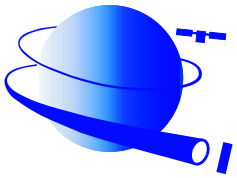
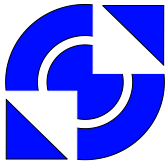


TELECOMMUNICATION
ENGINEERING



UNIVERSITY
of
TWENTE

University of Twente
Department of Electrical Engineering
Chair for Telecommunication Engineering

Hardware integration for a passive optical communication network

by

Frank R. Ellenbroek

Master thesis
Executed from July 1, 2004 to April 27, 2005

Supervisor: prof.dr.ir. W. van Etten
Advisors: dr.ir. C. Roeloffzen
dr. R. Srinivasan

Summary

With today's expanding communication capabilities, the need for bandwidth is continously increasing. Current Local Area Networks (LANs) predominantly use copper wires to transfer data between nodes, but these systems require new types of cable with each generation. In contrast, optical fibers provide a huge bandwidth which should be sufficient for future generations of communication systems.

During the past six years the EWI-TE Group at the university of Twente has designed an optically transparant network for local area applications, called the MOUSE project. The network is based on the Fast Ethernet protocol, to which a number of novel ideas are added to allow the system to work in a passive optical (shared) medium. Most of the basic hardware components for the system have already been designed by different students during their (master) assignments, but these components have not been connected together to form a working system. It is known that not all components function correctly. The goal of the assignment is to interconnect the different components in order to create a working system, to validate the proper operation and to assess the performance of the system.

During this assignment, the different components of the MOUSE system have been assembled to provide a working system. A number of problems in the MOUSE system have been found, which impacted the reliable operation of the system. A number of new PCBs have been designed which provide practical solutions to the encountered problems. Ultimately, the communication between the different components has been established, and simulations and measurements have been made to confirm the proper operation of the system.

Contents

List of Abbreviations

ARP	Address Resolution Protocol
COL	COLlision
CRS	CaRrier Sense
CSMA/CD	Carrier Sense Multiple Access with Collision Detect
ESD	End-of-Stream Delimiter
FCS	Frame Check Sequence
FPGA	Field Programmable Gate Array
ICMP	Internet Control and Management Protocol
IEEE	Institute of Electrical and Electronic Engineers
LAN	Local Area Network
LE	Logic Element
LLC	Logical Link Control
LUT	Look-Up-Table
MAC	Medium Access Control
MDC	Management Data Clock
MDIO	Management Data Input Output
MII	Medium Independant Interface
MOUSE	MultimOde Upgrade of Star-shaped Ethernet
NRZI	Non Return To Zero with Inversion
PHY	PHYsical layer device
PHYAD	PHYsical ADdress
PMD	Physical Media Dependant
POSC	Passive Optical Star Coupler
REGAD	REGister ADdress
SFD	Start-of-Frame Delimiter
SLA	Signaltap Logic Analyzer
ST	STart-of-frame
STA	STAtion management
TA	Turn-Around
TL	Transmission Line
VHDL	Very high speed integrated circuit Hardware description language
WDM	Wavelength Division Multiplexing

Chapter 1

Introduction

1.1 Project Description

Communication is becoming increasingly important in every aspect of our lives. The internet has and will continue to change the way we work, communicate and recreate. The possibilities are numerous, and Voice-over-IP, Video on demand and Teleconferencing are only a few of the emerging technologies which drive the need for more and more bandwidth. While the high-speed infrastructure for the internet (and also other networks) consists predominantly of fiber optic systems, the end-user is mostly connected with low-cost copper cables to an Ethernet based Local Area Network (LAN). The most important benefits of optical fiber compared to copper cables is the high bandwidth, the high reliability and the low attenuation. The low cost of the hardware in copper based networks is offset by the cost required to provide a new infrastructure for each new network generation. Due to the high bandwidth of optical fibers, the optical infrastructure does not have to be changed with each generation, only the end-points must be upgraded. Such an upgrade should be accomplished in a backwards-compatible manner, to provide an easy transition to higher communication speeds.

The MOUSE project (Multimode Optical Upgrade of Star-shaped Ethernet) aims to provide the benefits of optical fibers to the end-user, while keeping the system low-cost. The goal is the creation of a 100 Mbps passive optical communication network based on the Fast Ethernet standard [6]. See also Radovanović [1]. The network is passive in the sense that all nodes are connected to a passive optical splitter, also called a Passive Optical Star Coupler (POSC). The absence of active components in the POSC improves the reliability of the system compared to the copper based systems, which requires switches or hubs to be present and operational. Since the network is essentially a shared medium, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is used to handle collisions. All nodes connected to the POSC are in the same collision domain.

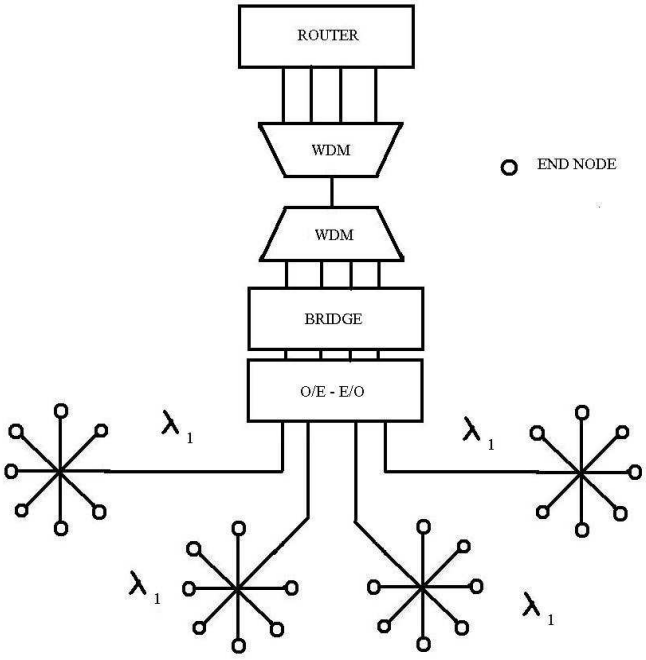


Figure 1.1: Passive optical star coupler with E/O and O/E converter

Each node (computer) on the network will have an optical transceiver which is connected with a fiber to a central POSC. Figure 1.1 shows a possible configuration for the MOUSE network. In this configuration, each POSC with the attached nodes will form a collision domain, and multiple domains can be connected together through a router. To save the costs associated with multiple electrical to optical (E/O) and optical to electrical (O/E) converters and other hardware, a Wavelength Division Multiplexer (WDM) can be used to route the data from multiple segments to a central router, see Figure 1.2. In this system, each POSC must have a different wavelength from the other POSCs. The choice for the wavelength to use can be made inside the optical transceiver.

The goal of the MOUSE project is to provide a low-cost solution for fiber-to-the-desk applications. The cost of the system can be minimized by using relatively cheap multimode fiber and Off-the-Shelf components.

One of the additional advantages of an optical LAN is that is can be used at locations where high electromagnetic fields are present, for example at an airport. Copper based networks can be influenced by the high field strengths of radar systems, but optical systems are nearly immune to these fields. Optical communication systems therefore have a distinct advantage at such locations.

To summarize the main advantages of the MOUSE system:

1. High bandwidth capability

2. Easily upgradeable

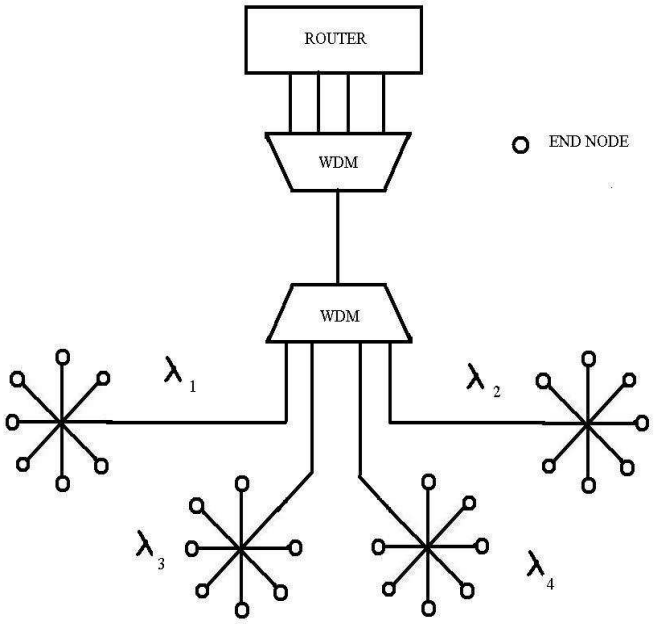


Figure 1.2: Passive optical star coupler with WDM

3. Low attenuation
4. Very good noise immunity
5. Low cost

1.2 Hardware Status

Figure 1.3 shows the proposed hardware setup for each node in the system. The setup consists of a PC with a special network card connected to a Field Programmable Gate Array (FPGA), which is used for frame processing and synchronization. The analog part of the design consists of a separate transmitter and receiver. The level converter is required to convert the 3.3 V signals at the FPGA to 5 V signals at the PC.

At the start of this assignment, there are two FPGA's available, two level converters, and two PC's with a network card. The analog hardware is not finished yet. There are problems with the communication between the PC and the FPGA, which results in glitches on the data lines, and there has been no communication between the FPGA's.

1.3 Assignment

The goal for this assignment is to integrate the different components of the MOUSE project, and to obtain a working system. The system will be build up step by step,

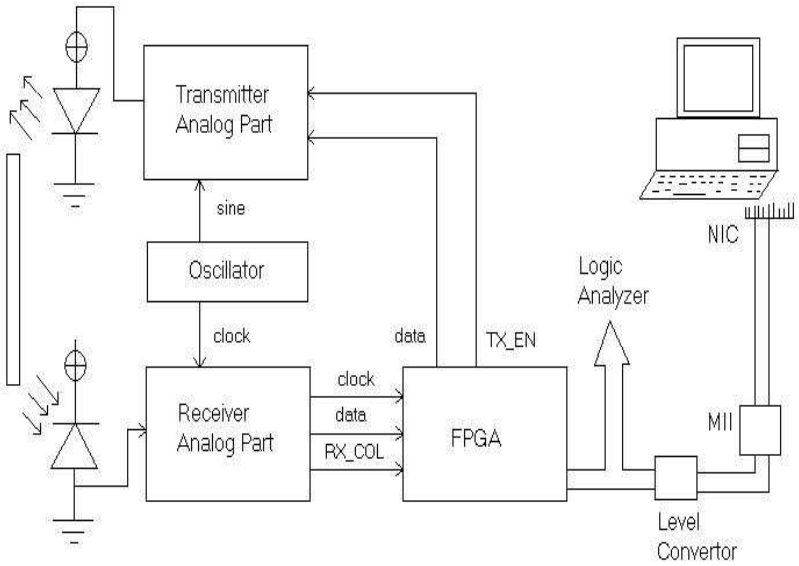


Figure 1.3: MOUSE system overview

starting at the communication between the PC and the FPGA, followed by the communication between two connected FPGA's, and finally establishing the communication between multiple FPGA's.

The system has to be tested and measured to confirm the proper operation of the network. A performance analysis is also presented.

Chapter 2

Ethernet and MOUSE Overview

2.1 Ethernet

2.1.1 Introduction

The MOUSE network is based on the Fast Ethernet protocol, which is described in the IEEE 802.3 LAN standard [6]. The OSI standard [3] defines 7 layers which can be implemented in a communication system. Figure 2.1 shows the 7 layers and the layers relevant for the MOUSE system. From the 7 layers, only the bottom two (data link layer and physical layer) are relevant to the MOUSE system. The MOUSE project implements a complete physical layer (PHY) implementation, which is connected to a MAC layer on an existing network card.

The bit times mentioned in this chapter are only valid for the operating conditions of the MOUSE system, which is 100 Mbps half-duplex CSMA/CD operation (see section 2.1.2), and can be different for other bit rates or protocols.

2.1.2 Data link layer

The data link layer provides the services necessary to transmit data between end-points, and also provides the CSMA/CD handling and error checking. It can be sub-divided into the Logical Link Control (LLC) and Medium Access Control (MAC). In the MOUSE system, only the MAC layer is important, the system is transparent to all layers above the MAC layer.

CSMA/CD Operation

The Collision Sense Multiple Access with Collision Detection (CSMA/CD) protocol allows multiple nodes to share a single medium without higher level synchronization. When a node has data to transmit, it monitors the medium, and if a transmission is detected (carrier sense), it will wait until the medium is free before it starts the

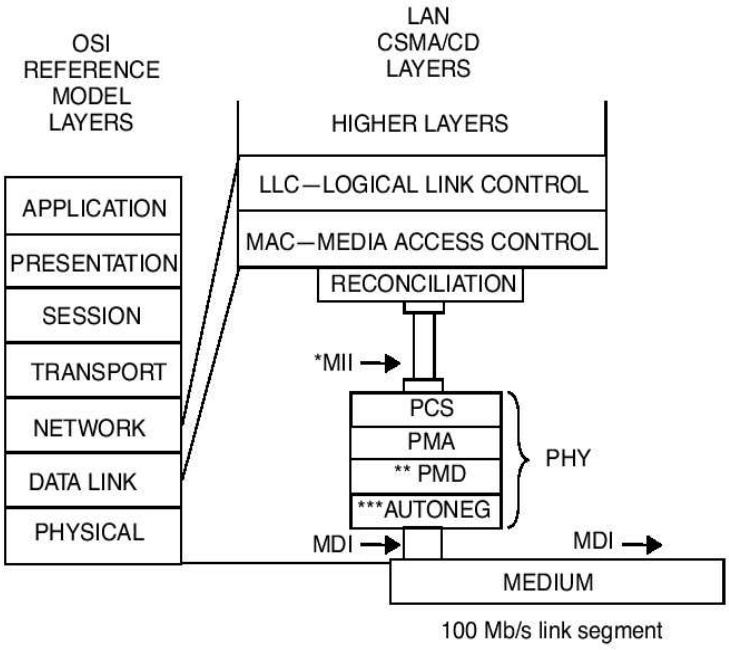


Figure 2.1: OSI 7 layer reference model

transmission. If two stations simultaneously start transmitting data, both nodes will detect this only after the frame reaches the physical layer of the other station, and the physical layer will signal a collision detect.

The MAC layer of the colliding nodes will then transmit a 32 bit jam sequence to indicate the failure to the other stations. After the jam sequence is transmitted, it will randomly back off a number of slot times (one slot time is 512 bits) before retransmission.

Half/Full Duplex

In half-duplex mode, all nodes are connected to a shared physical medium. When a node transmits data, it will be received by all other nodes present on the medium. When nodes attempt to send data simultaneously, the data will become corrupted and will be discarded by the MAC layer. The CSMA/CD protocol is used to reduce the chance of collisions on the medium, and is required for half-duplex communications.

In full duplex mode, two nodes are directly connected by a point to point link, and no further nodes can be present. Both nodes must be capable of full duplex transmission without interference. In this mode, no collisions can occur, and therefore CSMA/CD is disabled.

Full duplex mode is not supported on a shared network, and the MOUSE system can therefore only use half duplex.

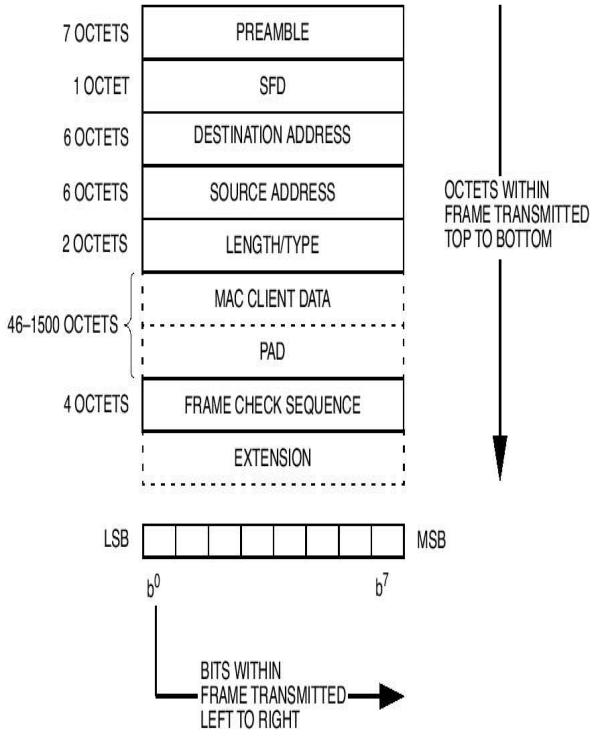


Figure 2.2: MAC frame format

2.1.3 MAC Frame Format

Data is transmitted over an Ethernet network in the form of packets or frames. Each Ethernet frames must be between 72 and 1536 bytes long, which is necessary for the correct operation of the CSMA/CD protocol. Figure 2.2 shows the format of a MAC frame. Data over the medium is sent with the least significant bit first, with the exception of the frame check sequence (FCS) (section 2.1.3).

The MAC frame consists of the following nine fields

Preamble

The preamble consists of 7 bytes of data with the pattern 10101010, which is used by a receiver to synchronize its clock on the incoming frame.

Start of Frame Delimiter

The Start of Frame Delimiter (SFD) indicates the start of a frame, and consists of the pattern 10101011.

Destination/Source Address

The destination and source MAC addresses are 48 bit (6 bytes) long and are unique for each physical layer device.

Length/Type Field

The length/type field has two possible functions:

1. If the field value is less than or equal to 1500, the field indicates the number of bytes which are present in the payload. This length can be between 0 and 1500.

2. If the field value is larger than 1536 then the field indicates which protocol is used for the payload.

If the size of the payload is less than 46 bytes, then the MAC layer will add a number of bytes to the payload (padding) so that the total size of the payload plus the padding is 46. The contents of the padding bytes is not defined, but is mostly taken as zero. The padding is used to meet the minimum frame size requirements of 72 bytes, which is required for the correct operation of the CSMA/CD protocol.

As an example, the type field can have the following protocols, which will be encountered in the MOUSE system:

Ptotocol type	Type field value
TCP/IP packets	0x0800
ARP for IP	0x0806

Frame Check Sequence

The Frame Check Sequence (FCS) consists of a 32 bit CRC value, calculated over the destination and source address, the length/type field, the data and the padding, but not over the preamble, SFD and FCS. The FCS is transmitted with the most significant bit first

The CRC generator polynomial is:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + ^x + 1$$

2.1.4 Medium Independent Interface

The interface between the MAC layer and the physical layer is called the Medium Independent Interface (MII). Through this interface, the physical layer can be implemented independently from the MAC layer. From the point of view of the MAC layer, it does not matter which type of physical layer is present (optical or electrical, 10

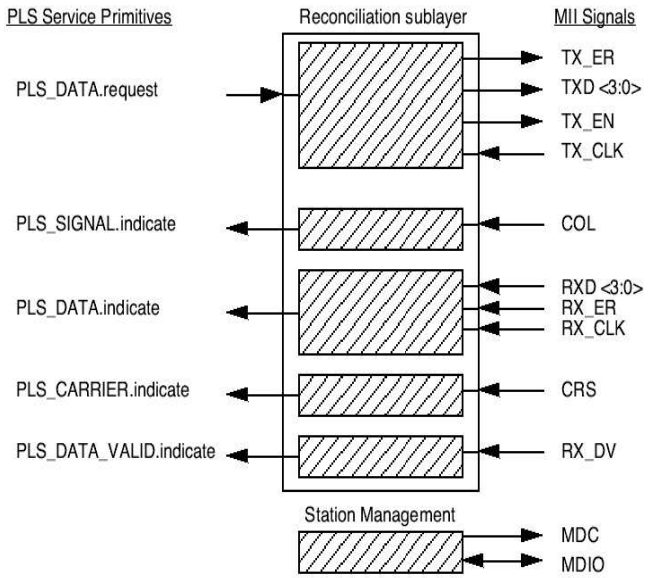


Figure 2.3: MII interface signals

Mbps, 100 Mbps or 1 Gbps, full or half duplex), as long as the PHY provides the correct information through the configuration interface (called the MII management interface).

The MII Interface consists of the following components, which are shown in Figure 2.3

Status Signals

`Collision Detect`	`COL`
`Carrier Sense`	`CRS`

The two status signals Collision Detect (COL) and Carrier Sense (CRS) are generated by the PHY, in response to the actual state of the transmission medium. These signals are required for the proper operation of the CSMA/CD protocol.

Carrier sense is activated (high) when the physical layer senses an incoming packet, or when it is busy transmitting a frame.

Collision detect is activated when the physical layer senses an incoming packet while a transmission is already in progress, and wil remain asserted while the collision condition continues. The collision will only be detected by the nodes which are transmitting data, all other modules will only detect an invalid frame, as the frame terminates prematurely and with an invalid FCS.

Transmit & Receive Path

`Transmit Clock`	`tx_clk`

```
Transmit Data       txd<3:0>
Transmit Enable     tx_en
Transmit Error      tx_er

Receive Clock       rx_clk
Receive Data        rxd<3:0>
Receive Data valid  rx_dv
Receive Error       rx_er
```

Data is transmitted over the MII Interface one nibble (4 bits) at a time, with the low order nibble first. For a 100 Mbps communication channel, the MII interface will operate at 25 MHz. Both the transmission clock (tx_clk) and receiving clock (rx_clk) must be generated at the physical layer. The clocks must be present at all times while the PHY is active.

The transmission path is independent from the receiving path, which allows the MII interface to handle both half-duplex and full-duplex communications.

When the MAC layer has data to send, it will wait for the medium to be free (CRS low), and then make tx_en high, to indicate the start of the transmission. While tx_en is high, data is valid at the rising edge of each clock.

During reception of a frame, the physical layer enables rx_dv, and ransmits data on the rising edge of the rx_clk. The data must at least include the start of frame delimiter, but may include one or more preamble nibbles.

Configuration Interface

```
Management data Input/Output   MDIO
Management data Clock          MDC
```

The MDIO and MDC signals are required for the MII manamgement interface described in section 2.1.6.

2.1.5 Physical Layer

The MOUSE project implements a complete PHY. According to the OSI standard layer model, the physical layer can be divided into the Physical Coding Sub-layer (PCS), the Physical Media Attachment (PMA) sub-layer and the Physical Media Dependent (PMD) sub-layer. For this assignment, it is not useful to make the distinction between the layers, and therefore the PHY will be treated as one entity.

4B/5B encoding

The data to be transmitted over the physical layer is encoded with a 4B/5B encoder. During transmission, each nibble in the frame is mapped onto a 5-bit codeword. The line rate of the system is therefore 125 Mbps for a 100 Mbps data rate. When receiving data, the 5-bit code words are again decoded into 4 bit nibbles.

The 4B/5B encoding is useful since the PHY is able to transmit status and control information with the data stream, which will not be mistaken for actual data. A number of codes have been defined, two codes for the Start-of-Stream Delimiter (SSD) and two codes for te End-of-Stream Delimiter (ESD). The other codes are either mapped to data nibbles, or are interpreted as an error. A second advantage of 4B/5B encoding is that the used codes can be chosen such that there are always transitions present in the 4B/5B encoded data-stream, which facilitates decoding and synchronization.

NRZI encoding

The Non-Return-to-Zero with Inversion (NRZI) encoding scheme is used to facilitate connections with balanced transmission lines or twisted-pair cables. A zero is transmitted as no transition, while a one is transmitted as a transition of the signal. With the NRZI encoding method, the polarity of a balanced transmission line or a twisted pair line is not important, as only changes in the signal levels indicate a one, while no changes indicates a zero. The MOUSE project utilizes Low-Voltage Differential Signaling (LVDS) to interconnect the different system components. LVDS makes use of a balanced transmission line to reduce electromagnetic interference for high speed data communications.

Frame alignment

When a frame is received by the PHY, it must first synchronize to the incominng preamble. During this synchronization, one or more of the transmitted preamble bits can get lost. It is important for the PHY to reliably detect the beginning of a frame, and the MOUSE system utilizes the SSD to synchronize to the incoming data. The SSD is compared bit-by-bit with the incoming serial data stream, and when a match is found, this indicates the start of a frame. See also [1].

CRS and COL signaling

The PHY must assert the CRS and COL signals depending on the state of the transmission and the medium. When the PHY is either transmitting or receiving data, the CRS signal must be asserted, and when the PHY is both transmitting and receiving, the COL signal must also be asserted.

	Management frame fields							
	PRE	ST	OP	PHYAD	REGAD	TA	DATA	IDLE
READ	1...1	01	10	AAAAA	RRRRR	Z0	DDDDDDDDDDDDDDDD	Z
WRITE	1...1	01	01	AAAAA	RRRRR	10	DDDDDDDDDDDDDDDD	Z

Figure 2.4: MII management frame format

2.1.6 MII Management Interface

The MII management interface is based on a simple 2 wire protocol, and is used to transmit status and control signals between the MAC layer and the PHY. The protocol is based on the master/slave principle, with the station management entity (STA) located at the MAC layer acting as the master and the physical layer device (PHY) acting as the slave. All transfers are initiated by the STA, which will also provide the clock signal (MDC) for the interface. The data is transmitted over the bidirectional MDIO line. A slave is only allowed to access the MDIO line when the STA requests data from the slave. The MDIO line requires a pull-up resistor to provide the line with a high level. When a slave has to transmit data, the MDIO line can be pulled down by the slave when a zero must be transmitted. The pull-up resistor will provide the high level for one bits. This mechanism prevents short circuits between simultaneously transmitting slaves.

The standard specifies a maximum operating speed of 2.5 MHz for the MII mamangement interface. The minimum clock high and low times are 160 ns, and the minimum clock period is 400 ns. This approach is taken to allow the interface to be implemented in software as well as in hardware. There is no minimum speed defined in the standard. Figure 2.4 shows the MII Management frame structure.

Preamble

A frame starts with 32 preamble (PRE) bits, in order for the PHY to synchronize onto the clock. The MDIO line is high during this period. The preamble can be omitted when a PHY indicates through its status register that it can receive frames without preamble.

Start sequence

The start-of-frame (ST) indicates the start of a frame. During the preamble all data bits are high, and the ST is a low data bit followed by a high data bit.

Operation

The operation (OP) field indicates whether the transmission indicates a read operation (data transfer from FPGA to PC) or a write operation. The write operation is indicated by a low data bit followed by a high data bit, and the read operation by a high data bit followed by a low data bit.

Physical layer address

The physical layer address (PHYAD) must be unique for each physical layer device on the MII bus. There are 32 addresses, and therefore there can be a maximum of 32 devices attached to the bus.

Register address

The register address (REGAD) indicates which of the 32 internal registers will be read or written. Address 0 through 15 are defined in the standard, and address 16 through 31 are application specific, and can be used for our own purposes. Table 2.1 shows the register layout for the MII management interface.

Turnaround time

The turnaround bits (TA) are used to allow a device time to set up the data transmission or reception. When data is transmitted to the FPGA, the MAC layer will make the first bit high and the second bit low. When data is read from the FPGA, the FPGA must leave the first bit high, and the second bit must be pulled low. When the line is not pulled low, it indicates to the STA that there is no device present on that physical address.

Data

The 16 data bits are transmitted following the turnaround time. During a write action, the FPGA must receive the data and act on it after the transmission. During a read transmission, the requested data is send back to the MAC Layer.

Registers

The IEEE 802.3 standards define a number of registers that are part of the MII management interface. The registers are shown in Table 2.1. Each physical layer device must implement at least register 0 (control) and 1 (status). All other registers are optional, except for the extended status register (15) which must be present for 1000 Mbps operation.

Table 2.1: MII management registers

register number	register name	function
0	control	set up PHY operating mode
1	status	indicate capabilities and status
2,3	indentification	unique indentifier for each PHY
4	auto-negotiate advertisement	
5	auto-negotiate link partner base page ability	
6	auto-negotiate expansion	
7	auto-negotiate next page transmit	
8	auto-negotiate link partner received next page	
9	master-slave control	only used in 100Base-T2
10	master-slave status	only used in 100Base-T2
11 - 14	reserved	
15	extended status	used in 1000 Mbps communication
16-31	vendor specific	

2.2 Altera FPGA

The Altera Field Programmable Gate Array (FPGA) contains all control, encoding and decoding logic for the processing of frames, and for interfacing the FPGA to the MAC layer. This section will introduce the FPGA and describe the main characteristics. There are two slightly different FPGA's available, the Altera EP20K400-1x and the EP20K1500-1.

2.2.1 Logic Element

The Logic Element (LE) is the basic building block inside the FPGA. The structure of a LE is shown in Figure 2.5. The LE contains a 4-input Look-Up-Table (LUT), carry and cascade logic, set/clear logic, a programmable register and multiple I/O options.

From a simplified design viewpoint, the LE consists of a 4 input LUT followed by a flipflop. The LUT can be used to implement any function of 4 variables, and the intermediate values are then stored in the flipflop. When more inputs are required, multiple LUT's can be cascaded while bypassing the flip-flop. Due to timing constraints, only a limited number of LUT's can be cascaded before the (intermediate) result must be buffered inside a flip-flop.

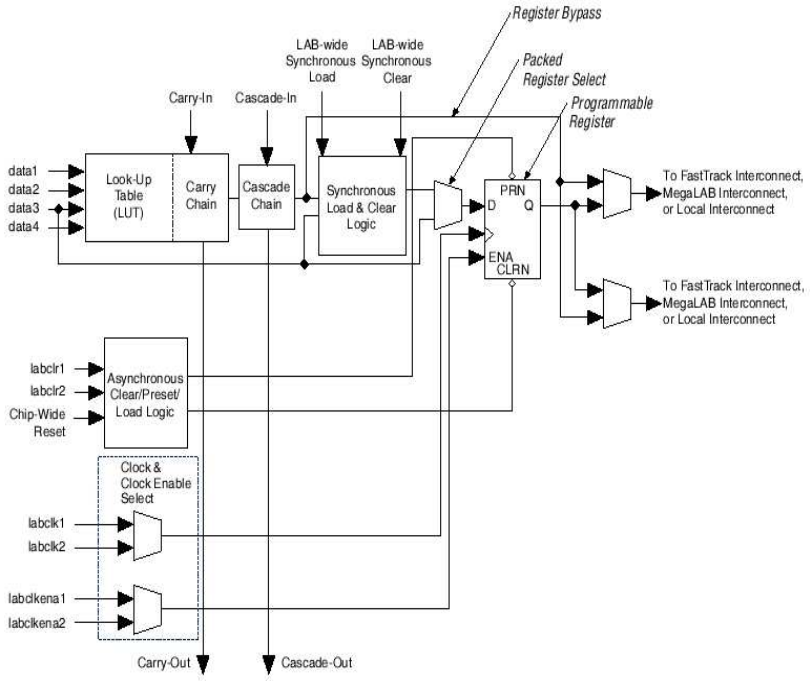


Figure 2.5: FPGA logic element

2.2.2 Logic Array Block

The logic Array Block (LAB) consists of 10 LE's and a local interconnect. The local interconnect is used to connect LE's together in the same LAB but also to the neighboring LAB's. Figure 2.6 shows the interweaving between the LE's and the local interconnects.

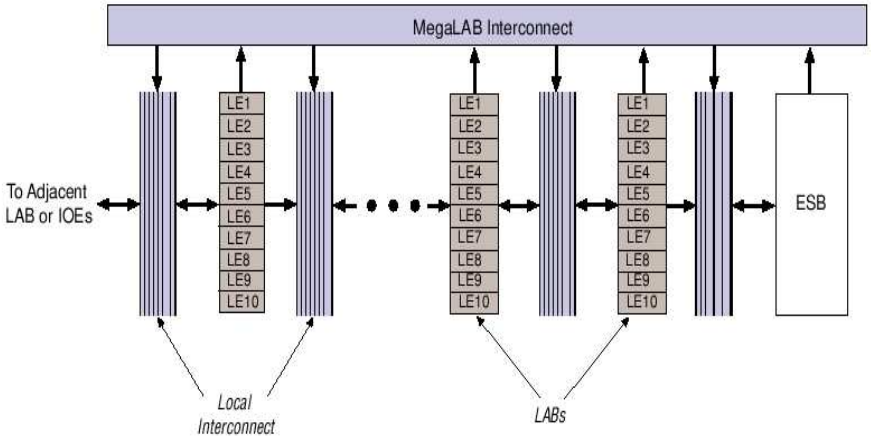


Figure 2.6: FPGA logic array block

Table 2.2: Logic element delay (in ns)

name	parameter	EP20K400EBC-1x	EP20K1500EBC-1
t_{SU}	minimum setup	0.23	0.25
t_H	minimum hold	0.23	0.25
t_{CO}	clock to output	0.25	0.28
t_{LUT}	LUT delay	0.70	0.80

2.2.3 MegaLAB

The MegaLAB structure is the largest basic structure present in the FPGA. It consists of 16 LAB's for each MegaLAB inside the EP20K400EBC-1 and 24 LAB's for each MegaLAB inside the EP20K1500EBC-1S. The MegaLAB has a MegaLAB interconnect which is attached to each LAB inside, as shown in Figure 2.6. The MegaLAB also contains an embedded system block (ESB) which can provide 2048 bits of memory in different configurations. MegaLAB structures are interconnected using the Fasttrack interconnect which, despite its name, is the slowest interconnect available inside the FPGA.

2.2.4 Timing

Logic element delay

The timing requirements for the available FPGA's are slightly different. Table 2.2 shows the timing requirements for a logic element in each of the FPGA's. t_{SU} is the minimum setup time (in ns) of the internal register in each LE; the input of the register should not change during this period before the clock flank. t_H is the minimum time (in ns) that the input should remain stable after the clock pulse. The output value of the register is available t_{CO} ns after the clock. t_{LUT} is the delay introduced by a LUT.

Interconnect delay

The time delay for the local interconnects is shown in Table 2.3. The Quartus software will attempt to optimize the timing parameters for the design by keeping related signals close together. This is not always possible, due to the physical layout of the in and output signals. The in and output signals are distributed around the edge of the FPGA, and therefore some signals need to travel over the Fasttrack interconnect to reach their destination.

Table 2.3: Interconnect delay (in ns)

Interconnect type	EP20K400EBC-1	EP20K1500EBC-1
local interconnect	0.25	0.28
MegaLAB interconnect	1.01	1.36
Fasttrack interconnect	3.71	4.43

2.3 Node Implementation

2.3.1 Overview

The three most important parts of the MOUSE system are shown in Figure 2.7. The Network Interface Card (NIC) provides the MAC layer for the system, the FPGA provides the digital processing for the system, and the analog board implements the required optical transmitter and receiver functionality (analog transceiver). The FPGA is connected to the MAC layer through the MII interface with a ribbon cable, and to the analog transceiver through a number of Low Voltage Differential Signalling (LVDS) signals. The FPGA must provide all PHY layer functionality from Section 2.1.

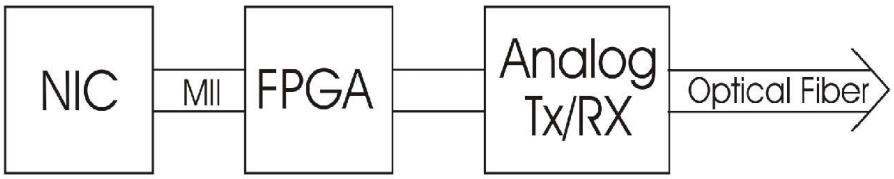


Figure 2.7: Block schematic overview of MOUSE system

2.3.2 Transceiver Implementation in the FPGA

The VHDL implementation for the frame processing has been developed by Robert O. Taniman [10], and is done according to the block schematic overview in Figure 2.8. The FPGA logic operates at 125 MHz, which is the same as the line rate of the MOUSE system.

In this section, the receiver implementation will be discussed. The transmitter performs the complimentary steps as discussed below, except for the alignment, which is not required for the transmitter.

Data that enters the FPGA is first NRZI decoded, after which it must be aligned to the 4 bit nibbles which will ultimately be sent over the MII interface. To facilitate the alignment, a Start-of-Stream Delimiter (SSD) is added to the start of a frame by the transmitting node, and the SSD will be detected by the receiver. By comparing the SSD bit for bit with the incoming serial data stream, the alignment can be made.

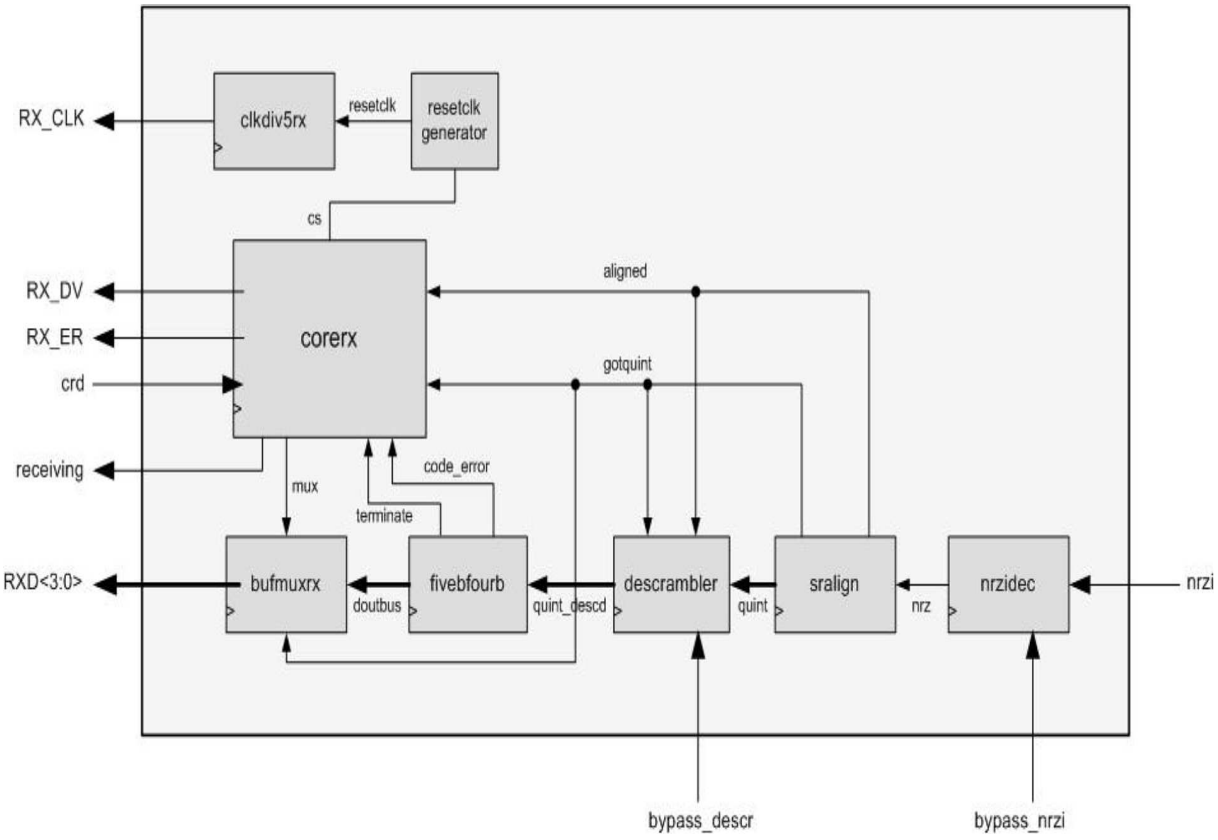


Figure 2.8: Receiver implementation

Next, the data will be descrambled, and 5B/4B decoded. This will remove all information, which is added to the frame by the transmitter. Finally the data is transmitted over the MII interface to the MAC layer, which will pass the frame to the higher layers.

2.3.3 Analog Transceiver

The analog transceiver was developed during a number of years by several students, and the latest design has been developed by Christian Dolea [11]. The analog transceiver will provide the FPGA with the clock, Carrier Sense (CRS) signal and the received data, and will transmit the data from the FPGA to the network.

Synchronization

An important function of the analog transceiver is to provide the FPGA with synchronized data. To facilitate the synchronization, the transmitter will transmit both the data and the clock over the optical medium. The receiving node can then synchronize its own clock to the clock received with the incoming data. Figure 2.9 shows the spectrum of a transmitted frame. The spectrum of the data will have a $\sin(x)/x$ shape

with the first zero at 125 MHz. The clock is superimposed to the spectrum at this frequency.

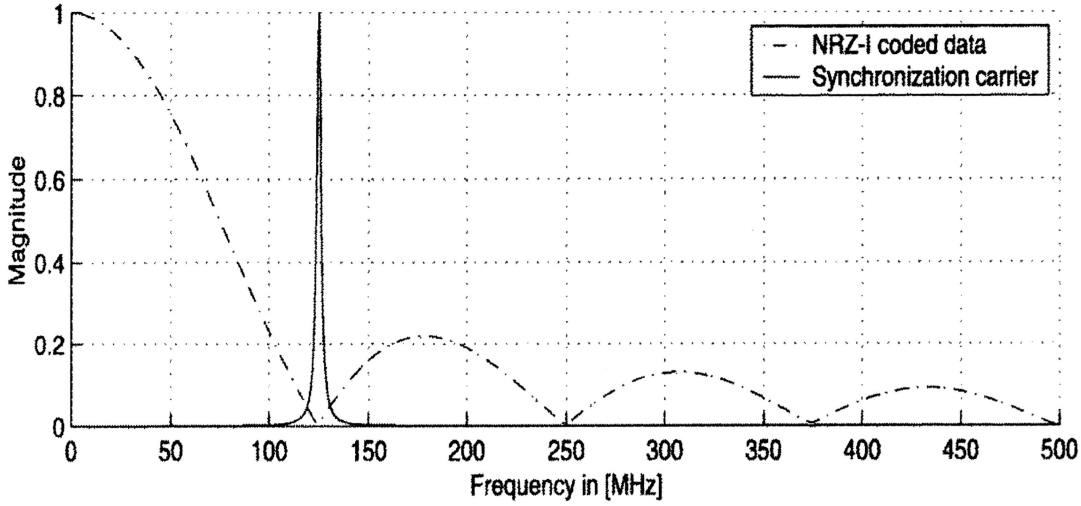


Figure 2.9: Spectrum of a transmitted frame

Chapter 3

Communication Between PC and FPGA

3.1 Introduction

As mentioned before, the problems will be investigated in order, starting at the MAC layer. The first problem to solve is the communication between the PC and the FPGA. According to Klein Kiskamp [2] there are a number of glitches present on the signals transferred over the MII interface, which interfere with the proper operation of the system. The first task is to find out the cause of the problem, and design a new level converter to solve it.

3.2 System Overview

Figure 3.1 gives an overview of the hardware available for the digital part of the mouse system, before this master assignment was started. The hardware described here is either bought from a manufacturer or developed especially for this project.

The system consists of the following parts:

1. A PC with the PCILAN-1 network card:
   This card is used for the external MII interface. The physical layer (consisting of the ICS1890 chip) can be switched off in software. The card implements the 10/100Base-TX, 10Base-T and 10Base-2 [6] specifications. The PCILAN card is actually a CompactPCI card, which is the industrial variant of the standard PCI interface, and is mainly used in industrial computer systems for data acquisition and control software. The CompactPCI standard defines a different connector for the interface, and therefore a CompactPCI to PCI converter is required to use the PCILAN network card. Figure 3.2 shows a picture of the PCILAN network card.

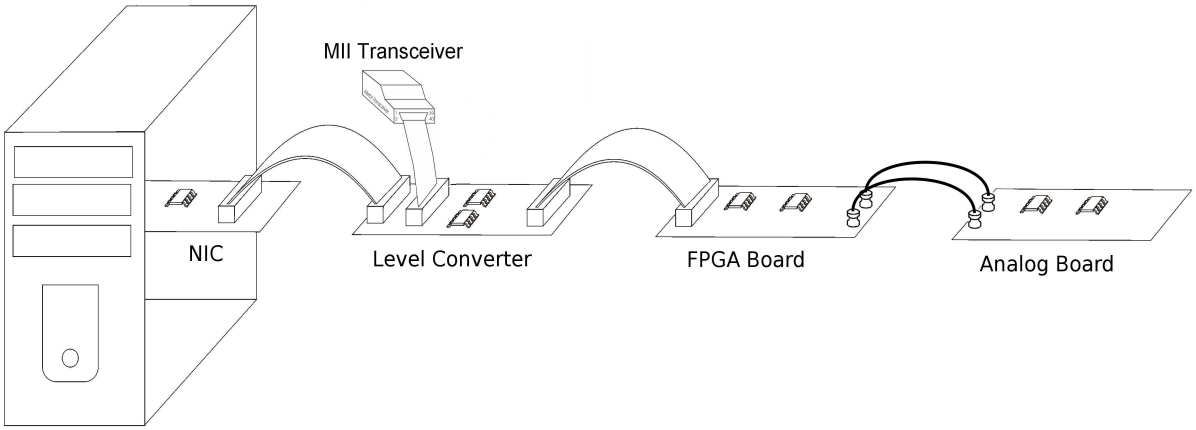


Figure 3.1: Mouse hardware overview

2. The Edimax MII Transceiver:
   This transceiver contains a complete physical layer implementation based on layer 1 of the OSI 7-layer model [3]. The transceiver implements the 10/100Base-TX Fast Ethernet specifications [6], and is used in the MOUSE system to emulate all MII Management functions. A picture of the transceiver is shown in Figure 3.3

3. The level converter:
   The level converter print provides a basic voltage translation between the PCILAN card (5V) and the FPGA (3V3). It further facilitates the connection of the MII Transceiver and a logic analyzer. Data is transmitted at a frequency of 25 MHz over the MII bus, four bits at a time. Each bit has a duration of 40 ns. Figure 3.4 shows a picture of the level converter. The level converter has been developed by Ronny Klein Kiskamp.

4. The FPGA:
   The FPGA is the heart of the new optical physical layer. It provides all necessary encoding and decoding logic required for transmitting and receiving data at 100Mbps. There are 2 types of FPGA used for the MOUSE system, the EPC20K400EBC-1 and the EPC20K1500EBC-1. Figure 3.5 shows one of the FPGA's in the system. The initial version has been developed by Jan Rutger Schrader, and a modified version has been developed by Ronny Klein Kiskamp.

5. The Analog PCB:
   The analog PCB will implement the optical transmitter and receiver. The clock extraction and synchronization on the incoming data will be performed on this board. This board has been developed by Christian Dolea, following multiple prototypes developed by different students before him. Figure 3.6 shows a picture of the latest version.

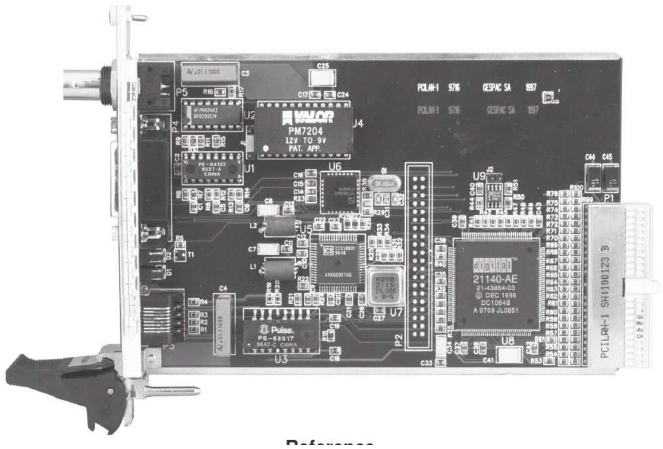


Figure 3.2: The PCILAN network card

Figure 3.3: The Edimax MII transceiver

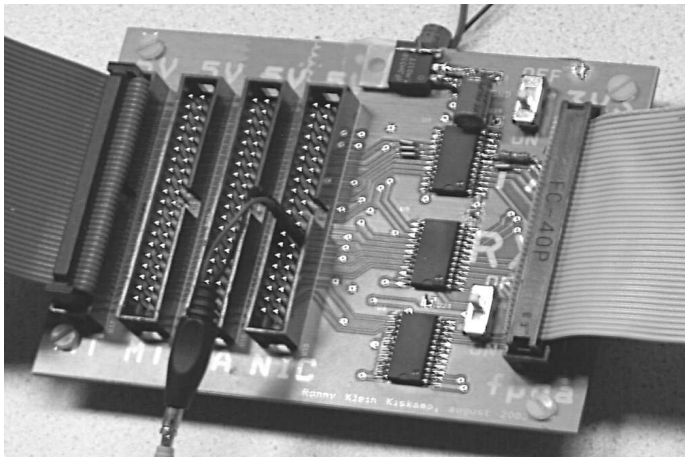


Figure 3.4: Level converter

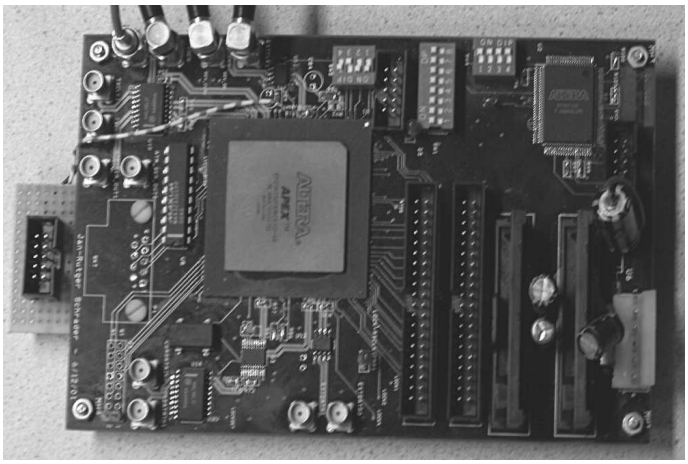


Figure 3.5: The FPGA

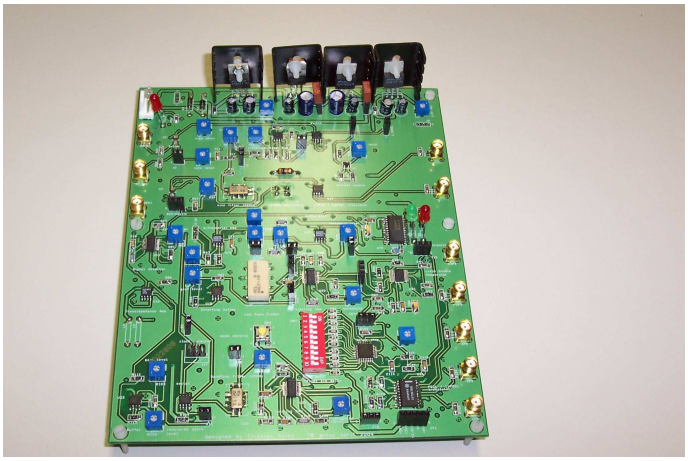


Figure 3.6: The analog transceiver

This system is expected to operate as follows: First, the PC is booted normally with the PCILAN card operating in the default mode. Then, the physical layer on the card must be switched off, and the level converter and the FPGA can be enabled. Next, the MII transceiver must be connected, to provide the MOUSE system with a MII Management interface.

All components in the above system are connected by either a standard 40 wire ribbon cable (ATA33 cable) or a modified 40 wire ribbon cable in the case of the MII transceiver.

3.3 Problem Identification

3.3.1 Cable Termination

One of the possible reasons for the glitches in the communication is that the ribbon cables (which are in fact transmission lines) are not properly terminated. The resulting reflections might cause unwanted behavior. To test the influence of the termination impedances on the system, a number of simulations and measurements have been made. A more complete treatment of the influence of the reflections is given in Appendix A.

In standard transmission line (TL) systems, the TL must be terminated with an impedance equal to the characteristic impedance Z_0, in order to eliminate reflections. The source and load impedances will form a voltage divider $Z_L/(Z_S + Z_L)$ which will reduce the voltage level at the input of the level converter. Evidently, this is not the optimal situation in digital communications. To prevent the voltage division, a small capacitor is added which will only terminate the TL with the characteristic impedance for the high-frequency reflections.

Simulations

The simulations are conducted using a simplified model for the system. The output of each chip is modeled as a source with a small output impedance of 30 ohm and a parasitic capacitance of 15 pF. The cable length shall be taken as 20 cm, or approximately 1 ns delay, and the cable has an impedance of 100 ohm. The termination is modeled by a large resistor (high impedance input) and 15 pF input capacitance (which also includes the impedance of the measurement probe). The total setup is shown in Figure 3.7. The system is excited with a periodic pulse train with a rise time of 400 ps and a fall time of 1 ns. Next to the circuit, the results are shows of the reflections in this setup. The reflections are clearly present, and could interfere with the signal reception.

The effect of adding a dampening resistor is shown in Figure 3.8. The result is the elimination of the reflections.

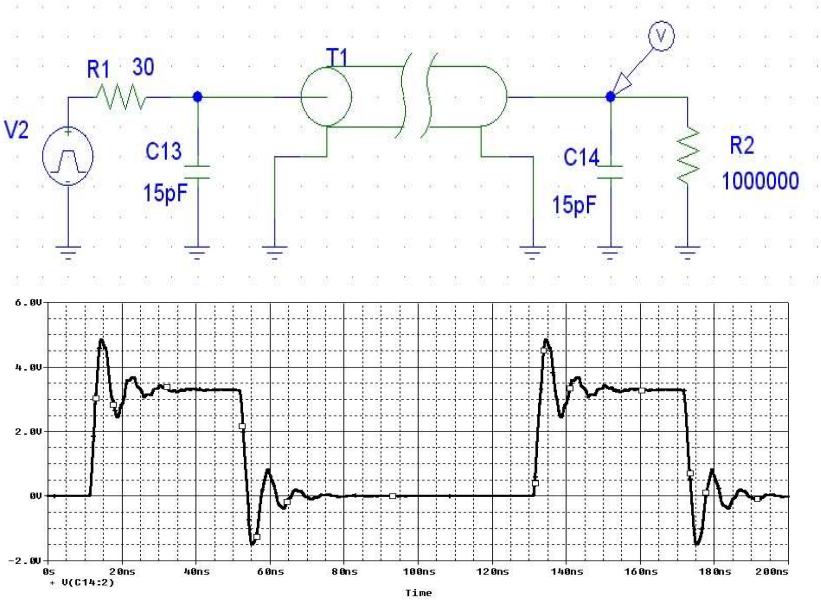


Figure 3.7: Transmission line without termination

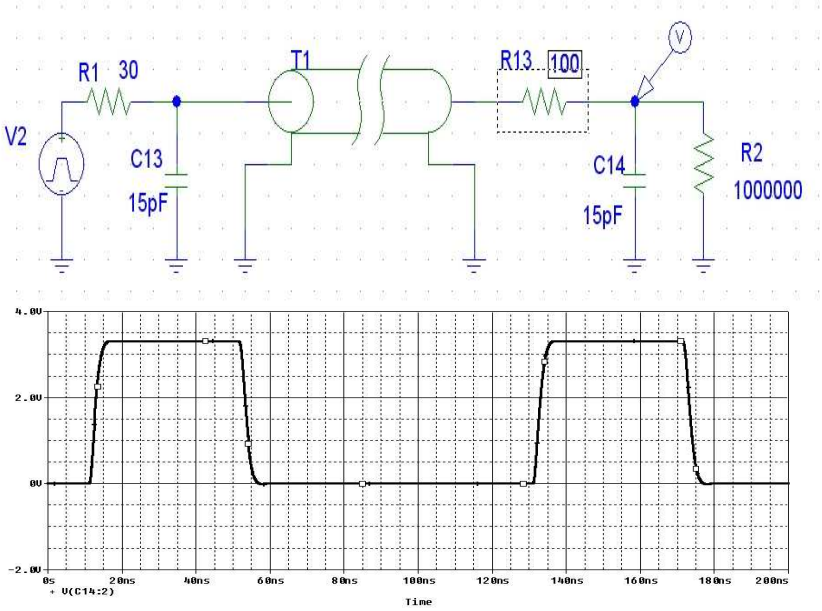


Figure 3.8: Transmission line with termination

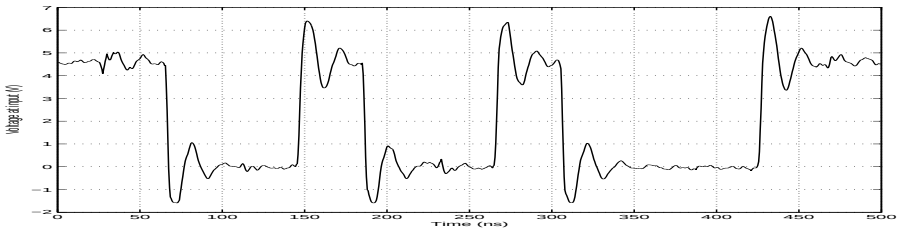


Figure 3.9: Overshoot at the input of the level converter

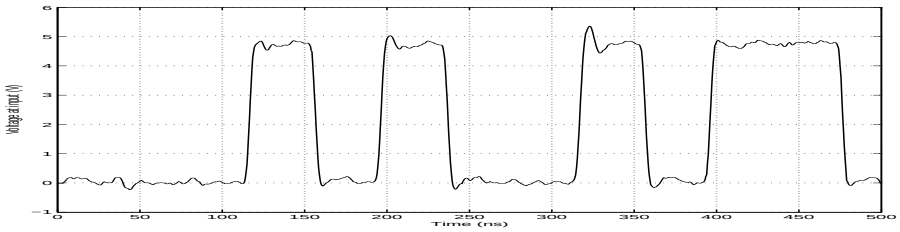


Figure 3.10: Signal terminated with 100 ohm

Measurements

To confirm the results of the simulations, a number of measurements have been made. Two PC's are communicating with each other through the Ethernet port of the PCILAN card at 100 Mbps. The level converter is connected to the MII bus, with the outputs disabled, and is only listening to the communication that is taking place between the two PC's.

The probes used to measure the signals have an input capacitance of 10 pF and the expected input capacitance of the level converter is of the same order. The probes should still suffice for an indicative measurement as is done here, but may slightly alter the resulting waveform.

Figure 3.9 shows a number of bits transmitted from the PCILAN card to the level converter. There are clearly distortions present on the waveform. To prove whether the distortions are reflections caused by improper terminations, the level converter board has been modified to include a dampening resistor of 100 ohm. The results are shown in Figure 3.10. As can be seen, the signals have improved considerably by the addition of a dampening resistor.

A number of eye-pattern measurements have been made for the system. Figure 3.11 shows the unterminated signal at the input of the FPGA. A large reflection is present near the position where the signal ideally would be sampled. The addition of a termination resistor improves the signal, as can be seen from Figure 3.12.

Figure 3.11: Eye pattern of the unterminated signal

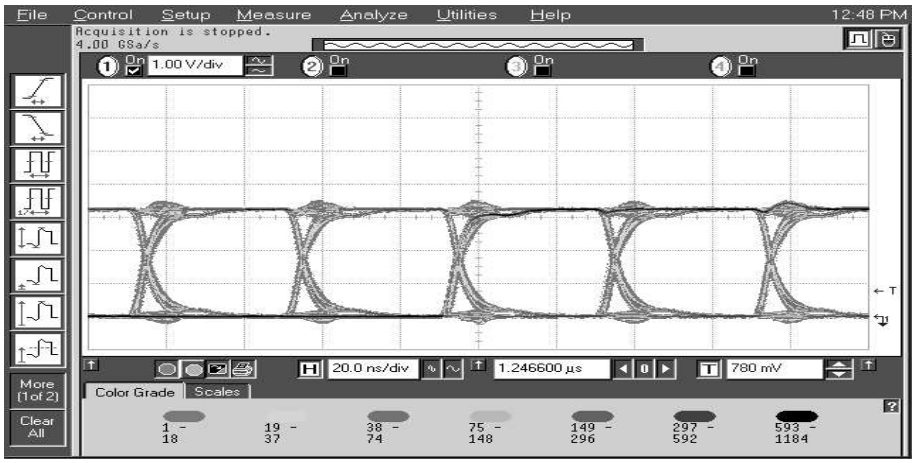


Figure 3.12: Eye pattern of the terminated signal

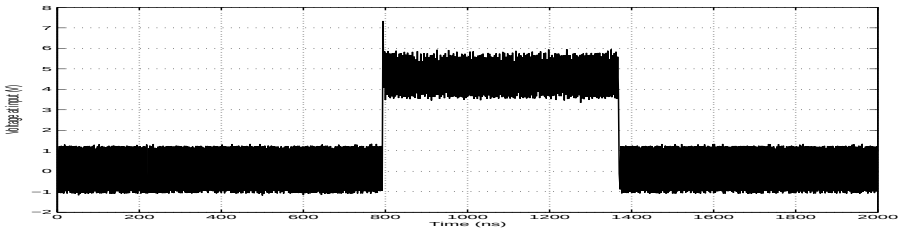


Figure 3.13: Crosstalk on the RX_DV signal

Conclusion

The reflections are easily removed by terminating the ribbon cables. The signal quality has improved considerably. The new level converter must therefore include proper terminating resistors. A value between 100 ohm and 120 ohm seems to work best.

3.3.2 Crosstalk

The MII interface operates at 25 MHz, and includes two independent clock signals at that frequency. The high switching speeds of the clocks and the close proximity to the other signals can generate crosstalk to the other signals.

Figure 3.13 shows a measurement of the RX_DV signal located close to the clock signal. The RX_DV signal indicates that an active transfer is taking place, and the signal only changes at the start and at the end of a frame. The noise here reaches 1 V, indicating a very strong coupling between the two signals. The minimum spacing between the two signal traces on the print is 1.2 mm, and the height of the traces above the ground plane is 1.6 mm. Therefore a large portion of the electromagnetic field generated by the clock line passes between the RX_DV trace and the ground. As an approximation, the crosstalk is proportional to the area of the loop between the signal trace and the ground, and inversely proportional to the spacing between both traces. See also Paul [4], Chapter 8.

The noise on signals at a further distance from the clock lines is less than that present on the RX_DV line.

The ground plane is also discontinued at a number of locations along the RX_DV trace, which causes the return current to flow around another path, and this increases the loop area, which results in more crosstalk. Special attention should be paid to the layout of the board to reduce crosstalk between neighboring signals. This includes moving the clock signal traces away from the other signals.

3.3.3 Timing

The MII transmit clock (tx_clk) is generated by the FPGA. The PCILAN card transmits data on the rising edge of this clock. From the viewpoint of the FPGA, data

arrives delayed with respect to the transmitted clock, and this delay is twice the propagation delay of the system. If this delay is not taken into account inside the FPGA, data might be read at the wrong time, especially if this delay is approximately half a clock pulse, which is the time that the data changes.

The MII bus frequency is 25 MHz, so the clock period is 40 ns. Signals over the ribbon cable travel at an estimated speed of 20 cm/ns, and the propagation delay of the 74lvx3245 IC on the level converter is approximately 5 ns. The one way delay can easily reach 8 ns, and the roundtrip delay would then be 16 ns. The ringing in the data bits due to incorrect termination can increase the possibility of read errors even more.

The problem can be solved inside the FPGA, by using a transmitted clock, and a delayed read clock for that data.

3.3.4 Short Circuit Protection

As a side effect of the termination resistors, they will limit the current that could flow when the level converter and PCILAN card try to access the MII bus simultaneously. This has occurred a number of times in the past, and is caused by the way that the driver for the PCILAN card works. This driver problem will be further investigated in Chapter 6.

In the worst case situation there will be 5 V across the resistor of approximately 100 ohms. The current will be limited to a maximum of 50 mA. The dissipated power (0.25 W) is actually too high for the resistors used (0.10 W), but it would at least prevent the PCILAN card from being destroyed. The level converter board can be repaired more easily.

The worst case situation is not likely to occur, since most signals default to a low level and will only become high when data is transmitted, and in that case the duty cycle would be 50% on average.

3.3.5 Power Supply

The level converter lacks a proper power regulator, which could result in external noise injected in the system due to insufficient load regulation or noise pickup on the supply cables. It can result in voltage drops due to the fast switching requirements (25 MHz) for the MII interface. Furthermore, the level converter requires two operating voltages, 5 V and 3.3 V, and these must be supplied separately by an external power supply.

For these reasons, a proper power supply should be added to the new level converter, which can provide both operating voltages.

3.3.6 MII Management

The setup of the system requires a number of manual actions to take place in a specific order, including hot plugging a ribbon cable to a running system, each time the system is started or modified. If a mistake is made, it can cause a short circuit between the PCILAN card and the level converter.

An improvement would be to incorporate the MII Management interface into the FPGA. This way, the FPGA would be in control of transmitting and receiving data through the MII interface. Also, the interface can be used to configure internal settings in the FPGA, or to read statistical data. The proposed interface would eliminate the need to plug cables in and out of a running system, reducing the risk of errors one might (and will, according to Murphy) make.

To incorporate the interface, a number of additional signals have to be added to the level converter. One of these is the output enable for the level converter, which can be used to isolate the level converter from the MII bus, by using software on the PC.

3.3.7 Inserting the Level Converter into the PC

In the original system, the PCILAN-1 card is not connected to the PC casing, since it is actually a CompactPCI card, and the only mechanical stability is provided by the PCI slot in which it is inserted. It is necessary to connect it to the PC case with the use of wires or tape to keep it in place.

The new level converter can be designed to fit inside the PC, directly on top of the PCILAN-1 card, and can be connected to the PC case directly, which provides mechanical stability and eliminates one ribbon cable in the system. The power supply of the PC can be used to power the card, eliminating the need for a seperate power supply.

As a side effect, the PC can be placed vertically on the table, instead of horizontally as is required at present. This reduces the required room for the setup, and also makes the system more easily accessible.

3.3.8 Signaling Leds

The new level converter should include an easy way to determine the status of the level converter, it should for example include a number of signaling leds to indicate the status of the system. Two leds to indicate power supply levels, one led to indicate that the MII interface is enabled, and of course two leds to indicate transmission and reception of data.

3.4 Conclusion

There are a number of problems with the current level converter design. It is possible to modify the level converter to provide a number of improvements, such as a power supply and the termination resistors. This does not solve some of the other problems, such as the crosstalk. In order to solve the rest of the problems, a new level converter is required.

It is therefore recommended to build a new level converter to include the improvements and solve the problems that were encountered.

Chapter 4

New Level Converter

4.1 Design considerations

With the results obtained in the previous chapter, a new level converter has been designed and built. One of the goals was to end-up with a device that is as flexible as possible, to provide multiple options in the unfortunate case that an idea does not work out as intended.

4.1.1 Features

The following features are implemented in the new level converter design:

1. A proper power regulator.
2. The MII management interface.
3. Terminating resistors for the ribbon cables.
4. Signaling leds.
5. Optional PC mountable system.
6. Optional MII transceiver attachment.
7. Switches to manually disconnect from the MII bus.

MII Management Interface

The MII management interface requires three additional signals; output enable, MDC and MDIO. To provide the bidirectional MDIO signal with only one wire will require additional components in the system (the MDIO signal also has to be converted between 3V3 and 5V). The easiest way to implement this is to split the signal into an input and an output signal. The input can be directly connected to an input on the FPGA.

Since the MDIO bus uses a pull-up resistor, the FPGA only needs to pull down the MDIO line when it needs to transmit a zero. This can be done by connecting the MDIO output to the output enable of a buffer on the level converter. When the FPGA sends a zero, the buffer pulls the MDIO line low; otherwise the pull-up resistor will pull the MDIO line high.

The FPGA that is used is not designed for the mouse project, and the extra signals we need cannot be connected directly to the FPGA with the cable that is already present (primary cable). There are three options to solve this problem. First, an extra (secondary) ribbon cable can be used between the FPGA and the level converter. Second, the FPGA board can be modified to add the signals to the primary cable. The last option is to build an add-on board which can combine the signals onto one cable.

The new design will use an add-on board for the FPGA, since this leaves the FPGA unmodified (less chance of broken hardware). It also combines well with the need for termination resistors. The add-on board must be designed to fit into the two 40 way connectors used as general IO on the FPGA.

Termination Resistors

To provide maximum flexibility, each MII signal will have multiple termination options. There is a termination resistor in front of the ribbon cable (source dampening resistor), a series resistor at the end of the ribbon cable (series termination resistor) and a parallel termination resistor combined with a capacitor (parallel termination). To provide the terminations at the PC side and the FPGA side, a number of add-on boards are designed.

PC Mountable System

To achieve maximum flexibility, both for testing/measuring and for unforeseen problems, the new level converter will be able to be placed outside the PC with a ribbon cable. In order to terminate the MII signals, an add-on board is required for the PCILAN card. When the level converter is placed inside the PC, the add-on board is not required.

Edimax MII Transceiver Attachment

The cable which is currently used for the MII transceiver is a modified ribbon cable with a manually soldered connector. A small add-on board has been made which replaces this cable.

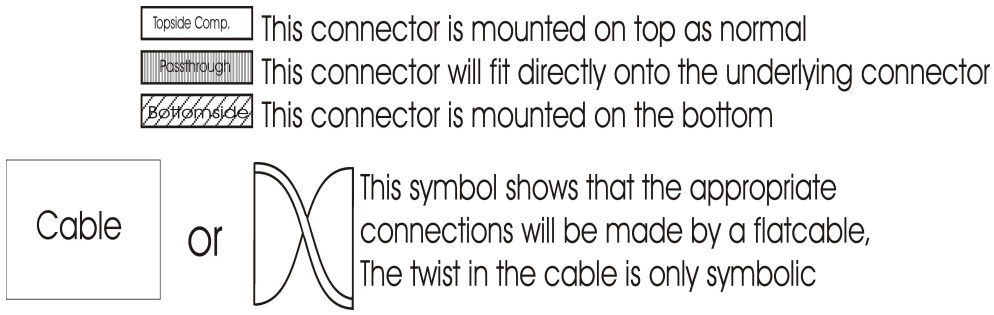


Figure 4.1: Symbols used in system overview

4.1.2 Further considerations

PCB Choices

To save costs, the level converter board is designed as a dual layer board, and not as a multilayer board. This requires more attention to EMC considerations with respect to the ground plane. By carefully placing the components and the ground, the amount of EMC noise pickup should not have much influence on the system

The new level converter and add-on boards are designed as a single PCB, and must be separated manually. This is done since each individual PCB has a relatively large startup cost associated with it.

4.2 System Level Setup

The new design consists of four boards: the new level converter, an add-on board for the PCILAN card when the level converter is placed outside the PC for measurements, an add-on board for the FPGA to add signals to the ribbon cable and to add termination resistors, and an add-on board to facilitate the connection of the MII transceiver.

The level converter can be connected directly to the PC, or it can be connected through a ribbon cable. Also, either the MII transceiver or the FPGA can be used for the MII management interface.

The following pictures show an overview the proposed setup. First, Figure 4.1 shows the symbols that will be used in the overview. The crossed cable indicates that the pin-out at opposite sides of the cable is mirrored with respect to each other. The text inside the connector indicates the purpose (target) of the connector.

The level converter and the MII Transceiver add-on board both have a connector on the solder side of the print. This is done mainly because it simplifies the attachment of the FPGA. When the level converter is placed inside the PC, and the PC is standing right up as normal, the level converter will face downwards, and thus this connector will face upwards. The FPGA can then be connected to the level converter without a twist.

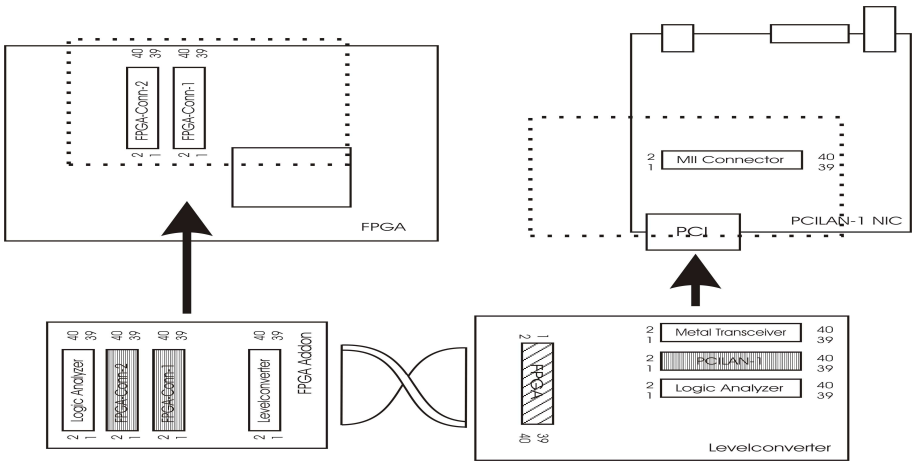


Figure 4.2: Proposed system setup

The pass-through header indicates that the connector is designed to fit directly on top of the underlying board. It can also be used to connect a logic analyzer to the FPGA add-on board.

Intended Setup

Figure 4.2 shows the intended configuration for the system. The level converter is connected directly on top of the PCILAN card, the FPGA add-on card is connected directly on top of the FPGA, and the FPGA add-on card is connected to the solder side of the level converter through a ribbon cable.

External Setup

Figure 4.3 shows the setup when the level converter is connected to the PC with a ribbon cable. The PCILAN add-on board fits on top of the PCILAN card, and a ribbon cable is connected between the two boards. The ribbon cable can be placed directly on top of the pass-through header.

Figure 4.4 shows the attachment of the MII transceiver to the level converter board. The ribbon cable can be connected to the backside of the add-on board as indicated.

4.3 Design

The schematics, layout and component list are placed in Appendix B. Figure 4.5 shows the combined PCB for the level converter and the different add-on boards.

Due to the speed of the interface, the items that are considered most important are:

1. Physical constraints

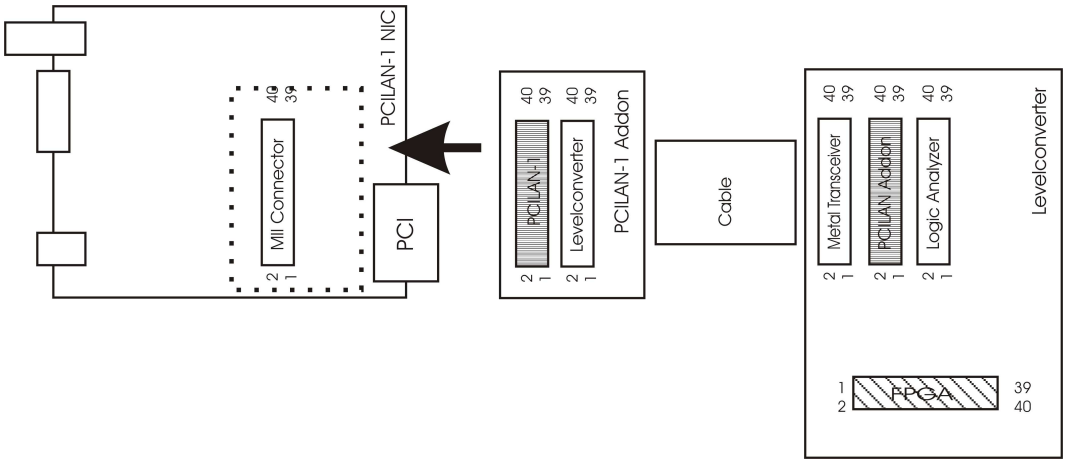


Figure 4.3: External mounted level converter

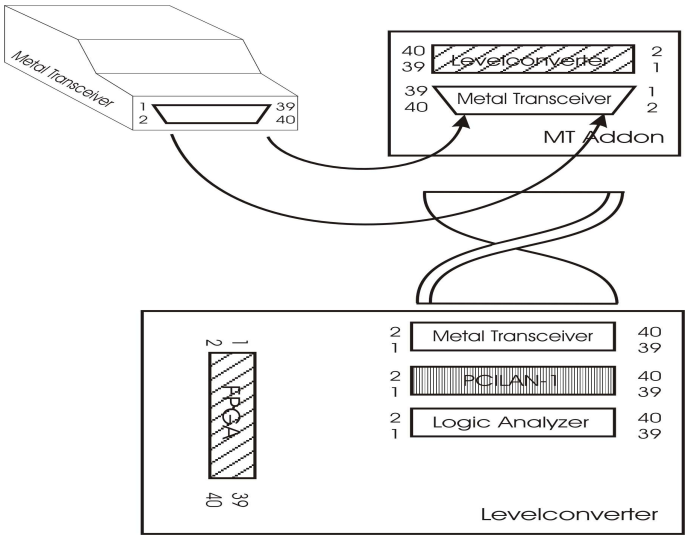


Figure 4.4: Attachment of MII Transceiver

2. Ground plane layout
3. Power trace layout
4. Logical component placement
5. Signal trace layout

4.3.1 Physical Dimensions

In order to fit the level converter inside the PC, the PCB must meet certain physical dimensions.

The power supply connector must be placed near the top of the PCB, in order to connect it to the PC power, and the FPGA connector is placed on the solder side of the PCB.

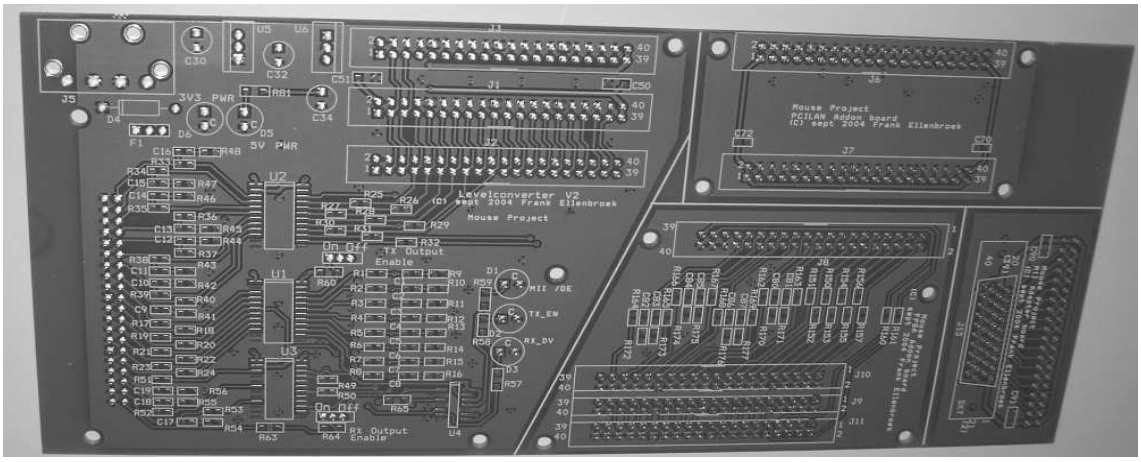


Figure 4.5: New Level Converter PCB

4.3.2 Ground Plane

In order to minimize interference between signals, and to avoid external noise to influence the system, the ground layout is important. Each signal must have ground nearby, and the total loop area between a signal and the ground must be minimized. At places where the ground plane is interrupted by a signal trace, a number of vias and traces have been manually added to improve the connectivity.

4.3.3 Power Traces

Attention was paid to the layout of the power traces for the IC's, to provide a proper ground plane and power decoupling as close to the IC's as possible. There is a central ground strip underneath the three level converter IC's, with numerous vias to the ground traces on top side. One both sides of this strip are the wide power traces (one for 5V and one for 3V3), and the local power supply decoupling is connected directly underneath the power pins for the IC's.

4.3.4 Logical Component Layout

The logical placement of components is mainly important when the PCB must be physically handled, which occurs during buildup and during measurements. This is considered less important than the power and ground layout. All terminating resistors on the PCB are placed close to the level converter IC, to reduce the path length between the resistors and the in-/outputs. All resistors have been placed on the topside of the PCB. This requires more space, but simplifies both measurement and the ground plane routing.

4.3.5 Signal Traces

The signal lines at the MII interface are fixed as they are defined in the standards, but the signals between the level converter and the FPGA can be reordered to improve the system performance. Advantage has been taken from this to swap signals around to obtain a good layout with only a few crossing signals.

The transmit and receive clocks are routed together, and use a separate IC from the other data lines. Both clocks are located near the edge of the reordered cable, instead of at the center as is the case in the MII interface connector. This should reduce crosstalk on the print and along the cable.

As there are multiple data lines that need to cross each other. the important concern is to minimize the overlap of traces, and to provide a continued ground along the traces. The layout of the traces is of course also important, but if the ground layout is good, it should not be crucial.

The transmit and receive signals are separated from each other. When a frame is transmitted or received, all data lines change simultaneously, and crosstalk will only be present at the edges of the data, reducing possible glitches on the other signals.

4.4 Results

The new level converter board has been ordered, built and tested. To perform functional testing, the FPGA has been programmed to transmit a pulse train to the MII interface. The FPGA is able to turn the levelconverter on and off with the output enable signal, and the transmit and receive leds are also functioning properly. Before data transmission can be established, first the MII management interface must be operational.

4.4.1 Hardware

Figure 4.6 shows the new level converter PCB. The PCB fits correctly inside the PC, as can be seen in Figure 4.7.

The FPGA add-on board figure 4.8 also fits properly onto the FPGA. The shape is designed fit onto the FPGA while allowing some room for the SMA connectors. Figure 4.9 shows a side view.

4.4.2 Measurements

To compare the new level converter with the previous one, a number of measurements have been made to compare the results. First, the influence of the termination resistors has been measured at the input of the level converter IC. Figure 4.10 shows a number

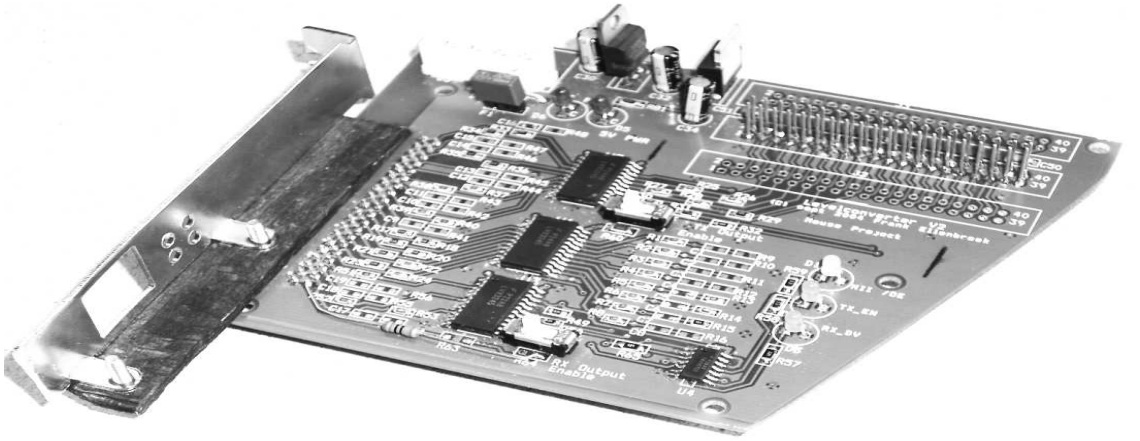


Figure 4.6: New Level Converter

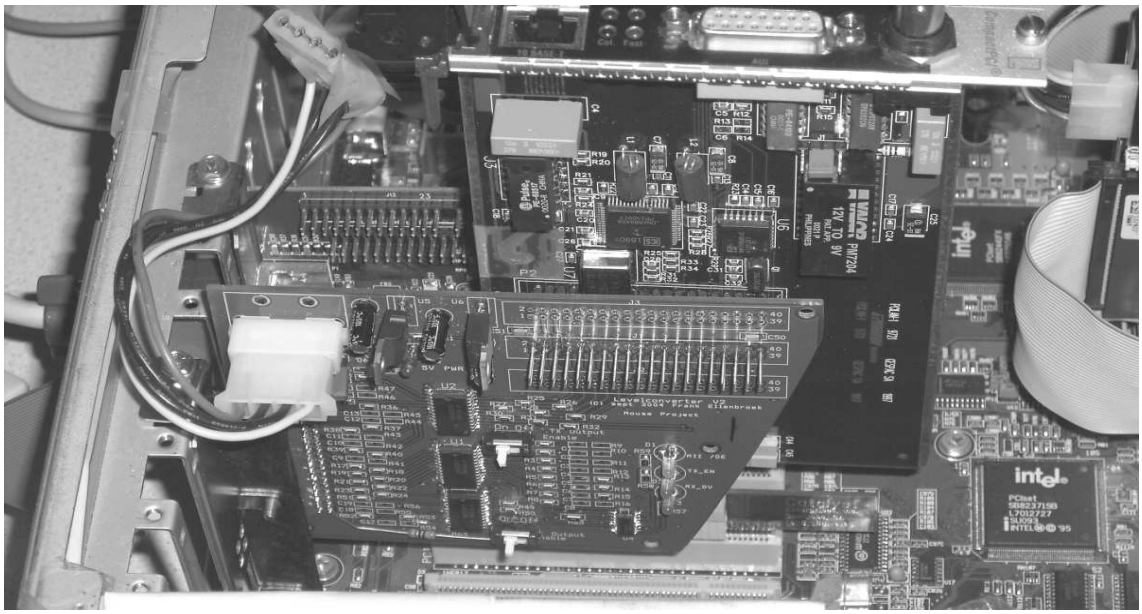


Figure 4.7: New Level Converter inside PC

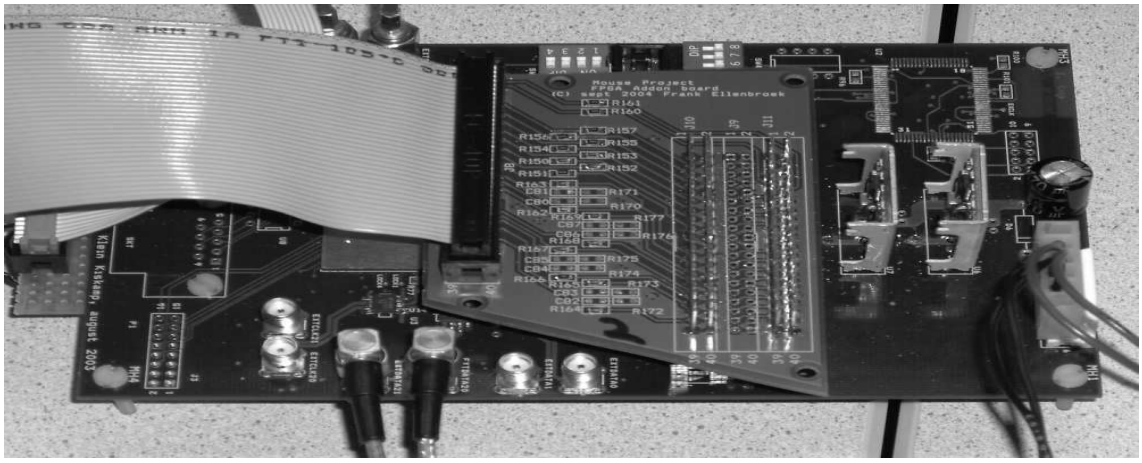


Figure 4.8: FPGA add-on board

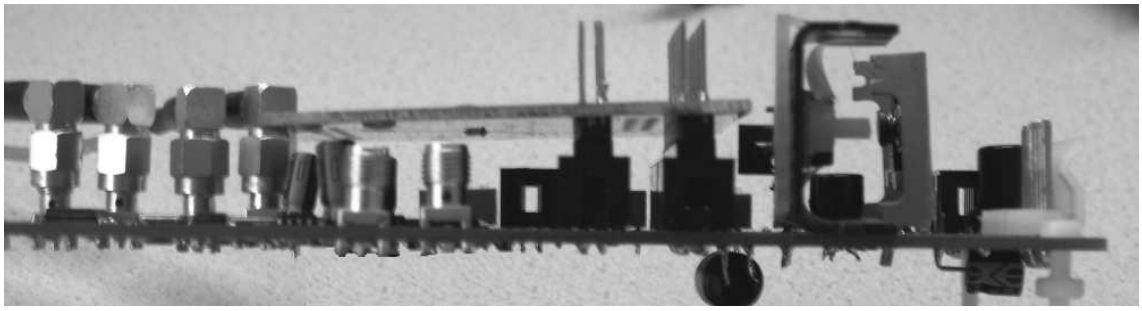


Figure 4.9: Side view of FPGA add-on board

of data bits received from the PC, and there is almost no ringing. There is still some noise present on the signal (0.3 V top-top), but the noise does not exceed the noise amrgin for TTL logic (0.8 V), and should not influence the correct detection of the data.

The RX_DV signal has also been measured again to get an indication of the crosstalk. Figure 4.11 shows the noise present on the signal. The clock and RX_DV signals run along the ribbon cable next to each other, but on the PCB the signals are seperated. The noise present on the signal is reduced considerably.

The small gap at the center of the pulse is an interframe gap, since the picture shows two frames which are sent back to back over the MII interface.

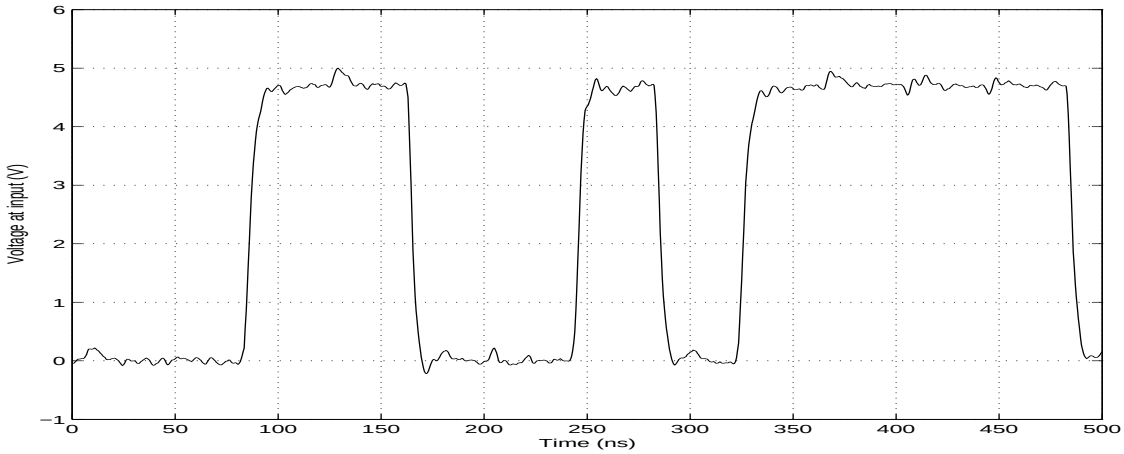


Figure 4.10: Effect of dampening resistors

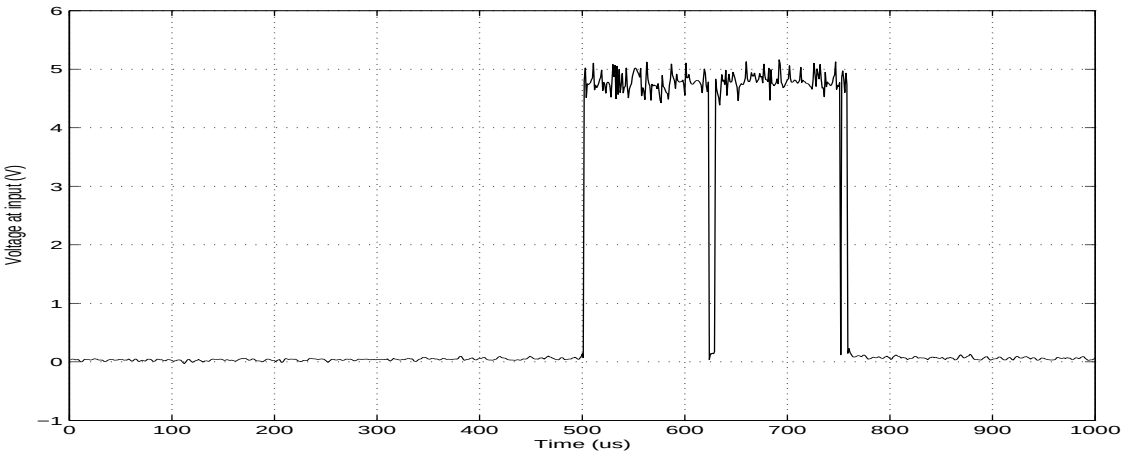


Figure 4.11: Noise on the RX_DV signal

4.5 Conclusions

The new level converter described in this chapter has been constructed, and is functioning properly. All signals are properly terminated, and almost no ringing is visible. The output enable also works correctly, allowing the FPGA to turn the level converter on and off.

Chapter 5

MII Management Interface

5.1 Introduction

The MII management interface provides a configuration interface for physical layer devices (PHY) on the MII bus. The interface is specified in the IEEE 802.3 standard [6]. The implementation of the interface inside the FPGA will provide the basic requirements from the standard, and also include a number of additional features. See also Chapter 7.

The MII management interface will be implemented with VHDL, and simulated inside the Modelsim simulation environment. Next, the design will be incorporated into the Altera Quartus integrated development environment where it can be programmed into the FPGA for actual testing.

Data transfer over the MII management interface is always initiated by the station management entity (STA) which is located in the MAC layer on the PCILAN card. The STA provides the clock (MDC) and initiates all frames which are then transmitted over the interface. A PHY can only access the MII management data bus (MDIO) after it has received the correct address and operation from the STA.

5.2 Considerations

Initially, the MII management interface will be designed with only the minimum requirements, but it can be extended when required. New registers can be added to the FPGA easily.

The physical MII management address for the FPGA will be 26 (11010), and the state machine will run at 25 MHz. A higher operating speed does not provide any benefits, since the maximum management interface operating frequency is 2.5 MHz. At 25 MHz, the state machine has sufficient states between each MDC or MDIO transition to perform multiple state transitions.

The initial implementation will include the control register (0), the status register (1) and an extended register (17) to configure specific settings inside the FPGA. The interface can be easily extended when necessary, after the basic structure is in place.

5.3 Register interface

The initial MII management interface will include the registers necessary to detect and configure the FPGA by the PC. The control and status register are a requirement for each PHY attached to the MII interface, and are used to detect the capabilities of the PHY, and to control the operating mode. The custom extended register (17) is also added to solve the timing problem with the MII transmit clock.

5.3.1 Control register

The control register is the first of the two mandatory registers of the MII management interface. Through this interface, the MAC layer can configure the physical layer operation. Table 5.1 shows the fields present in the control register.

reset

Bit 15 of the interface controls the reset of the interface. When a one is written here, the FPGA must reset itself into a default state. The reset must be completed within 500 ms. After the reset, the FPGA must detach itself from the MII interface, and wait for the STA to initialize the communication again.

transceiver mode

Bits 13 and 6 control the operating speed of the FPGA. The only valid value here is (1,0), which indicates that the FPGA is operating at 100 Mbps, which is the operating speed for the MOUSE system. Bit 8 controls the duplex mode, which is fixed to half-duplex (0). Write commands to the speed and duplex mode bits are ignored.

power down and isolate

While in either power down mode or in isolated mode, the MII management interface must remain operational.

When bit 11 is set, the FPGA is placed in power down mode. The FPGA implements this mode by turning off the transceiver. During power down, the FPGA should not transmit any data over the MII interface, except responses to MII management frames.

Table 5.1: Control Register

bit	function	initial value	writable	comment
15	reset	0	yes	write 1 for reset
14	loopback	0	yes	enter loopback mode
13,6	speed select	1,0	-	indicates 100Mbps
12	autonegotiate enable	0	-	
11	power down	0	yes	
10	isolate	0	yes	isolate MII Bus (output enable)
9	reset autoneg	0	-	
8	duplex mode	0	-	half duplex
7	collision test	0	yes	assert COL for 10 bit times
5-0	reserved	0	-	

When the isolate bit (10) is set, the FPGA must isolate itself from the MII interface. This is done by turning off the level converter through the new output enable signal on the level converter.

loopback and collision

The loopback (14) and collision test (7) bits can be used by the MAC layer to test the operation of the FPGA. When loopback node is selected, the FPGA is required to disable the receiver. Each frame which is sent by the MAC layer to the FPGA must be returned to the MAC layer, in order to test the transmission and reception of data.

When collision test is enabled, the FPGA must assert the COL signal within 512 bit times (5.12 us) after the TX_EN signal is asserted. When the TX_EN signal is de-asserted, the COL signal must be de-asserted within 4 bit times (40 ns).

auto-negotiation

The MOUSE project implements a fixed half-duplex operating speed of 100 Mbps; therefore auto-negotiation is not required and also not implemented.

5.3.2 Status Register

The status register informs the MAC layer about the status of the FPGA, and contains information about the capabilities of the PHY. The register is read only, but some registers will be cleared (or set) after the register is read. Table 5.2 shows the contents of the status register.

transceiver capability

The available operating modes for the FPGA are presented in bits 15 through 9 of the status register. Since the FPGA is only capable of 100 Mbps half-duplex operation, only bit 13 is set.

extended status and capability

When bit 8 is set or bit 0 is set in the status register, it indicates that the PHY has a number of extended capabilities. Bit 8 is set when the PHY has 1000 Mbps capability, which requires an extended status register to be present. Bit 0 is set when a PHY has implemented more than the basic registers (0 and 1). The extended registers are implemented in the FPGA, and therefore the bits are zero.

preamble suppress

Bit 6 indicates whether a PHY can accept frames without the 32 bit preamble. Since the FPGA can accept frames without the preamble (synchronization is done on the start-of-frame delimiter (ST)) this bit is set. The PC driver for the PCILAN card does not send frames without preamble, but some other drivers might do that. See also Section 5.4.2 for considerations regarding the absence of a preamble.

auto-negotiation status

Bits 5 and 3 of the status register indicate the status of the auto-negotiation process. Since auto-negotiation is not required, it is also not implemented in the FPGA, and both bits remain zero.

error status bits

The remote fault (4), link status (2) and jabber detect (1) bits indicate whether any problems have been detected with the communication.

The link status implementation is PHY specific, and is mainly useful for full-duplex point to point connections. The FPGA does not actually use the link status information, and currently the FPGA returns link status as one, indicating a valid link.

The remote fault bit is also PHY specific. It is currently not used in the FPGA, and remains zero.

The jabber detect bit indicates whether a jabber condition was detected inside the FPGA. The FPGA implements a jabber detector, and will set this bit on a jabber condition. The jabber detect bit is cleared each time the status register is read by the STA.

Table 5.2: Status register

bit	function	initial value	writable	comment
15	100Base-T4	0	-	
14	100BaseX-FD	0	-	
13	100BaseX-HD	1	-	MOUSE operating mode
12	10Base-FD	0	-	
11	10Base-HD	0	-	
10	100Base-T2 FD	0	-	
9	100Base-T2 HD	0	-	
8	rxtended status	0	-	no extended status registers
7	reserved	0	-	
6	MF preamble suppress	1	-	FPGA supports suppressed preamble
5	autoneg complete	0	-	
4	remote Fault	0	-	cleared when read
3	autoneg ability	0	-	
2	link status	1	-	zero on link failure, set when read
1	jabber detect	0	-	set after jabber condition, cleared when read
0	extended capability	0	-	

5.3.3 Custom Extended Register

The implementation of MII management registers 16 through 31 is up to the designer of the PHY. Initially, only one register is implemented, called the custom extended register with register address 17. Table 5.3 shows the contents of the register.

bypass scramble and nrzi

When the bypass scramble (6) bit is set, the transceiver will bypass the scrambling phase of the transceiver. The bypass operation can also be set by the dip switches on the FPGA board, and scrambling will only occur when this bit is cleared and the dip switch is off.

Bypass nrzi (5) works identically to the bypass scramble operation. When either this bit is set or the external dip switch is on, nrzi encoding and decoding is disabled in the FPGA.

In order to establish communication, all FPGA's must have the same settings. Unless mentioned otherwise, both scrambling and nrzi encoding have been disabled.

Table 5.3: Custom extended register

bit	function	initial value	writable	comment
15 - 7	reserved	0	-	
6	bypass scramble	0	yes	turn scrambling on/off
5	bypass nrzi	0	yes	turn nrzi encoding on/off
4	reserved	0	-	
3,2,1,0	MII tx delay	0,1,0,0	yes	configurable delay between external transmitted clock and internal read clock.

MII TX delay

The MII TX delay (3-0) bits configure a delay between transmitting the external TX clock on the MII interface, and reading the incoming data on the same interface. The reason for this delay has been discussed in Section 3.3.3.

The reference clock for the delay is the internal TX_CLK, which is obtained by dividing the 125 MHz clock by five. The external clock can be delayed between zero and four periods of the 125 MHz clock using delay elements

The MII TX delay value indicates the amount of delay in steps one 125 MHz clock cycle. The delay can be configured between 0 (no delay) and 4 (4 clock periods delay). The clock delay should help to compensate for the delay introduced by the level converter.

The default value of the register is 4 (0100), which indicates a delay of 2 full clock periods at 125 MHz (16 ns).

5.4 Interface Implementation

5.4.1 Overview

To implement the MII management interface, the FPGA must correctly detect and decode each incoming frame, and perform the actions requested by the STA.

The basic operation of the MII management interface has been discussed in Section 2.1.6. The MII management frame structure (Figure 2.4) consists of a number of fields. First, the preamble is send to the FPGA, followed by the start-of-frame delimiter (ST), which indicates that the frame has started. The ST consists of a low bit followed with a high bit on the MDIO line. The FPGA does not need a preamble to synchronize on the incoming clock, since the operating speed is higher than the MII management clock speed, and each transition can be detected correctly. The FPGA only requires the detection of the ST in order to determine whether a frame has started. Next, the

operation type, PHY address and register address are sent to the FPGA. After that, there is a small turnaround time included to give a PHY the time to prepare for data transmission. Finally, the data is transmitted to or from the FPGA. When the last data bit is send, the frame is complete, and the FPGA can prepare for the next frame.

In order to assist in the decoding of a frame, the design will consist of the following five phases:

1. The FPGA will wait until the ST is detected;
2. All relevant control information is read into a shift register (12 bits);
3. The operation is decoded and the data transfer is prepared;
4. 16 bits of data are transmitted or received;
5. The operation will be finalized and the process restarts.

5.4.2 Schematic Design

In order to implement the five phases mentioned in the previous paragraph, the MII management interface will consist of the modules shown in Figure 5.1. The following paragraphs will describe the functionality of the different modules in the MII management interface.

Synchronization

The MII management interface runs at a clock which is not synchronized to the internal clock on the FPGA, therefore the incoming clock and data are first synchronized with the internal clock.

Bit counter

The bit counter counts the number of bits that need to be shifted into or out of the shift register. It can be initialized to count to 12 in the case of the operation (2 bits operation, 5 bits physical address and 5 bits register address), and it can be initialized to count to 16 in the case of data transfer. The internal counter is decreased by one on each rising flank of the MDC signal.

While the counter is not zero, the shift enable signal is high, and the shift register can shift data on each rising flank of the MDC.

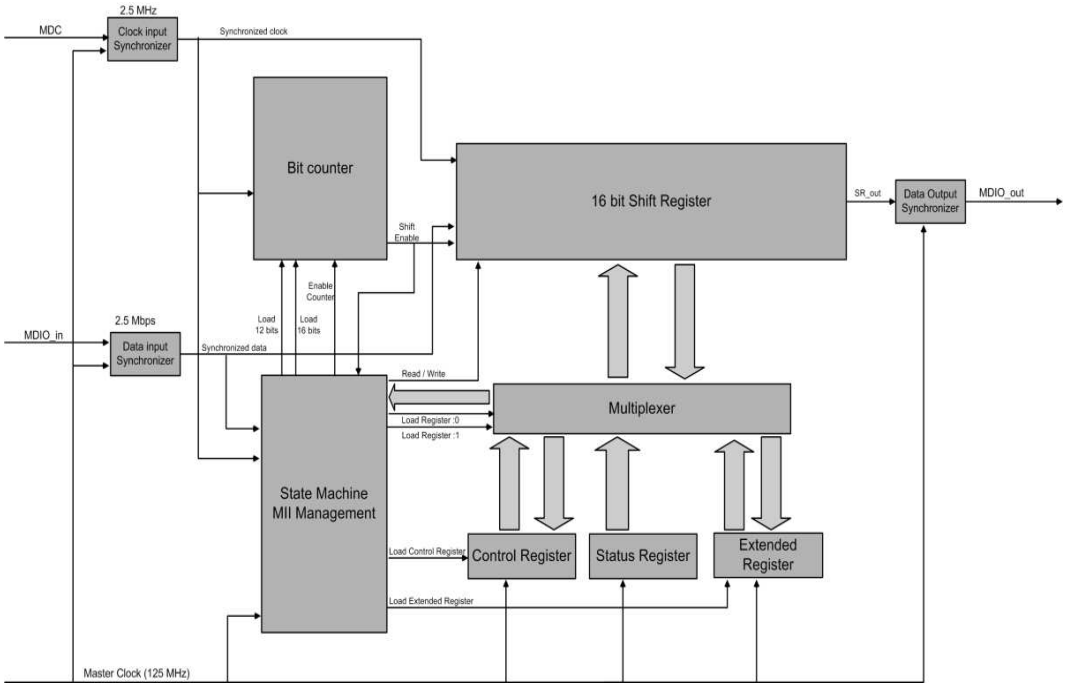


Figure 5.1: MII management schematic overview

Shift register

The 16 bit shift register receives the data from the external MDIO input, and also receives the data which must be transmitted to the STA during a read operation. On each rising flank of MDC, the shift register will shift all data one bit, from high to low, and the MDIO input will be shifted into bit 15. Bit 0 inside the shift register will be written to the MDIO output during a read operation.

Multiplexer

The multiplexer controls which register is read or written. New registers can be added in the multiplexer module without changing anything at the other modules. The select_mux input controls which register is active. If the load_register input is active, the contents of the currently selected register will be placed on the shift register input, and will be loaded into the shift register when the md_load_reg signal is active. Register bits that are not implemented are zero by default.

Timeout

The first implementation of the management interface included a timeout counter, to prevent the interface from losing synchronization. Misalignment can happen when the FPGA is powered up or reset during a transfer over the management interface. Since the IEEE 802.3 specification does not provide any minimum speed requirements, the

FPGA can not determine exactly whether data received after a reboot is an actual ST or part of an ongoing frame. The timeout counter will reset the interface to a known state when no communication has been received for a predetermined time.

Misalignment will normally not occur when a preamble is sent with the data, as was previously assumed. When a full preamble of 32 bits is sent at the start of a frame, any previous operation will finish before the next ST. The total length from the ST up to and including the last data bit is 32 bits, therefore any misaligned frame will finish with the 32 preamble bits as input, and the next ST will be detected correctly.

The misalignment of a frame can only occur in the case of a suppressed preamble, and the current network card driver does not include support for a suppressed preamble. Therefore the timeout module is not used anymore, but it is still left in the design.

State machine

The state machine keeps track of the progress in the frame, detects errors and controls the signals send to the other modules. A flowchart of the state machine is included in Figure 5.2.

After a reboot or software reset, the state machine will initialize the management interface, initialize the registers to the default value, and then wait for a frame.

When a start condition is detected, the bit counter is initialized to read 12 bits into the shift register, which contain the requested operation information. The counter will count down by one on each rising flank of the MDC signal. When the counter is finished, the operation is decoded and the necessary preparations to read or write data are made. Next, the counter is initialized to transmit 16 data bits into/out of the shift register. After the counter is finished, the data is transmitted to the correct register (in the case of a write operation) and the state machine will return to the default state.

5.5 Implementation

5.5.1 VHDL Code Structure

The software used to program the FPGA can recognize basic functions, to which it will assign a standard structure. The basic functions recognized include shift registers, counters and registers. The software does not optimize a complete design to minimize component count, but it will optimize the logic element (LE) layout to minimize timing delays. It is therefore important that the designer knows which layout the software recognizes. Inside the FPGA all combinatorial assignments occur simultaneously, and sequential assignments require the use of a register. This is in contrast with *normal* programming languages like *C* and *Pascal* which perform tasks in the order described in the source file.

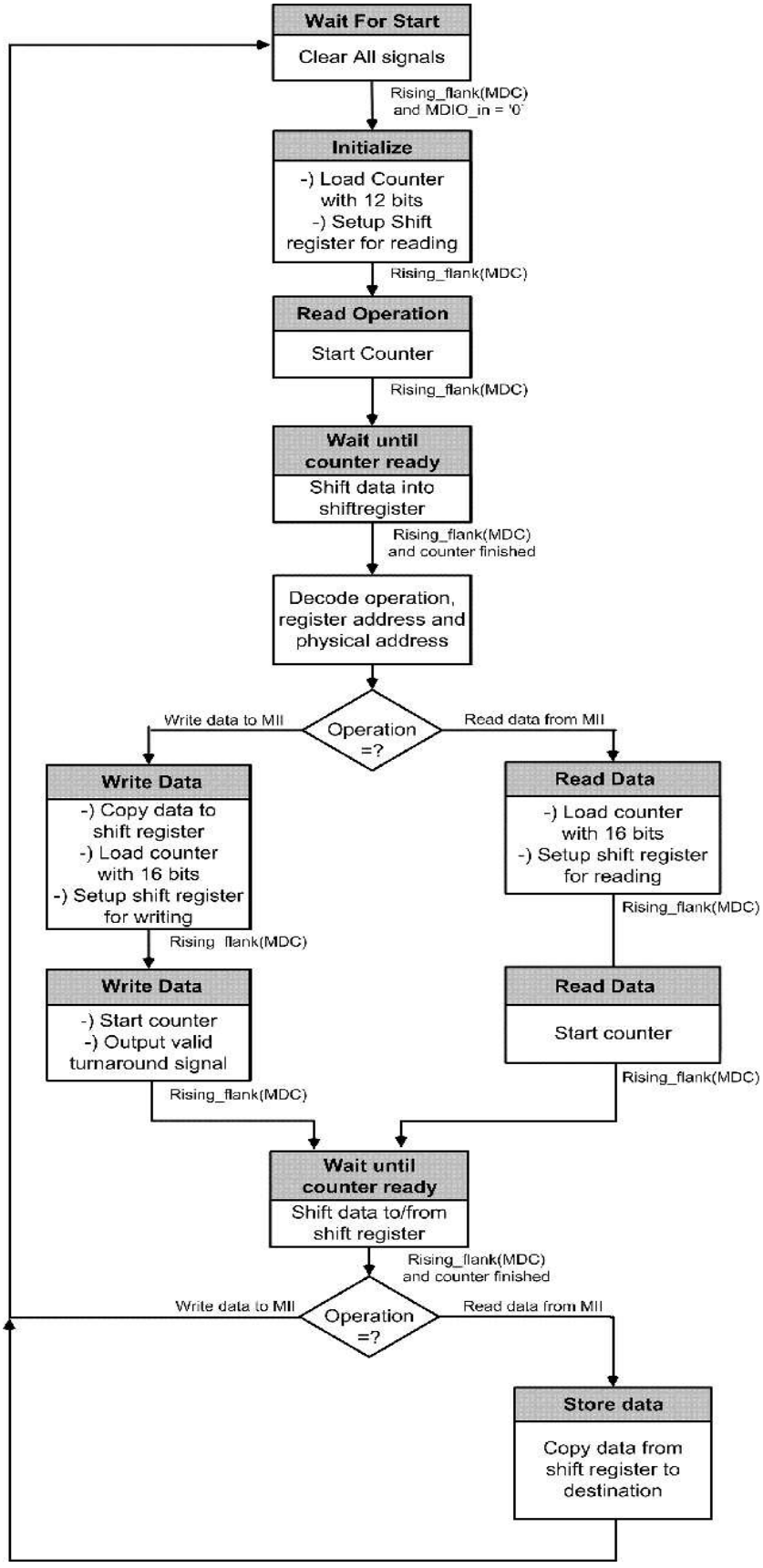


Figure 5.2: MII management state machine

Entity Declaration and Initialization

Each new VHDL component is placed in a separate VHDL file. It starts with the entity declaration, which lists the inputs and outputs of the component, followed by the architecture of the component. The architecture first lists all external components to include, followed by the signals used inside the component. The external inputs and outputs of the entity are already declared, and should not be redeclared here. All inputs and outputs for the included components must be declared here. Next, the instances of the included components are mapped onto the signals. An example structure is shown in Appendix D to show the basic layout.

Combinatorial logic

The combinatorial part of the new component is described in the combinatorial section of the VHDL file, and includes all logic functions and assignments required for the functionality of the module. Each signal can only be assigned one value at any moment, and even must be assigned a specified value at any moment. It is not allowed to assign a value to a signal only under specific conditions. A signal can be a function of a register value or of another combinatorial signal value.

If possible, a conditional statement should not depend on a combinatorial signal due to timin considerations, and should only depend on a register value. Section 5.5.2 will discuss the timing considerations.

Sequential logic

The sequential part of the component contains the signals which only change after a clock pulse (registers). First, the initial value of the register after an asynchronous reset is described, followed by a clock'event condition. This code structure is recognized by the software, and indicates that the structure describes a register. A register does not have to be assigned a value at any moment, since the register is only changed to a new value on a clock flank, and only when the specified conditions are true. A register cannot be assigned more than one value at a time.

5.5.2 Timing

The timing values mentioned here belong to the EP20K1500EBC-1 FPGA, which has slightly larger delays than the EP20K400EBC-1 FPGA. See Tables 2.2 and 2.3 for the delay specifications.

The maximum operating speed for the FPGA is 175 MHz. To obtain such a high speed (or a speed of 125 MHz) the design of the VHDL code must be optimized. At 125

MHz the clock period is 8 ns. Before the rising (or falling) clock flank, the combinatorial part of a design must be stable, and adhere to the setup and hold time of the registers.

Combinatorial delay

Combinatorial logic inside the FPGA is mainly implemented inside the look-up-table (LUT) of a logic element (LE). A LUT has a delay of 0.8 ns, and each interconnect requires a minimum of 0.28 ns. The software does not optimize signals by combining terms, so each combinatorial signal that is dependent on another signal utilizes a new LUT. Each new LUT including interconnects requires another 1.08 ns minimum.

The combinatorial delay can be decreased by adding an intermediate register in a combinatorial path, which must hold the intermediate value. This way the clock speed can be increased, by splitting up the combinatorial logic. This is required to keep the system synchronized. The cost is a number of extra bit times delay. Also, it is not always possible to add register delays to the system, since other parts of the design can depend on the immediate availability of the values. In these cases, all other signals must also be delayed.

Interconnect delay

When signals must travel long distances across the FPGA, the fast interconnect is used, which has a maximum propagation delay of up to 4.43 ns. If a signal is needed at multiple locations around the FPGA, the slow Fasttrack network is normally used to distribute the signals. It is therefore recommended to buffer the incoming signals at the exit of a structure (in the topmost VHDL structure) and just before the signal is used in another structure. Again the cost is number of extra bit times delay.

5.6 Simulations

The MII management interface has been implemented in VHDL, and a number of simulations have been performed in Modelsim to verify the correct operation.

In Figure 5.3 the results of a MII write transaction are simulated. The contents of the frame will change the loopback, power down and isolate signals in the FPGA. The management interface retects the start of the frame, and determines from the first part of the frame that the operation is a write action to the control register. The md_load_op and md_load_d signals are activated at the correct position in the frame, and all data is read into the shift register. Following the correct reception of the frame, the loopback, power down and isolate signals are modified according to the contents of the frame.

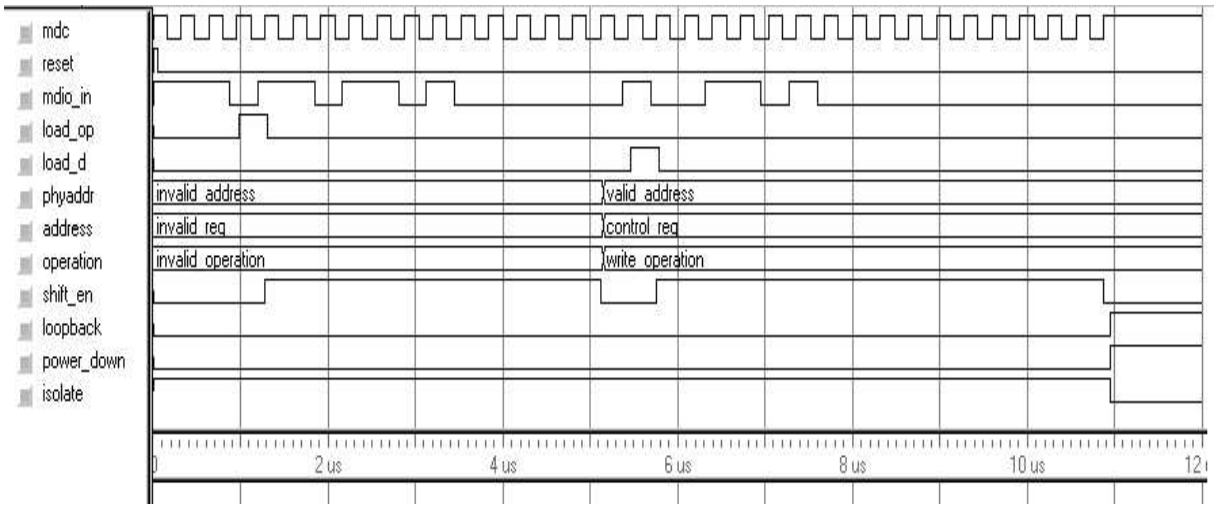


Figure 5.3: Modelsim simulation of a MII write operation

5.7 Conclusion

The MII Management interface has been designed and implemented, and the simulation results show that the interface works correctly. A code example has been provided in Appendix D.

Chapter 6

Configuring the FPGA

6.1 Introduction

Chapter 5 introduced the basic MII management interface, and presented working simulations for the interface. This chapter will investigate the actual performance of the interface in a physical system. First, the communication between the PC and the MII management interface is established, which is followed by an analysis of the boot sequence of the PC.

6.2 SignalTap Logic Analyzer

6.2.1 Hardware Overview

One of the many features of the Altera FPGA is the fact that is supports a powerful internal logic analyzer for the FPGA. The SignalTap logic analyzer (SLA) in the FPGA can be attached to the PC by means of the ByteBlasterMV programmer included with the Quartus software. To use the SLA, the FPGA must include a JTAG interface, as defined in IEEE 1149-1 [5]. This interface can also be used to program the FPGA.

6.2.2 Features

The Quartus IDE can be used to configure the SLA, by setting simple or complex trigger conditions, by defining the internal signals to capture, and by setting the amount of bits to capture for each signal.

Up to 128 kb of data can be captured for each signal, up to 128 signals can be watched, and the amount of data that can be captured in one run is limited only by the amount of memory available inside the FPGA. For the EP20K400EBC-1 the amount of free memory is 212992 bits. It is for example possible to capture a complete Ethernet frame (1540 bytes) which is send over the MII interface with 4 bits at a time.

Table 6.1: JTAG signals

signal	function	description	unused state
TDI	test data input	serial input for data and configuration	VCC
TDO	test data output	serial output for data and configuration	open
TMS	test mode select	configuration input to control JTAG operating mode	VCC
TCK	test clock input	clock input	GND
TRST	test reset input	asynchronous reset (optional)	GND

To capture all data, 4 times 4 kb is required, which is 2 kB. This leaves room for up to 48 status and control signals to monitor the internal state of the management interface.

It is also possible to simultaneously implement multiple independent SLA's in the FPGA, each with their own speed, signals and trigger conditions.

6.2.3 JTAG Interface

The JTAG interface consists of a serial communication setup, mainly designed for boundary scan testing. The interface defines 5 signals shown in Table 6.1. The maximum operating speed of the interface on the FPGA is 10 Mbps.

The JTAG specification allows devices to be chained together. In this mode, each device passes all information up through the chain to the next device. The TMS, TCK and TRST signals are common to all devices, and the TCO output of one device must be connected to the TDI input of the next device in the chain.

6.2.4 Required Modifications

It is very unfortunate that the designer of the FPGA board did not include this JTAG interface for the FPGA, especially since the EPC8 configuration device is programmed through another JTAG interface which has been implemented.

The current method of programming for the FPGA is either by programming the EPC8 configuration device, or by using a passive serial interface which is connected to the FPGA.

Figure 6.1 shows the JTAG connections, and Figure 6.2 shows the configuration of the JTAG port connectors on the FPGA board.

The FPGA has been modified to include a JTAG port for the SLA. The modifications required are limited to modifying resistor values and attaching a number of wires. The JTAG port which has been added to the FPGA board is shown on the left side of the FPGA in Figure 3.5.

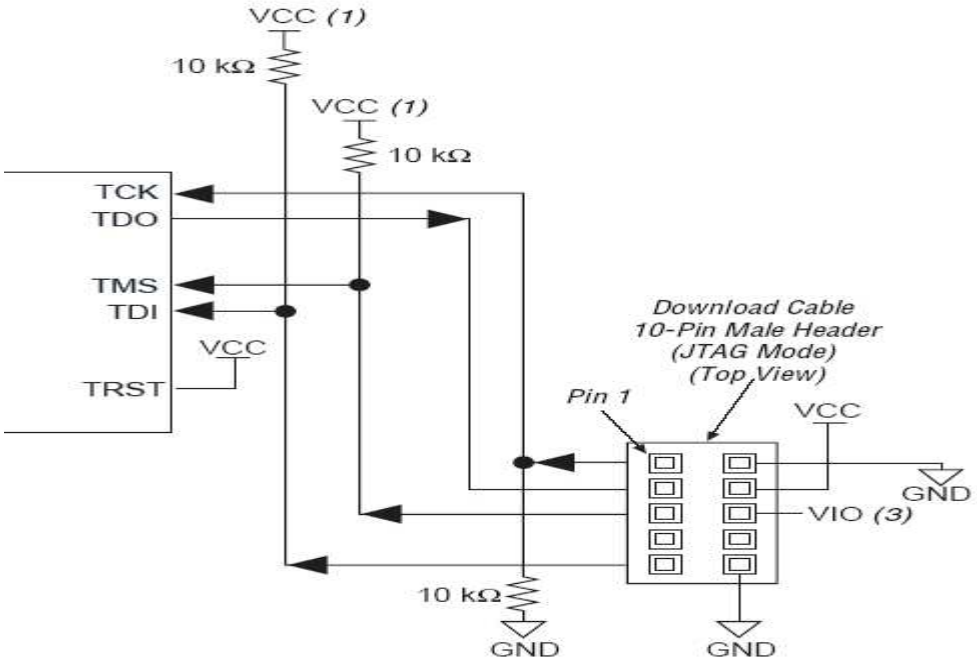


Figure 6.1: JTAG interface connection

6.3 Management Communications

6.3.1 Test Software

To test the MII management interface, the EVBDEBUG software is used. EVBDEBUG is a DOS program which can configure the DS 21140A controller on the PCILAN card. With this software it is possible to configure internal register settings for the 21140A, configure the PCI communication settings, construct and send ethernet frames over the MII interface, and most importantly to read and write MII management registers on a PHY.

To read a register on a PHY device, the following command must be entered:

```
rmi physical_address register_address <ENTER>
```

To write a value to a register on a PHY device, the following command must be used:

```
wmi physical_address register_address value <ENTER>
```

Note that the EVBDEBUG software does not make consistent use of data input formats. The physical address must be entered as hexadecimal (the FPGA at address 26 becomes 1a), the register address must be entered as decimal (which is 17 for the custom extended register, not 11 hexadecimal) and the value must again be entered as hexadecimal.

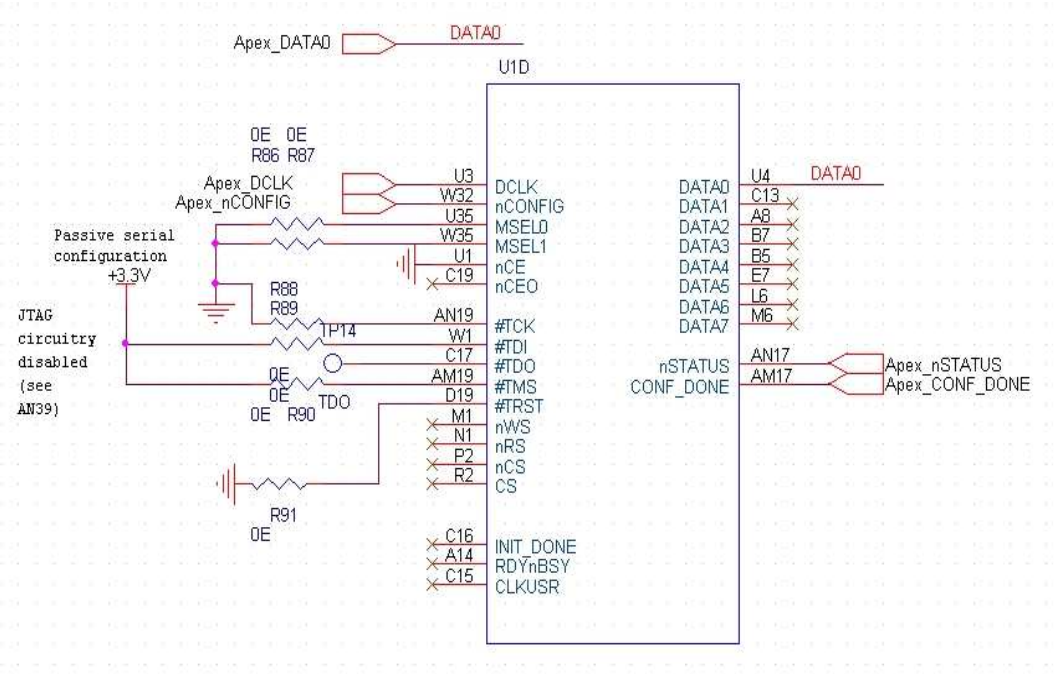


Figure 6.2: JTAG interface pins on FPGA

All bits as defined in Section 5.3 can be read or written in the manner mentioned above. The most common action is to enable or disable the FPGA. To disable the FPGA, it must be put in the power down/isolate mode (bits 11 and 10) which can be accomplished with the following command:

```
wmi 1a 0 0a00 <ENTER>
```

And to turn the FPGA on into the normal transceiver mode requires the command:

```
wmi 1a 0 0000 <ENTER>
```

By default the FPGA will be isolated and powered down.

The EVBDEBUG software implements the MII management interface completely in software. The clock period of the MDC is approximately 2 ms, with delays of 20 ms between multiple clock pulses. Figure 6.3 shows a typical incoming frame over the MII management interface, as it is captured by the SLA. The delay between multiple bursts of clock pulses is probably due to the task switching behavior of the Microsoft windows 98 operating system. It is not important what the cause is, since the MII management interface is required to operate correctly with very low and irregular operating speeds.

6.3.2 Test Setup

To provide sufficient test signals while debugging the MII management interface, a number of batch files have been created which repeatedly transmit data to the FPGA.

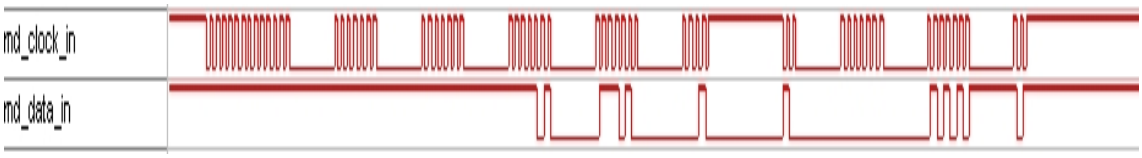


Figure 6.3: MII management frame received by FPGA

The initial test setup consists of a PC with a PCILAN-1 network card, the new level converter connected to the PCILAN card with a ribbon cable and the PCILAN add-on board, and the FPGA connected to the level converter with another ribbon cable. The MDC and MDIO signals from the PC are transmitted to the FPGA, but the MDIO output is not yet returned to the PC in order to protect the PCILAN card from possible bugs left in the VHDL code. The initial test attempts to write data to the FPGA.

6.3.3 Write Frame Results

Figure 6.4 shows a typical frame written to the FPGA over the MII management interface. The frame is captured with the internal SLA of the FPGA.

First, the 32 preamble bits are written to the FPGA, followed by the start-of-frame (ST). The md_busy signal becomes high to indicate that a frame is started, and at the same time the md_load_op signal becomes high, which loads the bit counter with 12 bits. Next the md_shift_enable signal becomes high indicating that data on the MDIO input will be shifted into the shift register. After 12 bits are shifted into the shift register, the md_shift_enable signal becomes low, and the state machine prepares the reception of 16 bits of data by enabling the md_load_data signal. Again the md_shift_enable indicates that the following data bits will be shifted into the shift register. After all 16 bits are read, the frame is finished and the md_busy signal becomes low again.

6.3.4 Read Frame Results

To test the read operation, the batch file is modified to transmit a read operation to the FPGA. The MIDO output must be connected to the PC, since the EVBDEBUG software will abort communications if the FPGA does not respond during the turnaround time.

Figure 6.5 shows a typical read session in the FPGA. A number of preamble bits are left out of the figure. After a ST is received in the FPGA, the md_busy signal is enabled, along with the md_load_op signal. The bit counter is initialized to 12, and the md_shift_enable signal becomes active. The 12 operation bits are read into the FPGA, after which the md_shift_enable signal goes low.

At this point, the operation is decoded, and the physical address (phyaddr), register address (md_select_mux) and operation (md_read_operation) are set. The md_readwrite

Figure 6.4: MII write frame inside FPGA

signal becomes low to indicate a read operation to the PC, and also acts as an enable to transmit the data from the requested register into the shift register.

Next, the md_load_d signal is activated to load the bit counter with 16 bits. Also, the force_mdio_low signal is activated to pull down the MDIO line as part of the turnaround process.

On the next clock pulse, the md_shift_enable signal becomes activated, and data is transmitted out of the FPGA to the PC.

When the frame is finished, the md_shift_ enable signal is deactivated again, and all other signals return to the default state.

6.3.5 Conclusion

The MII management interface is completed and functioning properly, and the FPGA can be configured with the use of software on the PC. It is possible to activate and deactivate the level converter through the interface.

6.4 Driver Problems

6.4.1 Introduction

To perform communications with the new FPGA design, the physical layer (PHY) of the PCILAN card has to be disabled. This is done through the EVBDEBUG software.

Figure 6.5: MII read frame inside FPGA

First, the PCILAN PHY is disabled (power down and isolate), after which the FPGA can be turned on.

One of the problems encountered while attempting to use the FPGA for data transmission is that the physical layer on the PCILAN card was activated again after a seemingly random time when it was disabled. This has happened a number of times after just a few seconds, but it has also been disabled sometimes for half an hour. Since there can be only one PHY active on the MII interface at any time, this causes problems.

6.4.2 Symptoms

The first sign of the problem was the spontaneous re-activation of the PCILAN PHY. Sometimes this happens seemingly random, but most of the time the PHY re-activated after a number of attempts of communication.

To identify the cause of the problem, two PC's with each a PCILAN card are set up to communicate through the internal Ethernet port. A continued stream of ping packets is send from one PC to the other. Next, one of the PHY cards is disabled through the MII management interface, noted below by the text between braces. The resulting communication can be seen below.

```
pinging 192.168.1.3
```

```
reply from 192.168.1.3
reply from 192.168.1.3
reply from 192.168.1.3
reply from 192.168.1.3
reply from 192.168.1.3

{- at this time the PHY is disabled - }

timeout
timeout
timeout
timeout
timeout
timeout
reply from 192.168.1.3
reply from 192.168.1.3
etc.
```

Immediately after the PCILAN PHY is turned off through the EVBDEBUG software, the ping packets do not arrive at their destination anymore. and the timeout is reached. However, after five or six missed frames, the driver is activated again, and communication continues. This experiment has been repeated on both PC's, and the results are consistent.

6.4.3 Driver Reset

It seems that the PC driver software for the PCILAN card will reset the card after a number of attempts to communicate have failed. The seemingly random resets can be explained by the attempts of windows networking to find other computers on the network, which happens occasionally. When enough failures have occured, the driver will reset the card.

In order to use the FPGA to communicate properly, the PCILAN PHY must not interfere while the FPGA is communicating over the MII interface. The current setup will result in conflicts between the FPGA and the PCILAN PHY.

There are 2 easy solutions to the problem.

1. The FPGA can be placed at physical address 0.
   According to the PCILAN datasheet, and through experiments with the Edimax MII transceiver, it is concluded that a device at physical address 0 will be initialized before the internal PHY on the PCILAN card, which has address 1.

If the FPGA fails to initialize properly during the PC boot process, or is not programmed or powered on, the PCILAN PHY takes over the communication.

2. The PCILAN PHY can be physically disabled.
   In this case the FPGA must take over all functions of the PCILAN PHY. This technique has been attempted , and works correctly with the Edimax transceiver, but only when the Edimax transceiver is present during the PC boot sequence. Otherwise, the windows network driver will generate an error and is not usable.

Option 1 has the disadvantage that the PCILAN PHY is still present on the MII interface, and can take over communications if the FPGA is not present or not working (programmed) correctly. Option 2 is the preferred method, since it should eliminate any chance of a conflict. Option 2 is therefore used.

6.4.4 Boot Sequence

In order to use the FPGA during startup, the boot-sequence of the PC has to be investigated. With the SignalTap logic analyzer, the MII management communication is captured during the boot-sequence. The PC driver attempts to determine the first PHY on the MII management interface by reading register 0 of each PHY in order, starting with physical address 0. When the driver detects a PHY (during the turnaround time the addressed PHY is required to pull down the MDIO line) it attempts to read all 32 registers of the PHY. The FPGA implementation of the MII interface did not implement all registers, and therefore the communication failed. The FPGA has been modified to return all zeroes on registers that are not implemented, and the boot-sequence is started again. This time all registers are read.

After the driver has read all 32 MII management registers, it will send a reset (bit 15 in the control register) to the PHY and attempt again to read all 32 registers. When this process is finished, the driver will activate the FPGA in on of the operating modes detected from the status register. The control and status registers are read again to confirm the configuration, and the boot-sequence is finished.

With the help of a number of improvements to the MII management interface, the PC can enable the FPGA during the boot sequence, and the communication can commence without human intervention, and without interference from the PCILAN PHY.

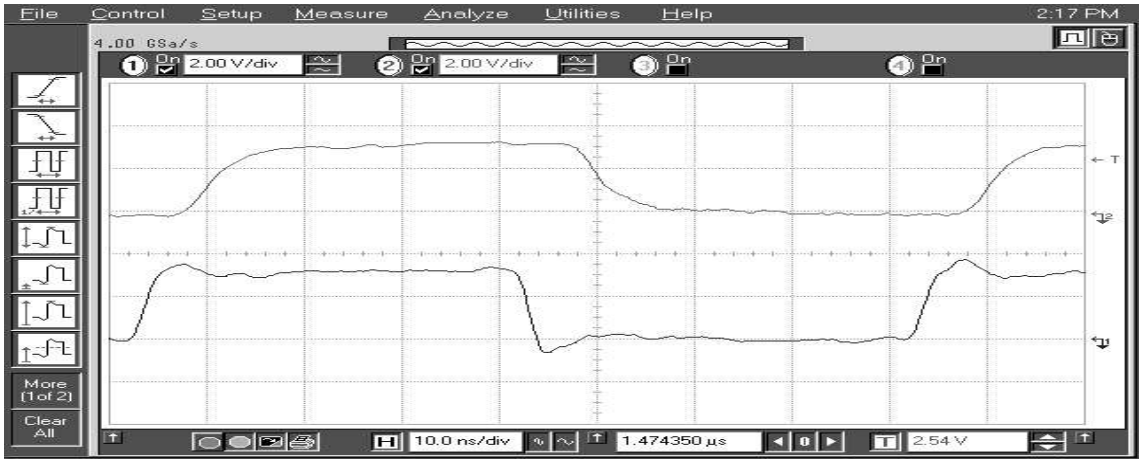


Figure 6.6: Delay for data bits between PC and FPGA

6.5 MII TX Delay

6.5.1 Delay Measurements

To get an indication of the delay the level converter introduces to the system, a measurement has been made to show the delay of a signal over the level converter and a ribbon cable. The measurements have been made with an operating speed of 100 Mbps. The signal is measured at the connector on the PCILAN card, and also at the input of the FPGA. This delay includes the delay of a 20 cm ribbon cable. Figure 6.6 shows the results. The delay of the rising flank is approximately 8 ns, and the delay of the falling flank is approximately 11 ns. These values were expected from Section 2.2.4. The round trip delay is twice as long.

6.5.2 Measurements

To solve the synchronization problem for the MII TX clock, the clock must be delayed. This has been implemented in the FPGA, and measurements have been performed to determine the correct operation. Figure 6.7 shows the relation between the transmitted clock and the received data without compensation.

With the MII TX delay register, the delay can be increased to provide a clock flank at the center of a bit. Figure 6.8 shows the measured relation between the internal TX clock and the received data when the external clock is delayed. The delay can be configured in 5 steps.

Figure 6.7: MII Data and corresponding clock without compensation

Figure 6.8: MII Data and corresponding clock with compensation

6.6 Conclusion

The current setup for the FPGA allows the FPGA to be configured by the driver and EVBDEBUG software on the PC, and allows the PC to automatically configure the network during system boot.

With the MII TX delay configuration registers, the delay of the tx_clk can be configured to provide a valid read clock for the transmitted data.

Chapter 7

Communication Between FPGA's

7.1 Introduction

The next phase of the project is to establish the communication between multiple FPGA's, in order to simulate a network. Initially, this will be done with two FPGA's, with the ultimate goal of connecting three FPGA's together to form a simple three terminal network. Two FPGA's are already available, a third FPGA has to be constructed, and a number of problems have to be solved in order to create the final network. This chapter will focus on establishing the communications between two FPGA's, and the next chapter will focus on the requirements for a three terminal network.

7.2 Synchronization

One of the challenges in (digital) communications is synchronization. When two nodes on a network use a different clock source, the clocks will drift apart; this can result in communication errors. In order to prevent the clocks from drifting apart, either one of the modules must synchronize its clock to the other, or both must use the same reference clock. In the analog part of the MOUSE project, the synchronization is solved by transmitting the clock and the data signal together over the optical fiber, and synchronizing the FPGA clock to the incoming reference clock. The analog part is not available at this moment, so the clock has to be obtained from another source.

The initial setup will use two clock generators, one reference clock, and the other clock synchronized to the first one. This will eliminate clock drift in the system, but will require a careful control of the delays in the system. In the following, the internal delays of the components are considered to be zero. This is a reasonable approximation, since input and output buffers in the FPGA are synchronized to in the incoming clock, and the second generator is supposed to be synchronized to the incoming signal.

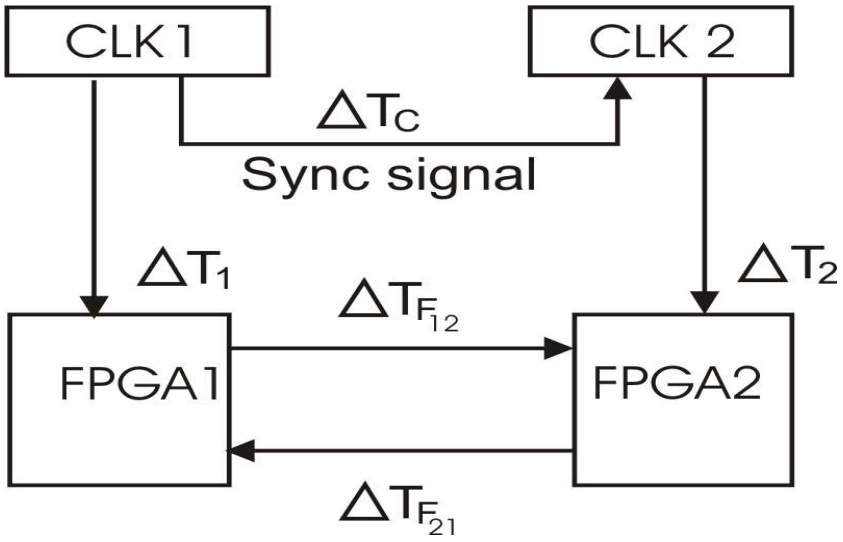


Figure 7.1: Delay in a 2 FPGA system

Figure 7.1 shows the situation with two FPGA's fed by different clock generators. For the data transmitted from FPGA 2 to FPGA 1 to arrive synchronized with the reference clock generator (CLK1), the following condition must be true:

$$\Delta T_1 = \Delta T_2 + \Delta T_C + \Delta T_{F_{21}} + NP \tag{7.1}$$

Here, NP is an integer number N of full clock periods P, and $\Delta T_{(x)}$ are the delays between the different components as shown in Figure 7.1.

For the data transmitted from FPGA 1 to FPGA 2 the following condition must be true:

$$\Delta T_1 + \Delta T_{F_{12}} = \Delta T_2 + \Delta T_C + NP \tag{7.2}$$

If the cables between the FPGA and the clock generators (ΔT_1 and ΔT_2) are of equal length and type, the delays are equal, and the equations can be simplified to:

$$\Delta T_{F_{21}} = NP - \Delta T_C \tag{7.3}$$

and

$$\Delta T_{F_{12}} = \Delta T_C + NP \tag{7.4}$$

It can be seen from Equation (7.3) and in Equation (7.4) that the system can be synchronized by carefully choosing the delays $\Delta T_{F_{21}}$ and $\Delta T_{F_{12}}$, by adjusting the cable lengths.

It is therefore possible to use a single reference generator to produce the required synchronized clock for the MOUSE system.

7.3 System Configuration

7.3.1 Timing

To enable communication between two FPGA's, the setup as shown in Figure 7.1 has been used, but the clock is obtained from the clock generator which will be discussed in Chapter 8. The reference clock has multiple outputs, which are synchronized. This setup is the same as when the delay T_C is zero in Figure 7.1.

The new level converter is mounted on the PCILAN card in the PC, and is connected to the FPGA through a ribbon cable. The serial outputs of the FPGA's are directly connected to the input of the other FPGA. This setup is the same setup which is required for a full duplex point to point link, but since the MOUSE project only supports a shared medium, half duplex mode is used. The transmit and receive paths are seperated from each other, therefore full-duplex operation is possible in this setup. The setup will only be used in half-duplex mode however, and a collision will occur if both FPGA's are transmitting simultaneously.

7.3.2 10 Mbps

The initial tests with 100 Mbps communication have failed, partly due to timing issues inside the FPGA. The choice has been made to implement 10 Mbps communication first, which does not have very strict timing constraints, and focus on 100 Mbps communication later. When the 10 Mbps communication works, there is proof that the logical operation of the system works correctly. In order to scale the system to 100 Mbps, attention will have to be paid to the delays and timings inside and outside of the FPGA.

7.3.3 Frame Transmission

With the 2 node system, a frame is succesfully transmitted over the MII interface to the FPGA, encoded and transmitted to the second FPGA, and received by the second PC. A frame has been captured with the Signaltap Logic Analyzer (SLA) for verification. The frame is correctly read and processed inside the FPGA. Figure 7.2 shows the frame as it is received at the MII interface on the second PC. It can be verified that the frame is a correct Address Resolution Protocol (ARP) frame. This verification will be done in Section 7.4.1.

7.4 10 Mbps Performance

7.4.1 Verification

To verify the correctness of the communication, a frame has been captured and manually checked. The captured frame is an ARP request frame, for which Figure 7.3 shows the structure. The frame is captured with the internal Signaltap logic analyzer (SLA) in the FPGA.

First, the FPGA receives 7 preamble bytes, followed by the start-of-frame delimiter (SFD). Next, the 48-bit MAC broadcast address (FF:FF:FF:FF:FF:FF) and MAC source address (00:C0:E5:80:05:A7) are send.

The frame type and options are shown in Table 7.1

Table 7.1: MAC/ARP option fields

Byte (hexadecimal)	Option	Description
08 06	ethertype	0x0806 = ARP for IP
00 01	hardware type	0x0001 = Ethernet
08 00	protocol type	0x0800 = IP
06	hardware addr. length	length = 6
04	protocol addr. length	length = 4
00 01	ARP operation	0x0001 = ARP request

The type and option fields are followed with the source address (again) and source IP (192.168.1.4) and finally the destination address and destination IP (192.168.1.3). The destination address is being requested, therefore the fields are zero.

Since the frame is shorter than the required minimum frame length, it is completed with a padding field. The CRC is correct, which will be shown in Section 7.5

TX_EN is active during the entire frame, and is deactivated after the last nibble is transmitted. Carrier sense becomes active after one nibble delay since the frame is started, and is deactivated when the frame is finished.

7.4.2 Throughput

The throughput of the 10 Mbps communication channel is not as high as expected. When taking into consideration the overhead of the frame structure, the half-duplex operation, the time delays between frames, and the possible collisions on the shared medium, the expected bit rate should be approximately 6-7 Mbps.

The data transfer rate between two PC's is approximately 3.6 Mbps. This is less than the expected value, but does indicate that the system works correctly. The

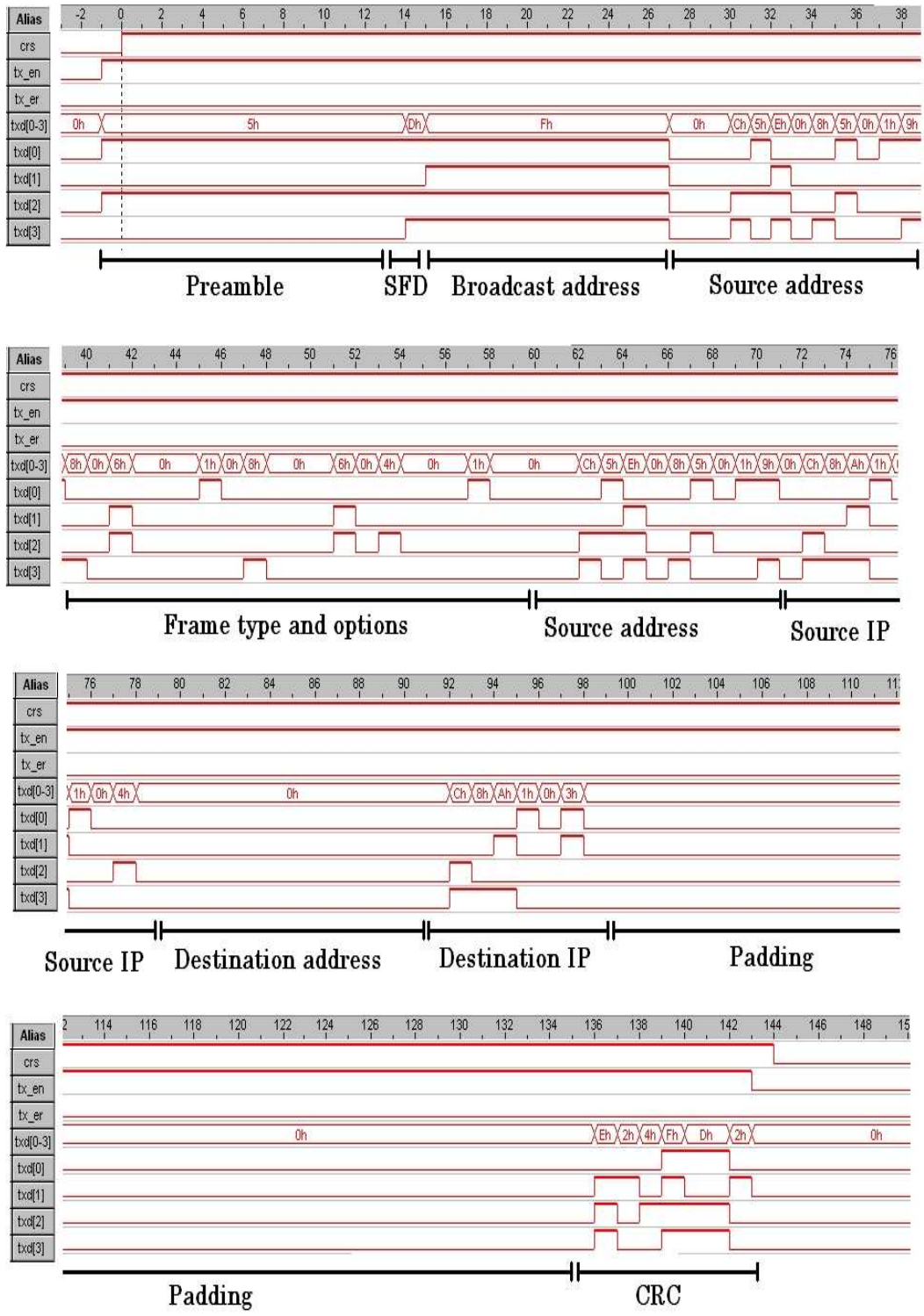


Figure 7.2: Decoded ARP frame

medium (direct connection) is idle for 50% of the time, so the maximum throughput for the system will be approximately twice the measured performance, or 7.2 Mbps.

Collisions occur during the communication, which can be expected on a half duplex

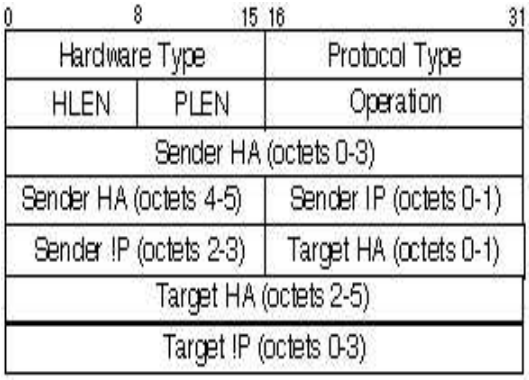


Figure 7.3: ARP header structure

link.

7.5 CRC Check

To test the quality of the transmission between the FPGA's, an error rate analysis must be performed. The analysis must include the entire system, starting at the data transmitted over the MII interface at one PC, through the source FPGA, to the destination FPGA, and finally transmitted over the destination MII interface to the destination PC. It is not feasible to perform a bit error rate analysis with the available bit error rate analyzer (BER), since the MII interface requires multiple signals to be generated simultaneously, which is not possible with the available analyzer. It is however possible to obtain a measure of the amount of dropped frames due to CRC check failures. This will be the basis of the error rate analysis.

7.5.1 MAC Layer

The MAC layer inside the PCILAN network card automatically discards frames which are not received correctly due to an invalid CRC check. When such a frame is received, the MAC layer will set a flag in a register indicating that an error has occurred, and drop the frame. The flag only indicates that an error has occurred, and not how many, or the size of the frame. It is therefore not suitable as a measure of the performance.

7.5.2 FPGA Statistics

The method chosen to test the performance of the system is to implement a CRC check routine inside the FPGA, and to gather statistics about the received frames. The statistics can be read with the MII management interface, and compared to data gathered with packet analyzer software on the PC. A packet analyzer will measure all transmitted and received frames, including ARP, ICMP and other protocols. The

amount of transmitted frames at one PC should ideally be equal to the amount of received frames at the other PC. When a frame becomes corrupted during transfer, the FPGA can determine that the CRC check is invalid, and the MAC frame at the destination PC will discard the frame. The packet analyzer at the destination will not see that a frame has been dropped.

While it is possible to test the general performance with only the packet analyzers on the PC, the extra statistics from the FPGA will improve the confidence of the analysis, since it is possible that the packet analyzers can miss certain frames. Also, the effect of collisions can not be determined from the packet analyzer data alone, since the MAC layer will autonomously retransmit frames which have collided.

7.5.3 CRC Implementation

The IEEE 802.3 [6] standard, clause 3.2.8, defines the algorithm of the CRC check which is included in each Ethernet frame. The standard does not provide a specific hardware-level implementation. An implementation of the CRC algorithm is first programmed in C, in order to confirm the correct implementation. The C program implements the CRC algorithm as a linear feedback shift register (LFSR), instead of as a long modulo-32 division. Figure 7.4 shows the gate-level configuration of the CRC check. The CRC check has been verified with the frame in Figure 7.2, and the program generates the same CRC.

The CRC generator polynomial is:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^0 \quad (7.5)$$

The first 32 bits of the frame are complemented. This is done by initializing all registers with '1'. Next, the contents of the frame are shifted into the CRC checker, starting at the broadcast address and continuing up to but excluding the CRC field. Each byte of the frame enters with the LSB first. When a padding field is present in the frame, this is also shifted through the CRC checker. The contents of the CRC checker, one clock after the last bit has entered, forms the CRC of the frame. The CRC is transmitted with the MSD first, starting at bit 31 down to bit 0.

7.6 100 Mbps Performance

With the 10 Mbps system functioning correctly, the system has been configured to operate on 100 Mbps. It is possible at 100 Mbps to send and receive ping packets, but full communication does not reliable work (yet) between two PC's. For a series of ping packets, approximately 94% are answered with a reply.

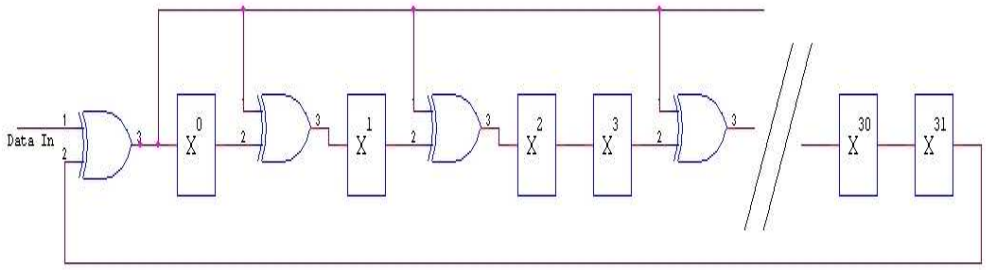


Figure 7.4: CRC gate level implementation

7.7 Conclusion

Communication between two FPGA's has been established at 10 Mbps, and works correctly. Also, partial communication has been established at 100 Mbps, but further testing is required.

Chapter 8

Digital Networking

8.1 Introduction

The communication between two modules has been established with the use of the MII management interface and the new level converter. The goal for the assignment is to communicate between three PC's. This requires new hardware to interface the communication between three FPGA's. The network is still digital, as the analog transceiver is not yet finished, but the network can be simulated with a digital counterpart. This chapter will investigate the required hardware for a digital network.

In order to produce a working three node network, a number of hardware components are required.

1. Third PC
2. Third FPGA
3. Simulated Network

8.2 Diginet

8.2.1 Overview

The high speed serial signals on the FPGA require LVDS inputs and outputs for proper operation. LVDS relies on balanced transmission lines in order to reduce common-mode interference to the digital signals. The output of one FPGA must be connected to the input of the other two. There are not enough inputs and outputs present on the FPGA, so a direct connection between the FPGA's is not possible. It is also not possible to split up or combine the LVDS signals without proper hardware. A new piece of hardware must be introduced, which simulates the shared medium and provides the correct LVDS terminations.

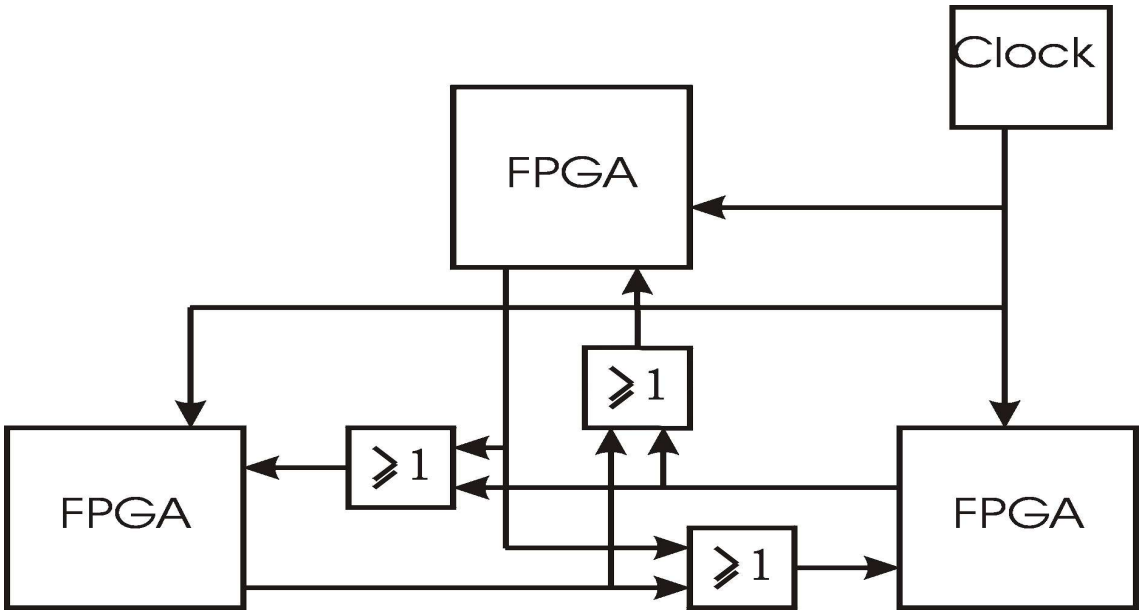


Figure 8.1: Diginet system overview

The goal of the new diginet print is to provide a test environment for the FPGA's, in order to test the performance of the digital part of the MOUSE system. The new hardware required to simulate a shared medium has been nicknamed diginet, in contrast to the optical shared medium which is the goal of the Mouse project.

8.2.2 Shared Medium Simulation

When multiple nodes on the optical cable are transmitting data simultaneously, the photo diode will detect that the power level is above a specified threshold, and will output a high level. The photo diode performs this operation independent of the number of transmitters. This is equivalent of the digital 'OR' function which the diginet print can use to simulate the shared medium.

A second requirement for the diginet is that a transmitter does not receive it's own data back through the network. In the MOUSE system, this is accomplished by using a passive optical splitter.

Figure 8.1 shows the intended system configuration. The $\geq$ symbol indicates the OR function. The system uses a single reference clock to keep all FPGA's synchronized.

8.2.3 Status Signals

The carrier sense signal from the analog board is not present in the digital network. It is possible to add a carrier sense signal, but it does not provide any advantages, since the carrier sense information can be generated inside the FPGA in response to data received from the digital network. With the digital network, the full preamble will be

received, which gives sufficient time to perform the required carrier sense operation.

The collision detect signal can also be generated inside the FPGA. The collision detect signal indicates that the FPGA is transmitting a frame, while at the same time data is being received from the digital network.

8.2.4 Clock Generator

The three FPGA's require three synchronized clocks to operate properly. This can be done with three separate synchronized clock generators, but it is decided to include a small clock generator with three outputs on the new board. This only takes a small amount of board space.

8.2.5 Board Layout

The diginet print will be designed on a dual layer PCB. Both the clock generator and the diginet design are build on the same physical PCB, and can be separated afterwards if required. Therefore both designs need their own power supply.

The layout of the board must be considered carefully to provide optimal performance of the system. The signal tracess should be as short as possible, in order to keep the delays low. One thing which is important with LVDS signals is the differential delay. Both traces of an LVDS signal should run close together, and also have the same length.

8.2.6 Trace Impedance

The LVDS traces require a characteristic impedance Z_0 of 50 ohm. The material used for the PCB is FR4 with $\epsilon_r = 4.7$. The height of the microstrip line above the ground plane is 1.6 mm (thickness of the PCB).

To obtain an impedance of 50 ohm, the width to separation ratio W/S must be 2, according to Matick [7], page 325. With a saperation of 1.6 mm for the double sided PCB, the stripline must be 3.2 mm wide. A trace of 3.2 mm does not fit between the pins of an SMA connector. The choice was made for a 2.0 mm trace, which does fit between the pins of the SMA connector. This smaller trace results in a higher trace impedance of 66 ohm.

The LVDS trace pairs are routed close together in order to reduce differential delay (skew) and noise pickup.

8.2.7 Features

The new diginet pcb will provide the following features:

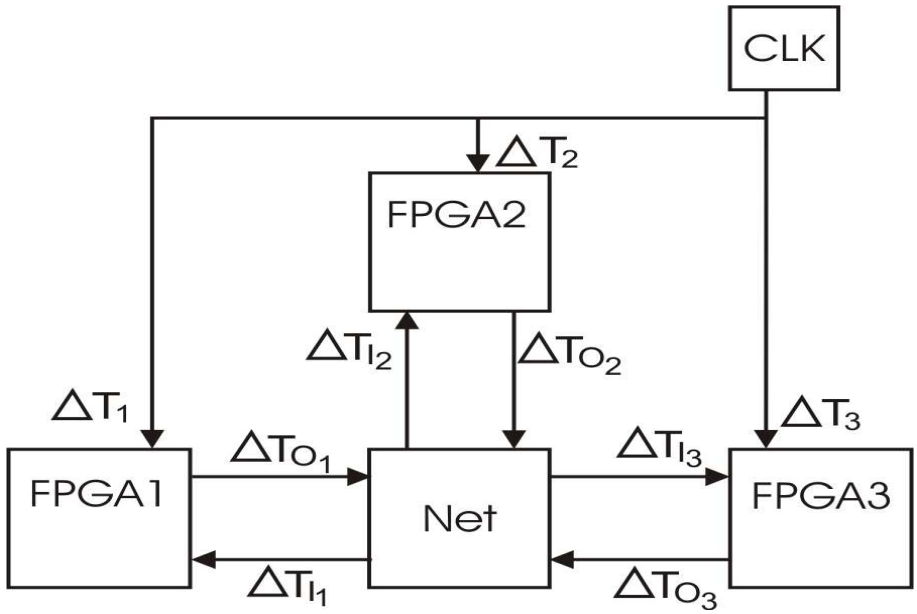


Figure 8.2: Diginet system timing

1. A synchronized 125 MHz clock generator with three outputs
2. Two separate logic paths with different IC types.
3. Signaling leds to indicate traffic over the network.
4. Low skew signal distribution for LVDS signals

8.2.8 Delays

Figure 8.2 shows the intended setup for the diginet system with the timing delays. The delay from the reference clock to the different FPGA's is assumed equal due to the same cable length and type, and therefore ΔT_1, ΔT_2 and ΔT_3 can be eliminated from the calculation.

NP indicates an integer number N of full clock periods P. The delay ΔT_N symbolizes the delay through the new print. This delay will be equal for all 3 FPGA's. In order for the data to arrive at the three FPGA's synchronized, the following set of equations must be valid:

$$\begin{aligned}
\Delta T_{F_{O_1}} + \Delta T_N + \Delta T_{F_{I_2}} &= NP \\
\Delta T_{F_{O_1}} + \Delta T_N + \Delta T_{F_{I_3}} &= NP \\
\Delta T_{F_{O_2}} + \Delta T_N + \Delta T_{F_{I_1}} &= NP \\
\Delta T_{F_{O_2}} + \Delta T_N + \Delta T_{F_{I_3}} &= NP \\
\Delta T_{F_{O_3}} + \Delta T_N + \Delta T_{F_{I_1}} &= NP \\
\Delta T_{F_{O_3}} + \Delta T_N + \Delta T_{F_{I_2}} &= NP
\end{aligned} \tag{8.1}$$

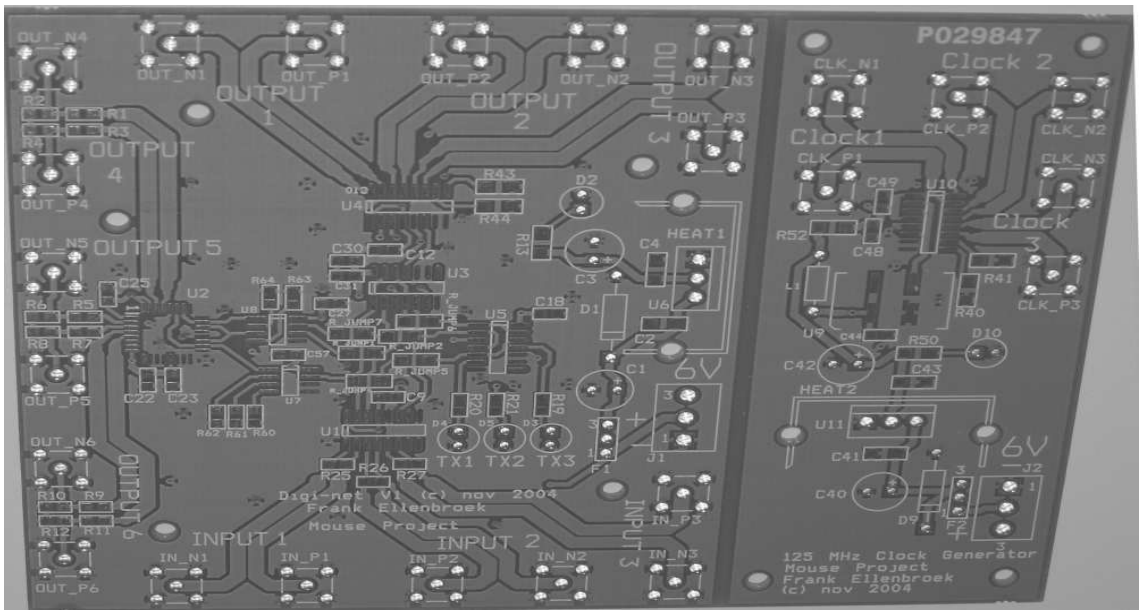


Figure 8.3: Diginet PCB

There are multiple solutions to this set of equations. If we take all input lines $\Delta T_{F_{I_x}}$ equal (ΔT_{F_I}), and all outputs $\Delta T_{F_{O_x}}$ equal (ΔT_{F_O}), the system can be simplified and the following equation is obtained.

$$\Delta T_F = NP - \Delta T_N - \Delta T_{F_I} \tag{8.2}$$

From Equation 8.2 we can see, that the delay can be compensated by choosing the correct line lengths for $\Delta T_{F_{I_x}}$ and $\Delta T_{F_{O_x}}$.

8.3 Diginet PCB

The diginet PCB has been designed and ordered. The schematics and layout plots are presented in Appendix C. Figure 8.3 shows the completed PCB, and Figure 8.4 shows the diginet PCB with most components soldered on.

The clock generator works correctly, and is capable of providing a 125 MHz clock to all three FPGA's in the system.

The functional performance of the diginet PCB is correct, the inputs from multiple FPGA's are correctly combined to the output. The FPGA's have not been connected to the diginet PCB.

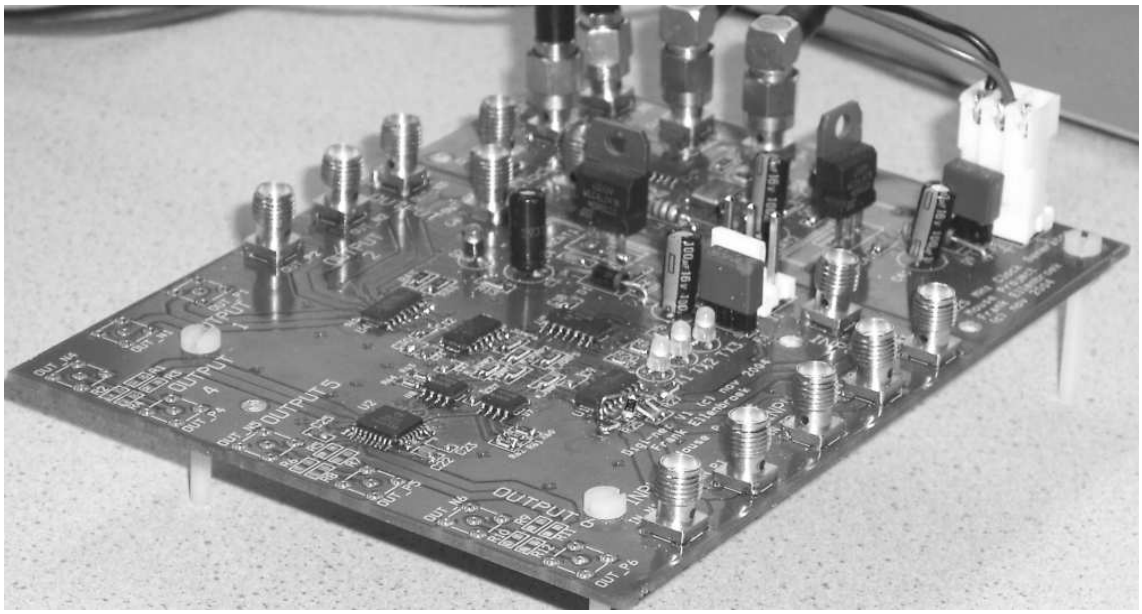


Figure 8.4: Diginet PCB with components

Chapter 9

Conclusions and Recommendations

9.1 Conclusions

1. A new level converter has been designed and tested, which works correctly.
2. A MII management interface has been implemented in the FPGA, which is capable of correctly and automatically initializing the MOUSE hardware. Through this interface, the configuration and status of the FPGA can be determined.
3. The communication between two nodes at 10 Mbps has been established, and communication works correctly.
4. The communication between two nodes at 100 Mbps has been partially established, communication works correctly for small packets.
5. A CRC check routine has been implemented in the FPGA, which is usefull in gathering statistical information about the performance of the system.
6. A digital simulation network has been designed, and the functional performance is correct at 100 Mbps.

9.2 Recommendations

1. The digital part of the system has to be tested for 100 Mbps with 3 nodes. The hardware is ready and should work correctly, but it will require some time to determine the correct cable lengths and configuration settings in the FPGA.
2. The analog board has to be tested, and a redesign is required for the board. A number of problems have been identified, which will impact the performance of the system. Someone with a firm grasp of High Freqency design and preferrably Electromagnetic Compatibility will be required to correctly design the new board.

3. Proper measurement equipment should be available continuously when measuring and testing the analog board, and for the 100 Mbps communication between three nodes. It is therefore recommended to purchase a decent (4 Gs/s 4 channel) digital sampling oscilloscope *before* starting the follow-up assignments. Proper measurements require at least two good probe sets to be available.

References

[1] Igor Radovanović,
Optical Local Area Networks: New solutions for fiber-to-the-desk applications,
Ph.D. Thesis, Twente University Press, Enschede, The Netherlands, 2003

[2] R. Klein Kiskamp,
Mouse, the digital part, Internal document,
Telecommunication Engineering chair, University of Twente, The Netherlands, 2002

[3] ISO/IEC 7498-1:1994,
Open Systems Interconnection Reference model

[4] Clayton R. Paul,
Introduction to Electromagnetic Compatibility,
John Wiley & Sons, inc. 1992

[5] IEEE Std 1149.1-1990,
IEEE Standard Test Access Port and Boundary-Scan Architecture,
Institute of Electrical and Electronic Engineering Press, 1990

[6] IEEE Std 802.3-2002,
Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access method and physical layer specification,
Institute of Electrical and Electronic Engineering Press, 1998

[7] Richard E. Matick,
Transmission Lines for Digital and Communication Networks,
McGraw-Hill, New York, 1969

[8] F.A.Benson, T.M.Benson,
Fields, Waves and Fransmission Lines,
Chapman & Hall, London, 1991

[9] Howard W. Johnson, Martin Graham,
High-Speed Digital Design, A Handbook of Black Magic,
Prentice Hall, New Jersey, 1993

[10] Robert O. Taniman,
Digital Part Implementation of a Transceiver for Optical LAN,
Individual research assignment, Telecommunication Engineering chair, University of Twente, The Netherlands, 2002

[11] Christian Dolea,
Optical LAN Transceiver: Design and Realization,
Master assignment, Telecommunication Engineering chair, University of Twente, The Netherlands, 2004

Appendix A

Reflections

A.1 Introduction

One of the problems with the old level converter is the absence of termination resistors. In this appendix, an investigation will be made to determine the best termination options.

The situation to model is as follows. One IC is transmitting data through a ribbon cable to another IC. The clock frequency is 25 MHz, therefore a data bit is 40 ns long. The system is measured with a probe of 10 pF/1 Mohm internal impedance. IC's are connected on both sides of the cable, with will have a typical input capacitance between 5 pF and 15 pF. I have chosen 15pF on both sides as the capacitance to use, which includes the probe capacitance.

The ribbon cable is a standard 40-wire ribbon cable, as used in IDE computer cables. This cable configuration has an approximate impedance of 100 ohm. See for example Johnson and Graham [9], page 325.

The output impedance of the transmitting port is not specified in the datasheets, but because the typical output impedance of TTL gates is between 20 ohm and 40 ohm, a value of 30 ohm is used.

The rise and fall times of the IC's are not fully specified. Measurements from the level converter configuration indicate that the rise should be less than 4 ns, for example 1 ns. Actually, since there are 2 flanks (rising and falling), the second flank will have a faster rise time of 400 ps.

We can model the system as a transmission line (TL) problem with a characteristic impedance of 100 ohm.

The speed of light in a medium is:

$$v_r = \frac{v_0}{\sqrt{\epsilon_r \mu_r}}$$

For most coaxial cables, the dielectric medium is teflon, with an ϵ_r of 2.1 and μ_r of

unity. The speed of the electromagnetic waves in the cable is therefore approximately 2/3 of the speed of light in vacuum. The ribbon cable used for the measurements is 17 cm long, excluding connectors, and with the assumption that the signals travel through the cable at approximately 2/3 of the speed of light (20 cm/ns) (which is a reasonable approximation, see Johnson [9]), the electrical length of the cable will be in the order of 1 ns.

A.2 Theoretical Investigation

Transmission line theory [7], [4] will be used to predict the effect of terminations on the transmission line. The system to investigate is depicted in Figure A.1. The transmission line is excited by a step source $1(t)$ of V_0 volt. v_r is the speed of the electromagnetic wave in the medium. The length of the transmission line is L.

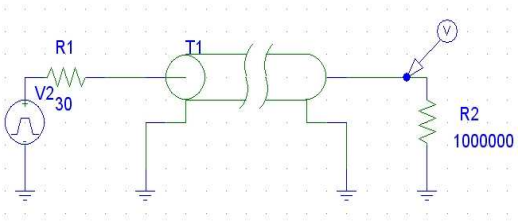


Figure A.1: Basic Transmission Line Model

At $t = 0$, the source will change its output from 0 to V_0. This voltage is then divided between the output resistor of the source, and the characteristic impedance of the transmission line. This will generate a forward travelling wave $x(t)$. Up until the time that the wave reaches the end of the transmission line, the voltage at each point x will be:

$$0 \leq t < \frac{L}{v_r} \qquad V(x,t) = \begin{cases} V_0 \frac{Z_0}{Z_0+Z_S}, & x < v_r t, \\ 0, & x > v_r t \end{cases} \tag{A.1}$$

At the end of the line, the wave will reflect on the termination impedance Z_L, which will create a reflected wave of $V_0 \frac{Z_L - Z_0}{Z_0+Z_L}$. This wave is added to the forward travelling wave, and will result in the following line voltages:

$$L_{v_r} \leq t < 2L_{v_r} \qquad V(x,t) = \begin{cases} V_0(\frac{Z_L-Z_0}{Z_L+Z_0}\frac{Z_0}{Z_0+Z_S} + \frac{Z_0}{Z_0+Z_S}), & x > v_r t - L_{v_r}, \\ V_0 \frac{Z_0}{Z_0+Z_S}, & x < v_r t - L_{v_r} \end{cases} \tag{A.2}$$

where L_{v_r} is the delay over the line $\frac{L}{v_r}$.
If we denote $\Gamma_L = \frac{Z_L - Z_0}{Z_L+Z_0}$ and $V_i = V_0 \frac{Z_0}{Z_0+Z_S}$ then we can write Equation (A.2) as:

$$L_{v_r} \leq t < 2L_{v_r} \qquad V(x,t) = \begin{cases} V_i(1+\Gamma_L), & x > v_r t - L_{v_r}, \\ V_i, & x < v_r t - L_{v_r} \end{cases} \tag{A.3}$$

When the reflected wave reaches the beginning of the line, there will be a reflection $\Gamma_S = \frac{Z_S - Z_0}{Z_S + Z_0}$ which will create a new forward travelling wave. The voltage over the line will be as described in Equation (A.4):

$$2L_{v_r} \leq t < 3L_{v_r} \qquad V(x,t) = \begin{cases} V_i(1+\Gamma_L+\Gamma_L\Gamma_S), & x < v_r t - 2L_{v_r}, \\ V_i(1+\Gamma_L), & x > v_r t - 2L_{v_r} \end{cases} \tag{A.4}$$

The reflections will continue on infinitely. The limit for the line voltage can be calculated as follows:

$$V_\infty = V_i(1+\Gamma_L+\Gamma_L\Gamma_S+\Gamma_L^2\Gamma_S+\Gamma_L^2\Gamma_S^2+\cdots) \tag{A.5}$$

Or, written as a sum:

$$V_\infty = V_i\left(\Gamma_L\sum_{n=0}^{\infty}(\Gamma_L\Gamma_S)^n + \sum_{n=0}^{\infty}(\Gamma_L\Gamma_S)^n\right) \tag{A.6}$$

Since $|\Gamma_L\Gamma_S| < 1$ the sum converges to

$$V_\infty = V_i\left(\frac{\Gamma_L}{1-\Gamma_L\Gamma_S} + \frac{1}{1-\Gamma_L\Gamma_S}\right) = V_i\left(\frac{1+\Gamma_L}{1-\Gamma_L\Gamma_S}\right) \tag{A.7}$$

A.2.1 Examples

As an example, we will calculate the voltage at the end of the line when the source and load resistors are equal. In this case, the voltage will be equally divided between the source and load resistors, so the load value will be $\frac{1}{2}V_0$.

All resistances equal

If we take $Z_S = Z_L = Z_0 = 50\Omega$ and $V_0 = 1$, then the following calculation can be made:

$$V_i = V_0\frac{Z_0}{Z_0+Z_S} = \frac{50}{50+50} = \frac{1}{2} \tag{A.8}$$

$$\Gamma_S = \frac{Z_S - Z_0}{Z_S + Z_0} = \frac{50-50}{50+50} = 0 \tag{A.9}$$

$$\Gamma_L = \frac{Z_L - Z_0}{Z_L + Z_0} = \frac{50-50}{50+50} = 0 \tag{A.10}$$

$$V_\infty = V_i\frac{1+\Gamma_L}{1-\Gamma_L\Gamma_S} = \frac{1}{2}\ \frac{1+0}{1-0} = \frac{1}{2} \tag{A.11}$$

The output is therefore as expected.

Source and load resistor equal

If we take $Z_S = Z_L = 50\Omega$, $Z_0 = 100\Omega$ and $V_0 = 1$, then we would still expect that the final voltage would be $\frac{1}{2}$, but this time there are reflections.

$$V_i = V_0 \frac{Z_0}{Z_0 + Z_S} = \frac{100}{100 + 50} = \frac{2}{3} \tag{A.12}$$

$$\Gamma_S = \frac{Z_S - Z_0}{Z_S + Z_0} = \frac{50 - 100}{50 + 100} = -\frac{1}{3} \tag{A.13}$$

$$\Gamma_L = \frac{Z_L - Z_0}{Z_L + Z_0} = \frac{50 - 100}{50 + 100} = -\frac{1}{3} \tag{A.14}$$

$$V_\infty = V_i \frac{1 + \Gamma_L}{1 - \Gamma_L \Gamma_S} = \frac{2}{3} \frac{1 - \frac{1}{3}}{1 - \frac{1}{3}\frac{1}{3}} = \frac{2}{3} \frac{\frac{6}{9}}{\frac{8}{9}} = \frac{1}{2} \tag{A.15}$$

The final voltage is again correct.

A.3 Unterminated Line

For the transnsmission line system, a number of simulations have been performed to test the validity of the previously derived results. The simulation is done with Pspice.

Source	Voltage	Rise time	Fall time	Pulse width	Period	TL delay	TL Z_0
Pulse	3V3	400 ps	1 ns	40 ns	120 ns	1 ns	100

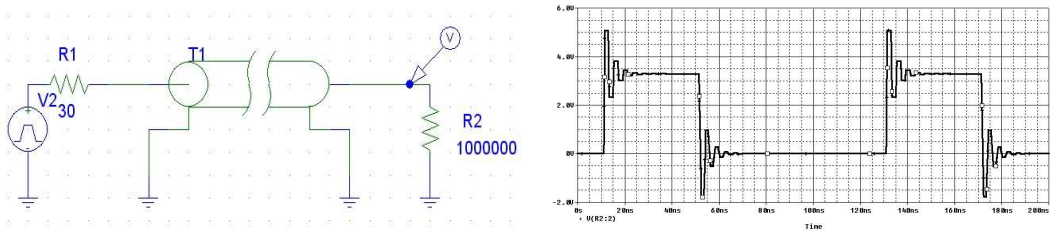


Figure A.2: System response of unterminated line

The addition of parasitic capacitances will give more realistic results, see Figure A.3.

A.4 Series Terminated Line

When the transmission line is series terminated with the characteristic impedance of the line, the overshoot should be completely removed. Figure A.4 shows the resulting signal.

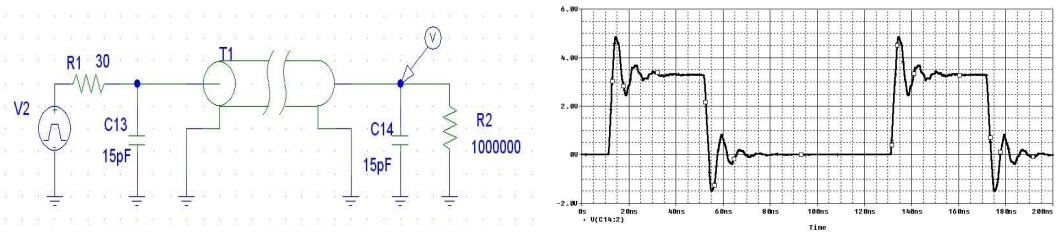


Figure A.3: System response of unterminated line with capacities

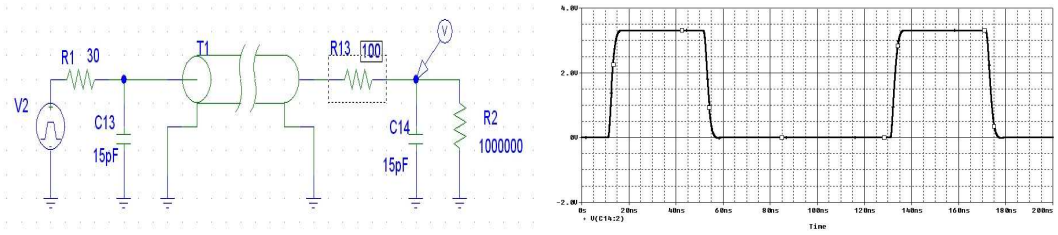


Figure A.4: System response of series terminated line

The reflections are eliminated completely, as expected.

If the termination resistance is smaller than the characteristic impedance, there will still be overshoot, but the amplitude of the overshoot will be smaller than the unterminated case. Figure A.5 shows the simulation with a 60 Ω resistor.

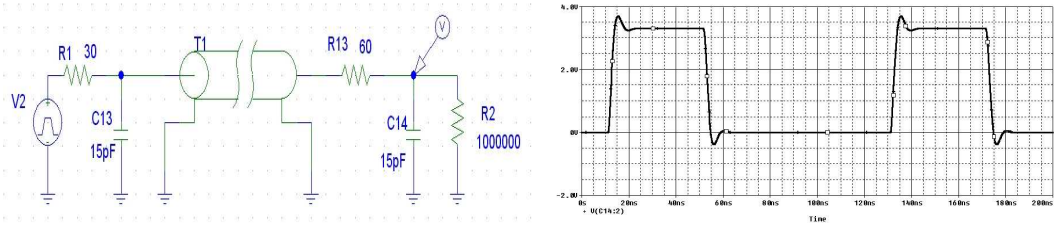


Figure A.5: System response of series terminated line with 60 Ω

Appendix B

Level converter 2 design

This appendix includes the schematics and layout designs for the new level converter.

B.1 List of Components

SMD Resistors & Capacitors:

```
Size: 0805              Layout Package: SM/C_0805 & SM/R_0805
Size: 1206              Layout Package: SM/C_1206 & SM/R_1206
Size: 1812              Layout Package: SM/C_1812

Size: 47u               Layout Package: CYL/D.225/LS.125/.031
Farnell OC: 771739      CAPACITOR 47UF 25V
```

LEDs

```
Layout Package: CPCYL/D.275/LS.125/.034
Farnell OC: 178293      LED_3MM RED
Farnell OC: 322465      LED_3MM YELLOW
Farnell OC: 322477      LED_3MM GREEN
```

Diode

```
Type: 1n4001            Layout Package: DAX1/1N_4001-4007
Farnell OC: 251677      DIODE_1A 50V
```

FUSE

```
Value: 500mA            Layout Package: BLKCON.100/VH/TM1SQ/W.100/3
Farnell OC: 529382      FUSE_TE5 QUICK BLOW 500MA
```

Connectors

```
Type: HEADER 20X          Layout Package: WALCON.100/VH/TM20E/W.200/40
Farnell OC: 3167094       HEADER_STRT 40WAY

Type: 4 Way IDE 4-H       Layout Package: Custom
Farnell OC: 148086        HEADER_4WAY:

Type: MII DIN 40_ABC      Layout Package: DCON.050/VS/TM/40
Farnell OC: 965558        SOCKET_SCSI PCB STR 68W

Passthrough header 2x20 Layout Package: WALCON.100/VH/TM20E/W.200/40
Farnell OC: 359907        SOCKET_PC104 THROUGH 40WAY
```

Switch

```
Type: SW SPST/SM          Layout Package: Custom
Farnell OC: 733647        SWITCH_SLIDE SP 2 POS VERT
```

Integrated Circuits

```
Type: 74LVX3245/SO        Layout Package: SOG.050/24/WG.420/L.600
Farnell OC: 642435        IC_SM 74LVX LOGIC

Type: 74VHC125            Layout Package: SOG.050/14/WG.275/L.450
Farnell OC: 676135        IC_SM 74VHC CMOS LOGIC
```

Voltage Regulators

```
Type L7805                Layout Package: TO220AB
Farnell OC: 412594        IC_REGULATOR +5.0V

Type: LD1117              Layout Package: TO220AB
Farnell OC: 352-9976      IC_REGULATOR LDO +3.3V
```

B.2 Schematics

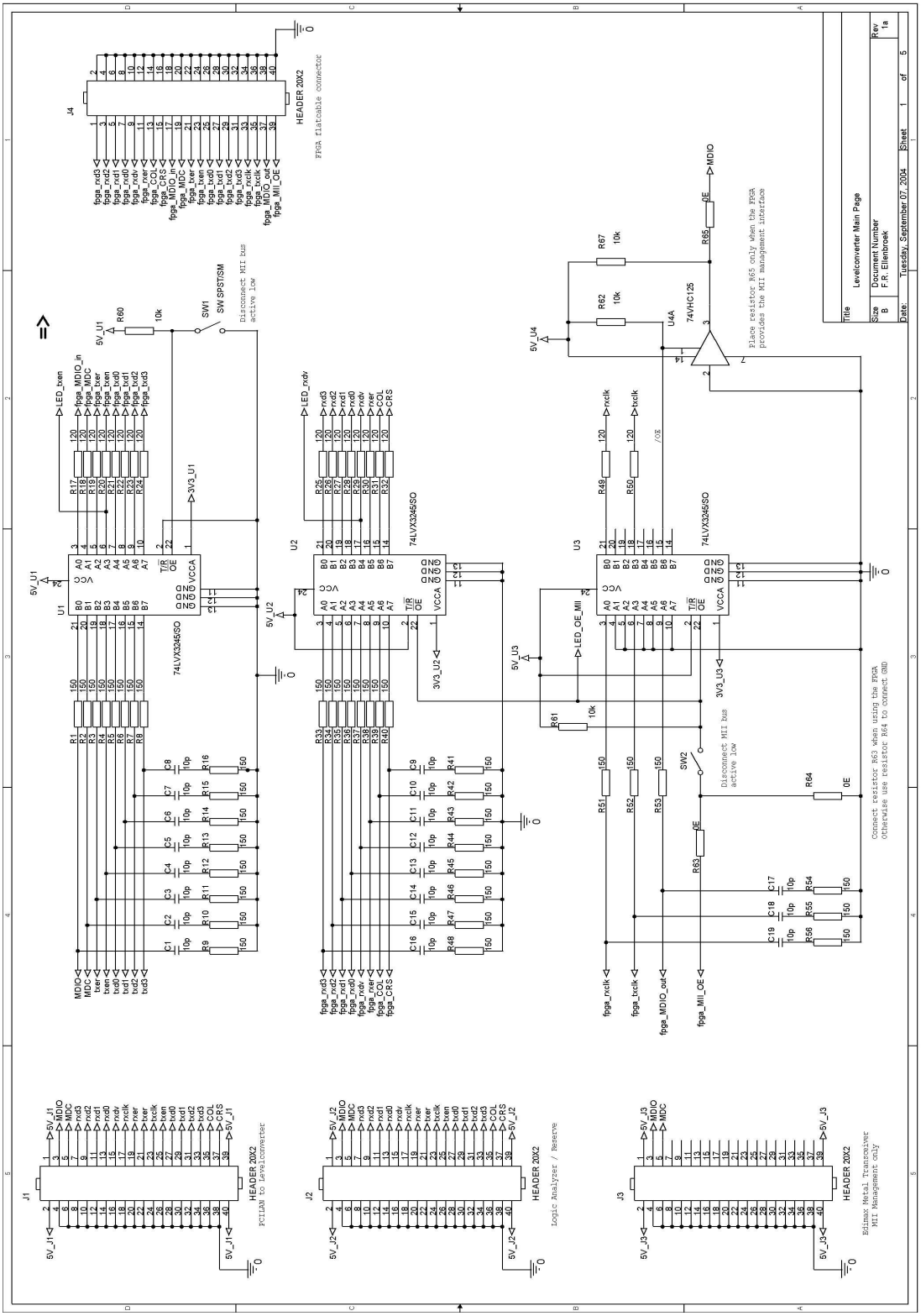


Figure B.1: Level converter 2 main schematic

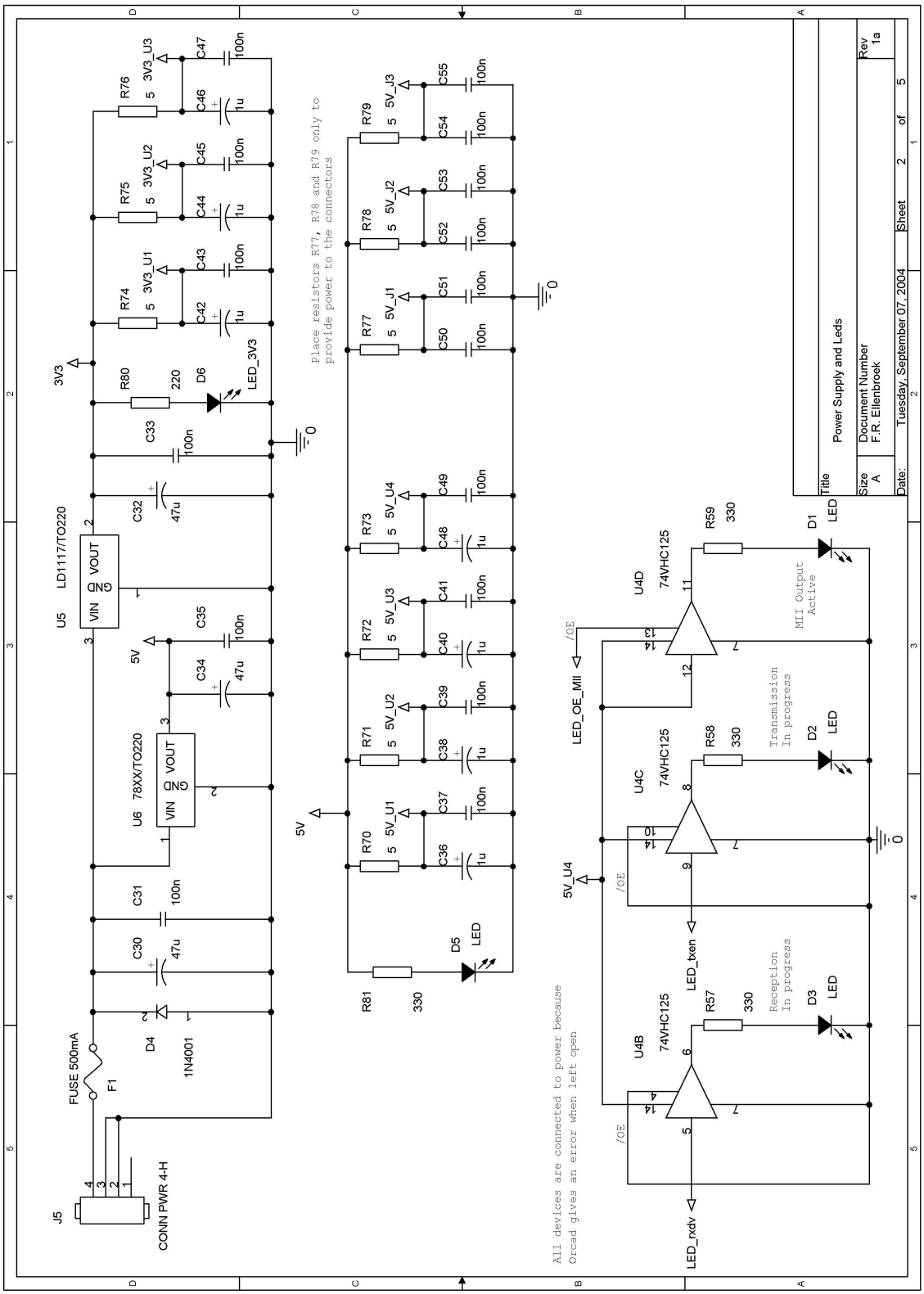


Figure B.2: Level converter 2 power supply

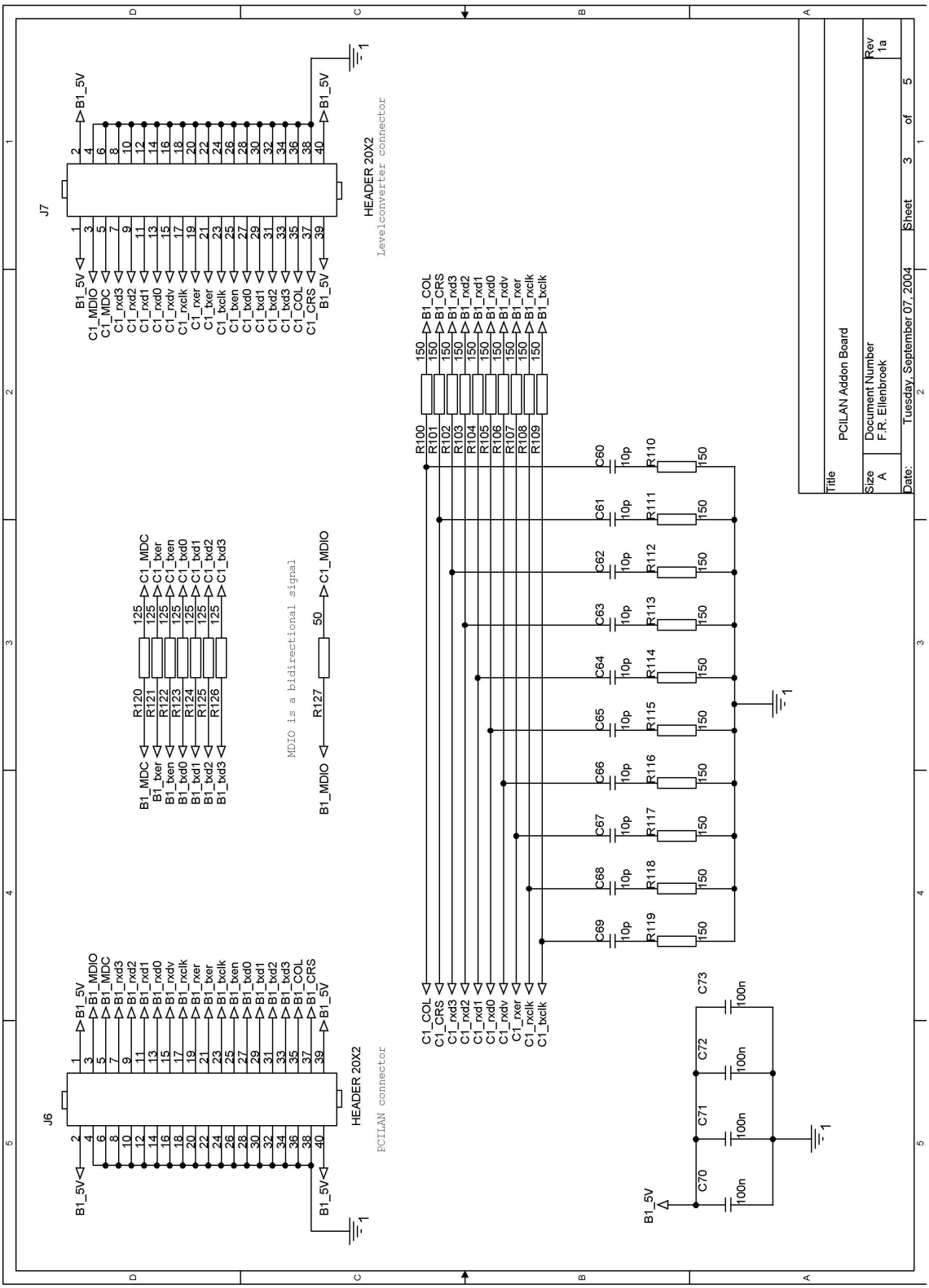


Figure B.3: Level converter 2 PCILAN add-on board

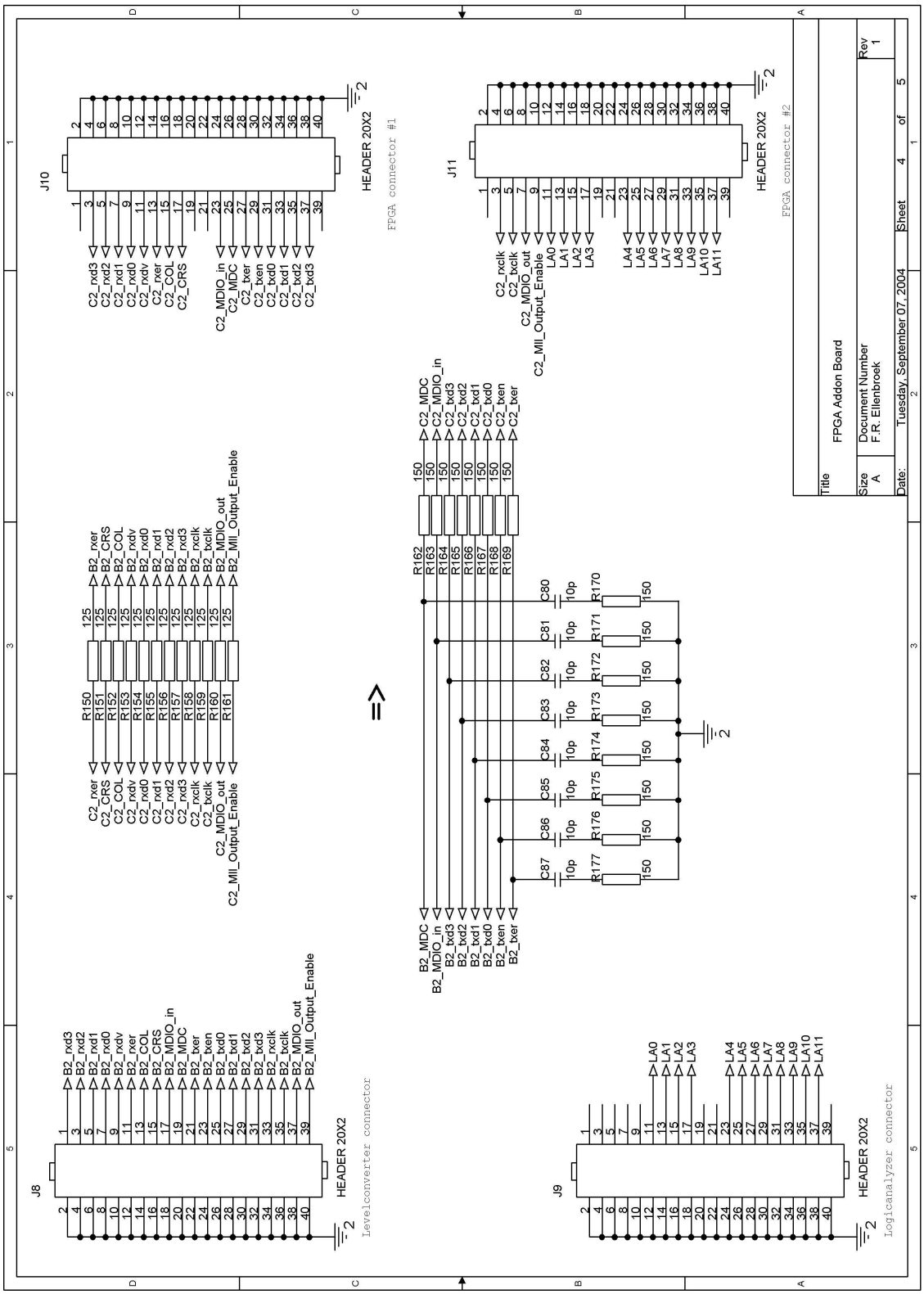


Figure B.4: Level converter 2 FPGA add-on board

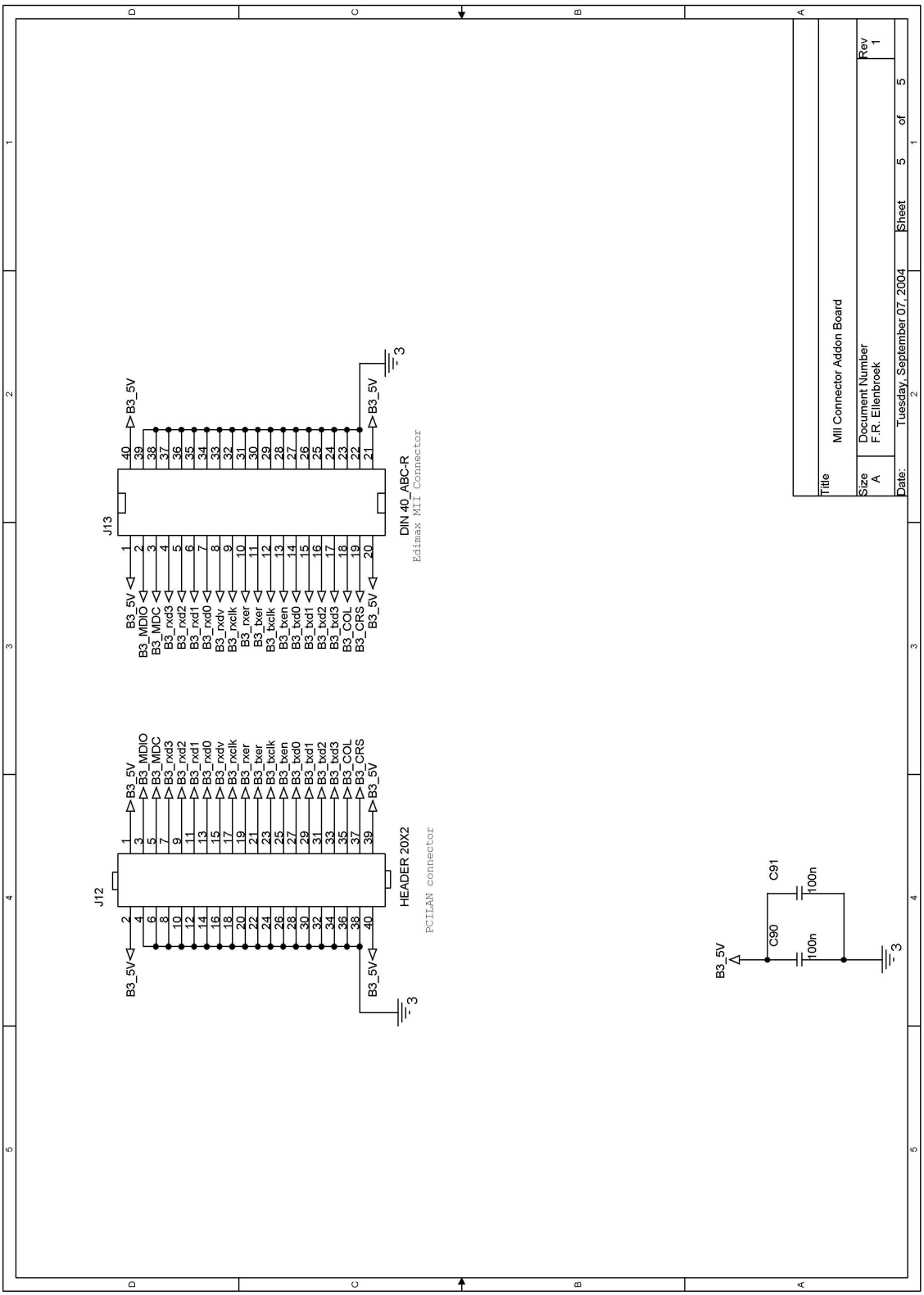


Figure B.5: Level converter 2 Edimax add-on board

B.3 PCB design

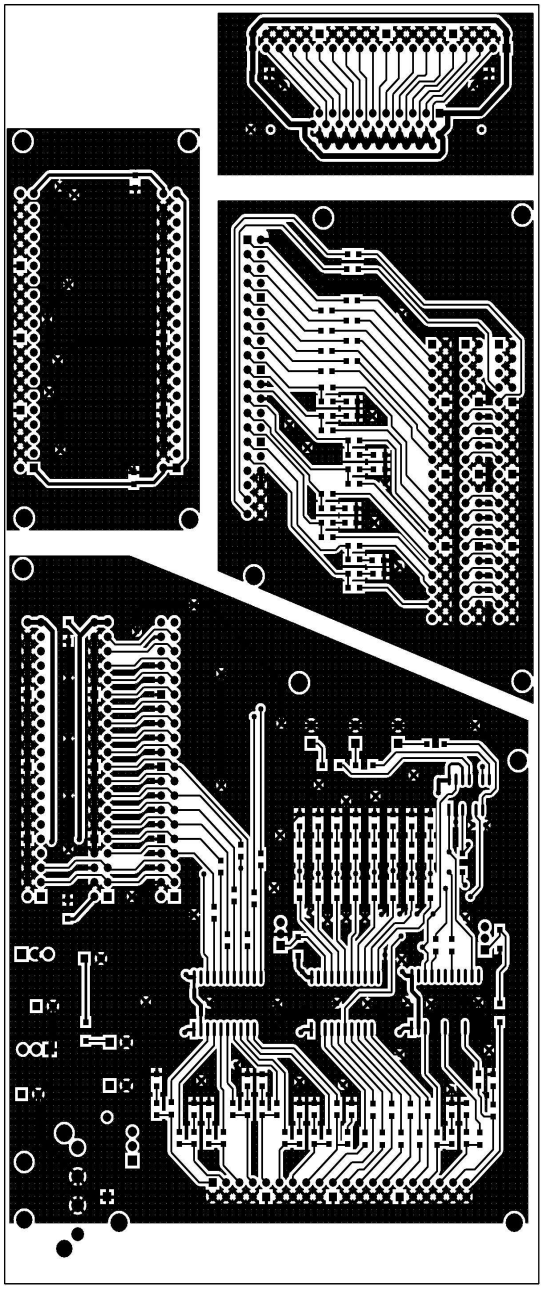


Figure B.6: Level converter 2 PCB top side

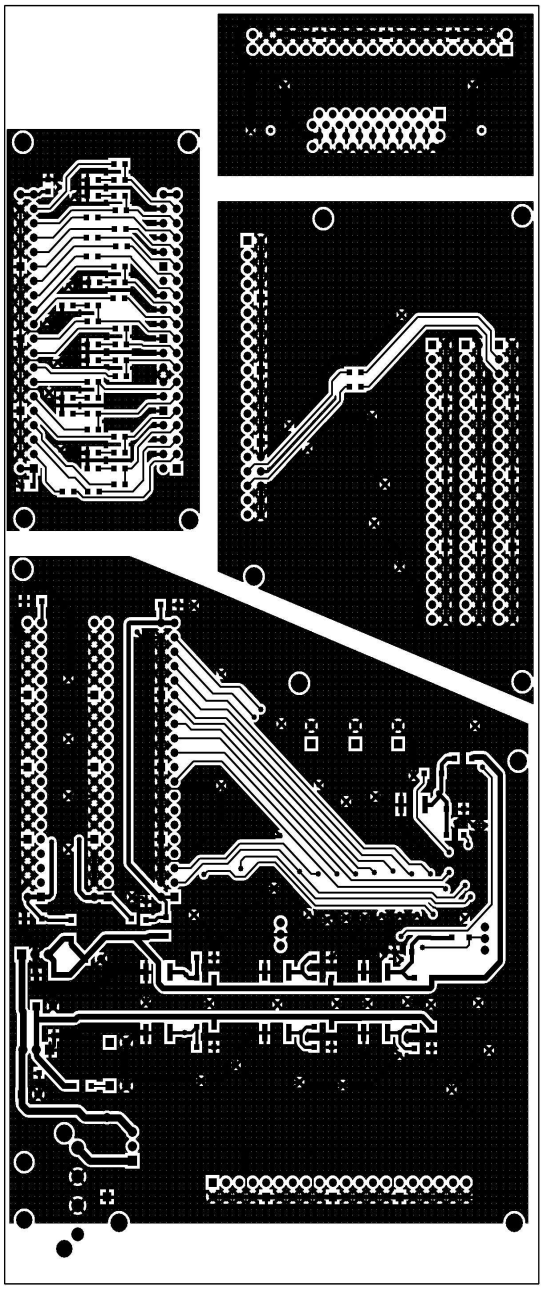


Figure B.7: Level converter 2 PCB bottom side

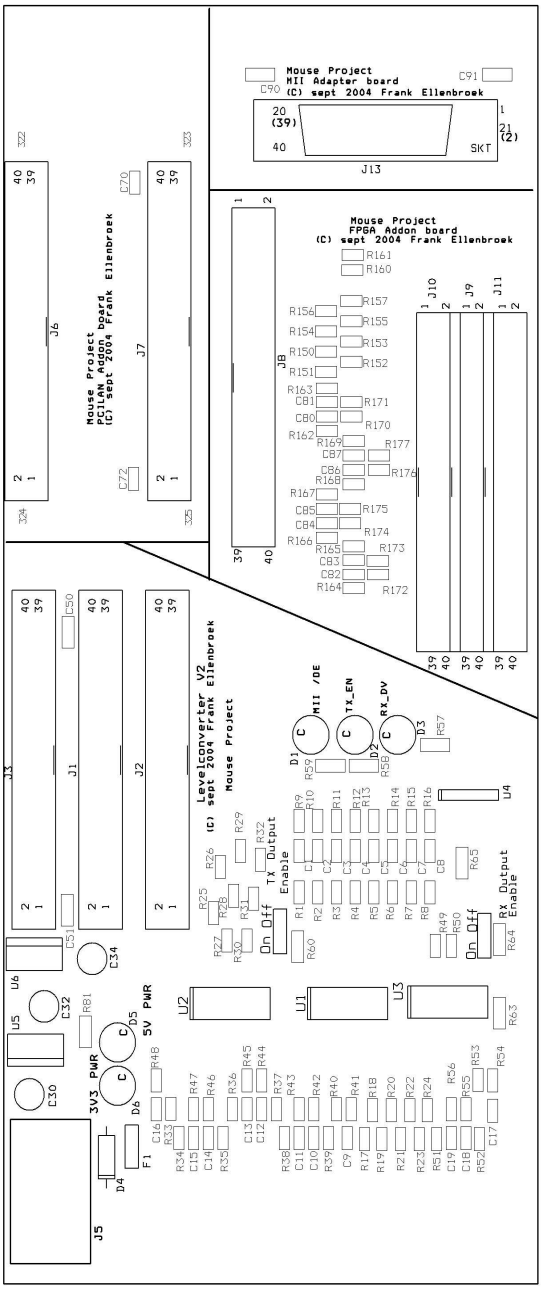


Figure B.8: Level converter 2 components top

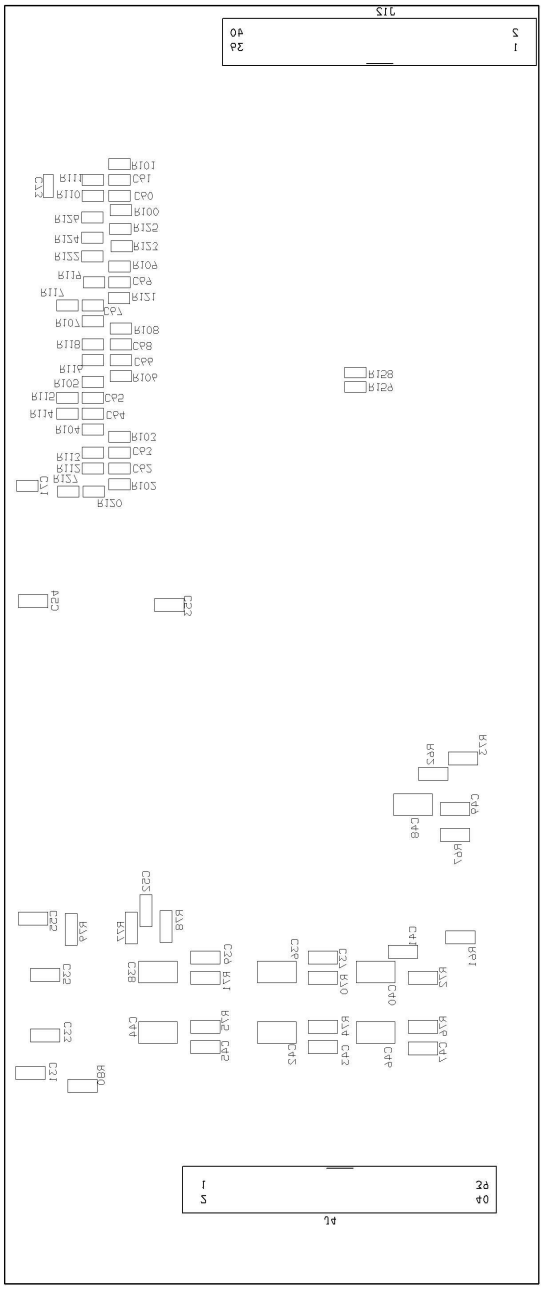


Figure B.9: Level converter 2 components bottom

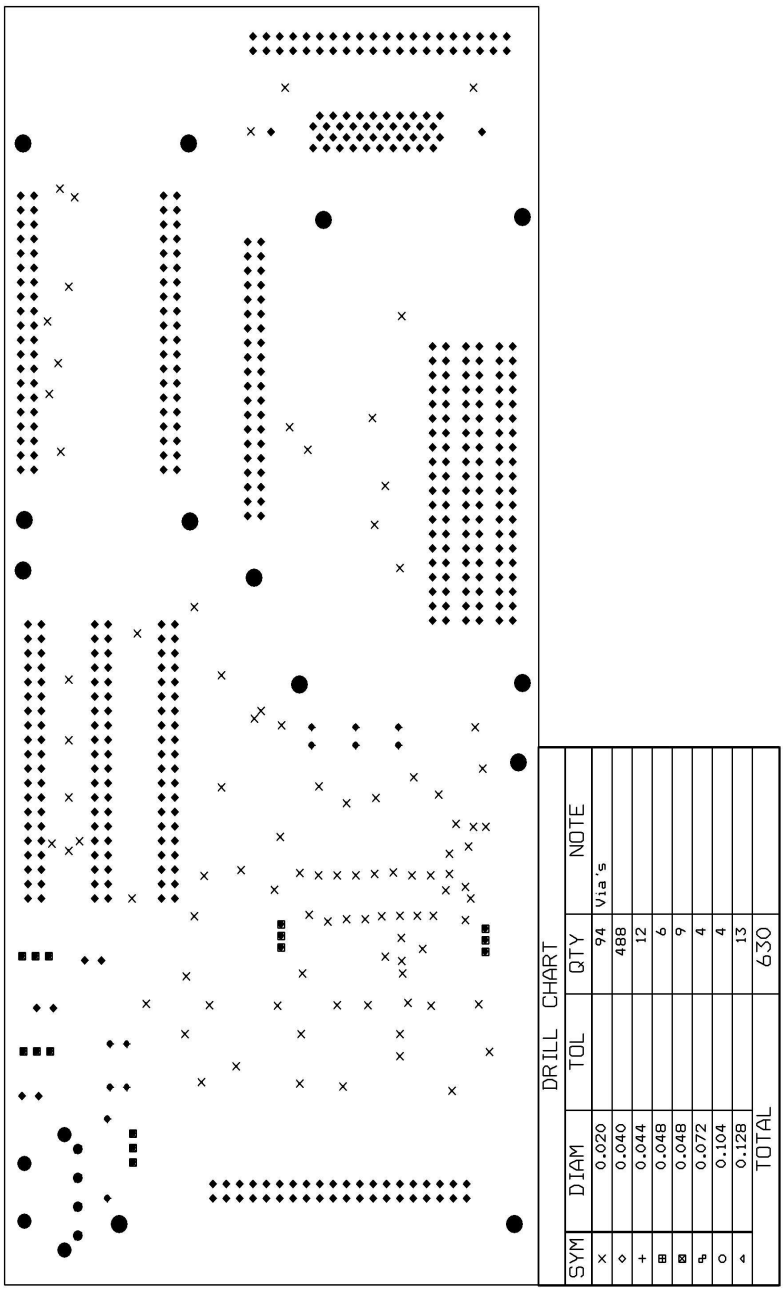


Figure B.10: Level converter 2 drill chart

Appendix C

Diginet design

This appendix includes the schematics and layout designs for the diginet pcb.

C.1 List of Components

SMD Resistors & Capacitors:

```
Lots of resistors/capacitors etc.

Heatsink
Leds
Fuse
Power connector
Inductor
Diodes

DS90LV048
DS90LV047
74ALVC32
74ALVT32
LD1117

mc100EPT22
mc10EP101

125 MHz Osc.
```

C.2 Schematics

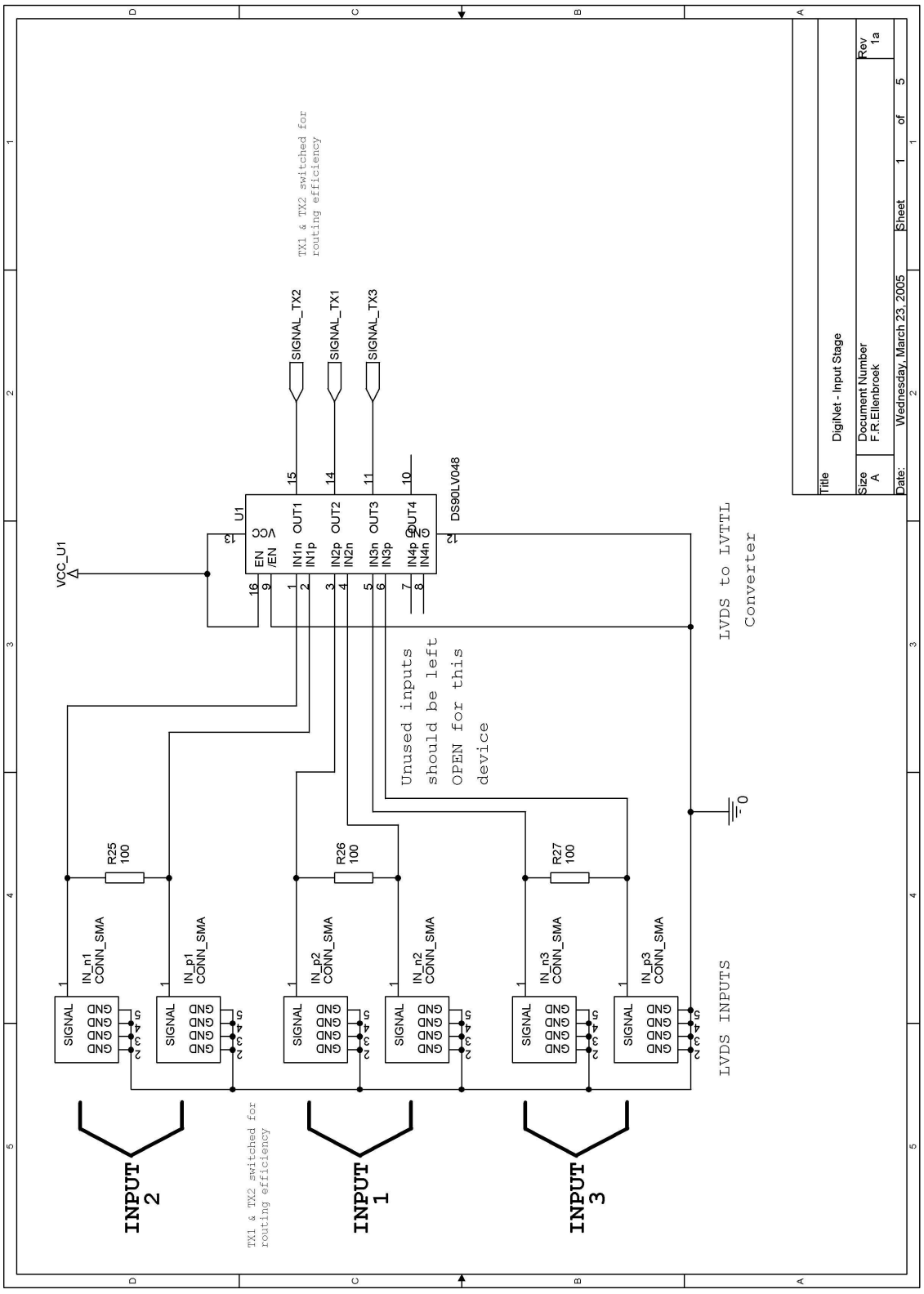


Figure C.1: Diginet LVDS input stage

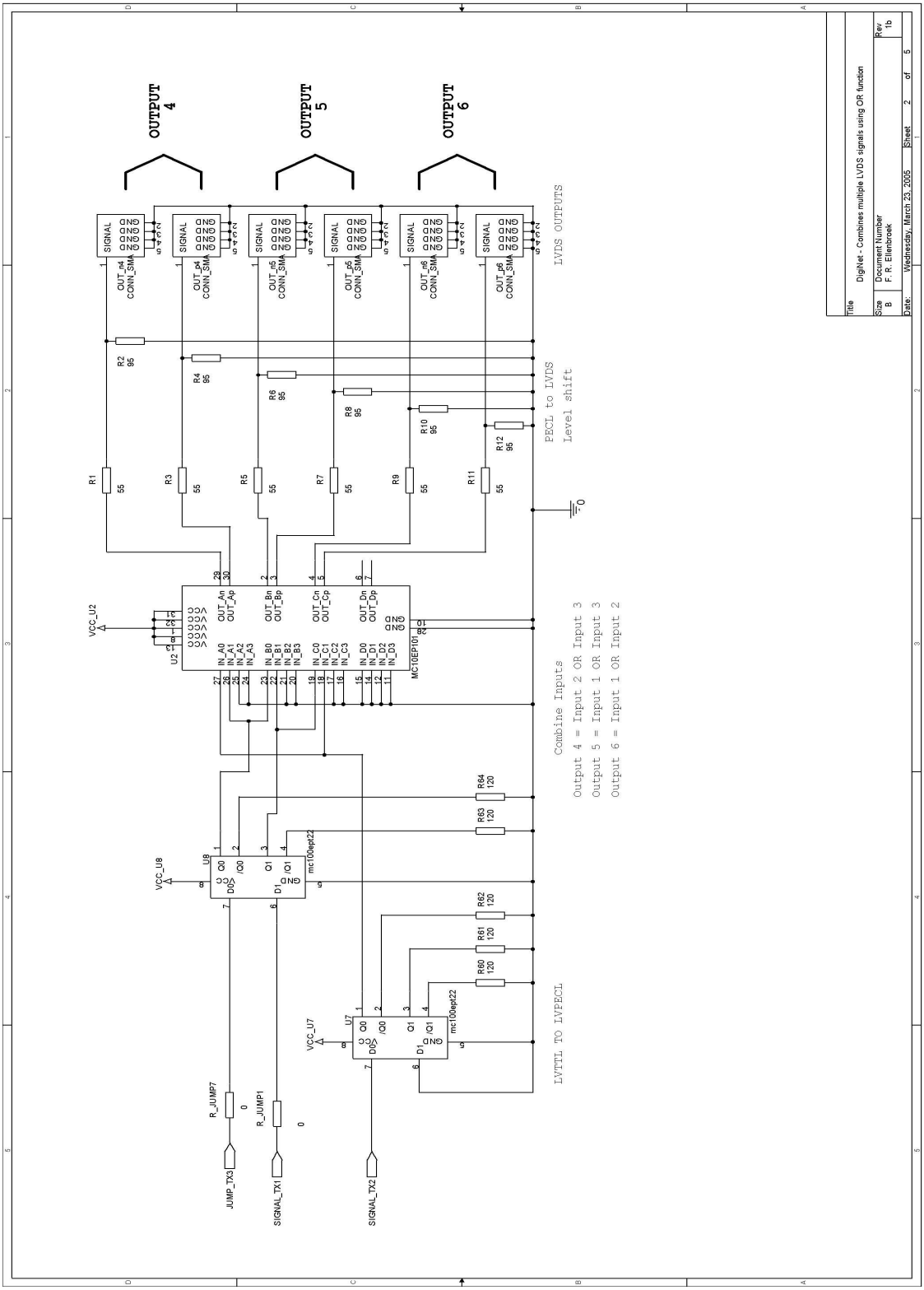


Figure C.2: Diginet logic option 1

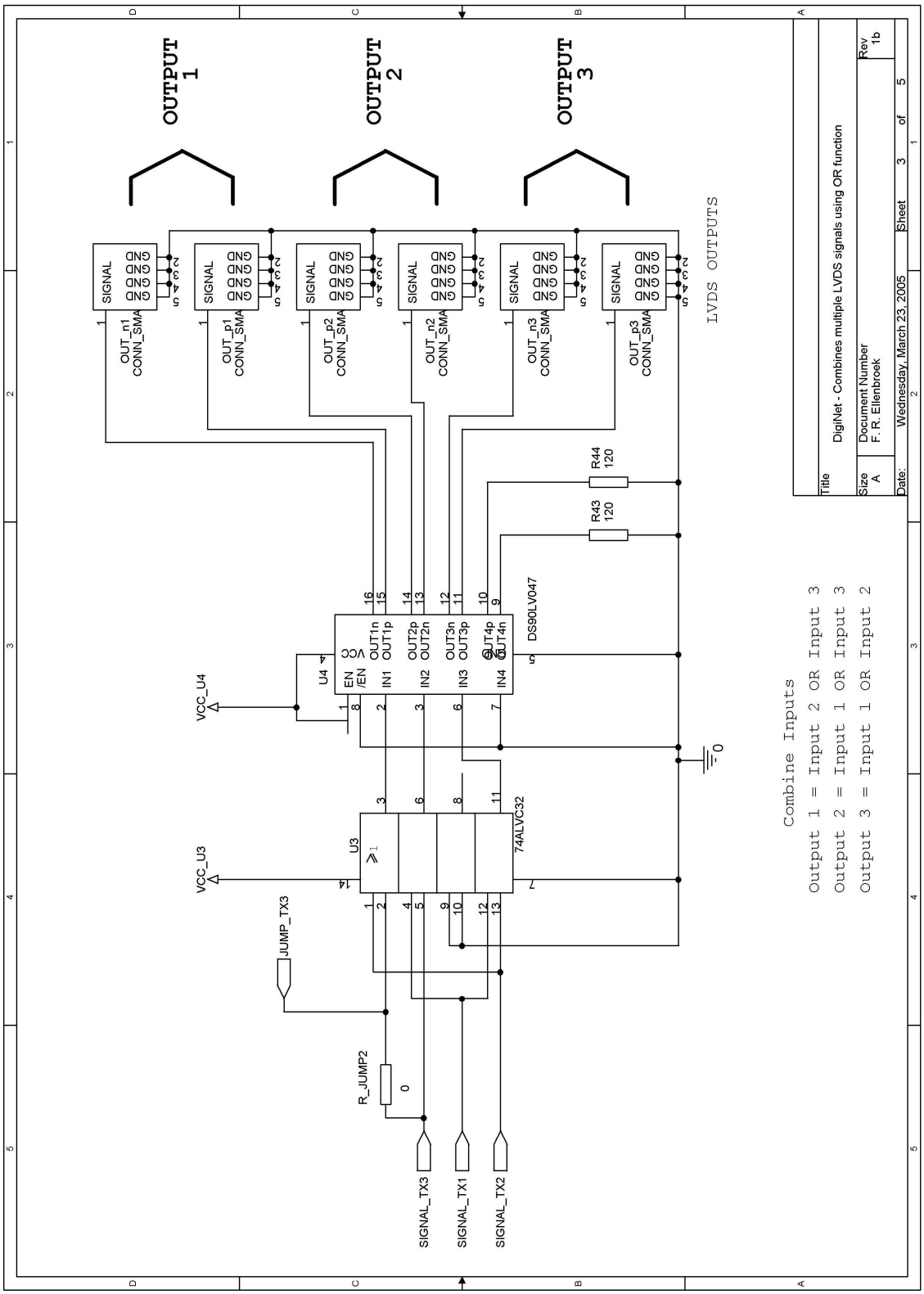


Figure C.3: Diginet logic option 2

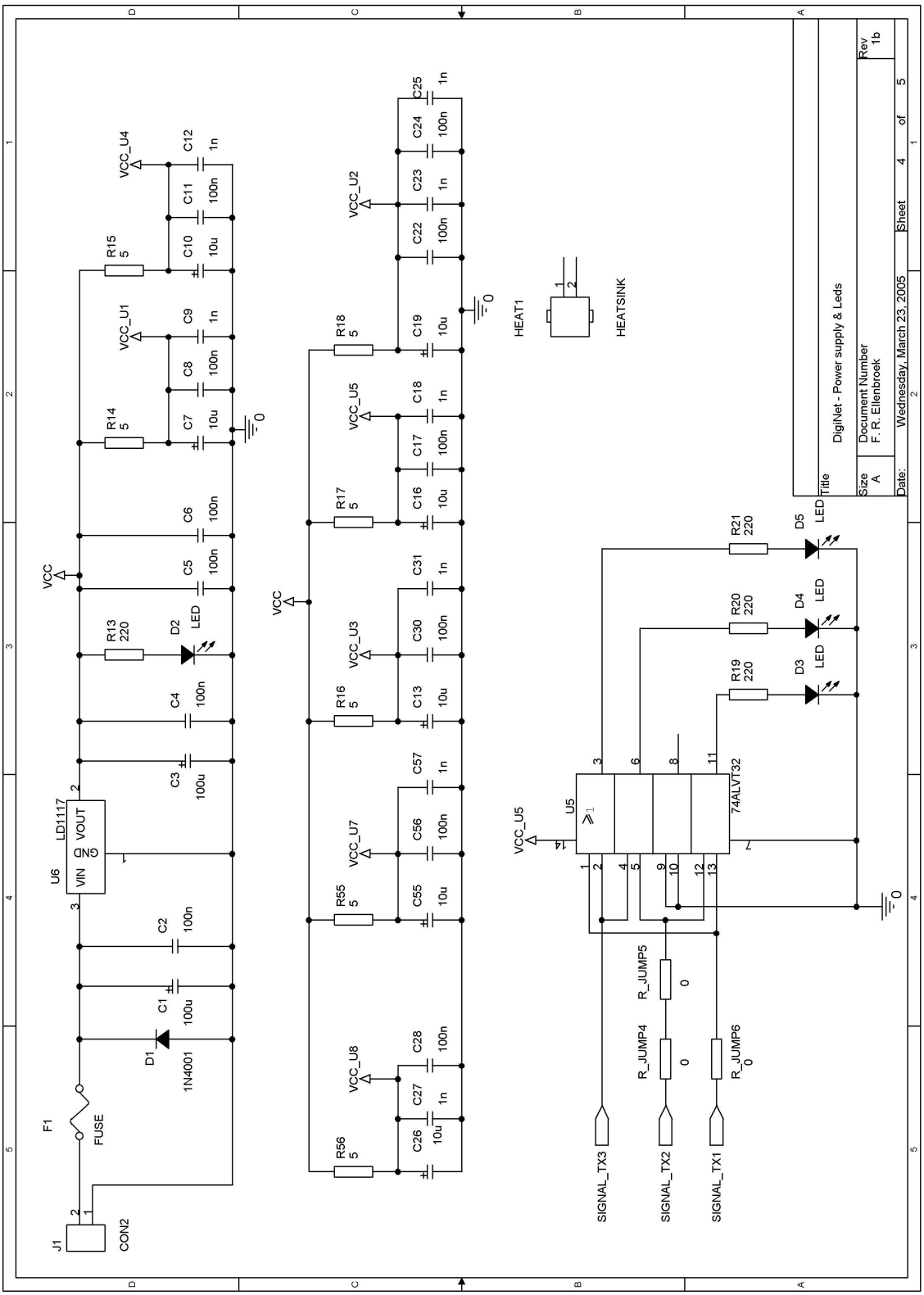


Figure C.4: Diginet power supply & leds

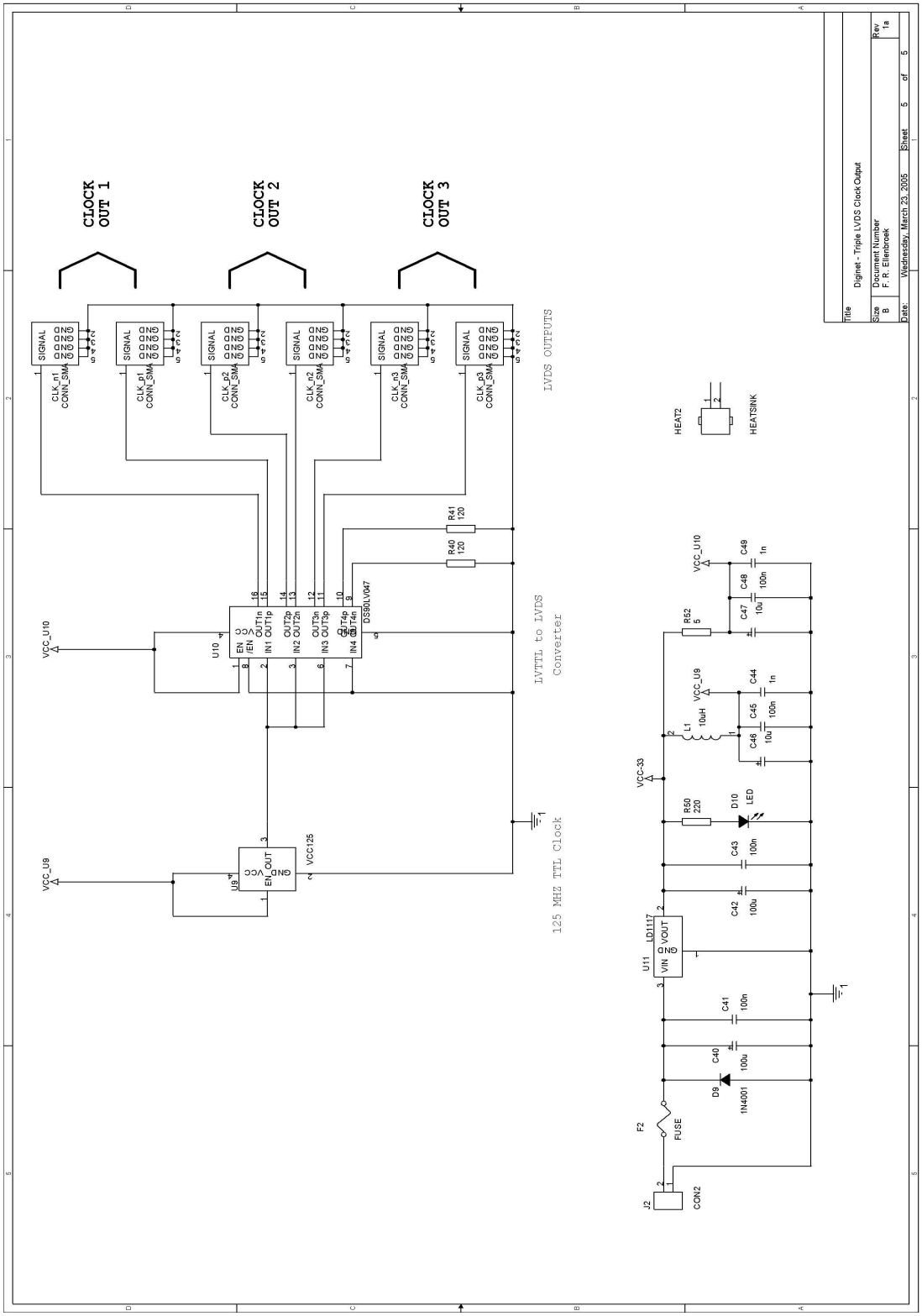


Figure C.5: Diginet Triple LVDS clock output

C.3 PCB design

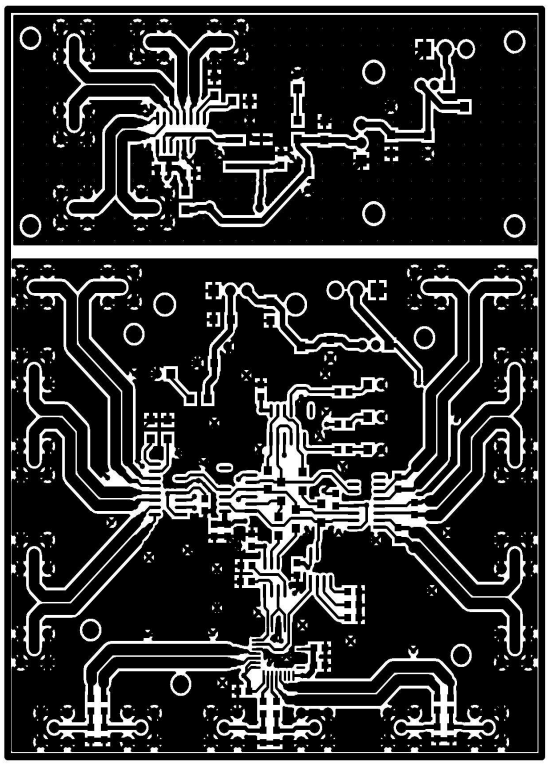


Figure C.6: Diginet PCB top side

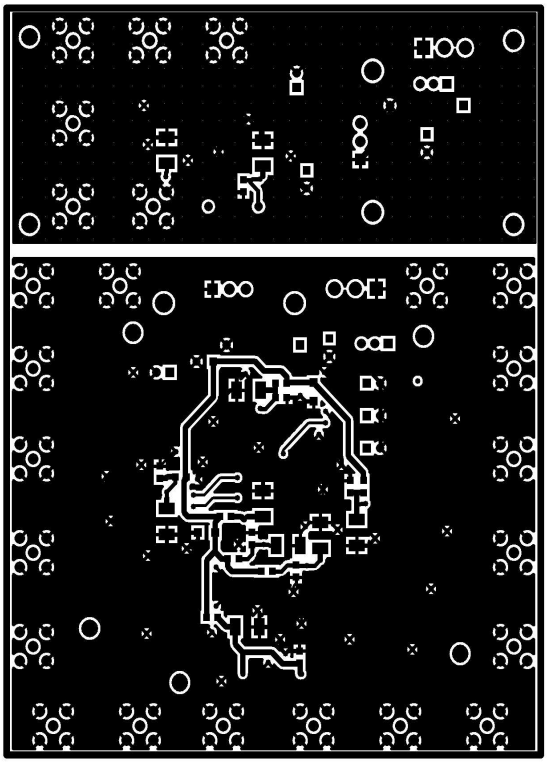


Figure C.7: Diginet 2 PCB bottom side

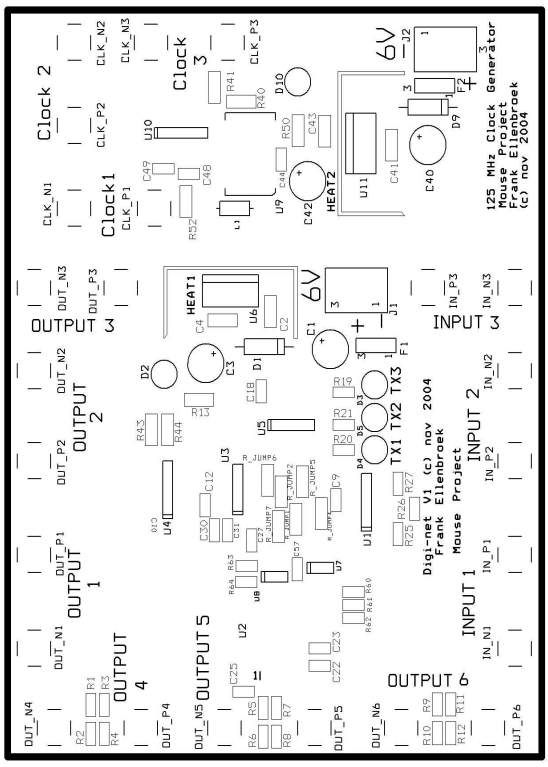


Figure C.8: Diginet components top

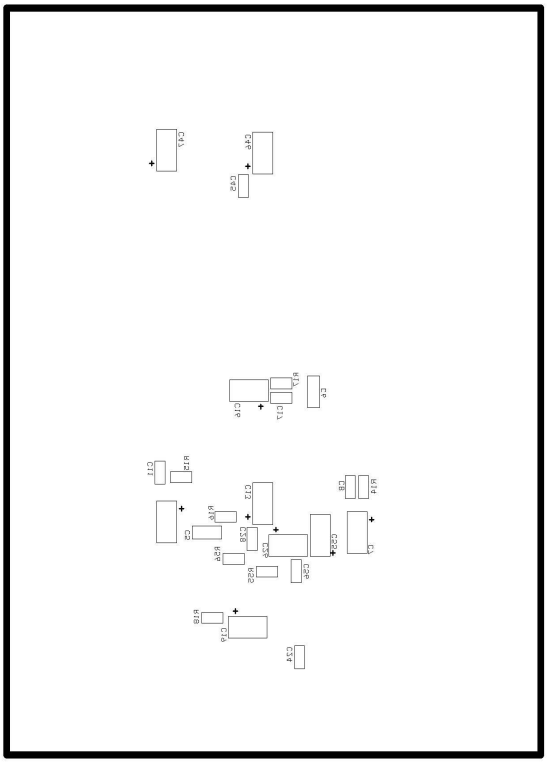


Figure C.9: Diginet components bottom

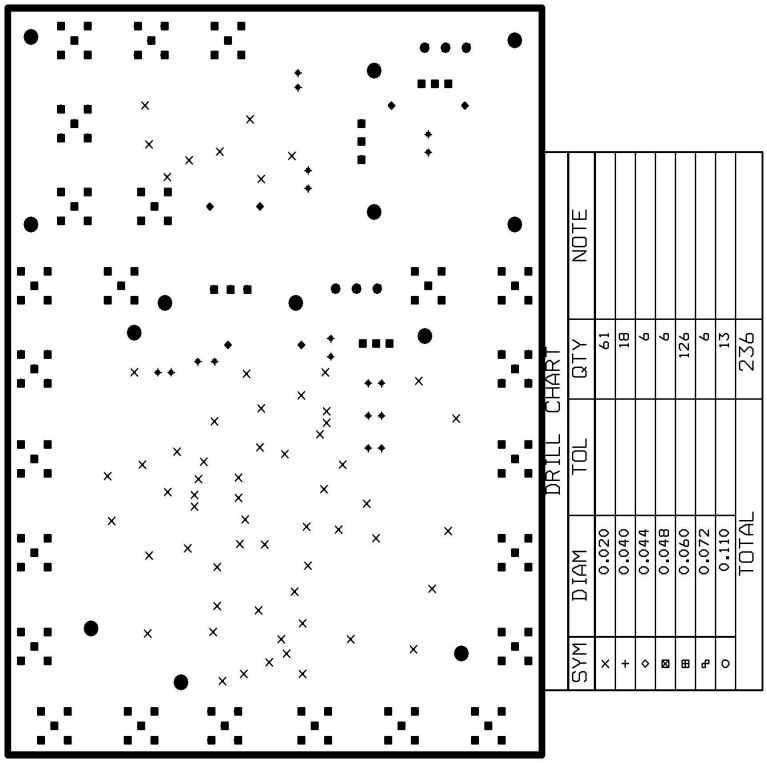


Figure C.10: Levelconverter 2 drill chart

Appendix D

VHDL code - MII Management Interface

This appendix contains a part of the VHDL source code for the MII management interface: The full source code will be provided on a CD.
The md_management module is the main entity in the VHDL design. All other mii management components are included here. It is mainly provided here as an example.

```
-- md_management
-- This entity is the topmost part of the MII management implementation. All sub
-- components are initiated from here.
-- The input clock should run at 25 MHZ (a derivative of tx_clk would be fine)
--


library ieee;
use ieee.std_logic_1164.all;

-->> The entity definition provides the external interface of the component

entity md_management is
   port(
      clock            : in     std_logic; -- synchronous 25 MHz master clock
      reset            : in     std_logic; -- asynchronous reset
      md_clock_in      : in     std_logic; -- asynchronous 2.5 MHz management clock
      md_data_in       : in     std_logic; -- asynchronous data input
      md_data_out      : out    std_logic; -- synchronous data output

      -- status bit inputs ----

      jabberdetect_in  : in     std_logic; -- status bit input
      linkstatus_in    : in     std_logic; -- status bit input
      remotefault_in   : in     std_logic; -- status bit input
```

```
      -- internal registers

      loopback         : out    std_logic; -- loopback 0 mode
      isolate          : out    std_logic; -- isolate MII bus
      collision_test   : out    std_logic; -- loopback 1 mode
      mii_tx_delay     : out    std_logic_vector(3 downto 0); -- delay before
                                                              -- data valid
      bypass_nrzi      : out    std_logic; -- internal bypass bit
      bypass_scramble  : out    std_logic; -- internal bypass bit
      md_busy          : out    std_logic
   );
end md_management;

architecture structure of md_management is

-->> first all external components must be declared

component md_counter
   port (
      md_clock, reset     : in std_logic; -- clock input
      md_load_op, md_load_d : in std_logic; -- (re)load bitcount
      md_shift_enable       : out std_logic -- enable shiftregister
);
end component;

component md_regmux
   port(
      clock              : in std_logic; -- clock inputs
      reset              : in std_logic; -- reset
      sr_in              : in std_logic_vector(15 downto 0); -- parallel data input
      md_store_register  : in std_logic; -- load data into MII register
      md_select_mux      : in natural range 0 to 31;
      md_clear_register  : in std_logic;
      sr_out             : out std_logic_vector(15 downto 0); -- parallel data input

      -- configuration registers
      jabberdetect_in         :     in std_logic;
      linkstatus_in           :     in std_logic;
      remotefault_in          :     in std_logic;
      md_invalid_operation_in :     in std_logic;
      md_wrong_address_in     :     in std_logic;
      mdc_timeout_in          :     in std_logic;
      md_sw_reset             :     in std_logic;

      -- counter signals
      col_in                  :     in std_logic; -- collision count
```

```
      crc_in                 :     in std_logic; -- crc error count
      tx_in                  :     in std_logic; -- transmit packet count
      rx_in                  :     in std_logic; -- receive packet count

      loopback               :     out std_logic;
      power_down             :     out std_logic;
      isolate                :     out std_logic;
      collision_test         :     out std_logic;

      mii_tx_delay           :     out std_logic_vector(3 downto 0);
      bypass_nrzi            :     out std_logic;
      bypass_scramble        :     out std_logic

);
end component;

component md_shiftregister
port (
      md_clock, reset    : in std_logic;  -- clock input
      md_load_reg        : in std_logic;  -- (re)load bitcount
      md_shift_enable    : in std_logic;  -- enable shiftregister
      mdio_in            : in std_logic;  -- input data
      mdio_out           : out std_logic; -- outputdata
      sr_in              : in std_logic_vector(15 downto 0); -- parallel data input
      sr_out             : out std_logic_vector(15 downto 0) -- parallel data output
);
end component;

component md_state
port(
      clock             :     in std_logic; -- 125 MHz clock input
      reset             :     in std_logic; -- reset
      md_clock          :     in std_logic; -- management clock input
      mdio_in           :     in std_logic; -- input data
      md_shift_enable   :     in std_logic; -- enable shiftregister
      sr_in             :     in std_logic_vector(15 downto 0); -- parallel input
      md_timeout        :     in std_logic; -- external mdc timeout counter activated

      md_readwrite      :     out std_logic; -- operation, read (low) or write(high)
      md_load_op        :     out std_logic; -- load clock count (12) in counter
      md_load_d         :     out std_logic; -- load data count (16) in counter
      md_load_sr        :     out std_logic; -- load shift register
      md_store_register :     out std_logic; -- load data into MII register
      md_clear_register :     out std_logic; -- clear data
      force_mdio_low    :     out std_logic; -- force the mdio signal low
      md_select_mux     :     out natural range 0 to 31;
      md_busy           :     out std_logic; -- processing a frame
```

```
      set_mdio_out_i    :     out std_logic; -- processing a frame
      clear_mdio_out_i  :     out std_logic; -- processing a frame
      md_sw_reset       :     out std_logic
);
end component;

component md_timeout
port(
      clock      :  in  std_logic; -- clock inputs
      reset      :  in  std_logic; -- reset
      md_clock   :  in  std_logic;
      md_busy    :  in  std_logic; - used by state_machine to start counter
      md_timeout :  out std_logic -- enable shiftregister
);
end component;

component md_sync
port(
      clock       :  in std_logic; -- 125 MHz clock
      reset       :  in std_logic; -- reset
      md_clock_in :  in std_logic; -- asynchronous clock input
      md_data_in  :  in std_logic; -- asynchronous data input

      md_clock    :  out std_logic; -- synchronized clock
      mdio_in     :  out std_logic  -- synchronized data
);

end component;

-->> before the begin statement, the signal assigments must be made.

signal invalid_operation_in   : std_logic;
signal wrong_address_in       : std_logic;
signal md_clock               : std_logic;
signal md_data                : std_logic;
signal load_sr                : std_logic;
signal load_op                : std_logic;
signal load_d                 : std_logic;
signal shift_enable           : std_logic;
signal clear_register         : std_logic;
signal store_register         : std_logic;
signal select_mux             : natural range 0 to 31;
signal sr_in                  : std_logic_vector(15 downto 0);
signal sr_out                 : std_logic_vector(15 downto 0);
signal md_time_out            : std_logic;
signal power_down             : std_logic;
signal readwrite              : std_logic;
```

```
signal force_mdio_low         : std_logic;
signal busy                   : std_logic;
signal rx_in                  : std_logic;
signal tx_in                  : std_logic;
signal crc_in                 : std_logic;
signal col_in                 : std_logic;

signal md_data_out_i          : std_logic;
signal set_mdio_out_i         : std_logic;
signal clear_mdio_out_i       : std_logic;
signal enable_mdio_out        : std_logic;

signal md_sw_reset            : std_logic;


begin

-- From this point onwards, all component instances are created
--
--temporarily disabled inst_sync: md_sync port map(clock, reset,
                     md_clock_in, md_data_in, md_clock, md_data);

     inst_shift: md_shiftregister port map (md_clock, reset, load_sr,
                    shift_enable, md_data, md_data_out_i, sr_in, sr_out);

     inst_regmux: md_regmux port map (clock, reset, sr_out, store_register,
                     select_mux, clear_register, sr_in, jabberdetect_in,
             linkstatus_in, remotefault_in, invalid_operation_in,
                     wrong_address_in, md_time_out, md_sw_reset, col_in, crc_in,
                     tx_in, rx_in, loopback, power_down, isolate, collision_test,
                     mii_tx_delay, bypass_nrzi, bypass_scramble);

     inst_state: md_state port map (clock, reset, md_clock, md_data, shift_enable,
                     sr_out, md_time_out, readwrite, load_op, load_d, load_sr,
                     store_register, clear_register, force_mdio_low, select_mux,
                     busy, set_mdio_out_i, clear_mdio_out_i, md_sw_reset);

     inst_count: md_counter port map (md_clock, reset, load_op, load_d, shift_enable);

--obsolete inst_timeout: md_timeout port map(clock, reset, md_clock, busy, md_time_out);

-->> The following assignments constitute the combinatorial part of the entity

    md_clock <= md_clock_in;
    md_data <= md_data_in;
    tx_in <= '0';
    rx_in <= '0';
```

```
    crc_in <= '0';
    col_in <= '0';
    md_busy <= busy;


    comb: process ( force_mdio_low, md_data_out_i, enable_mdio_out)
    begin
       invalid_operation_in <= '0';
       wrong_address_in <= '0';

       if force_mdio_low = '1' then
           md_data_out <= '0';
       elsif enable_mdio_out = '1' then
           md_data_out <= md_data_out_i;
       else
           md_data_out <= '1';
       end if;
    end process;

-->> The last parf of the entity is the sequantial logic

    reg: process(reset,  set_mdio_out_i, clear_mdio_out_i, clock)
    begin
      if reset = '1' then
        enable_mdio_out <= '0';
      elsif clock'event and clock = '1' then
        if clear_mdio_out_i = '1' then
          enable_mdio_out <= '0';
        elsif set_mdio_out_i = '1' then
          enable_mdio_out <= '1';
        end if;
      end if;
    end process;

end structure;
```

Appendix E

CRC Implementation in C

This appendix contains the test implementation for the CRC algoritm in Ethernet.

```
#include <stdio.h>
#include <stdlib.h>

unsigned char bitnr(unsigned char charin, unsigned int bt)
{
  unsigned char tbit, maskbit;

  if(bt <0 || bt >7)
  {
    printf("Error: Bit out of Bounds: bit is %d", bt);
    exit(1);
  }

  maskbit = 1<<bt; /* mask the bit position */
  tbit = (charin & maskbit)>>bt; // mask and divide: output is 1 or 0
  if (tbit !=0 && tbit !=1)
  {
    printf("Error: tbit not a bit (0,1): tbit = %d ",tbit);
    exit(1);
  }

  return tbit;
}

unsigned char xor(unsigned char x1, unsigned char x2)
{
  if ( ( x1!=0 && x1!=1 ) || (x2!=0 && x2!=1) )
  {
    printf("Error, x1/x2 is not a bit (0,1): x1 = %d, x2 = %d",x1,x2);
    exit(1);
  }
```

```
  if ( ( x1 == 0 && x2 == 0) || (x1 == 1 && x2 == 1) )
    return 0;
  else
    return 1;
}

int main(void)
{
  int i, j;
  unsigned char frame[]={

        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, /* broadcast addr */

        0x00, 0xc0, 0xe5, 0x80, 0x05, 0x91, /* source addr */

        0x08, 0x06, 0x00, 0x01, 0x08, 0x00, 0x06, 0x04, 0x00, 0x01, /* misc info */

        0x00, 0xc0, 0xe5, 0x80, 0x05, 0x91, /* source addr */

        0xc0, 0xa8, 0x01, 0x04,  /* source IP */

        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* dest addr */

        0xc0, 0xa8, 0x01, 0x03,  /* dest IP */

        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* stuffing */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* stuffing */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }; /* stuffing */

  unsigned char bucket[32], t1;
  int t2;

  /* start with clearing bucket */

  for ( i=0; i<32; i++)
    bucket[i]=1;

  /* there is the real implementation usng bucket as a shiftregister.
     The program loops along all chars ( i ), and along all bits ( j ),
     and updates bucket everytime with the new values. t1 holds the output
     of bucket[31]

     formula for CRC-32: CRC32 = x32 + x26 + x23 + x22 + x16 + x12 + x11 +
                                 + x10 +x8 +x7 +x5 +x4 +x2 +x1 +1
  */
  if(sizeof(frame)== 0)
  {
```

```
  printf("Error: Framesize is zero");
  exit(1);
}
printf("sizeof frame : %d", sizeof(frame)/sizeof(char));

t2 = 0;
for ( i=0; i<sizeof(frame)/sizeof(char); i++ )
{
  for ( j=0; j<8; j++ )
  {
    t1 = xor( bucket[31], bitnr( frame[i], j) );

    bucket[31]=bucket[30];
    bucket[30]=bucket[29];
    bucket[29]=bucket[28];
    bucket[28]=bucket[27];
    bucket[27]=bucket[26];
    bucket[26]= xor(bucket[25],t1);
    bucket[25]=bucket[24];
    bucket[24]=bucket[23];
    bucket[23]= xor(bucket[22],t1);
    bucket[22]= xor(bucket[21],t1);
    bucket[21]=bucket[20];
    bucket[20]=bucket[19];
    bucket[19]=bucket[18];
    bucket[18]=bucket[17];
    bucket[17]=bucket[16];
    bucket[16]= xor(bucket[15],t1);
    bucket[15]=bucket[14];
    bucket[14]=bucket[13];
    bucket[13]=bucket[12];
    bucket[12]= xor(bucket[11],t1);
    bucket[11]= xor(bucket[10],t1);
    bucket[10]= xor(bucket[9],t1);
    bucket[9]=bucket[8];
    bucket[8]= xor(bucket[7],t1);
    bucket[7]= xor(bucket[6],t1);
    bucket[6]=bucket[5];
    bucket[5]= xor(bucket[4],t1);
    bucket[4]= xor(bucket[3],t1);
    bucket[3]=bucket[2];
    bucket[2]= xor(bucket[1],t1);
    bucket[1]= xor(bucket[0],t1);
    bucket[0]= t1;
    printf("nr: %d, in: %d, out: %d , bucket: %d%d%d%d%d%d%d%d%d%d
                %d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d\n",
                t2,bitnr( frame[i], j), t1, bucket[31], bucket[30], bucket[29],
```

```
                bucket[28], bucket[27], bucket[26], bucket[25], bucket[24],
                bucket[23], bucket[22], bucket[21], bucket[20], bucket[19],
                bucket[18], bucket[17], bucket[16], bucket[15], bucket[14],
                bucket[13], bucket[12], bucket[11], bucket[10], bucket[9],
                bucket[8],  bucket[7], bucket[6],  bucket[5],  bucket[4],
                bucket[3],  bucket[2],  bucket[1],  bucket[0] );
                t2++;
    }
  }
  printf("CRC : %d%d%d%d %d%d%d%d %d%d%d%d %d%d%d%d %d%d%d%d
            %d%d%d%d %d%d%d%d %d%d%d%d\n",
          1-bucket[31], 1-bucket[30], 1-bucket[29], 1-bucket[28], 1-bucket[27],
          1-bucket[26], 1-bucket[25], 1-bucket[24], 1-bucket[23], 1-bucket[22],
          1-bucket[21], 1-bucket[20], 1-bucket[19], 1-bucket[18], 1-bucket[17],
          1-bucket[16], 1-bucket[15], 1-bucket[14], 1-bucket[13], 1-bucket[12],
          1-bucket[11], 1-bucket[10], 1-bucket[9],  1-bucket[8],  1-bucket[7],
          1-bucket[6],  1-bucket[5],  1-bucket[4],  1-bucket[3],  1-bucket[2],
          1-bucket[1],  1-bucket[0] );
  printf(" \n");
  return 0;
}
```