Maximizing Serial Ports for File Transfers between Computers: Design Issues

Moses E. Ekpenyong, Odudu-Obong Udocox
Department of Mathematics, Statistics and Computer Science, University of Uyo,
P.M.B. 1017, 52001, Akwa Ibom State, Nigeria., *e-mail: (ekpenyong_moses@yahoo.com.)

Abstract

Serial communication is by default meant for computer to modem communication. One fascinating idea presented in this paper is the ability to effect serial communication between two computers. We desire to maximize the serial ports such that file(s) can be transferred directly between two computers connected with a serial cable and connectors. The essence is to widen the understanding on how this kind of communication can be made possible. Our methodology configures a null modem cable in a way that the transmitting computer is deceived that it is transmitting a modem. A C program is then written to manage the transmission. Test transmission shows that file(s) can be effectively transmitted using serial communication.

Keywords*: RS232 standard, Loopback test, Serial Communication, Synchronous and Asynchronous signals*

1. Introduction

Since the emergence of computers, there has been an intense need for sharing and transferring of files from one terminal to another. A collection of electronically stored files can be moved by physically moving the electronic storage medium, such as a flexible disk, hard disk, or compact disk from one place to another or by sending the files over a telecommunication medium.

On the Internet, the File Transfer Protocol (FTP) is a common way to transfer a file or relatively small number of files from one computer to another. Other means by which data can be transferred from one computer to another include: (i) the use of an infrared wireless connectivity, (ii) the use of Blue-tooth wireless connectivity, (iii) the use of flash disk or drive usage.

It has been observed that there has been a faithful replication of serial and parallel ports on every laptop, desktop or server, and these parts are not intensely utilized due to the presence of faster interfaces like Universal Serial Bus (USB), Ethernet, etc, [1], [2].

At first sight it would seem that a serial link is inferior to a parallel link, because it tends to transfer less data in each lock cycle. However, it is often the case that serial data links can be clocked considerably faster than parallel links to achieve a higher data rate. Some factors that allow serial link to be clocked at a greater rate include: (i) clock skew between different channels is not an issue, (ii) serial connections require fewer interconnecting cables (e.g. wires/fibres) and hence occupy less space. The extra space allows for better isolation of the channel from its surrounding, (iii) cross talk is less of an issue, because there are fewer conductors in close proximity.

In many cases, serial communication is a better option because it is cheaper to implement. Many Integrated Circuits (ICs) have serial interface, as opposed to parallel ones, such that they have fewer pins and are therefore cheaper.

The most important limitation of communications through serial port is the speed of data transfer. Originally, serial port communication limited the maximum transfer speed to 20kbps, which is quite slow, compared to transfer on an Ethernet, which could be 10Mbps or 100Mbps and recently 1Gbps.

Another limitation is in the utilization of the PC bus bandwidth. Serial port communicates with the computer via the PCI bus, the LPC bus, X-bus, or ISA bus. The PCI bus is either 32 or 64 bits wide, but the serial port only sends a byte (8 bits wide) a time. For the LPC bus, which is only 4-bits wide, the serial port only sends one byte at a time. It provides an efficient interface for serial port transfers. The ISA bus is usually 16-bits wide as such the efficiency is intermediate as compared with efficient LPC and inefficient PCI.

Most problems encountered in serial communications lies in the software and not the hardware. Some of such software limitations are: (i) timing in program (hanging), (ii) timing in program (incomplete transfer), and (iii) instrument's termination character.

The justification of this paper is as follows:

(1) In the field of scientific and technical instrumentation, most devices are microprocessor based and usually have an RS232 part for connection to the computer. This simply means that serial products are easier to develop as compared to USB.

(2) Serial cables are longer in length (about 50ft), cheaper in cost and are much more cost effective to produce than parallel cables. Serial cables transmits a '1' as –3v to –25v and a '0' as +3v to +25v, whereas parallel cables transmits a '0' as 0v and a '1' as 5v. Therefore the serial port can have a maximum swing of 50v compared to the parallel port which has a maximum swing of 5v, as such, information loss through cabling will not be much problem for serial cables than they are for parallel cables, [3].

(3) Serial transmissions do not require many wires as opposed to parallel transmissions. If devices need to be mounted a bit far from the computer, then 3-core cables will be a lot cheaper than running 19 or 25 core cables. However, the cost of interfacing at both ends must be taken into account, [3].

(4) Micro-controllers have in recent times been proven popular. Many of these have inbuilt Serial Communication Interface (SCI), which can be used to communicate with the outside world.

2. Serial Communication

Serial communication, like any other data transfer procedure, requires coordination between the sender and receiver. For instance, when to begin the transmission and when to end it, when one particular bit or byte ends and when another begins, when the receiver's capacity has been exceeded, and so on. A protocol defines the specific methods of coordinating transmission between a sender and receiver. The scope of serial data transmission is large and a bit complex, encompassing everything from electrical connections to data encoding. Such electrical connection is through the RS232 standard, which is one of the oldest physical communications standard in the computer world. The standard defines low-cost serial communication in a robust way where bits are sent sequentially on a copper line. It was originally defined for connecting devices such as computers, terminals, and printers to modems, [4].

The RS232 governs the physical and electrical characteristics of serial communications. This specification defines several additional signals that are meant for information and control beyond data signal ground. The RS232 standard defines 25 signal lines in its interface, although in practice PCs rarely use more than nine of these lines. Just three of these lines - RD, TD, and GND are required for bi-directorial serial data communication. The rest including the remainder of the basic nine – DCD, DTR, DSR, RTS, CTS and RI which are designated for a variety of control purposes.

RS232 signals can be synchronous or asynchronous. Synchronous RS232 signals are synchronized by a clock that dictates the timing of each bit that is sent. Both sides of the serial connection share the timing provided by the clock, so each side is aware of the timing of the next byte of data. Asynchronous RS232 signals are described by voltage changes that will identify the start and stop of any byte of data. Within any byte of data, the receiver is actually applying a clock to measure the elements of the data transmission and will sample the voltage level within the byte, the number of times that correspond to the number of discrete bits of data it expects from the byte, along with its framing and possible parity bits.

We will in this paper focus on Asynchronous RS232 serial port communications, primarily in the PC world, and will emphasize the connectors and UARTs most frequently seen in today's PCs. Serial communication can be said to be the transmission and reception of data one bit at a time. Today's computer generally addresses data in bytes or some multiple thereof. A byte contains 8 bits. A bit is basically either a logical 1 or 0. The serial port is used to convert each byte to a stream of ones and zeros. The serial port contains an electronic chip called a Universal Asynchronous

Receiver/Transmitter (UART) that does the actual conversion. It converts parallel bytes from the CPU into serial bits for transmission and vice versa. It also generates and strips the start and stop bit appended to each character.

Serial Port speed

The speed of the serial port is largely determined by the UART used by it. The most typical types of UARTs (the 16550 and 16650 UART) have theoretical maximum speeds of 115.2kbps and 460.8kbps respectively. The advanced UARTs have substantial speed advantages over earlier design. However, for a serial connection to take advantage of an advanced UART, several conditions are necessary:

(1) there must be a driver software present that is designed to handle the UART.
(2) the serial port's buffer and trigger levels must be properly configured.
(3) wherever possible, hardware flow control should be used in preference to software flow control.
(4) the peripheral involved must be fast enough to benefit from the UART.

These conditions if not met will cause a serial connection to operate below the theoretical speed of its UART. Many other factors can influence port speed. In other words, significant speed differences may result in actual use. On the hardware side these factors include the speed and architecture of the computer's CPU and peripheral need for high-speed serial ports. On the software side, the type of code and operating system will affect port speeds. For example, DOS is expected to have faster serial port transfer than windows.

Baud Rate

An important feature that determines the speed of the serial port is the Baud Rate. *The Baud Rate is the number of times the signal can switch states in a second*. The Baud rate is configured as bit per second. The transferred bits include the start bit, the data bits, the parity bit (if used), and the stop bit. However, only the data bit is stored. Communicating computers must be configured to the same Baud Rate before data can be successfully read or written. The default baud rate for any communication is 9600bps. Other standard baud rates are 110, 300, 600, 1200, 2400, 4800, 14400, 19200, 38400, 57600, 115200, 128000 and 256000 bits per second. However, the value of the Baud Rate is dependent on the serial port concerned.

There are available software used for file transfers, some of the products include: Fastlynx, [5], Winxfer, [6], etc.

Serial Port Interfacing

A serial port is harder to interface than the Parallel Port. In most cases any device connected to the serial port will need the serial transmission converted back to parallel so that it can be used. This is done using a UART. On the software side, there are more registers that must be attended to than on a Standard Parallel Port (SPP), [7].

3. Design Components and Methods

The Serial Port design pattern defines a generic interface with a serial port device. Our main intention is to completely encapsulate the interface with the serial port hardware device. The embedded software has to be able to interact properly with the hardware that is connected to the serial port. Serial port is implemented with the Serial Port and Serial Port-Manager classes. The Serial Port Manager maintains an array of Serial Port objects. Each Serial Port object manages the transmit and receive buffers. The Serial Port Manager class implements the interrupt service routine.

Implementing the serial port design pattern keeps the hardware dependent code confined to a few classes in the system. The implementation of this design pattern is also explained in terms of

handling of message transmission and reception. An important point to note here is that the code executing in the context of the ISR is kept at the minimum.

Design Requirments

To design a serial communication medium, we need the following:

(1) A Null Modem Cable (with RS232 connectors at both ends)
(2) A functional serial port at both ends
(3) An x86-based processor with a minimal 64MB of RAM.
(4) Windows 9x/Windows NT/Windows ME/Windows 2000/Windows Xp.
(5) The data to be transferred which has to be singular at a given time. This is because both terminals cannot be sending or receiving at the same time. If one terminal is sending then the other terminal must be receiving and vice versa.
(6) A program written in any programming language to instruct the CPU to effect the transfer or in general, effect the communication medium.

Protocol Definition

A protocol can be defined as an agreed upon format for transmitting or receiving data between two devices. The protocol determines the following: (i) the type of error checking to be used, (ii) data compression method, if any, (iii) how the sending device will indicate that it has finished sending a message, and (iv) how the receiving device will indicate that it has finished receiving a message.

From a user's point of view, the only interesting aspect about protocols is that the computer in use or device must support the right protocol and there must be a communication. In serial communications there are 3 protocols that can be used to achieve a successful transmission:

1) Xmodem
2) Ymodem
3) Zmodem

Xmodem: This is one of the most widely used file transfer protocols. The original Xmodem protocol uses 128-byte packets and a simple "Checksum" method for error detection. A later enhancement (Xmodem-CRC) uses a mere secure Cyclic Redundancy Check (CRC) method for error detection. Xmodem always attempts to use CRC first. Xmodem can also be called Modem2 or Modem. Xmodem 1K is essentially Xmodem-CRC with 1K (1024 byte data blocks) packets.

Ymodem: Ymodem is essentially Xmodem 1K that allows multiple batch file transfers. On some systems it is listed as Ymodem Batch. And a variant of Ymodem is Ymodem-g. The Ymodem is designed to be used with modems that support error control. This protocol does not provide software error correction or recovery, but expects the modem to provide the service. It is a streaming protocol that sends and receives 1K packets in continuous streams until instructed to stop. It does not wait for positive acknowledgement after each block is sent, but rather sends blocks in rapid succession. If any block is unsuccessfully transferred, the entire transfer is cancelled.

Zmodem: Zmodem is generally the best protocol to use if the electronic service called supports it. Zmodem has two significant features. It is extremely efficient and it provides crash recovery. Like Ymodem-g, Zmodem does not wait for positive acknowledgement after each block is sent, but rather sends blocks in rapid succession. If a Zmodem transfer is cancelled or interrupted for any reason, the transfer can be restarted later and the previously transferred information need not be sent again. Xmodem and Ymodem are half duplex protocols. This avoids buffer overrun problems that result from attempting to exploit full duplex asynchronous file transfer protocols.

Cable Interfacing

A Null modem is used to connect two DTE (Data Terminal Equipment) together. This is commonly used as a cheap way to transfer files between computers using Zmodem Protocol, Xmodem Protocol, etc, [8], [9], [11].

To deceive the computer, we configure the null modem as shown below:

D9	D25 D9				D25	
3	2	TD	→	RD	3	2
2	3	RD	→	TD	2	3
5	7	SG	→	SG	7	5
4	20	DTR	→	DTR	20	4
6	6	DSR	→	DSR	6	6
1	8	CD	→	CD	8	1
7	4	RTS	→	RTS	4	7
8	5	CTS	→	CTS	5	8

Fig. 1. Interfacing of a Null Modem Cable

Fig. 1. Can be redrawn as below with respect to the shape of a D9 connector; to enhance understanding;

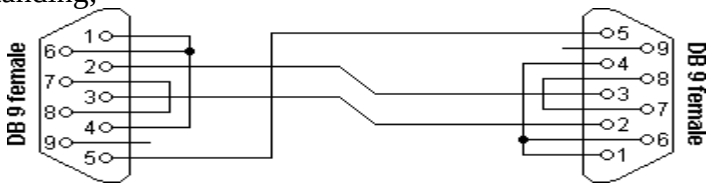


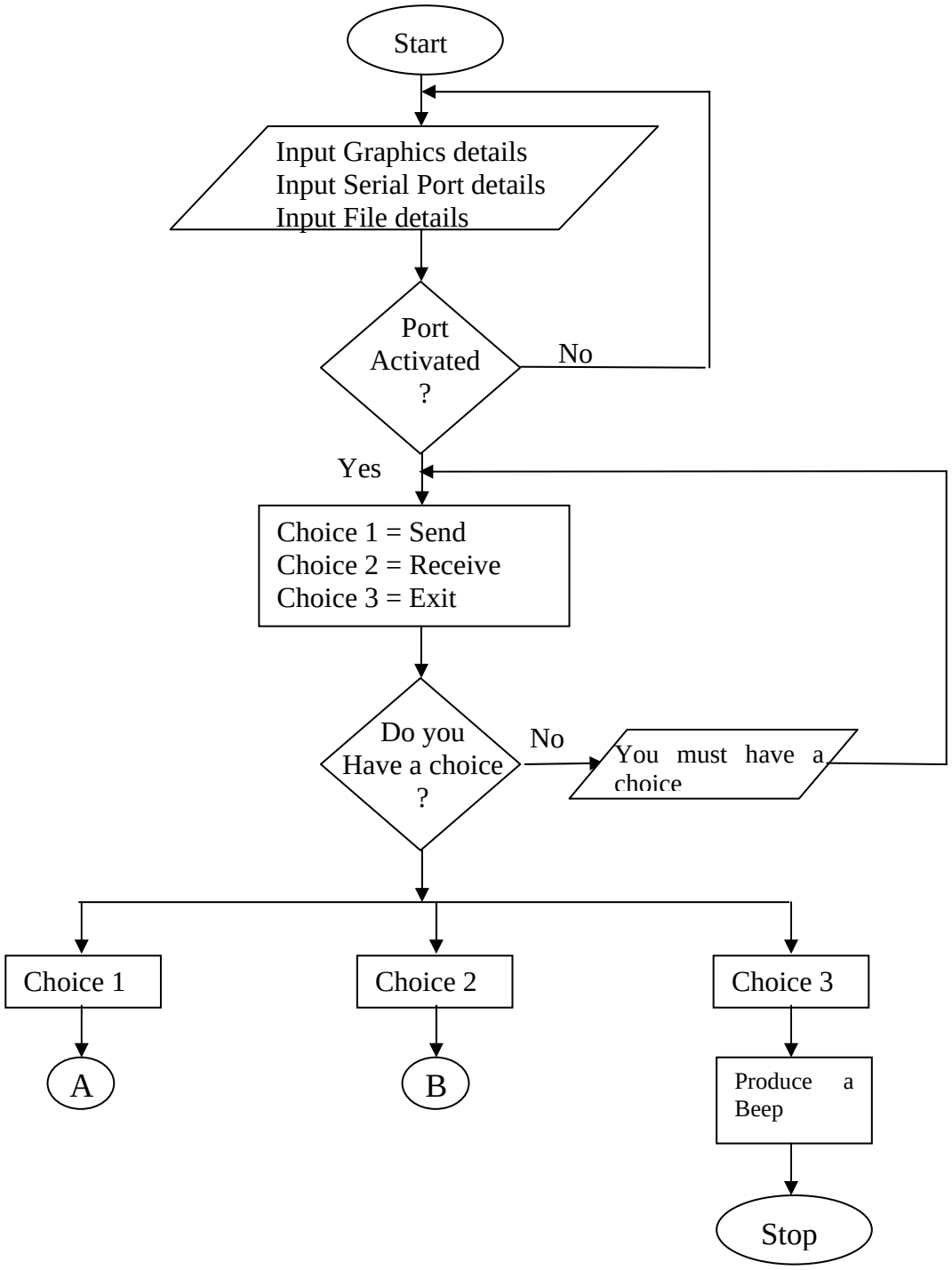
Fig. 2. Redrawn version of fig. 1.

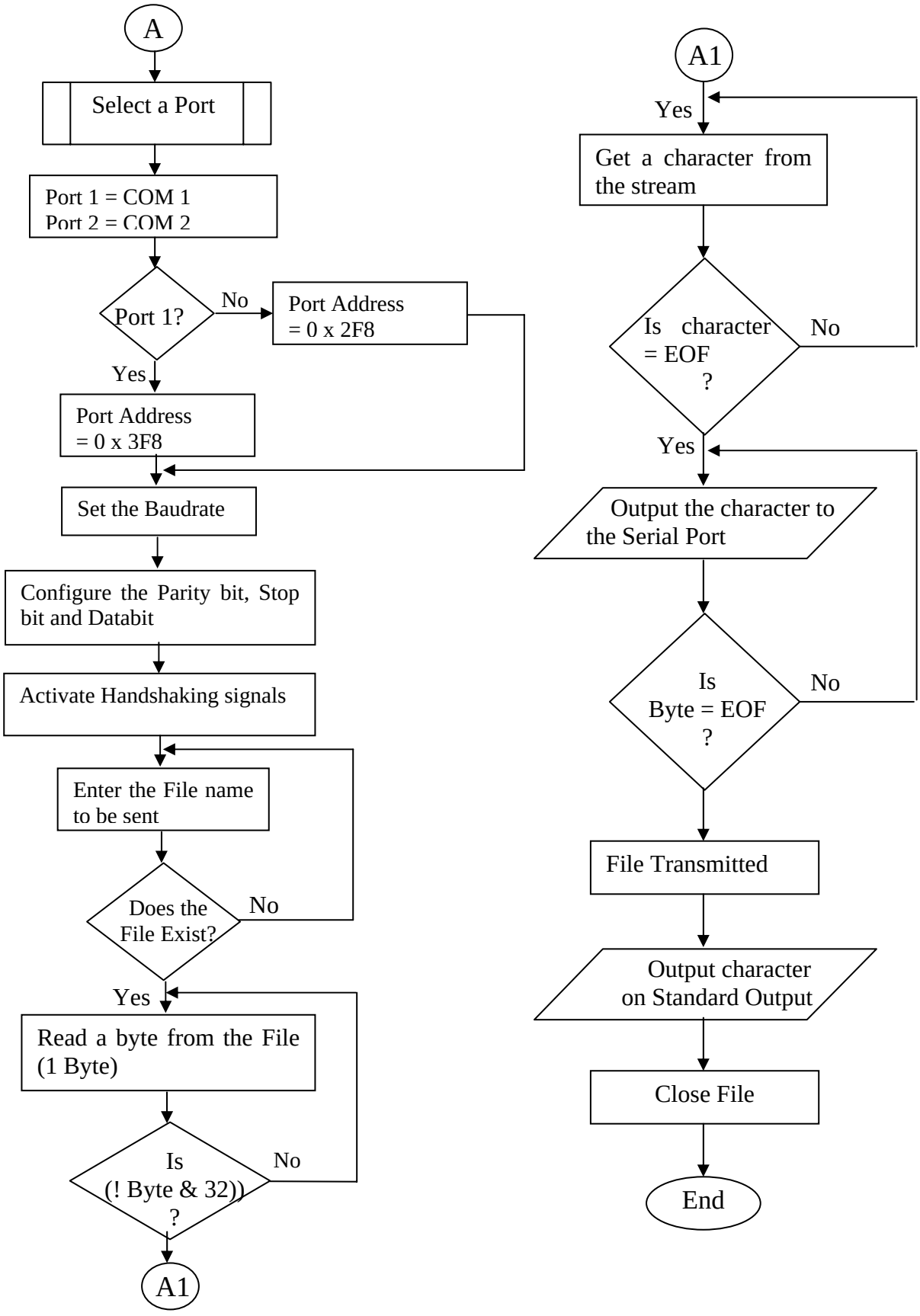
Figures 1 and 2 shows the wiring of the null modem cable. It only requires 3 wires (TD, RD, and SG) to be wired directly (i.e. the transmitting and receiving computers connected directly), though this makes it more cost effective to use with long cable runs.

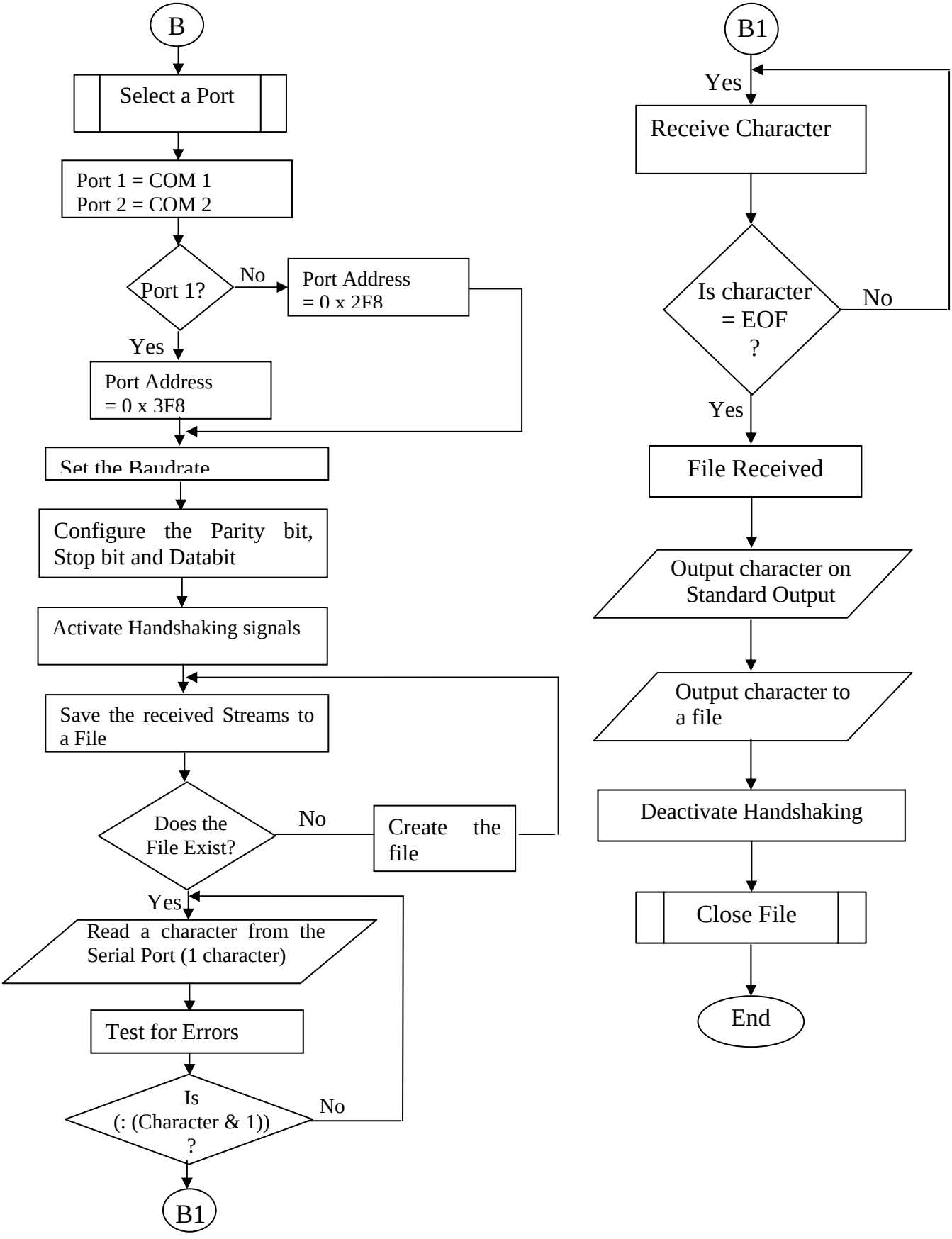
The theory of operation of the Null modem operation is reasonably easy. The aim is to make a computer think it is talking to a modem rather than to another computer. Any data transmitted from the first computer must be received by the second; thus Transmit Data (TD) is connected to Receive Data (RD). The second computer must have the same set-up, thus RD is connected to TD. Signal ground (SG) must also be connected, so both grounds are common to both computers.

Program Design

We here present the communication program algorithm.




A
Select a Port
Port 1 = COM 1
Port 2 = COM 2
Port 1?
No
Port Address
= 0 x 2F8
Yes
Port Address
= 0 x 3F8
Set the Baudrate
Configure the Parity bit, Stop bit and Databit
Activate Handshaking signals
Enter the File name to be sent
Does the File Exist?
No
Yes
Read a byte from the File (1 Byte)
Is
(! Byte & 32))
?
No
A1
A1
Yes
Get a character from the stream
Is character
= EOF
?
No
Yes
Output the character to the Serial Port
Is
Byte = EOF
?
No
File Transmitted
Output character on Standard Output
Close File
End


B
Select a Port
Port 1 = COM 1
Port 2 = COM 2
Port 1?
No
Port Address
= 0 x 2F8
Yes
Port Address
= 0 x 3F8
Set the Baudrate
Configure the Parity bit, Stop bit and Databit
Activate Handshaking signals
Save the received Streams to a File
Does the File Exist?
No
Create the file
Yes
Read a character from the Serial Port (1 character)
Test for Errors
Is (: (Character & 1)) ?
No
B1
B1
Yes
Receive Character
Is character = EOF ?
No
Yes
File Received
Output character on Standard Output
Output character to a file
Deactivate Handshaking
Close File
End

4. System Implementation

Serial port communication implementation is somehow complex. However the extra complexity is meant to make things ultimately more flexible and powerful.

The implementation is done using C programming language. Regardless of the Application Programming Interface in use, the main steps to implementing the serial communication are the same. These are:

(1) Open the serial port
(2) Configure the connection, [10]
(3) Data handling
(4) Close the port

A program to implement the program algorithms was coded in C programming language and tested.

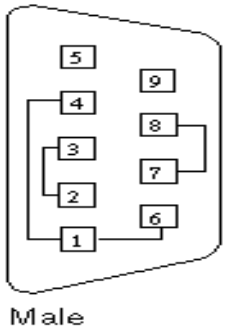
System Testing

Our purpose here is to provide information on the basic instruction on how to test the RS232 serial port functionality for the communication software.

There are two ways to test the serial communication software:

Loopback Test: A loopback test is the ideal method to determine if a serial port is working correctly or not. A loopback test made up of a loopback connector consisting of a connector without a cable and includes an internal wiring to reroute signals back to the sender. This DB-9 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it received came from a remote DCE device set to echo characters, or from a loopback connector. As such loopback connectors are used to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable to be used and attach the loopback to the end of the serial cable to verify the cable. In this case, Transmit Data jumpers with Receive Data, Request-to-send jumper with Clear-to-send and DTE-Ready jumpers with DCE-Ready and Receive Line Signal will be detected.

Diagrammatically we have:



Connect: First connector	to	Second connector
Pin 2		Pin 3
Pin 1		Pins 4 and 6
Pin 7		Pin 8

Fig 3. Loopback connector

Breakout Box Diagnostic: Another way to test a serial connection is by using a breakout box. The breakout box is put in the mainline (i.e. between the PC and the host) and each end's response is watched as characters go out and come in.

System Evaluation

The essence of evaluation is determined if the serial communication software properly meets our needs. That means our needs must be defined at the very first instance.

This program can be evaluated based on some important attributes. It is these attributes that determine the benefits, drawbacks and risks of using the program. Some attributes are:

Functionality

Here we try to answer the question: does the program perform what we expected of it? As an answer we would say "Yes". Our soul aim is to ensure that data is transmitted and received without errors or crashes. In a case where a new functionality needs to be added, the source code is open source. Adding functionality to an existing program is usually less costly than building the program from scratch.

Cost

Cost is an important issue for anyone who is interested in deploying a program. Potential costs for this communication software are: (i) installation cost, (ii) support/maintenance cost (include troubleshooting), (iii) indirect costs (such as downtime and training), (iv) transition costs (such as data transition and/or transition to upgrades), (iv) cost of any necessary hardware (purchase and upgrades). Generally, the cost of deploying this serial communication software is not much compared to other media of communication. A null modem cable can be bought here in Nigeria at a cost of ₦4,500.00, or it could be constructed locally, provided there is a cable with RS232 connectors at both ends (DB9). The cost of maintaining serial communication software is not too different from other programs. The maintenance/support should be focused more on the hardware (i.e. the serial port, cables and connectors) rather than on the software. Often it is less costly to troubleshoot and maintain hardware than to replace it.

This software is less likely to incur indirect costs, except in cases were the host operating system gets faulty in any way. All things being equal, this serial communication program can run for hours without a down time. The training aspect here is not quite necessary except for individuals who have some difficulty typing, using a mouse, or in most cases, understanding English. Hence this software is quite easy to use. Usually as years go by, software become obsolete with respect to some functionalities, and needs a kind of upgrade or transition. The cost of doing this varies with software but for our serial communication software, the cost of upgrade solely depends on what functionality is to be added or subtracted. The cost of hardware needed varies with the area in question e.g. a null modern cable of ₦4,500.00 in Nigeria will cost approximately 35.00 dollars in US. The most important issue is to know the hardware needed where to get it and have some money, not too much though.

Maintenance / Longetivity

Few useful programs are completely static. They need changes, new uses are continuously created, and the programs must be maintainable into the future. This serial communication program can be maintained both in the hardware and the software thereby increasing its longetivity.

Reliability

Reliability measures how often the program works and produces the appropriate results. Reliability is difficult to measure and strongly depends on how the serial communication software is used. The reliability of this communication medium was measured. Though roughly estimated, has a fair reliability level, but could be improved with further research and funding.

Scalability

Scalability, in this context, suggests the size of data or problem the program can handle. This communication software can handle any size of data stream but the user has to know that the time it takes to completely transfer a file is directly proportional to the size of the file. This communication software "may not" be able to handle multimedia files or images, but with additional functionalities, this can be possible.

Flexibility and Customizability

Flexibility and customizability are highly interrelated attributes of this communication software. By being flexible means, the program can be used to handle circumstances that it wasn't originally designed for, while, its customizable ability measures how well the communication software can be customized to fit in a specific environment.

6. Conclusion

Serial Communications is not a new development. Traditionally, it is designed for computer-modem communication. As such the speed requirement is not much of a factor. As years went by and previous inventions and discoveries in the computer industry become obsolete in the new era, more researches carried out on the capability of the serial port reveal that more can be done. For this reason, communication can be effected from a computer to a computer, computer to a camera, computer to a phone, etc, all via serial port.

On a platform where speed is a factor and a prerequisite for communication, serial communication is not recommended due to its bit by bit transmission. The UART buffer is anything between 16 and 64KB. This is quite small and will result in large documents taking a while to be completely transmitted. A situation where flow control is not enabled, then there is an evident occurrence of an overrun Error. Serial Communication is not also recommended for long distance transmission. The longest distance it can go is 50ft.

Despite all these, Serial Communication is recommended in platforms where all other means of communication are not supported or enabled. It is also recommended for a computer-camera and computer-phone communication or for car use where the issue of being slow may actually be beneficial.

5. References

1. http://www.lvr.com
2. www.lavalink.com
3. http://www.beyondlogic.org/serial/serial.htm
4. http://www.netbsd.org/Documentation/Hardware/Misc/serial.html#connect
5. http://sewelldirect.com/
6. http://www.wcscnet.com/WXferBro.htm
7. http://www.eg3.org/serial.htm
8. http://www.arcelect.com/rs232.htm
9. http://www.electrosofts.com/
10. http://www.8052.com/tutser.phtml
11. http://www.captec.co.uk/pdf/serial-communications-tutorial.pdf

Article received: 2006-10-28