Domain Engineering Sciences
Rte du Rawyl 47
CH-1950 Sion 2
Phone +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Degree Systems Engineering

Option Infotronics

Diploma 2011

Yann Santschi

Ethernet traffic measurement

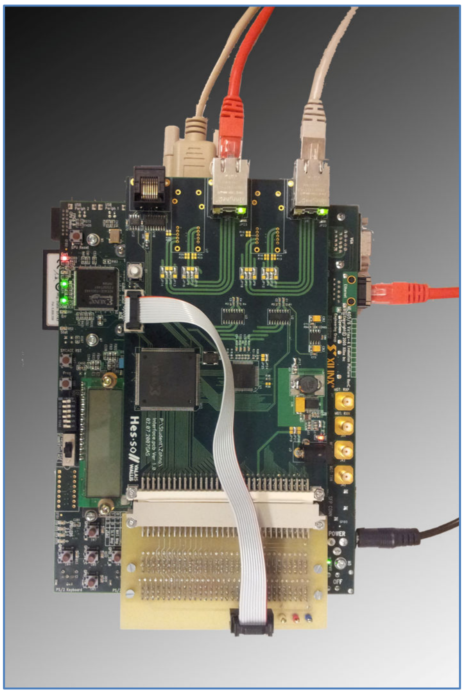
Professor	François Corthay
Expert	Patrik Arlos

Karlskrona, November 25th 201

Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

Ethernet traffic measurement

Graduate Yann Santschi



Bachelor's Thesis
| 2011 |

Degree course
System Engineering

Field of application
Orientation Infotronics

Supervising professor
Dr. Corthay François
francois.corthay@hevs.ch

Partner
Blekinge Institute of Technology
Dr. Arlos Patrik
patrik.arlos@bth.se

Objectives

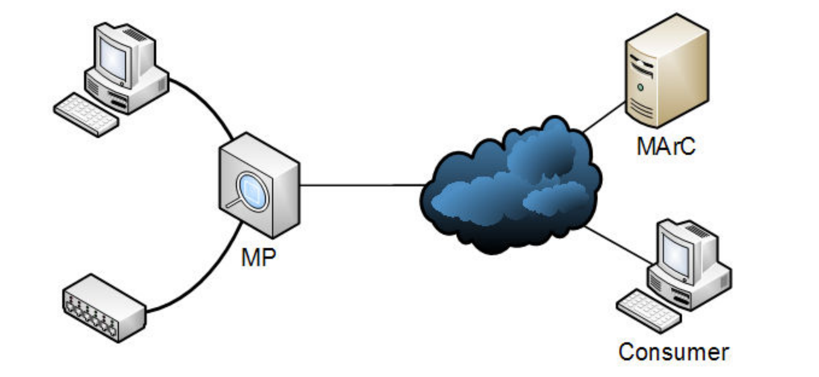
The goal of this project is to create a FPGA Ethernet traffic measurement point and interface it with an existing measurement network, named DPMI, that is developed in Blekinge Institute of Technology.

Methods | Experiences | Results

The measurement point captures Ethernet frames on a network link and filter them according to rules given by the Measurement Area Controller. If the captured frame matches filter rules it is written in a buffer. When the buffer is full, its content is sent to the consumer.
Each captured frame is tagged with a timestamp. For having an accurate timestamp, the measurement point receives time synchronisation signals from a GPS receiver.
A Measurement Area contains at least one controller, one measurement point and one consumer (see picture below).

During the development phase the whole VHDL design was simulated to verify its good functioning and a test Measurement Area was set up for testing the VHDL design in the FPGA.

Finally the measurement point is working almost fine.



Minimal Measurement Area with the MArC (Measurement Area Controller), the Consumer and the MP (Measurement Point) that is capturing data on the link between the computer and the switch.

HES-SO Valais
Route du Rawyl 47
1950 Sion

Phone 027 606 85 11
Website www.hevs.ch

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.AB
pof/31/01/2009

☒ FSI □ FTV	Année académique / *Studienjahr* **2010/2011**	No TD / *Nr. DA* **it/2011/55**
Mandant / *Auftraggeber* □ HES—SO Valais □ Industrie ☒ Etablissement partenaire *Partnerinstitution* **Blekinge Institute of Technology**	Etudiant / *Student* **Yann Santschi** Professeur / *Dozent* **François Corthay**	Lieu d'exécution / *Ausführungsort* □ HES—SO Valais □ Industrie ☒ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* □ oui / ja [1] ☒ non / nein	Expert / *Experte* (*données complètes*) **Patrik Arlos** Blekinge Institute of Technology	

Titre / *Titel*

Ethernet traffic measurement point

Description et Objectifs / *Beschreibung und Ziele*

The aim of this project is to continue a previous diploma work on an Ethernet Measurement Point (MP) for a Distributed Passive Measurements Infrastructure (DPMI). In that diploma work, a sniffer board was realized and a MP circuit was developed on an Actel FPGA board. The board was connected to a Xilinx ML405 development kit.

In this diploma work:

— The system has to interface to a measurement zone Ethernet line via the DPMI API

— The Ethernet packet filters have to be finalized.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
29.08.2011

Remise du rapport / *Abgabe des Schlussberichts:*
25.11.2011 | 12h00

Défense orale / *Mündliche Verteidigung:*
semaine 48 / *ab Woche 48*

Signature ou visa / *Unterschrift oder Visum*

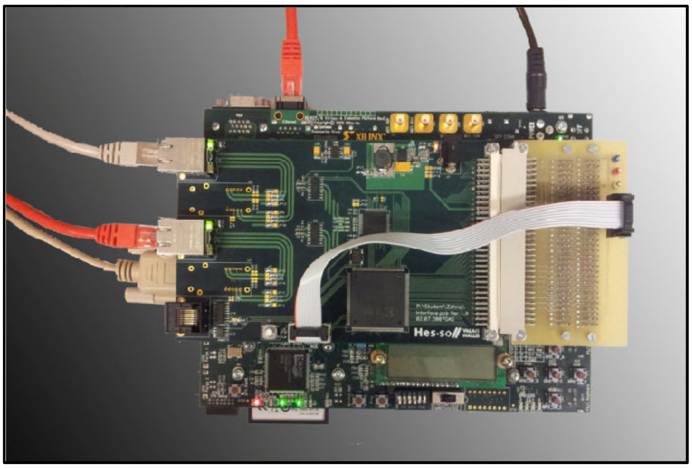
Responsable de l'orientation
Leiter der Vertiefungsrichtung: .... [1] Etudiant/*Student:* ....

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* .... Visa du secrétariat / *Visum des Sekretariats* ....

Ethernet traffic measurement

Report



Summary

Domain Engineering Sciences
Rte du Rawyl 47
CH-1950 Sion 2
Phone +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Degree Systems Engineering

Option Infotronics

Diploma 2011

Yann Santschi

Ethernet traffic measurement

Professor	François Corthay
Expert	Patrik Arlos

Karlskrona, November 25th 201

1. Introduction

1.1 Overview

This diploma work is the continuation of a previous diploma work on an Ethernet Measurement Point (MP). This project takes place in a Distributed Passive Measurement Infrastructure (DPMI) which consists of analyzing the traffic in computer network. The aim of this project is to build a network Measurement Point and interface it with an existing DPMI.

1.2 Initial state

The following components are already done:

- A passive wiretap board able to capture packet in both directions of the cable.
- A converter board that connects the wiretap board and the development board.
- A VHDL Ethernet controller design that implements IP, ICMP, ARP, UDP and DHCP made in the HES-SO Valais
- A VHDL MP controller design without communication protocols stack made by Silvan Zahno and Carlo Arnold.

1.3 Specifications

In this diploma work, following work must be done on the Measurement Point:

- It has to interface to a measurement zone using the DPMI API. As discussed with Mr. Arlos, the measurement point must be able to auto-configure at boot using DHCP and communicate using DPMI protocol in its version 0.7
- The Ethernet packet filters have to be finalized: for the moment there are only two fixed filters. The Measurement Point must be able to receive, update and delete filters. As discussed with Patrik Arlos it must work with at least one filter and the measurement point must be able to send captured data to consumers using Ethernet Multicast.
- Time must be synchronized using a GPS receiver.

1.4 Development environment

The following development tools are used:

- Mentor Graphics HDL Designer, version 2009.2
- Mentor Graphics ModelSim, version 6.6a, revision 2010.3
- Xilinx ISE, version 12.1
- QT Development Framework 4.7.3 - http://qt.nokia.com

For testing purposes, the following tools are used:

- Wireshark, version 1.4.4 - http://www.wireshark.org
- Oracle VirtualBox, version 4.0.4 - http://www.virtualbox.org
- Ubuntu Server, version 11.04 - http://www.ubuntu.com
- DPMI software version 0.7 source code - http://194.47.151.119/releases

The FPGA is programmed in VHDL.

1.5 Hardware

This chapter gives a quick overview about hardware components used in this project

1.5.1 Xilinx ML-405 development board

The Xilinx ML-405 development board has a Xilinx Virtex4 XC4FX20 FPGA. The board features used in this project are:

- Expansion headers, for connecting the wiretap board
- RS232 Serial port for debugging

The system clock (SYSCLK) frequency is 100MHz. For more information about this board, see the user Xilinx user guide.

1.5.2 Wiretap and converter board

The wiretap board is a passive Ethernet frames capture board. It has a dual PHY chip where both RX channels listen to each direction of the cable using high-impedance buffers. TX channels aren't used. The wiretap board is totally invisible in the network.

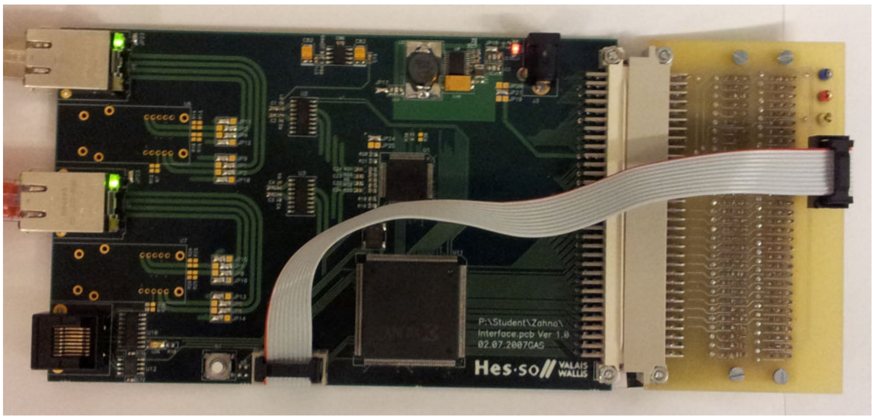


Figure 1 : Wiretap and converter board

The wiretap board has also a CPLD programmed to put data coming through the MII on 4 bits to a 16 bits bus for each capture direction. It doesn't calculate frame checksum. Schematics of the board are in appendix 1, "Wiretap Interface HDL schematics" and HDL Designer project in appendix 2, "Wiretap Interface HDL Designer project".

The converter board connects wiretap board to Xilinx ML-405 development board. It permits JTAG chain extension for programming the CPLD. Schematics are in appendix 3, "Converter Board schematics".

1.5.3 GPS board

The GPS board interfaces the NAVMAN Jupiter 30 development board to use it as time distribution server in this project. This GPS sends a PPS (Pulse Per Second) signal and has a SiRF binary serial line for getting time information.

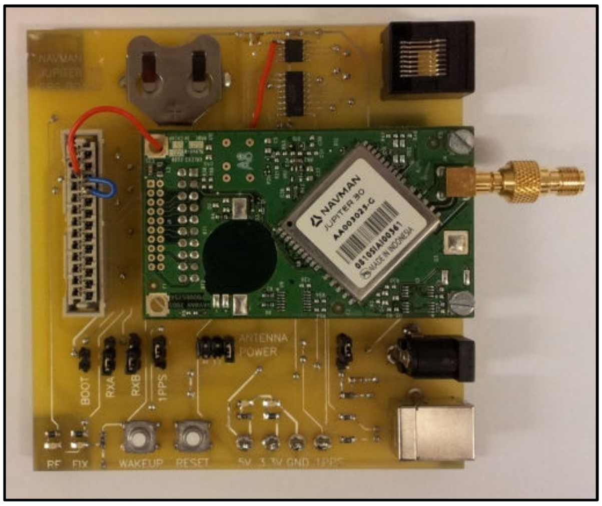


Figure 2 : GPS board

GPS Board documentation and schematics are in appendixes 4, "GPS Board documentation" and 5,"GPS Board schematics".

Modifications are made to the board in order to transmit GPS SiRF binary serial line directly on RS-422 line and to retransmit directly "delay_in" signal on "delay_out" signal. The modifications made are in the following figure:

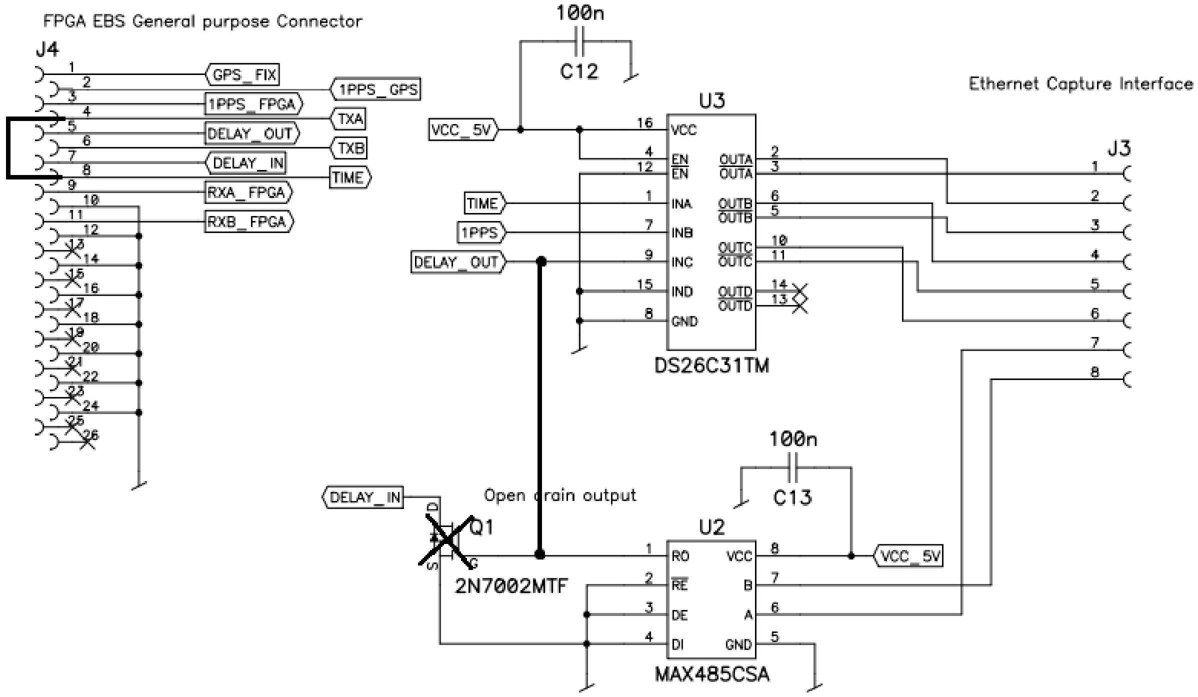


Figure 3 : GPS board modifications

1.6 About this document

All data sizes given in this document are in bytes (8 bits). The appendixes that cannot be printed are on the CD. The whole HDL Designer project is in appendix 7 on the CD, because printing source code and diagrams would need a huge quantity of paper.

2. Distributed Passive Measurement Infrastructure

This chapter is an overview about the Distributed Passive Measurement Infrastructure (DPMI). To allow an efficient network analysis, it's important to have up-to-date and accurate measurement data. The DPMI allows capturing and filtering network frames and storing them with a timestamp for further analysis by users.

The DPMI consists of the following parts:

- The Measurement Area (MAr)
- The Measurement Point (MP)
- At least one consumer

All these devices are in the Measurement Area Network (MArN). For more explanation about DPMI, see the documentation of Patrik Arlos (PAM2005 paper, "A Distributed Passive Measurement Infrastructure").

2.1 Measurement Area

The Measurement Area (MAr) handles communication with the MPs. It is a dedicated network (to not influence measurements or overload network with measurement data) which contains at least one Measurement Area Controller (MArC), one Measurement Point (MP) and one consumer, as in following figure:

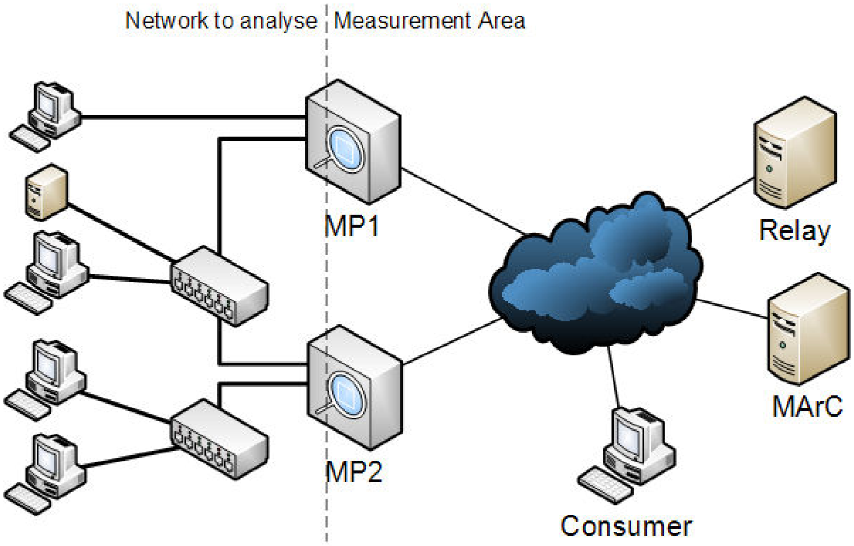


Figure 4 : Measurement Area

The MArC is the main component in the MAr. It manages MPs by tracking their status, giving them filters and providing the user a graphical user interface to set-up and control measurements.

2.2 Measurement Point

The Measurement Point (MP) is the device that captures packets, filters packets and distributes measurement data to consumers. Its structure is the following:

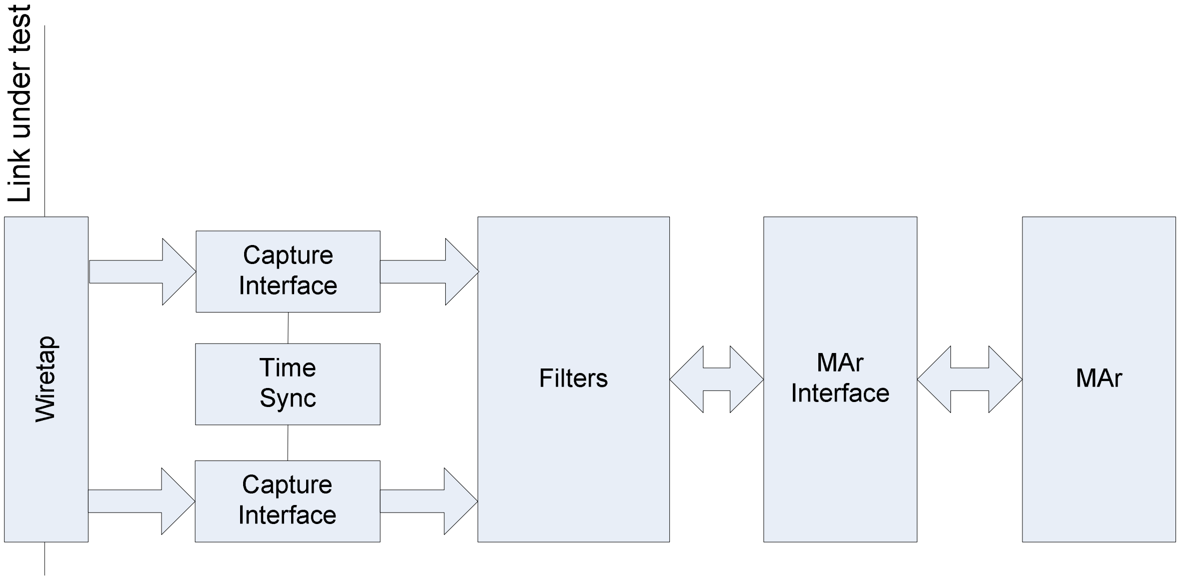


Figure 5 : MP structure

The wiretap is able to tap one link in both directions. When captured data matches filter rules, it is stored in a buffer with a capture header. This header contains notably a timestamp and a capture interface identifier. When the buffer is full data is sent to the filter specified consumers. The filters are supplied by the MArC. They contain rules for filtering Ethernet destination address, Ethernet source address, VLAN TCI, Ether type, IP protocol, IP source address, IP destination address, UDP/TCP source port and UDP/TCP destination port.

2.3 Consumer

The consumer is an application that reads and stores measurement frames sent by MPs. After that users can retrieve this data for analysis.

3. Network protocols overview

The goal of this chapter is to give a quick overview about the network protocols those are used in this project.

3.1 Ethernet

Ethernet is a commonly used local area network protocol. There are many physical implementation of it but the only one that used in local networks is the twisted pair cable with an RJ45 connector, because of its low price. In this project, the wiretap is able to capture frame on such transmission media at a speed of 10 and 100 MBit/s. In the OSI model, Ethernet is in levels 1 (physical) and 2 (link).

The Ethernet frame structure is as following:

Field name	Size	Description
Preamble	7	7 bytes containing 0b10101010
Start delimiter	1	1 byte containing 0b10101011
Destination MAC address	6	Frame destination
Source MAC address	6	Frame source
VLAN TPI (optional)	2	If the value is 0x8100 the VLAN tag is set
VLAN TCI (optional)	2	VLAN ID and priority
Ether type	2	Indicates what type of data is in the payload field. In example for the IP protocol it is 0x0800 or for ARP it is 0x0806
Payload	46-1500	Data
CRC	4	32 bit CRC of all preceding fields (in white)
Interframe gap	12	Minimum pause time between two frames.

Table 1 - Ethernet frame

The fields in gray don't contain data. They are for MAC (Media Access Control).

The MP sends measurement frames using Ethernet multicast. When the least significant bit of the first byte of the destination MAC address has the value '1', the Ethernet frame is treated as multicast. For example, the MAC address 01:00:00:00:12:34 is a multicast address.

The minimal frame size (without VLAN tag and a payload of 46 bytes) is 64 bytes. The maximal frame size (with VLAN tag and a payload of 1500 bytes) is 1522 bytes.

In a gigabit network, the payload size can be bigger than 1500 bytes. This kind of frames is called "Jumbo Frames". Because it isn't commonly used, these frames must only be sent in a network with devices that can support them.

3.2 Internet Protocol

Internet Protocol is the most used protocol for transmitting and relaying packets across networks. Each device has a unique address (except for private address ranges). The version that is currently used in networks is IPv4. A new version IPv6 was developed because there is a lack of public addresses. The version of IP implemented in this project is IPv4. In the OSI model IP is in level 3 (network).

The IPv4 datagram structure is as following:

Field name	Size	Description
Version	1/2	Version of protocol. For IPv4 it is 4.
Header length	1/2	Length of the header in words of 32 bits. Without options length is 5. Maximal length is

Field name	**Size**	**Description**
		15.
Differentiated Service Code Point	6/8	Used for real-time data streaming
Explicit Congestion Notification	2/8	End-to-end notification of network congestion. The use of this field is optional.
Total length	2	Length of header and data in bytes. Minimal length is 20 (only the header).
Identification	2	Used to reconstitute packets when they are fragmented
Flags	3/8	Fragmentation state
Fragment offset	5/8	Fragment position from first datagram
Time to live	1	Packet life time. This value is decremented at every router and when it equals 0 packet is dropped.
Protocol	1	Type of data. For ICMP it is 0x01, for TCP it is 0x06 and for UDP it is 0x11.
Header checksum	2	Checksum of IP header
Source IP Address	4	Datagram source
Destination IP Address	4	Datagram destination
Options (optional)	0 - 40	Additional header field that isn't often used.
Data		IP datagram payload. Its size depends of the data and of the link layer protocol maximal payload length.

Table 2 : IP datagram format

When IP is used over an Ethernet network, the IP datagram takes place in the Ethernet payload field. The Ether type field is set to 0x0800.

3.3 User Datagram Protocol

UDP (User Datagram Protocol) is one of the principal protocols used in Internet. It is connection-less and easy to use. Each entity is defined by an IP address and a port number. UDP is OSI level 4 (transport) and the UDP datagram takes place in IP data field.

The UDP datagram structure is as following:

Field name	**Size**	**Description**
Source port	2	Port of the sender
Destination port	2	Port of the receiver
Length	2	Length of payload and data
Checksum	2	Checksum computed on UDP header, data and on an IPv4 pseudo header.
Data		User data

Table 3 UDP datagram format

3.4 Other used protocols

Other protocols are used in this project:

- ARP (Address Resolution Protocol): this protocol is used to get the MAC address of a network device from its IP address. This FPGA design can only respond to requests.

- DHCP (Dynamic Host Configuration Protocol): this protocol is used to get automatically an IP address from a DHCP server. This FPGA design can get its IP address using this protocol.
- ICMP (Internet Control Message Protocol): this protocol is a part of the Internet Protocol Suite and it is used to transport control and error messages. This FPGA design can respond to "Ping" requests.
- TCP (Transmission Control Protocol): this protocol is a connection-oriented protocol for transmitting data. This FPGA can't handle this protocol but can recognize it in the packet filter.

4. DPMI Protocol

This chapter is written in accordance with MArC source code (appendix 6, "DPMI version 0.7 source code") and oral information given by Patrik Arlos and David Sveningsson.

Communication between MArC and MP is made using UDP/IP. The measurement frames are sent to the consumer using Ethernet multicast.

4.1 Messages sequences

4.1.1 Initialization

After IP auto-configuration using DHCP, the MP initializes itself with the MArC according to the following sequence:

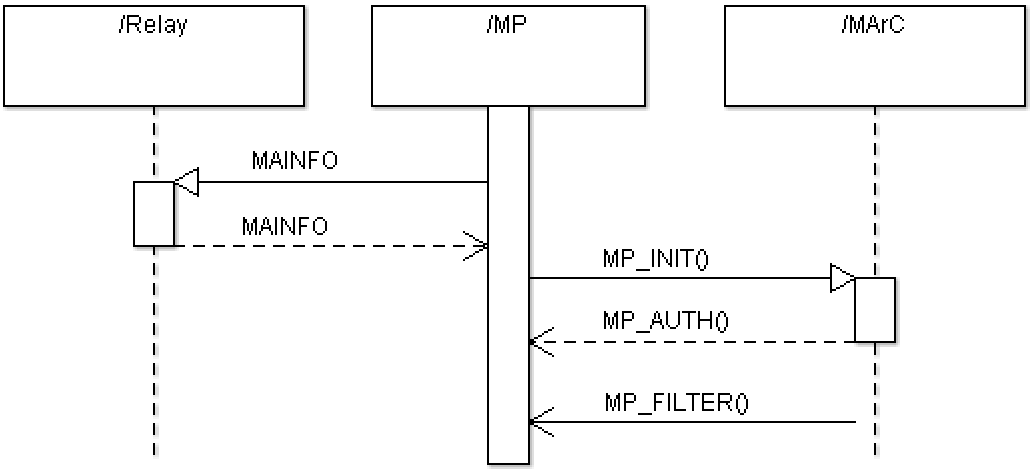


Figure 6 : Initialization sequence

First the MP broadcasts an empty MAINFO message and waits on relays response. The relay writes MArC IP address and port in the message and sends it back to the MP.

After that the MP knows how to reach the MArC and sends MP_INIT message to the MArC. The MArC will check if the MP is authorized and sends a MP_AUTH message to the MP for authorizing it (or not). If the MP is authorized and filters are set, MArC sends them after authorization.

MP can be managed using the MArC web interface.

4.1.2 Status message

Once the MP is authorized it sends regularly (approximately every 5 seconds) a status message to the MArC.

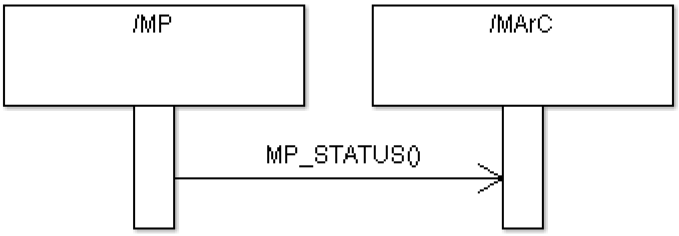


Figure 7 : Status message sequence

This message contains information about the number of active filters, packet counters, etc.

4.1.3 Measurement frames

The MP stores captured frames that match filters in a buffer. When this buffer is full the MP sends the data to the consumer defined in filter specification.

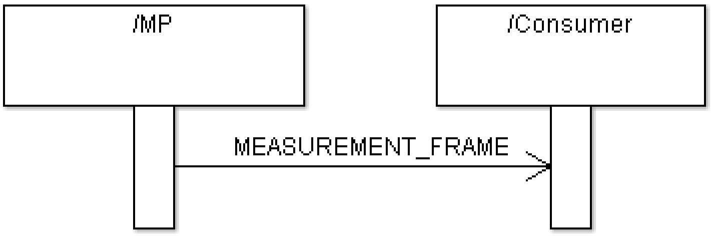


Figure 8 : Measurement frame sequence

In the current DPMI implementation the consumer is an Ethernet multicast client.

4.1.4 Filter update

If a filter is added or needs to be updated, the MArC sends a filter request to the MP. When the MP is ready to change the filter, it sends the filter request back and the MArC sends the filter.

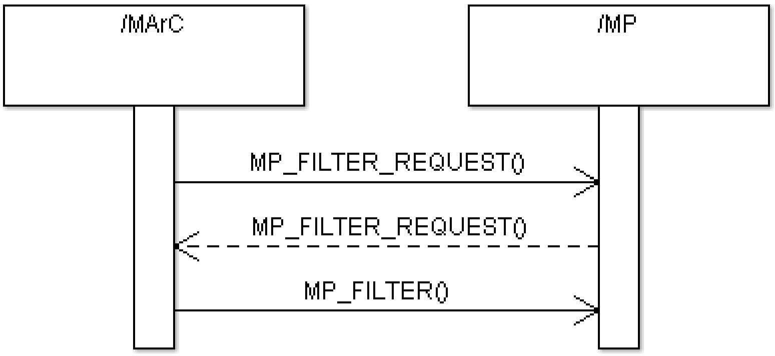


Figure 9 : Filter update sequence

4.1.5 Filter deletion

The MArC sends the MP_FILTER_DELETE message to the MP when a filter is removed. It there is data in the filter buffer, the MP will send it to the consumer before deletion.

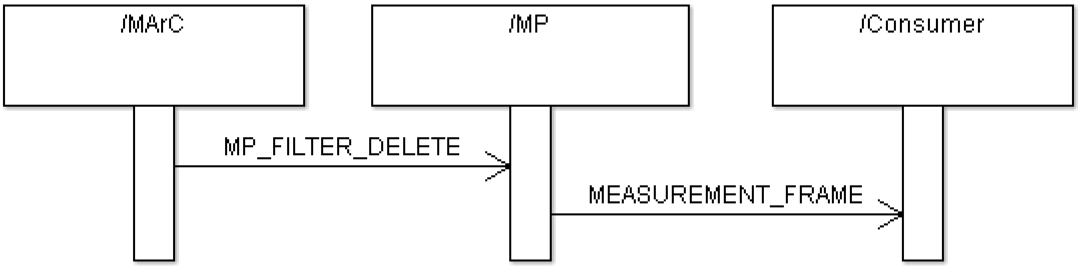


Figure 10 : Filter deletion sequence

4.2 MAINFO message

The MP sends a UDP broadcast message indicating its version on port 1500. A relay sends the message back after completion of the other fields.

MAINFO message contains following fields:

Field name	Size	Description
Version	4	MP client version. The version for the FPGA MP should be 3.
Address	16	MArC IP address (in full text)
Port	4	MySQL database port on the MArC. For compatibility only.
Database	64	MySQL database name. For compatibility only.
User	64	MySQL user name. For compatibility only.
Password	64	MySQL password. For compatibility only.
PortUDP	4	UDP port for communicating with the MArC.

Table 4 - MAINFO message

Remark: the current MP implementation requires that MArC and relay are on the same machine for the following reason: the MP needs the MAC address from the MArC too. For retrieving it it's necessary to send an ARP request and the Ethernet Dissolver design can't do that. Therefore the address field isn't considered.

4.3 MP_INIT message

After the MAINFO-reply from the relay, the MP sends MP_INIT message for trying to authenticate with the MArC.

MP_INIT message contains following fields:

Field name	Size	Description
Type	4	Message identification. MP_INIT value is 1
HWAddr	6	Ethernet MAC address of the MP.
Padding	2	For compatibility only.
Hostname	200	Hostname of the MP.
IPAddress	4	IP address of the MP.
Port	2	UDP port which the MP listens to.
maxFilters	2	Maximum number of filters the MP can handle.
noCI	2	Number of capture interfaces the MP has.
MAMPid	16	Unique ID string given by the MArC in the MPAUTH message.

Table 5 - MP_INIT message

Remarks:

- In the FPGA design, the UDP port which the MP listens to is 1212.
- The hostname of the MP is "FPGA"
- Even if the MP captures data in both directions, it has only one capture interface (named "NIC0X". The "X" is for differentiating direction.

4.4 MP_AUTH message

The MP_AUTH message is the response give by the MArC after the MP_INIT message. It contains the MAMPid that will uniquely identify the MP. If the MAMPid field is empty, MP isn't authorized.

MP_AUTH message contains following fields:

Field name	Size	Description
Type	4	Message identification. MPAUTH value is 128.
MAMPid	16	Unique ID string given by the MArC. This is the value that must be used by the MP.
Version	4	DPMI protocol version.

Table 6 - MP_AUTH message

Remarks:

- The given MAMPid is usually MP Hostname with concatenation of two alphanumerical characters. For example if the hostname is "FPGA", the MAMPid can become "FPGA23".
- Version field contains major version (2 bytes) and minor version (2 bytes). Current DPMI version is 0.7.

4.5 MP_FILTER message

The MP_FILTER message contains filter data for the MP. Each field has a value and a mask, except "ip_proto" that only has the value.

It contains following fields:

Field name	Size	Description
Type	4	Message identification. MPFILTER is 67.
MAMPid	16	Unique ID string given by the MArC.
FilterID	4	Filter identifier
Index	4	Indication of which fields should be tested
Iface	8	Capture interface whose filter applies
vlan_tci	2	VLAN TCI
eth_type	2	Ethertype
eth_src	6	Ethernet source MAC address
eth_dst	6	Ethernet destination MAC address
ip_proto	1	IP Protocol
padding	32	For compatibility only
src_port	2	UDP or TCP source port
dst_port	2	UDP or TCP destination port
vlan_tci_m	2	VLAN TCI mask
eth_type_m	2	Ethertype mask
eth_src_m	6	Ethernet source MAC address mask
eth_dst_m	6	Ethernet destination MAC address mask
padding	32	For compatibility only
src_port_m	2	UDP or TCP source port mask
dst_port_m	2	UDP or TCP destination port mask
consumer	4	Not used
caplen	4	Number of bytes to capture
Dest	30	Consumer address. The content of this field will be

Field name	Size	Description
		explained in a further chapter.
version	4	Version of DPMI. Current version is 0.7
starttime	12	Capture start time
endtime	12	Capture end time
mampid	8	Not used
ip_src	4	IP source address
ip_src_m	4	IP source address mask
ip_dst	4	IP destination address
ip_dst_m	4	IP destination address mask

Table 7 - MP_FILTER message

The "Index" field indicates which data has to be filtered in captured frames. Each field to filter has its corresponding bit.

Remarks:

- Some fields cannot be set in the current implementation of MArC web interface and therefore they aren't considered by the MP. These fields are "starttime" and "endtime".
- As the MP has only one capture interface, the "iface" field is ignored.

4.6 MP_FILTER_REQUEST message

The MP_FILTER_REQUEST message is sent by the MArC for asking the MP to reload a specific filter, identified by its filter id.

MP_FILTER_REQUEST message contains following fields:

Field name	Size	Description
Type	4	Message identification. MP_FILTER_REQUEST value is 66.
MAMPid	16	Unique ID string given by the MArC.
FilterID	4	Filter identification.

Table 8 - MP_FILTER_REQUEST message

When the MP receives this message, it sends it back to the MArC.

4.7 MP_FILTER_DELETE message

The MP_FILTER_DELETE message is sent by the MArC to remove a specific filter, identified by its filter id, from the MP.

MP_FILTER_DELETE message contains following fields:

Field name	Size	Description
Type	4	Message identification. MPFILTERDELETE value is 69.
MAMPid	16	Unique ID string given by the MArC.
FilterID	4	Filter identification.

Table 9 - MP_FILTER_DELETE message

4.8 MP_STATUS message

The MP_STATUS message is sent by the MP to the MArC to indicate its current status.

MP_STATUS message contains following fields:

Field name	Size	Description
Type	4	Message identification. MPSTATUS value is 65.
MAMPid	16	Unique ID string given by the MArC.
packet_count	4	Captured packet count
match_count	4	Filter match count
status	1	Indicates MP status
nofilter	1	Assigned filter count
Noci	1	Capture interfaces count
padding	1	Padding

Table 10 - MP_STATUS message

Remark:

- I couldn't find the values to write in "status" field. I decided to set the field to 0x00 when the MP is idle and to 0x01 when it's capturing data.

4.9 Measurement frame

When a filter buffer is full, the MP sends captured data to the consumer specified in the filter using Ethernet multicast. The payload of the capture frame has the following format:

Send header	Capture header 1	Captured data 1	Capture header 2	Captured data 2	...

Table 11 : Measurement frame

The Send Header (SH) contains following fields:

Field name	Size	Description
sequence_nr	4	Measurement frame identifier. Its value is incremented by 1 every time a frame is sent. Value is modulo 1024.
nopkts	4	How many packets are in the frame
flush	4	Indicates it's the last captured packet
version	4	Version of the file format (not used)

Table 12 : Send header

The Capture Header (CH) contains following fields:

Field name	Size	Description
nic	8	CI identifier
mampid	8	MP identifier
timestamp	12	Frame arrival time
len	4	Length of the whole frame
caplen	4	Length of captured data

Table 13 : Capture header

Ether type field value for DPMI is 0x0810.

5. VHDL Development

5.1 Design overview

The Measurement Point design can be separated in five distinct parts:

1. Ethernet Dissolver: this is the network protocol stack that can send and receive Ethernet frames. It handles IP, ARP (reply only), ICMP (ping), UDP and DHCP (client) protocols.
2. DPMI interface: it is the part that will communicate with the MArN. It must be able to identify itself with the MArC, receive commands and send captured data to the Consumer.
3. Filters: there can be one or more of them. They have to compare captured frame with a set of rules and store it in a buffer after adding a header if the captured frame matches to the filter rules.
4. Capture Interface: it receives frames captured by the wiretap board, checks the CRC and transmits data to the filters.
5. Time Synchronization: this part is for having an accurate timestamp on each captured packet. In this project I use an own made GPS board. In BTH there is an infrastructure using Endace products and I will give some information about it.

In addition to these blocks there is a debug block for sending some information over a RS232 serial line.

5.2 HDL Designer project, libraries and simulation files

The HDL Designer library in which the project files are created is "EthernetDissolver". The test benches are in "EthernetDissolver_TB" and will be simulated in ModelSim. The "Board" library is the design that will be synthesized with Xilinx ISE and programmed in the FPGA. The VHDL development tool chain works as follow:

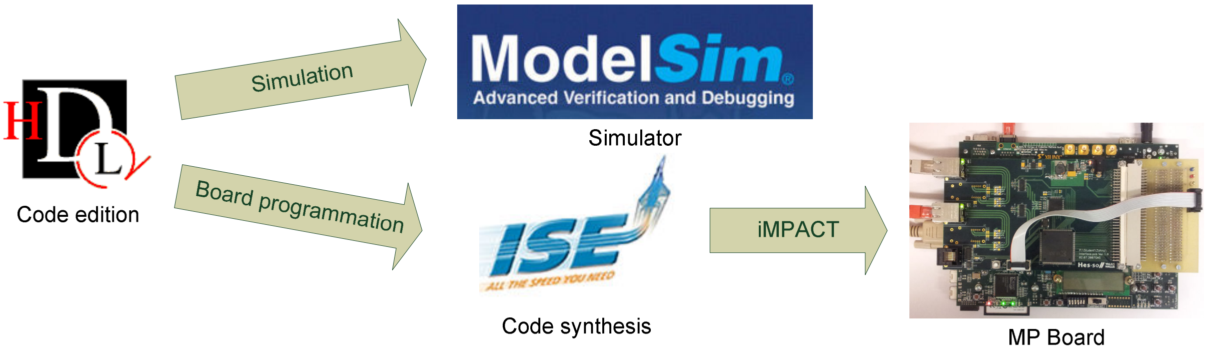


Figure 11 : VHDL development tool chain

The VHDL code is written in HDL Designer. For simulation the code is compiled and executed in ModelSim. For programming the board, the VHDL code needs to be synthesized in Xilinx ISE. A programming file is generated and can be downloaded in the FPGA using Xilinx iMPACT.

All the constant values and data type definitions for DPMI are in the following file of the project: "EthernetDissolver/dpmi_definition".

For simulation, Ethernet traffic is generated using Wireshark capture files. These files are in

The MP design made by Silvan Zahno and Carlo Arnold isn't used anymore. I made a new design from scratch because there were too many changes to make on the old design:

- Capture Interface data bus had to be enlarged on 16 bits
- Filtering had to be redesigned in order to receive them from the MArC.
- DPMI interface had to be created completely.
- Project is using hardware dependent libraries (Xilinx Cores)

5.3 Ethernet Dissolver

Ethernet Dissolver design is able to handle Ethernet frames. It can handle IP-based communication. Data in incoming Ethernet frames is analyzed in each layer level and sent to the next layer if protocol is recognized. Data to send is encapsulated in each layer and sent through MII interface. The following figure shows how it's implemented:

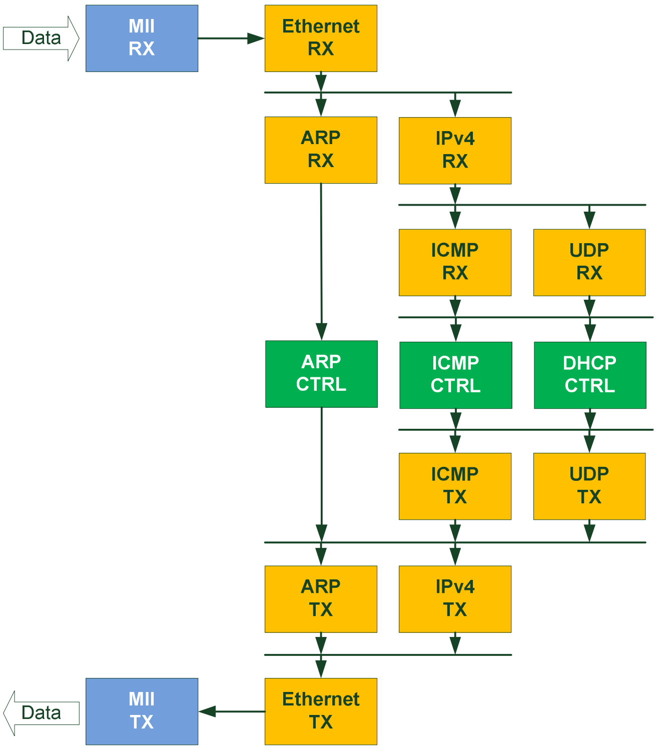


Figure 12 : Ethernet Dissolver structure

The DPMI controller, which contains all the other parts of the project, takes place next to the DHCP controller block.

5.4 DPMI interface

The DPMI interface is made of three principal blocks:

- Controller heart: it is the main state machine that controls all the other blocks.
- Controller RX: it is the state machine that manages reception of the UDP data coming from the protocol stack. It only read messages the "Controller heart" asks for. Other messages are discarded. The received data is given to the "Controller heart" or to the "Filter manager" if it is filter data.
- Controller TX: it is the state machine that manages data sending over UDP or Ethernet. It only sends messages when the "Controller heart" asks for or when a filter buffer is full.

Each controller block communicates with the "Filter manager" block and has its own external registers and counter blocks. These blocks will not be described here. The DPMI interface architecture is as following:

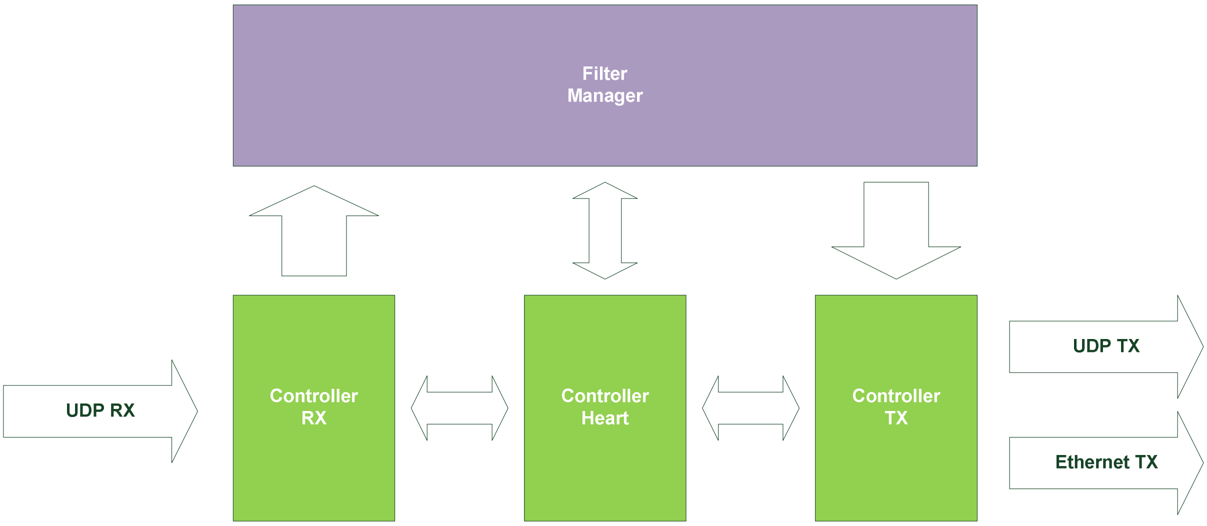


Figure 13 - DPMI Interface

The RX controller state machine transmits filter data directly to the filter manager and the TX controller receives data to send (measurement frames) directly from the filter manager. The controller heart state machine just handles control signals.

Remark: the DPMI interface can handle UDP consumers but can't get their MAC address. If the consumer is UDP, the destination MAC address will be the broadcast MAC address. It isn't a good way to proceed but sending ARP requests isn't implemented in the design yet. Moreover the consumer software supports only Ethernet multicast.

5.5 Frame capture

The frame capture part has two Capture Interfaces (A and B) for capturing data in both directions on the link under test. The Time Synchronization blocks generates a timestamp that will be added to each captured frame. The Interface Selection block selects which Capture interface will be able to transmit the captured data to the filters. Selection works with a round-robin algorithm. The next figure illustrates the frame capture architecture.

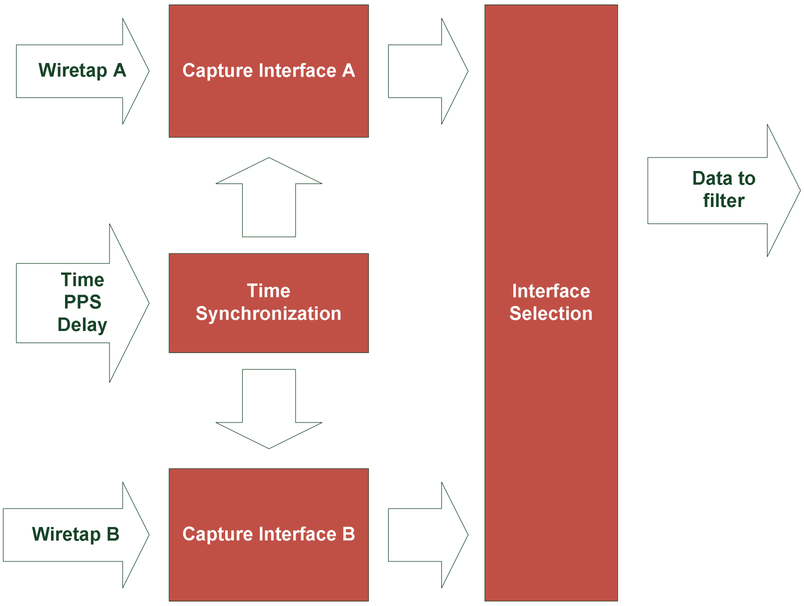


Figure 14 - Frame capture design

Each Capture Interface has following behavior: the RX Controller block receives data from the wiretap, writes it in the dual-port buffer and in the CRC32 computing block. If CRC is correct the Capture Interface Heart block will notify Interface Selection block that it has new data and disable frame capture. Once data is read, frame capture is enabled again. The next figure shows Capture Interface design:

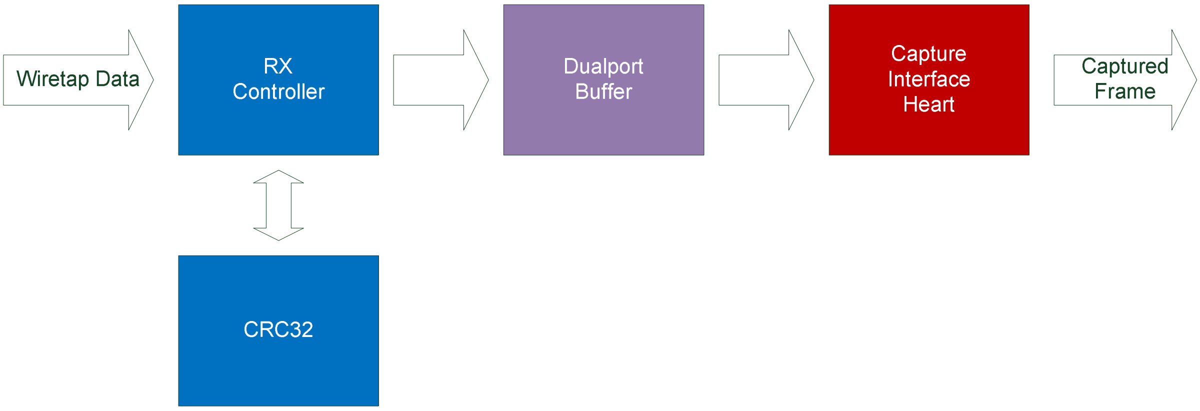


Figure 15 - Capture Interface design

There are two clock domains in the Capture Interface:

- RX Clock: it is the MII Receiver clock. Depending on link speed this clock can be at 2.5 MHz for 10 MBit/s or 25 MHz for 100MBit/s.
- SYS Clock: it is the system clock of the FPGA at 100 MHz.

A dual-port buffer is used to synchronize received data between clock domains. The buffer must be able to contain a complete Ethernet frame (up to 1522 bytes). Buffer size is set to 2 Kbytes (2 bytes data bus and 10 bits for address bus).

In order to avoid frame loss the dual-port buffer has to be read before a new frame arrival. This must be done during the Ethernet inter-frame gap (12 bytes), preamble (7 bytes) and start delimiter (1 byte). The available time for reading captured frame for each interface with two of them, without losing data can be calculated as follow:

$$t_{read(\max)} = \frac{t_{gap}}{n_{interfaces}} = \frac{(12+7+1)Bytes * \frac{8Bits}{Byte} * \frac{1}{\frac{100Mbit}{s}}}{2} = \frac{1.6\ us}{2} = 800\ ns$$

In this design t_{read} depends of "caplen" value in the filter module. T_{read} will be calculated in chapter "Filter block".

5.6 Filters

The capture data filtering is made of two distinct parts:

- The filter module: it filters captured frames and if data matches filtering rules it adds a capture header and writes the data into a buffer. There can be more of them. They treat captured frame in parallel.
- The filter manager: it receives filtering rules from the DPMI interface and set them into a free filter. When a filter buffer is full, it sends the data and the destination information (consumer) to the DPMI interface.

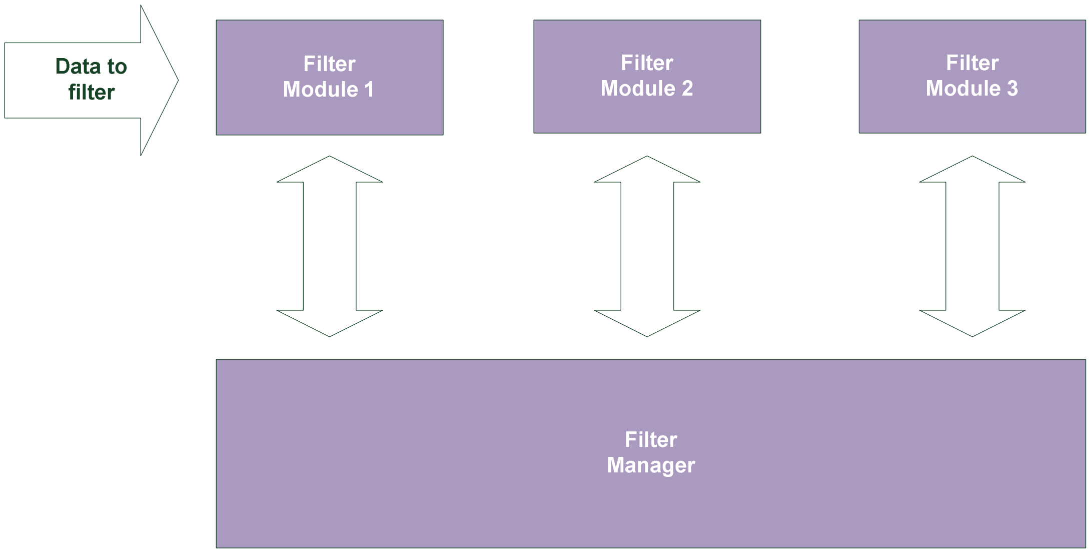


Figure 16 - Filter design

For communication from Filter Manager to all Filter Modules there is one record with following signals:

Signal name	Function
read_frame	Indicates that a new captured frame will be read and has to be filtered.
selected	Each filter has its own selection bit (at filter number position). Indicates which filter is managed.
add_filter	Indicates that new filter will be set or updated.
del_filter	Filter with corresponding filter_id will be deleted.
read_buffer	Order to empty filter buffer
Data	Filter data
is_filter_id	Indicates that filter data is the filter_id. Each filter checks if it corresponds to its own filter_id.
is_...	Indication of what kind of data is coming (one signal for each field)

Table 14 : Manager to filters record

For communication from Filter Modules each filter has its own record with following signals:

Signal name	Function
is_my_filter_id	Indicates that the filter filter_id matches filter_id on incoming record
filter_present	Indicates that the filter is set
buffer_full	Filter buffer needs to be sent. No more data can be stored.
buffer_data	Outgoing data bus
buffer_data_valid	Outgoing data is valid
buffer_end_of_data	Indicates that there isn't more data to send
matched_count	Count of filter matches
caplen_reached	Indicates that the filter doesn't need to read more data from the captured frame
destination	Outgoing data destination. If destination is Ethernet multicast it contains the MAC address. If destination is UDP it contains IP address and port.
dest_is_ethernet	Indicates that the consumer uses Ethernet multicast

Signal name	Function
dest_is_udp	Indicates that the consumer uses UDP.

Table 15 : Filter to manager record

This way of implementation allows expanding the number of filters in a very easy way. For adding a new filter block following steps have to be done:

- Update the value of "dpmi_maxfilters_c" constant.
- Place "filter_module" component in the "controller_dpmi" block.
- Connect signals and configure filter number in generic parameters.

The new filter will work without any other modification in the design.

5.6.1 Filter management block

The Filter Management block is made of two parts:

- The Filter Management Heart that decodes filter messages and transmits data to the Filter Interface. It also transmits buffer data to the DPMI TX interface.
- The Filter Interface that chooses the filter that will be managed. Each filter indicates if it is set and if it has data to send. The filter interface will chose one and make it communicate with the filter decoder.

Remark: Filter Interface block should determine which filter has the highest priority and decide which filter must save data if the captured frame matches to more than one filter. The filter with lowest ID has highest priority. This feature isn't implemented yet because only one filter fits in the used FPGA.

5.6.2 Filter module block

The Filter module block compares captured frames with filter rules received from the MArC. If the frame matches the filter it will be written in a buffer. It is made of four main blocks:

- Captured Frame Dispatcher: this block decomposes incoming frame in order to get data to filter field per field. It can handle VLAN tagging, IP, TCP and UDP.
- Filter: this block filters incoming frames according to the filtering rules.
- Buffer Manager: this block is the heart of filter module. It coordinates data filtering, buffering and sending.
- RAM: it is the data buffer. It must be big enough to contain the maximal Ethernet payload (1500 bytes). Its size is 2 Kbytes.

The following figure shows the Filter module architecture:

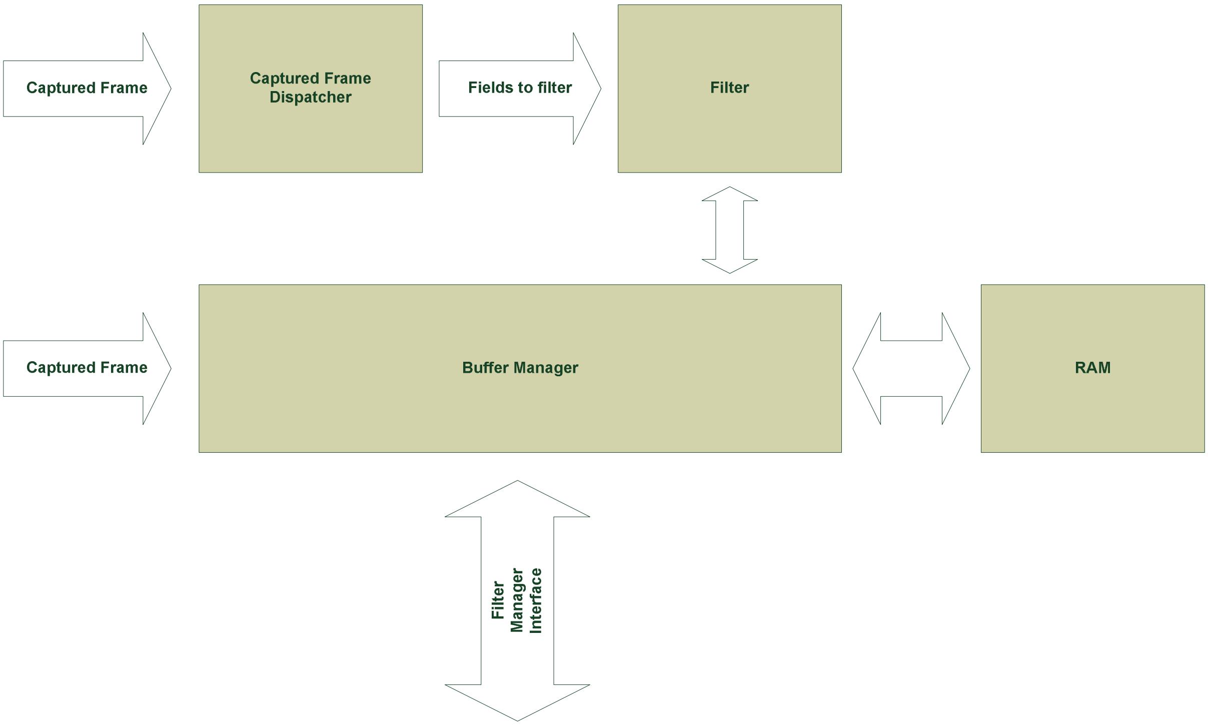


Figure 17 - Filter Module design

Captured data matches a filter when:

$$captured\ data \ \&\ filter\ mask = filter\ value\ \&\ filter\ mask$$

The buffer manager stores every incoming frame (within the specified capture length) into the RAM. If the capture matches the filters the Capture Header is written in the memory and the free memory address register is updated. If the capture doesn't match the filters written data will be overwritten at next frame arrival. When buffered data reaches the maximal size that can be sent in one measurement frame, the buffer manager activates the "buffer_full" signal and waits until the filter management reads stored data. The buffer size is calculated as follows:

$$size_{buffer} = ethernet_{maxpayload} - ip_{maxheader} - udp_{header} = 1500 - 60 - 8 = 1432\ bytes$$

IP header and UDP header size are removed for compatibility with UDP consumer. With a capture length set to 54 bytes in the filter, the buffer will be sent when its size is 1350 bytes. For sending this buffer it takes 13.7µs (including time to write headers). Each more byte in the buffer will take 10ns (t_{clk}) more. For 1432 bytes it would take 14.52µs.

During this period of time no more captured frames will be saved and this may cause data loss. During the simulations I saw that the "EthernetDissolver" design needs between 13µs and 50µs for initializing the transmission part of the design. This increases data loss in a significant way. With a network link at 100Mbit/s, every µs without capture can cause a loss of 12.5 Bytes. For avoiding that design has to be changed but I saw this too late to start to make big changes in the design.

If capture length ("caplen" parameter from filter) is too big, data can be lost too. For a 54 bytes capture length, the necessary time to write data and capture header in the buffer is t_{read1}=520ns. When two frames comes at the same time on each capture interface, the time to write both frames and headers in the buffer is t_{read2}=1060ns. Each more pair of bytes (16 bits) to capture will take 10ns more. Maximal capture length without losing data during buffering is:

$$caplen_{max} = 54 + \frac{(n_{ci} * t_{read\_max} - t_{read2})}{t_{clk}} * \frac{2\ bytes}{n_{CI}} = 54 + \frac{2 * 800ns - 1060\ ns}{10ns} * \frac{2bytes}{2} = 108\ bytes$$

Remark: If a captured frame matches more than one filter it must be saved in the filter with highest priority (lowest filter ID). The frame must not be duplicated. See remark above (Filter Management block).

5.7 Time synchronization

For having an accurate timestamp, time is synchronized using a GPS receiver. The following figure shows how the GPS board is connected to the MP:

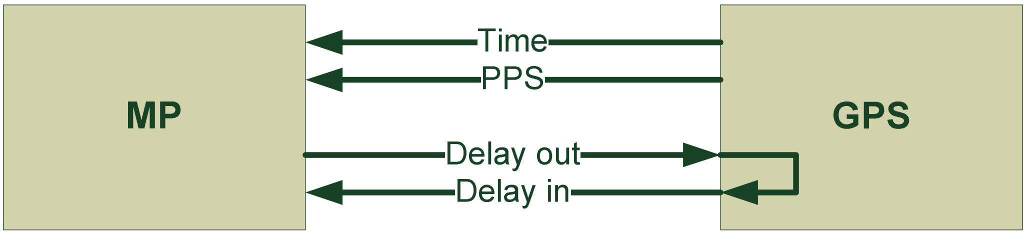


Figure 18 : MP and GPS

The GPS receiver (see appendix 4, "GPS Board documentation") sends time signal over a serial line using SiRF binary protocol at 38'400 bauds. This signal is redirected on "Time" signal. Every second it sends a 1µs pulse on the PPS signal. If the distance between the MP and the GPS receiver is long, PPS signal will be delayed. To measure the transmission delay the signals "Delay out" and "Delay in" were added.

The following figure shows how the time synchronization client block is implemented:

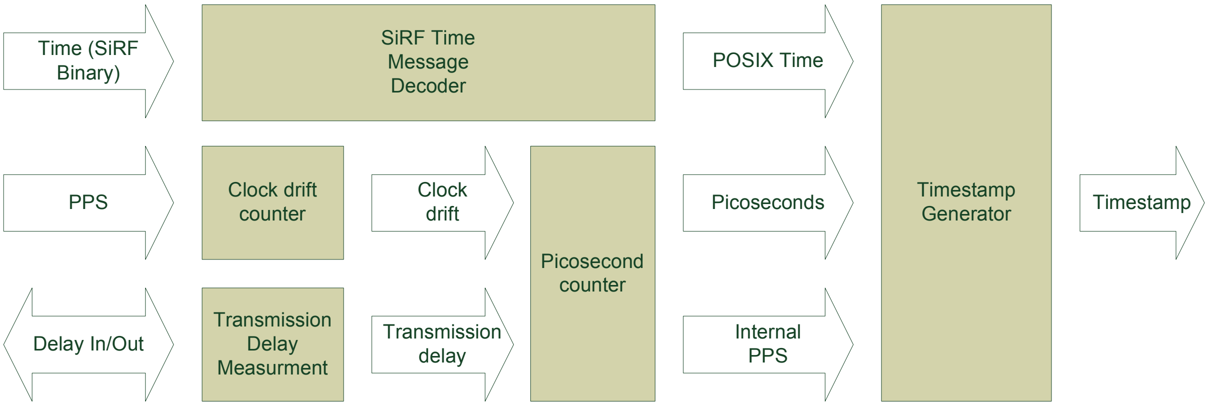


Figure 19 : Time synchronization design

The PPS signal indicates the beginning of the second sent in the next incoming SiRF message. For measuring clock drift, the number of clocks is measured between two PPS pulses. It's assumed that the PPS signal given by the GPS receiver is totally accurate. The picoseconds counter generates an accurate internal PPS signal using clock drift counter and transmission delay.

The required format for the capture timestamp is as following:

Field	Length	Description
Seconds (POSIX)	4	Seconds since POSIX Epoch
Picoseconds	8	Picoseconds

Table 16 : Timestamp format

The resolution of the "Picoseconds" field is limited by the FPGA clock frequency (100MHz in this case). The resolution can be calculated as following:

$$\Delta_{picocounter} = \frac{\mathbf{1}}{\boldsymbol{f_{clk}}} = \frac{\mathbf{1}}{\mathbf{100}\ \boldsymbol{MHz}} = \mathbf{10}\boldsymbol{ns}$$

Formula 1 - Picoseconds counter resolution

5.7.1 SiRF Binary protocol

SiRF Binary protocol is a standard protocol used by GPS receivers. Instead of NMEA where the data is sent as text, the messages are in binary format and it is the reason why I use this protocol in the FPGA design.

The format of a SiRF message is as following:

Field	Length	Description
Start sequence	2	Every message starts with the sequence 0xA0A2
Payload length	2	Payload length on 15 bits but not more than 1023
Payload	< 1023	Message payload. The first byte is the message ID
Checksum	2	Checksum on 15 bits
End sequence	2	Every message stops with the sequence 0xB0B3

Table 17 : SiRF binary message format

Every second the used GPS (Navman Jupiter 30) sends a "Clock Status" message that contains the GPS time at the last PPS. Its payload contains following fields:

Field	Length	Description
Message ID	1	Message identifier. For "Clock Status" it is 0x07
Extended GPS Week	2	Week number since GPS Epoch
GPS Time Of Week	4	Elapsed seconds since week beginning, multiplied by 100
SVs	1	Number of satellites used to compute this solution
Clock Drift	4	Rate of change of the Clock Bias
Clock Bias	4	Difference in ns between GPS time and internal clock
Estimated GPS Time	4	GPS Time estimated before the measurement

Table 18 : SiRF clock status message

The required fields for computing timestamp are "Extended GPS Week" and "GPS Time Of Week".

5.7.2 GPS Time and POSIX time

GPS Time is the time used by GPS system. It can be computed with this information:

- GPS Epoch: time zero of GPS. It is January 6th 1980 00:00:00.
- GPS Week: it is a modulo 1024 value that indicates the week number since last week rollover (when week number is 0 again). Last rollover (rollover 1) was August 22th 1999. As the GPS receiver sends the "Extended GPS Week" (week number since GPS Epoch) week rollover isn't an issue.
- Leap seconds: GPS Time doesn't handle leap seconds (15 since GPS Epoch). GPS Time is also 15 seconds ahead of UTC time.

POSIX Time (or UNIX time) is the UTC time expressed in seconds since POSIX Epoch (January 1st 1970 00:00:00). It handles leap seconds.

The POSIX timestamp of GPS Epoch is 315964800. POSIX timestamp can be computed from GPS time as following:

$$t_{POSIX} = t_{GPS\ EPOCH} + n_{WEEKS} * n_{SECONDS\ IN\ WEEK} + t_{TIME\ OF\ WEEK} - n_{LEAP\ SECONDS}$$

Formula 2 - POSIX timestamp from GPS Time

5.7.3 Transmission delay measurement

To improve timestamp accuracy the transmission delay measurement is measured between the GPS receiver and the MP. The propagation speed of an electrical signal in a wire is approximately 2/3 of light speed. The transmission delay of the PPS can be calculated as follow:

$$t_{delay} = \frac{cable\ length}{\frac{2}{3} * c_0}$$

Formula 3 : Transmission delay

Following table shows transmission times for various line lengths:

Cable length [m]	Delay [ns]
1	5
10	50
100	500

Transmission delay is measured as follows:

The "delay_out" signal is set to '1' and a timer is started. When "delay_in" comes to '1' the timer is stopped and divided by 2 to have transmission delay for only one direction. The measurement error for both directions is between 0 and $\Delta_{picocounter}$. The error for one direction is between 0 and $\Delta_{picocounter}/2$.

5.7.4 Endace Time Distribution Server

The time synchronization infrastructure used in BTH is Endace time distribution system. It works as follows:

- GPS Time and PPS signals are received by the Time Distribution Server (Endace TDS-2).
- TDS-2 has two time synchronization outputs for capture devices and has two more outputs for connecting repeaters (Endace TDS-6) in a daisy-chain.
- PPS and Time signals are relayed to each connected device.

The following figure shows how Endace Time Distribution works:

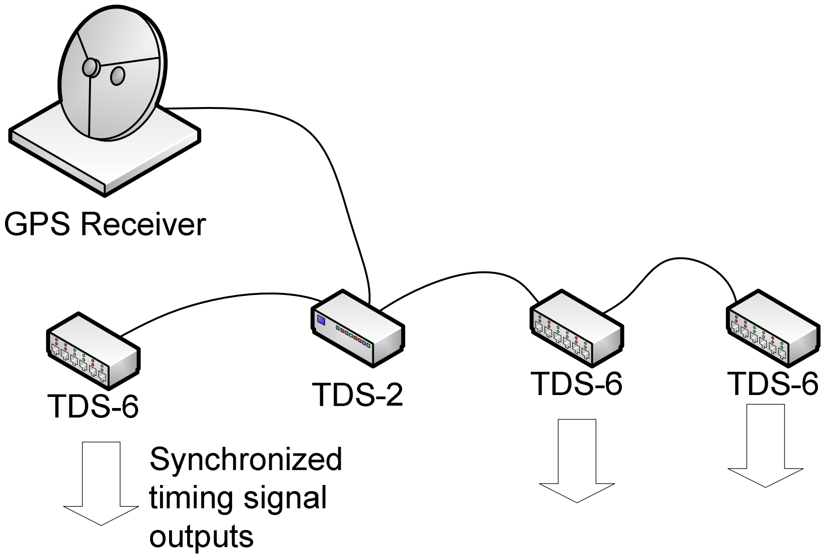


Figure 20 : Endace time distribution

Time signal is sent over a serial line. Its format depends on the type of used time source. Endace TDS-2 supports GPS and CDMA receivers. The GPS used for synchronizing time in BTH (Acutime 2000 from Trimble) sends NMEA (it is a text-based protocol) or TSIP (Trimble proprietary binary protocol) messages. The MP time synchronization only supports SiRF messages. Therefore Endace system can't be used with the MP for now.

More information about Endace TDS and Trimble Acutime GPS receiver can be found in corresponding documentations (see bibliography).

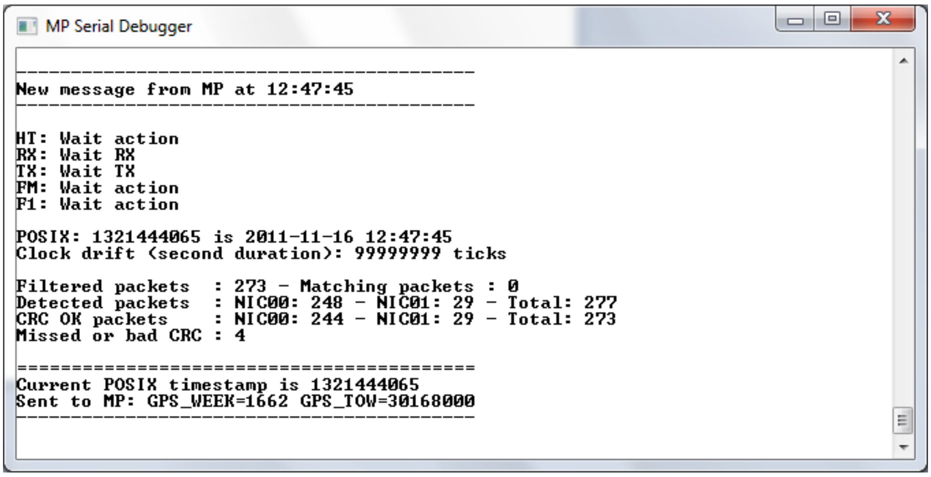
5.8 Serial debug interface

The serial debug block sends every second a debug message containing following information:

- DPMI heart state machine status (1 byte)
- DPMI Rx state machine status (1 byte)
- DPMI Tx state machine status (1 byte)
- Filter manager state machine status (1 byte)
- Filter 1 state machine status (1 byte)
- POSIX timestamp (4 bytes)
- Clock drift counter (4 bytes)
- Filtered packets counter (4 bytes)
- Filter matches counter (4 bytes)
- Count of all incoming frames on capture interface A (4 bytes)
- Count of all frames with good CRC and written to memory in capture interface A (4 bytes)
- Count of all incoming frames on capture interface B (4 bytes)
- Count of all frames with good CRC and written to memory in capture interface B (4 bytes)

Baud rate is set to 34'000 Bps.

I developed software in C++ using QT libraries for displaying human readable debug information in the system console. State machines states codes are described in "MPDebug.h" file. This application can only run in a Microsoft Windows environment. For running this application on another operating system serial line functions have to be modified. The source code and executable file are in appendix 8, "MPDebug software". The next figure shows how the debug application looks like:



```
-------------------------------------------
New message from MP at 12:47:45
-------------------------------------------

HT: Wait action
RX: Wait RX
TX: Wait TX
FM: Wait action
F1: Wait action

POSIX: 1321444065 is 2011-11-16 12:47:45
Clock drift (second duration): 99999999 ticks

Filtered packets  : 273 - Matching packets : 0
Detected packets  : NIC00: 248 - NIC01: 29 - Total: 277
CRC OK packets    : NIC00: 244 - NIC01: 29 - Total: 273
Missed or bad CRC : 4

===========================================
Current POSIX timestamp is 1321444065
Sent to MP: GPS_WEEK=1662 GPS_TOW=30168000
-------------------------------------------
```

Figure 21 : Serial debug application

This application allows detecting if one of the state machines is locked in a particular state. Not all the states have a status code, but only states susceptible to block the design have one. The debug software also converts computer time to GPS time and sends it in SiRF format over the serial line.

Remark: I didn't write the functions for reading and write COM port. I reused source code I got during a summer job.

6. Test benches

6.1 Simulation with ModelSim

The first tests are done in the ModelSim simulator. In order to simulate network traffic, the test bench has a block able to parse PCAP files and to send them via a MII interface. The behavior is also the same than with a real MII. For the Capture Interface inputs, data has to be adapted on a 16bit bus, like the wiretap board does.

For each test, the procedure is as following:

- Incoming messages must be recognized and treated well.
- Concerned states machines must react when events happen and go back in their "idle state" when job is done.
- Data written in outputs must correspond to DPMI frame format and contain relevant data.
- No signal must be in an unknown state (value 'X') in the simulator.

6.1.1 DPMI Interface

- The MAINFO request message must be sent automatically after receiving an IP address using DHCP. The controller must wait on the MAINFO response to continue.
- After receiving the MAINFO response, the MP_INIT message must be sent using the destination port specified in the MAINFO response. The controller must wait on the MP_AUTH message to continue.
- When the MP_AUTH message is received, the MAMPid must be updated and the controller must be ready to receive commands from the MArC.
- When a MP_FILTER message is received, the MP must check if there is a free place for this filter or if the filter ID matches an existing filter. If this is the case filter data is read and sent to concerned filter for configuring it.
- When a MP_FILTER_REQUEST message is received, the MP must send it back to the MArC.
- When a MP_FILTER_DELETE message is received, the MP must remove filter with corresponding ID.

6.1.2 Filter management and filters setup

- Make one filter be set correctly: all the filter registers are set with the correct filtering rules, it works.
- Make the design compile with three filters: it works.
- Make three filters be set correctly: all the filter registers are set with correct filtering rules in each filter. It works
- Try to add a fourth filter. It shouldn't work but not crash: the fourth filter isn't set and the design goes back to normal working state. It works.
- Try to remove filters: each filter can be removed. Capture stops when all the filters are removed. It works.
- Update filter rules: the filter registers are set with new filter rules. It works.
- Filter buffer must be sent when it is full: when the fixed limit is reached, data is sent over Ethernet and headers are correct. It works.
- Filter buffer must be sent when it isn't empty at filter removal: it works.

6.1.3 Capture Interface and data filtering

- Filter dispatcher must handle incoming data correctly with different types of frames. Testing with ARP (Ethernet but not IP), ICMP (IP but not TCP or UDP), TCP, UDP and VLAN (tag before Ether type) covers all the incoming frame types: each field in each case is correctly detected and sent to the filter. It works.
- Incoming frames must be filtered according to the filter rules: it works.
- Frames those match filter rules must be stored in the buffer with correct header: it works.

6.1.4 Time synchronization

- A SiRF binary message with a predefined GPS time is sent through the serial line. The generated POSIX timestamp corresponds to the predefined time: it works.
- The "delay_in" signal has 5µs delay on "delay_out" signal in the simulation. The measured time corresponds to 5µs: it works.
- Internal PPS signal is generated according to the measured clock drift. Clock drift is measured by counting clock rising edges between two GPS PPS pulses. It works.

6.2 Onboard test

I set up a test environment that consists of an Ubuntu Server virtual machine that acts as DHCP server, MArC and Consumer. The virtual network card is bridged to a physical network card of the computer and the MP board is directly connected. The following figure illustrates the test network:

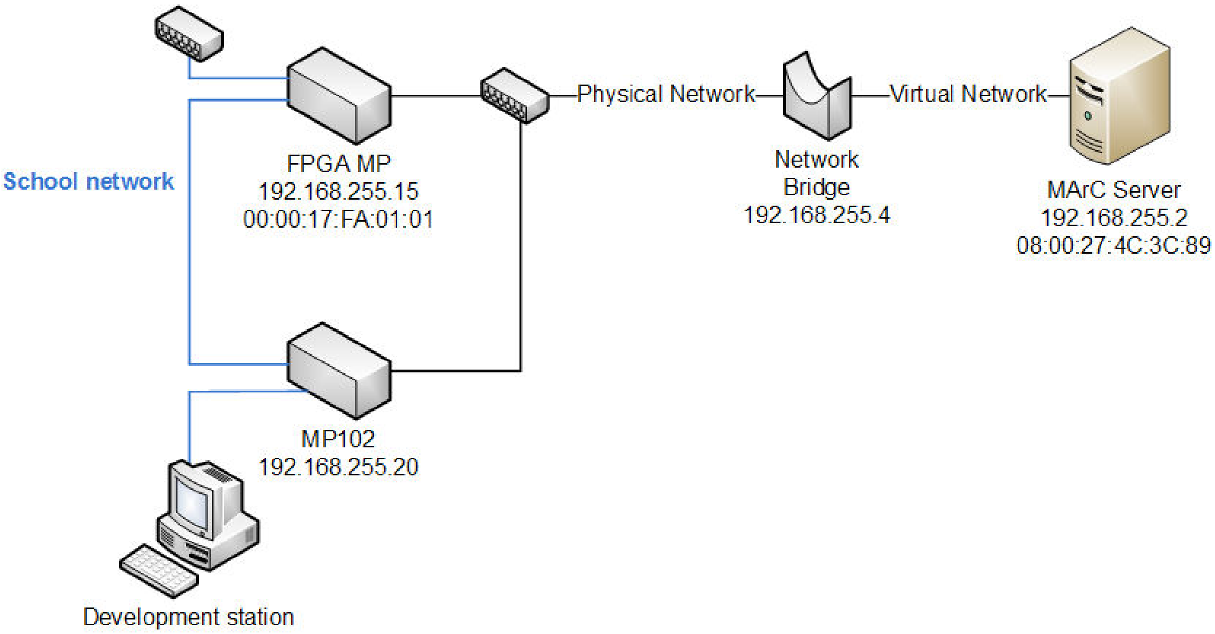


Figure 22 : Test network schema

The MArC server needs to run a web server (Apache + PHP) and a MySQL database server. Information about how to install a Linux Ubuntu server can be found at *https://help.ubuntu.com/.* The "MP102" device is a working MP and it will be used for MP comparison.

Remark: if the MArC server virtual machine is copied to another computer, its MAC address must be the same. All services are configured to work network interface named "eth2" and if the MAC address changes, the name of network interface will change and the server will not be working anymore.

6.2.1 IP auto configuration

The MP must get automatically an IP address. This can be controlled with Wireshark, in the DHCP server leases file and on the serial debug interface. The MP gets an IP address. IP auto configuration works.

6.2.2 MAINFO message

MAINFO message exchange can be verified in Wireshark and with the serial debug interface. The MP sends an empty MAINFO messages and the relay responds to it. MAINFO message exchange works.

6.2.3 Authentication with the MArC

MP_INIT and MP_AUTH message can be visualized in Wireshark. With the serial management interface it is possible to see if this step could be done. The MP must be visible in the MArC web management interface.

The MP sends a MP_INIT message. Data in the message corresponds to expected data. After that the MP is visible in the MArC but not authorized until user accepts the MP. The MArC sends an empty MP_AUTH message (it means the MP is not authorized now) and the MP sends a new MP_INIT message.

When the MP is authorized, it stops to send MP_INIT messages and serial debug applications indicates that DPMI heart state machine is its ready state that is named "Wait action". If a filter is set in the MArC it is sent to the MP.

Authentication with the MArC works.

6.2.4 Filter update

When the filter is modified in the MArC web management interface, a MP_FILTER_REQUEST message is sent to the MP. The MP sends it back to the MArC. Afterwards the MArC sends the updated filter. By looking in the measurement frames content I can see that captured data matches to the new rules.

Filter update works.

6.2.5 Filter deletion

When the filter is deleted in the MArC web management interface, the MArC sends a MP_FILTER_DELETE message. Then the MP stops to capture new data and sends content of the buffer. The serial debug interface indicates that the filter module is waiting for a new filter.

Filter deletion works.

6.2.6 Frames filtering

I started the MP and Wireshark with the same filtering rules (only ICMP protocol) and watched packet counters on the serial debug interface and in Wireshark. Data counters have not the same value (I couldn't start them at exactly the same time) but data counters values are consistent: the difference between both counters stays the same.

Frame filtering works.

6.2.7 Measurement frames

For testing measurement frames I created a filter that only captures ICMP packets and I sent "ping" requests to a computer in the network.

The measurement frame has correct send header and capture headers. Captured data length corresponds to the length defined on the MArC web management interface.

Measurement frames sending works.

6.2.8 Time synchronization

For testing time synchronization, the POSIX timestamp generated from GPS time data is sent over the debug serial line. The serial line debug application reads the timestamp and displays it in the console. The timestamp that the MP sends corresponds to the GPS time sent by the debug application and timestamp is automatically incremented every second, even if the time signal isn't present.

For testing transmission delay measurement a 20 meter Ethernet cable is used. The measured delay is sent over the serial line, in picoseconds. Transmission delay must be close to 100ns:

$$t_{delay} = \frac{cable\ length}{\frac{2}{3} * c_0} = \frac{20\ m}{\frac{2}{3} * 3 * 10^8 \frac{m}{s}} = 100\ ns$$

Unfortunately the transmission delay measurement cannot be tested because of a hardware failure: after testing the boards for short-circuits between concerned signals with the multimeter and verifying the CPLD design, I deduce there is probably a short-circuit between "delay_in" and "delay_out" signals in the CPLD.

Time synchronization works, except transmission delay measurement.

6.2.9 Capture issues

I found out, by using the serial line debug interface, that some frames are lost (between 1% and 2% data loss). It could be for two main reasons:

- Packets can be lost when filter buffer is sent to the consumer. In my test I set a filter that doesn't match to any frames on the network. No data is captured and therefore no measurement frames are sent, but the issue is still remaining.
- There could be transmission errors between the wiretap board CPLD and the FPGA. In Silvan Zahno's report (chapter 6.1.6) it is spoken about CRC errors due to cross-talking. It is the most probable reason why there is data loss, because I didn't really paid attention on this point when I made the converter board and adapted the CPLD design before starting the project.

For finding the reason of this issue and correcting it (if possible), following steps must be done:

- Create a test design for the FPGA that only receives data from the wiretap interfaces, counts incoming frames, checks CRC, counts CRC errors and sends this data over the serial line debug.
- Try to send data on others signals (data buses from CPLD to FPGA are on 32 bits but only 16 bits are used) and see if improvements can be made.

Unfortunately I don't have enough time left for doing it.

6.3 Comparison with another MP

I compare the FPGA MP with a software MP that uses an Endace DAG card, which is guaranteed to capture 100% of packets at any speed. For each test I set the same filter rules for both MP but different consumers. The count of captured packets is shown on the consumer debug output.

For the first test, I chose to capture 54 bytes of all UDP packets transmitted between the development station and the school network. At first glance I see in Wireshark that the software MP sends measurement frames more often but with smaller data amount. After one night of capture, the FPGA MP lost ~120 frames (including 20 matches) on ~305'700 frames.

For the second test, I chose to capture TCP traffic while I'm streaming a movie from Internet (it generates a lot of TCP traffic). After 5 minutes I notice that more than the half of the packets isn't captured. After analysis (see chapter 5.6.2 "Filter module block") I found out that with a high matching frame rate, the time when buffer is sent (and capture disabled) is really an issue. I propose a solution for avoiding that in the chapter 7.1.

I notice this issue very late in the project because during simulation I wasn't looking at that and with the first onboard test with ICMP the matching frames rate wasn't high enough to point out the problem.

7. Further work and improvements

7.1 Filter buffer issue

To avoid the capture issue when sending the filter buffer, I propose to modify the "Filter manager" and the "RAM" blocks in this way:

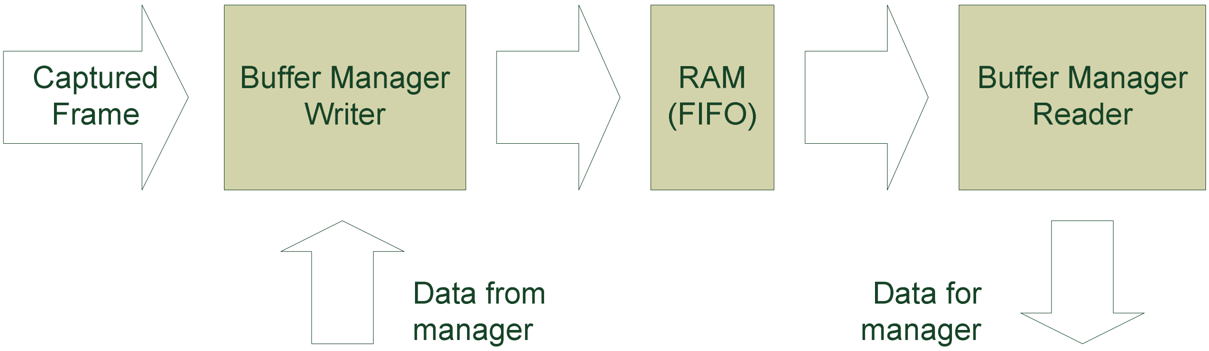


Figure 23 : Filter buffer issue solution

One state machine just writes captured data into the FIFO memory. The second one just reads data from the memory and sends it to the filter manager when there is enough data to send a measurement frame. Priority is given to the writer. The buffer contains the same data than now (capture header and capture data) but must be able to store more data, to avoid loss of captured frames if data can't be sent immediately.

7.2 Other improvements

It would be better to implement the DPMI interface part of the design in a microcontroller, using C or C++ programming languages. There are some text-based elements in DPMI protocol (for example the MArC IP address in MAINFO message) and it is tricky to parse that kind of data in a FPGA. As DPMI protocol is still in development, some message format might change and it would be simpler to reflect the changes (as the DPMI network in developed in C). But capturing and filtering frames must be done with a FPGA, mainly for performance reasons. The border between the microcontroller and the FPGA would be in the middle of the "Filter Management" block.

In the current VHDL design, some improvements can be made:

- Allow the design to parse MArC's IP address in MAINFO message and to send ARP requests. The MP would be able to communicate with the MArC when it isn't on the same machine than the relay.
- Transform capture interface to allow full frame capturing without data loss.
- Ethernet Dissolver can only handle 10/100 MBit/s network speeds. The PHY chip on Xilinx ML-405 board can handle 1GBit/s network speed. It would be a good thing to make the design work at this speed.
- If full frame capturing works, Ethernet Jumbo Frames must be implemented to be able to send all the captured frames. In a standard Ethernet frame, maximal payload size is 1500 bytes. In this case the entire frame will be bigger. If that frame is totally captured and sent to a consumer with capture header and payload its size will be bigger than 1500 bytes and won't fit in a standard Ethernet frame.

8. Conclusion

Finally the measurement point is working almost fine.

It is able to interface with a measurement zone using DPMI protocol in its version 0.7. It also auto configures itself at boot using DHCP. The most important messages of DPMI are supported, except the plain-text MArC IP address in the MAINFO message. The MP is recognized by the MArC and it sends status messages regularly.

One filter can be set, updated and removed dynamically. Captured data is successfully sent to an Ethernet consumer using Ethernet broadcast. The consumer displays correct debug information. For now the maximal capture length without data loss is 108 bytes. But this capture length is long enough if the network analysis only concerns packets header. For full frame capture the capture interface design needs to be modified.

There is a little conception issue with the filter buffer, which causes data loss. I hadn't the time to correct it but the modifications are described above in chapter 7.1.

Time can be synchronized using a GPS receiver that sends SiRF binary messages. The GPS receiver can be replaced with the debug application that sends computers time if GPS signal isn't accessible.

Yann Santschi

9. Bibliography

- A Distributed Measurement Infrastructure (PAM2005)
  written by Patrik Arlos, Markus Fielder, Arne Nilsson
- Xilinx ML405 Evaluation Platform - User Guide (UG210 v1.5.1)
- Xilinx ML405 Schematics
- Diploma work "Frame Capturing and Sending in FPGA" report
  written by Silvan Zahno
- Diploma work "Ethernet traffic measurement point" report
  written by Carlo Arnold
- IP Protocol documentation
  http://www.frameip.com (11/2011)
- SiRF Binary Protocol Reference Manual
  http://www.sirf.com (11/2011)
- GPS Week Rollover Number
  http://tycho.usno.navy.mil/gps_week.html (11/2011)
- Leap Seconds List
  http://www.hko.gov.hk/gts/time/Historicalleapseconds.htm (11/2011)
- Epoch & Unix Timestamp conversion tools
  http://www.epochconverter.com (11/2011)
- Endace TDS-2 Module TDS-6 Unit User Manual (EDM05.05-01r1)
- Trimble® Acutime™ Gold GPS Smart Antenna - User Guide

Appendixes

1. Wiretap Interface schematics
2. Wiretap Interface HDL Designer project (on CD)
3. Converter board schematics
4. GPS Board documentation
5. GPS Board schematics
6. DPMI version 0.7 source code (on CD)
7. Measurement Point HDL Designer project (on CD)
8. MPDebug software (on CD)