>-FEP</td>
<td>-FEP</td>
<td>-FEP</td>
<td>-FEP</td>
</tr>
</tbody>
</table>

Notes: All odd pins except Pin 25, should be connected to ground. Pin 25 should be left open.
Below is the Command FSM that handles the read byte (or ROM) and it needs 10 do on asynchronous RST/ACK handshake for each byte.

When in the Command bus phase, the SCSI disk only needs to read 6 bytes from the

Bus Phase FSM:
The Phase[1:0] FSM takes care of data coming out of the FIFO to the SCSI disk. This FSM.

does a REQ/ACK handshake and is set up to only read one byte from the FIFO.

When data to the FIFO so there is a REQ/ACK handshake for the one byte that gets

The Phase[1:0] FSM handles the Data, Status, and Message phases. All of these phases
The fact that I was writing for the Hardware to be able to write to the port.

I believe that this loop would be adequate for one byte reads.

However, even though the implementation was straight forward, I have also had a

address

When a read control string was passed the disk would then return what data was in that

Write: 0x06 + Device ID (4 bytes) + Address (20 bits) + Length (1 byte) + Control (1 byte)

Read: 0x03 + Device ID (4 bytes) + Address (20 bits) + Length (1 byte) + Control (1 byte)

within the disk the following control string is passed:

created then the rest of the program deals with the SCSI interface. For the reading and

The first thing one must do is to create a "handle", which is how Windows keeps track of

Serial communication in Windows NT is straightforward and well documented.

IV. Software

When we decided to move options on using the serial bus in place of the USB, this

change has obviously affected the software side of the project, by making it easier to find

them, we needed to use our existing code base.

The original proposal was that we implement a program that could read and write
The device is implemented using a simple A/D converter with a thermocouple. This device uses the 0804 A/D converter chip to implement...
where \( C1 \) is the \( +3V \) buffer of the counter, the desired value appears on the counter. The number of bytes in the input FIFO is (16-Cy).

The switch is set to counter mode, which disconnects the counter switch from

**Output:**

- External clock input
- 8 TTL-level data lines from A/D converter

**Input:**

- 8 CMOS-level data lines for building and connection with Xilinx
- Decoupled manual clock signal
- 4 LED counter toggle value
- 1st-to-push-in to output FIFO in the form of 8 LEDs

The device consists of:

- Switch to allow manual setup of the counter (and thus the number of bytes in the Xilinx bus)
- 4 bit LED bank to view the counter value
- 8 LED bank to view the values in the last-byte-chip memory
- 4-bit binary counter (pseudostart in input FIFO)
- TTL-CMOS drivers to receive proper CMOS input levels to the Xilinx
- 8 D-latches (last five pushed)

The remaining inputs/outputs to the FIFOs are either not involved in the device was created to allow intervention and reprogramming to move forward when the

**Outputs:**

- None
- Clock push, data[7:0]
- V/D <-> Xilinx FIFO
- V/D <-> A/D FIFO
- Empty
- Pop
- Xilinx <-> V/D FIFO

This device was created for the purpose of enabling the portability of the Xilinx FIFO (see BOGOFIFO 16-bit FIFO emulator).
Once the desired input FIFO counter is asserted, the switch is set back into full connection mode, which connects the clock signal to everything. From this point on:

The counter value of the A/D converter lines is asserted and pushed onto the context.

The input FIFO is popped to remove the incoming data. This is modeled.

The input FIFO is popped to remove the incoming data. This is modeled.

In this circuit, the counter value is increased by one:

Once the counter reaches zero, further clocks are ignored until the counter

is manually reset. This represents the condition of the EMIPTY signal coming from the input FIFO. As long as this signal is high, the following happens each time the clock is asserted:

The output of the AND gate represents the EMIPTY signal coming from the input FIFO.
output FIFO as the key term.

Pushing bytes into the input FIFO at the desired frequency and reading the bytes from the
FIFO implementation are known data can be obtained at a consistent stream by simply
get as much as the input data size wants on demand. If the precise timing values of the
central value of the digital thermometer onto the output FIFO. This allows the user to
every line a byte is sent into the input FIFO, the device responds by pushing the

Usage:

When differences through the Xilinx module to the serial port or a computer
Bus final protocol FIFOs are a very flexible method of interfacing low complexity
This interface was created to allow modular development of devices for our Serial

16-byte = 8 bit FIFO

Output:

16-byte = 8 bit FIFO (see datasheet for details)

Input:

This device has all the hardware required for manual testing and evaluation. This also
The previous two components work together to form my Digital Thermometer Device.
Thermometer, the BOCOPPO, and the Xilinx Interface Board should be installed.

NOTE: When all three modules are interconnected in this way, the values show on the display of the value of the Thermometer:

Every time the clock is asserted, the value of the Thermometer will be displayed on the interface module (should be chosen and connected to the clock input for the BOCOPPO).

The desired clock (manual from the BOCOPPO, or 15Hz from the Xilinx Digitally

Usage:

Optional input clock

8 LED bank showing clear read into D-Buttons

Optional manual clock input

8 bit C.MOS level data lines from Thermometer

The original intention of this module was to allow interconnection with the Digital

C. Xilinx Interface Module
VI. Conclusion

Know enough to help debug problems, in creating software.

We're used to creating a thread in C++. Through our time, we have been challenged with creating communication with the bus. We were not doing difficult, the change from when Windows NT had just been supported until very recently. In the second part of the research, we believe that the Xilinx to SCSI interface should have been a great deal easier. SCSI is now understood with greater detail.

The Xilinx to SCSI interface was very challenging. We worked with our project and the other members did not.

Our group could have improved what we did by assigning two people to a task.

During the final part of the semester, we looked into how we could enhance the LSI.

We have found that the software side of this project to be very challenging.

mistakes might not have been made, but the design issues persist. If we could have known better, we would have been able to have a second person helping on various design issues. This part of the project might not have worked due to timing issues.

Also cause problems, also can cause issues with timing issues with the enable on the 6850.

Our group could have improved what we did by assigning two people to a task.

After the week from the USB to the serial interface, several weeks were spent trying to use the LSI for the serial interface. With these weeks remaining, the text.

The personal computer to Xilinx interface over the serial bus was a difficult task.
and Graphical LCD Controllers
Lansing, Paul J., Ahrendt, Jason A., Marsack, Jason D., Serial Communication with Text


Jersey 1991

Palmu Microelectronics, Fast Track to SCSI A Product Guide, Prentice Hall, New

VI. Sources
Appendix

VI. Appendix
<table>
<thead>
<tr>
<th>14</th>
<th>44</th>
<th>SEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0/0</td>
<td>I/O</td>
</tr>
<tr>
<td>7</td>
<td>48</td>
<td>REN</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>C/D</td>
</tr>
<tr>
<td>5</td>
<td>36</td>
<td>BUS</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
<td>MSG</td>
</tr>
<tr>
<td>3</td>
<td>38</td>
<td>ACK</td>
</tr>
<tr>
<td>8+</td>
<td>16</td>
<td>DB7</td>
</tr>
<tr>
<td>8+</td>
<td>14</td>
<td>DB6</td>
</tr>
<tr>
<td>82</td>
<td>12</td>
<td>DB5</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>DB4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>DB3</td>
</tr>
<tr>
<td>79</td>
<td>6</td>
<td>DB2</td>
</tr>
<tr>
<td>77</td>
<td>4</td>
<td>DB1</td>
</tr>
<tr>
<td>70</td>
<td>2</td>
<td>DBO</td>
</tr>
</tbody>
</table>

Xilinx to SCSI cable pinout
```c
#include "windows.h"
#include <memory.h>
#include <conio.h>
#include "ComIO.h"
#include "SCSI.h"
#include "resource.h"

BOOL WINAPI MainDlgProc(HWND, UINT, WPARAM, LPARAM);

BOOL WriteText(HWND);

BOOL ReadText(HWND);

void MyDlgWriter(HWND, char *);

void MyDlgClearer(HWND);

BOOL WINAPI MainDlgProc(HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch(msg)
    {
        case WM_INITDIALOG:
            return( TRUE );
        case WM_COMMAND:
            if(wParam == IDCANCEL)
                EndDialog(hDlg, TRUE);
            return( TRUE );
            break;
            case WM_DESTROY:
                EndDialog(hDlg, TRUE);
            break;
    }
    return( FALSE );
}

HINSTANCE hInst;
ComIO com2;
SCSI scsicom;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrev, LPSTR lpCmd, int nShow)
{
    hInst = hInstance;
    DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL, MainDlgProc);
    return 1;
}

BOOL WriteText(HWND)
{
    // Function to write text to the dialog box
    return TRUE;
}

BOOL ReadText(HWND)
{
    // Function to read text from the dialog box
    return TRUE;
}

void MyDlgWriter(HWND, char *)
{
    // Function to display text in the dialog box
}

void MyDlgClearer(HWND)
{
    // Function to clear the dialog box
}
```

BOOL WriteText(HWND hDlg)
{
    char SendCode[2], ScsiCode[7];
    int i;
    MyDlgClearer(hDlg);  // clear the text box
    if(com2.Status())
        MyDlgWriter(hDlg, "Error Creating Com Port Link");
    GetDlgItemText(hDlg, IDC_EDIT1, SendCode, 2);
    scsicom.scsiwrite(SendCode[0]); // converts char to 7 byte scsi code
    for(i=0; i<7; i++)
        ScsiCode[i] = scsicom.buffer[i];
    if(!com2.Write(SendCode, sizeof(SendCode)))
        // checks to see if
        MyDlgWriter(hDlg, "Error sending");
    else
    {
        MyDlgWriter(hDlg, ScsiCode);
        MyDlgWriter(hDlg, "was sent to the Serial Port.");
    }
    return(TRUE);
}

BOOL ReadText(HWND hDlg)
{
    char ReturnCode[2], ScsiCode[6];
    int ReturnSize = 3,
    i;
    MyDlgClearer(hDlg);  // clear the text box
    scsicom.scsiread(); // reads the 6 bytes to read
    for(i=0; i<6; i++)
        ScsiCode[i] = scsicom.buffer[i];
    if(!com2.Write(ScsiCode, sizeof(ScsiCode)))
        // checks to see if
        MyDlgWriter(hDlg, "Error sending");
    for(i=0; i<3; i++)
    {
        if(!com2.Read(ReturnCode, ReturnSize))
            // checks to see if something was read
            MyDlgWriter(hDlg, "Error receiving");
        else
        {
            MyDlgWriter(hDlg, ReturnCode);
            i=11;
        }
    }
    return(TRUE);
}

void MyDlgWriter(HWND hDlg, char *write)
{
    SendDlgItemMessage(hDlg, IDC_EDIT2, LB_ADDSTRING, 0, (LPARAM)write);
}
```c
void MyDlgClearer(HWND hDlg)
{
    // clears the console
    SendDlgItemMessage( hDlg, IDC_EDIT2, LB_RESETCONTENT, 0, 0);
}

#include "windows.h"
#include <memory.h>
#include <io.h>
#include <commctrl.h>

class ComIO{
public:
    ComIO();
    ~ComIO(){};
    BOOL Write(char *, DWORD);
    BOOL Read(char *, DWORD);
    int Status();

private:
    HANDLE h;
    int therewasanerror;
};

ComIO::ComIO()
{
    COMMTIMEOUTS timeouts;
    DCB dcb;
    // initialize the error code
    therewasanerror = 0;
    // open a port
    h = CreateFile("Com2", GENERIC_READ|GENERIC_WRITE,
        0, 0, OPEN_EXISTING,
        FILE_FLAG_OVERLAPPED,
        GENERIC_READ|GENERIC_WRITE);
    if(GetLastError() == ERROR_FILE_NOT_FOUND)
        therewasanerror = 1;
    // setup timeouts in milliseconds
    timeouts.ReadIntervalTimeout = 2000;
    timeouts.ReadTotalTimeoutMultiplier = 1000;
    timeouts.ReadTotalTimeoutConstant = 1000;
    timeouts.WriteTotalTimeoutMultiplier = 1000;
    timeouts.WriteTotalTimeoutConstant = 1000;
    SetCommTimeouts(h, &timeouts);
    // set timeouts
    // setup DCB
    FillMemory(&dcb, sizeof(dcb), 0);
    dcb.DCBlength = sizeof(dcb);
    BuildCommDCB("1200,n,8,1", &dcb);
    // set to 1200 k
    // n = noparity
    // 8 = transfer
    // 1 = stop bit
    // transfer
    BuildCommDCB("1200,n,8,1", &dcb);
}
```
BOOL ComIO::Write(char * buf, DWORD bytestowrite){
// Write will make sure that it did not timeout before
// if was finished writing.
OVERLAPPED WriteOL = {0};
DWORD byteswritten, dres;
BOOL fres;
WriteOL.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
if (WriteOL.hEvent == NULL)
    return FALSE;
if (!WriteFile(h,
    buf, bytestowrite, &byteswritten, &WriteOL))
    if (GetLastError() != ERROR_IO_PENDING) {
        // Write failed
        fres = FALSE;
    } else{
        // Write pending
        dres = WaitForSingleObject(WriteOL.hEvent, INFINITE);
        switch(dres){
            case WAIT_OBJECT_0:
                if(!GetOverlappedResult(h, &WriteOL, &byteswritten, FALSE))
                    fres = FALSE;
                else{
                    if(byteswritten != bytestowrite)
                        fres = FALSE;
                    else
                        fres = TRUE;
                }
                break;
            default:
                // Write failed
                fres = FALSE;
                break;
        }
    }
else{
    // Write completed
    if (byteswritten != bytestowrite)
        fres = FALSE;
    else
        fres = TRUE;
}
CloseHandle(WriteOL.hEvent);
return fres;
}

BOOL ComIO::Read(char buf[], DWORD bytestoread){
// This read will make sure the read didn't timeout
}

BOOL ComIO::Read(char buf[], DWORD bytestoread)
{

return fres;
}

CLOSEHANDLE(WriteOL, ReadOL);

if (WriteOL.hEvent == NULL)
    return FALSE;
else{
    // Write completed
    if (byteswritten != bytestowrite)
        fres = FALSE;
    else
        fres = TRUE;
}
CloseHandle(WriteOL.hEvent);
return fres;
}

{ // It was flushed writing. //
Write command::Write(char * buf, DWORD bytestocommit)
    //
    
} // I stop //
return FALSE;
if (!ReadFile(h, buf, bytestoread, &bytesread, &ReadOL)) {
  if (GetLastError() != ERROR_IO_PENDING) {
    // Read Failed
    fres = FALSE;
  } else {
    // Read pending
    dres = WaitForSingleObject(ReadOL.hEvent, INFINITE);
    switch (dres) {
      case WAIT_OBJECT_0:
        if (!GetOverlappedResult(h, &ReadOL, &bytesread, FALSE))
          fres = FALSE;
        else {
          if (bytesread != bytestoread)
            fres = FALSE;
          else
            fres = TRUE;
        }
        break;
      default:
        // Read Failed
        fres = FALSE;
        break;
    }
  }
} else {
  // Read completed
  if (bytesread != bytestoread)
    fres = FALSE;
  else
    fres = TRUE;
}
buf[2] = (char)"\0";
// to end the string
CloseHandle(ReadOL.hEvent);
return fres;

// The ComIO:Status:
)
//
// return fress;
// CloseHandle(ReadOL.hEvent);
// buf[2] = (char)"\0";
//
// if (fress) {
//  if (fress) {
//    if (bytesread == pyrestread)
//      Read completed //
//    else if (bytesread > pyrestread)
//      Read Failed //
//    else if (bytesread < pyrestread)
//      Read Failed //
//    else
//      Read Failed ;
//    break;
//  }
//}
//
// break;
// break;
//
// break;
//
// else if (fress)
// else if (fress)
// else if (fress)
// else if (fress)
//}
//
// if (GetLastError() == ERROR_IO_PENDING) { 
// if (GetLastError() == ERROR_IO_PENDING) {
// case WAIT_OBJECT_0: 
// case WAIT_OBJECT_0: 
// dres = WaitForSingleObject(ReadOL.hEvent, INFINITE);
// dres = WaitForSingleObject(ReadOL.hEvent, INFINITE);
// Read pending //
// Read pending //
// (fress = FALSE)
// (fress = FALSE)
// (fress = FALSE)
// (fress = FALSE)
//)
//)
//)
//)
// if (GetLastError() == ERROR_IO_PENDING) {
// if (GetLastError() == ERROR_IO_PENDING) {
// (fress = FALSE)
// (fress = FALSE)
// (fress = FALSE)
// (fress = FALSE)
//)
//)
//)
//)
// return FALSE;
// return FALSE;
// return FALSE;
// return FALSE;
//}
```cpp
#include "SCSI.h"

void SCSI::scsiwrite(char temp){
    buffer[0] = (char)WRITE;
    buffer[1] = (char)ID0;
    buffer[2] = (char)ADDR;
    buffer[3] = (char)ADDR;
    buffer[4] = (char)WRITE_LENGTH;
    buffer[5] = (char)CONTROL;
    buffer[6] = (char)temp;
}

void SCSI::scsiread(void){
    buffer[0] = (char)READ;
    buffer[1] = (char)ID0;
    buffer[2] = (char)ADDR;
    buffer[3] = (char)ADDR;
    buffer[4] = (char)READ_LENGTH;
    buffer[5] = (char)CONTROL;
}
```