RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Robin Gareus

Slow Control - Serial Network and its implementation for the Transition Radiation Detector

Diplomarbeit

HD-KIP-02-25

KIRCHHOFF-INSTITUT FÜR PHYSIK

Faculty of Physics and Astronomy

University of Heidelberg

Diploma Thesis
in Physics

submitted by

Robin Gareus, `robin@gareus.org`
born in Erlenbach am Main

October 30, 2002

Slow Control - Serial Network
and its implementation for the Transition Radiation Detector

final release - built no. 713
$Id: main.tex,v 1.34 2002/10/30 00:18:12 rgareus Exp $

This Diploma Thesishas been carried out by Robin Gareus, `robin@gareus.org` at the
Kirchhoff Institute of Physics
under supervision of Prof. Dr. Volker Lindenstruth

Slow Control - Serial Network: This document describes the architecture and implementation of a universal serial network interface, designed and developed as diploma thesis. The network is capable of connecting a large number of clients in a hierarchical structure, and allows full duplex communication at fixed bandwidth. Main application will be the slow control network of the Transition Radiation Detector, Large Hadron Collider experiment (LHC) at CERN, where 65000 clients will be configured at 24Mbit/sec. This work includes as well generic interfaces and software for use in other applications.

Ich möchte mich bei Prof. Lindenstruth bedanken, der es mir ermöglicht hat, mich mit interessanten Themen der Physik und Hardware Informatik zu beschäftigen, Ideen zu entdecken und sie zu realisieren. Mein Dank gilt der ganzen Arbeitsgruppe für stetige Hilfsbereitschaft und angenehme Atmosphäre. Namentlich M. Gutfleisch, C. Reichling, G. Torralba, T. Krawutschke, D. Atanasov und T. Alt. Insbesondere danke ich Dr. Falk Lesser und Rolf Schneider, ohne die ich diese Arbeit nicht hätte realisieren können. Dr. Venelin Angelov erweckte in mir die Leidenschaft zu löten, in Hardware zu denken und Kekse zu essen. Vielen Dank für die Beantworung meiner vielen Fragen. Nicht zuletzt bedanke ich mich bei meinen Eltern, die mir das Studium der Physik ermöglicht haben, sowie bei Florian Gmeiner, für eine der besten Freundschaften.
Carolina, I love you.

Contents

1 Introduction

All science is either physics or stamp collecting.

–E. Rutherford

1.1 Motivation

The ALICE experiment Scientists believe there was a Big Bang from which everything in the Universe emerged. In the beginning everything was squeezed into a tiny volume no bigger than a flea. All the particles which make up everyday matter, from which we and everything around us are made, had yet to form. The quarks and gluons, which in today's cold Universe are locked up inside protons and neutrons, would have been too hot to stick together. Matter in this state is called Quark Gluon Plasma, QGP.[6]

Now, approximately fifteen billion years later, mankind is trying to find and study QGP in the Laboratory. To do so, they collide ions, atoms stripped off electrons, into each other at very high energy, squeezing the protons and neutrons together to try and make them melt. Experiments at CERN[1] through the 1980s and 1990s have smashed ions of oxygen, sulphur and lead into stationary targets. The results have given tantalizing hints that QGP might have been created for fleeting moments before cooling down into ordinary matter again.[6]

At the LHC[2], lead ions will collide head-on at energies 300 times higher than at CERN's present day experiments. Physicists believe that these energies will be ideal for making QGP. But how do we search and identify quark matter?

The ALICE (A Large Ion Collider Experiment) uses a head-on collision of a Lead (208 Pb) projectile with a Lead target nucleus, at the SPS beam energy of 160 GeV per nucleon in the Pb projectile, which may compress and heat the nuclear matter contained in the two nuclei. It may thus reach the required energy density (20-fold higher than that of the initial nuclei) in a short-lived "fireball" volume. After about $8 \cdot 10^{-23}$ sec this state expands, cools down and emits hadrons (pions, kaons, lambdas, phi.....) into the detector system. There are about 1500-2000 charged particles created in each of these violent collision events. The detector system thus has to have an extreme spatial resolution to separate the particle tracks.[6]

ALICE aims to study the properties of hot QGP, its dynamical evolution, phenomena associated with the phase transition of rehadronization and finally the evolution of the hadronic final state until freeze-out. To achieve this goal ALICE, as the only dedicated heavy ion experiment at LHC, is designed to measure a large set of observables over as much of phase space as achievable and thereby covering hadronic and leptonic observables as well as photons. The ALICE experimental setup is shown in Fig. 1. The experiment will have a central barrel, housed in the L3 magnet, covering in pseudorapidity the range $-0.9 \leq \eta \leq 0.9$ with complete azimuthal coverage. This central barrel comprises an inner tracking system of Silicon detectors (ITS), a large time projection

[1]CERN: Organisation Europeenne pour la Recherche Nucleaire. (European organization for nuclear research.

[2]LHC: Large Hadron Collider

chamber (TPC), a transition radiation detector (TRD), and a time-of-flight array (TOF). In addition there will be close to mid-rapidity two single arm detectors, an array of ring-imaging Cherenkov counters (HMPID) to identify hadrons up to high momenta and an array of crystals (PHOS) for the detection of photons.[5]

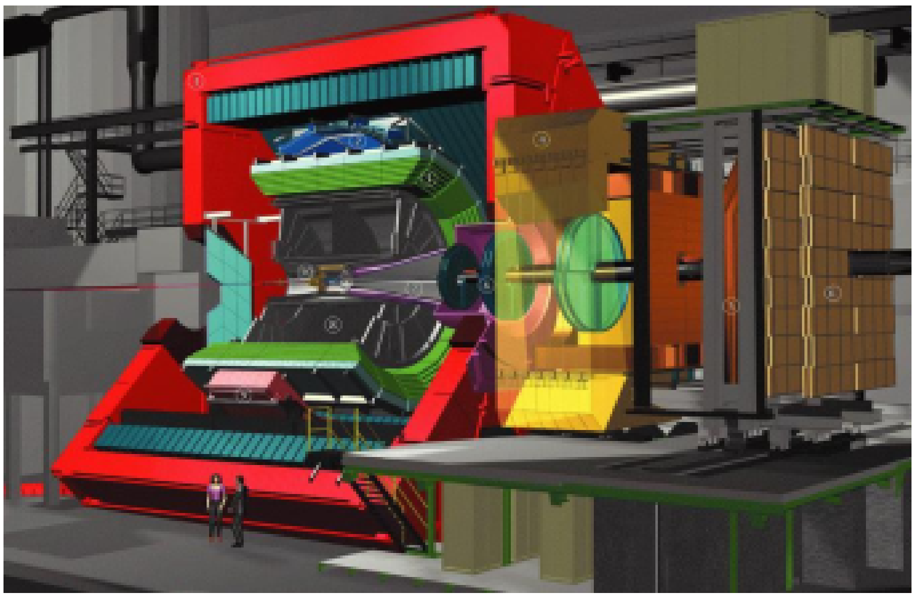


Figure 1: **Layout of the ALICE detector.** The Transition Radiation Detector (3) is the green component placed between the TPC (8) and TOF (cyan, below 2).

The Transition Radiation Detector Every individual particle can be detected and traced inside the ALICE detector system, where particle tracks are converted to digital electronic signals, which are then processed online. The data readout rate in this detector exceeds 15 Terabytes/sec. In order to cope with this massive amount of data, online-processor systems are designed to identify and select the relevant information. A processing system is being designed that performs track fits on about 20,000 tracks, consisting of about 20 space points within two microseconds. The estimated necessary compute power corresponds to $40 \cdot 1012$ arithmetic operations per second.[11]

The chief goal of the TRD is to provide electron identification in the central barrel at momenta in excess of 1 GeV/c where the pion rejection via energy loss measurement in the TPC is no longer sufficient. As a consequence, the addition of the TRD [5] significantly expands the physics objectives of the ALICE experiment[7, 8] .

As detailed in [5] the TRD achieves this goal by reading out and analyzing the charge induced on 1,156,032 pads located in 540 individual readout chambers arranged in 6 layers in the TRD barrel. Most of the front-end electronics sits directly on the readout chambers. (see Fig. 2).

TRAP - The TRAcklet Processor Front-end electronics (FEE), consist of preamplifiers, analog digital converters, tracklet processing and global tracking. The hardware for multiple pads is

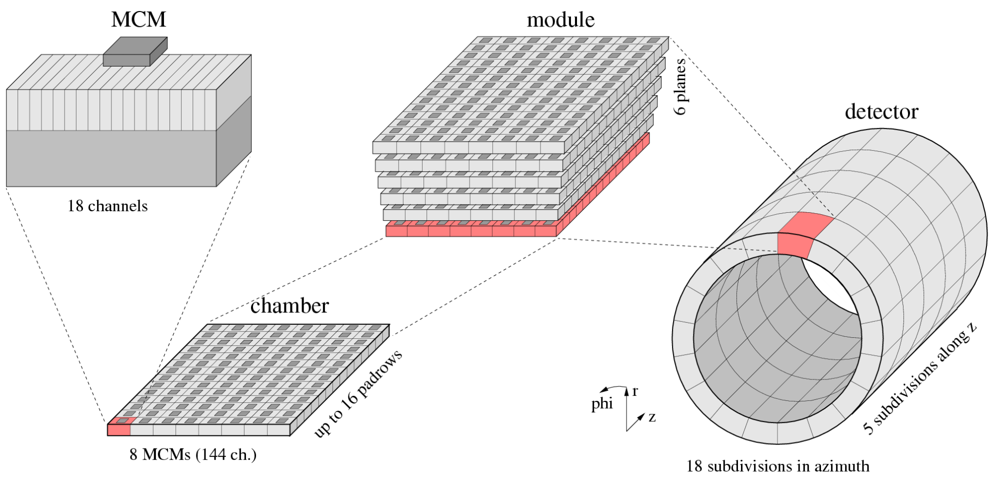


Figure 2: **The Transition Radiation Detector Architecture.**

grouped into Multi Chip Modules (MCM). There are 64224 MCMs mounted on the detector, making the MCM one of the most crucial electronics components which have to be mass produced.

The main part of the MCM is the Local Tracking Unit (LTU) with the Tracklet Processor (TRAP). The LTU functionality includes everything after the ADC. It comprises a so called tracklet preprocessor (TPP), which includes storage of the raw ADC data in the event buffer, a MIMD microprocessor, which subsequently computes and selects the tracklet and the read out part. A picture of these internal parts of the TRAP-1 can be found in chapter 3.3.

slow control network In order to control and configure that large set of hardware, a network with requirements summarized in table 1 is needed: This document describes the architecture of the "slow control network"[3] for the TRD, its implementation, design details, as well as measurements and benchmarks.

Although the motivating application is very special, during the development, the network has become far more universal, than previously assumed, so further part of this documentation will not mention the ALICE experiment any more...

The first approach was to find an existing network specification, that accomplishes all the tasks. Several networks and field busses like CAN, SMBus, JTAG, i^{2}c, and Ethernet have been considered, but not found appropriate. The chosen solution uses Ethernet[4] to connect to the 512 chambers of the TRD, where the data is distributed further to the MCMs. The second choice was to create a new network on chamber level to connect the multi chip modules. Each chamber covers 2 Altera^{TM}Excalibur NIOS FPGA controller, that interface between Ethernet and the local network to the MCMs. The MCMs will be subgrouped into $\approx$ 20 MCMs[5] connected in a ring[6]. (20 MCMs per

[3]*slow* means that now fast data readout is done via this network.
[4]even here a modified version of Ethernet is used, to work in the magnetic field. see [13]
[5]number varies, with the geometry of the detector
[6]Using a ring structure benefits to the proposed TRDs cabling architecture

- Connect all clients (≈65000 MCMs) in a hierarchical structure.
- Provide failsafe full duplex communication.
- Allow broadcasts, since most configuration will be identical to groups of MCMs.
- Failure of a single MCM should not affect (many) others.
- Analyzation of the network (broken Links, MCM keep-alive) and system checks of network clients should be possible.
- leave flexibility in case of failure. (When once in use, the Detector and all its internals become almost inaccessible.)
- Complete configuration (TRD ≈ 1 MByte/MCM) in ≈ 1sec.
- work inside the detector system (high magnetic Field, particle beams,...) without interfering with the measurement (as less wires as possible, low currents,...)
- (Re)Use of proposed/tested TRD architecture, specified in the TDR [5].

Table 1: **TRD slow control network requirements.** This list summarizes the specifications the network has to fulfill.

ring, 3 rings per controller, 2 controllers per chamber times 540 chambers makes 64800 MCMs in total.)

1.2 Overview and Features

The **slow control - serial network**[7] (scsn) is a high speed interface, that provides a reliable asynchronous serial communication in between one *master* and multiple clients over a single signal wire. To gain redundancy and full duplex communication, 4 signal wires[8], two for data transmission, and two for receiving data are used.

The network is written in VHDL[9] and so can be synthesized in hardware on any kind of PLD[10] or via ASIC[11].

scsn has the goal of being reusable and not bound to the application, so the design contains a lot of generic values and plug-ins. Most of them can be changed quite easily, and require just constants to be changed or certain parts of the vhdl code to be adjusted.

[7] initial development considered the slow control to be a bus. Therefore all development resources and software had been named *scsb*. The final version of scsb is postulated to be a network (*scsn*).

[8] current implementations use a differential signal, so 8 wires.

[9] VHDL: Very High speed integrated circuit Hardware Description Language (HDL)

[10] Programmable Logic Device (see FPGA, CPLD,...)

[11] ASIC: Application Specific Integrated Circuit

[12] The maximum number of slaves is a generic value in the protocol header

[13] There are further possibilities to extend reliability. see chapter 4.1.8

physical connection	4 twisted pair LVDS signals. (8 wires)
network speed	max. 1/3 of clock speed. (tested up to 24 MBit/s @ 72MHz)
network topology	double ring, with one *master* and up to 126 *slaves*[12]
power consumption	depends on implementation
data exchange format	generic application layer, current implementation provides a 16bit address, 32 bit data bus
data checksum	Cyclic Redundancy Check (CRC-16)
Minimum Clock speed	DC
Maximum Clock speed	implementation specific. FPGA: 96MHz / UMCL.18μ250t2: 120MHz
Redundancy	network works in half duplex mode with one broken (excluded) client. If more than one client breaks, the nodes in between the two will be lost.[13]

Table 2: **TRD scsn specifications.** This tables summarizes the features and specifications of the TRD scsn network appliance.

2 Architecture

The most brilliant decision in all of Unix
was the choice of a single character
for the newline sequence.

–Mike O'Dell, only half jokingly

In this section, the network structure and protocol definition of scsn are described. This will also give an overview of the complete network and its capabilities. Note that the discussion of the chosen structure has been separated from the specifications and can be found in the section 2.7. Most of the details in data link layer are left generic, and are described only abstractly. The chosen implementation methods are topic of the following chapters.

2.1 Network structure

The main structure of the slow control network emerged from the TRD[14] Technical Design Report (TDR) [5]. The requirements to have a redundant architecture with few wires only, and still be able to connect $\approx$ 65000 clients, led to a daisy-chain like architecture (Figure 3). Up to 126 clients (called *slaves*) are connected in a ring structure with exactly one controller (called *master*)[15]. Between two devices (master or slave) there are 2 links, one for each data flow direction, each link being single serial signal. Current Implementations use LVDS[16] @ $\leq$ 40 MHz.

To gain redundancy each of the slaves supports *X-bridging* (see Figure 4). In "normal" (un-bridged) mode, a slave forwards the data to the next slave (until it arrives at the master) in the same ring. Whereas in "bridged" mode, the data is sent back on the other ring, breaking up the full duplex ring into two half-duplex rings. Usage of this method, allows the network to operate with broken clients, as well.

2.2 Network protocol

First of all a two basic statements which are discussed in the following sections:

- Data is exchanged in fixed size packets called *frames*. Each frame start is indicated by a start-bit (1) and terminates after a fixed length or an receive-error.
- Principle : One Frame In - One Frame Out! Each frame is created by the master and terminates there. The slaves only forward or alter the frame.

The network interface has to take care about certain services, like (de)serialization, addressing, processing or forwarding the data. Figure 5 shows, how each of these tasks are capsulated into different abstraction layers.

[14]Transition Radiation Detector

[15]The number of clients per ring is only limited by required latency, redundancy, and header address length restrictions.

[16]Low Voltage Differential Signal: 2 Wires per signal.

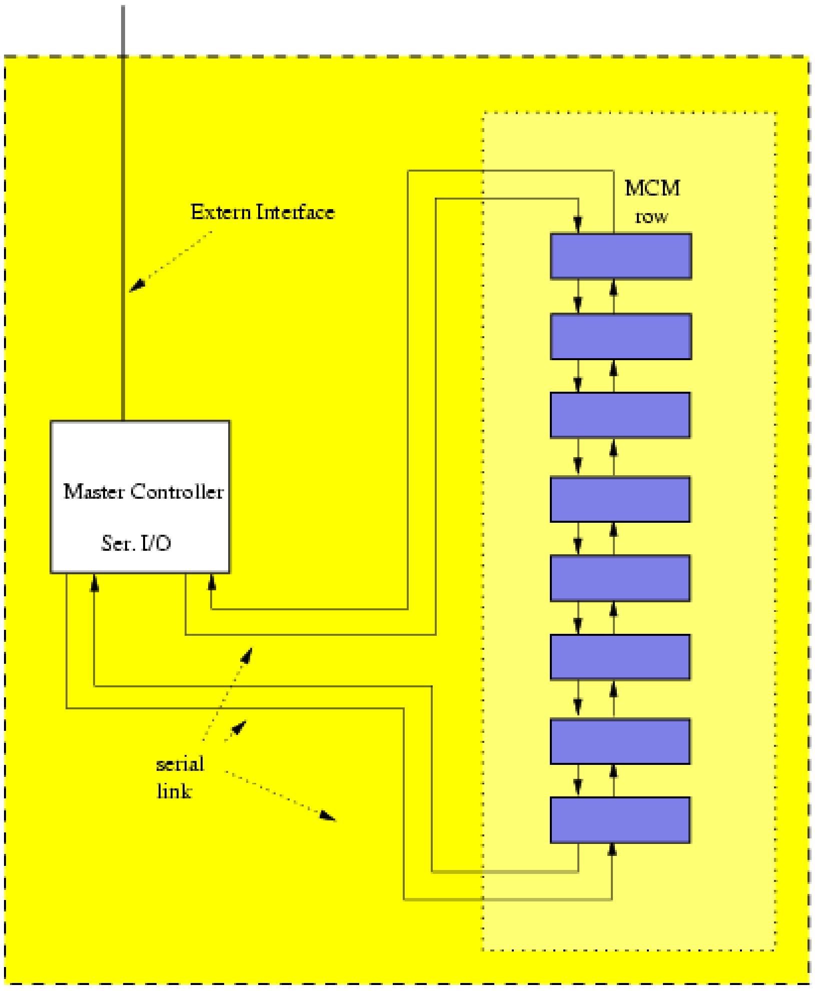


Figure 3: **Architecture:** This figure depicts the ring-topology of the network on a chain of 1 master and 8 slave devices.

Starting a bottom up approach, will describe the architecture of the scsn:

Physical Layer The physical layer, is the abstraction layer of the wire and Input/Output electronics. The LVDS signal from the wire is decoded and synchronized to the local clock signal. There is no need for a PLL synchronization here, since scsn provides Data Link Layer synchronization. A digital low pass filter is applied to smooth the signal before handing it data to the layer.

Data Link Layer The data link layer has to carry out specific functions to provide a well-defined service interface to the network layer. What the Data Link Layer does, is accept a raw bit stream and attempt to deliver it to the destination. This bit stream is not guaranteed to be error free. The number of bits received may be less that, equal to, or more than the number of bits transmitted, and they may have different values. It is up to the data link layer to descry, and if necessary, correct errors[17]. First of all the bits from the wire have to be detected and deserialized. Data is sent

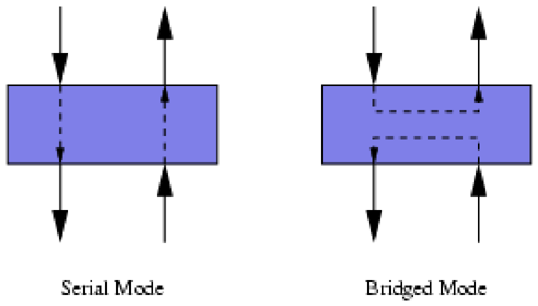


Figure 4: **network bridge - internal x-bar.** This feature allows the split the network into two independent Rings, necessary to exclude broken clients.

with LSB[17] first. Transmission speed in scsn is usually slower than the clock speed of the slave's network controller, so that the job of deserialization can be done in Data Link Layer, which allows more complex resynchronization algorithms. Since the frame is fixed size, the start-bit is used to synchronize data to the internal clock. Every edge of the data signal can used for resynchronization. To keep in sync, after a given period of a non changing Signal, *stuff bits* are inserted to force a transition that is used for resynchronization. (see chapter 2.7.2, Fig. 11).

All the bits of the physical layer are grouped into frames which are also buffered here. Framing is implemented via a start bit, and a bit counter, using a fixed size frame. This is discussed later in chapter 2.7.2. Dealing with transmission errors, (re)calculating the frame CRC-sum[18], as well as the implementation of the network bridge is also part of the data link layer. All buffer errors (except the CRC-Errors) are handled internally, details are in chapter 4.1.3.

If a complete frame has arrived, the data link layer sends the received frame (without checksum) up to the network layer (*NWL*). Checksum errors are also reported and left to be handled by the NWL. (Signals in Figure 5 data path up: *new, error, data(69)*). Since it takes some time to process the frame, frames have to be separated from each other by an *interframe space*. This is done by sendtiming/outputbuffer units. After flushing the output buffer, the buffer is locked for a short time (not ready to send). Usually the network and application layer transactions are much faster (< 10 clocks) than the network speed (including interframe-space ≈ 250 clocks, see also Eqn. 2).

The data link layer provides a *ready to send* signal to the layer above, which is driven to 1 (high active) if the output buffer is empty and ready to send a frame. The network layer may then write data and start sending by rising the *send data* signal. (Signals in Figure 5: data path down)

2.3 Data frame definition

To understand the function of the network layer, the frame and addressing mode have to be defined first. Each Frame consists of a header (like the envelope of a letter, containing the address and

[17]least significant bit first: 13 would result in "1011"
[18]CRC: cyclic redundancy check
[19]OSI: Open Systems Interconnection reference model

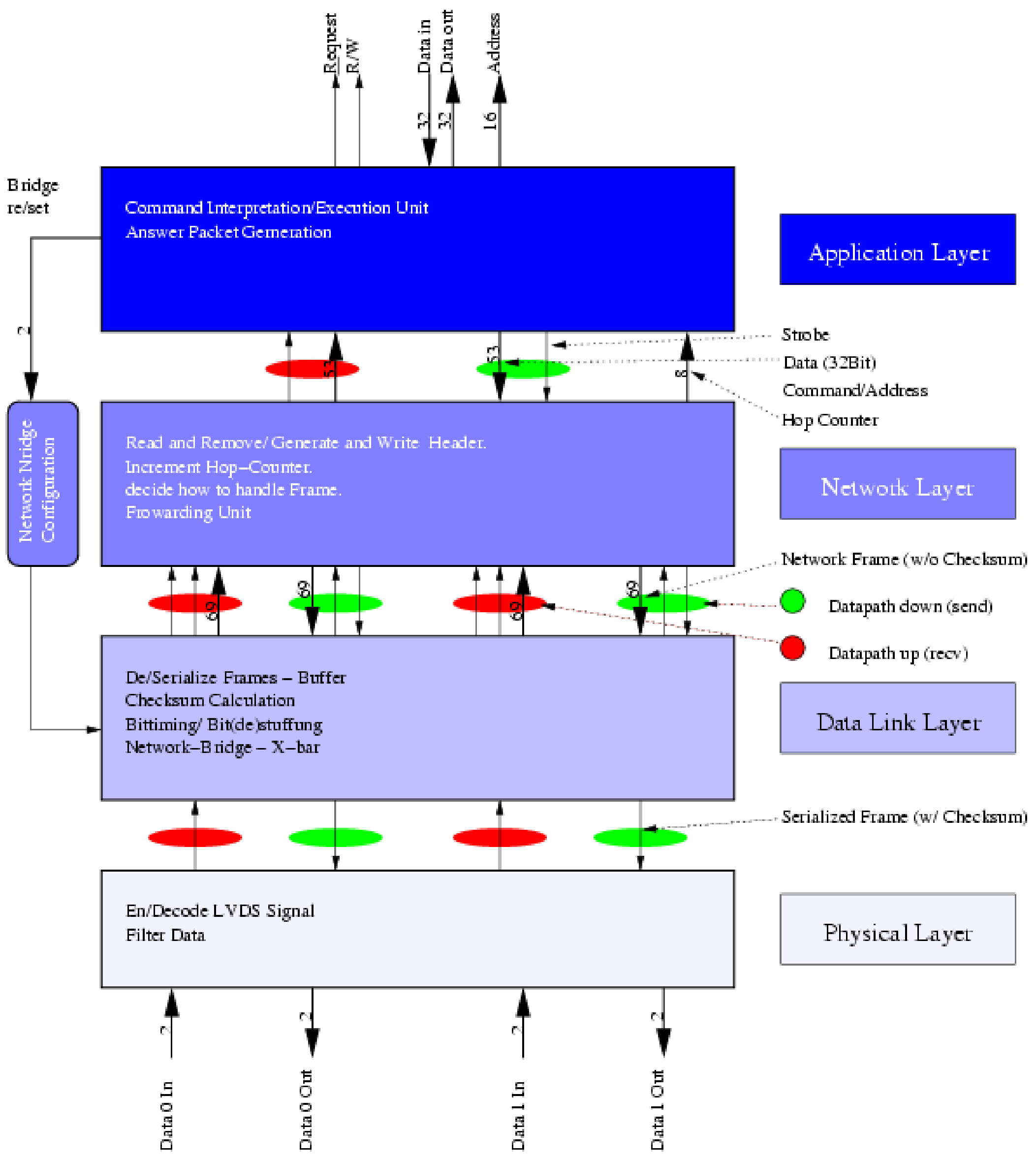


Figure 5: **OSI[19]Layer Diagram.** This Figure shows the capsulateion of functions in the different abstraction layers of the network. The different layers are explained in the text.

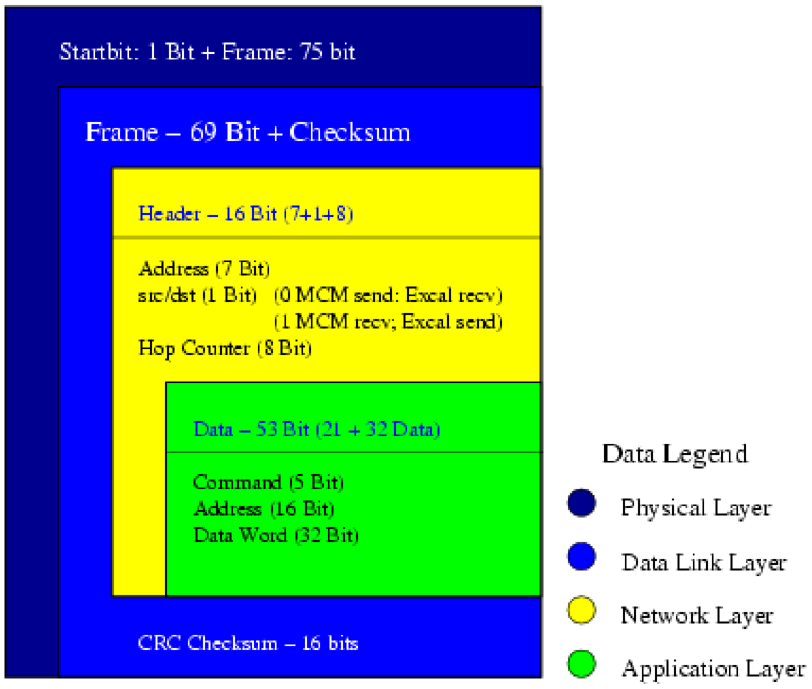


Figure 6: **Frame payload definition:** The frame is a 76 bit fixed size packet.

sender), and the "letter" itself, which is called the data or the payload.

The scsn network allows only the master to "write" a letter. It is addressed to one or all slaves in the network. The receiver opens the letter, reads and/or alters the data and sends it back to the master. Slaves may not address frames to any other slaves[20]. If a slave obtains a frame not addressed to itself, the frame is just forwarded to the next slave until the frame is back at the server.

Figure 6 shows the frame definition, and the according layers: the startbit and the checksum are handled in data link layer, and the payload data will be explained later in Application Layer paragraph. So the header is left to be processed by the network layer...

Network Layer The network layer is concerned with getting frames from the source all the way to the destination. This function clearly contrasts with that of the data link layer which has the modest goal of just moving frames from one end of the wire to the other.[17] To achieve its goal, the network layer must know about the topology of the network.

Addressing in the scsn network is done by assigning the frame to the n'th hop in a row. Every node that the frame is passed by, increments the *hop counter* of the frame. If the (incremented) hop counter matches the address, the data is processed in the application layer. If the hop counter and the address do not match, the frame is simply forwarded to the next in the row, until it arrives back at the master. If the bridge of a slave is in crossed state, the network layer has to disable any broadcast requests before forwarding the frame back. This is necessary because all the slaves would process the same frame again when it is on the way back. Basically this is not a problem, since the

[20]although this would be possible with the actual frame specifications.

Address	src/dst	Description
1	1	master sends to (*requests*) client 1
127	1	master broadcasts to all clients
1	0	client 1 answers the master (replacing the master request)
0	0	Error frame, forward to master, unknown sender (see hopcounter)

Table 3: **Network Address Table examples.**The Address and src/dst bits are part of the frame header.

same transaction is just repeated, but there might be (and are in the TRAP-1) memory-address-auto-incrementers as internal parts of the client.

Another job of the network layer is to schedule commands from both rings to a single application layer interface (which might stall). This is becoming quite complex, since bridge-switches and error handling is also part of the algorithm, which is described in chapter 4.1.5. Although this functionality would correspond to a Session Layer in the OSI reference design, it was named and included in the network layer of scsn because of implementation and capsulateion motivations.

2.4 Frames and Frame Handling

The Header of the frame is 16 bit long (see Figure 6), containing a 7 bit address, 1 source/destination flag, and the 8 bit hopcounter. In worst case, the frame has to go completely down the chain, is reflected at the last in the row (bridged) and goes all the way up, back to the master. So the hopcounter has to be 1 bit larger than the address. The src/dst bit is used to determine if the frame is a request from the master (1) or an answer of a slave (0).

Apart from addressing, the network layer (*NWL*) arbitrates two rings to a single application layer. All in all this is not an easy job. The NWL is the place where all control signals congregate. Depending on the buffer statuses (new frames, CRC errors, free send buffer(s)), the configuration (switch X-bridge) and the application Layer (idle/busy), a decision has to be made how to handle an arrived frame.

Each received frame can either be

- *just forwarded*. (thereby increasing the hop counter)
- *processed*, but **not** altered and forwarded (thereby increasing the hop counter)
- *processed*, altered. This creating a new frame (hop counter 0 - by default this is an error-frame), addressed back to the master, that is put back on the wire, instead of the original frame.
- *answered with error*.

Application Layer Having finished all the preliminaries, here is the description of the application layer, with the implementation specific interface. The layers below the application layer are to provide reliable transport, but they do not do any real work for users[17]. There are currently two

Signal Name	width	APL	Description
arbiter request	1	out	request the bus, wait for ack.
arbiter ack	1	in	arbiter allows request in the next clock cycle. (if this device is acting as bus master, set this statically to 1)
bus select	1	in	select/use the bus. performs the bus transaction.
bus $\overline{\mathrm{R}}$/W	1	out	write enable. 0: read transaction, 1: write transaction
bus data in	32	in	data from a read request
bus data out	32	out	data to be written
address	16	out	address where to read from/write to

Table 4: **Application Layer Bus Interface Signals.** This is a generic slave side interface signal definition.

application layer appliances of the scsn. One that was used for debugging, which is a generic VHDL interface, and the one implemented in the TRAP-1. Both are described later in chapter 3.

As displayed in Figure 6, the current payload definition provides 16 bit address, 32 bit data and 5 bit command requests, which allow standard input/output to almost all known bus systems.

2.5 generic Interfaces

From the outside, the scsn network has just two interfaces: The application layer of the master and the slave. The slave's interfaces is a generic Bus interface. The signals are summarized in Table 4, and are described in detail in chapter 3.

The interface on the master side is different, because it interfaces directly to the data link layer. The network layer (and layers above) functionality is left up to software and/or the user. For common use a master has been mapped to a PCI-Card FPGA, and Software for standard PCs has been developed as part of this work(see chapter 4.2).[21]

2.6 Network Error recovery strategies

There are three kinds of error conditions, that require recovery:

- *transmission errors:* bad wires, clock delays may lead to wrong, incomplete or lost data. These errors are handled by the Data Link Layer. Bitstuff errors are treated as frame end, which may lead to a receive error for the following frame, also. The DLL will create a single error frame instead.

- *postponement errors:* Due to different length executing time in the application layer, buffer overruns may occur. If a new frame is arriving, whereas an other is still being executed in the application layer, a postponement error error occurs in Network Layer.

[21]There are implementations, that reduce the master network layer to a simple state machine, that does not support Type 3 error recovery (see 2.6), but can be implemented in micro-controllers or FPGAs.

- *broken Hardware:* A client board is broken, and there is no access to the network. Or even worse, the client is broken and generates random data on the wire.

Type 1 and 2 errors are handled internal by the hardware. Transmission errors are part of the data link layer, and postponement errors are caught by the network layer.

Changing the Network Topology In order to cope with broken clients, the network bridge was introduced in chapter 2.1. The network bridge can be configured by sending a specific command, to induce the slave to change an internal state. A successive recovery would bridge the first client in the row, and check if the request returns. If yes, reset bridge and test 2nd client, until the test frames run through the complete row (frames arrive back at the master) or the broken client has been detected. Playing the same game on the second ring allows to exclude the broken client(s). See Figure 7. If more than one client breaks, all slaves in between the two broken become unreachable.

Note also, that the master needs to know the current bridge setup, so as to correctly route frames.

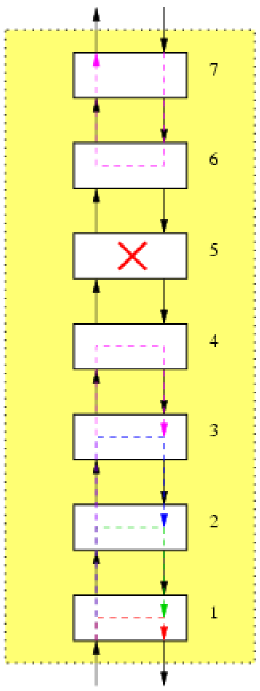


Figure 7: **Error recovery:** Client 5 is broken. By successively bridging and testing clients 1 to 4 and 7 downto 6, it is possible to determine and exclude the broken client.

2.7 Discussion and Extensions

So far, there were only definitions and statements but no discussion or reasons why things have been implemented that way. Most of the structure evolved directly follows the goals and motivation. One of the major focuses was to capsulate the functionality and retail flexibility. The following section will briefly explain the major choices:

2.7.1 Topology and Redundancy

Once the TRD is installed, the clients are almost inaccessible. So the network must be able to accomplish with possibly broken clients. From this point of view, the choice of a daisy-chain like structure might seem to be a basic construction mistake. But as Figure 8 exemplifies: In topology a),

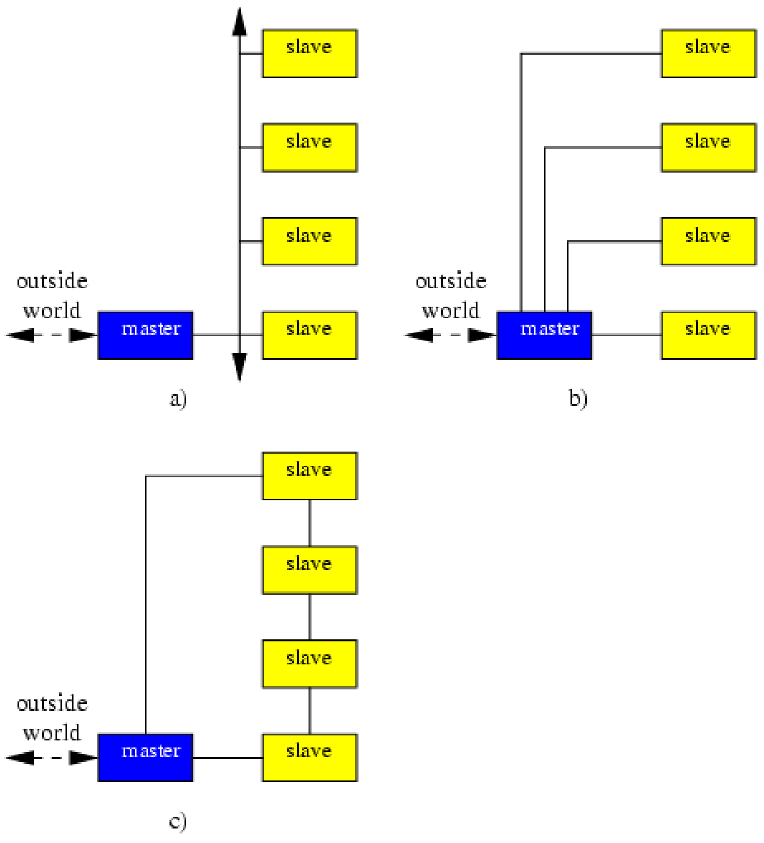


Figure 8: **Network Topologies.** Discussed network structures. see text.

a single broken client might cause the whole network to fail (e.g. shortcut the wire). Furthermore, all Ethernet like CSMA/CD[22] functionality must be implemented, although cabling would have been nice. Topology b) is not suitable to connect all clients without big wiring problems.

To gain even more redundancy, the ring topology c) has been used twice, which leaves some alternatives in building a two-ring structure: Figure 9 shows the two mainly discussed alternatives for using two rings, of which option a) was chosen. Option b) here would have a lower loss rate/probability if only few devices break (see Fig. 10), but is was discarded for the following reasons, although none of them is a really good argument against:

- loosing two in a row breaks the complete network.

[22]CSMA: Carrier sense multiple access; CD: collision detection

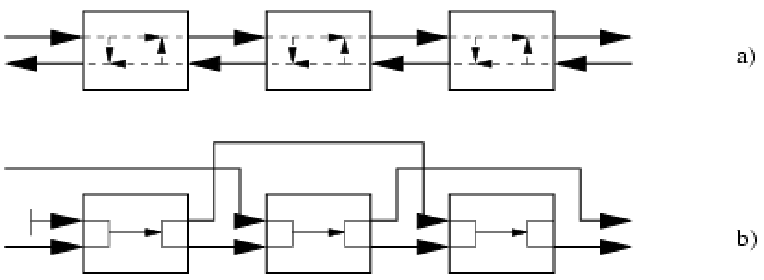


Figure 9: **Bridge topologies.**

- more complicated wiring.
- only half duplex communication.
- offers less debugging methods and recovery strategies, if broken devices are in the ring.

2.7.2 Transmission Protocol

The main intention here was to adopt existing standards: CRC is fine, since the checksum can be computed while (de)serializing in a shift-register[18]. The CRC-16 algorithm with polynom $x^{16} + x^{15} + x^2 + 1$ is used, therefore all single and double bit errors, with an odd number of bits, and all burst errors of length 16 or less, 99.997 percent of 17 bit burst errors, 99.998 percent of 18-bit and longer bursts are caught[17].

Another major decision was whether to support full duplex communication, with separate buffers for each ring, or to merge the rings in data link layer. Trade offs are about 300 Flip-Flips and the extra logic for the second buffers, which were taken into account.

The actual Version inserts a Bit(de)stuffing Unit in between Data Link and Physical Layer, which can be bypassed at compile time. Bitstuffing is a standard, elucidated e.g. in [17], the implementation details can be found in chapter 4.1.3. Note that scsn does not append a stuff bit at the end of the frame (even if the last n bits were identical). Since there is no need to escape characters, the bitstuffing is only for:

Early error abort When the data link layer of the destination sees the start-bit, it knows how many bit will follow, and hence where the end of the frame is. There is no *EndofFrame* character (like the startbit). The trouble with this algorithm is the the count can be garbled by a transmission error. Even if the checksum is incorrect, therefore the destination knows, that the frame is bad, it still has no way of telling where the next frame starts without pausing a complete frame[17].

When using bitstuffing (see Figures 11, 13), a series of 8 identical bits on the wire (all zero, or all ’1’) will be interpreted as a stuff error and the receiver resets its i/o counters, waiting for a new startbit. This considerably simplifies framing and frame location algorithms.

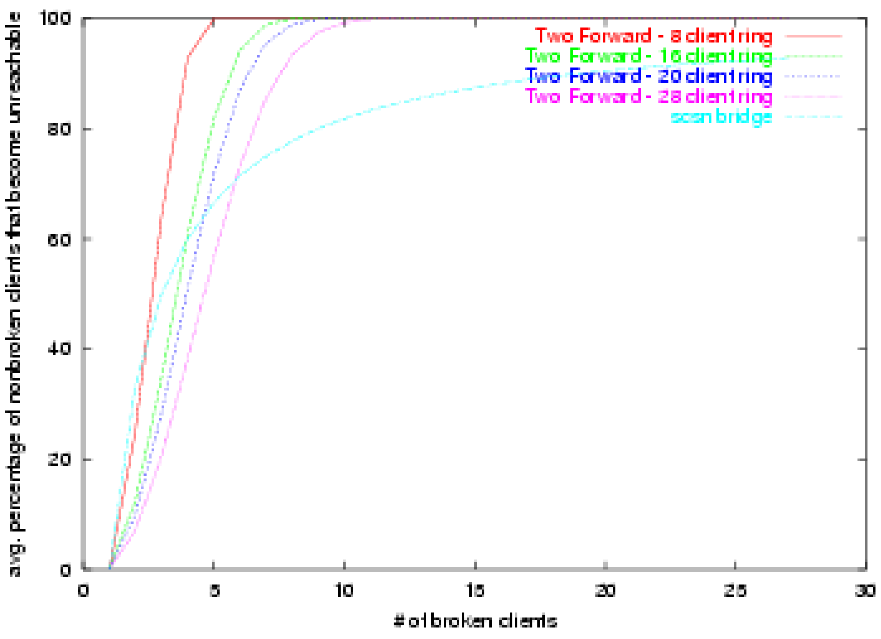


Figure 10: **Bridge Topology Loss calculation.** This diagram compares the loss rates of both bridge topologies. The amount of working but inaccessible clients are shown depending on the total number of broken clients in the ring. The loss rate of "two-forward" bridge depends on the total number of clients, while the scsn bridge algorithm is independent of that.

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1
(b) 0 1 1 0 1 1 1 1 1 1 1 1 **0** 1 1 1 1 1 1 1 1 **0** 1 1 1 1 1 1 1 1 **0** 1 0 0 1
(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1

Figure 11: **Bit stuffing** with a stuff-length of 8 bits. (a) the original data. (b) The data as they appear on the line. Inserted stuff bits are printed in bold letters. (c) The data as they are stored in the receiver's memory after destuffing.

Timing and resynchronization The physical layer does not have a PLL. "Physical Synchronization" is done by a flip-flop running with local clock and a low pass filter[23]; see section 4.1.2. All further adjustments are done in data link layer. A discussion of hardware resynchronization can be found in [15]. Every edge of the data signal is used for resynchronizing the data signal to the internal clock. Usually this is not necessary, since clocks go more exact than needed (see equation 3), but it allows nodes to have clock differences of $\approx 4.1\%$ (eqn. 5)[24].

$$\begin{aligned} \text{transmission speed} &= \text{clk speed} \cdot \left(\text{network speed ratio}^{-1}\right) \\ &= 72\,[\text{MHz}] \cdot \frac{1}{3}\left[\frac{\text{bits}}{\text{clockcycle}}\right] = 24\left[\frac{\text{MBit}}{\text{s}}\right] \qquad (1) \end{aligned}$$

$$\begin{aligned} \frac{\text{clock cycles}}{\text{frame}} &= (\underbrace{84\,[\text{bits}]}_{\text{frame}} + \underbrace{9\,[\text{bits}]}_{\text{interframe space}})\left[\frac{1}{\text{frame}}\right] \cdot \frac{1}{3}\left[\frac{\text{clockcycles}}{\text{bits}}\right] \\ &= 252\left[\frac{\text{clock cycles}}{\text{frame}}\right] \qquad (2) \end{aligned}$$

Equation 1: **Bit timing.** simple speed calculation of the scsn network. Values from the TRAP-1 implementation. Eqn. 2 shows the number of (internal) clock cycles, it takes to transmit one frame. To separate two frames, a timeout *interframe space* > *stuff length* (here: 8) was chosen. [25]

$$\frac{\text{synclen}-1}{\text{synclen}} \leq \frac{\text{CLK}_{\text{src}}}{\text{CLK}_{\text{dst}}} \leq \frac{\text{synclen}+1}{\text{synclen}} \qquad (3)$$

Equation 3: **Resynchronization**. The start bit is used for initial synchronization of each frame. Until the end of the frame clocks of the sender and receiver have to stay synchronized. Depending on the clockcycles/bit (*synclen*), a certain clock-shift is tolerated. Current scsn implementation has 3 clocks/bit, and thus a tolerance of 1 clock cycle. The synclen can be drastically reduced, by using resynchronization, or a larger network clock ratio.

$$0.996032 = \frac{251}{252} \leq \frac{\text{CLK}_{\text{src}}}{\text{CLK}_{\text{dst}}} \leq \frac{253}{252} = 1.003968 \qquad (4)$$

$$0.958334 = \frac{23}{24} \leq \frac{\text{CLK}_{\text{src}}}{\text{CLK}_{\text{dst}}} \leq \frac{25}{24} = 1.041667 \qquad (5)$$

Equation 4, 5: **Synchronization to stuff length and not framelength.** When using bit stuffing, it is guaranteed to have a bit transition each (clocks/bit · stuff-length) = 24 clocks. As a result, it is possible to resynchronize to (every) transitions on the wire. Thereby clock jitter can now be maximal $\approx 4.1\%$!

2.7.3 Frame Definition

Yes, the frame has an overhead of 30% and has no protocol version information in the header. The frame definition is changeable in compile time. Things have been defined to be useful for the TRAP-1 environment.

[23]Note: transmission (service) speed is $\geq \frac{1}{3}$ of local clock.
[24]The value depends on the network clock ratio
[25]The interframe spaces, does not have to be larger than the bitstuff length, but it makes things safer.

There were several discussions about the addressing method:

- Switching the bridge changes the addresses of the clients. Since the master will be written in software, this does not seem to be a big deal. This method has the advantage, that the clients do not have to be configured, and do not have to store a configuration (except of their bridge state).

- Broadcasts: If the network is running in bridged mode, each client will receive broadcasts request twice. This issue can be resolved if the bridged client catches and voids all broadcast requests.

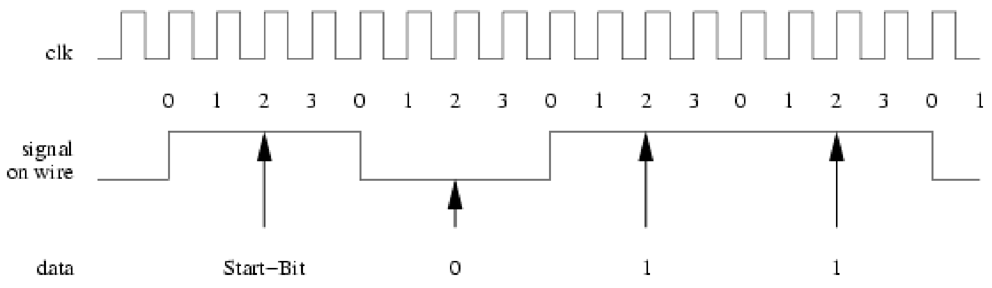


Figure 12: **Startbit and Timing.** This figure show the internal timing counter (here: transmission speed = 1/4 clock speed: timing counter from 0 to 3[26]. The data is captured in stage 2.) The counter is reset with the rising edge of the startbit data signal.

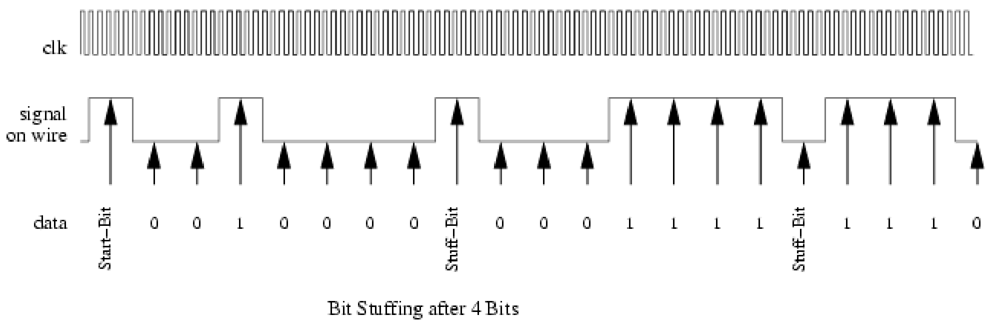


Figure 13: **Bit Stuffing** example with a stuff length of 4. After 4 identical bits raw a stuff bit is inserted, to escape the signal and force a bit transition on the wire.

[26]Note: this gives a jitter tolerance of 1 clock cycle. (see jitter)

2.7.4 Routing

Routing is easy. In redundant mode there is exactly one way to reach a slave (if any), but in full duplex mode, each slave can be addressed on both rings. The current routing algorithm sends consecutive frames alternating on both rings. This may lead to chronological inconsistencies, i.e. when addressing the auto incrementer. As a simple workaround for sequence transmissions, routing can be forced to use one ring only.

2.7.5 Buffers and synchronization

As described above, every network node has two input and output buffers. Simultaneously, there may be as many frames on the ring, as nodes (master and slaves) in the network. Note: since the master has to keep track of all coexistent frames, it should provide a buffer that disallows contemporary sending of identical frames. The following figures illustrate the input/output buffer usage forwarding a frame over two scsn devices in a chain. Figure 14 a) explains the notation, while Fig. 14 b) - 15 show the transmission of 1,2 and finally 3 frames.

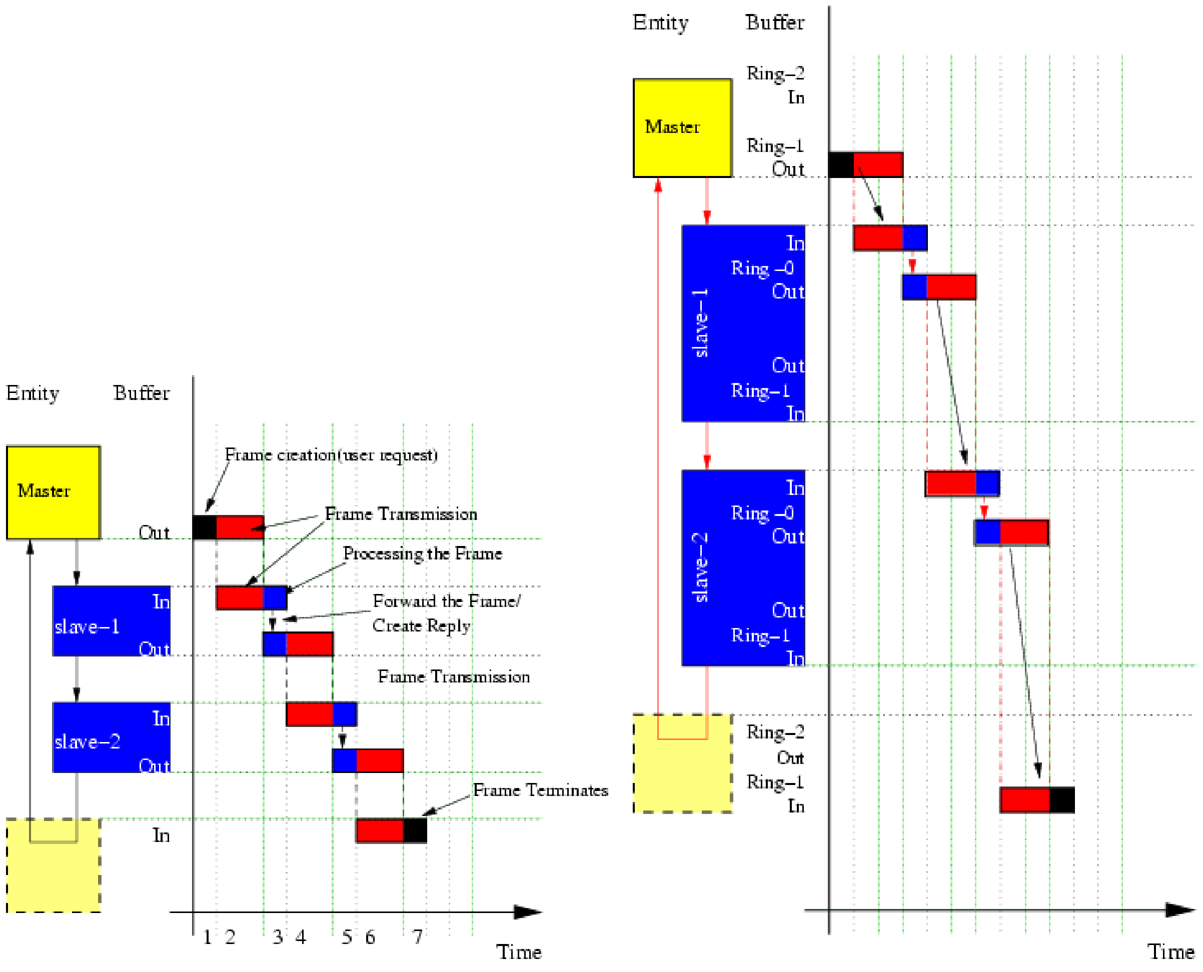


Figure 14: **a) simple Transmission**: 1) The frame is generated and put in the outputbuffer of the master (black). 2) frame is sent to first client (frame color: red). 3) frame is processed on the first client and moved from client-1/input buffer to client-1/output buffer (blue). 4) frame is sent to 2nd client. 5) client 2 processes the frame (see 3) 6) client 2 outputbuffer sends the frame. 7) master receives frame. **b) complete views**: same Transmission, showing buffers for both rings.

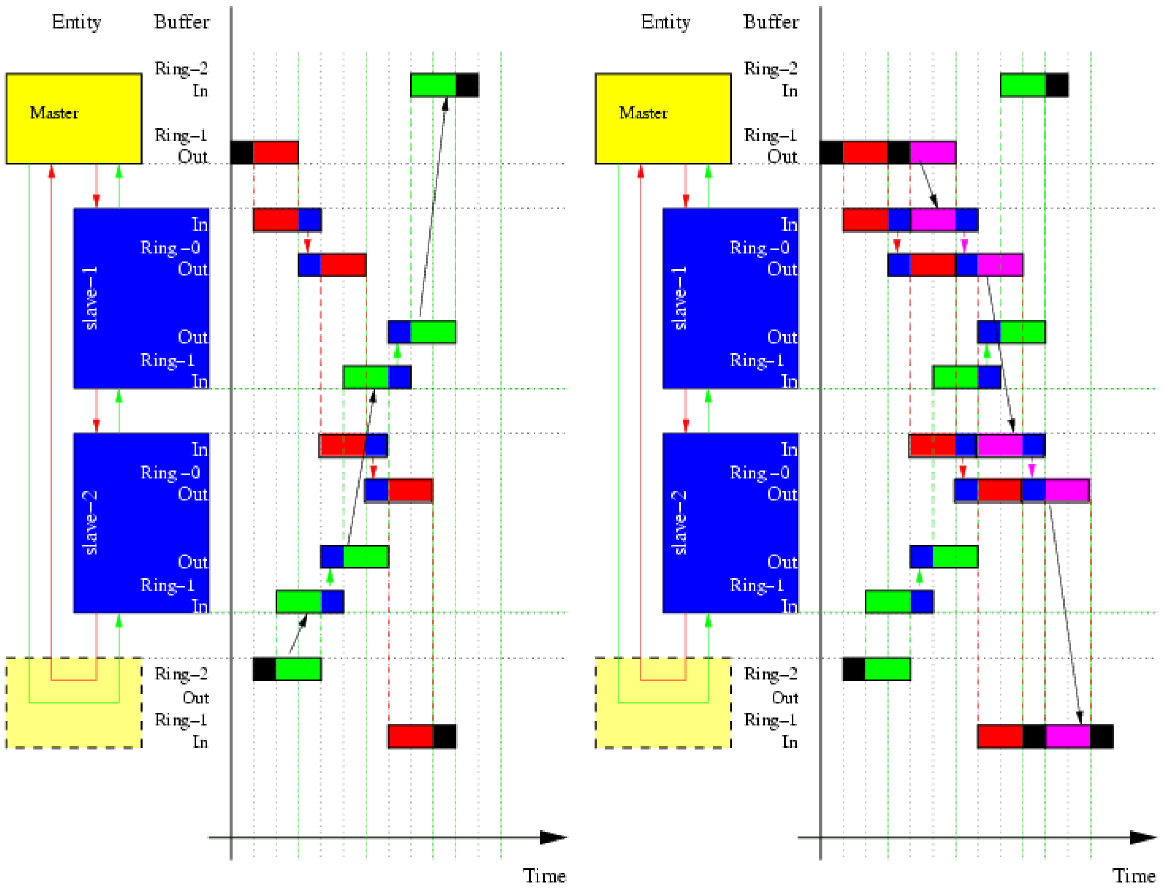


Figure 15: **More simultaneous transmissions**: This figure illustrates frame forwarding in a 2 slave chain. Things can grow quite complex in a larger network.

3 Implementations

never underestimate the bandwidth of
a station wagon full of tapes.

–Andrew S. Tannenbaum

This chapter deals with the currently existing designs, their problems, applications and implementations. It creates an overall picture of the designs, especially the TRAP-1:

The network was implemented in Hardware using VHDL[27]. The physical and data-link layer of both master and the slave are identical. Above these layers the master uses software to generate or handle received data. The slaves are completely coded in hardware, with the application layer being a common bus-interface to issue read and write commands.

3.1 Designflow

Before going into details about the implementations, here is some information about the tools that were used to create the hardware designs. All VHDL code has been written with the VI editor. Simulations have been done with Modelsim^{TM}, so the testbenches should work out of the box with most simulation software.

FPGA implementations have been assembled and compiled with multiple Tools, on different Operating systems, without encountering any problems. Altera^{TM} Max+plus^{TM} has been used to generate all test and sample implementations while Altera^{TM}Quartus^{TM} II was needed to map the PCI designs for the ACEX. The VHDL source of scsn has only rarely been used directly in Max+plus^{TM}and Quartus^{TM}. For further use, EDIF files of scsn master and slaves have been generated from the source with Exemplar Leonardo Spectrum^{TM}and the Synopsys^{TM} FPGA compiler.

The configuration unit of TRAP-1 has been synthesized with the Synopsys^{TM} design analyzer, using the UMCL18u250t2 library. More information about the TRAP-1 design flow can be found in [14].

3.2 The Tracklet Processor 1 (trap1)

The TRAP-1 is a 0.18μm technology integrated circuit, with the task of performing high speed tracklet calculations on data of the TRD. Internally this is done by four CPUs (MIMD[28]) and a data preprocessor, which both can be configured individually. The tracklet processor is designed that way that all major blocks (CPU, RAM, preprocessor,...) are internal on die. So the chip just has raw data inputs, a high speed network interface for the data readout and the scsn slave interface used as configuration unit of the TRAP-1. A detailed description of the TRAP-1 can be found in [14].

All communication on the trap chip is done via the global I/O bus (see Fig. 16). The config unit is bus master of the global I/O. In some cases it is necessary to power up the chip[29] before executing

[27]Very high speed integrated circuit Hardware Description Language
[28]Multiple Instruction, Multiple Data
[29]to save power, not all clocks are switched on all the time

bus transactions. Therefore the application layer of scsn has an additional interface to the *global state machine (GSM)*. By using special read/write commands, the GSM is requested before issuing a transaction on the bus. The configuration entity is part of the *core120*, which contains all CPUs,

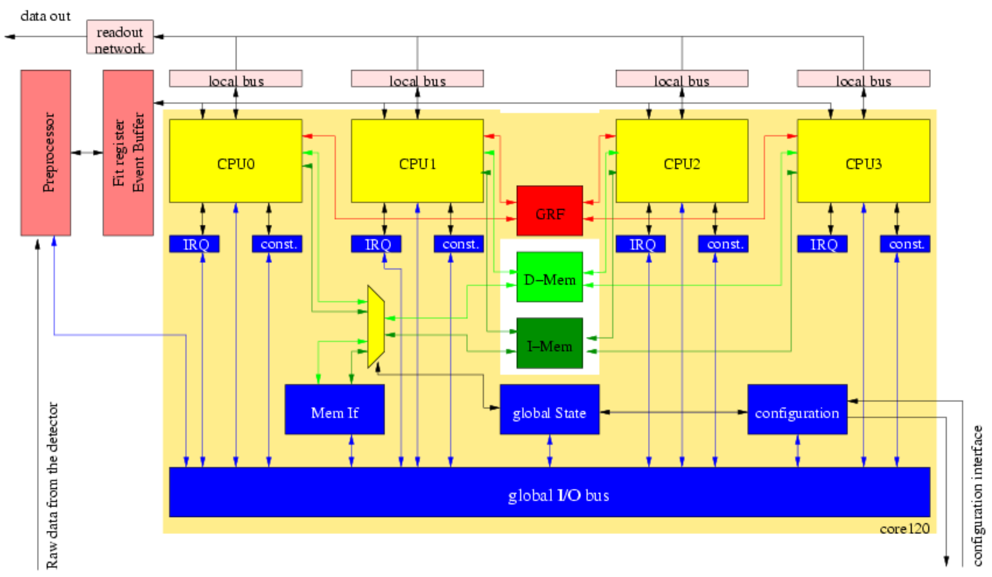


Figure 16: **Trap1 Block Diagram.** This figure shows a block diagram of the four CPUs connected via the global bus with the configuration unit (scsn). All global I/O devices are printed in blue, memories in green and the global register file (GRF) in red. The data and instruction memory can be accessed, using the memory interface to the global I/O. The exact interface descriptions can be found in [14].

and the global I/O, running at 120 MHz (see Fig.16). To access the memory which is not part of the main core, there is an extra device in the global I/O: the I/D Memory-Auto-Incrementer, which is a device of the global I/O and allows successive reads or writes to data and instruction memory.

3.3 scsn - Trap1 implementation specifications

The trap1 will operate in a high magnetic field close to a particle beam, which might lead to possibly flipping bits. Therefore all state machines are protected with a hamming encoder. The hardware is designed that way, that an error will cause the state machine to be reset in a well defined state, so that there is no dead or life lock possible. (see also chapter 4). i

Comprehensive tests have been performed with the TRAP-1's slave interface, without encountering any problems. More on this in chapter 5. On re-engineering the code, a bug was found in the resynchronization state machine of the TRAP-1. Though it is only a minor bug, automatic resynchronization does not work properly on the first Version of TRAP-1.

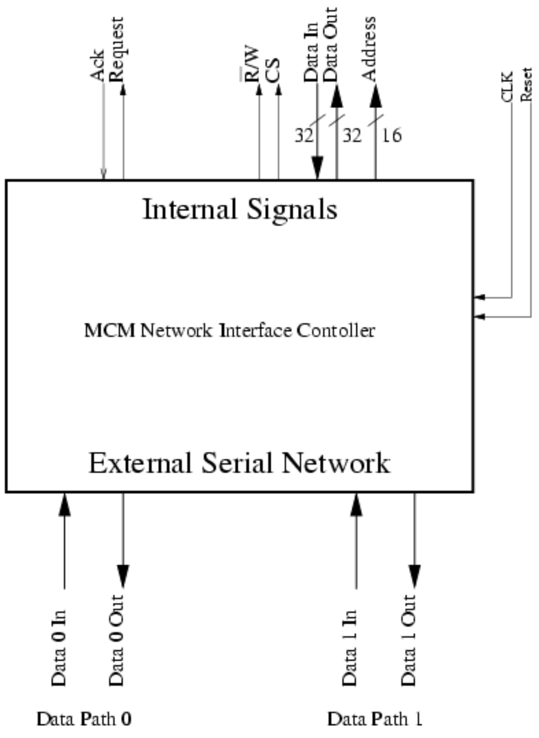


Figure 17: **TRAP-1 configuration unit.** Blackbox view of the scsn configuration entity for each of the the trap1 Multi chip modules. The internal signals are to the global I/O and to the GSM, to request "configuration mode" power on, which is acknowledged by the GSM. The reset input is to force a reset of all buffers and state machines. Usually this is not needed. The TRAP-1 hardware has been designed that way, that it will recover (end go to idle mode) from any possible state it wakes up from, so there does not have to be an initial reset.

3.4 Master Implementations

The master is the interface between up to 126 slaves and a central control computer. Currently there are only prototype and test implementations of a master, since this work is due to Tobias Krawutschke [13].

The Technical Design report recommends using Altera^{TM}Excalibur NIOS softcore FPGA, running μcLinux in the detector chambers which will be used as master for the scsn network. Hardware resources of the Excalibur allow 4 interfaces to be mapped to a single FPGA. Frame handling, as well as network recovery and routing is left to software. The link between the NIOS master and the "outside world" in the TRD will be Ethernet (see figure 3), so the master has to handle all the slaves transparently.

The software has been split into several levels:

clock speed	120 MHz
network speed	1/5 of clock speed. 24 MBit/s
bitstuffing	stuff length: 8 bit
bus width	16bit address, 32 bit data
checksum	CRC-16 Polynom: $x^{16}+x^{15}+x^2+1$
Technology	UMCL18u250t2
scsn Nets / Cells	2285 / 2191
scsn Cell Area	87833 μm^2
scsn Longest Path	1,59 ns

Table 5: **TRAP-1 specifications.** Summary of the hardware implementation of the configuration unit in TRAP-1.

Software - lowlevel driver . The lowlevel driver is a kernel module[30] that provides a buffered interface to the data link layer hardware via a Unix `/dev/scsb*` device file. File I/O reads or write blocks of 9 bytes, the frame specifications are included in the Kernel module.

Software - network master . This program communicates with the slaves by using the lowlevel device interface and provides full master functionality; there is an API[31] that allows sending, and receiving of frames.

Software - man machine interface (MMI) . Eventually, this is the program that allows the user to speak with the the slaves, by calling API functions of the network master. This program can be run remotely to allow multiple masters and therefore up to 65000 slaves to be addressed.

The software is no official part of this thesis, but for testing and debugging issues, all three software layers[32] have been implemented on GNU/Linux using system-independent autoconf, automake, POSIX-C[33]. A documentation can be found in section 4.2.

3.4.1 ACEX Board

Before submitting the TRAP-1 all entities have been mapped to FPGA boards and tested (see section 5), leaving behind lot of designs: network toys, benchmark environments and "master simulators". Most of them were either hard-coded state machines, or simple keyboard/mouse controlled assembler programs[34]. The designs might be interesting for developers and are useful for some demonstrations. The source code and all design files can be found to the Software CD-ROM.

[30] μcLinux is not modularized, so this has to become a monolithic kernel driver
[31] application programming interface
[32] Note: the network master has been implemented without automatic error recovery.
[33] tools for system independent software development
[34] sweet-16 CPU, and assembler by [2]

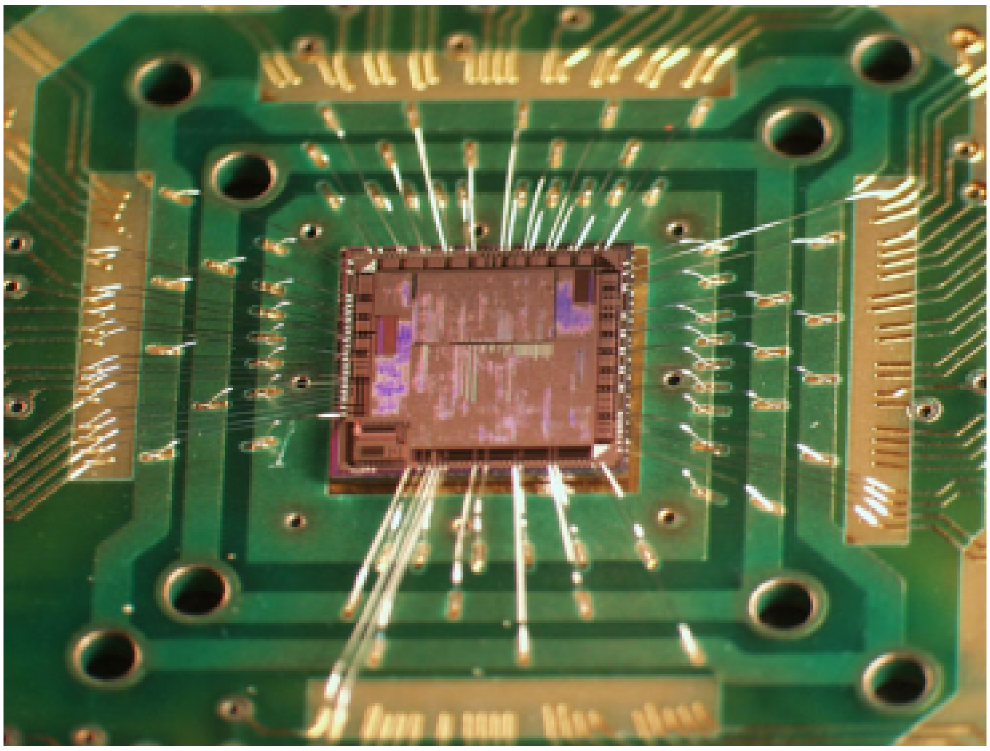


Figure 18: **Trap1 Prototype.** The chip is a 5x5mm square. Bottom left is the (unbonded) analog design (ADC), the large block bottom right contains 19 preprocessor channels, connected to the event buffers (right edge). The block top middle of the picture is the core120 containing the global I/O configuration and 4 CPUs. The blocks on the left and right sides are event buffers, data and instruction memory.

3.4.2 PCI Interface

One of the ACEX designs (the `mcm_excalibur_simulator`) survived all the testing and is used as master-hardware. The ACEX test board can be used as a standard PCI card. The scsn data link layer is connected via a FIFO buffer to a PCI bus interface, using the Altera PCI Soft Core$^{\mathrm{TM}}$. more details follow in 4.1.7.

3.5 Altera NIOS

The NIOS which will be used as a final version in the TRD uses a rewritten version of the `mcm_excalibur_simulator`, all documentation here is due to [13].

4 Design

6802 hackers made use of the SEX instruction.

–found on fortune(6)

This chapter delves into implementation details of the hardware projects, the documentation of the source code of scsn and the manual for the software that have been written within the development.

To gain flexibility, efforts have been made to capsulate the hardware into pluggable modules (entities). It is possible to change any layer of the network device to adopt needs of the application. So here follows the description of the implemented entities and how they fit together using an application programming interface (API) for the different layers in the NIC / bus controller.

4.1 VHDL API documentation

The source code of scsn (mcm_network_interface) can be found on the Software CD-ROM (ISO is available via Internet download [`http://ti.uni-hd.de/`]). This chapter describes the structure, implementation details and state machine algorithms. The source code is well commented, so not all details are described here. Table 6 gives a list of the source files. All functionality is implemented in state machines, which are hamming encoded. The hamming registers, are designed to reset themselves when the hamming decoder detects an error. Thereby the state machines are forced to enter a reset and idle state. This documentation does not include the state machine diagrams, since all the details are in the source code. Documentation has been moved to comments there.

4.1.1 VHDL generic Values

The code was written that way, that almost all semantic units, can be changed at compile time. This does hold good for the network speed and timings, but also counts for the frame length, buffers and filters. All generic values can be set in the top entity (`mcm_network_interface.vhd`), although most values are used only in the Data Link Layer. In addition to the generic values, there few constants have been defined to be even more flexible. Both generics and constants are summarized and described in Table 7 and 8.

4.1.2 Physical Layer

The physical layer is kept simple. A 2 bit flip-flop shift-register, running with local clock is used to synchronize and low pass filter the incoming data.

4.1.3 Data Link Layer

The data link layer is the most complex of all. There are two receiver and two sender units here.

File	Description
`hamm34enc67.vhd`	Hamming encoder
`hamm67dec34.vhd`	Hamming decoder
`hamm_reg.vhd`	Hamming register
`mcm_pci_sender.vhd`	PCI Master implementation
`mcm_network_interface.vhd`	Slave Top Level
`mcm_nw_apl.vhd`	Application Layer
`mcm_nw_bittiming.vhd`	Data Link Layer - receiver deserialization
`mcm_nw_destuffing.vhd`	Data Link Layer - destuffing unit
`mcm_nw_dll.vhd`	Data Link Layer - Top Level
`mcm_nw_inbuf.vhd`	Data Link Layer - input buffer
`mcm_nw_nwl.vhd`	Network Layer - Top Level - merger
`mcm_nw_nwsl.vhd`	Network Layer - sublayer: frame handling
`mcm_nw_outbuf.vhd`	Data Link Layer - output buffer
`mcm_nw_pl.vhd`	Physical Layer
`mcm_nw_sendtiming.vhd`	Data Link Layer - serializer
`mcm_snoop_switch.vhd.vhd`	CPLD design network switch.
`mcm_nw_stuffing.vhd`	Data Link Layer - bit stuffing unit
`mcm_nw_timer.vhd`	simple Counter - reused for send and receiver units
`ser_int.vhd`	serial interface master simulator for the TRAP-1 test environment - reads frames to send from File
`ser_tb.vhd`	Testbench for the serial interface

Table 6: **VHDL Source Files**.

The Receiver See Figure 19 for a block diagram. Main part is a state machine to generate a strobe signal using a counter (bittiming-logic), which is also doing the resynchronization. The exact documentation can be found in the source code, basically a state machine listens on the data from the wire, resetting a counter on each transition. If the counter has a certain value (network speed ratio; see chapter 2.7.2) the data is strobed to the input buffer. The counters are generic, see Table 7. Buffering and deserialization is done by a shift-register. The data is strobed through Flip-Flops using the generated strobe signal, and can be read out in parallel (see Fig. 19). CRC is implemented using XOR gates, a Flip-Flop shift register and a compare to zero unit. See Figure 20 for a sample CRC schematic. The destuffing unit is trivial. It can be transparently inserted in any datapath and will remove stuff bits, with a latency of 1 clock cycle, since the data is buffered in a D-Flip-Flop. The destuffing unit also outputs a strobe signal, which is is inserted between the bittiming and the input buffer.

The Sender Sending works similar; see Figure 21. The data to send may be written in parallel (when buffer_ready via data_in, write_enable) to the output buffer shift register. Transmission is started by rising the initiate_send signal, that starts a counter, which generates the strobe signal on overflow. A counter is used keep track of sent bits and checksum calculation to switch the data to CRC and finally close the stream. The stuffing unit is inserted after the buffer. The strobe signal is

Name	Description
timing_count_range	defines the network speed in fractions of internal clock. Internally the timing is done with a counter that starts counting at zero, up to ’timing_count_range’. Due to limitations of the Data Link Layer state machines, the fastest possible speed is 1/3 of internal clock. so this value has to be >1. (default : 2)
timing_recv_on	timing-offset. After detecting an edge on the data_in, the bittiming waits for (’timing_recv_on’ + 1) internal clocks before strobing data into buffer. This needs to be < ’timing_count_range’ (default: 1)
stuff_length	specifies how many identical bits are sent or received, before inserting/removing a stuff bit. Since a bitstuff error is treated as EndOfTransmission this indirectly defines the timeout. (default: 8) Note: timeout is (stuff_length ·(timing_count_range+1))[clock cycles]
timing_sleep_length	after transmitting a frame, wait ’timing_sleep_length’ clock cycles before allowing to send the next frame. (default: 63)

Table 7: **VHDL compile time generics**.

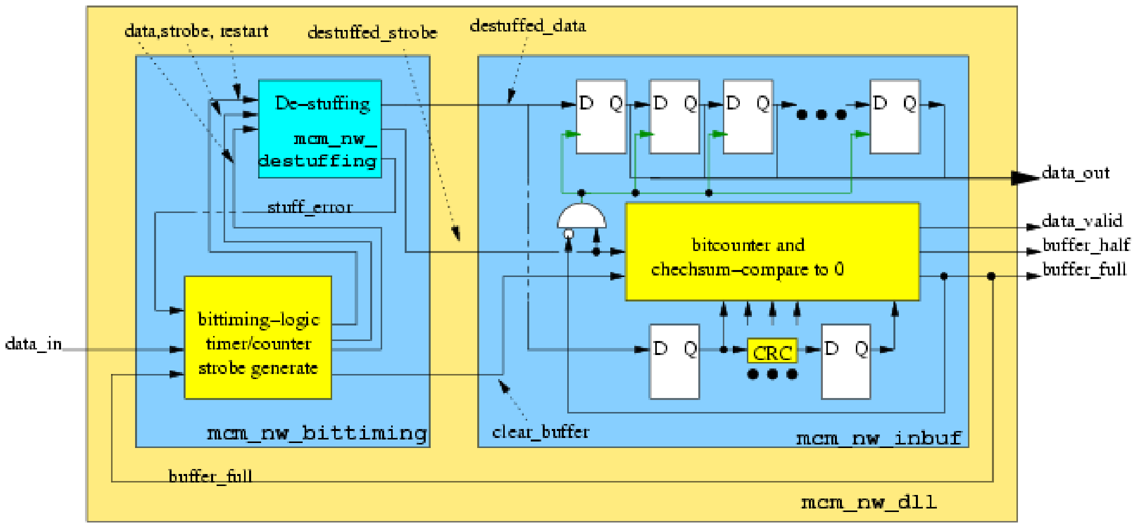


Figure 19: **Data Link Layer Block Diagram.** This shows the receiver part. The filtered data of from the physical layer arrives (left) is deserialized (bittiming), buffered (DFF shift register) and CRC checked (inbuf). The names in the figure correspond to the VHDL file and signal names.

fed though the "stuffer" before causing buffer shifts. Which leads to the fact, that no stuff bit can occur at the end of a transmission. Finally a multiplexer routes the serialized data to the configured

Name	Description
BUFSIZ	size of the input/output buffers. This defines the frame length. Since the network layer hard-codes the header fields it has to be > 16. Changing this value, requires the network and application layer to be adapted. (default: 69)
BUFHALF	"almost full". send error when DLL is still sending out data and 'BUFSIZ'-'BUFHALF' bits of a new frame have already been received. (default: 4)
COUNTER	Since all values are generic. This number represents the bits needed to store the bitcounter. The bitcounter has to count from 0 to (BUFSIZ+CRCLEN) (default: 7)
CRCLEN	Length of the Checksum. (default: 16)
CRC_POLY	Polynom of the CRC algorithm. omit the leading 1. (default: LSB "1000000000000101" : $x^{16}+x^{15}+x^2+1$)

Table 8: **VHDL compile time constants**.

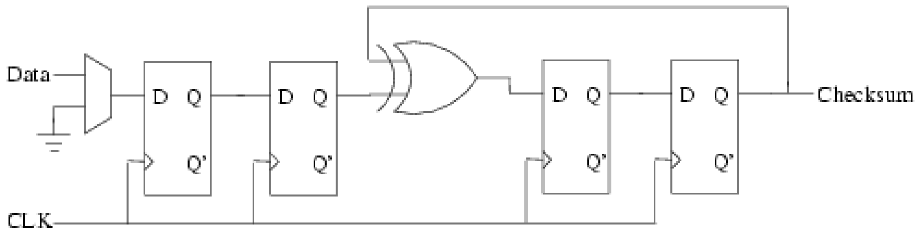


Figure 20: **CRC shift register** This shows a 4 bit CRC register with the polynom x^2+1. The input multiplexer is used to fill the register with zeros when reading out the checksum.

(bridge) output.

4.1.4 Network Sub Layer

The network sub layers take care of the frame handling. For each input/output buffer pair there is one network sub layer. A state machine first analyzes the frame: increment hop counter and compare address. If the frame is to be processed, the application layer is requested, and waited for data to return from. Finally the frame is forwarded by copying the data into the outputbuffer (see Data Link Layer interface). There is an extra state to send an error frame and wait for its transmission completion. If a one or more new frames arrive and the state machine is not in idle mode, the current execution of the frame in application layer is canceled and a single error frame is scheduled.

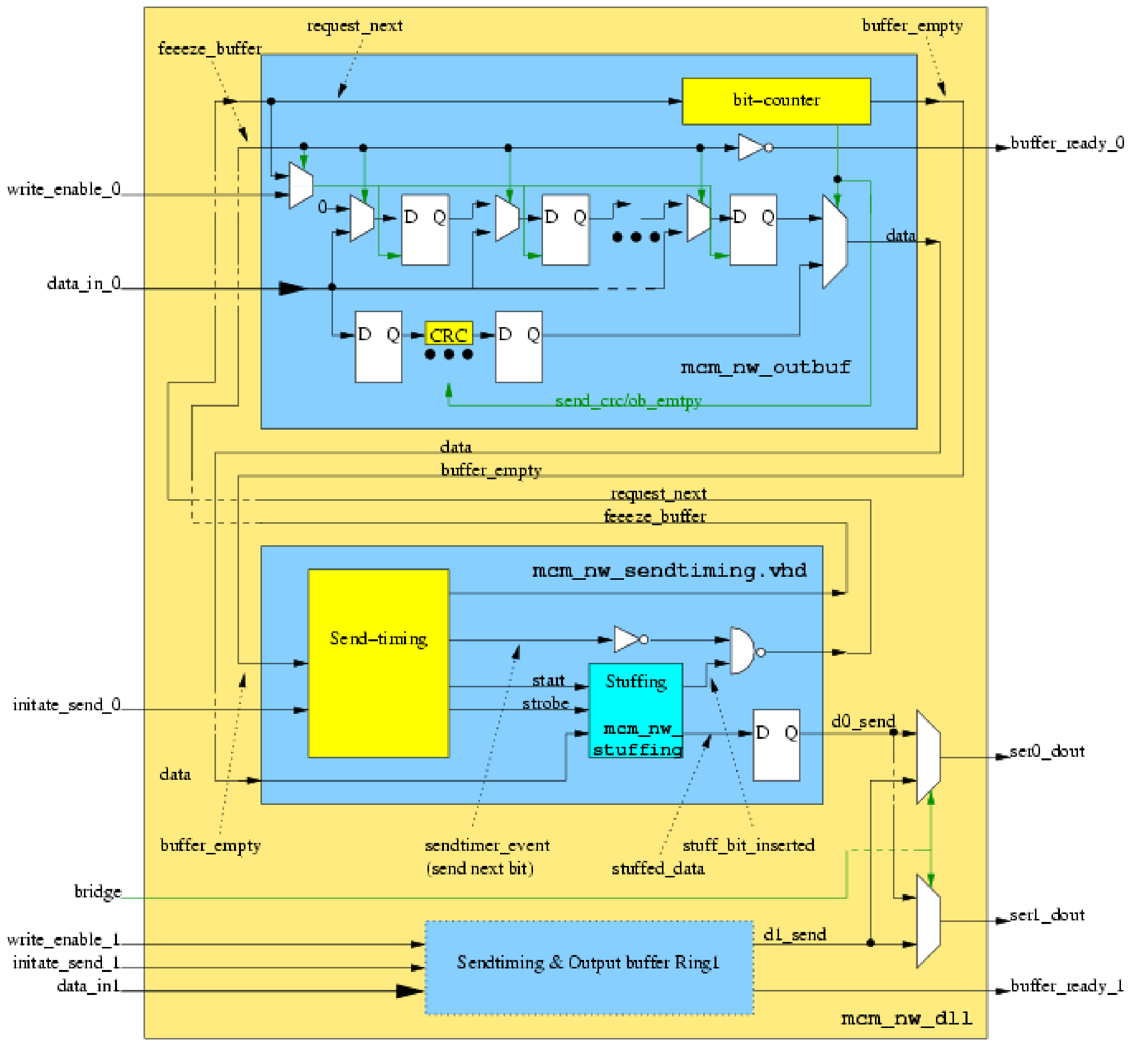


Figure 21: **Data Link Layer Block Diagram.** This shows the sender part. a description can be found in 4.1.3

4.1.5 Network Layer

The session layer protocol of scsn is part of the network layer: The two network sub layers get merged to a single application layer. The network bridge is also implemented here, since a change of the topology can only be done when none of the output buffers are currently sending. To save logic the bridge functionality has been moved to data link layer, since there is is only a small multiplexer what would have been a large here, with the trade off to have slightly more complex logic: The network layer state machine simple waits for application layer requests of each sub layer. First one requesting will be first to be processed. If both request simultaneously, the first sublayer (sublayer 0) will win. If a bridge change is pending, none of the sublayers are allowed to access the application layer (until both outbutbuffers are flushed and the bridge can be switched[35]).

4.1.6 Application Layer

The application layer processes the 52 bit payload. According to the command bis of the data from the network layer, either a bus transaction is requested or an internal state change (bridge, cycle time) is executed. The commands are summarized in Table 9. Finally the resulting answer is strobed to the network layer.

hex	LSB-binary	Description
0x0	0000	network error reply. (invalid checksum, bitstuff error, execution time exceeded)
0x1	1000	read.
0x2	0100	write.
0x3	1100	read broadcast. slave performs read. and if read_data[0] = ’1’ this command is interpreted as read and therefore replied, else the command-request is forwarded to the next slave.
0x4	0010	bridge set/reset. data bit[0] determines state of bridge. 0: serial (non bridged) 1: crossed (bridged) mode
0x5	1010	CYCLE : bit(31) determines the the state of the internal bus transaction mode. 0: single cycle. All bus transactions take 1 clock cycle. 1: long cycle. Bus access times are doubled, to (possibly) allow bus access when the chip has timing problems.
0x6	0110	reserved.
0x7	1110	NOP : no operation (ping).
0x8	0001	reserved.
0x9	1001	read. with requesting test mode from the global state machine (GSM) before issuing the bus transaction.
0xA	0101	write. with GSM request.
0xB	1101	broadcast read. with GSM request.
0xC	0011	reserved.
0xD	1011	reserved.
0xE	0111	unknown Command error reply.
0xF	1111	reserved. (snoop switch configuration see 4.1.8)

Table 9: **Command Set** of the scsn application layer for the TRAP-1.

[35]multiplexing the data before the output buffers would allow processing data at any time, but since a network topology change is performed only when the network is under error recovery, there is no need to support fast frame processing here.

Bus interface The TRAP-1 global I/O is an arbitrated 16 bit address, 32 bit data single cycle bus[36]. Separate read and write data lines are used, to avoid internal tri state busses. The configuration unit is bus master, thus all bus requests will be successful. A write cycle puts data and address simultaneously on the bus with the write enable and bus select set high; a read transaction can buffer the data one clock after requesting the specified address with rising edge clock signal (see [14]). Before issuing the bus transaction the chip can requested to power up, which sets the request signal to high and waits for the acknowledgment. Note that, this request method can also be used as a bus grant signal, when the interface is not a bus master.

4.1.7 Network Master PCI

The PCI master is an FIFO buffered interface to the data link layer of scsn, for standard PCs. A Linux driver for the PCI card and software is available on the software CD. This design uses the Altera PCI Soft Core^{TM}pci core, LPM RAM FIFO buffers running at scsn network speed and some small state machine logic, with local PCI bus clock. The PCI is a 64-bit/66MHz PCI design, using only 32 data bits for compatibility reasons. The data of the frame to be sent is written in 3 steps (69 bit on 32 bit PCI bus). The third write enqueues the frame in the output FIFO. The input buffer can be read out in the same manner, reading the header address flushes the frame of the FIFO. The output buffer is flushed as fast as possible, transmitting at scsn network speed. Newly arrived frames are pushed on the input buffer. If the buffer is already full, or the frame checksum is incorrect, the frame is discarded. The buffer status flags (empty, full) are available via a status register.

4.1.8 Network Snoop Switch

If more than one client in the network breaks, all slaves in between the two broken become inaccessible. To gain more redundancy in the TRD for testing and debugging reasons a "back door" has been created: The needs were to have an easy way to bypass broken clients with less further hardware and without interfering with the given structure and underlying network topology.

A CPLD[37] is used as an extra network bridge. Instead of a slave, a CPLD is inserted in the ring. The CPLD is connected to a sub-ring of slaves, which is either inserted or bypassed in the main ring. The implementation requires the master to know about the network topology, and the network switches. The master configures the switches by sending special frames. These are ignored (and just forwarded) by all slaves, but are snooped off the wire by the switches, which then change their configuration.

This allows to build groups of slaves that can either be bypassed (disabled) or included in the ring. If some of the slaves break, the complete group can be bypassed to gain access to the network. To gain full flexible access, the switches can be reconfigured any time, but the master has to keep track of pending frames, since the CPLD just snoops the wire and does not have buffers at all.

The implementation goal was to have a really small design, that a single CPLD chip can provide snoop switches for multiple sub groups. The current design uses the bittiming unit and needs 52

[36]the TRAP-1 configuration unit supports a failsafe double time bus transaction mode. see Command Set Table

[37]Complex Programmable Logic Device

Flip-Flops to check the frame checksum and count the bits and state machine logic. In addition to that for each sub group, two flip flops to snoop and remember the state (bypassed:0 | active:1) are required, as well as two 2-bit multiplexers, which bridge the sub ring. Bypassed output lines are forces to ground (0). Synopsys™reports 203 cells for a 4 subring design. The switch can be configured by sending a frame with the command "1111" (0xF), and specifying the bridge states in the data field of the frame. The number of subrings is generic, data bit(0) corresponds to sub ring 0, data bit(1) to sub group 1, and so on. Rising the bit activates the bypass. By default all rings are included (configuration "0"). The Header and payload address field of the configuration frame are ignored.

4.1.9 Testbenches

There were mainly two testbenches in use for simulations of the trap1 with Modelsim™. The first one was generated by a small c program reading the values to send via the configuration network from a file, without using any VHDL code. The second (`ser_int.vhd`) uses the VHDL code of a scsn master and also reads the data to send from file. This file is a simple text file, space separated value. Each line is a frame to be sent, in format `destination command address data` in decimal values [38]. The scsb software is able to produce this scripts.

Additionally there is a testbench (`ser_tb.vhd`) to simulate a plain scsn network with multiple clients. During the development process, most of the generic testbenches became obsolete.

4.2 PCI Interface Driver

In order to have nice access to the TRAP-1 chip for testing issues, a PCI-card slow control interface was developed, which gives nice access to user space software, allowing complex test runs.

The internals of the hardware PCI interface can be found in the VHDL documentation. The design has been mapped to an Altera-AXEC-EP1K100 FPGA card using Quartus™Software and the Altera PCI Soft Core™. The Programming/Software Files are available on the Software CD-ROM. See also How To in Appendix B, a picture of the ACEX board can be found in Appendix D.

The Card can be plugged in any Computer with PCI bus[39]. To access it, simply map the cards I/O address space and use the memory map. The TRAP-1 test setup, includes a complete Linux driver set, and API for the scsn. They are capsulated in 3 Layers: The lowlevel driver is a kernel module, handling the pci card I/O, and allows user space programs to access the pci card as device (`/dev/scsb[0-n]`). Above that there is a tool that provides transparent access to the network for read/write transactions. This Tool handles network errors, buffering, network recovery,... And finally a small shell interpreter allows the user to run more complex tests combing read/write transactions, using a nice 'easy to use' UI.

[38] VHDL: `conv_std_logic_vector` is used, so values > 16 bit have to be represented with negative values.
[39] the currently used ACEX FPGA, is a 3.3 V device. So only PCs with a 3.3V PCI bus can be used with this FPGA card.

4.2.1 Linux kernel lowlevel driver

To impart knowledge of the internals can not circumvent citing the program code. For appropriate reasons, the documentation of the lowlevel driver has been included in the source code. The source code is well documented and can be found on the Software CD-ROM. The driver has been successfully tested on Linux-2.4.18 and Linux-2.5.27.

Since this is not a main part of the diploma thesis, here is only a short summary of has been implemented and how it works:

- Kernel PCI lowlevel I/O using hardware interrupts. On load of the driver, the pci bus is scanned for devices. All found cards are activated. An system IRQ is requested, and the pci memory bar is mapped.
- There may be *n* rings per device, each ring has an offset of 0x20 bits in the cards memory I/O space. For each ring, the user space interface devices `/dev/scsb[`$0-n$`]` are generated, where *n* is the number of rings connected to the hardware, which is identical to the minor kernel hardware number (see /usr/src/linux/Documentations/devices.txt [4]). The major device numbers is dynamically allocated. There is a load script included in the software package, that calls `mknod(1)` to create the device files.
- Sending and receiving a frame to ring *n* is simply done by writing to or reading 9-byte blocks from /dev/scsb[*n*]. The device supports system poll functionality to support `select(1)`, and fast interrupt driven lowlevel I/O. The kernel driver knows the frame format and reads/generates data like this: 8bit hopcounter, (7+1)bit header address, 1byte command, (2+4)bytes address and data.
- The driver registers in the system proc filesystem. Reading from `/proc/scsb` generates a status report of the hardware buffers and driver configuration.
- A small benchmark tool is included in the driver distribution, which also provides sample code to communicate via the lowlevel I/O. If called without any arguments it runs a complete benchmark, using all possible FIFO buffer sizes and reports statistics in a gnuplot readable text format. The benchmark program accepts a integer value $0 < n < 127$ as first argument, and then performs a single write cycle sending *n* frames, reporting detailed timing information.

4.2.2 Linux network device driver

This is a collection of ANSI-C routines interfacing to `/dev/scsb[0-n]` and providing an undocumented API (use the Source. The API has become quite obsolete, because the configuration shell (which interfaces to this) provides a better one:

As, until now, not all error recovery functions have been implemented, most of the errors will just ask for a device reset. This work is left as part of Tobias Krawutschke's doctor thesis.[13].

[40] reading this register resets the new-frame arrived IRQ.

Offset	r/w	Description
0x00	r/w	Ring 0 - send: I/O-Data
0x04	r/w	Ring 0 - send: I/O-Command(20-16) I/O-Addr(15-0)
0x08	r/w	Ring 0: enable IRQ
0x0c	r/w	Ring 0 - send: Network slave destination address. writing here also starts sending. address "0" is the reserved id for this master-unit. so first slave in row is addressed with "1". 127 means broadcast to all.
0x10	r	Ring 0 - recv: I/O-Data
0x14	r	Ring 0 - recv: I/O-Command(20-16) I/O-Addr(15-0)
0x18	r[40]	Ring 0 - recv: Network slave source address.
0x1c	r	Ring 0: recv-hopcounter(15-8) Interface-Status(7-0) ; the status register shows the hardware FIFO flags of both interfaces. status(0): input buffer 0 not empty (new frame(s)), status(1): unused, status(2): unused, status(3): output buffer 0 full, status(4): new frame(s) in input buffer 1, status(5): unused, status(6) unused, status(7): output buffer 1 full.

Table 10: **PCI interface memory map.**

4.2.3 Linux network configuration shell

The intention here was to build an easy usable tool for sending commands to scsn devices. This program became quite complex, since it transparently manages the network by forking API threads. On the other hand, it parses user commands and processes them.

The user interface was designed to be as intuitive to use as possible: a command prompt in a shell environment!

- On line Help. pressing `?` immediately outputs a short list of possible commands or options.
- Tab expansion. All commands as well as Data are extended reasonable to the context. By pressing `Tab` twice, a list of all possible options is generated.
- Built in help. executing a command without any arguments, shows a more detailed help function of the command.
- Command line buffer. Previously exerted commands are accessible via ↑ and ↓ keys. The buffer can be searched backwards by pressing `CTRL`-`r`
- Automatization. There shell is script able, and provides commands to produce scripts and log output.
- TRAP-1 global I/O address parser. There is a plug-in that parses the address map of the TRAP-1. It is possible to type commands like `write GSM lowpower` instead of `write 5004 0123` .

Most of the documentation was build in the scsn Tool and in the Source, but here is a usage:

Starting the shell - command mode:
This shell allows to control any scsn-devices "connected" to /dev/scsb*, starting the shell initialized the devices and locks them. The software starts in *command mode* which allows to change default settings, enable debugging, choose an interface to transmit configuration commands, or exit.

Command	Usage/Description
assembler	assembler <file.asm> run TRAP-1 CPU assembler on file. This generates a .o and .mif file for the given .asm assembler code.
channel	[no] channel [console|stdout|timings] select output channels to dump on screen. If called without any arguments, the currently selected channels are displayed. see channels.
command	command [debug|extended|standard|none] undocumented features for debugging facilities. If called without any arguments, the currently chosen command-set is printed on screen.
defaultdestination	defaultdestination [ID] set address for read/write commands in configuration mode. If called without argument, the current value is printed on screen. (default: 1)
exit	exit exit the scsn shell.
gpib	gpib <status|power <on|off> Yet hard-coded access to a power supply using an IEEE488.2 GPIB-ENET interface.Code is under development.
interface	interface n Enter interface configuration mode for interface n.
output	[no] output [csv|csverr|short|full|parse] Set style how to display frames. If called without any arguments, the currently chosen output style is printed on screen (default: parse). parse: print nice (colorful) address/data parsed output. csv: comma separated value csverr: comma separated value to standard error short: print only frame payload

	full: print only frame header and data
pretrigger	pretrigger <0-7> call system set_mem to send send a pretrigger signal to TRAP-1. Pretrigger signals 0-7 and their usage are documented in [3].
print	print [text]+ prints all given arguments on as info text screen. Nice for making comments in scripts.
sleep	sleep <0-99> delay for $0 < n < 100$ 1/10 seconds
pause	pause [0-99] wait for user to press the `enter` key. If a numeric value $0 < n < 100$ is given as first argument, the command automatically continues after *n* seconds

Table 11: **command mode commands.** Short description and usage information of the main scsn commands.

Invoke interface configuration mode:
Typing `interface 0`, selects the first interface and changes the config mode. Right now it's up to the user to directly communicate with the clients via read/write commands. The command set in interface mode has been extended to needs of the TRAP-1 configuration. Data exchange format has been adopted from the simulation testbenches. Some of the command mode commands (e.g. channel and sleep) have also been mapped in the interface command set, to simplify the writing of scripts. Table 12 gives a list and usage information of all configuration commands.

Command	Usage/Description
read	read <addr> [compare-value|X] [(compare-fail)message] sends a frame on the selected interface to the default network destination address, requesting a read of the given address. and displays the result. The address is parsed by a parser-plug-in and thereby the read command is generated. An optional compare-value can be specified. If the read data is not equal to the given value, an error message will be created. If the read instruction is part of a script, the script will be able to generate read-compare-errors. see also: file. On specifying `X` as compare value, the message will be printed always with the arrived data.
write	write <addr> <data> [bool-OR-data]*

	issues a write transaction (see read). According to the given address, the data argument(s) are parsed to reasonable values. Appending more than one data argument, disjunctive combines (wired-or) the values.
alter	alter <addr> <set|clear|or|nor|and|nand|xor|xnor> <HEX|DEC|[BIT]+> read address, alter data and write back. Set and clear accept up to 32 integer arguments (value 0-31); all logic operations accept a single hexadecimal or a decimal value (as for all data: decimal values have to be prepended with an ampersand ’&’). .
aread	aread <startaddr> <endaddr> [compare-value] [compare-fail-message] read (and compare) memory region.
assembler	assembler <file.asm> run TRAP-1 CPU assembler on file see command mode commands.
awrite	awrite <startaddr> <endaddr> <data> fill memory area with given data.
close	close close all open channel log files. This closes files opened with save and net-dump.
channel	[no] channel <console|stdout|timings> select output channels to dump on screen. see channels and command mode commands.
dmem	dmem <read|write|compare|dump> <baseaddress> <endaddress|filename> access the TRAP-1 data memory via the address auto incrementer. write and compare: specify startaddress and file to store/compare the values read or dump: specify start and end address of memory region to read. Dump produces a hexdump, while read displays all single read frames.
done	done leave interface configuration mode.
file	file <filename> read commands from file. save generated scripts can be reread with the file command. The parser reading the script file is currently being rewritten, to allow shortcuts and special script commands. Use the built in documentation.

imem	imem <read|write|compare|dump> <baseaddress> <endaddress|filename> nice access to the TRAP-1 instruction memory via the memory auto incrementer. The syntax of this command is identical to dmem.
netdump	netdump [-a|-o] <filename> dump all sent/received frames to a file using internal channels. This generated file can be used as input for the scsn simulation testbench. -o overwrites a file if it exists, -a appends to an existing file. The file must be closed with the close command.
pause	pause [0-99] wait for user to press the `enter` key. If a numeric value $0 < n < 100$ is given as first argument, the command automatically continues after n seconds see command mode commands.
ping	ping Send a dummy frame to benchmark and check the network. This command activates the timing channel to report timing statistics.
print	print [text]+ prints all given arguments on as info text screen. see command mode commands.
repeat	repeat <0-99> <...> repeat a given command $0 < n < 100$ times.
reset	reset reset internal buffers and interface configuration. This also sends a ping frame to discover the network, and might be useful, since automatic recovery is not completely implemented, and yet disabled by default.
save	save [-o|-a] <filename> log all subsequent commands until the close command to file. -o overwrites a file if it exists, -a appends to an existing file. see close and file.
scan	scan <startaddr> <endaddr> clear, set and read memory region. Display data bitwidth of the read address.
send	send <dest> <command> <addr> [data]

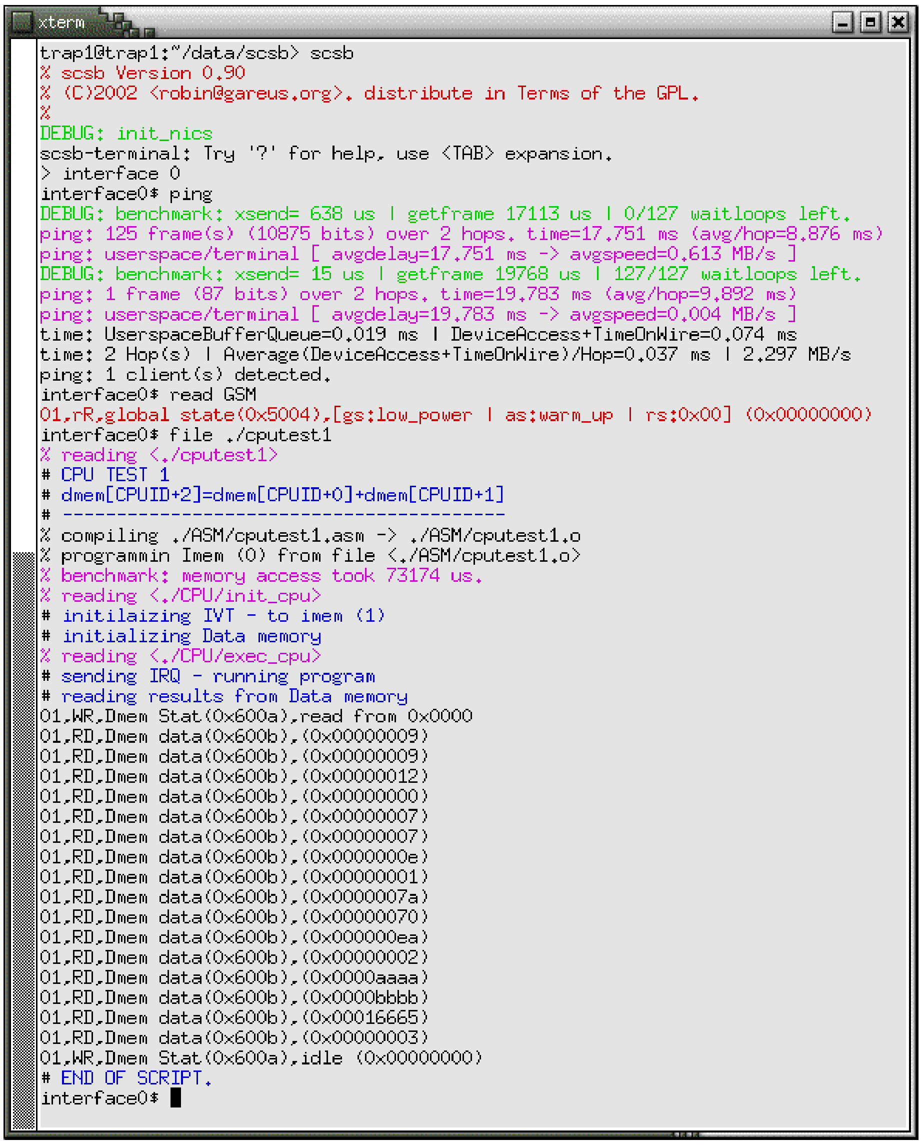
	send a frame to network destination.
sleep	sleep <0-99> delay for $0 < n < 100$ 1/10 seconds see command mode commands.

Table 12: **configuration mode commands.** Short description and usage information of the configuration commands .

Channels The scsn shell provides an abstract communication layer, that is based on communication channels. All log messages and command output are piped into different queues (called channel). The user decides how these channels are displayed or handled. A channel can be connected to the local console (standard output), syslog, a file or any combination of these, including none (discard messages). The *[no] channel* command can be used to attach channels to the current terminal. The write and netdump commands use channels to write output to files.

The current version provides three user accessible channels: *console*-, *stdout*-, and a *timing* information channel. The internal buffer structure includes time information of each frame in the buffer; when flushing the buffer, this information is piped to the timing channel. The *console* and *stdout* channels are special, since they are internally identical: Both contain the same information (the standard output), if at least one of both is connected the standard output is shown on screen. If both are enabled, the information is shown only once. This allows the scripts to suppress output (by disabling the stdout), and the user can still force it to be printed by using the console channel.

Programming internals The scsn forks a complete buffer interface process, that connects to the kernel device and provides software buffering of frames. The buffers contain relevant information to provide a network layer functionality and network recovery information. The user front end simply access these buffers. The different commands are mapped to a combinations of buffer functions. The initial intention to keep the API separated from the user front end failed, since lot of complex functions have been reused in the buffer code, and direct linking was easier to implement in a development version. The source code is distrubuted in terms of the GPL, and a MMI is needed for the TRD, so it seems that there will be other Versions or rewrites of this software, soon.



```
trap1@trap1:~/data/scsb> scsb
% scsb Version 0.90
% (C)2002 <robin@gareus.org>, distribute in Terms of the GPL.
%
DEBUG: init_nics
scsb-terminal: Try '?' for help, use <TAB> expansion.
> interface 0
interface0$ ping
DEBUG: benchmark: xsend= 638 us | getframe 17113 us | 0/127 waitloops left.
ping: 125 frame(s) (10875 bits) over 2 hops, time=17.751 ms (avg/hop=8.876 ms)
ping: userspace/terminal [ avgdelay=17.751 ms -> avgspeed=0.613 MB/s ]
DEBUG: benchmark: xsend= 15 us | getframe 19768 us | 127/127 waitloops left.
ping: 1 frame (87 bits) over 2 hops, time=19.783 ms (avg/hop=9.892 ms)
ping: userspace/terminal [ avgdelay=19.783 ms -> avgspeed=0.004 MB/s ]
time: UserspaceBufferQueue=0.019 ms | DeviceAccess+TimeOnWire=0.074 ms
time: 2 Hop(s) | Average(DeviceAccess+TimeOnWire)/Hop=0.037 ms | 2.297 MB/s
ping: 1 client(s) detected.
interface0$ read GSM
01,rR,global state(0x5004),[gs:low_power | as:warm_up | rs:0x00] (0x00000000)
interface0$ file ./cputest1
% reading <./cputest1>
# CPU TEST 1
# dmem[CPUID+2]=dmem[CPUID+0]+dmem[CPUID+1]
# ---------------------------------------------
% compiling ./ASM/cputest1.asm -> ./ASM/cputest1.o
% programmin Imem (0) from file <./ASM/cputest1.o>
% benchmark: memory access took 73174 us.
% reading <./CPU/init_cpu>
# initilaizing IVT - to imem (1)
# initializing Data memory
% reading <./CPU/exec_cpu>
# sending IRQ - running program
# reading results from Data memory
01,WR,Dmem Stat(0x600a),read from 0x0000
01,RD,Dmem data(0x600b),(0x00000009)
01,RD,Dmem data(0x600b),(0x00000009)
01,RD,Dmem data(0x600b),(0x00000012)
01,RD,Dmem data(0x600b),(0x00000000)
01,RD,Dmem data(0x600b),(0x00000007)
01,RD,Dmem data(0x600b),(0x00000007)
01,RD,Dmem data(0x600b),(0x0000000e)
01,RD,Dmem data(0x600b),(0x00000001)
01,RD,Dmem data(0x600b),(0x0000007a)
01,RD,Dmem data(0x600b),(0x00000070)
01,RD,Dmem data(0x600b),(0x000000ea)
01,RD,Dmem data(0x600b),(0x00000002)
01,RD,Dmem data(0x600b),(0x0000aaaa)
01,RD,Dmem data(0x600b),(0x0000bbbb)
01,RD,Dmem data(0x600b),(0x00016665)
01,RD,Dmem data(0x600b),(0x00000003)
01,WR,Dmem Stat(0x600a),idle (0x00000000)
# END OF SCRIPT.
interface0$ █
```

Figure 22: **Screenshot** of the configuration utility

5 Tests and Measurements

jah work is never ever done.

–Ben Harper

Finally, here follows the documentation of the measurements and benchmarks done of the scsn network. The complete development process has been iterated several times after testing the components of the network to ensure stability and increase performance.

5.1 Physical Layer Measurements

The most basic speed limitation is the underlying physical layer. Signal transport in time and space over a electrical wire limits the bandwidth and restricts the performance of the network.[9] The LVDS circuit provides a nearly ideal termination to the differential transmission lines. This termination permits the circuit to reliably drive data at speeds of up to 622 Mb/s. Data and clocks can be transmitted over longer cables exceeding 5 ns in electrical propagation delay, limited only by the quality of the cable, namely the cable attenuation caused by skin effect losses at high frequencies[12]. Tests ranged from "no wire" (master and slave on a single FPGA) to 10m plain copper (Klingeldraht) in between two FPGAs running at different clock rates up to 40Mbit/s, and have all been successful. Figure 23 shows a LVDS signal transition.

5.2 Data Link Test Setups

Synchronization and network timing have mostly been simulated, using Modelsim^{TM}. There seemed to be a problem with the state machine algorithm when using certain generic values for the counters. There were few tests to proof it working and measure the maximum possible clock jitter. The TRAP-1 does have a bug in the resynchronization state machine, so test setups had to be done with ACEX boards.

Measurements brought up, that in case of sender and receiver clocks oscillating around some slightly different values, the resynchronization state machine can also start to oscillate. This bug was fixed by limiting synchronization loops. A rewritten version now does not reset the counters when detecting a transition, but inserts a leap second or skips one counter clock. Actual tests have been done with the broken TRAP-1 (30 MHz; 1/5 network clock ratio; 6Mbit/s) connected to a PCI master (36/2 MHz; 1/3 network clock ratio; 6Mbit/s). The clock of TRAP-1 can be changed by 0.58% before frames start getting lost. Clock differences above 0.68% lead to loosing of the link. The new resynchronization algorithm has only been tested using a fixed clock ratio 33.000MHz / 33.333MHz $\equiv\approx$ 1% clock jitter. Two ACEX boards (33.000MHz, 33.333MHz; 1/3 network clock ratio; $\approx$ 11.0Mbit/s) could communicate fine. No frames were lost during a 24 hour test.

5.3 Frames on the wire

To analyze the load of the network, and test its functionality, it was most useful to attach a scope to the wires in between the clients. Figure 24 shows a typical frame being processed by a slave and

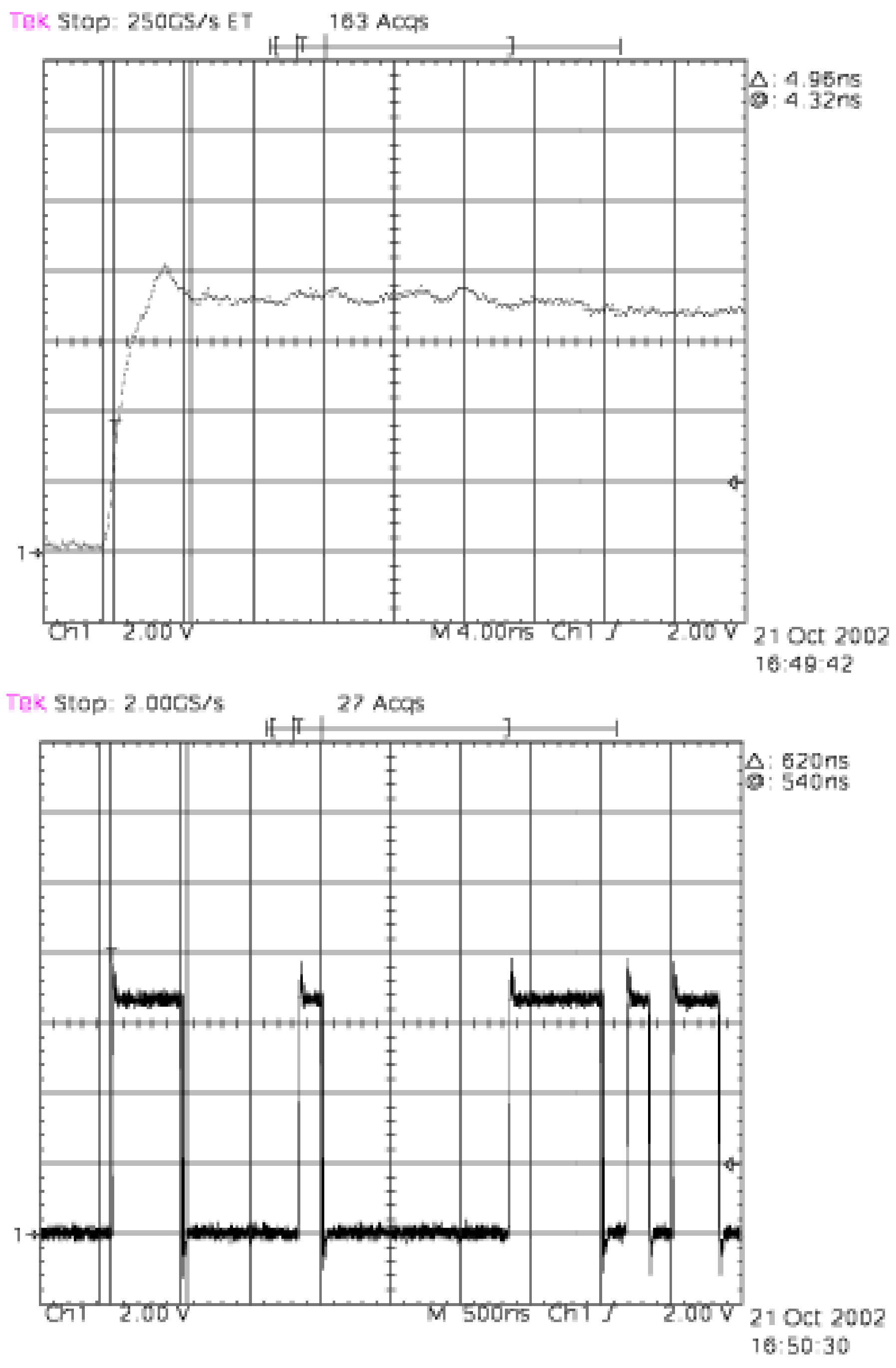


Figure 23: **LVDS 0/1 transition.** This Signal was traced off a scsn frame using a 1.5m twisted pair cable and standard "off the shelf" header connectors (the LVDS standard recommends using 100 mil IDC connectors).

the answer that is sent shortly after the frame has arrived.

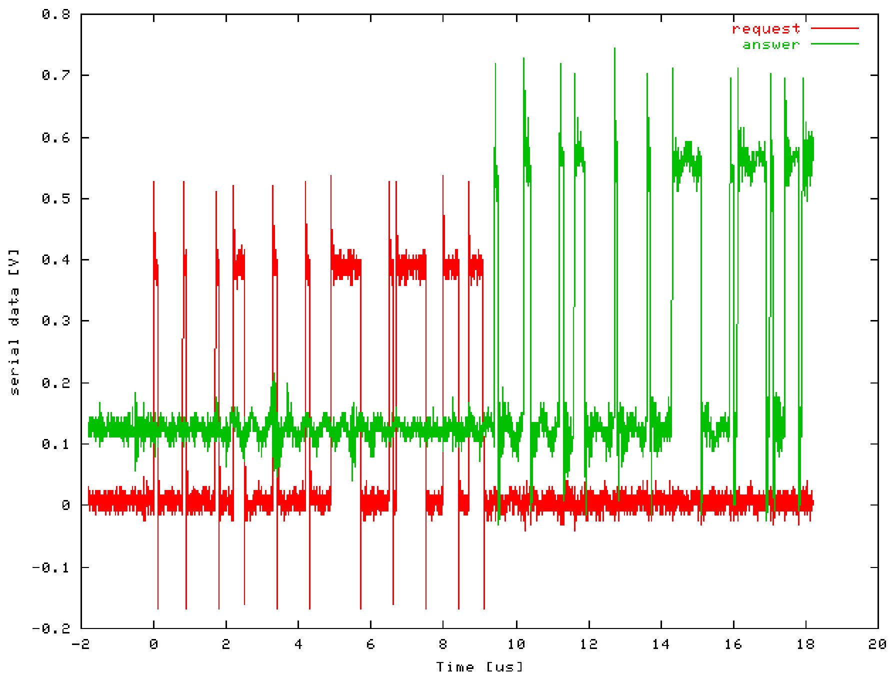


Figure 24: **Frames on the wire.** Frame and it's answer being transmitted over the wire.

To test the network layer functionality, checksum algorithm and general functionality, some long test runs have been performed, trying to send any kind of possible frames over the network. Since the payload is 69 bits, it would take too long to try sending all frames after each other. All extensive tests have been performed using up to 3 ACEX boards (with own clock oscillators). And a maximum chain of length 6.[41] The longest successful test ran over a weekend ($\approx$ 72 hours) and successfully transmitted 43217348832 frames (50MHz; 1/3 network clock ratio, 16.6Mbit/s). The first incorrect frame was detected, when the "Tester" switched on the light, causing the power supply to flicker.

5.4 Lowlevel Benchmarks

The linux kernel driver has been benchmarked in a loopback mode and also with a real scsn network. The frames are each 9 byte data blocks, which is a very small granularity for the linux operating system. This causes the system overhead to limit the bandwidth. Loopback mode benchmarks simply measure the time, the operating system takes to process a transmission.

Measurements have been done with the benchmark tool (see chapter 4.2.1) that simply repeats sending 1 up to *n* frames and wait for them to return. The benchmark tool is running in user-space and uses raw device access to the kernel driver. Benchmarks on a dual AMD-Athlon 1600 MHz resulted in < 2.300 MBit/s sending a single frame and up to 99.037 MBit/s sending 127 frames in

[41]chains up to length 127 have been simulated using modelsim.

parallel.

Using the TRAP-1 prototype test environment which is described in section 5.5, realtime benchmarks have been done; with the result, that speed is heavily limited when not using the input/output buffers and burst writes. The hardware FIFOs of the PCI design should be increased[42] and the kernel driver has to be checked for waitstates.

The theoretical maximum of 6Mbit/s throughput has been approached up to average 4.6Mbit/s. The average transmission speed varies with the output FIFO buffer usage of the master. Furthermore, the interframe space of the pci master is conservatively[43] above the hard edge limit (here: 63 clock cycles delay, min. 24 $\equiv$ stufflength/network clock ratio), and the data exchange rate is reduced by inserted stuff bits. So the actual maximal data transport speed is $<$ 5.2 Mbit/s. This rate is approached when using the output buffer for burst transactions.

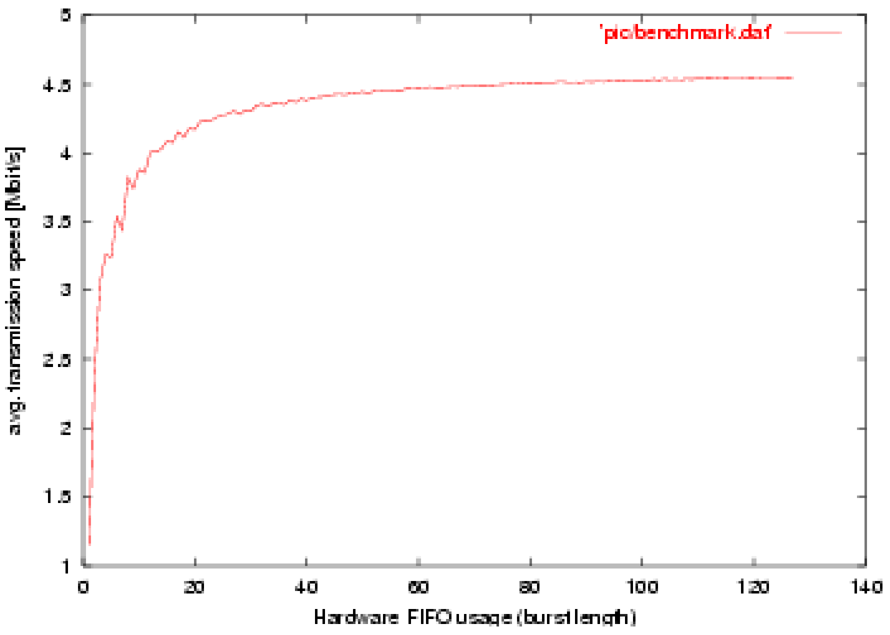


Figure 25: **Speed benchmark depending on buffer usage.** Output generated with the benchmark tool. Frame bursts of 0 to 127 frames have been timed. This figure shows an average of 1389888 totally transmitted frames.

5.5 Application Benchmarks

The tests so far have considered the lowlevel hardware, but no real time application. To really learn about performance, the design has been benchmarked with the PCI card master (30 MHz;

[42]the ACEX card which is currently used, is too small to hold buffers with depth > 127

[43]it may occur, that due to a checksum recalculation and hop counter increment, the stuffed frame grows. Execution time in application layer may also differ, so the choice was, to give it the maximum time needed to be on the safe side. This can be optimized even more, when considering frame and client internals.

1/5 network clock ratio; 6Mbit/s) and the TRAP-1 prototype. Test environment is a GNU/Linux operating system[44], scsn kernel device driver and the scsn user interface application.

The network was designed, to configure the TRAP-1 chip, which in general is a lot of broadcast write traffic, that can be sent in larger units than single frames. Instruction and data memory as well as global I/O configuration writes are examples for that kind of very fast traffic. The data can be enqueued to the FIFO, and after all frames have returned, they are checked in parallel.

Real Time applications using the network for chip configuration and communication often take decisions of how to proceed upon results of read/write requests. In most of these cases, a single frame is sent and the software waits for it to return. Depending on the number of slaves in the ring the maximum latency for a frame to return is (clients+1) $\cdot$ (time to transmit a single frame). The operating system has to access the PCI card (sending is fast) and wait for a interrupt and user-space read request. The average transmission speed sending and receiving a single frame is 1.076 Mbit/s. User-space terminal i/o and file access reduce the speed even more. A typical scsn software configuration run takes $\approx$ 2 seconds[45].

The current software does not allow two identical frames to be enqueued at the same time. Therefore all read requests addressing the auto incrementer are sent separately, which drastically increases instruction/data memory access times (> factor 4). Table 13 shows times for typical user-space configuration commands. The measured time is the real time from command execution start until end of the output and command prompt return. So it is not appropriate calculate a overall configuration speed (which would be around 512 kbits/sec average here).

Operation	Time [μs]
read or write any address (1 frame)	19,841 μs
read and dump memory area 0x000 - 0x100 (256 frames)	37,364 μs
write program to instruction memory (76 frames, some identical)	158,734 μs
scan bit-width of a memory region 0x00 - 0x100 (300 single frames)	1,016,834 μs
read and dump instruction/data memory 0x000 - 0x100 (258 frames)	3,831,840 μs
complete TRAP-1 selftest program	5,559,000 μs

Table 13: **scsb real application benchmark.** This table summarizes the transaction speed of common configuration commands. All measurements have been made with a 6Mbit/sec scsn network, using the Linux software suite.

[44]The PCI card is plugged in 3.3V / 33MHz 32-bit PCI bus (chipset AMD-760MP) on a dual AMD-Athlon 1600.092 MHz, running Linux 2.4.18-4GB-SMP.

[45]Note: the test network is yet running 4 times slower than the final TRAP-1 in the TRD.

6 Conclusions and summary

...und wenn du dieses Buch gelesen hast, dann binde einen Stein daran und wirf es in den Euphrat.

– Jeremia 51,63

6.1 Summary and Outlook

Nowadays, with electronics becoming faster, smaller and more complex, there is a great need for adequate network technologies, which transparently[46] provide access to larger structures. With FPGAs and CPLDs being available off the shelf, it is time to think about new possibilities. Currently existing networks were designed for communication between microcontrollers or computers, and in many cases also fit the needs of FPGA communication.

The Transition Radiation Detector hardware has been tested on FPGAs and is implemented in ASIC. The Man-Machine-Interface(MMI) is a standard Computer. The underlying structure uses Ethernet and scsn to communicate with the hardware. As discussed in this diploma thesis, there was no known network that fits the TRD's needs. Although it has been tried to generalize scsn it does not claim to be a universal FPGA network.

But it seems that scsn has some unique features and unprecedented applications. It provides a reliable fixed speed communication between a limited number of clients, using a simple network structure. Data information is exchanged by issuing read/write transactions on an abstract bus interface. This is leaving a large field for applications. Designs for use in FPGAs and the VHDL source code of scsn are freely available. Software tools and drivers are distributed in terms of the GPL[47].

Measurements and Benchmarks proof the reliability of scsn, but also point out some weak points, that have to be reengineered:

- Resynchronization is not really needed, since clocks show enough accuracy.
- Bitstuffing is nice and useful, but not necessary.
- The frame header format is hard-coded in VHDL. To change the frame format, the bit assignments have to be done by hand. This can become better generic VHDL code.
- Snoop switch protocol should be extended, to allow to ping and detect switches. It should be become possible to build larger hierarchic plain scsn networks.
- Current master buffer sizes are too small, but this is application specific, anyway.

Table 14: **Weak Points of the current scsn implementations.**

Finally Table 15 lists a comparison between scsn similar networks that have been considered for use

[46] transparently means, invisible to the user

[47] GNU General Public License [10].

in the TRD:

	scsn	i^2c	SMBus	10Base2 802.3
Transmission	asynchron	synchron	synchron	asynchron
Wires	2 (1[48])	2	2	1
Speed	40 Mbit/s	2Mbit/s	100kBit/s	10 Mbit/s
Minimum Clock speed	DC[49]	DC	10kHz	-
Maximum Clock speed	96MHz/120MHz[50]	2MHz	100kHz	-
max clients per segment	generic (127)	127	127	100

Table 15: **Comparison of proprietary field busses and networks with scsn.**

6.2 Future Applications - Trap2

Although there were requests to generalize and more standardize scsn as a configuration network, the first major milestone of scsn is still the Transition Radiation Detector. The design and software has been optimized for expected traffic and configuration commands of the TRD. The current design of the scsn unit in TRAP-1 has been revised and the following changes have been committed:

- New resynchronization logic. A bug in the TRAP-1 bittiming state machine has been fixed.
- Extended application layer interface. The application layer becomes more application specific by including an direct interface to the data and instruction memory of TRAP2.
- Some global I/O DFF-memory has to be added to the global I/O bus for a) broadcast-read checks and b) debugging/testing issues. So far other unused registers have been abused for this.

Apart from changes for the TRAP2, scsn is further developed and tested. A 64 bit version of the bus has been simulated and a "future-save" standardized protocol version is worked on. An alternate master implementation for FPGAs was written from scratch according to the scsn architecture specifications by Tobias Krawutschke et al.[16, 13]. Furthermore, the software tools are under constant development and there are rumors that the developer is secretly experimenting with a wireless physical layer.

[48]The scsn network can run with a single data wire, but current implementations use a differential signal.

[49]Some FPGAs require minimum clock speeds.

[50]FPGA: 96MHz / ASIC: 120MHz. The PCI card master is limited due to FPGA speed. Using a faster FPGA than the ACEX allows to increase speed. The TRAP-1 is a .18μ UMC process. The design has been stopped optimizing at 120MHz.

Appendix

A CD-Rom

Here is a overview of the Software on the Software CD-ROM:

```
CD-ROM
|-- ACEX
|   |-- digital_scope
|   |-- lib
|   |-- pci
|   |   |-- core
|   |   `-- programming_files
|   |-- scsb_debug
|   |-- terminal_sender
|   |-- terminal_sof
|   |-- terminal_trace
|   `-- trap1_debug
|-- software
|   |-- other_trap1_software
|   |-- pci_linux
|   |-- scsb-0.91
|   |-- sweet16
|   `-- texdiff
|-- thesis
`-- trap1
    |-- top_sim
    |-- top_gatelevel.zip
    `-- trap1_scsn
```

A.1 ACEX

All designs for the ACEX boards, including the KIP[51] library (*libkip*) have been copied to a separate directory. All temporary data and compiled designs are left in these folders. So it should be easy to just reopen the project files with the according software. Version information about the used tools is available in the sources and scripts on the CD.

The `digital_scope` Folder contains a design to use the ACEX board as a digital oscilloscope. This design was reused and extended in the terminal tracer. The lib directory contains commonly used hardware designs for the ACEX, such as a VGA module or Keyboard/Mouse interfaces, written by R. Gareus and V. Angelov[1].

Subdirectory `pci` contains the PCI design, to be used with the Linux kernel driver. The pci core has been separated from the scsn files. The most recent design, stored in the main folder, is a

[51]Kirchoff Institute for Physics

Quartus^{TM}II.1 project. There were problems with the pci core, when compiling this design with other software. Old ACEX-programming files have been stored in a subdirectory. Note that the ACEX LPM FIFO is too small to provide buffers for both rings. So the current design contains a PCI and scsn interface for two rings, but buffers for only one. For safety the second ring has been disabled. (comments are in the source; it's easy).

`scsb_debug` and `trap1_debug` contain older designs that have been used to do "real" simulations and tests. The `trap` folder contains a reduced version of the TRAP-1 hardware for FPGA, before it has been implemented in $.18\mu$ technology. `scsb_debug` directory includes designs from the initial development, that might be useful when iterating the engineering process for a new scsn release. They are mainly undocumented.

The last ACEX designs have been made for an eye-catching presentation at the "Heidelberger Wissenschaftsmarkt". Two ACEX cards are connected together by only two wires (1 scsn ring) one ACEX card (terminal_sender) is connected to a keyboard, and the other one has a mouse and a VGA Screen connected. The keyboard ACEX uses the sweet16 CPU and an assembler program to interface to a scsn master. On the other ACEX board (`terminal_trace`), the slave's bus interface is directly connected to the VGA video buffer. The video memory is split into two displays. One showing the data the user typed on the keyboard. and another one, that is connected to a signal trace design (also on the ACEX), that traces the LVDS signal off the wire on screen.

A.2 software

Here is the Linux Software to be used with the PCI design. Since the software emulates a bus the development name *scsb* has been kept for the software. A linux kernel module, including makefile and load scripts can be found in the `pci_linux` directory. The user space program requires pthread libraries[52] and has been developed in a POSIX[53] autoconf/automake environment. The autoconf/automake suite is required to compile this source on other systems than Linux. Otherwise scsb can simply build by typing `./configure; make install`.

Furthermore, the software folder contains some c programs that have been used to generate a hardcoded testbench and other small hacks for scsn debugging in folder `other_trap1_software`. The assembler (DOS executable) to compile sweet16 code is included by friendly permission of V. Angelov [2].

Finally the texdiff folder, contains bash scripts to mark differences between two latex files. Execute `texdiff <tag/revision>` in the source directory of the latex files. It will commit the current files to a CVS repository, find the differences between the actual and a previously tagged release, and patch the files to mark differences with LaTeX diffbars. Since the diffbars are not available in all TeX environments `texclean` is used to fix all LaTeX errors, that are caused by the diffbars. Both tools are not yet for universal use, and have been specialized to work with this document. `texdiff` requires few LaTeXcommands to be defined. see `diploarbeit.sty` style file.

[52]GPL thread library
[53]Portable Operating System Interface

A.3 thesis

This folder contains the LaTeX source, figures and pictures of this document. If the university does not claim rights, I would like this document to be distributed in terms of the GNU Free Documentation License (FDL) [10]

The source includes a makefile for GNU Make (GNU Make version 3.79.1), and uses latex (TeX, Version 3.14159 (Web2C 7.3.7)), pdflatex(pdfTeX, Version 3.14159-1.00a pretest 20011114 ojmw (Web2C 7.3.7)), figtodev (xfig 3.2.3d), gnuplot (Linux version 3.7 patchlevel 1), convert (ImageMagick 5.4.2), and optionally some psutils p17-501. The complete document (including the plots) can be rebuilt by just typing `make`, which generates the files `main.pdf main.ps book.ps`. book contains a Din-A5 printable version of the main document.

Other make options are `make [ all | showpdf | show | main.ps | main.pdf | book.ps | book.ps.gz | version | clean ]`. *showpdf* builds the pdf file and launches the acrobat reader. same with *show* that calls gv with the postscript being build. *all* is just the same as calling `make` without any arguments and builds all of the files `main.[pdf|ps] book.[ps|ps.gz]`. *clean* removes all generated files as well as temporary and backup data in the source directory. There is a build counter (file `build.cnt`, comments in `diplomarbeit.sty`) which is incremented every time a changed source-code is compiled. An increment can be forced by issuing a `make version` or by editing the counter file.

There is additional software to mark changes between two releases of this document, using CVS version control. These scripts are in the software folder on the CD.

A.3.1 Trap1 sources

Finally here are the TRAP-1 design files. As a spin-off of the scsb implementation, the whole TRAP-1 design has been simulated by the author. So this Folder contains the complete TRAP simulation source (Folder `top_sim`), which includes the slow control source. Additionally to the functional simulation, a gate-level simulations have been done, source files can be found in a zip file. Again the source, and its final resource Files, including reports and build scripts for Synopsys^{TM} have been separated and sorted in `trap1_scsn` (this is directly copied from the final TRAP-1 backup tape).

B How To

Here is a short description about how to set it all up without really having to know about internal details. As prerequisite at least one of the ACEX boards, or knowledge of how to build and map the designs to a FPGA that is essential.

The user design, that interfaces to scsn should provide an internal bus to the application layer of scsn. This can also be a simple buffer connected to a LED for testing. And indeed, there is no need to have a slave anyway. Simple tests can be done by looping the signal back to the master without any clients. This can be done in the top level design of the master in FPGA, or on physical layer via wires.

B.1 Setup Master

So first of all, a master is needed to initiate data transfer. It is recommend using the PCI card and software, but since the ACEX card is currently only available for 3.3V PCI busses only, there are few designs that require no other hardware than the FPGA, but are mostly undocumented, so far.

ACEX - PCI card use the compiled `sof` or `.pof` Programming files from the CD (or recompile the source), and configure the ACEX. To be flexible with the transmission speed, network clock and PCI clock have been separated. To use the ACEX-PLL for the network clock a small change to the board has to be applied. Note that the design on the Software CD-ROM uses a pinout corresponding to these changes.

The output of the FPGA pin 39 has to be connected to the PLL input of the FPGA. Furthermore, the PCI clock has not to be routed to the PLL input but to some other clock input. This is easily be done by removing R500, R501, R502, and putting a 0Ω resistor in between R500/R502 (PCI_sys0 to FPGA_pin 71). The oscillator clock is routed to the PLL by soldering a wire from the ADC connector Pin1(FPGA_pin 39) to the common pad of R500/R501 (ACEX PLLCLK pin 79). see Fig. 26 and the ACEX schematics available at [1].

Wiring Before plugging the card in, LVDS cables have to be connected. It is needed to retain LVDS polarity and reverse the send/receive lines. To avoid getting confused, this problem is already solved by the pinout of the ACEX design, and the LVDS transponder hardware. The LVDS connector of the ACEX board is shown in Fig. 27. Two boards can be connected by just reversing the wires on both connectors (0-12 $\leftarrow$ 12-0). Connection of three or more boards can easily be arranged by using two 4pin plugs with crossed cables in between each client.

B.2 Software

lowlevel driver After booting gnu/linux (version $\geq$ 2.4.0) and *untar* of the source, the kernel module can be build by typing `make`. When run as *root*, the script `./scsb_load` loads the module and creates the /dev/scsb device files. There should be initialization or error messages to syslog (`/var/log/messages`). When the load was successful, status information is readable in

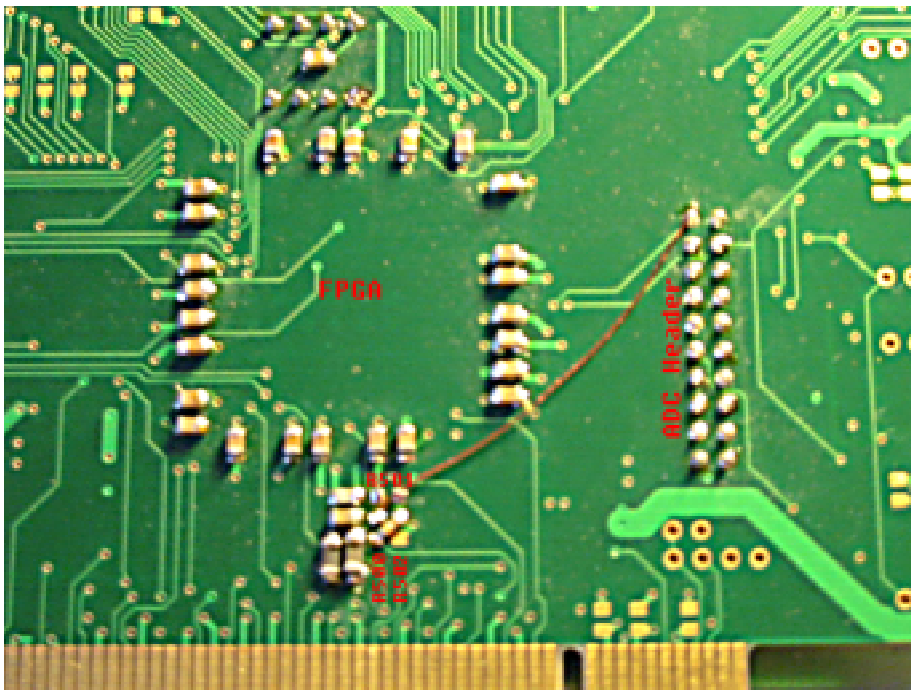


Figure 26: **ACEX board modification.** For use as a PCI card, the clock signals to the FPGA have to be rerouted. Although this bottom view of the ACEX board is not very good, the added wire for the PLL clock input and the place where to resistors have to be rearranged becomes more obvious.

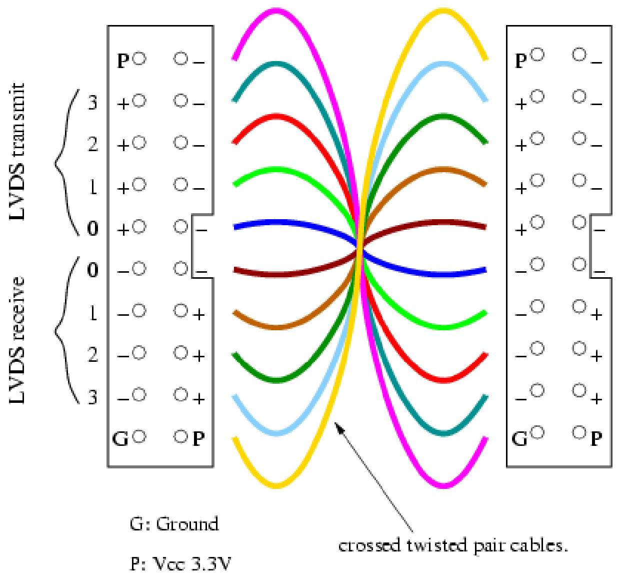


Figure 27: **Pinout of the ACEX LVDS header.** This Figure shall give a hint, how to construct scsn cables for use with the ACEX board.

`/proc/scsb`. Test the network by running `./benchmark 1`, which should send a single frame and report timings.

scsb configuration shell The configuration shell provides a bus abstraction layer of the scsnetwork which is documented in chapter 4.2.3. Compile and build information is included in the source code. The shell right now includes lot of TRAP-1 specific commands, and uses only one of the two rings. Although the API is almost capable of this functionality, due to lack to time, they have not been debugged and tested completely. Quick start: `interface 0` $\boxed{\texttt{enter}}$ `ping` $\boxed{\texttt{enter}}$ `. send` $\boxed{\texttt{enter}}$ will display a short help for the lowlevel command to transmit a frame. When the clients use the TRAP-1 scsn application layer, the TRAP-1 `read` and `write` commands can be used.

B.3 Setup Slave

The easiest way to set up a slave is to use an EDIF file of scsn (note that the network speed ratio is no longer generic in the EDIF files) and insert it as a block in the top layer. It provides the described TRAP-1 global I/O bus. One of the easiest test designs can be found in the ACEX folders on the Software CD-ROM: The *cfg_ack* is stuck at V_{CC}, the `bus_req`, `bus_we` and `bus_dout(0)` are connected to a D-Flip-Flop. The output of the DFF drives an LED and is sent back to `bus_din(0)`. All other bus data lines are driven to Ground.

C Internal interfaces

Appendix C contains plain signal lists and short descriptions of the major internal parts. Refer to figure 5 for an overview.

C.1 Data Link to Network Layer

Per Data Path there are 4 Signal lines. Since there are 2 path, replace X with either of 0 or 1:

Data-Path	Signal Name	width	DLL	NWL	Description
send	dX_send	69[54]	in	out	the data to be buffered/sent.
send	dX_we	1	in	out	write enable signal for data dX_send.
send	dX_send	1	in	out	initiate sending of the buffered data.
send	dX_buffer_ready	1	out	in	status of the outputbuffer: 0: busy - currently sending data (valid and send signals are ignored). 1: idle - data may be written. start of send causes this signal to become 0.
recv	dX_to_nwl	69	out	in	data that arrived and is currently buffered.
recv	sX_to_nwl	1	out	in	strobe line. is ’1’ as long as dX_to_nwl holds valid data. (until next first bit of next next frame arrives
recv	bufX_half	1	out	in	the input buffer is almost full.
recv	bufX_err	1	out	in	the checksum of the arrived frame is invalid.
-	bridge	1	in	out	cross the outgoing data lines. break topology, into two Rings.

Table 16: **Signals between Data Link and Network Layer.**

C.2 Network to Application Layer

The Network Layer does not buffer data: So if the data in the input buffer is lost (due to receiving a new frame), there is no chance to forward the data to the layers above. There is functionality to cause sending of an error-frame if this happens. The Network layer also includes a simple arbiter, to decide which input-path may currently ”use” the interface to the upper layer.

Note: When there is a request to switch the X-bar, there is some logic in the network layer, that ensures, that there is no data on the outgoing line, when switching the X-bar.

All requests have to be executed, and require a reply. Setting the *reply_valid* signal, enqueues the

[54] The bit-width of the buffers are generic values, and can easily be changed for the data link layer.

Signal Name	width	Data-Path	NWL	APL	Description
request	53	recv	out	in	the request to be executed (command,address,data)
request_valid	1	recv	out	in	strobe. is '1' as long as request data is valid.
reply	53	send	in	out	the (generated) reply to the request.
reply_valid	1	send	in	out	setting this to '1' signals the NWL, that the bus-transaction has been completed and the reply is to be sent.
altered_frame	1	send	in	out	the reply is an answer packet.
bridge_mode	1	-	in	out	mode to set the bridge to. 0: serial (non bridged) 1: crossed (bridged) mode
bridge_alter	1	-	in	out	bridge write enable. request change of bridge.

Table 17: Signals between Network and Application Layer.

reply data to the output buffer. If no operation is to be done in application layer, simply forward the *request* without raising the *altered_frame* signal.

C.3 Application Layer bus interface

The bus interface is described in chaptergeneric Interfaces and [14]. See table 4 for the bus signals.

D Pictures

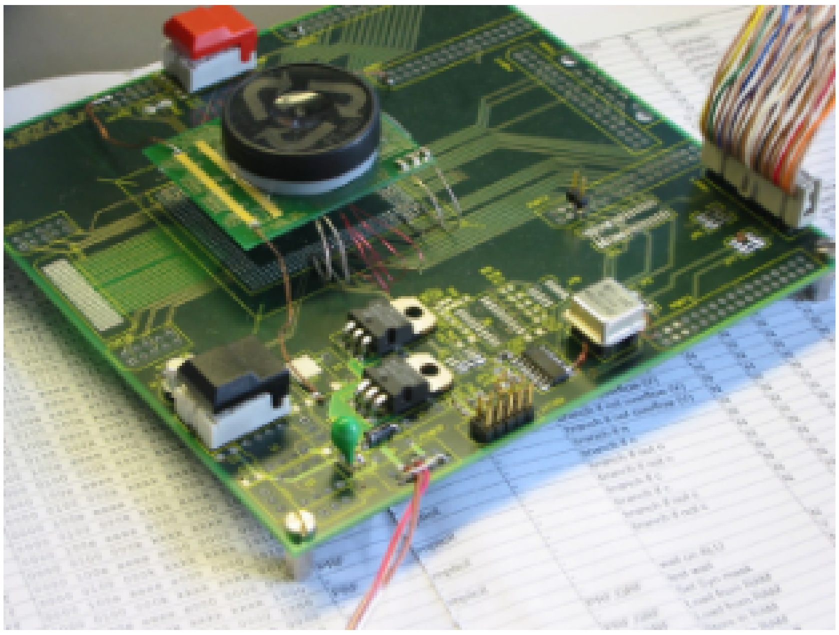


Figure 28: **First Testboard of the Trap 1.** All prototype tests were performed with this board.

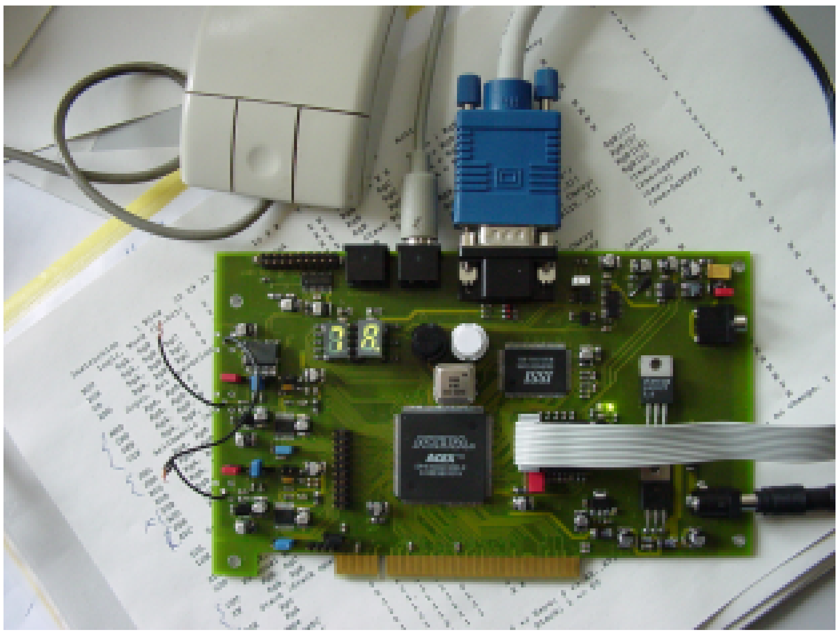


Figure 29: **ACEX board.** These multi-functional FPGA boards, developed by V. Angelov [1], have been used for testing and debugging the design process of scsn.

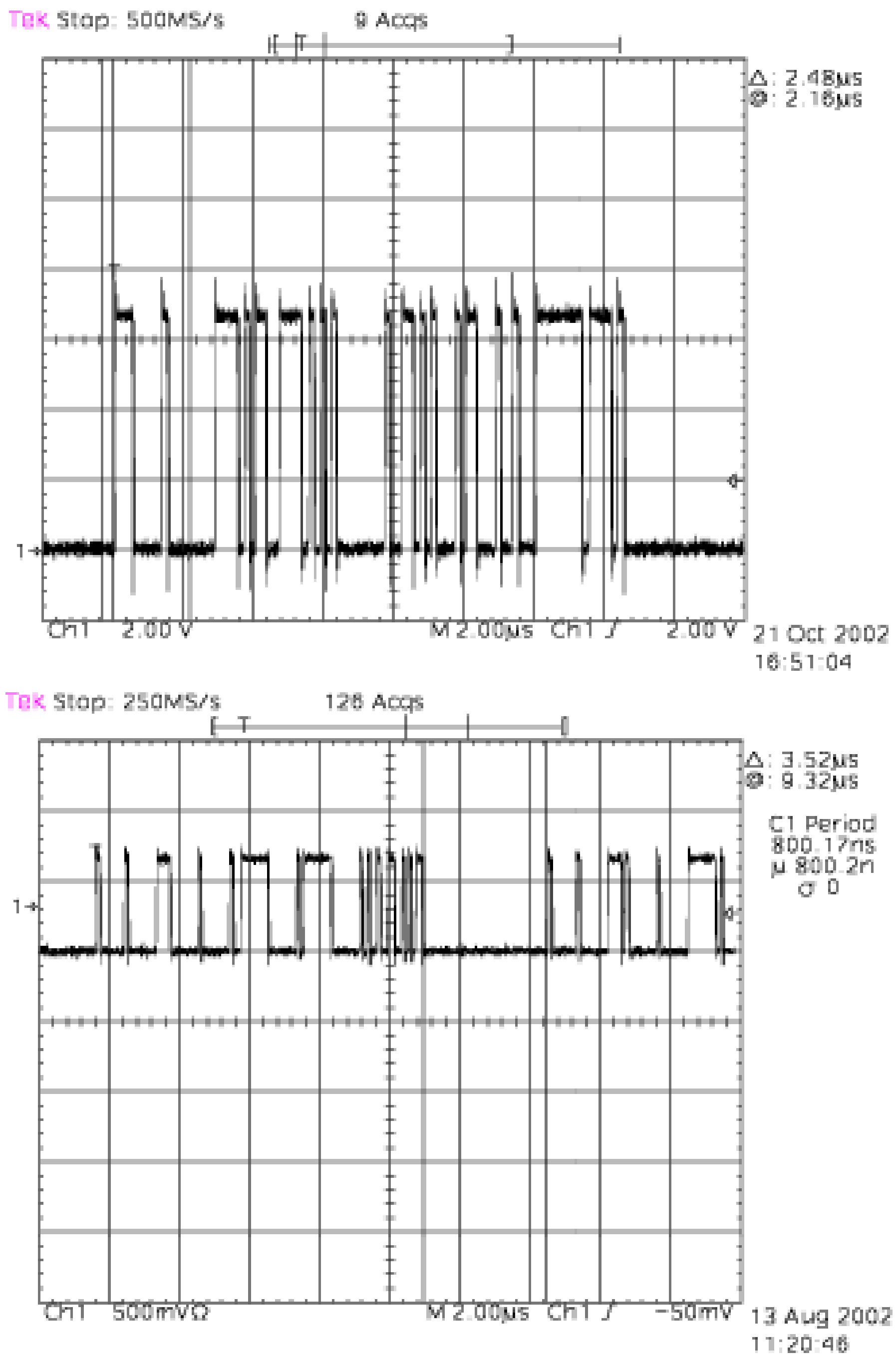
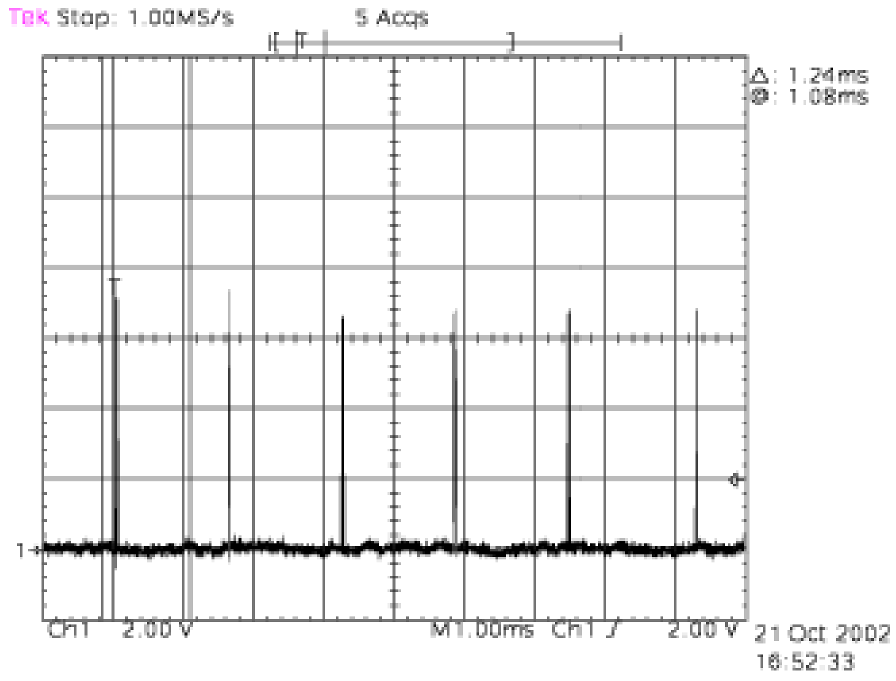


Figure 30: **Scope: transmitting ping Frames on the wire.** The top frame is hard to read, but the Figure b) shows two nice frames: After the startbit (1), the hopcounter follows "0000001", then the address(3)+src/dst(1) bit. "..001111". Next follow is the command (7) "1110...", address `0xabcd` , data "`0x12345600`" (LSB, w/ stuffing), and the checksum.



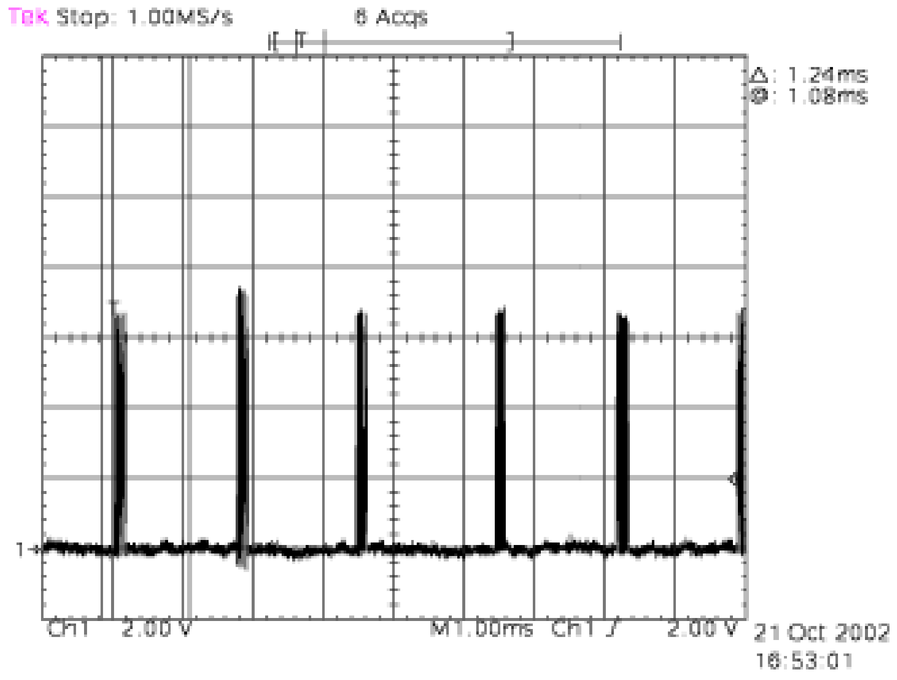
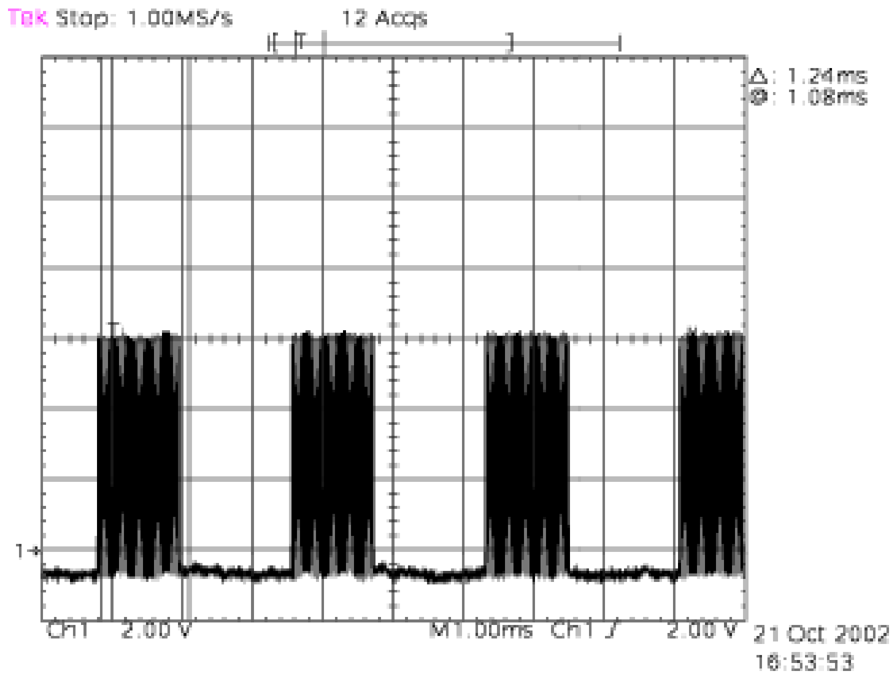


Figure 31: **Benchmark: 1 and 8 frames.** Software benchmark. a) transmission of 1 frame, wait until it arrives back, then send another one. b) same with 8 frames.



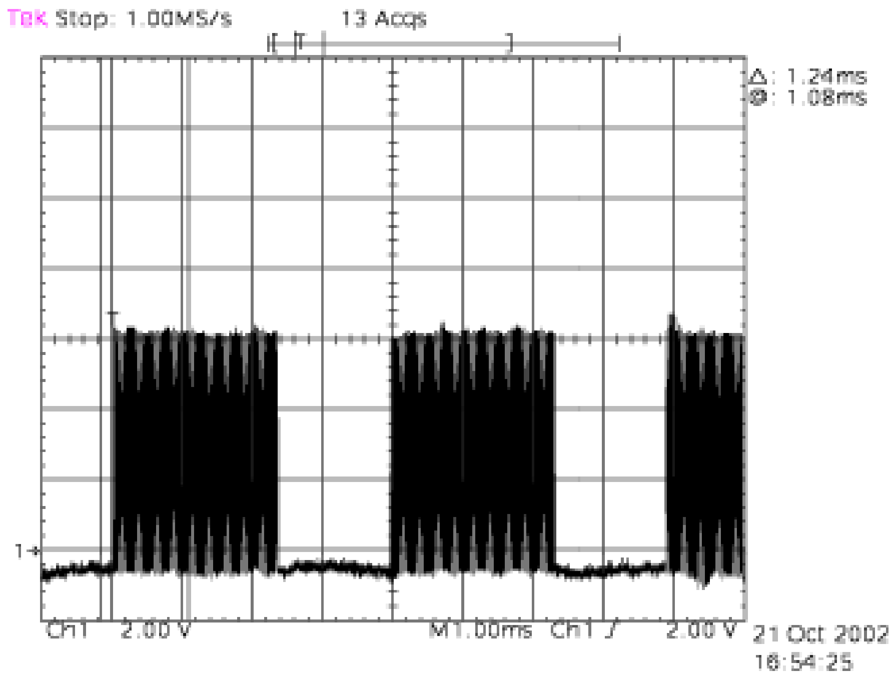


Figure 32: **Benchmark: 64 and 128 frames.** Software benchmark. a) transmission of 64 frame, wait until they arrive back, then send another 64 frames. b) same with 128 frames.

E List of Tables and Figures

List of Tables

List of Figures

References

[1] Dr. Venelin Angelov. Acex board. 2001. Online Documentation `http://www.kip.uni-heidelberg.de/ti/ACEXBoard/`.

[2] Dr. Venelin Angelov. Vhdl sweet 16 cpu - internal rewrite of the web sweet16. 2001.

[3] Dr. Venelin Angelov. Trap1 documentation and private communication. 2002.

[4] H. Peter Anvin. Linux allocated devices. June 2001. Linux Kernel Documentation - /usr/src/linux/Documentation.

[5] CERN. *ALICE TRD Technical Design Report*. CERN/LHCC, 2001.

[6] Alice Collaboration. on line documentation. Online Documentation `http://www.alicetrd.uni-hd.de/`.

[7] Alice Collaboration. Technical proposal. *CERN/LHCC*.

[8] Alice Collaboration. Technical proposal, addendum 1. *CERN/LHCC*.

[9] John L. Hennessy David A. Patterson. *Computer Organisation and Design - The Harware / Software Interfacesecond edition*. Morgan Kaufmann Publishers, Inc., 1998.

[10] Free Software Foundation. The gnu general public license. Online Documentation `http://www.gnu.org/licenses/gpl.txt`.

[11] Alice TRD group. on line documentation. Online Documentation `http://alice.web.cern.ch/Alice/`.

[12] Brian Von Herzen Ph.D. Jon Brunetti. The lvds i/o standard. *Xilinx Application Note: E-Virtex Family*, XAPP230, November 1999.

[13] Tobias Krawutschke. *Dissertation, unreleased thesis*. PhD thesis, 2002.

[14] Dr. Falk Lesser. *Entwurf und Realisierung eines vierfach MIMD Prozessor fuer eine Anwendung in der Hochenergiephysik*. PhD thesis, University of Heidelberg, 2002.

[15] A B D Reid M Mulvey, I Y Kim. Timing issues of constant bit rate services over atm. *BT Technology*, Vol 13 No 3, July 1995.

[16] Joerg Stinner. Soc-entwicklung eines kommunikationsmoduls zwischen leitrechner und intelligenten sensoren im rahmen des alice-projekts. 2002.

[17] Andrew S. Tannenbaum. *Computer Networks*. Prentice Hall PTR, 1996.

[18] Ross N. Williams. *a painless guide to crc error detection algorithms*. version 3. Rocksoft Pty Ltd, 1993.

Index

Erklarung:

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg October 30, 2002

Robin Gareus