CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of measurement

Bachelor's Thesis

Automotive Ethernet Analyzer

Jan Nejtek

2019
Supervisor: doc. Ing. Jiří Novák, Ph.D.

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Nejtek Jan** Personal ID number: **466128**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Open Informatics**

Branch of study: **Internet things**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Automotive Ethernet Analyzer

Bachelor's thesis title in Czech:

Analyzátor pro Automotive Ethernet

Guidelines:

Design and implement a prototype for 100Base-T1 (Automotive Ethernet) communication analysis. Follow the steps listed below:
1. Acquaint yourself with 100Base-T1 technology, especially with the physical layer functionality.
2. Design a concept of device for passive communication monitoring of a single network segment. Use a Gbit Ethernet interface as an output.
3. Implement necessary hardware; preferred variant should be based on the FPGA kit DE0-Nano-SoC.
4. Implement software in VHDL and test the complete functionality.

Bibliography / sources:

[1] Bučkovskij, D.: Využití sítí Ethernet v osobních automobilech. Bakalářská práce ČVUT FEL, Praha 2016
[2] Correa, C.: Automotive Ethernet - The Definitive Guide. Interpids Control Systems 2014, ISBN: 978-0990538806

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Jiří Novák, Ph.D., K 13138 - katedra měření

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **12.02.2019** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until:
by the end of summer semester 2019/2020

doc. Ing. Jiří Novák, Ph.D. Supervisor's signature	Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

Acknowledgement / Declaration

I would like to express my sincere gratitude to my supervisor doc. Ing. Jiří Novák, Ph.D. for all his valuable advice, his mentorship and his guidance. Also, I want to thank to my coworker Ing. Lukáš Krejčí for the implementation of support software for the development board that was not assigned as part of this thesis and for his help with troubleshooting miscellaneous issues with the project and general discussion. Last but not least, I have to thank to my dog Benjamin for providing unconditional support during my work on this thesis.

I hereby declare that I am the sole author of this thesis and that all resources that were used in the making of this thesis are identified in the reference section.

Prague, 18 April 2019

.....

Abstrakt / Abstract

Cílem této práce je navrhnout nástroj pro analýzu rozhraní 100Base-T1 (též známé jako Automotive Ethernet či BroadR-Reach). Práce je rozdělena do čtyř částí. V první části se seznámíme se standardem 100Base-T1 a čím se podobá a odlišuje od nejpoužívanějších standardů Ethernet. Následně na základě těchto znalostí navrhneme koncept jak pasivně monitorovat komunikaci na jednom síťovém segmentu tohoto rozhraní. Ve třetí části je popsána implementace potřebného hardware podle tohoto konceptu. Hardware je navržen tak, aby komunikoval s vývojovou sadou Terasic DE0-Nano-SoC. Poslední část popisuje návrh zdrojového kódu v jazyce VHDL, který dovoluje tomuto hardware fungovat a přeposílat zaznamenané pakety přes rozhraní Gigabit Ethernet, které je zabudované ve vývojové desce.

Klíčová slova: 100Base-T1, FPGA SoC, MII, PHY, analyzátor paketů, odposlech paketů

Překlad titulu: Analyzátor pro Automotive Ethernet

The goal of this thesis is to design an analyzer tool for 100Base-T1 (also known as Automotive Ethernet or BroadR-Reach). It is split into four parts. In the first part, we acquaint ourselves with the 100Base-T1 standard and its intricacies, similarities and dissimilarities to most often used Ethernet standards. Next, based on this knowledge, we design a concept on how to passively monitor communication on a single network segment. In the third part the necessary hardware is implemented in accordance with said concept. The hardware interfaces to an FPGA development board Terasic DE0-Nano-SoC as per the assignment. The last part outlines the implementation of VHDL code that enables the hardware that was implemented to work and forward recorded packets over the development board's built-in Gigabit Ethernet interface.

Keywords: 100Base-T1, FPGA SoC, MII, PHY, packet analyzer, packet sniffer

/ Contents

/ Figures

Chapter 1
Introduction

Recent developments in the automotive industry are slowly but surely rendering existing communication standards (such as CAN, LIN, FlexRay or MOST bus) either too slow or too expensive for certain purposes. Autonomous driving, advanced safety assistants and distributed infotainment are among the most bandwidth intensive features emerging in new models of cars across all manufacturers. Furthermore, as more and more cars feature a permanent connection to the Internet (via LTE or otherwise), security and integrity are also becoming very pressing issues. [1]

Many of these questions, such as bandwidth, network separation (VLAN), quality of service and time synchronization were already solved quite some time ago in the field of computer networks, so naturally car manufacturers have experimented with Ethernet networks in their products. This has however introduced some new issues, such as increased cable weight and cost, worse availability of automotive grade components for said Ethernet standards and enormous issues with meeting some EMI standards. [2] Some vehicles use 100Base-TX in their diagnostic port. For this reason, a new standard was developed jointly by the companies Broadcom, NXP and BMW which enables full duplex communication with 100 megabits per second bandwidth over a single unshielded twisted pair of cables. This standard was first called BroadR-Reach and was subsequently standardized with minor changes as 802.3bw a.k.a. 100Base-T1. [3] These two standards are compatible with each other.

More details on physical layer functionality of 100Base-T1 will be covered in the next chapter.

1.1 Measuring Automotive Ethernet

Since Automotive Ethernet uses a single differential signal pair for bidirectional communication, which means that it can't be simply measured, recorded and decoded by taking samples from the wire directly like it is possible with (for example) 10Base-T and 100Base-TX. [4] To get around this, the link must be split in two and bridged. Measurements then can be taken at the bridge point.

There is a commercial solution for measuring Automotive Ethernet available from Rohde & Schwarz for their RTP and RTO range of oscilloscopes. This also involves splitting the link and bridging it via their RT-ZF5 probing board. This probing board is equipped with directional coupling circuitry which allows to perform completely passive probing of transmitted data much like with the older Ethernet standards. Decoding then happens inside the oscilloscope. [5] This has two disadvantages, the first being that it a high end oscilloscope along with extra hardware and software is required to carry out measurements, and the second one is the lack of extensibility. For example, since the system is completely passive, it can't modify or inject extra data into the link if desired.

Chapter 2
The 100Base-T1 Standard: The practical parts

A very extensive description of BroadR-Reach (in Czech) by Dmitrij Bučkovský can be found at [1]. The (newer) IEEE 100Base-T1 standard (in English) can be found at [6]. For comparison, the general Fast Ethernet standard can be found at [7]. 100Base-TX is covered under the 100Base-X[1] section inside said standard.

In this chapter we will mainly explain the physical layer of 100Base-T1 to give more insight to statements from last part of introduction, as well as describe the similarities to 100Base-TX especially regarding communication with PHY chips to set the stage for the following chapter.

2.1 Difference between 100Base-T1 and 100Base-TX

So, the big question everyone must be asking is: "How did the OPEN Alliance manage to transmit at the same speed as Fast Ethernet while only using roughly half the signal bandwidth?" The answer is disappointingly simple. They sacrificed the robustness of the encodings used in the transmission of data. As a result, Automotive Ethernet is only capable of transmitting over distances less or equal to 15 meters, while regular 100Base-TX can work up to 100 meters.

Interestingly, this sacrifice also simplifies the sublayers that exist inside the physical layer. Regular 100Base-TX PHY contains **three** distinct sublayers: the Physical Coding Sublayer (PCS), the Physical Media Attachment (PMA) and the Physical Medium Dependent (PMD). The PCS sublayer receives data that is 4 bits wide in parallel and clocked at 25 MHz from MII (alternatively 2 bits wide at 50 MHz from RMII, which is equivalent) and uses a 4B5B encoding which converts said data to serial bursts of five bits at 125 MHz. The PMA sublayer receives this data and converts it to non-return-to-zero inverted code. Finally, the PMD sublayer encodes the data again using MLT-3 encoding. [7] This results in signal bandwidth of around 65 to 80 MHz depending on circumstances.

Now, Automotive Ethernet does away with the PMD sublayer and only has **two** sublayers total — the PCS and the PMA. The Physical Coding Sublayer receives the same data from the same MII or RMII, but instead of converting parallel to serial, it converts parallel to parallel using a 4B3B encoding, resulting in 3 bits of data at 33.3 MHz. The Physical Media Attachment sublayer uses PAM3 encoding which serializes the data while maintaining 100 megabit per second transmission speed and results in an unchanged signal rate of only 33.3 MHz.

Detailed functional block diagrams for physical layers of 100Base-T1 and of 100Base-TX can be found below.

[1] 100Base-X is a blanket term for 100Base-TX and 100Base-FX standards.

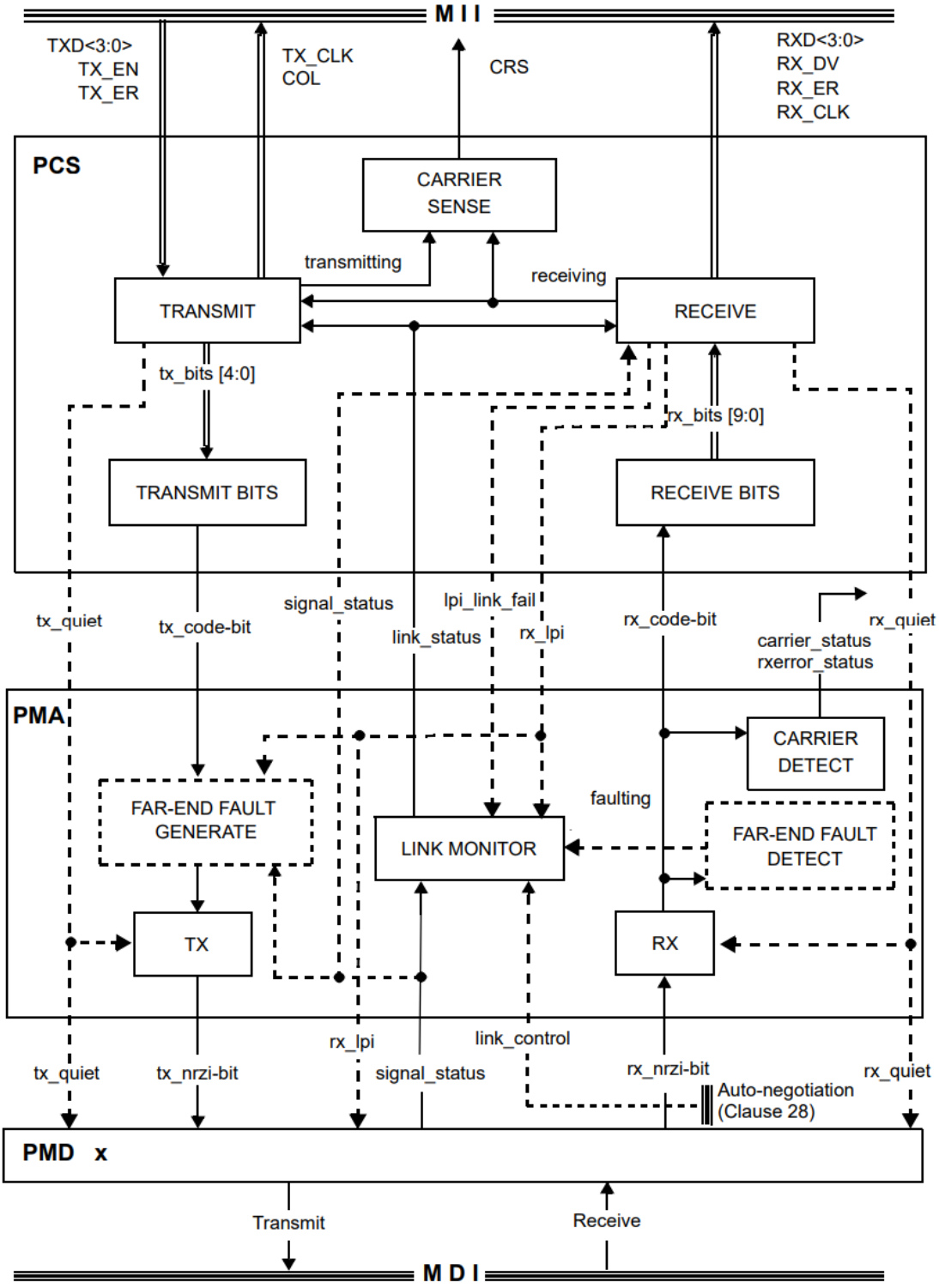

Figure 2.1. Functional block diagram of 100Base-TX PHY taken from [7]

NOTE: The same diagram is also applicable to 100Base-FX fiber optic standard.

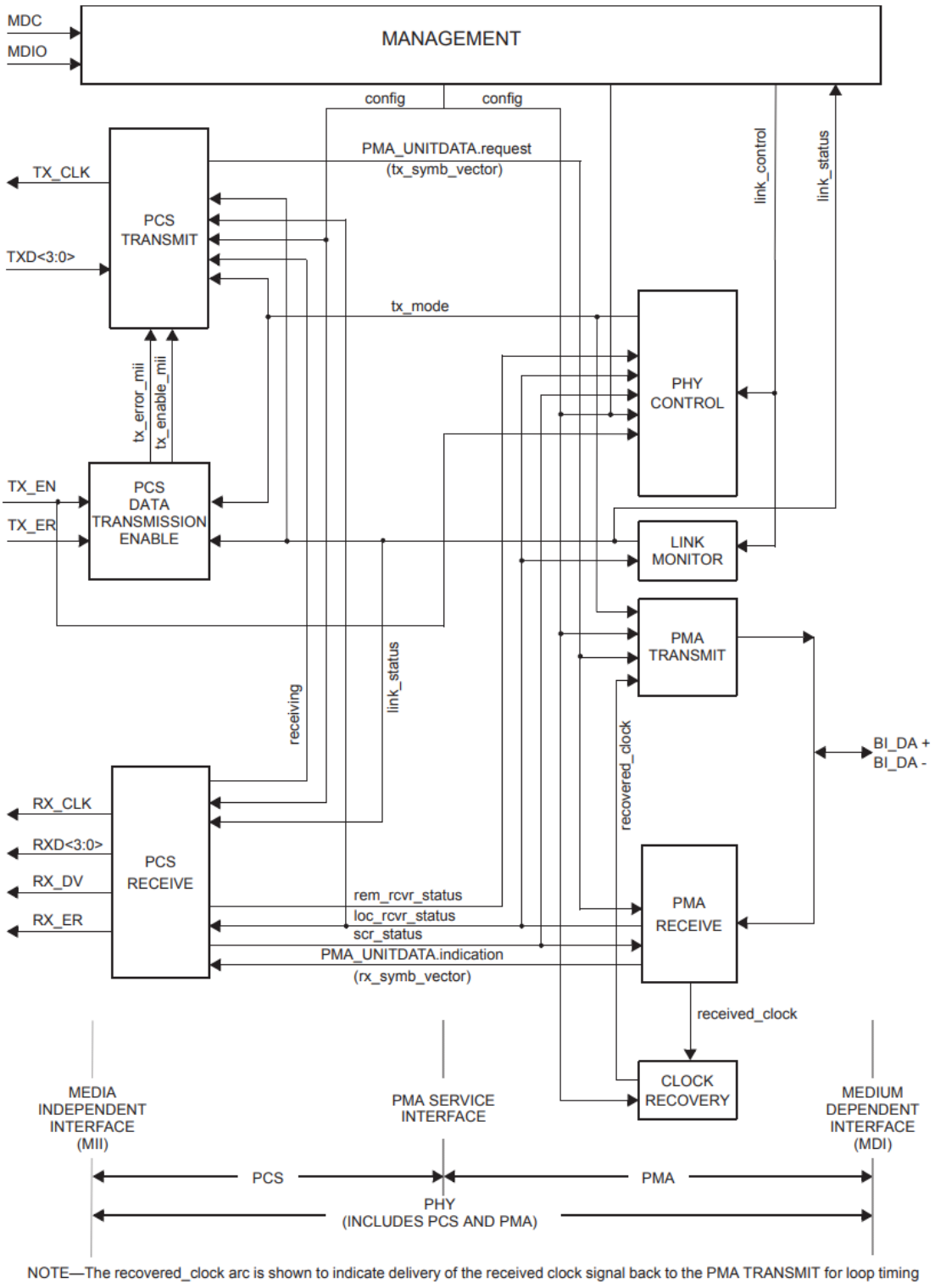

Figure 2.2. Functional block diagram of 100Base-T1 PHY taken from [6]

Now that the basic difference between 100Base-T1 and 100Base-TX has been covered, I will explain the intricacies of 100Base-T1 physical sublayers and the encodings they use in more detail.

2.2 Physical Coding Sublayer of 100Base-T1

The PCS Transmit functionality starts with receiving data from the MII interface. As was previously noted, this interface is exactly the same one that is used by other physical layer standards of Fast Ethernet and this will be taken advantage of in the following chapters.

The transmit part of this sublayer first converts the 4-bit parallel packet data nibbles coming from the MII interface clocked at 25 MHz to 3-bit parallel data clocked at 33.3 MHz. When the total number of data is not a multiple of three, one or two extra stuff bits are appended to the end of a packet.

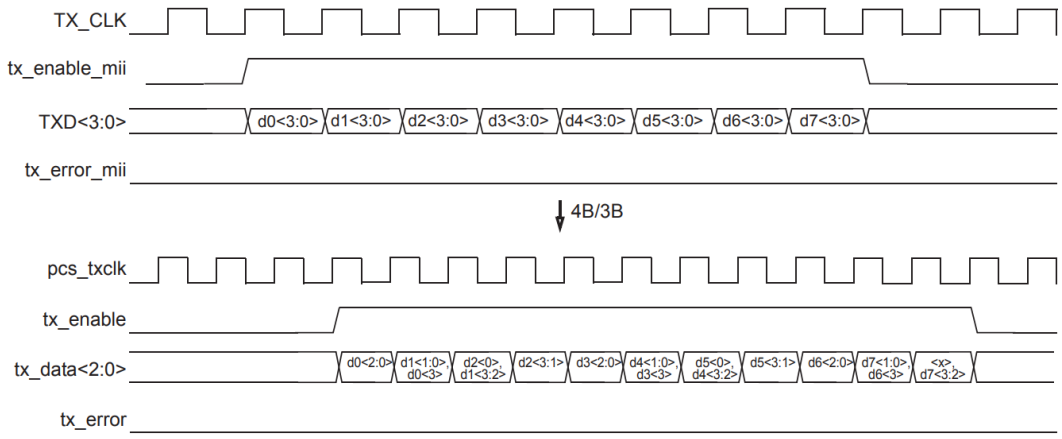

Figure 2.3. 4B/3B parallel data conversion in PCS taken from [6]

As noted above, the diagram sees 32 bits of data coming in through the MII interface, which is not a multiple of three, so a single stuff bit is appended at the end of transmission.

What is also worth noting, if the transmit error MII signal comes high at any time of the transmission for any period of time, the internal transmit error signal subsequently also comes high and stays high until the end of transmission (e.g. transmit enable MII signal comes low). This is because it would be very impractical to try to salvage the rest of transmission for subsequent encodings and somehow try to translate this signal to the coming domains.

The parallel 3-bit data is then subsequently scrambled via side-stream scrambling. [6] This is to help reduce the occurrence of spectral lines in the EMI spectrum. The scrambling is carried out using these two following polynomials:

$$g_M(x) = 1 + x^{13} + x^{33}$$

$$g_S(x) = 1 + x^{20} + x^{33}$$

where $g_M(x)$ stands for the polynomial used when the PHY is in master mode and $g_S(x)$ stands for the polynomial used when the PHY is in slave mode. More information on master and slave modes will be in the next section. Interestingly, these polynomials are **exactly the same** as the ones used in the PCS of 1000Base-T Gigabit Ethernet.

The correlation with 1000Base-T continues in the next part of the encoding, where the scrambled 3-bit data is mapped to two ternary symbols. This corresponds to a similar (but more complex) process in 1000Base-T where the scrambled data is mapped to four quinary [sic] symbols before finally being passed to the PMA sublayer.

How the two ternary symbols map to incoming 3-bit data can be seen in the table below. Please note that this mapping applies when the PCS as a whole is in a transmit state, and that it is different when the link is idle or being established. Another important thing to comment is the existence of special symbol sequences that mark the start and end of a stream, known as SSD – Start-of-Stream Delimiter and ESD – End-of-Stream Delimiter.

$Sd_n[2:0]$	TA_n	TB_n
000	–1	–1
001	–1	0
010	–1	1
011	0	–1
Used for SSD/ESD	0	0
100	0	1
101	1	–1
110	1	0
111	1	1

Figure 2.4. Data symbol to ternary symbol mapping in the PCS taken from [6]

The Start-of-Stream Delimiter is represented by a sequence (0, 0), (0, 0), (0, 0), the End-of-Stream delimiter is represented by (0, 0), (0, 0), (1, 1). Additionally, the End-of-Stream delimiter when an error has occured is represented by (0, 0), (0, 0), (-1, -1).

The PCS Receive functionality starts with receiving the ternary data coming from the PMA sublayer. The process starts with the PCS looking for a sequence of symbols signifying the start or end of a stream (SSD, ESD or ESD with error). Afterwards, it starts decoding these symbols per the above table and sets the internal data valid and error signals accordingly to what symbol sequence was identified.

The resulting sequence of parallel 3-bit data is then descrambled using the following polynomials:

$$g'_M(x) = 1 + x^{20} + x^{33}$$

$$g'_S(x) = 1 + x^{13} + x^{33}$$

where $g'_M(x)$ stands for the polynomial used when the PHY is in master mode and $g'_S(x)$ stands for the polynomial used when the PHY is in slave mode. Note that these polynomials are the same as the ones that were used in scrambling, but swapped.

Last, the descrambled 3-bit data is converted to 4-bit data using the reverse of the process that was illustrated in Figure 2.3. As was previously noted, if the amount of received data is not a multiple of four bits, the extra data will be discarded. [6]

2.3 Physical Media Attachment sublayer of 100Base-T1

The PMA Transmit functionality is quite simple. The sublayer receives ternary symbols from the Physical Coding Sublayer and transmits them over a single differential (two signals) line using the PAM3[1] modulation.

There are two interesting points to note here. The same modulation (PAM3) as 100Base-T1 is used in the (rarely used) 100Base-T4 Fast Ethernet standard. Similar, although more complex modulation (PAM5) is used in 1000Base-T Gigabit Ethernet.

The PMA Receive functionality is also simple. It is tasked with receiving the PAM3 modulated signal from MDI which is then decoded into said ternary symbols that are then sent to the Physical Coding Sublayer receive circuitry.

Another part of this sublayer worth noting is **The PMA Clock Recovery**. This circuitry is utilized when the PHY chip is running in slave mode (more information about master and slave modes is in the next section). It allows the chip to recover a clock signal that the other side of the link is running on, from the received symbols. When the PHY chip is in master mode, it uses its own clock source and clock recovery does not occur.

2.4 Master and Slave modes of 100Base-T1 physical layer

100Base-T1 employs a master-slave hierarchy similar to some other Ethernet standards such as 100Base-T2 or 1000Base-T. There has to be exactly one master and one slave device for the link to establish. [6] However, because of requirements in the automotive industry, the negotiation protocols known from these standards have been greatly simplified.

How? All devices are in slave mode by default, unless they get switched to master mode through their respective management interfaces. This means that it is assumed that all 100Base-T1 networks will be tightly designed and tested, and all links are wired as part of non-interchangeable connectors. After all, the internal network of a car is not expected to be changed or reconfigured at all in its lifetime, aside from repairs. This allows much faster initialization times than what we might expect from most commonly found Ethernet standards.

So, what is the functional difference between a master and a slave PHY chip?

On start up, the master PHY initiates the training sequence by transmitting idle pulses. This sequence is very very different from the known auto-negotiation that other Ethernet standards use – both sides of the link transmit data at the same frequency and phase and take measurements. These measurements are then used to tune the echo cancellation circuitry that enables separation of data sent and data received on a single fully duplex link. More on this in the next section.

Additionally, the master device uses its own oscillator to generate clocks for transmission over the link. The slave device uses clock recovery to match the clock of its master. The scrambling polynomials are also different between devices in master and slave modes.

100Base-T1 is also able to detect polarity of the link and adapt to it. The polarity of the master device is assumed to be the correct one, and the slave device adapts if the wires seem to be switched from its point of view. [8]

[1] Three level Pulse Amplitude Modulation

2.5 Full duplex communication on single twisted pair

100Base-T1 physical layer chips use hybrid function signal joining along with echo cancellation to effectively separate incoming ant outgoing waveforms. For the echo cancellation to work, it must be tuned at link start up, this is described in the previous section.

Unfourtunately, the 100Base-T1 standard does not provide a clear illustration or diagram of the signal separation happening between the transmit and receive parts of the PMA blocks and the actual MDI outlet, however a paper (cited below) from Texas Instruments does.

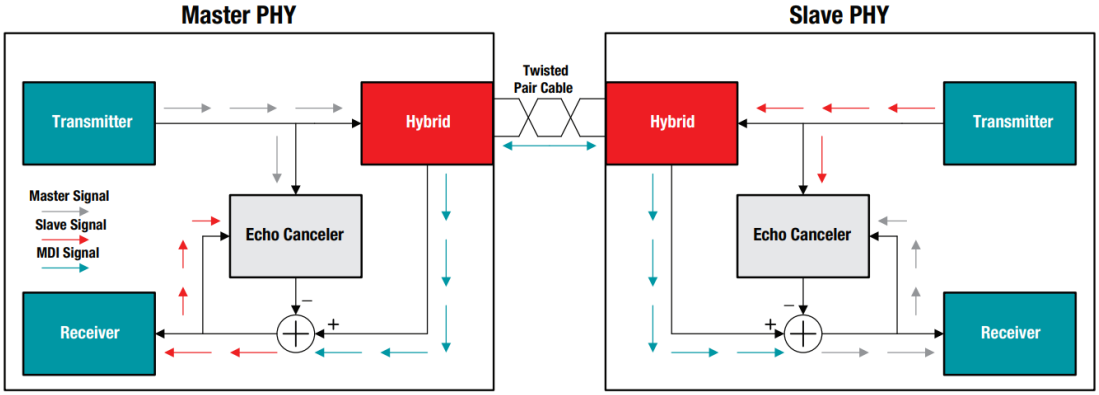

Figure 2.5. Block diagram of echo cancellation and hybrids in 100Base-T1 taken from [9]

2.6 Beyond the physical layer chip: notes on 100Base-T1 hardware design

When designing hardware that contains PHY chips for 100Base-T1, there is one big difference in circuit design when compared to PHY chips for 100Base-TX or even 1000Base-T. That is: Automotive Ethernet does away with Ethernet magnetic transformers and instead uses capacitive coupling and a low pass filter, along with some extra circuitry to comply with automotive EMI requirements and to protect from ESD. [8] An illustration is below.

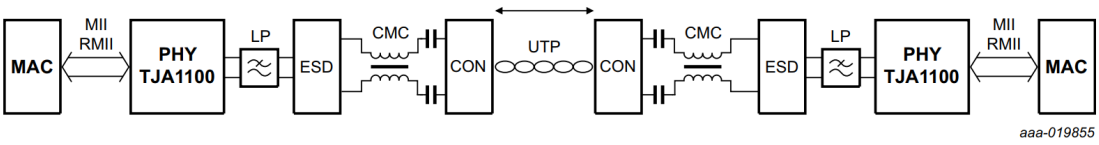

Figure 2.6. Block diagram of a physical 100Base-T1 link taken from [8]

This might seem like a setback in terms of PCB complexity, but one must consider that the risks that a magnetic transformer protects from might not be present in an automotive setting, moreover, its absence also has a part in reducing EMI and a tiny amount of weight might also be saved. All of this are strong points when considering automotive use.

Chapter 3
Designing a monitoring device for 100Base-T1

Please note that in this and the following chapter, I will be writing about some discoveries that me, my coworkers or my supervisor made before the official assignment of this thesis was written. If you start feeling at any point like you are experiencing a small time space anomaly, I apologize for the inconvenience.

In the second part of the introduction I wrote about an approach to measure and monitor a single 100Base-T1 network segment by Rohde & Schwarz. For obvious reasons, I decided against replicating this approach. First, it already exists and is being marketed and sold by a reputable company, there would be no point replicating it as a bachelor's thesis. Second, it requires a high-end oscilloscope. While it might be possible to replicate it with more readily available equipment (either a lower end oscilloscope or logic analyzer) or even redesign it with entirely custom hardware, it would still be at the very least inheriting the disadvantages present in said commercial solution.

Thanks to my supervisor, I learned that there's a considerable demand in the industry for an *active* analyzer, which would be capable of not only monitoring, but also modifying the packets coming through for purposes of development and testing. This is not a part of this thesis' assignment per se, but we decided to design the device in such a way that would allow to implement this functionality later via a software update, for example.

However, since I was assigned to design a *passive* monitoring device, I would some day be tasked in changing the way it is internally wired in order to transform it into an active device. If one needs to make changes in hardware with a software update, the first thing that comes to mind is an FPGA device.

I have to mention at this point that approaching this issue using Ethernet switch integrated circuits with port mirroring and concentration functionality was also considered. Unfortunately, this carries the same disadvantage as the setup used by Rohde & Schwarz, the impossibility to do signal modification in the future.

3.1 Capturing packets passing through

The basic idea was to take two 100Base-T1 PHY chips that would be bridged to each other via an FPGA, and then add another interface with higher speed which would output copies of packets that came from both sides, as is pictured below.

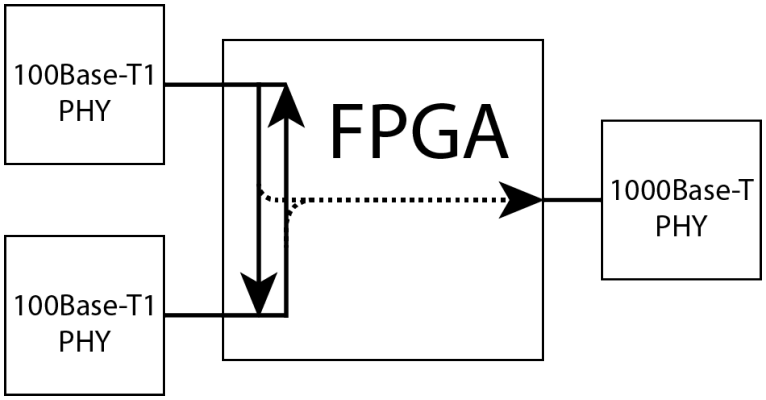

Figure 3.1. Basic concept of FPGA based monitoring device with three PHY chips

Now, in order for the device to appear passive to the link being monitored, there have to be no MAC blocks along the way of data between the two 100Base-T1 PHY chips, the FPGA needs to simply connect their respective MII interfaces back to back while monitoring the data coming in both directions.

Some PHY chips for 100Base-T1 (and likely for other Ethernet standards as well) support a mode called "Reverse MII". This mode is inteneded for media converters and signal repeaters and it allow you to connect two PHY chips back to back without the need for any other circuitry and signal processing. One of such PHY chips is NXP's TJA1100. [8]

So, we take two of these chips, set one up in the Reverse MII mode, and use the GPIO ports of an FPGA development board to make connections according to the TJA1100 data sheet and application notes. This gives us a nice reliable 100Base-T1 repeater just as NXP intended, with the added benefit that we have all the access we need to the MII data passing through.

Now we are met with another issue, arguably the biggest problem to solve in this thesis. **How do we reliably record and send out copies of all the packets that pass through?**

3.2 Forwarding copies of captured packets to an external interface

We can draw some inspiration from the aforementioned approach using Ethernet switch chips. Nevertheless, reimplementing a such chip whole in VHDL is clearly both beyond the scope of a bachelor's thesis and unnecessary for our purposes. Since all the MII data is passing through the FPGA, we can take some internal cache memory and save copies of all the packets there. Subsequently, we can resend them from there, with the possibility of grouping smaller packets together to reduce the transport layer overhead.

As with most FPGA projects, the first course of action is to solve the repetitive problem using a soft microprocessor, such as Nios II in case of Altera/Intel FPGA based development boards. However, one can quickly find out that such system running at a frequency typical for an FPGA will have major issues saturating a Gigabit Ethernet link. In fact, in tests conducted by my colleague at the department, the soft microprocessor could not saturate the bandwidth required to forward two saturated 100Base-T1 links (it was well under 200Mbps) even with no other processing going on in the background.

The next course of action was to look at FPGA boards featuring hard processors. This did the trick, as an ARM processor running at a frequency several times higher than what is usual for an FPGA, while still not guaranteed to saturate a Gigabit link, has no issues fulfilling the requirements to transmit packets from two 100 megabit interfaces.

The last thing to determine is the type of the memory that we want to use. I will write in more detail on this topic in the next chapter, but to put things short, we have evaluated both using memory blocks that are internal to the FPGA (M10K or MLAB blocks in case of Altera/Intel FPGA Cyclone V) and using on-board DDR3 memory shared with the ARM processor via DMA. The latter turned out to be more suitable.

3.3 The final specification

Now is the time to take into consideration the actual implications and pitfalls of using actual FPGA hardware. Since the 100Base-T1 PHY chips run on their own 25MHz oscillators, we have to account for different clock domains. Fortunately, the 25MHz clock signal is provided to us as a part of the MII interface.

Since our internal cache memory has a wider data input port than four bits of the MII interface, we need to adapt the width first. This means we need a "parallel-in, parallel-out" shift register. I decided to create such functionality using a three bit nibble counter (because $8 = 2^3$ nibbles fit in a single 32-bit word) and a custom 4-bit to 32-bit multiplexing register.

The "overflow" of the counter needs to trigger data writes to the FPGA internal memory. These writes have to happen at exactly two occasions: when a full word is completed (e.g. we are at the last nibble and the MII data valid signal is still high) or when there is a falling edge on the MII data valid signal. This functionality was moved to the multiplexing register for ease of implementation.

Now, since we are still in the MII 25MHz clock domain, we need to adapt all required signals to the faster FPGA clock in order to prevent metastability from occurring. One possible solution to this problem would be to double buffer all signals passing over to the faster clock domain. However, this could cause unwanted slowdown in propagation of some of the signals and unnecessary complication in VHDL code.

For this reason it was decided to double buffer only the "data ready" signal from the aforementioned multiplexing register. We leverage that the FPGA clock is several times faster than the MII clock and this allows us to look for a rising edge of said signal on the 25MHz domain, and the moment we identify it send a trigger pulse signal on the faster domain that causes the rest of the entities to sample their respective signals at that moment.

Since all signals in the MII clock domain keep their value over a whole 25MHz period, we have plenty of time to sample data without having to worry about causing metastabilities, even when considering worst-case clock drift and/or skew.

We define a simple 32-bit register to capture the valid word when the "data ready" signal comes from the other clock domain. At that moment we have the word captured in this register and we can write it to memory. Since the memory is defined using the Intel Platform Designer tool and thus is accessed from the Avalon bus, an Avalon to External bus IP component was also used.

In order to capture the whole packet correctly, we need another counter to keep track of data address in said memory. Since we write whole 32-bit words, we use a 9-bit counter. Together with the upper two bits of the 3-bit counter mentioned before this gives us a complete 11-bit address signifying the end of data written and thus the length of data to copy over. Thus, the total memory capacity would be 2048 bytes, enough to store the largest possible Fast Ethernet frame.

The Avalon bridge IP has an "acknowledge" output signal which is used to increment the 9-bit counter. This means that this counter will be incremented with each 32-bit word passing into the memory. When the MII data valid signal falls low, the address can be relayed over to the ARM CPU via a memory mapped parallel input output register. I should also mention at this point that the memory in which the actual data is stored is defined as dual port, with one port being accessible from the FPGA and the other port also mapped into the memory of the ARM CPU.

Since we can not possibly expect the internal architecture of the FPGA SoC chip to allow us to copy out the entire contents of a frame (up to 1522 bytes) before another packet can arrive (0.96 μs) [7], we double the memory to 4096 bytes and use the uppermost bit as a bank switch. Then, we add a 1-bit "counter" to keep track of the active memory bank. This way, we can read the last packet from one bank of memory, while a new can be written to the other.

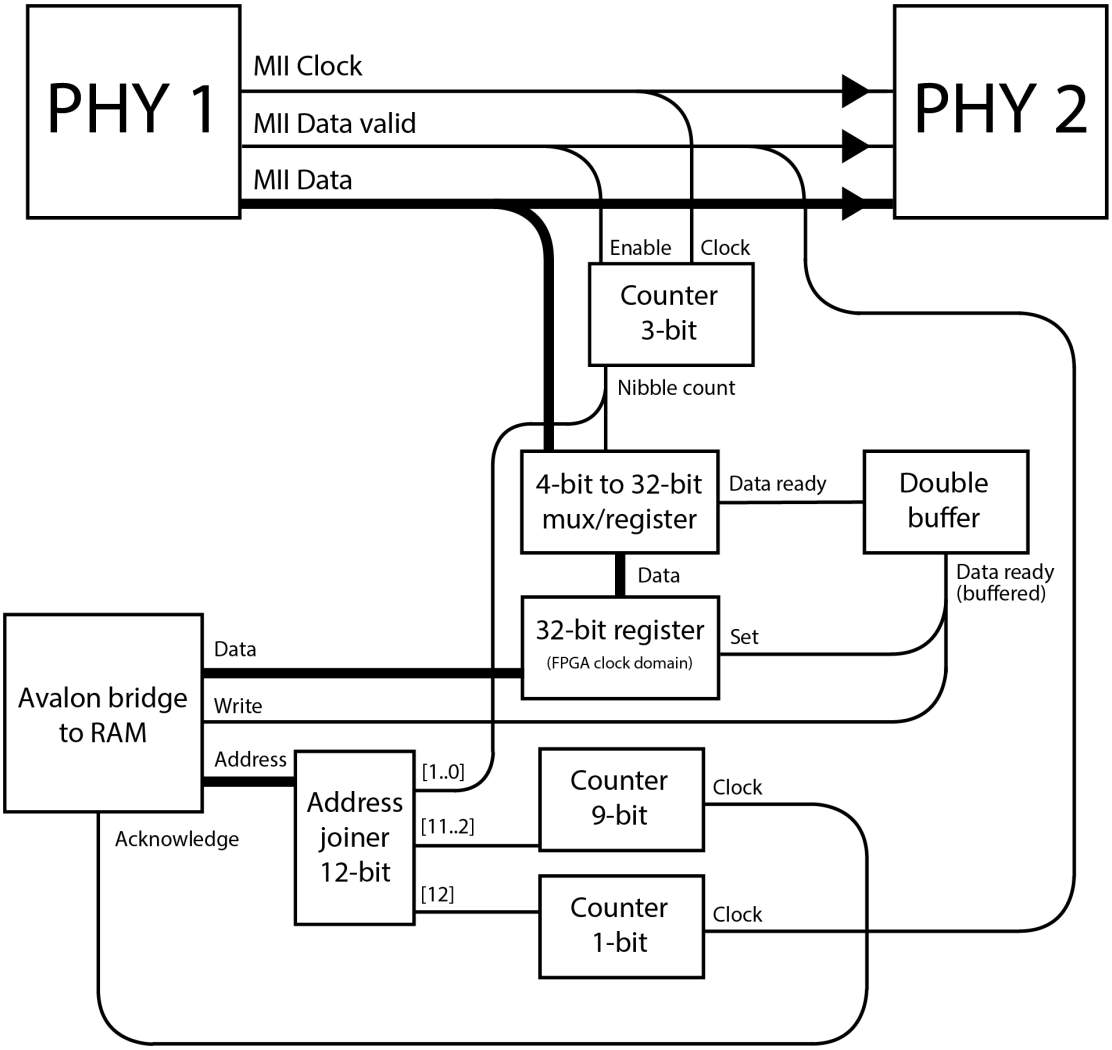

Figure 3.2. Detailed specification of entities defined inside the FPGA and their interconnects. Please note that this does not completely reflect the resulting Quartus project, because some functionality had to be moved between entities and some extra helper entities had to be added to cover all edge cases and to ensure reliable functionality. More on this subject in Chapter 5.

Chapter 4
Hardware design of a daughterboard for Terasic DE0-Nano-SoC

After consultation with my supervisor we decided to use the DE0-Nano-SoC development kit as the basis for this project. Its FPGA GPIO ports provided ample space for connecting two full MII interfaces, and the ARM processor part already has built-in Gigabit Ethernet connectivity and ample DDR3 memory capable of doing DMA transfers between the FPGA and the operating system. For further expandability, a dual CAN interface and two protected external inputs were also specified for the daughterboard.

I chose to design the schematic and printed circuit board in KiCAD. I have some prior experience with the EAGLE editor from prior personal hobby projects, but I wanted to learn to use an open source editor. Furthermore, a colleague from the department provided me with a tested footprint for the TJA1100 PHY chip and for the 100Base-T1 specific ESD diode and EMI choke.

The first step I had to take was to design a special footprint for the GPIO ports of said Terasic development kit. I took care to create the footprint (and manage the KiCAD project) in such a way that any footprints or schematic symbols I create, modify or extract from other libraries are easy to reuse in future projects.

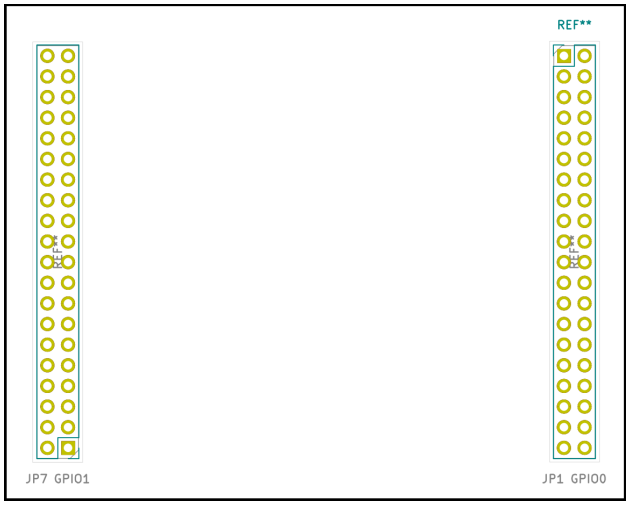

Figure 4.1. Basic KiCAD footprint for GPIO ports of Terasic DE0-Nano-SoC board (approx. 1:1 scale)

As I mentioned in the previous chapter, I drew the schematic based on information found in the TJA1100 data sheet [8] and application note [10], specifically the Reverse MII schematic from the data sheet and the supply schematic from the application note.

I made sure to connect all communication pins to the FPGA board GPIO ports, including interrupt output, enable and reset input and the SMI bus. Only pin left unconnected is the inhibit pin (used for external voltage regulator control), since it is explicitly specified to be left floating in our case.

The TJA1100 chip supports configuration of the most basic parameters (MII mode, master or slave mode, managed or independent operation) either via by pin strapping (aside from the SMI protocol). Pin strapping works by connecting pull-up or pull-down resistors on pins specified by the manufacturer. The chip's address on the SMI bus can also be configured by the same means, this allows more than one TJA1100 PHY on one control bus.

Since the configuration values for the pin strapping were very likely to change during development, I opted for using three-pad solder jumpers that would allow a given configuration pin to be pulled either high, or low, or neither way. I also decided to use resistor arrays for said pull-up and pull-down resistors to simplify PCB layout and assembly.

4.1 The first iteration

The first manufactured version of the hardware was designed basically in full accordance to the above paragraphs. I chose to use 0805 (imperial) size passive components and opted for KiCAD's hand soldering variants of physical footprints. Later, it would turn out that they were unnecessarily large for anyone with at least intermediate soldering skills.

In order to minimize the amount of vias (and complexity of PCB layout), I decided to connect the second PHY chip in reverse order to its respective GPIO port and subsequently rotate the part 180 degrees in physical layout. This means that the chips are upside down in respect to each other, with the MDI interface pins facing each other, allowing for almost mirrored board design. This doesn't cause any further issues, since the GPIO pins on the Terasic development board are not connected in any particular order to the balls on the FPGA chip anyway.

I also added two TJA1051 CAN drivers, also connected to the FPGA board GPIO ports. My supervisor requested to use DB9 connectors for CAN and a 3.50 millimeter pitch two pin connectors for the 100Base-T1. The DB9 connectors would prove problematic as the regular stabilized PCB mount versions of them would not fit two side by side at the narrow end of the daughterboard, should the daughterboard have the same dimensions as the main board. At the time I opted for floating DB9 connectors without stabilization or screw holes, but they turned out to be hard to obtain and impractical.

One annoying issue were the solder jumpers on this board, since they did not end up in the same order between the two PHY chips due to the aforementioned rotation layout trick. Furthermore, they did not end up automatically annotated in order, so the silkscreen documentation did not help determining to which configuration bit belongs which configuration jumper. Also, I did not add the solder jumpers necessary for configuring the SMI address, but it didn't cause any issues, since this protocol was not used at all during this part of development.

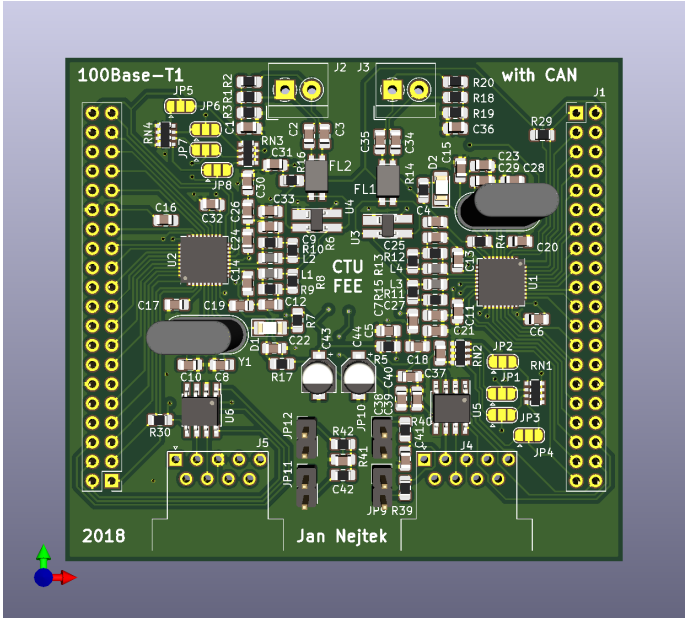

Figure 4.2. 3D Render of the first iteration PCB (approx. 1:1 scale)

This iteration of the board was successfully used to develop, troubleshoot and test all the functionality of this project. However, aside from the small issues I discussed above, we also realized that such finished product would be very difficult to place inside an enclosure, due to the daughterboard being significantly shorter (and interestingly, a little wider) than the main development kit PCB. Thus, I was asked to design a second, improved version of the printed circuit board. With the experience gained and knowledge of all outstanding issues, I excitedly accepted.

4.2 The second iteration

I started with redesigning the footprint of the dev kit's GPIO ports, except this time I captured everything about the base printed circuit board, including outline, profile of the USB ports, the power port and the Ethernet port and the screw holes.

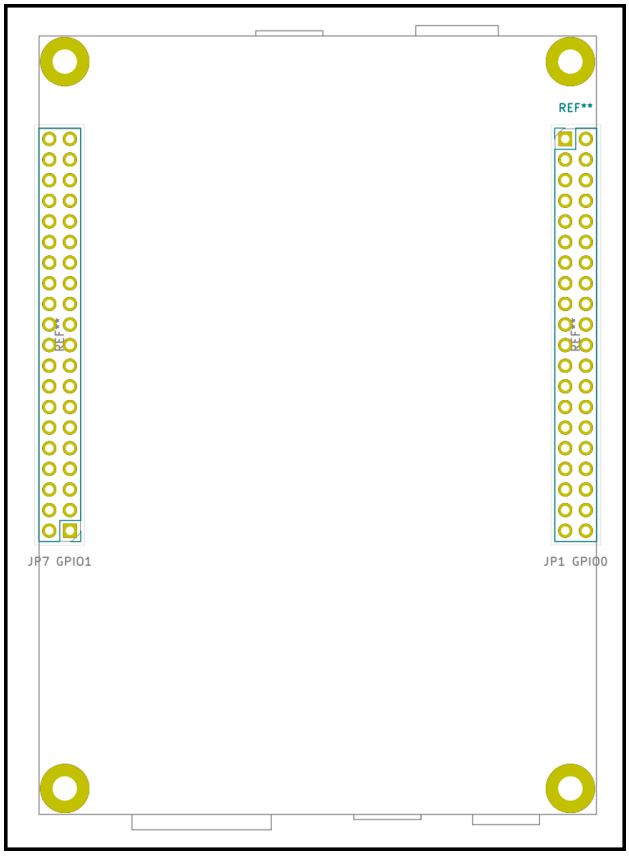

Figure 4.3. Full KiCAD footprint for GPIO ports of Terasic DE0-Nano-SoC board (approx. 1:1 scale)

With the daughterboard layout now being much taller, I also decided to swap the sides that the 100Base-T1 connectors and the CAN connectors were on. This provided me with a lot of new space to design the supporting circuitry for the MDI side of the 100Base-T1 chips with perfect symmetry (see Figure 2.6).

Then I cleaned up the schematic design, reviewing the TJA1100 application notes again and separating filter capacitors into neatly labeled groups to ensure they are indeed placed as close as possible to their respective power pins. I switched the footprints from their hand soldering variants to their smaller, regular counterparts.

In order to make it possible to fit the whole resulting hardware into an enclosure, I had to consider the placement of all ports to line up with flat enclosure walls. Since there were issues with fitting the DB9 connectors for CAN port connectivity, I decided to use board-edge mounted DB9 connectors and switched their footprints that one side would have a female connector and the other would have a male one. In order to fit said board-edge mounted connectors to flat enclosure, I designed them to sit inside a cut-out to move their flanges towards the inside of the board.

I also redesigned the configuration solder jumpers, so that they are in the same order for both PHY chips at the cost of slightly more complicated PCB traces. Then, I added more information to the silk screen and hid their respective annotation. Furthermore, I added solder jumpers for SMI address configuration. Since there are only two bits per chip, I could use a single resistor array for both pull-up and pull-down resistors, because the part I used did not utilize a common pin.

Then, I added the aforementioned circuitry for external inputs, with Zener diode protection from over-voltage, Schottky diode protection from negative voltage spikes, and a low pass filter for EMI protection. For improved thermal performance, I added nine vias (3x3 grid) under both the TJA1100 chips.

When we were testing and troubleshooting the first iteration of the board, it proved very annoying and time consuming to always have to look up where the power and ground pins are on the GPIO connectors. (For example for connecting oscilloscope probes.) For this reason, I labeled the respective pins using the front PCB silk screen.

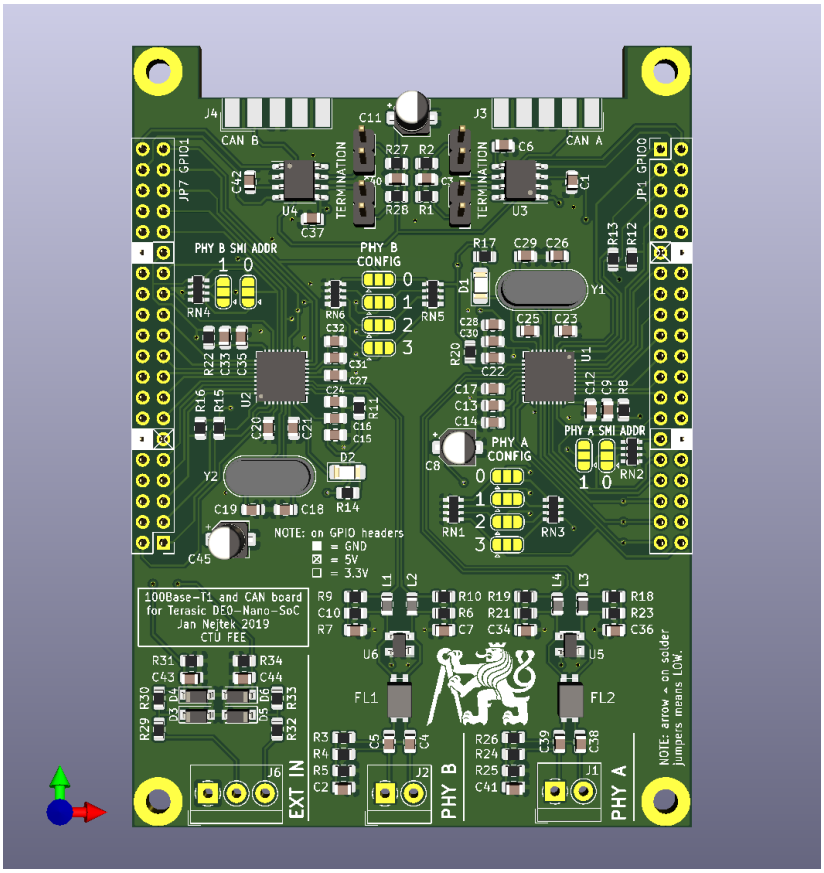

Figure 4.4. 3D Render of the second iteration PCB (approx. 1:1 scale)

Last but not least, I ensured that the passive components are aligned to make the PCB more aesthetically pleasing, and if possible, grouped them by their values to make assembly easier and faster. I also added the logo of the Czech Technical University.

The full KiCAD projects for both iterations of this hardware can be found on the included optical disc. Furthermore, schematics for both iterations are printed at the end of this thesis in full size.

Chapter 5
Implementation of functional VHDL code

Now comes the time to actually write and test the VHDL code for the logic that was outlined in Chapter 3. During the development I wrote a testbench along with the individual entities. This surprisingly helped speed up development, because ModelSim[1] is both faster and stricter in VHDL compilation than Quartus Prime.

The testbenches I wrote feature simulated MII data input. The simulated clock is 25MHz and shifted out of sync from the FPGA clock. The simulated data is a counting pattern to help verify that it is correctly processed and ends up in the correct order in 32-bit words to be written to memory.

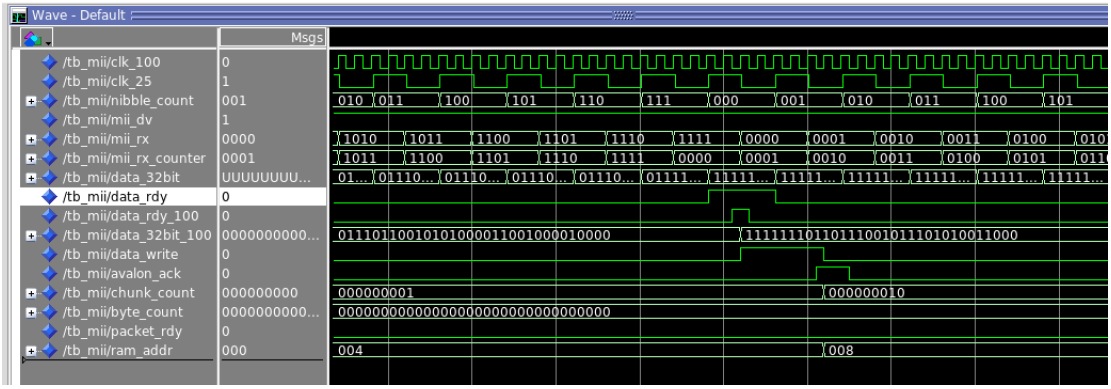

Figure 5.1. Simulation of a single word being written to FPGA memory. This is from a testbench which assumes memory with two banks. Notice the propagation of `data_rdy` signal. Also notice that the `chunk_count` signal is incremented and `data_write` falls low only when an acknowledge signal is received from the Avalon bus (`avalon_ack`). This signal comes surprisingly fast even on actual hardware.

Towards the end of Chapter 3 I spoke about using two memory banks to compensate slower transfer of recorded data to the ARM CPU. Unfortunately, it turned out that this setup is not fast enough. Due to the speed and latency of the memory accesses from the CPU, both very short and very long packets would not get copied over in time. Tuning memory bus widths and the controlling software yielded negligible improvements. At this point we experimented with implementing DMA transfers which did make a noticeable improvement.

Even with the utilization of DMA transfers, the time to copy over a very long packet (1522 bytes) was approximately 45 microseconds, which is enough time for seven very short packets to arrive. Including preamble and interpacket gap, the shortest possible Layer 1 packet is 84 bytes total. [7] Since Fast Ethernet has a bit time of 0.01 μs, this equals to 6.72 μs per packet, so seven packets should arrive in around 47 μs.

For this reason, the FPGA memory was expanded to eight banks of 2048 bytes for a total of 16384 bytes or 16 KiB. The bank counter was also expanded to 3-bit from 1-bit.

[1] ModelSim Intel FPGA Starter Edition

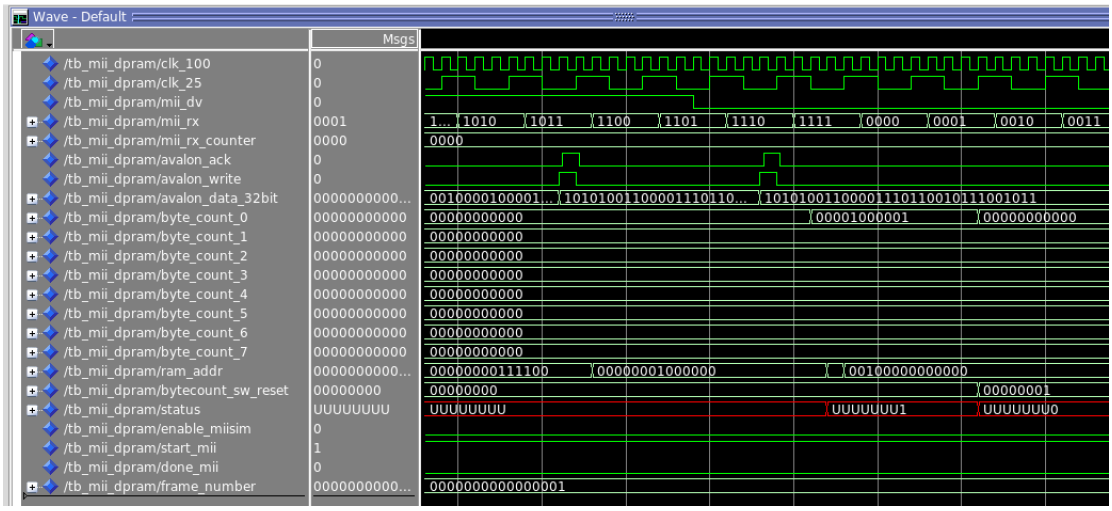

Figure 5.2. Simulation of an end of a packet with a length that is not a multiple of 32 bits. This testbench assumes memory with eight banks, thus the eight `byte_count_*` registers for communication with a CPU. Notice that the `byte_count_0` signal indeed has non-zero two lowest bits. Also notice the `ram_addr` signal increment the upper three bits to switch memory bank while zeroing out the rest. Also notice that the `byte_count_0` register gets zeroed out when the `bytecount_sw_reset` signal changes. This is an acknowledge signal coming from the CPU, and would normally get sent much later.

During early development we also uncovered a **massive pitfall** that currently occurs in Quartus during project creation. When a project is created for the DE0-Nano-SoC/Atlas-SoC development board, the downloaded example top-level file (`de0_nano_soc_baseline.v`) **incorrectly** assumes that the onboard 50MHz clock pin is called `FPGA_CLK1_50`. The correct name is `CLOCK_50`. This also applies to the other two clock pins `FPGA_CLK2_50` and `FPGA_CLK3_50`, which should be called `CLOCK2_50` and `CLOCK3_50`, respectively. This can be uncovered in the assignment editor after compiling the project, where `FPGA_CLK1_50` and possibly some other signals end up undriven.

This causes a very cryptic problem when instantiating the HPS bridge. The HPS bridge needs a clock input, and it causes very weird behavior if it ends up connected to an undriven signal. It seems that it depends on the individual compilations, some cause the ARM processor to lock up immediately after the FPGA is programmed, others can still work. Yes, you read correctly. Different compilations of a same project plagued by this mistake may or may not work. I suspect it depends on the work of the fitter.

The full Quartus Prime project can be found on the included optical disc.

Chapter 6
Conclusion

I was tasked to design a device that performs analysis of communications over the 100Base-T1 standard. This required solving a very wide range of problems, ranging from theoretical to practical, and from hardware to software. I started this thesis out by researching the functionality of said standard and how it differs and coincides with other Ethernet standards.

This allowed me to design a concept on how to carry out monitoring of a single network segment. I leveraged both the newly gained knowledge on 100Base-T1 and previous knowledge of hardware and software design from my university. Because there is currently no suitable hardware that the requirements of this work on the market, I designed an expansion board for an existing suitable FPGA development kit.

I took advantage of this opportunity to acquaintance myself with a new electronic design suite. After the printed circuit boards were manufactured, I personally assembled two specimens. Finally, I went on to implement the VHDL code under the guidance of my amazing supervisor. After tackling all the challenges and pitfalls and many adjustments, the device has indeed proven to be working and functional, capable of capturing packets sent from both other development tools and those sent from real car electronics.

One spot that can definitively be improved in the future is the VHDL code base. Although I tried to rigorously test it and make it well arranged and easy to understand, the issues encountered during development took their toll. But then again the same is true for most code bases in the world, it really is a never ending battle.

It would also be interesting to make the CAN part of the hardware work. Presently it is just connected to the GPIO ports of the development board. It was not assigned as part of this thesis, but making it work will be with no doubt also beneficial to future users.

Thanks to this project I gained a lot of theoretical as well as practical knowledge and I am looking forward to extending it in order to create a true active signal analyzer. The fact that there is a demand in the industry for such product is very exciting and personally I would love to see this tool being actively used.

References

[1] Dmitrij Bučkovský. *Využití sítí Ethernet v osobních automobilech*. České vysoké učení technické v Praze. 2016.
http://hdl.handle.net/10467/64842.

[2] Charles M. Kozierok, Colt Correa, Robert B. Boatright, and Jeffrey Quesnelle. *Automotive Ethernet: The Definitive Guide*. Intrepid Control Systems, Inc., 2014.
https://cdn.intrepidcs.net/brochures/icsusa/Sample-AutomotiveEthernet-TheDefinitiveGuide.pdf.

[3] David Maliniak. *What's the Difference Between BroadR-Reach and 100Base-T1?* 2018.
https://www.electronicdesign.com/automotive/what-s-difference-between-broadr-reach-and-100base-t1.

[4] Tektronix Inc. *Troubleshooting Ethernet Problems with Your Oscilloscope*. . Tektronix, Inc.. https://www.tek.com/document/application-note/troubleshooting-ethernet-problems-your-oscilloscope.

[5] Anna Flockett. *Oscilloscopes for debugging automotive Ethernet networks*. 2019.
https://automotive.electronicspecifier.com/connectivity/oscilloscopes-for-debugging-automotive-ethernet-networks.

[6] *8802-3:2017/Amd 1-2017 - ISO/IEC/IEEE International Standard - Part 3: Standard for Ethernet - Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1)*. 2018.
https://ieeexplore.ieee.org/document/8310988.

[7] *802.3-2018 - IEEE Standard for Ethernet*. 2018.
https://ieeexplore.ieee.org/document/8457469.

[8] *TJA1100 Product data sheet*. 2018.
https://www.nxp.com/docs/en/data-sheet/TJA1100.pdf.

[9] Donovan Porter. *100BASE-T1 Ethernet: the evolution of automotive networking*. 2018.
http://www.ti.com/lit/wp/szzy009/szzy009.pdf.

[10] *Application hints for TJA1100 Automotive Ethernet PHY*. 2017.
https://www.nxp.com/docs/en/application-note/AN12088.pdf.

Appendix A
Glossary

- ARM — Advanced RISC Machine
- CAN — Controller Area Network
- CPU — Central Processing Unit
- DMA — Direct Memory Access
- EMI — Electromagnetic Interference
- ESD — End-of-Stream Delimiter
- FPGA — Field Programmable Grid Array
- GPIO — General Purpose Input and Output
- MAC — Medium Access Control
- MDI — Media Dependent Interface
- MII — Media Independent Interface
- MLAB — Altera/Intel FPGA Memory Logic Array Block
- MLT-3 — Multi-Level Transmit encoding
- M10K — Altera/Intel FPGA Memory 10 Kilobyte Block
- PAM — Pulse Amplitude Modulation
- PCB — Printed Circuit Board
- PCS — Physical Coding Sublayer
- PHY — Physical Layer chip
- PMA — Physical Media Attachment
- PMD — Physical Medium Dependent
- RMII — Reduced Media Independent Interface
- SMI — Serial Management Interface
- SSD — Start-of-Stream Delimiter
- VHDL — VHSIC Hardware Description Language

Appendix B
Schematic diagram of first version hardware

Please see next page.

Appendix C
Schematic diagram of second version hardware

Please see next page.