EE 552 Final Report
Netplexor

Submitted by:

Angela Wong	acwong@ualberta.ca
Richard Mao	rmao@ualberta.ca
Colin Durocher	cdd2@ee.ualberta.ca
Jeffrey Spiers	jspiers@yahoo.com
Date:	April 2, 2001
Instructor:	Duncan Elliott

Declaration of Original Content

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

Figure 1: Typical Twisted-Pair Interface and Supply Filtering taken from [5] pg. 47
Figure 2: MII Data Interface taken from [5] pg. 22
Sram controller code will be modified from sram_controller.vhd [2]
CRC generator code will be modified from vcrc32_8.vhd [12]
Hash test bench will be modified from adder_test.vhd [13]

Abstract

Broadband Internet customers who want to make the Internet available to more than one workstation on their home/office network typically must pay extra in order to rent multiple IP addresses from their service provider. With as few as 8 computers sharing the connection, the price of the connection can double. For this reason, many of these users have turned to a technique called IP masquerading, which allows the seamless sharing of the Internet connection while looking to the outside world like a single workstation. This usually requires a dedicated computer with 2 Ethernet cards, typically running the Linux operating system. This is not always easy to setup. The Netplexor will be an easy to use IP masquerading solution for these users. The prototype as developed in this course will have certain limitations in order to fit within the space constraints of the Altera FLEX10K20 FPGA device as well as to meet our team's time constraints. Curently, a maximum of fifteen computers can be connected to the Netplexor, and a maximum of 256 different connections (IP's) can be stored in the IP LUT. These limitations should be acceptable in the home or small office environment to which this product is targetted.

Table of Contents

Achievements

At this point of the project, we have achieved some of our goals, but others remain to be fulfilled.

Functional

- The two top-level entities, Ethernet and Masquerade, individually compile and simulate with no known bugs.
    - Ethernet simulations indicate the ability to send and receive data and convert it to byte format for ease of interpretation by Masquerade
    - Masquerade has been simulated and appears functional in both the masquerade and demasquerade directions

- Netplexor, which instantiates the both the Ethernet and Masquerade components compiles, and fits on the Altera 10K20 FPGA with the proper Synthesis and fitting options selected in MaxPlus II. Netplexor has been simulated and the whole system appears to function correctly.

- The SRAM interface has been tested in the laboratory and it appears to function properly, though it has not yet been tested with the Masquerade entity that uses it.
    - The test performed involved storing three bytes of data at memory addresses that were set using DIP switches, then retrieving the correct data from those addresses and displaying them on LEDs.

- The Ethernet module's receive functionality has been tested in the lab and is known to function correctly. The transmit functionality was tested as well. While it was possible to confirm that bytes were being transmitted, the test was inadequate to ensure that the correct bytes were being transmitted.

Untested

- Netplexor has not yet been tested in the lab. Two logistical issues have thus far prevented this:
    - Obtaining a full duplex 10 Mbit/s switch for testing and demo purposes has proven to be difficult. The alternative is to add code to Netplexor to buffer an entire packet before sending it out. In order to do this, and conform to the IEEE 802.3 standard, Netplexor would have to implement backoff logic to minimize ethernet collisions. Since only 8% of the Altera 10K20 remains free, this would be very difficult, if not impossible, to implement.
    - In whatever laboratory the Netplexor FPGA is programmed, our group also requires a computer with software capable of sniffing packets on the network in order to verify that the correct fields in the ethernet and IP headers are being replaced.

Further Work

- Netplexor should respond to ARP packets
- Netplexor should implement DHCP in order to automatically request an IP from the ISP. Currently, the user must recompile Netplexor after changing a constant in Masq_pkg.vhd.
- Netplexor should drop any packets which are not IP packets addressed to its MAC.

Description of Operation

Overview

On an IP based network like the Internet, every node in the network must have its own IP address. Due to the limited number of addresses and the immense growth of the network over the last decade, IP addresses are becoming more and more expensive. Currently, cable and ADSL providers charge about $5.00 per address / month. It is therefore desirable to share IP addresses where practical. IP masquerading is one way of doing this.

The basic idea behind IP masquerading is to have all connections to the external network pass through a gateway which substitutes the source address and port number of each TCP/IP packet with its own address and a new port number. To the external network, all traffic then seems to be coming from a single node on the network even though multiple computers are actually making requests. In order to properly route packets coming back from the external network, a table is kept in memory which links the new port number, old port number, and real IP address together. The gateway can then look at the port number on the incoming packet (which is one it made up), look it up in the table, and then substitute the destination IP and port fields in the packet with their original values and send the packet to the appropriate node on the internal network. An added benefit of this technique is that all connections must be initiated by computers on the internal network, leading to a high degree of security from network intrusions.

Netplexor Implementation Description

Due to resource limitations, the initial version of the Netplexor will take a slightly less sophisticated approach to masquerading. Instead of using the source/destination port numbers to route packets, it will use the source/destination IP addresses to route packets. This means that the Netplexor will be keeping two lookup tables implemented in a single 256kb asynchronous SRAM. One will be a set of tuples consisting of <destination IP, source IP> while the other will be a set consisting of <source IP, source MAC>. When a connection is initiated from the internal network, Netplexor will examine the ethernet and IP headers and store the <destination IP, source IP> in the Src IP - Dest IP table and the <source IP, source MAC> in the Src IP – Src MAC table. When a packet is received from the external network (Internet), its source IP will be examined and compared to each of the destination IP's in the table. When a match is found, the destination IP in the packet will be replaced with the source IP entry in the table. The destination MAC address contained in the ethernet header will also be replaced with the appropriate MAC from the lookup table so that the ethernet frame can be forwarded to the correct node on the internal network. A maximum of 16 computers can be connected to the Netplexor and a total of 256 connections can be stored in the IP LUT. Due to the nature of the hash functions, it is possible that two connections have the same hash index. The hash function for the IP LUT hashes based on each byte of the external IP. Each of the four bytes in the ip is XORed together to yield an 8-bit value which is used as the least significant 8 bits of the SRAM's 15 bit address line. The hash function for the MAC LUT hashes based on the least significant byte of the internal IP. The upper four bits and lower four bits of this byte are XORed together and used as bits 7 to 3 of the SRAM's address line and bits 14 to 8 are preset to '1' so that the MAC LUT is stored in higher addresses of the SRAM. Bits 2 to 0 of the address are determined by the offset value of the MAC to be stored. For instance, the third byte of the MAC will be stored in address "1111 1111 HHHH 011" where HHHH is the value obtained by XORing IP(7 downto 4) with IP(3 downto 0). This is a very simple hash and where duplicate addresses are found, the old value is simply replaced. In making this compromise, our design relies on the TCP protocol to ensure communication reliability where that reliability is needed. Due to the SRAM size, FPGA size, and time constraints of this project, this is a suitable compromise.

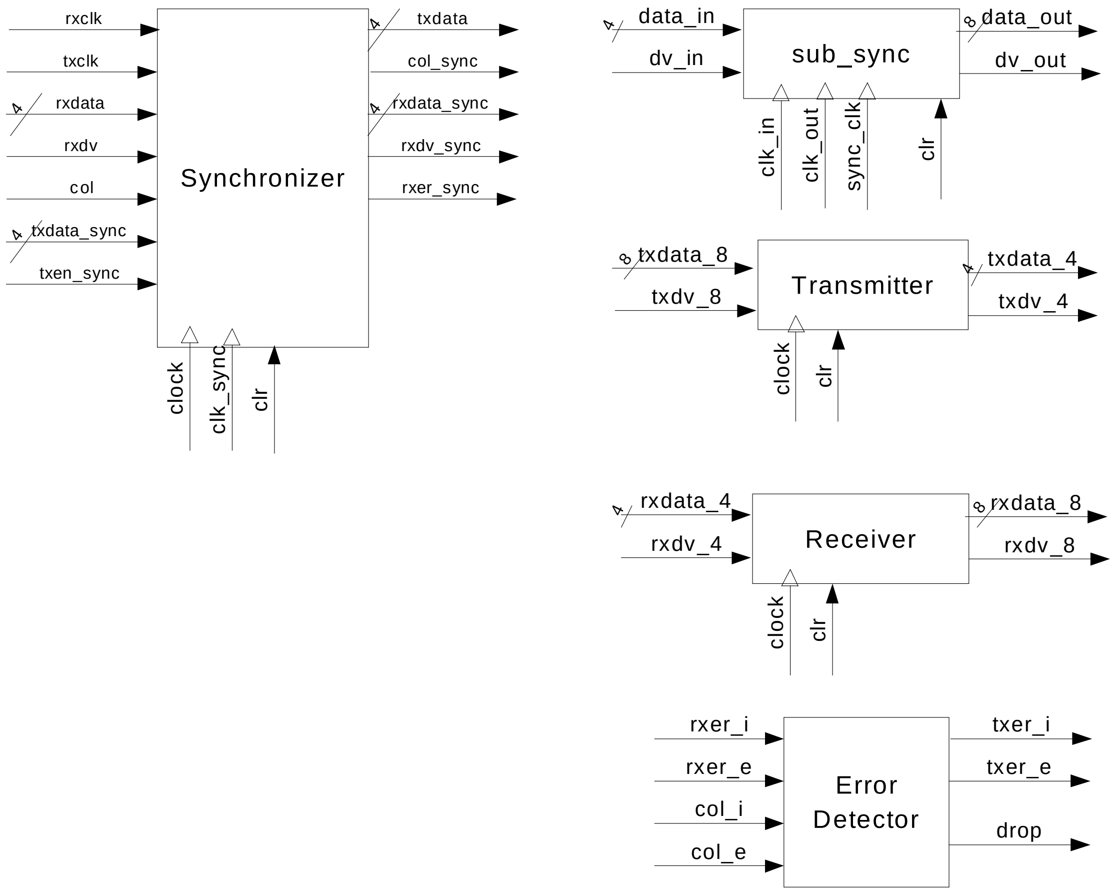
The drawback to this method, and the reason it is considered to be less sophisticated is that two computers on the internal network cannot simultaneously connect to a single IP on the external network. The motivation for implementing masquerading in this way is that all routing is done at the network and link layers instead of at a combination of transport, network, and link layers. This means that the Netplexor will not have to explicitly understand the TCP, UDP and ICMP protocols in order to handle all types of Internet traffic. The result is a less complex product implemented in fewer gates.

Usage

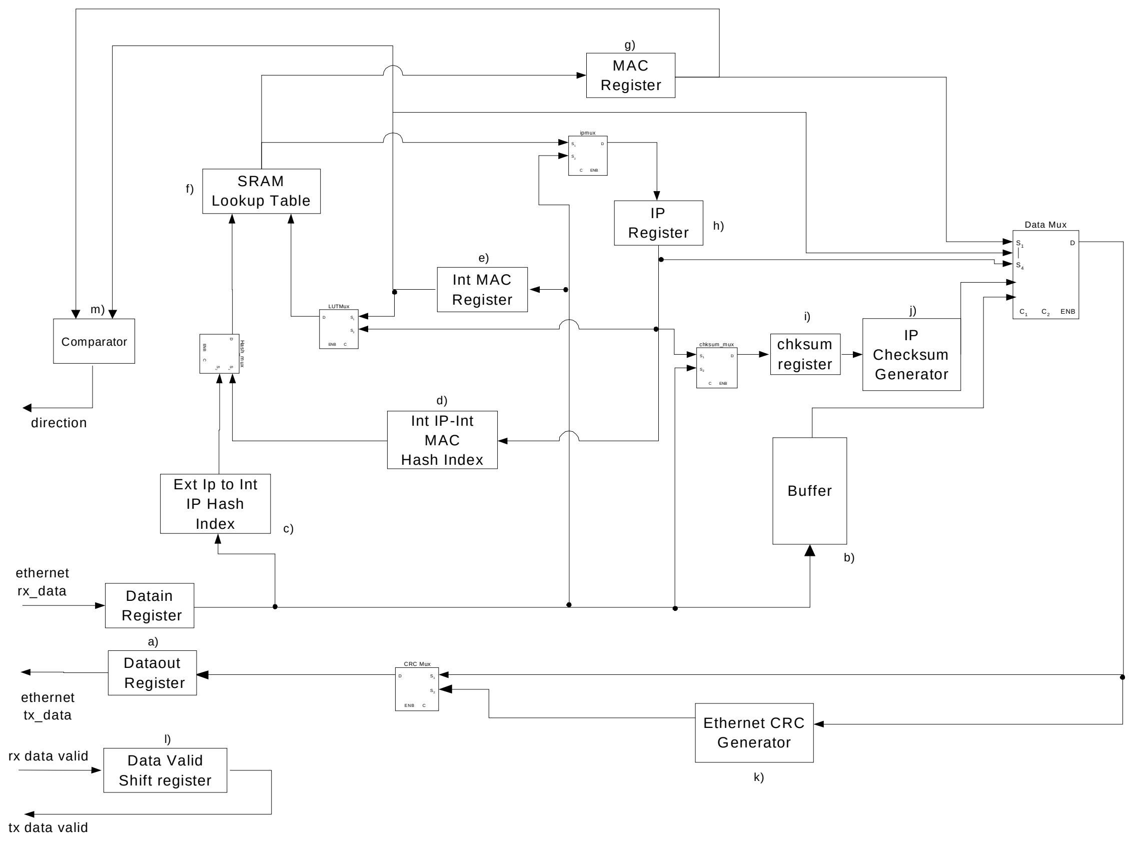
Because, in the current prototype of Netplexor, the cable modem MAC address and Netplexor externally visible IP address are hardcoded, this product is not yet at the stage of development where it would be easily employed by members of the public – One of the Netplexor project goals. In order to use this prototype, the user would need access to the source code and programming hardware in order to change these two constants to match their particular home network. In order to achieve the goal of ease of use, Netplexor would have to initially broadcast internet-bound frames and sniff the cable modem MAC address from returned packets. It would also need to either support the whole DHCP protocol or simply sniff DHCP packets to obtain its IP address from the service provider. Finally, Netplexor would have to respond to ARP packets on the internal network so that computers on that network could translate Netplexor's IP address into a MAC address at the ethernet level.

Netplexor Implementation

Ethernet Components



Masquerade Datapath

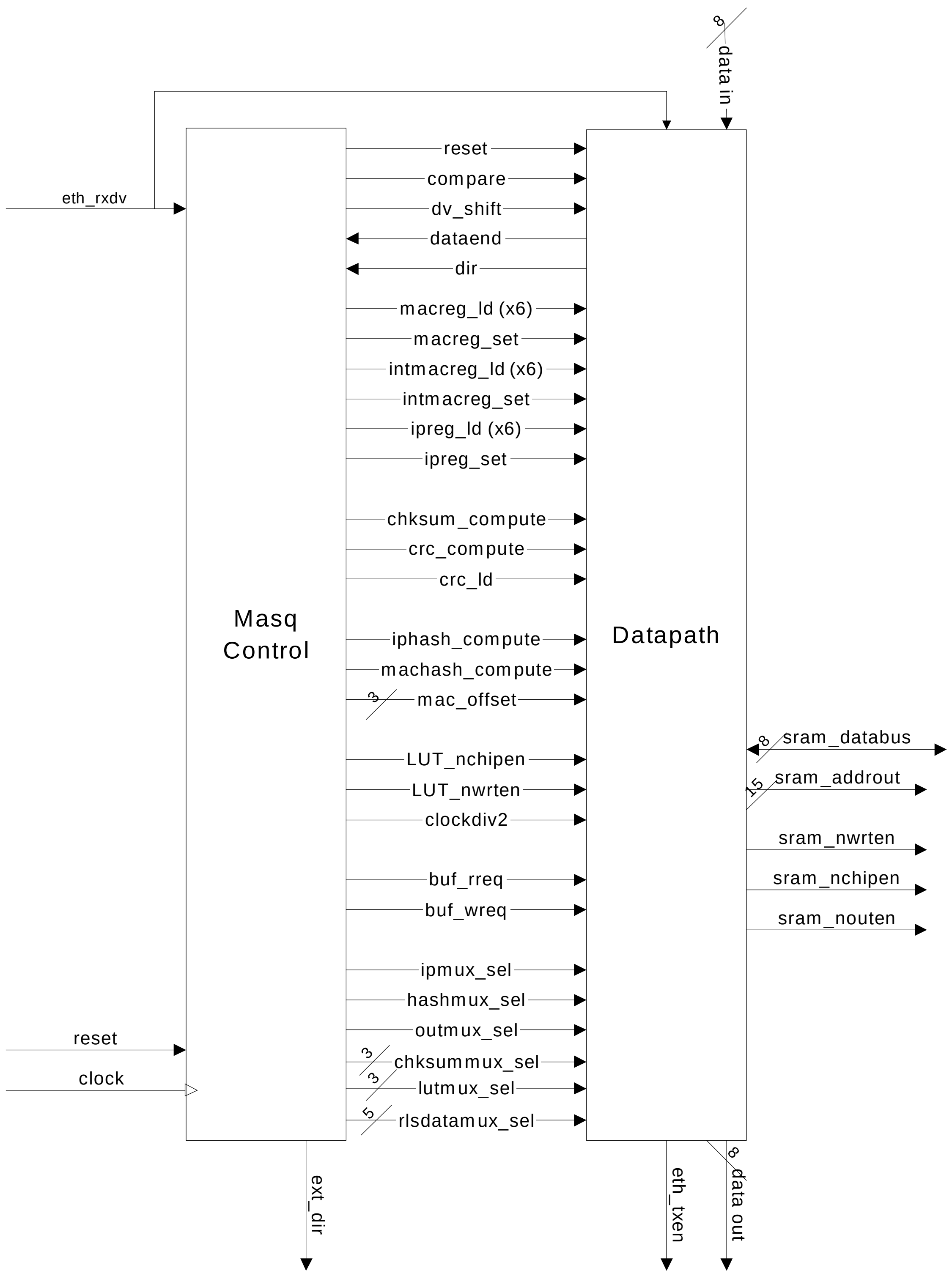


Data path component descriptions:

a) *Data in / data out registers*: incoming and outgoing data for transfer from / to Ethernet component.
b) *Buffer*: Stores frame data until ready for transfer. The current design requires 45 bytes of data to be buffered.
c) *External IP to internal IP hash*: Takes in the external IP from the data path and provides the index to the lookup table to read or write the internal IP depending on the direction. Only the least significant byte of the IP address will be stored.
d) *Internal IP to internal MAC hash*: Takes in the internal IP from the IP register and provides the index to the lookup table to read or write the internal MAC addresses. In order to store all 6 bytes of the MAC address, the hash function includes a control signal to provide an offset of 6 to the generated index.
e) *Internal MAC register*: Holds the internal MAC address from the data path until the hash index has been calculated for storage into the lookup table, and for comparison with the MAC address of the router. It can also can be set to the global shared MAC address (the MAC of our externally transmitting Ethernet transceiver) , which will replace the internal MAC address in the Ethernet frame.
f) *SRAM controller*: Accesses SRAM where 2 lookup tables are stored. The first contains the least significant byte of the internal IP address, and the second stores the internal MAC addresses.
g) *MAC register*: Holds the internal MAC as read from the lookup table or can be set to the global destination MAC of the router of the external network, for substitution into the Ethernet frame.

h) *IP register*: Holds the internal IP address for storage into the lookup table, the external IP address for generating a lookup table index, or the IP address read from the lookup table for calculating the IP checksum.
i) *Checksum register*: Holds 8 bit data so that it can be sent to the checksum generator as 16 bits.
j) *IP checksum generator*: Incrementally calculates the IP header checksum.
k) *Ethernet CRC generator*: Calculates the CRC to be appended to the Ethernet frame.
l) *Receive data valid shift register*: The data valid bit is shifted in from the Ethernet receive side, and shifted out to the Ethernet's transmit side.
m) *Comparator*: Compares the source MAC of the incoming packet with the hard coded router MAC to determine the direction of the packet.

Masquerade Controller/Datapath Interface



Netplexor Datasheet

Description

The Netplexor allows multiple internal network computers to share an Internet connection using only one ISP account, a full duplex switch, and a Cable or DSL modem. To configure Netplexor, users need to determine the IP address that their ISP has assigned to them as well as the MAC address of their cable or ADSL modem. Currently, these values must be hardcoded into Masq_pkg.vhd. The Netplexor project must then be recompiled from the top-level and an FPGA must be programmed. Netplexor should then be placed in the following network topology using a Class C address space for the internal network.

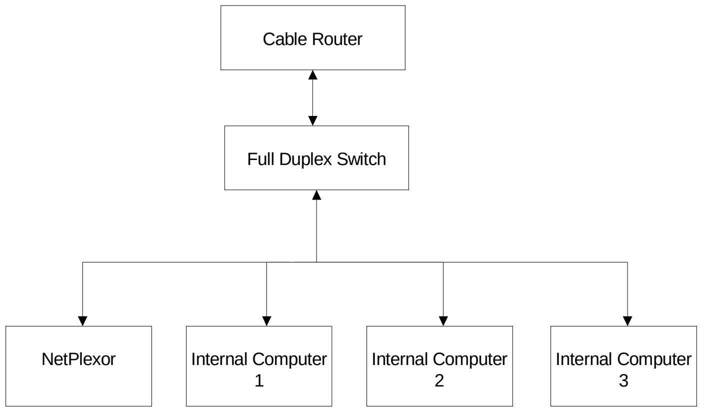


Figure 1: Netplexor Usage Diagram

Features

- Allows as many as 16 internal network computers and 256 simultaneous connections to share one ISP account
- Increased network security since all connections must be initiated by computers on the internal network
- Compatible with IEEE 802.3 Ethernet: 10BaseT and 10BaseFX
- Supports 10Mbit/s operation with any transceiver that supports 40-pin MII interface
- Maximum Ethernet Diameter: 100m
- Uses Altera Flex10K20 FPGA on UP1 Prototype Board, IDT 71256 SRAM, and IEEE802.3u compliant Ethernet transceiver

Connector Pinout

RJ45 10BaseT Connector Pinout:

Pin	Name	Description
1	TX+	Transmit Data +
2	TX-	Transmit Data -
3	RX+	Receive Data +
4	N/C	Not Connected
5	N/C	Not Connected
6	RX-	Receive Data -
7	N/C	Not Connected
8	N/C	Not Connected

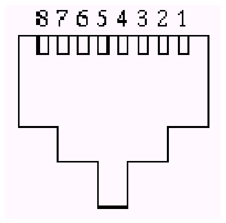


Figure 2: RJ45 connector pin numbering as seen looking at the front of a male connector. Ref [15]

MII Connector Pinout:

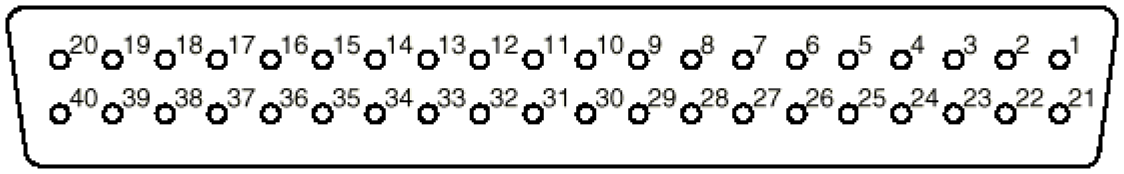


Figure 3: MII connector pin numbering as seen looking into the contacts of a female connector from the mating side. Ref [11]

Pin	**Name**	**Description**	**Pin**	**Name**	**Description**
1	+5 V	VCC	21	+5 V	VCC
2	MDIO	Not connected	22	COMMON	GND
3	MDC	Connected to ground via 1.5 kΩ	23	COMMON	GND
4	RXD(3)	Data nibble being received	24	COMMON	GND
5	RXD(2)	Data nibble being received	25	COMMON	GND
6	RXD(2)	Data nibble being received	26	COMMON	GND
7	RXD(1)	Data nibble being received	27	COMMON	GND
8	RX_DV	Receive data valid	28	COMMON	GND
9	RX_CLK	Receive clock	29	COMMON	GND
10	RX_ER	Not connected	30	COMMON	GND
11	TX_ER	Not connected	31	COMMON	GND
12	TX_CLK	Transmit clock	32	COMMON	GND
13	TX_EN	Transmit enable	33	COMMON	GND
14	TXD(0)	Data nibble being transmitted	34	COMMON	GND
15	TXD(1)	Data nibble being transmitted	35	COMMON	GND
16	TXD(2)	Data nibble being transmitted	36	COMMON	GND
17	TXD(3)	Data nibble being transmitted	37	COMMON	GND
18	COL	Indicates a collision	38	COMMON	GND
19	CRS	Not connected	39	COMMON	GND
20	+5 V	VCC	40	+ 5 V	VCC

FPGA and UP1 Header Pinouts:

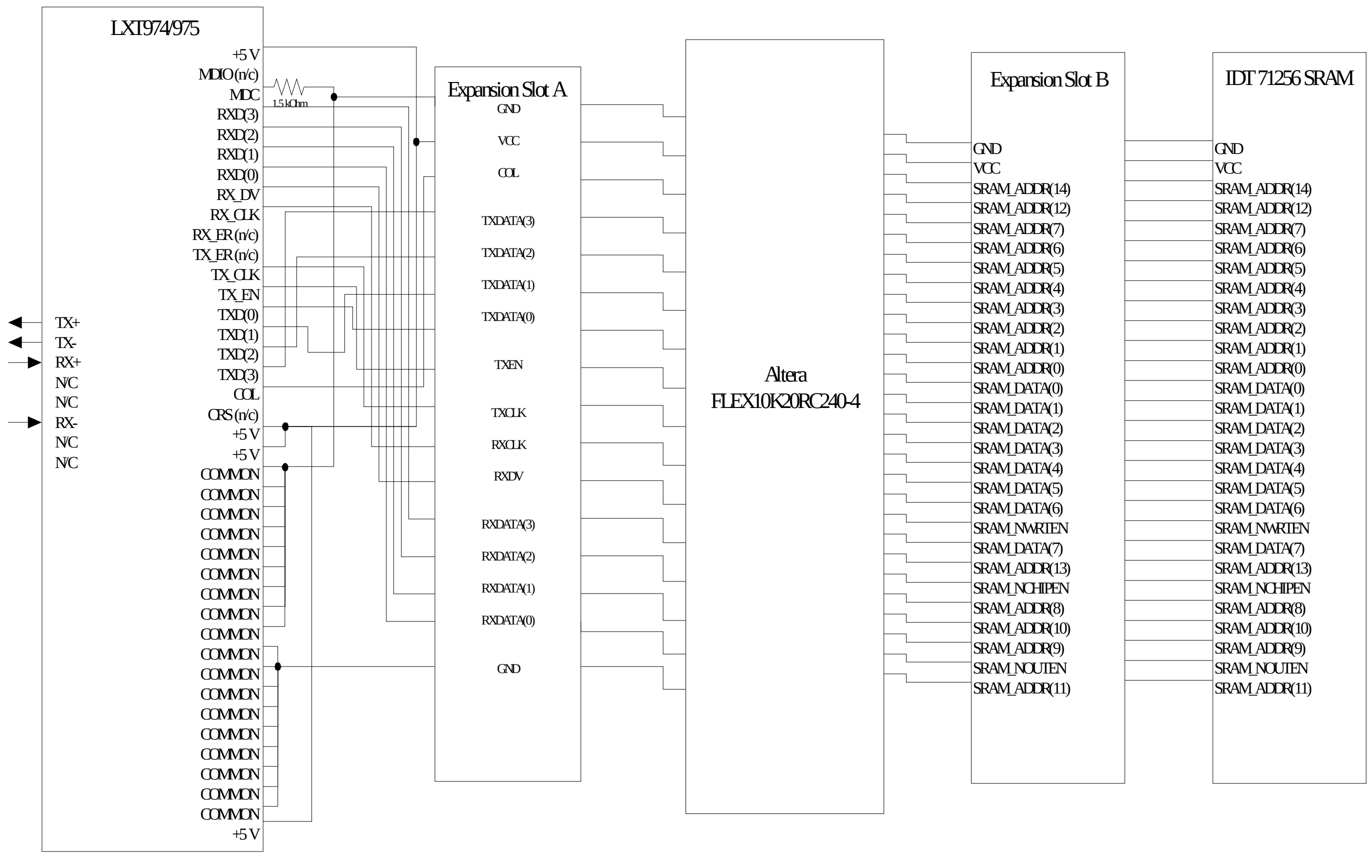


Figure 4: Schematic of FPGA connections to MII interface and SRAM

Pin Name	Pin No	Hole No	Type	Description
RESET	28	FLEX_PB1	I	Global reset
GND	GND	A 2	O	Ground
GND	GND	A 60	O	Ground
GND	GND	B 2	O	Ground
VCC	VCC	B 3	O	VCC
VCC	VCC	A 3	O	VCC
VCC	VCC	A 57	O	VCC
SYSCLK	91	Global Clock	I	25.175 MHz clock
TXCLK	35	A 39	I	TXDATA and TXEN are synced to these clocks
TXEN	31	A 35	O	High when TXDATA is valid
TXDATA(3)	9	A 19	O	Data nibble being transmitted
TXDATA(2)	15	A 23	O	Data nibble being transmitted
TXDATA(1)	20	A 27	O	Data nibble being transmitted
TXDATA(0)	24	A 31	O	Data nibble being transmitted
RXCLK	39	A 43	I	RXDATA and RXDV are synced to these clocks
RXDV	45	A 47	I	High when RXDATA is valid

RXDATA(3)	50	A 49	I	Data nibble being received
RXDATA(2)	52	A 51	I	Data nibble being received
RXDATA(1)	54	A 53	I	Data nibble being received
RXDATA(0)	56	A 55	I	Data nibble being received
COL	45	A 15	I	Indicates collision
SRAM_ADDR(14)	109	B 15	O	Address line
SRAM_ADDR(13)	156	B 50	O	Address line
SRAM_ADDR(12)	111	B 17	O	Address line
SRAM_ADDR(11)	163	B 56	O	Address line
SRAM_ADDR(10)	159	B 53	O	Address line
SRAM_ADDR(9)	161	B 54	O	Address line
SRAM_ADDR(8)	158	B 52	O	Address line
SRAM_ADDR(7)	114	B 19	O	Address line
SRAM_ADDR(6)	116	B 21	O	Address line
SRAM_ADDR(5)	118	B 23	O	Address line
SRAM_ADDR(4)	120	B 25	O	Address line
SRAM_ADDR(3)	127	B 27	O	Address line
SRAM_ADDR(2)	129	B 29	O	Address line
SRAM_ADDR(1)	132	B 31	O	Address line
SRAM_ADDR(0)	134	B 33	O	Address line
SRAM_DATA(7)	154	B 49	I/O	Data from/to SRAM
SRAM_DATA(6)	152	B 47	I/O	Data from/to SRAM
SRAM_DATA(5)	149	B 45	I/O	Data from/to SRAM
SRAM_DATA(4)	147	B 43	I/O	Data from/to SRAM
SRAM_DATA(3)	144	B 41	I/O	Data from/to SRAM
SRAM_DATA(2)	142	B 39	I/O	Data from/to SRAM
SRAM_DATA(1)	139	B 37	I/O	Data from/to SRAM
SRAM_DATA(0)	137	B 35	I/O	Data from/to SRAM
SRAM_NCHIPEN	157	B 51	O	Active low chip select
SRAM_NWRTEN	153	B 48	O	Active low write enable
SRAM_NOUTEN	162	B 55	O	Active low output enable

Resource Requirements

The following resources were used when Netplexor was configured on the EPF10K20RC240-4:

```
Total dedicated input pins used:                  3/6      ( 50%)
Total I/O pins used:                             52/183    ( 28%)
Total logic cells used:                        1095/1152   ( 95%)
Total embedded cells used:                        8/48     ( 16%)
Total EABs used:                                  1/6      ( 16%)
Average fan-in:                                3.06/4      ( 76%)
Total fan-in:                                3271/4608     ( 70%)
```

Electrical Characteristics

Input Power Requirements:	+5V to Altera UP1
Absolute Maximum Power Ratings:	Tolerance –2 to 7 V

Timing Considerations

As long as RXDV and RXDATA are valid on the rising edge of the RXCLK and TXEN and TXDATA are valid on the rising edge of the TXCLK then Netplexor will take care of timing concerns such as synchronizing.

Experiments & Characterizations

Netplexor Area

It was necessary to experiment with different optimization settings to make our design fit onto the chip. Various settings were attempted, however only one combination was able to successfully fit

Area / Speed Slider Setting	Automatic Fast I/O	Automatic Register Packing	Automatic Open Drain	Automatic Implement in EABs	Try Harder	Logic Cells (Out of 1152)	Percentage	EABs (Out of 6)
5	No	No	No	No	No	Does Not Fit	> 100%	N/A
0	Yes	Yes	Yes	No	No	Does Not Fit	> 100%	N/A
5	Yes	Yes	Yes	No	No	1095	95%	1
5	Yes	Yes	Yes	No	Yes	1095	95%	1
10	Yes	Yes	Yes	No	No	Does Not Fit	> 100%	N/A
5	Yes	Yes	Yes	Yes	No	N/A	N/A	Does Not Fit

Buffer Size

We tested various implementations of the FIFO component in the LPM to use as our buffer. The two that were most suited for our use was the lpm_fifo and the csfifo. Although Altera recommends using the csfifo, as opposed to using the lpm_fifo, with FLEX10K devices, this component requires 2 clocks. A regular clock, and another at double the frequency. Characterizations of these components were necessary to determine if the space savings would be worth the added complexity of using 2 clock signals. As it turns out the csfifo was much smaller but slower ($T \approx 20$ns) compared to the lpm_fifo ($T \approx 10.9$ns). Since speed isn't a concern for us but area is, we decided to use the csfifo after all.

csfifo (8x45)

Logic Cells	**Percentage**	**EABs**
42	3%	1

lpm_fifo

Logic Cells	**Percentage**	**EABs**
232	20%	2

CRC Generator Characterization

In order to verify that the CRC generator did indeed generate the correct Ethernet CRC, the outputs of the module in simulation were compared against the outputs of a C program [14], which implemented the CRC generating algorithm.

References

Background information was gathered from the following websites:

[1] T. Bensler and E. Chan. Interfacing External SRAM.
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1999_w/SRAM/ (accessed Feb. 6, 2001).

[2] T. Bensler and E. Chan. SRAM Controller.
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1999_w/SRAM/ (accessed Jan. 30, 2001).

[3] R. Droms. Dynamic Host Configuration Protocol. ftp://ftp.isi.edu/in-notes/rfc2131.txt (accessed Jan. 21, 2001).

[4] IDT. 71256L Datasheet.
http://www.idt.com/docs/71256L_DS_17524.pdf (accessed Feb. 5, 2001).

[5] Intel. LXT974/LXT975 Datasheet.
ftp://download.intel.com/design/network/products/lan/datashts/24927401.pdf (accessed Feb. 3, 2001).

[6] National Semiconductor. DP83846A DsPHYTER – Single 10/100 Ethernet Transceiver [Preliminary].
http://www.national.com/pf/DP/DP83846A.html (accessed Jan. 20, 2001).

[7] J. Postel. Internet Protocol. ftp://ftp.isi.edu/in-notes/rfc791.txtRFC Editor (accessed Jan. 21, 2001).

[8] J. Postel. Transmission Control Protocol. ftp://ftp.isi.edu/in-notes/rfc793.txt (accessed Jan. 21, 2001).

[9] M. Smith. Ethernet.
http://www-ee.eng.hawaii.edu/~msmith/XCoNET/Ethernet.htm (accessed Jan. 15, 2001).

[10] University of Hawaii. IEEE 802.3 Ethernet Standard 1996.

[11] University of Hawaii. IEEE 802.3u 10/100 Ethernet Standard 1995.

[12] Vautomation. Vautomation Free Cores. http://www.vautomation.com/free/vcrc32_8.vhd (accessed Jan. 30, 2001).

[13] J. Koob, R. Sung, D. Elliott. Adder_test.vhd.
http://www.ee.ualberta.ca/~elliott/ee552/labs/lab5/adder_test.vhd (accessed Mar. 10, 2001).

[14] C.M. Heard, crc32h.c
http://cell-relay.indiana.edu/cell-relay/publications/software/CRC/32bitCRC.c.html (Accessed Mar16th, 2001).

[15] S. Caplan et al. eSafe Final Report. http://www.ee.ualberta.ca/~elliott/ee552/projects/1998f/esafe/esafe-final.pdf (accessed Mar.30, 2001).

Datasheets for Chips used

The first pages of the datasheet for the IDT 71256L SRAM are attached. A table of the Flex10K device features is also attached. For the Ethernet transceiver we are using the Addtron AEF-100MT (100 BaseTX to MII mini-transceiver). No datasheet is available for this part because it is a completely integrated part. To interface with it refer to IEEE 802.3u for specifications of the MII transceiver.

Design Verification

Two Simultaneous Connections

This simulation demonstrates the fundamental purpose of Netplexor. Two simultaneous connections are masqueraded and demasqueraded between two separate internal clients and two separate external Internet servers, thus “multiplexing” Internet connections.

First, client 1 sends an IP packet destined for server 1. The Netplexor intercepts the packet and modifies (masquerades) its source MAC address, destination MAC address and source IP address before forwarding it on. Next, client 2 makes a similar request to server 2. It is also masqueraded by the Netplexor. The response from server 1 is then intercepted on the return path. It is demasqueraded and sent to client 1. The same thing happens for the response from server 2 to client 2.

The client IP addresses are 4.3.2.35 and 4.3.2.67, and the server IP addresses are 152.186.220.254 and 101.135.169.203. The Netplexor uses IP address 4.3.2.1 both internally and externally. This is the only publicly known IP address.

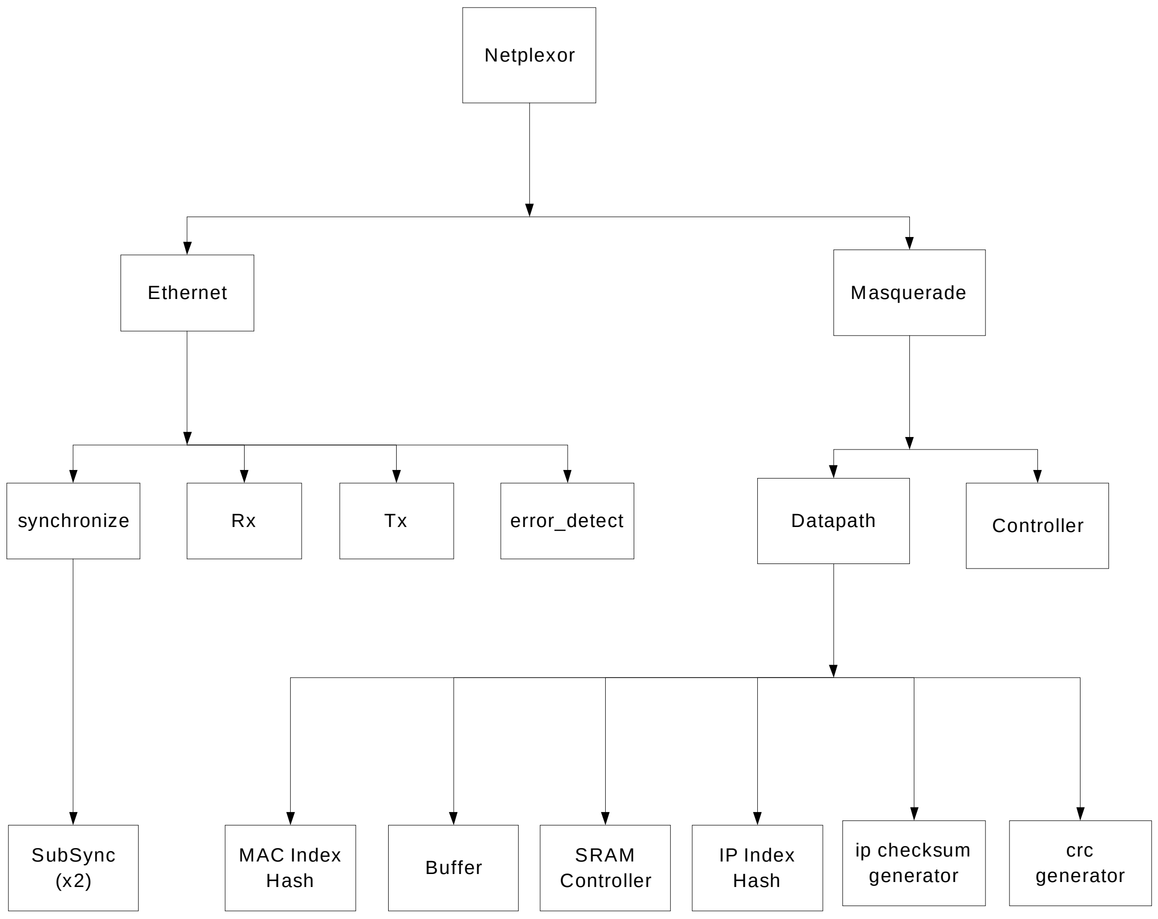
The cable/ADSL modem and Netplexor MAC addresses have been hard coded in this prototype design. A production version of Netplexor would determine the cable/ADSL modem MAC address by monitoring traffic on the external network. The hard-coded cable/ADSL modem MAC address is 0F:0E:0D:0C:0B:0A and the Netplexor MAC address is 09:08:07:06:05:04.

A summary of the simulation is presented in the table below. It shows the source and destination MAC and IP addresses of packets before and after Netplexor masquerades or demasquerades them.

Summary of Simulation

Packet	**Client 1 Request**		**Client 2 Request**		**Server 1 Response**		**Server 2 Response**	
	In	**Out**	**In**	**Out**	**In**	**Out**	**In**	**Out**
Source MAC	65:87:A9: CB:ED:0F	0F:0E:0D: 0C:0B:0A	23:76:45: DC:EF:BA	0F:0E:0D: 0C:0B:0A	09:08:07: 06:05:04	0F:0E:0C: 0C:0B:0A	09:08:07: 06:05:04	0F:0E:0C: 0C:0B:0A
Destination MAC	A9:CB:ED :0F:21:43	09:08:07: 06:05:04	A9:CB:ED :0F:21:43	09:08:07: 06:05:04	0F:0E:0D: 0C:0B:0A	65:87:A9: CB:ED:0F	0F:0E:0D: 0C:0B:0A	23:76:45: DC:EF:BA
Source IP	0xC0A80123 192.168.1.35	0x04030201 4.3.2.1	0xC0A80143 192.168.1.67	0x04030201 4.3.2.1	0x98BADCFE 152.186.220.254	0x98BADCFE 152.186.220.254	0x6587A9CB 101.135.169.203	0x6587A9CB 101.135.169.203
Destination IP	0x98BADCFE 152.186.220.254	0x98BADCFE 152.186.220.254	0x6587A9CB 101.135.169.203	0x6587A9CB 101.135.169.203	0x04030201 4.3.2.1	0xC0A80123 192.168.1.35	0x04030201 4.3.2.1	0xC0A80143 192.168.1.67

Design Hierarchy



Index to VHDL code

Main Code

Netplexor: top-level file that provides interface between the Ethernet and Masquerade components
- compiled – no errors

Eth_pkg: package with all Ethernet components
- compiled – no errors

Ethernet: Ethernet component that encompasses all the Ethernet functionality for Netplexor
- simulated – no known bugs

Receiver: converts 4-bit rx signal to 8-bit and transmits it every second clock cycle
- simulated – no known bugs

Transmitter: converts 8-bit tx signal to 4-bit transmit data
- simulated – no known bugs

Synchronizer: synchronizes all Ethernet transceiver signals to a common clock
- simulated – no known bugs

Sub Sync: Connects to either internal receive, internal transmit, external receive, or external transmit
- simulated no known bugs

Error-detector: detects and signals errors
- simulated – no known bugs

Masq_pkg: package with all the components and constants required by the Masquerade components
- compiled – no errors

Masquerade: Masquerade component that instantiates the controller and datapath
- not compiled

Controller: contoller for Netplexor
- compiled no errors

Datapath: data path for Netplexor.
- compiled but missing checksum generator

IP Index Hash: provides an index to the destination IP to source IP lookup table
- Simulated – no known bugs

MAC Index Hash: provides an index to the source IP to source MAC lookup table
- Simulated – no known bugs

Buffer: used to buffer data until ready for transmission
- conpiled – no errors

IP Checksum
- compiled – no errors

Code for testing:

Repeater => substitutes for masquerader to implement an Ethernet repeater
- Simulated – no known bugs

Memory Test => used to test the interface to the SRAM
- Simulated – no known bugs

Code used from other sources:

SRAM controller=> SRAM interface [2]. Reused code.   No modifications required (so far)
- compiled

vcrc32_8=> 8-bit parallel 32-bit CRC generator for Ethernet [12].   Reused code. No modifications required (so far)
- compiled

VHDL design

- VHDL code for the entities listed above is presented in the next 50+ pages.

Test Bench Index

Hash _Test: Randomly feeds values to the either of the IP Hash or the Mac Hash components. Modified from the ee552 Lab 5 adder test bench[13].

Schematics

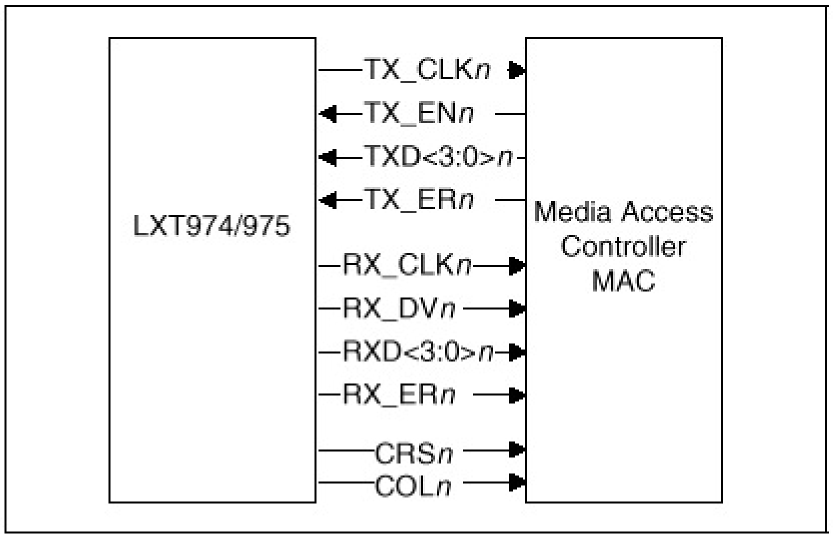


Figure 3: MII Data Interface [5]

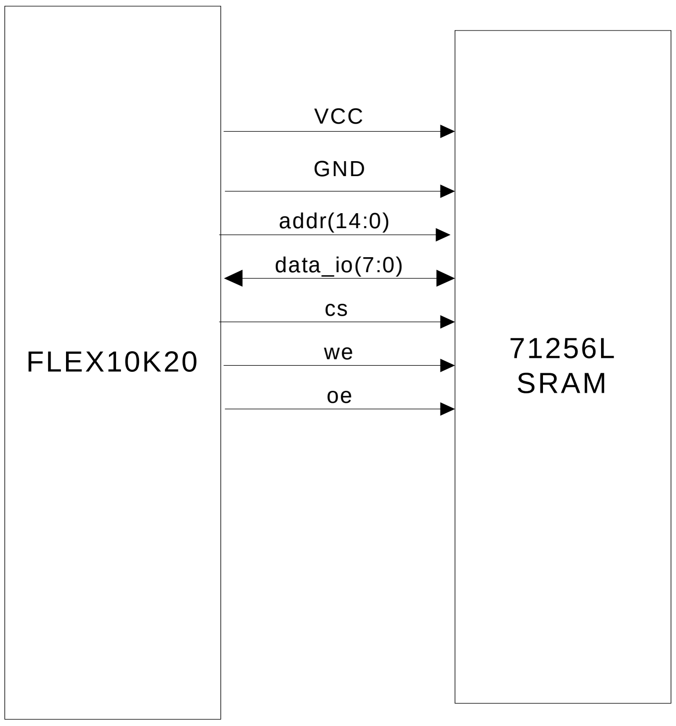


Figure 4: SRAM Interface