Host-Controlled Ethernet Switch

Final Project Report
December 9, 2000
E155

Taufiqul Kazi and Hyun Choi

Abstract:

Starting in the fall of 2000, students in J. L. Atwood dorm were allowed the option of purchasing a 100Mbit Ethernet connection. The 100Mbit service allowed for students to have a faster connection to the network, but the students were unable to access KATO. To access KATO, a student had to manually switch the Ethernet cable from the 100Mbit port to the 10Mbit port. To alleviate this problem, we designed an Ethernet switch that would allow a user to control which port their computer is connected to. By using this Ethernet switch, there would be no physical switching of cables. We used reed relays to switch the Ethernet signals and a 68HC11 to communicate with the host PC.

Introduction

Our Ethernet at Harvey Mudd College uses twisted-pair Ethernet, Type 10base-T with RJ-45 jacks. Although the cable itself has eight wires, Ethernet only used 4 of the cables. Two are used as a pair to transmit and two as a pair are used to receive. The signals traveling through each pair of wires were not 5 V and 0 V like most digital circuits but were either +2.5 V or –2.5 V. Because the signals were not digital levels and because they were bi-directional, we were not able to switch the signals through an FPGA. We needed something that would allow us to switch between two wires at all voltage levels and in both directions. We decided to use a reed relay, which uses a current to choose between two signals to carry.

To communicate with the host computer, we used a 68HC11. The HC11 received a signal from the host computer and then sent a signal to an FPGA so that the FPGA could control the relays. The communication between the HC11 and the Host computer was done through a serial interface. The HC11 was set up to use its built in SCI interface to receive incoming serial data. Taufiqul had some previous experience using Java to communicate through serial ports, so the decision was made to use Java as the code base on the Host computer side of the system. We wrote two separate java programs with the only difference being what bit value was sent through the serial port. One program sent a bit sequence to set the relay input to port A, while the other program set the relay circuit to B.

New Hardware

In our project we needed to find out which four of the eight wires were used for Ethernet. We were easily able to find them on the web and the following table shows which wires were used for which function:

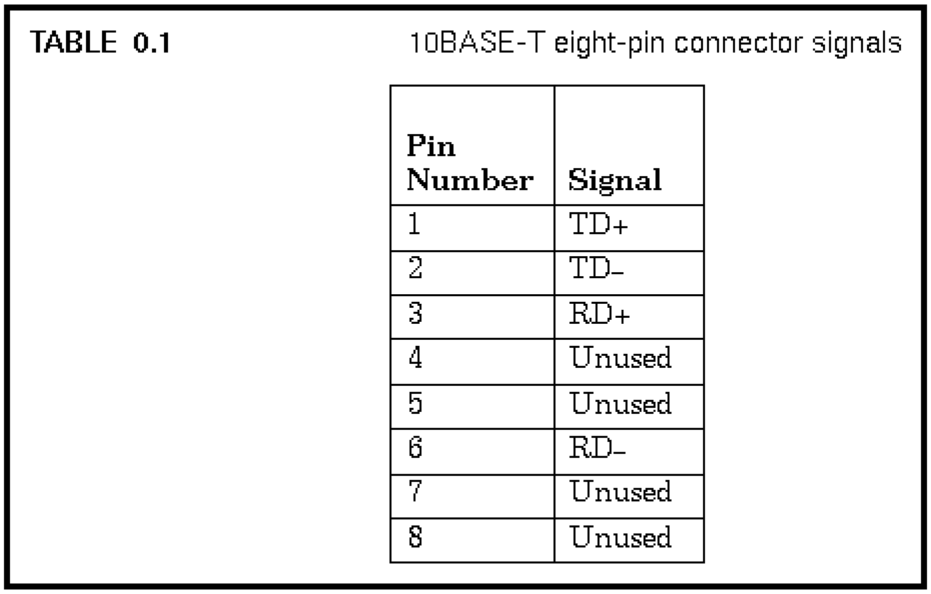


TABLE 0.1 10BASE-T eight-pin connector signals

Pin Number	Signal
1	TD+
2	TD-
3	RD+
4	Unused
5	Unused
6	RD-
7	Unused
8	Unused

Table 1. Ethernet Cable Signals

We also used RJ-45 jacks and ports to connect the wires. To actually switch the wires, we used reed relays. When enough current is passed through the coils, the coils produce an electromagnetic field that provides a force on the selecting wire so that wire A is connected to wire C. When there is no force acting on the selecting wire, a spring forces the wire to choose circuit B to connect to wire C.

The layout of the serial port was also needed to determine which pins to tie to the HC11. We found the pin layout online and is as follows:

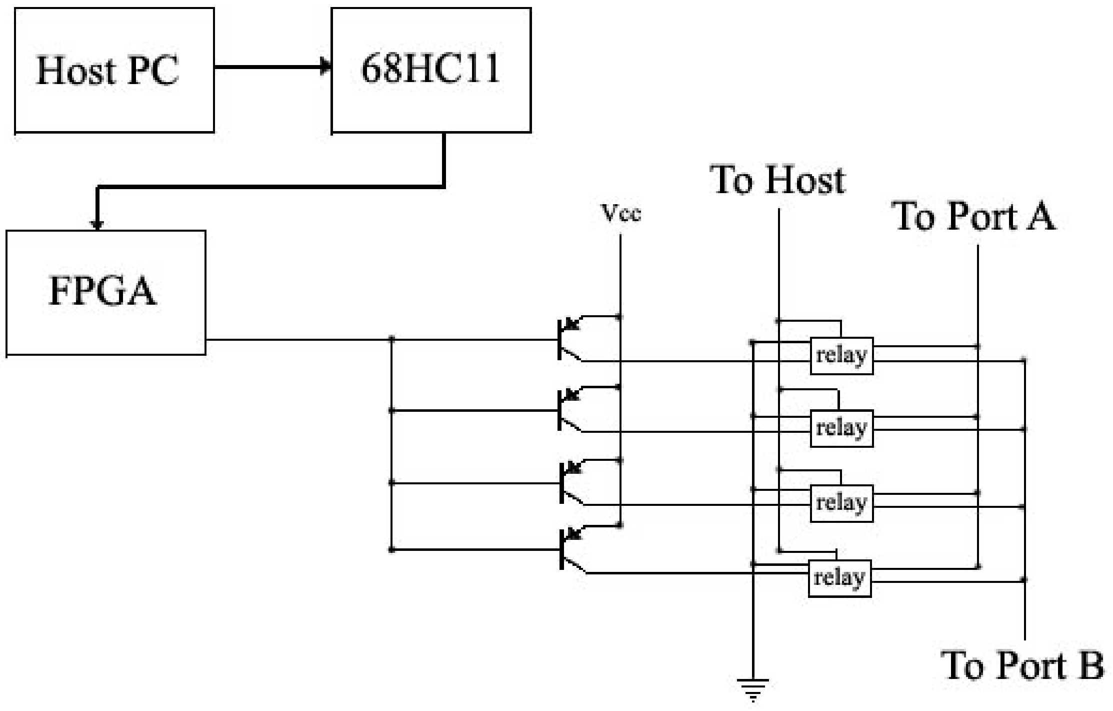
1	DCD	Data Carrier Detect
2	RD	Receive Data
3	TD	Transmit Data
4	DTR	Data Terminal Ready
5	SG	Signal Ground
6	DSR	Data Set Ready
7	RTS	Request to Send
8	CTS	Clear to Send
9	Ring	Ring Indicator

Table 2. Serial pin layout for DB-9 style cables and ports

For our purposes, we only needed two pins. The transmit data pin and signal ground were used in our design. The signal ground was connected to a common ground between all components to make sure that all signal levels were readable to each component. The Transmit data pin was connected to the SCI receive pin of the HC11. Through this connection, data could be sent from the host computer and received by the HC11 for processing.

Schematics

This is a schematic of how the hardware was set up.



The relays that we used needed a minimum of 89.3 mA and the FPGA could only sink 12mA, so we decided to use PNP resistors. We connected a pin from the FPGA to the bases of the transistors, connected V_{cc} to the emitter, and connected the collector to the relay. This allowed us to create a selector circuit that was active low. Thus when the FPGA sunk some current, the transistors would allow current to flow through the coils within the relay and switch the input port.

The pins from each relay were then collected in a RJ-45 jack and connected to the host, port A and port B.

Microcontroller Design

The program for the HC11 had 4 goals for its design. The HC11 must receive data serially from the host computer. This data must then be sent in parallel to the FPGA. The HC11 must also be able to continually look for incoming data and change the parallel output when new data arrives. Finally the HC11 must be able to hold the output value and thus maintain the data flow through the relays. Appendix A has the HC11 program in its entirety.

The program achieves its functions by first setting up the SCI hardware on the HC11, then entering into an infinite loop. All functions are then triggered by interrupts. When an interrupt is generated by the receiver hardware, the program jumps to the interrupt service routine and reads the incoming data. In theory, the interrupt is only generated on a completed transfer of 8 bits from the host computer. During the interrupt service routine, the read data is then stored in register $1004. This register is the data register for the parallel output. The parallel output functions by continuously outputting the value stored in the data register. For example, if the bit sequence 01000000 was stored in the data register for the parallel port, then pin 6 of the parallel port would be seen high while the rest were low.

With this method of interrupts, the program matches all the goals described above. It continuously checks for incoming data, through interrupts processes the data to the parallel port and since the parallel register only changes on incoming data, the output maintains its value.

FPGA Design

The FPGA was programmed simply to input a signal from the HC11 and invert the signal. This signal was then sent to a port to control the relays.   We programmed an inverter into an EEPROM and used that to program the FPGA. An important thing to note is that we only used one output pin from the parallel port of the HC11. Thus one bit was used as the selector signal for the relays. The bit chosen was bit 6. Thus when the HC11 had an output of 01000000, the signal into the FPGA would be the logical 1. This logical 1 would then be inverted and used as the selector signal to the transistors controlling the relays. In this example, the bit sequence would turn the relays “on” and thus switch the input.

Java Design

We used Java version 1.1.8 as our base platform for coding. This SDK can be easily downloaded from Sun's java website. The SDK, when properly installed, allows a user to code almost any java application. The SDK provides a compiler for java code as well as libraries to allow a user to implement basic functions. To supplement this SDK we used an API provided by Solutions Consulting(www.serialio.com). This API, named SerialPort, allows java programs to interface and control serial ports on the computer the program is run on. This API is not free however, and a small licensing fee was paid to obtain the software. Sun does provide its own free solution. Sun's java API is called CommApi and can be downloaded from java.sun.com. The API from Solutions Consulting was used due to Taufique's prior experience with it. Both APIs also come with extensive documentation and examples to help a user learn how to use the API.

The use of the API is fairly straightforward. The exact code can be seen in Appendix B. To interface with a serial port, the program must first instantiate a serial port class with a serial configuration object. The user then sets parameters in the configuration object to change the serial port characteristics. At this point the program can perform standard send and receive functions with the serial port. There are many methods to send data, but these methods only put data into the serial buffer. The Txdrain method is used to actually instruct the serial port to send the data. The putByte method was what we used to send bit sequences. To control which port you are controlling, the name of the serial object created must match a name of a port on the system. Thus in our case, the devname parameter seen was COM1.

Results

In theory, our design should have worked perfectly. Though one strange bug caused our system to not function. The hardware worked as intended. Using a manual switch as the selector signal, we were able to successfully switch the input to the host computer between two Ethernet ports. The java program on the host computer sent data successfully through the serial port. The problem we encountered was in the HC11 program. For some reason, the program seemed to hang after it recognized one transfer of serial data. During testing, we were able to verify that the program actually did receive the serial data. This was tested by simply changing the RTI instruction at the end of the code to SWI. By this change, we were able to see that the interrupt service did indeed occur and that the data was sent and interpreted correctly. But when the code was run as originally coded, the system did not recognize any subsequent data after the very first transfer. This was the only critical problem still unsolved.

References

[1] Reference site for small networks http://www.johnscloset.com
[2] Reference for Serial architecture http://www.geocities.com/TheTropics/Shores/2250/pinout.html
[3] Documentation for the Java SDK http://java.sun.com

Parts List

Part	**Source**	**Vendor Part #**	**Price**
5VDC Relay	Radio Shack	275-240A	$1.99
Eight-Conductor Data Grade Telephone Jack	Radio Shack	930-0597	$8.49

Appendices

Appendix A.	Eth.asm – HC11 program to handle serial data transfer
Appendix B.	SetA.java and SetB.java – 2 java programs used to control system

Appendix A.

```
*       Ethernet Control Program
*       Written by Taufiqul Kazi
*       12/04/2000
*
*       This program initializes and uses The SCI functionality of the HC11 to communicate
*       to a host computer and set a control bit.  This control bit is sent via the
*       built in parallell port to an FPGA controlling our switch circuitry.
*
*        The program functions by entering in infinite loop and waiting for an interrupt
*        The interrupt is generated by a compete serial transfer.  The program then runs
*        off to handle the incoming data and move it into the parallel port.

*       Standard Set up

REGS    EQU     $1000
STACK   EQU     $CFFF
PORTB   EQU     $1004
TEMP    EQU     $0000

*       SCI Set up

SCDR    EQU     $2F             * Serial Data register
SCR1    EQU     $2C             * SCI control reg 1
SCR2    EQU     $2D             * SCI control reg 2
SCSR    EQU     $2E             * SCI status reg
BAUD    EQU     $2B             * SCI Baud rate register
BDSET   EQU     %00110101       * Set Baud to 300 bits per second
SCRSET2 EQU    %00100100       * Enable Reciever and interrupt
SCRSET  EQU     $00             * Turn off all interrupt generation
SCRMSK  EQU     %00100000       * Mask used to check for transfer complete flag

*        SCI interrupt setup

        ORG     $00C4           * $00C4 is the Buffalo redirected address for SCI
                interrupts
        JMP      SCIISR

*       Main program

        ORG     $D100
        LDX     #REGS
        LDS     #STACK

*       SCI INIT

*       LDAA    #SCRSET
        CLR     SCR1,x
        LDAA    #SCRSET2
        STAA    SCR2,x
        LDAA    #BDSET
        STAA    BAUD,x
        LDAA    SCSR,x          * Clear any initial flags
        LDAA    SCDR,x

*        Turn on Interrupts

         CLI

*        Spinning Loop

WHEE    BRA      WHEE

SCIISR  BRSET   SCSR,x SCRMSK DATA     * Check that Reciever generated interrupt
        BRA     RETRN           * ignore all other interrupts
DATA    LDAA    SCDR,x          * Finish clearing interrupt flag
        STAA    PORTB           * Store new word to parallel port
RETRN   RTI                       * Return from interrupt service
```

Appendix B.

```
SetA.java
/*-------------------------------------------------------------------------
 * Com Program to set switch to input A
 * This file was built using examples from Solutions Consulting.
 * This program tells the controller to switch to input A, which is no current
 *
 *------------------------------------------------------------------------*/

import java.io.*;
import Serialio.*;

class SetA {

	public static void main(String args[]) {
	SerialPortLocal sp;
	String devName = "";

	if (args.length == 0){
		PrintUsage();
		System.exit(1);
	}
	else {
		devName = args[0];
	}

	try {
	Serialio.SerialConfig serCfg = new Serialio.SerialConfig(devName);
		serCfg.setBitRate(SerialConfig.BR_300);
		serCfg.setDataBits(SerialConfig.LN_8BITS);
		serCfg.setStopBits(SerialConfig.ST_1BITS);
		serCfg.setParity(SerialConfig.PY_NONE);
		serCfg.setHandshake(SerialConfig.HS_NONE);
		sp = new SerialPortLocal(serCfg);
		sp.setTimeoutRx(1000);
		System.out.println("\nSending data...");
		int i = 0;
		sp.putByte((byte)i);
		sp.txDrain();
	}
	catch (Exception ioe) {
		System.out.println(ioe);
		System.exit(1);
		}
}

	static int PrintUsage() {
		 System.out.println("\nThis program sends 00000000 to the HC11.");
		System.out.println("The parameters are 9600 bps, No Parity, 8 data bits, 1 
stop bit.");
		System.out.println("\nThe byte sequence tells the switch to change to 
input A");
		System.out.println("To use COM1, enter this command: java ComTest COM1");
		return 0;
	}
}
```

```
SetB.java
/*--------------------------------------------------------------------------
 * This program is the same as SetA but tells the switch to select inout B
 *
 *
 *
 *------------------------------------------------------------------------*/

import java.io.*;
import Serialio.*;

class SetB {

        public static void main(String args[]) {
        SerialPortLocal sp;
        String devName = "";

        if (args.length == 0){
                PrintUsage();
                System.exit(1);
        }
        else {
                devName = args[0];
        }

        try {
        Serialio.SerialConfig serCfg = new Serialio.SerialConfig(devName);
                serCfg.setBitRate(SerialConfig.BR_300);
                serCfg.setDataBits(SerialConfig.LN_8BITS);
                serCfg.setStopBits(SerialConfig.ST_1BITS);
                serCfg.setParity(SerialConfig.PY_NONE);
                serCfg.setHandshake(SerialConfig.HS_NONE);
                sp = new SerialPortLocal(serCfg);
                sp.setTimeoutRx(1000);
                System.out.println("\nSending data...");
                 int i = 127;
                 sp.putByte((byte)i);
                sp.txDrain();
        }
        catch (Exception ioe) {
                System.out.println(ioe);
                System.exit(1);
                }
}

        static int PrintUsage() {
                 System.out.println("\nThis program sends 01000000 to the HC11");
                System.out.println("The parameters are 9600 bps, No Parity, 8 data bits, 1
stop bit.");
                System.out.println("\nThe byte sequence tells the switch to select input
B");
                System.out.println("To use COM1, enter this command: java ComTest COM1");
                return 0;
        }
}
```