

Studiengang Systemtechnik

Vertiefungsrichtung Infotonics

Diplom 2007

Silvan Zahno

*Frame capturing and
sending in FPGA*

Dozent

François Corthay

Experte

Prof. Jorgen Nordberg
Patrik Arlos

Hes·so  **VALAIS
WALLIS**
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland



Systems Engineering: Infotronic

Diploma Work

- Frame Capturing and Sending in FPGA -

Author

Zahno Silvan

Under guidance of

Patrik Arlos, Corthay Francois

Version

v 1.0

Karlskrona, 25.03.08

Table of content

1	Acknowledgments	5
2	Preface	5
2.1	Introduction.....	5
2.2	Appendix.....	5
3	Overview	6
4	Hardware	7
4.1	Interface	7
4.2	Converter.....	9
4.2.1	Pin assignment on the Controller	10
4.3	Controller.....	11
4.3.1	Memory	11
4.3.2	Arm7	13
5	Passive Measurement Infrastructure	14
5.1	MARc.....	15
5.2	Consumer.....	15
5.3	MP.....	16
6	Software.....	17
6.1	CPLD on Interface-board	17
6.1.1	High-impedance Buffer.....	18
6.1.2	RS422 interface.....	18
6.1.3	Serial management interface	18
6.1.4	Phy clock	19
6.1.5	Reset gen	19
6.1.6	Mac data interface.....	19
6.1.6.1	Reduce frequency.....	19
6.1.6.2	Reduce cross talking	21
6.2	FPGA on Dev-board	22
6.2.1	RAM Structure	22
6.2.1.1	Measurement frame.....	22
6.2.1.2	Control and Status frames	24
6.2.1.3	Intermediate frame	25
6.2.1.4	Filter data base structure (FDB).....	26
6.2.1.5	Dataflow	26
6.2.2	Top-level	27
6.2.3	ResetGen	28
6.2.4	Capture interface.....	28
6.2.4.1	Synchronization	28
6.2.4.2	Receiver	29
6.2.4.3	Filter.....	30
6.2.5	Conversion filterrules	32
6.2.6	Arbiter.....	32
6.2.7	CI-demux.....	34
6.2.8	Output-choice.....	36
6.2.9	Sender.....	37
6.2.9.1	Transmitter	39
6.2.10	UART	41
6.2.11	TSC	43
7	Tests.....	45
8	Actual state	45
9	Further work	46
10	Conclusion and remarks	47

11	Glossary.....	48
12	References	49
13	Appendix.....	49

Table of figures

Fig. 3.1	Bloc diagram of the hardware.....	6
Fig. 4.1	Bloc diagram of the Interface.....	8
Fig. 4.2	Interfaceboard.....	8
Fig. 4.3	Blocdiagramm of Converter-board	9
Fig. 4.4	Converterboard	9
Fig. 4.5	Table pin assignment	10
Fig. 4.6	Table memory overview.....	11
Fig. 4.7	RAM block schema	12
Fig. 4.8	Table utilization and performance from Arm7 soft core	13
Fig. 4.9	Actel CoreMP7 Developement Kit	13
Fig. 5.1	Measurement Area	14
Fig. 5.2	MARc Interface	15
Fig. 5.3	MP schematic overview	16
Fig. 6.1	Top-level of the CPLD design.....	17
Fig. 6.2	Table MII Management Serial Protocol	18
Fig. 6.3	Serial management interface	18
Fig. 6.4	FSM of serial management interface	19
Fig. 6.5	Mac data interface	20
Fig. 6.6	Mac data interface	21
Fig. 6.7	Bloc diagram of the dataflow inside the FPGA.....	26
Fig. 6.8	Bloc diagram of the capture interface, the demux and the filters.....	27
Fig. 6.9	Table I/O signals resetgen	28
Fig. 6.10	Table Input signals synchronisation.....	28
Fig. 6.11	Table I/O signals receiver	29
Fig. 6.12	Table I/O signals receiver	29
Fig. 6.13	Table I/O signals receiver	30
Fig. 6.14	Block schema of filter.....	31
Fig. 6.15	Statemachine filter_heart.....	32
Fig. 6.16	Table I/O signals arbiter	33
Fig. 6.17	Statemachine arbiter	33
Fig. 6.18	Table I/O signals ci-demux	34
Fig. 6.19	Block schema of CI-demux	35
Fig. 6.20	Statemachine CI-demux_heart	36
Fig. 6.21	Table I/O signals output-choice	37
Fig. 6.22	Table I/O signals sender	37
Fig. 6.23	Block schema of sender	38
Fig. 6.24	Statemachine transmitter_controller	39
Fig. 6.25	Table I/O signals transmitter.....	40
Fig. 6.26	Block schema of transmitter	40
Fig. 6.27	Received filterbuffer	40
Fig. 6.28	Table I/O signals UART	41
Fig. 6.29	Block schema of UART	41
Fig. 6.30	Received filterbuffer	42
Fig. 6.31	Statemachine UART_heart.....	42
Fig. 6.32	Table UART configuration	43
Fig. 6.33	Time synchronisation network	43
Fig. 6.34	Table I/O signals TSC	44
Fig. 6.35	Block schema of TSC	44
Fig. 6.36	Statemachine delay_calc	44

1 Acknowledgments

First, I would like to thanks all of the people who helped me and gave me the opportunity to come here to BTH, Sweden to do my diploma work.

A special thanks goes to the supervisor and mentor of the project, Patrik Arlos. He assisted me in many ways and gave me useful tips during the whole project.

A big thanks belongs to Francois Corthay and Olivier Gubler, who sent me some required electronic parts and helped me a lot with the VHDL programming.

Everyone of my student colleagues and friends with whom I stayed in contact during my work. They have supported me from Ireland, China, Japan and also Switzerland.

And of course my family, who helped and motivated me all the time.

2 Preface

2.1 Introduction

This diploma work is the second part of a project, which began in Switzerland and was continued in Karlskrona, Sweden.

The subject of the Diploma Work is “Frame capturing and sending in FPGA”. In this part of the project, all components are brought together and developed into an MP. The hardware, or a part of it, I already made during my semester project in Switzerland. It is the sequel of the work in Switzerland. A first prototype of an MP was made by Olivier Gubler. He also did his work in Sweden with Patrik Arlos.

The goal is to explore and to develop the functionalities of a MP and then to implement and test these functions. For the work I have several hardware available:

- CoreMP7 Development Kit made by Actel
- The Interface I created during my semester project
- Converter which allows to attach the Interface with the Dev-Board

I will also give a small overview on the programs I used during this project.

- Libero v8.0
- CoreConsole v1.3
- SoftConsole (Eclipse) v1.1
- Xilinx Webpack v9.2
- HDLDesigner v2002.b
- Synplify

Before you read this report, it would be useful, if you have already read the semester project report. There you have a good overview about the hardware I used.

2.2 Appendix

Some articles and parts of the work are given in the appendixes. They can be found at the end of the report. A few appendixes are just given on the enclosed CD. In chapter 13 you can see a list of all appendixes.

3 Overview

The goal is to build a type of Hardware, called *Measurement Point (MP)*. Such a device does packet capturing. Then the captured data will be filtered, and a time stamp added onto each packet. The new packet will be stored into a buffer. After all, this packet will be sending in data packets in a measurement frame with defined sizes.

You also have the possibility to send some requests from a GUI called DPMI which was made in BTH by Patrik Arlos. This DPMI will analyze the captured packets and make some graphical approaches.

I will realize this MP with an FPGA, which can implement a microprocessor as an IP soft-core. Therefore, the used programming languages are C/C++ and VHDL.

The Ethernet input connections are:

- 10/100 Copper Ethernet RJ45
- Fiber Ethernet 100Base-FX (optional)

Because an MP is a passive device to the Ethernet input, it must be invisible to the *Measurement Area Network (MArN)*. To do a serious time stamp, an exact time synchronization is needed. Therefore an RS422 Interface is added, to synchronize with a master's clock.

The captured data will also be sent with a 10/100 Copper Ethernet RJ45. The programming interface of the board will be done via JTAG interface for the FPGA and also the Soft ARM7-Core.

It should also be possible to exchange the Interface with a different one, so that the input sources can be enlarged again; for example, to implement Giga Ethernet or other transmission mediums. For this project, it is sufficient to have a 10/100Base –TX or a FX support. However with a new Interface, it is possible to attach other transmission mediums also. It is not necessary to develop the Controller again.

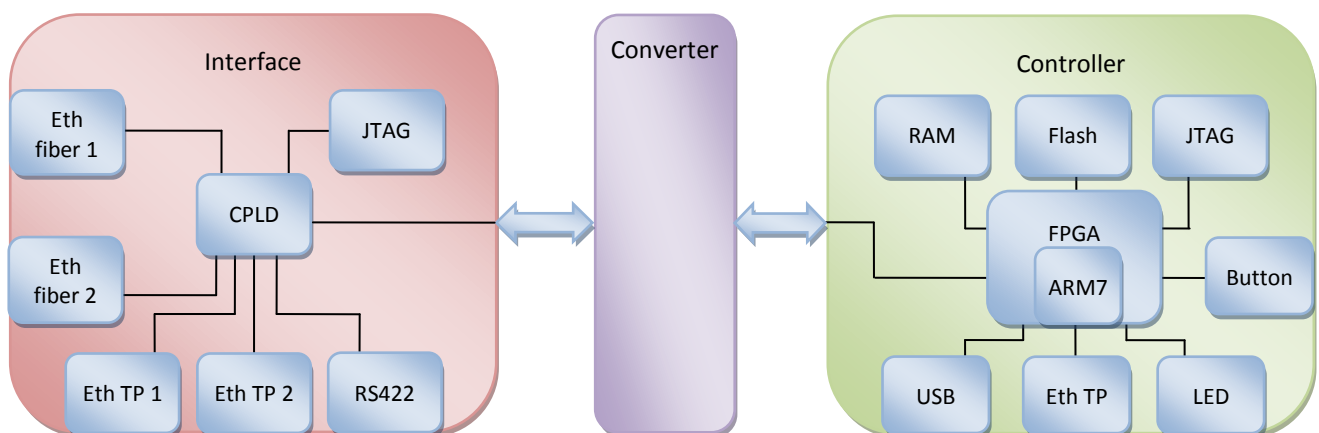


Fig. 3.1 Bloc diagram of the hardware

As discussed with Mr. Arlos, this MP should contain the following functionalities:

- Auto configure at boot
- Capture frames
- Filter frames
- Build and send measurement frames
- Send status messages
- Accept and act according to control messages
 - o Add filter
 - o Change filter
 - o Remove filter
 - o Flush buffers
- Use UDP/IP for messaging
- MP's should comply with v0.6/0.7 of the DPMI interface

4 Hardware

The Hardware is divided into three main parts:

- Interface to the MArN
- Converter between Interface and Controller
- Controller to generate the captured packets

I will give here just a little overview about the hardware I used during this project. More detailed information can be found on the semester project-report.

☞ Appendix 1 Report Semester project

4.1 Interface

This part of the hardware submits the captured packages of the MArN-Network to the Controller, without changes in the content. The network entrance can be carried out via a 100Base-TX twisted pair, a 10Base-T twisted pair or a 100Base-FX fiber (optional).

This Interface can be used for the following tasks:

- Active tap : Both input-lines go directly to *the physical Interface* (Phy) without any influence.
- Passive tap: Both Rx-Channels are connected through the high-impedance tap to the Phy. Since we are just listening, we do not need the Tx-Channels

In this project, only the passive tap is in use, because a normal MP is always invisible to the MArN-Network. The active tap is to implement functions in the future.

For example, it would be possible to attach either the *twisted pair* (TP) or the fiber. There are the following possibilities:

- TP \leftrightarrow TP: Both twisted pair inputs are used
- TP \leftrightarrow fiber: One twisted pair and one fiber are used (TP – FX conversion)
- fiber \leftrightarrow fiber: Both fiber inputs are used

In the passive case, it is only possible to attach the twisted pair. It is not possible to listen completely passively to the fiber-network. The fiber plug, already do a conversion between the light pulses and the electrical Signals.

- TP \leftrightarrow TP: Both twisted pair inputs are used

With Jumpers it is possible to switch the input-line between the RJ45 and the optical fiber input, as well as for the switch from the active to the passive tap. This is illustrated in the following picture.

The schematic is given in the appendix.

☞ Appendix 2 Schematic Interface

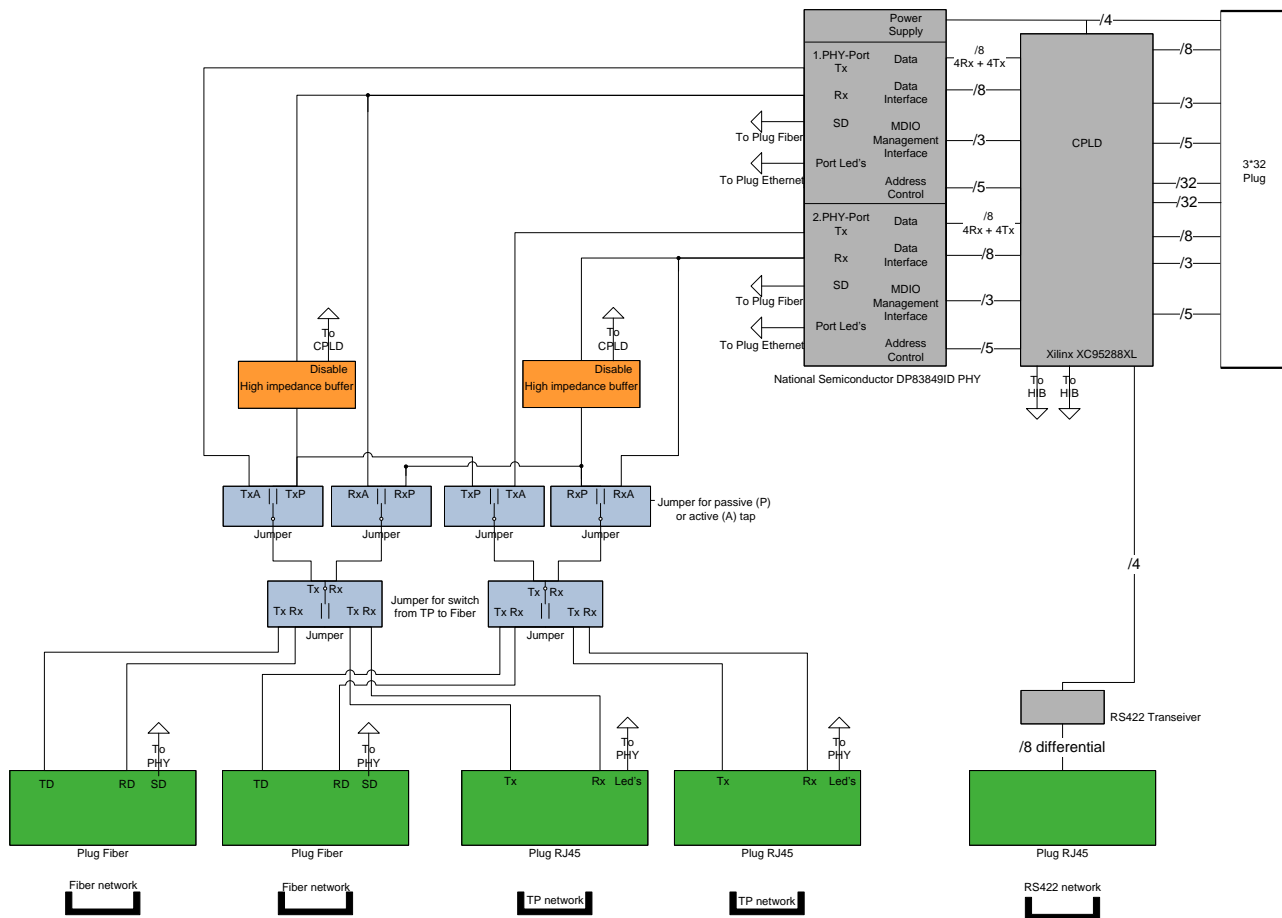


Fig. 4.1 Bloc diagram of the Interface

As you can see in the picture above, the board also provides a CPLD to implement some conversions between the interface board and the controller board. These conversions must be done to avoid transmission problems of some packets between these two boards. In the section software is a more detailed explanation.

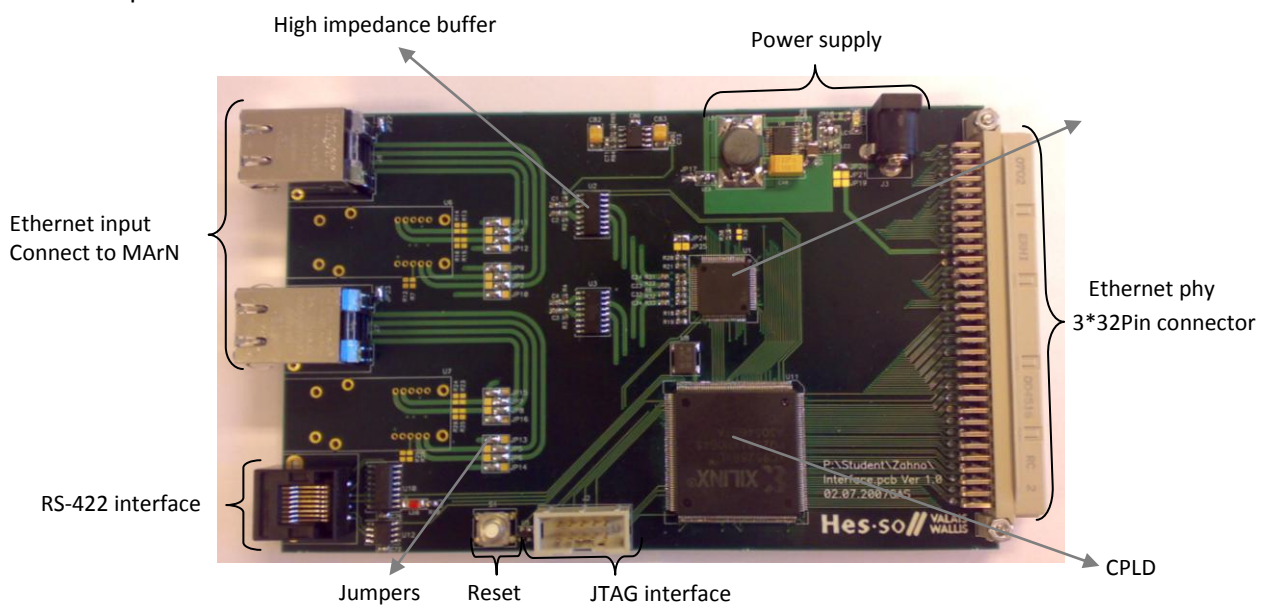


Fig. 4.2 Interfaceboard

4.2 Converter

We need a Converter board so that our Interface board is able to connect to the Controller board. A new Controller which can be done in future works, could be attached directly to the Interface without using the Converter board. However, in my case, I use a prefabricated Dev board as Controller and therefore, I have to use the Converter board.

This converter has on one side a 96-Pin connector for the connection with the Interface and on the other side, three 40Pin connectors to attach the Controller. The connectors on the Controller board also offer several powerlines, like 5V, 3.3V and GND. Therefore, we do not need the external 5V connector in the Interface. The power can be directly taken from the Dev board.

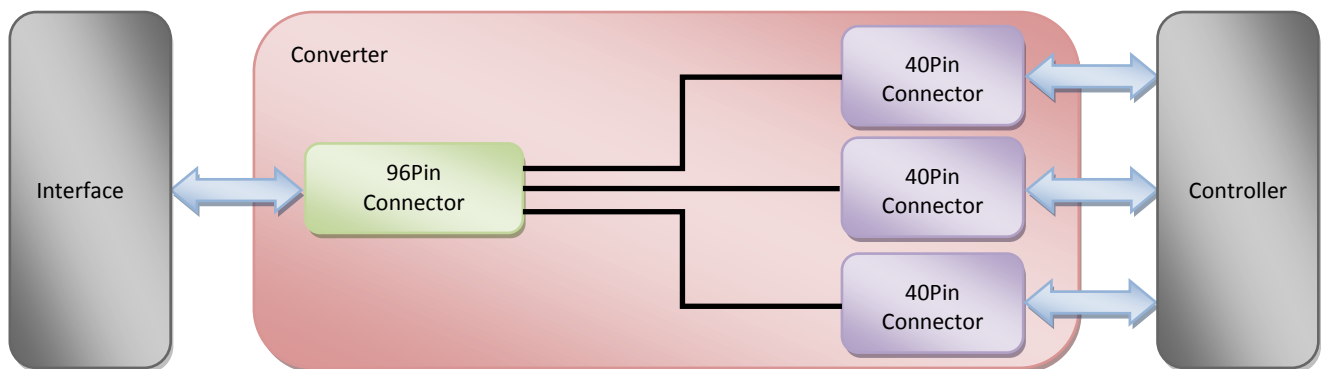


Fig. 4.3 Blocdiagramm of Converter-board

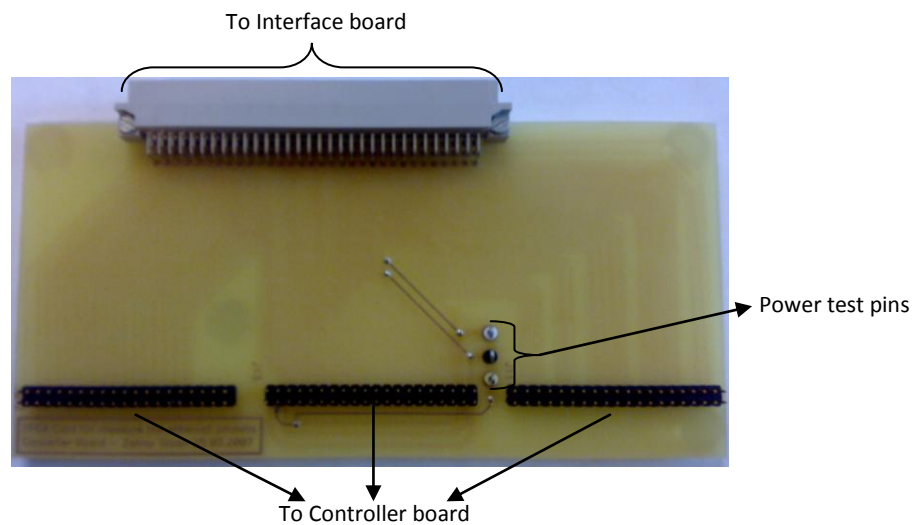


Fig. 4.4 Converterboard

4.2.1 Pin assignment on the Controller

With the help of the Converter board, the signals of the Interface, which belongs together, can be consolidated on one or several I/O Banks of the Controller.

The pin assignment of the different signals are shown in the following table.

Pin from Interface			
Signal description	Number of pins	Pin of the Header	I/O Bank of the Controller
Header J11 (control)			
RES 1-5	5	[4..8]	Bank 0 [3..5] & Bank 1 [0..1]
RS422 Interface	4	[11..14]	Bank 1 [4..5] & Bank 2 [0..1]
Control signals of port B	10	[17..24]	Bank 2 [4..6] & Bank 3 [0..4]
Control signals of port A	10	[27..34]	Bank 3 [7..11] & Bank 4 [0..2]
MDIO Interface	2	[37..38]	Bank 4 [5..6]
High impedance buffer disable	2	[39..40]	Bank 4 [7..8]
Header J12 (data port B)			
Data signals of port B	32 / 4	[1..32]	Bank 4 [9..23] & Bank 5 [0..5] & Bank 6 [0..10]
Header J13 (data port A & power signals)			
Data signals of port A	32 / 4	[5..36]	Bank 7 [0..31]
5V	1	[38]	-
3.3V	1	[37]	-
GND	1	[39..40]	-
Total Nbr of Pins	129		

Fig. 4.5 Table pin assignment

For further information, look at the schematic given in the appendixes

Appendix 3 Schematic Converter

4.3 Controller

The Controller part will receive all the Rx packets. These packets will be filtered and a time stamp will be added. After this, the packets will be reorganized and saved in RAM-Modules. To send the filtered packets to a computer, an Ethernet connection is used. On the computer, we have a DPMI-Interface, which is developed by Arlos Patrik, this interface will analyze the captured packets which were sent by the Controller.

The board consists of the following:

- Wall-mount power supply connector with switch and LED indicator
- Switches to select from 1.5 V, 2.5 V, and 3.3 V (I/O Bank) voltages on banks 3–4
- 10-pin, 0.1"-pitch programming connector compatible with Altera connections
- 50 MHz oscillator and 32kHz oscillator for real-time clock (RTC) calculations
- Eight LEDs driven by outputs from the device
- Jumpers allowing disconnection of all external circuitry from the FPGA
- One monostable pulse generator switch
- Eight switches providing input to the device
- Two RS-232 serial interfaces
- Two 10/100 Ethernet interfaces
- One Controller Area Network (CAN) 2.0B serial interface
- One USB 1.1 serial interface

🔗 Appendix 4 CoreMP7 development kit users guide

Remark: A reason why I chose this Dev-Board is, that the FPGA works with a 50MHz oscillator(clock). The packets leave the Phy with a speed of 25Mhz, so we have a certain surplus and can work without problems on the data in the FPGA. Otherwise, we could have an anti-aliasing effect.

4.3.1 Memory

One of the major tasks is to store all the data from the captured interface in a memory; to filter, modify and send them forward to the DPMI interface.

On the Dev-Board, SRAM and Flash memory are provided. We have two external memories and two internal memories. The external memories are slower than the internal, also the width and the depth are more or less fixed. The advantage of the external memories is that they are much bigger.

On the following table you can see an overview about all provided memory from the Dev-Board

Internal Memory					
Type	Description	Number	Total size	Address bits	Data bits
SRAM	Built-in SRAM	32 block à 4608bit	18kbytes	0 ... 16	0 ... 576
Flash	Built-in Flash	1	128bytes	6	7
External memory					
Type	Chip description	Number	Total size	Configuration	
SRAM	STMicroelectronics M29W800DT	2	2Mbytes	1M x 16 or 512k x 32	
Flash	GSI Technology GS8001BT	2	2Mbytes	1M x 16 or 512k x 32	

🔗 Fig. 4.6 Table memory overview

In this project, the internal SRAM is used as filter storage and buffers for the Capture blocks. The external SRAM has to be used as program storage for the Arm.

One of the advantages of the internal SRAM is, that it is a true Dual-Port Ram and each block can have his separate address- and data-lines, therefore, it is possible to work parallel on the different buffers.

External SRAM

Flash

The evaluation board includes two STMicroelectronics M29W800DT Flash memory chips, totaling 2 MB, which can be arranged in either a $1\text{M} \times 16$ or a $512\text{K} \times 32$ configuration. The Flash memory is intended for use as executable program storage for the embedded ARM microprocessor. However, it can also be used as nonvolatile memory for the storage of system constants and parameters.

SRAM

There are also two GSI Technology GS8001BT Synchronous SRAM provided on the board, totaling 2 MB, which can be arranged in either a $1\text{M} \times 16$ or a $512\text{K} \times 32$ configuration like the external flash memory. The SRAM memory is used for the embedded ARM microprocessor stacks (both hardware and software) and for dynamic system data.

Internal SRAM

In the FPGA we have chosen for this project, it is 144kbits(18kbyte) of RAM in 4,608bit (576bytes) blocks provided. Inside the FPGA, these blocks are arranged along the top and bottom of the device to allow a better access from the core and the I/O. All these blocks are true dual-port and can be configured in many different aspect ratios. The true dual-port functionality allows to read and write independently on two ports. Both ports can also have different clock speeds, because these RAMs are used as buffers. It is possible to read faster than write, which is a big advantage in this case. With the same RAMs, it is also possible to make a two-port RAM which has also two independent working ports. The difference to the dual-port is that on one port you can just write and on the other just read.

The maximum space taken in the RAM to store one whole Ethernet packet is 759 lines. The maximum length of an Ethernet frame is 1'518 bytes (12'144 bits) and the RAM width is 16 bits, in order to have an sufficient gap to write the capture header.

So it is necessary to have 10 bits to write the length of a frame and 10 bits it is also the minimum number of RAM address. With 10 bits, there are 1024 lines in RAM.

In this case, the data length is 16 bits and the address length is 10 bits. This means that each buffer has an available space of 2kbytes.

Because there are 18kbytes of RAM available and 4kbytes are already used from the ARM7 Core, is possible to have 7 buffers of 2kbytes space.

In the following picture, you can see both possible RAM blocks with the input and output signals.

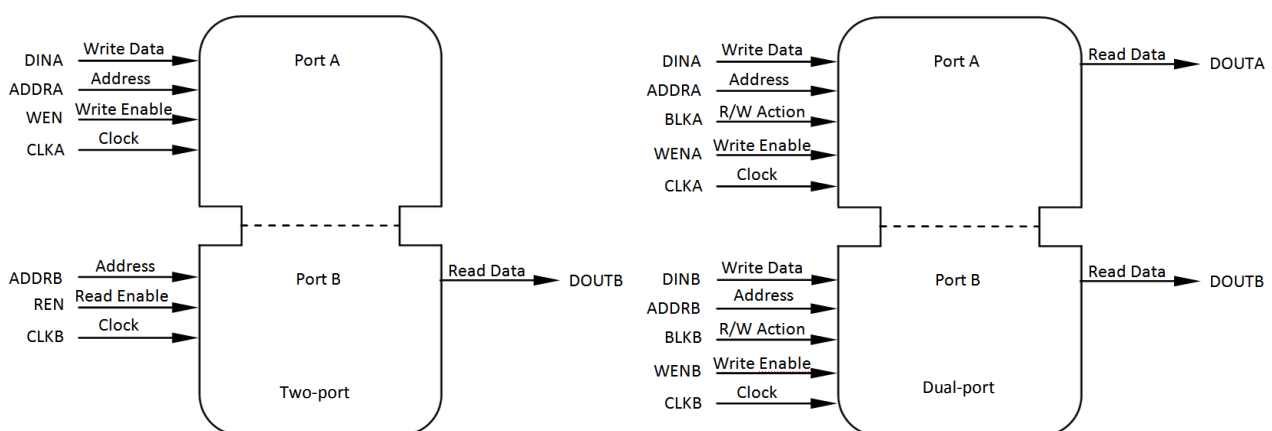


Fig. 4.7 RAM block schema

4.3.2 Arm7

The CoreMP7 soft IP core is an ARM7 family processor optimized for use in Actel ARM-ready FPGAs and is compatible with the ARM7TDMI-S.

CoreMP7 is supplied with an Advanced Microcontroller Bus Architecture (AMBA) Advanced High-Performance Bus (AHB) compliant wrapper for inclusion in an AMBA-based processor system.

The microprocessor has the following features:

- FPGA Optimized ARM7™ Family Processor
- 32/16-Bit RISC Architecture (ARMv4T)
- 16-Bit Thumb Instruction Set
- 32-Bit Unified Bus Interface
- 3-Stage Pipeline
- 32-Bit ALU
- 32-Bit Memory Addressing Range

Utilization and Performance:

Core variant	Performance (MHz)	Tiles	RAM Block	Utilization (%)
Core only	28.12	6083	4 (4kbytes)	24.8%
Core with debug	21.7	7931	4 (4kbytes)	32.3%

Fig. 4.8 Table utilization and performance from Arm7 soft core

Unfortunately, there is no Interface between the FPGA logic and the ARM processor (AMBA Bus). The time was too small to develop this interface, also a cheap Ethernet interface was required, but the Actel 10/100 Ethernet core costs 2000\$ per project, because of these reasons, the Arm Core was not used as intended.

The core was implemented and tested, but it is not used for the MP system. The tested subcores are listed below.

- Core MP7 & Core MP7 Bridge
- GPIO-Interface
- UART-Interface
- Interrupt core
- AHB2APB core
- AHB SRAM

Appendix 5 Datasheet CoreMP7

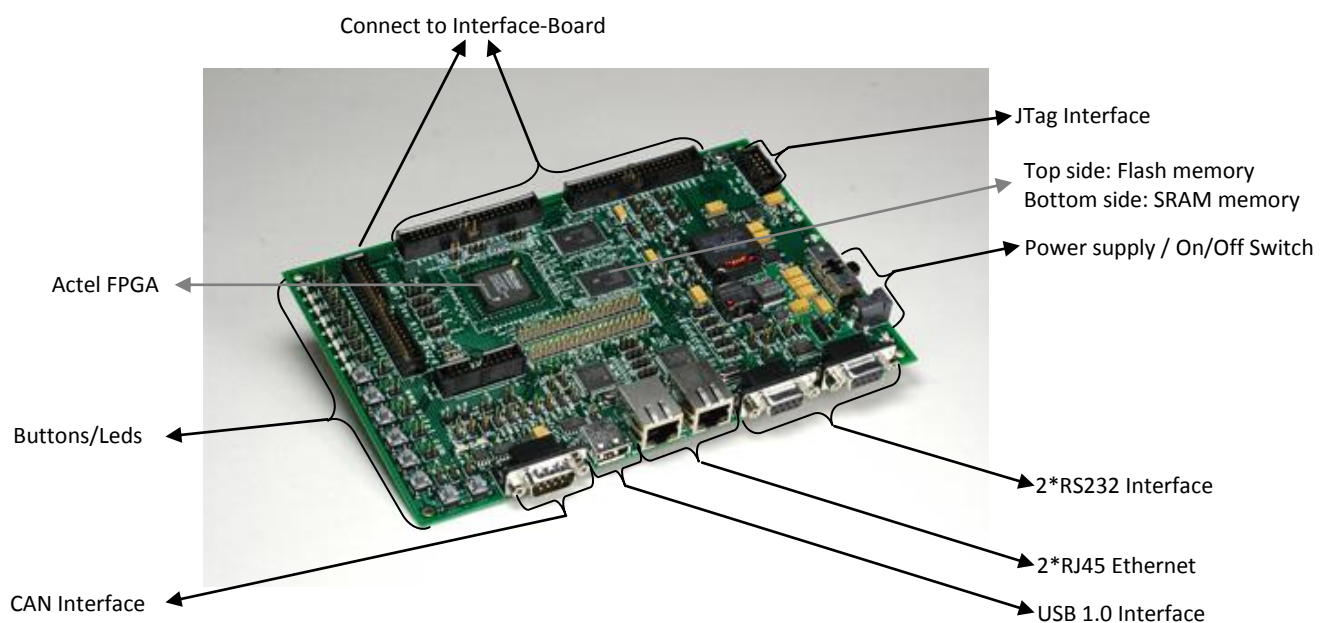


Fig. 4.9 Actel CoreMP7 Development Kit

5 Passive Measurement Infrastructure

In this section you will find an overview about the passive measurement infrastructure. For a good quality analysis of networks, it is necessary to have exact and accurate packet capturing. A passive measurement infrastructure allows to capture and filter data from a computer network. After that, it can be analyzed and displayed for the user.

A passive measurement infrastructure consists of 4 components which are connected together:

- *Measurement Area Controller (MArC)*, this device controls the MP
- *Measurement Point (MP)*, device which physically listens to channels, captures the wanted packets and distributed it.
- *Time Synchronization Device (TSD)*, which gives the exact time to the MP
- Consumer receive the sent data by the MP and analyze it and finally displays to the user

The passive measurement infrastructure is further explained in the documentation of Patrik Arlos.

Appendix 6 Passive measurement infrastructure

All these devices are connected together through several networks. A RS 482 connection is between the TSD and the MP.

The consumers, the MArC and the MP's are connected over the MArN network. In the following picture, you can see a schematic overview about a Measurement Area (MAr) with two MP's and Consumers.

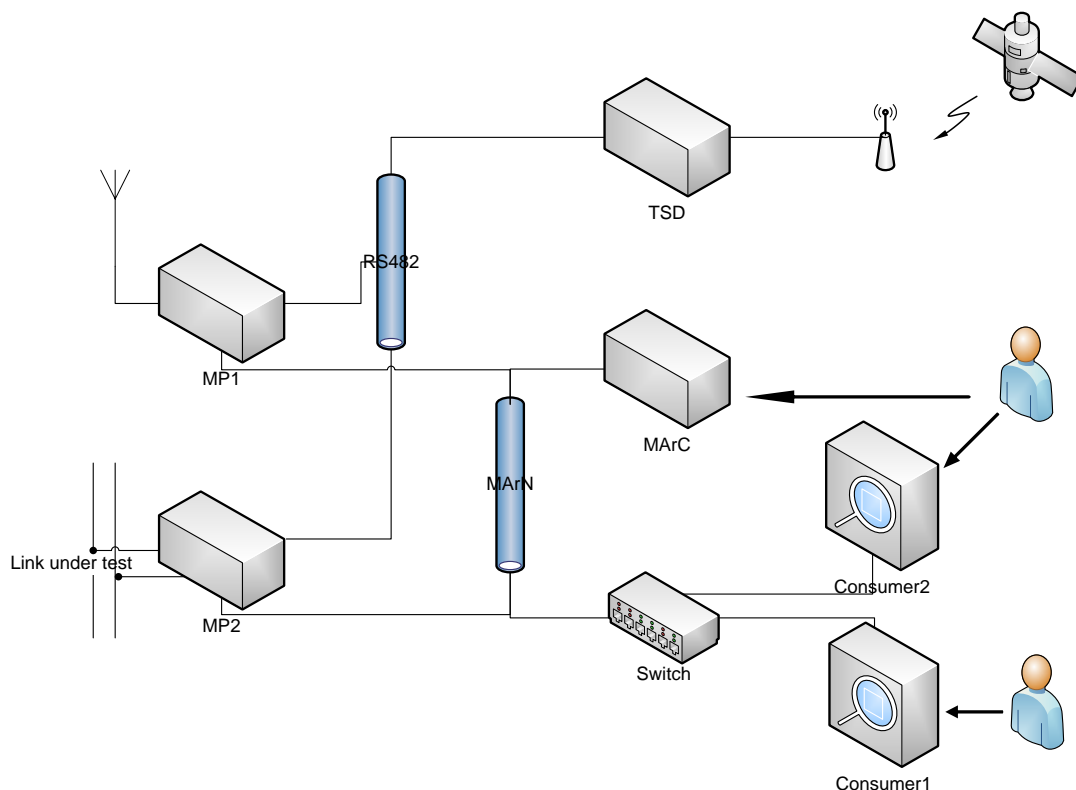


Fig. 5.1 Measurement Area

5.1 MARC

The MARC is the central component of the Infrastructure, and provides the user an Interface to the MP. In the MARC, the filter can be set and changed, as well as the status of the MP checked. An example of an MARC can viewed on the BTH website <http://inga.its.bth.se/projects/dpmi>, there is also a consumer integrated.

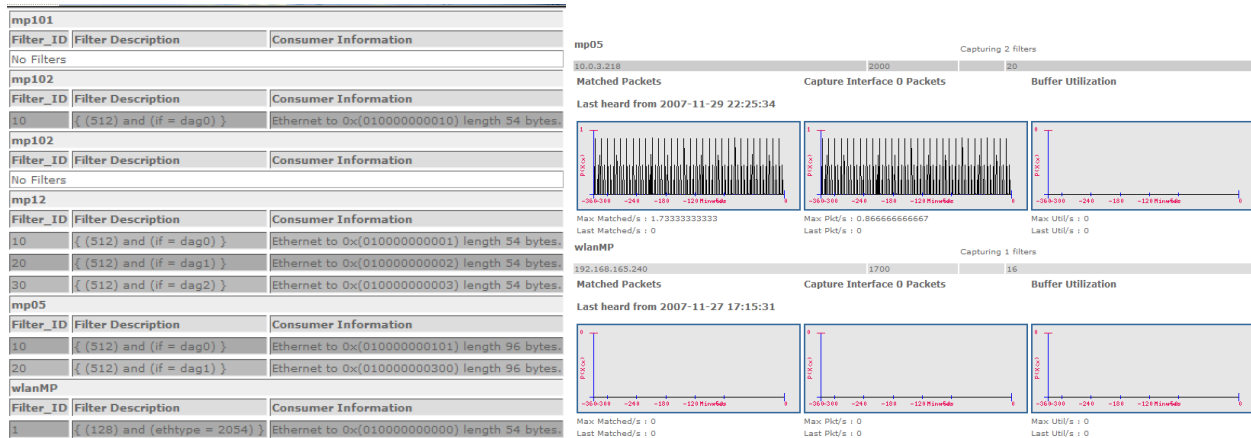


Fig. 5.2 MARC Interface

5.2 Consumer

In the DPMI Interface, there is also a consumer integrated which displays the results of the analysis for the user. These are the possible diagrams in the actual DPMI:

- Packet inter arrival time
- Link utilization
- Link utilization over 24h
- Hosts > 5 pkts/min
- Hosts < 5 pkts/min
- Protocol distribution VLAN
- Protocol distribution network
- Protocol distribution Transport
- Protocol distribution application
- Protocol distribution ICMP
- Beacon test
- Paket inter arrival (comparison)
- Kilroy classical
- Kilroy Nuevo
- Kilroy
- OWTT
- BPS comparison classical
- Timestamp comparison

5.3 MP

The MP is the device which has to be developed. In this project, it listens to one or several links under test. It can be a logical or a physical device, in this case the MP will be a physically device, made in custom hardware.

To the captured data there will be several additional pieces of information's added. These are listed in chapter 6.2 (see 6.2 RAM-structure).

In the following picture, you can see a schematic overview of an MP with two capture interfaces.

- Capture interface: Each MP has at least one capture interface. It listens to a channel, captures and filters the arriving packets. If a frame arrives which should be stored, it adds also a capture header to each frame.
- Time synchronization client: This block receives the actual time and forwards it to the CI. It is necessary for the timestamp in the capture header.
- Sender: The sender interacts with the MArN and if a full Ethernet packet is filled with captured frames, he adds an measurement header and send the packet through the MArN to the consumer.
- Controller: The communication between the CI and the sender is managed by the controller.

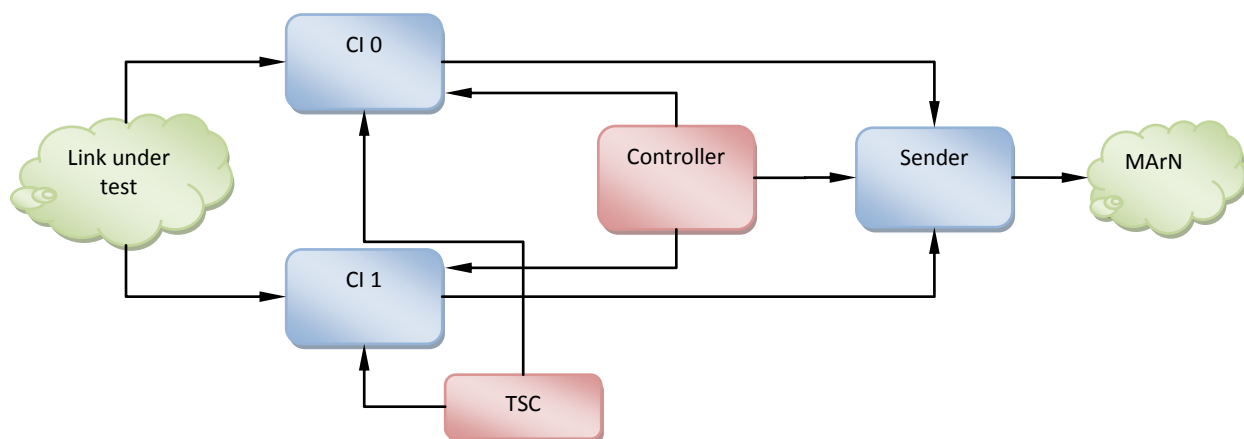


Fig. 5.3 MP schematic overview

6 Software

In this project, there are 2 tasks to implement and test:

- Implementation of the Interface functionalities in the CPLD. The hardware will be described in VHDL
- Implementation of the Controller functionalities in the FPGA. The hardware will be described in VHDL

The following features are integrated in the MP to be developed:

- 2 *Capture Interfaces (CI)*, which represent the connection to the MARN
- *Time synchronization client (TSC)*
- Filtering possibility for 2 filters
- Adaptable filter- and capture-length
- An Ethernet Interface to connect to the MARC
- Sending of status message every second

6.1 CPLD on Interface-board

On the Interface is a Xilinx XC95288XL CPLD mounted. There we have enough space to implement the Interface functions, which are:

- Activate the high impedance buffers
- Pass on the RS422 interface to the Controller
- Pass the control signals from Phy to the Controller
- Manage the bidirectional MII interface
- Generate the 25Mhz clock for the Phy

These functions were implemented with HDL-Designer. In the following section, I will give you a more detailed view. In the next picture, you can see the top-level of the CPLD design. I split it up into 7 sub bloc's to separate the different problems. On the picture you can see on the left side of the blocs, the signals connected to the components in the interface-board and on the right side, the signals going to the controller board.

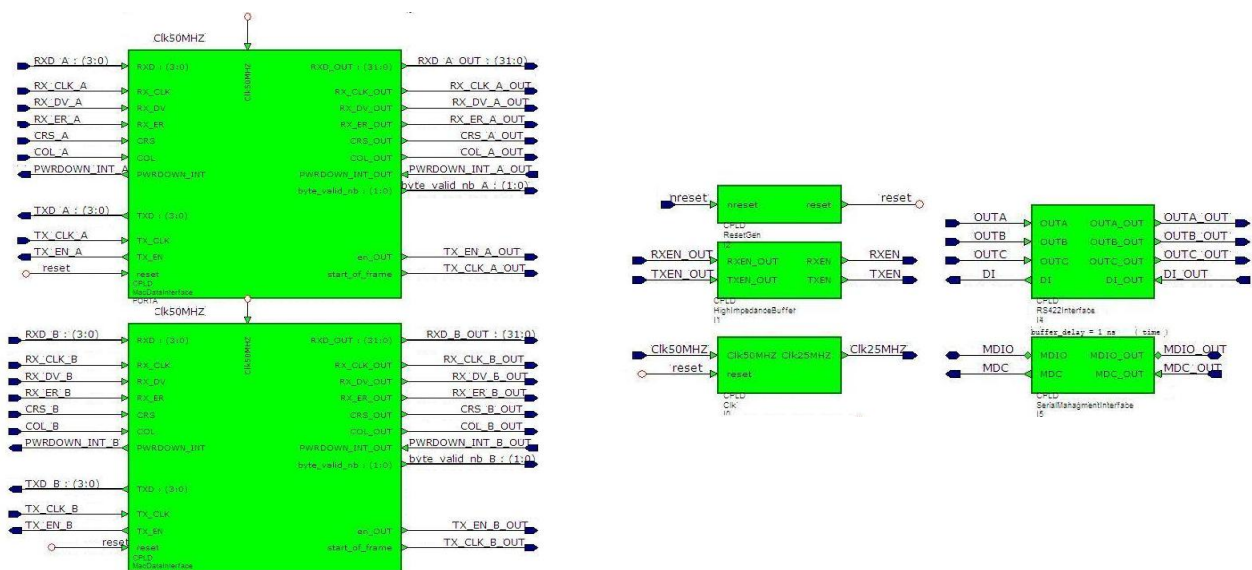


Fig. 6.1 Top-level of the CPLD design

6.1.1 High-impedance Buffer

In the high impedance buffer, it is possible to control the input from the Ethernet plugs to the Phy. You can either turn it on and let pass on all the packets to the Phy or bloc all signals to the Phy. In this case we do not need this functionality. Each buffer is enabled all of the time.

6.1.2 RS422 interface

Since the Dev-board does not have a RS422 interface for the time synchronization, it was added in the interface-board. The CPLD gives the signals directly to the FPGA without any changes made to the signals. The interface for the RS422 is done in the FPGA.

6.1.3 Serial management interface

In the Phy, some control and status register are available. To reach these registers a MII serial management interface is used.

This interface consists of two pins: Management Data Clock (MDC) and Management Data Input/Output (MDIO). MDC has a maximum clock rate of 25MHz and no minimum clock rate. In this case I used a clock rate of 25MHz. The MDIO line is bi-directional and can be shared with other devices. The MDIO frame format for read and write access is shown in the Figure below.

MII Management Serial Protocol	
Operation \ Protocol	<idle><start><op code><device addr><reg addr><turnaround><data><idle>
Read operation	<idle><01><10><AAAA><RRRR><Z0><xxxx xxxx xxxx xxxx><idle>
Write operation	<idle><01><01><AAAA><RRRR><10><xxxx xxxx xxxx xxxx><idle>

Fig. 6.2 Table MII Management Serial Protocol

The communication with the Phy will be done in the Dev-Board. In the CPLD the signals will be connected together. Since the MDIO signal is bi-directional, it is necessary to build a logic who can detect a read or a write access and enables the right buffer. This problem was solved by two tri-state buffers and a finite state machine. The following figures show the logic.

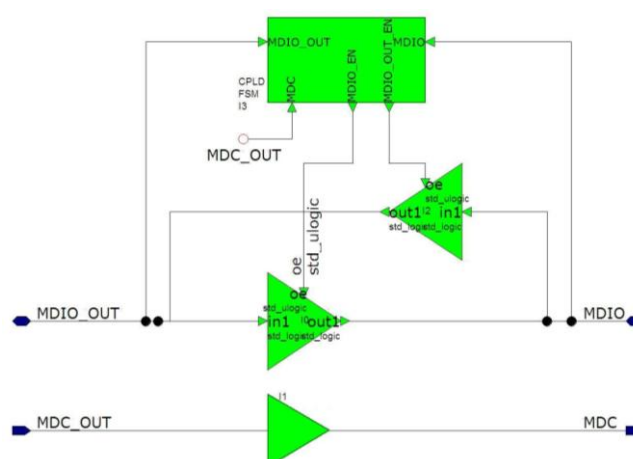


Fig. 6.3 Serial management interface

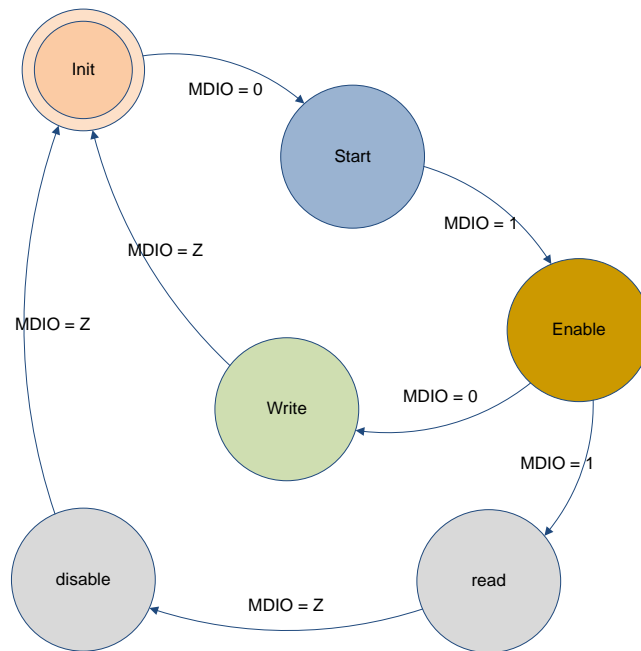


Fig. 6.4 FSM of serial management interface

6.1.4 Phy clock

To work in the CPLD with the phy-signals, it is indispensable to have a clock with a higher frequency than the clock in the phy. He needs a 25MHz clk. On the board is a 50MHz +/-10VPP mounted. The CPLD use this clock to create the 25MHz for the phy.

$$\text{Clock25MHz} = \text{Clock50MHz} / 2$$

6.1.5 Reset gen

This block is necessary to synchronize and invert the nreset signal with the used clock in the CPLD. As output we finally get a well synchronized reset signal.

6.1.6 Mac data interface

One of the problems of the ancient Diploma work was that the sending of the packets from the Interface to the Controller Board caused some errors in the CRC-Check. Because of this, some frames were lost.

To prevent the lost of a package, I tried 2 possibility the first was to reduce the frequency of the data between the Interface and the Controller and the second was to reduce the cross talking during transmitting over the ribbon cable.

6.1.6.1 Reduce frequency

The phy has a data-output of 4 signals, which means he gives the data nibble per nibble to the Controller. The maximal frequency of a 1000Base-TX Ethernet on the 4 Phy-Output lines is:

$$F_{max} = \frac{D}{nbrOfLines} \approx \frac{1Gb/s}{4lines} = 250MHz$$

We have chosen to use 32 lines/channel, this gives a maximal frequency of:

$$F_{max} = \frac{D}{nbrOfLines} \approx \frac{1Gb/s}{32lines} = 31.25MHz$$

For the assignment about a ribbon cable or hard plug, 31.25MHz does not represent a problem. Some of the CRC-checks were right after that modification but the problem was not completely solved, there were still some problems during transmitting which affected the CRC-check. In the next section you can see my implementation to change the number of data signals from 4 to 32.

Three sub-blocks were used to implement this function in the CPLD.

- Synchronization: All the used signals from the Phy will be synchronized by a D-FlipFlop with the rx_clk from the phy.
- Shift register: In this shift register we take the synchronized nibbles and put it in our 32-bits output signals. If all the signals are filled, we generate an additional word valid signal (en) to indicate that the 32-bits are ready to read. Since a frame doesn't have to be a multiple of 32bit, there is another signal (byte_valid_nb) to say, which bytes are valid from the sent 32bit data. Finally the signal rx_dv will also be generated to avoid synchronization problems between the two boards, so that we have the guarantee that the rising and falling edge is at exactly the right time.
- Frame detector: For the Shift register we have to know, when the data should be shifted. This is the case when the start of a frame delimiter is detected (after the preamble and SOF). This bloc generates this impulse if a SOF is detected in the rx-data signals. This signal will also be sent to the Controller, so that we do not have to test it again.

The following figure shows the three blocks with the in- and output signals.

All error control signals are just going through, so that the Controller can see directly, when an error occurred.

Because there are 2 *Capture Interfaces* (CI) on the Interface board, the mac data interface bloc is implemented twice in the CPLD-Design, once for the RX lines and once for the TX lines.

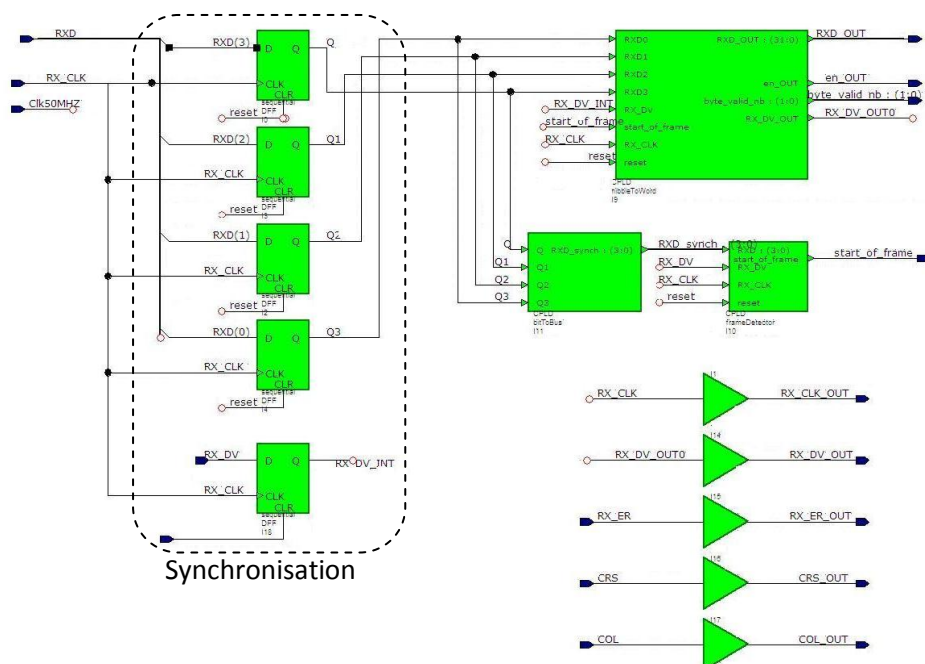


Fig. 6.5 Mac data interface

6.1.6.2 Reduce cross talking

In the finally version of the MP, this implementation was used. In a ribbon cable the signals were parallel and very close to each other, this affected the signal during transmitting. Therefore there were some glitches of the signals. To prevent this, I putted between each signal a ground line, that the signals are shielded from each other. Therefore, the number of signals could not be adapted, some signals are used as ground between the these lines.

To have a more stable and better synchronized signals, a D-FF and a buffer was inserted on each signal. This means on all control and data signals from the Phy. This solution was better and eliminated the wrong CRC-Checks. All packets arrive properly with a right CRC calculation. In the next figure you can see the implemented buffer and D-FF's.

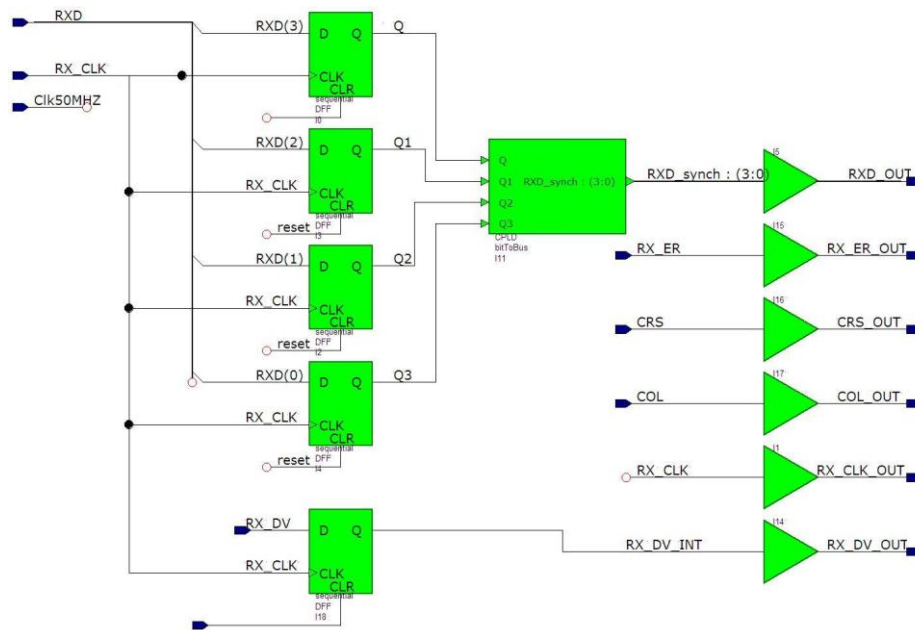
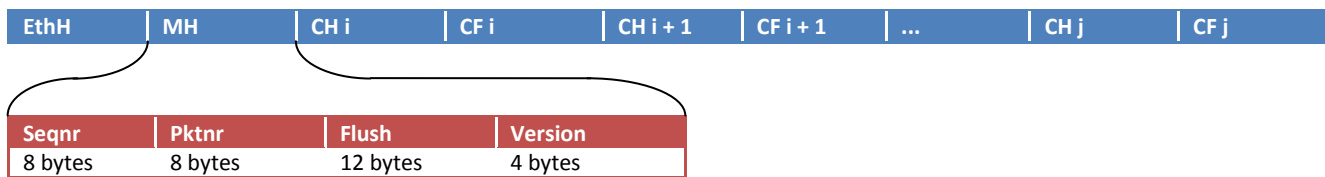
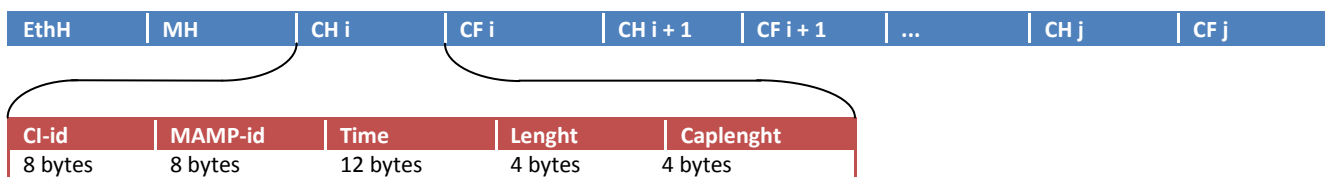


Fig. 6.6 Mac data interface

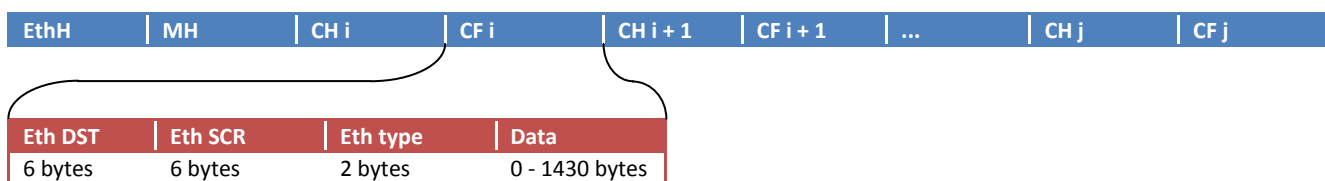
Measurement header

Seqnr Sequence number, which increments each time a full packet is sent.
 Pktnr Packet number, says how many captured packets are in the frame.
 Flush Indicates the last packet.
 Version Here it is possible to indicate the version number for the file format which is used.

Capture header

CI-id Captured interface id, it identifies the CI where the frame was caught.
 MAMP-id This field identifies the MP, were the frame was caught.
 Time The Time field is divided into a 4 bytes value which indicated the time in seconds and an 8 byte value, which indicates the time in picoseconds.
 Lenght Indicates the length of the whole frame.
 Caplenght Indicates which length of the frame is being captured.

It has to be mentioned, that the CI-id and the MAMP-id in this headers are human readable, therefore are the different fields quite large.

Capture frame

In this part is a captured packet stored. The preamble, the SOF and also the CRC will be removed. If we do not want a full frame catch, we store just the number of bytes indicated in the caplen of the filter.

6.2.1.2 Control and Status frames

All control and status frames were sending to the DPML interface over IP/UDP. The status messages will be sent from the MP to the DPML to indicate the current state of the MP.

The control messages will initialize the MP on the DPML. It provides several status information's. There exist other control messages, but because the receiving part of the MArN interface is not yet finished, this messages are not implemented. In the other control messages can be found several information's, like flush buffers, add- , remove- or change filters.

In the following pictures all field are explained in detail, the EthH rests the same as in the measurement frame.

EthH	IP header	UDP header	Data
14 bytes	20 bytes	8 bytes	0-1472 bytes

IP header

The IP header has the IPv4 format.

Ethernet II header											
EthH	IP header		UDP header		Data						
<div></div>											
Version	IHL	TOS	length	ID	Flags	Offset	TTL	Protokol	Checksum	IP source	IP dest
4 bit	4 bit	1 byte	2 bytes	2 bytes	3 bit	13 bit	1byte	1 byte	2 bytes	4 bytes	4 bytes

- Version: IP header version, always version 4.
- IHL: IP header length, length of the header.
- TOS: Type of service, not used here.
- Length: Total length of the entire packet.
- ID: Identification, for reassembling of IP-packets. It is not used here
- Flags: Flags for the reassembling, not used.
- TTL: Time to live, always 128.
- Protokol: Identify the next protocol, in this case udp, value 17.
- Checksum: Header checksum, it is the once complement of the one's complement sum of the IP header. It is defined in the RFC 791 standard.
- IP-source: IP source address, little endian.
- IP-dest: IP destination address, little endian.

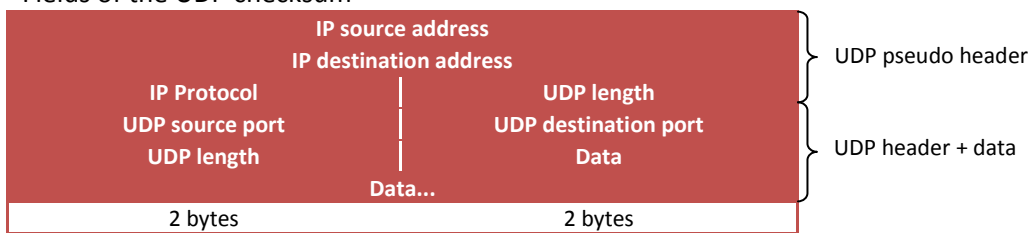
UDP header

Inside the IP data is a UDP header

EthH	IP header	UDP header	Data								
		<table> <tr> <th>Source port</th><th>Destination Port</th><th>length</th><th>Checksum</th></tr> <tr> <td>2 bytes</td><td>2 bytes</td><td>2 bytes</td><td>2 bytes</td></tr> </table>	Source port	Destination Port	length	Checksum	2 bytes	2 bytes	2 bytes	2 bytes	
Source port	Destination Port	length	Checksum								
2 bytes	2 bytes	2 bytes	2 bytes								

- Source port: Source port number, always 0.
- Destination port: Destination port number, always 1600.
- Length: Size of the UPD-header and data in octet.
- Checksum: The checksum is the ones complement of the ones complements sum of the UDP-pseudo header, the UDP header and the data. It is defined in RFC 768 standard.

Fields of the UDP checksum

**Status messages**

Like mentioned above, these messages are send by the MP to the DPML every second, it indicate the status of the MP. The information's are given in the data section of the frame described above.

Msg_type	MAMPid	No_filters	Matched	No_CI	CI_msg
4 bytes	16 bytes	4 bytes	4 bytes	4 bytes	30 bytes

- Msg type: Identification of the message type, for status messages always 2.
- MAMPid: MP id, always "FPGAMP0".
- No filter: Number of active filters in this case max 2.
- Matched: Number of packets who pass a filter.
- No CI: Number of active CI, max 2.
- CI msg: An optional message for the DPML. In this case, it is always "STATUSMESSAGE".

6.2.1.3 Intermediate frame

Inside the MAC-receiver and -transmitter are dual port buffers integrated, they store the intermediate frames. All the received packets from MAC-receiver are stored inside the rx-buffer, also the frames ready to sent will be stored with the same frame structure inside the tx-buffer.

It also adds a *frame header* (FH) if the packet was received or stored properly or an error header (ErrH) will be added, if there was an error during packet receiving. There are two different structures, one if the frame is valid and the other if an error is occurred. At the end of a packet structure or an error structure there is always an *empty header* (EH) which represent the frame header or error header of the next packet structure.

Valid frame structure

FH	Frame data	EH
----	------------	----

The valid frame structure contains a frame header, the frame data and an empty header which represent the space for the frame header of the next frame structure.

Error frame structure

ErrH	EH
------	----

In this version of the MP the empty header is not intended to use. But in further version it can be used to know, in how many packets an error was occurred during the sending, or receiving, and also which kind of error.

Frame header

Header valid	0	0	0	0	0	Frame length
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit9 - 0

Header valid The header valid bit indicate, when the valid frame is ready to read.

Frame length The length of the frame data is stored in these nine bits. It shows the offset to the next header.

Error header

Header valid	Error name	0	0	Error number
Bit 15	Bit 14 – 12	Bit 11	Bit 10	Bit9 - 0

Header valid The header valid bit indicate when the frame is being processed, but an error is occurred.

Error name It exist 4 different types of error, which can be
 “000” : nothing / “001” : CRC error / “010” : Rx error / “100” : buffer is full

Empty header

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Bit 15 – 0

The empty header is always at the end of a structre and represent the space for the next frame header. Therefore, it is filled with Zeros.

6.2.1.4 Filter data base structure (FDB)

In this buffer the filterrules are stored. The received filterrules will be converted into this structre, which contains 4 entries per Ram line. Each filter has a value and a mask with a size of 16bits so that it is possible to read the two buffers at the same time, the buffer must have a databus of 64bits. Each filter has a size of 84 bytes (42 bytes mask and 42 bytes value), therefore both filter occupied 21 RAM lines. The system is build for an easy enlargement. Because the minimal size of an internal RAM is 2kbyte and we just use 21 Ram lines in the actual filter, it remains 235 lines in the RAM. Theoretically, if there are two filters implemented, the first 512bytes of a PDU could be filtered.

Ram line structure

Value filter1	Mask filter1	Value filter2	Mask filter2
2 bytes	2 bytes	2 bytes	2 bytes

Value This 2 bytes value shows the desired packet content

Mask In the mask is defined, which bits should be checked

6.2.1.5 Dataflow

The data goes through the different blocks, each block adds an additional header to the measurement frame.

The Rx- and Tx-buffers contains the intermediate frames with the FH or ErrH. The Filter bloc adds the CH and the CI-demux bloc adds the MH to the frames and store it into the Filterbuffer.

The control and status messages are generated directly in the sender and will put into the Tx_buffer at the right time.

In the next picture you can see an overview, when which header will be added.

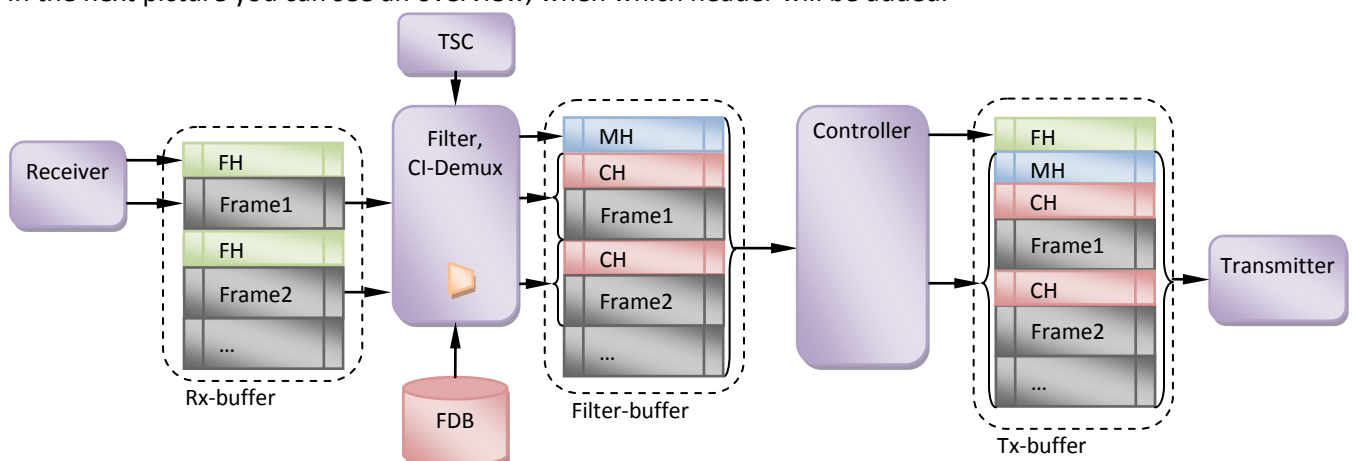


Fig. 6.7 Bloc diagram of the dataflow inside the FPGA

6.2.2 Top-level

The top-level of the FPGA design contains the entire logical blocks from the MP. Each bloc will be described in detail in the following section.

The next picture shows an overview about the entire top-level design in the FPGA. It has to be mentioned that there were 3 different designs implemented. One with the 32 bit input and the other with a 4 bit input in the CI. The explained design is with the 4bit version.

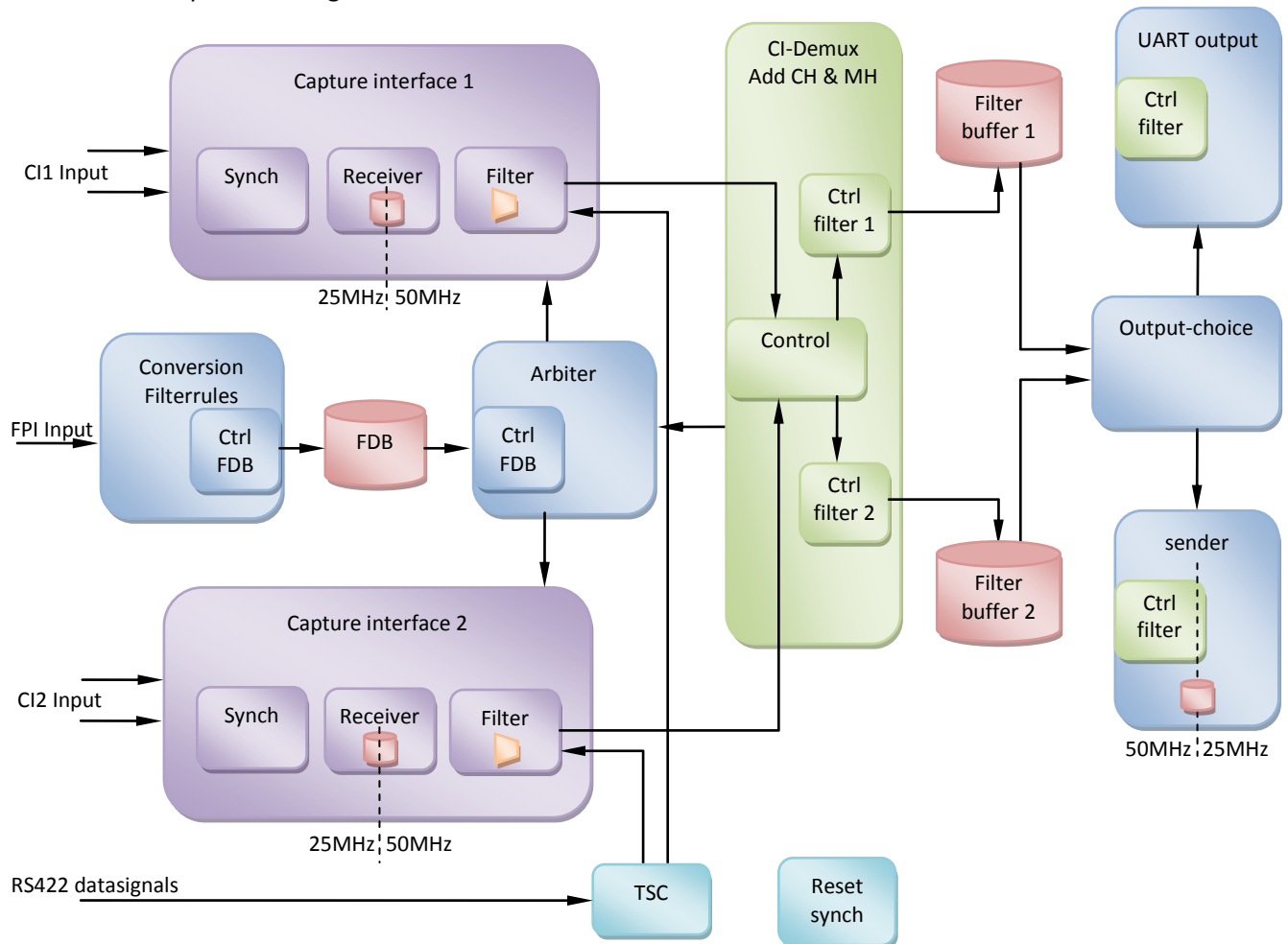


Fig. 6.8 Bloc diagram of the capture interface, the demux and the filters

The system is build for an easier enlargement of the number of CI and filters in another hardware. In the used FPGA, the number of CI is fix defined, because of the Interface board which provides two CI. Also the number of filter cannot be greater than two in the actual FPGA, because the numbers of available internal RAM's were limited. To increase the number of CI it is necessary to adapt the Arbitrer, the CI-demux and just copy the CI-bloc.

6.2.3 ResetGen

Because in the FPGA are four main clocks signals, the reset signal has to be synchronized with each clock, and connected to the belonging blocs. It contains some D-FF with different clock inputs.

In the following table you can see all the signals of this bloc.

Signal	Type (direction, type)	Description
Reset input		
Reset	In, std_logic	Reset signal, connected to the reset button
Reset for systemclock		
Clock	In, std_logic	50Mhz systemclock input
Clk_reset	Out, std_ulogic	Clock synchronized reset signal
Reset for CI1 interface		
Rx_clk_a	In, std_logic	25MHz CI1 clock signal
Rx_clk_a_reset	Out, std_ulogic	CI1 clock synchronized reset signal
Reset for CI2 interface		
Rx_clk_b	In, std_logic	25MHz CI2 clock signal
Rx_clk_b_reset	Out, std_ulogic	CI2 clock synchronized reset signal
Reset for transmitter		
Tx_clk	In, std_logic	25MHz transmitter clock signal
Tx_clk_reset	Out, std_ulogic	Transmitter clock synchronized reset signal

Fig. 6.9 Table I/O signals resetgen

6.2.4 Capture interface

The *capture interface* (CI) receives all packets from MAR_N, perform a CRC-check and store it into an intermediate buffer. After that, the packets will be filtered and the CH will be added with the exact timestamp, if the filterbuffer's are not used by the other CI, he send the packets to the CI-Demux. A capture interface can just listen to one channel.

This block contains a synchronization, a Mac-receiver and a filter. The receiver has a two-port buffer which works like an endless queue. When a packet arrives, the Mac receivers store it into the buffer and perform the CRC-check.

As soon as the whole packet is stored, a flag in the frame header will be set, that the filter know, the frame is ready and valid. If the filter buffers are also ready to store a new packet, the filter takes the packet, performs the filtering and sends it forward to the CI-Demux.

In the following sections, each block will be described carefully.

6.2.4.1 Synchronization

The synchronization-block is to avoid synchronization problems with the data, which are coming from Interface-board. Like in the CPLD, the signals will synchronized with the receiver clock coming from the Interface board. All of the following signals will be synchronized.

Signal	Type (direction, type)	Description
Data signals		
Rx_data	In, std_logic_vector(3 DOWNT0 0)	4bit data signals
Control signals		
Rx_dv	In, std_logic	Data valid signal
Rx_er	In, std_logic	Error signal

Fig. 6.10 Table Input signals synchronisation

6.2.4.2 Receiver

For the Eth receiver it was possible to take a core that was already made by the school in Switzerland. I made some changes to adapt it to my design and hardware, but the system remains the same. The receiver takes data from the Interface-board and saves them in a RAM, if the CRC-check was correct. A RAM structure is created to recognize what is saved. ☞ See chapter 6.2.1.3 Intermediate frames

To read the correct data from the RAM, it is necessary to set the signals: base address and address. It gives also a start of frame signal to the TSC that he knows, when exactly the frames arrived.

For additional information's about the receiver, have a look at appendix 7.

☞ Appendix 7 SimAP Design report

The changes include the following blocs:

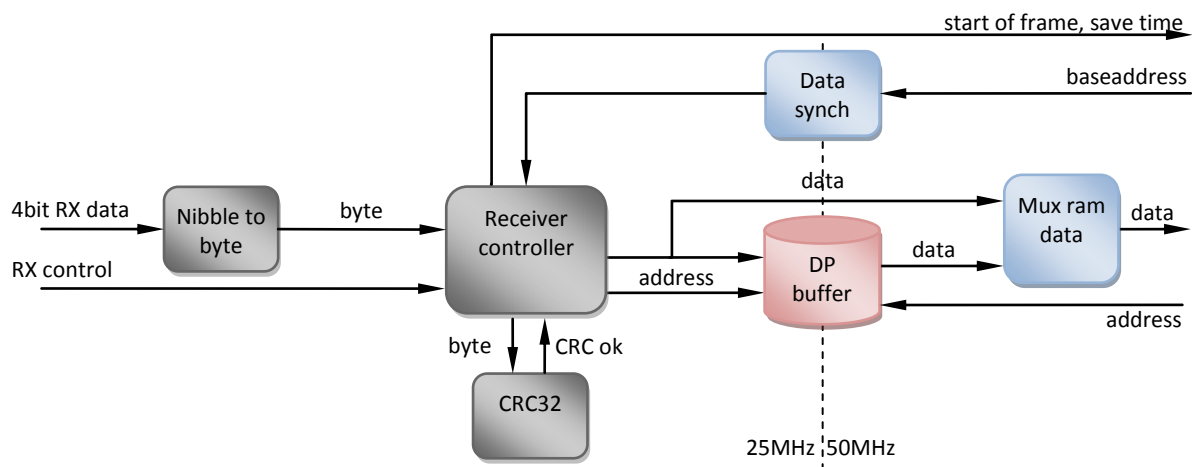
- The reset synchronization bloc was moved into the resetgen bloc, in the Toplevel, to combine all the reset synchronization's in one bloc
- Replaced the Xilinx buffer by an Actel dualport buffer
- Some small changes in the nibble_to_byte (4bit => 8bit) and byte_to_word (8bit => 16bit) bloc to avoid synchronization problems
- Add the save_time signal for an exact timestamping

In the next table shows you the I/O signal of this bloc

Signal	Type (direction, type)	Description
From synchronization		
Rx_data	In, std_ulogic_vector(3 DOWNT0 0)	4bit data signals
Rx_dv	In, std_ulogic	Data valid signal
Rx_er	In, std_ulogic	Error signal
To/from filter		
Address	In, std_ulogic_vector(9 DOWNT0 0)	Actual read address of the filter
Baseaddr	In, std_ulogic_vector(9 DOWNT0 0)	Boarder address for the write access of the buffer. The receiver can write till this address
Data_out	Out, std_ulogic_vector(15 DOWNT0 0)	Read data output of the buffer
Save_time	Out, std_ulogic	Signal to indicate that a frame is completely stored in buffer, and the time can be saved.
Start_of_frame	Out, std_ulogic	To know when the a frames is arrived, It will be set during the start of frame delimiter (5D)

☞ Fig. 6.11 Table I/O signals receiver

A schematic overview about the bloc inside the receiver is given below.



☞ Fig. 6.12 Table I/O signals receiver

6.2.4.3 Filter

All received packets from the receiver are stored inside the DP-buffer. As soon as a frame is finished, it will update the frameheader. When the filterbuffers are not used by the other CI, the filterbloc will read the ready frame and give it to the CI-Demux to store it into the filterbuffer. At the same time it will also perform the filtering as well as the CH will be added if the frame is properly stored.

At the beginning we do not know to which filterbuffer(s) the packet belongs, therefore it will be written first in both filterbuffers, if the filtering says that is just for one or none filterbuffer, we jump to the beginaddress in this buffer and the already written frame-part is erased.

In the following pictures you can see the table of I/O signals, the schematic and also the statemachine of the bloc filter_heart.

In the next table shows you the I/O signal of this bloc

Signal	Type (direction, type)	Description
To/from Receiver		
Baseaddr	Out, std_ulogic_vector(9 DOWNT0 0)	The receiver can write data till he reach this addr
Address	Out, std_ulogic_vector(9 DOWNT0 0)	Actual read addr of the filter
Data	In, std_ulogic_vector(15 DOWNT0 0)	Read data-bus from the buffer inside the receiver
Read_en	In, std_ulogic	Read enable from the buffer inside the receiver
Start_of_frame	In, std_ulogic	SOF signal to know exactly when a packet is arrived
Save_time	In, std_ulogic	Signal that say that a packet is safely stored in the buffer, the SOF time can be saved
To/from Arbiter		
Ci_id	In, std_ulogic	To identify which CI it is
Req_filter	Out, std_ulogic	Signal to request a filter and the filterbuffers
Av_filter	In, std_ulogic	To accept the filter request
End_filter	In, std_ulogic	Indicate the end of the filtering
Mask1	In, std_ulogic_vector(15 DOWNT0 0)	Mask of filter 0
Value1	In, std_ulogic_vector(15 DOWNT0 0)	Value of filter 0
Mask2	In, std_ulogic_vector(15 DOWNT0 0)	Mask of filter 1
Value2	In, std_ulogic_vector(15 DOWNT0 0)	Value of filter 1
Caplen_filter0	In, std_ulogic_vector(9 DOWNT0 0)	Capture length of filter 0
Caplen_filter1	In, std_ulogic_vector(9 DOWNT0 0)	Capture length of filter 1
To/from CI-Demux		
Data_ready	Out, std_ulogic	Say when data_out can be read
Data_out	Out, std_ulogic_vector(15 DOWNT0 0)	Data output
Filtermatch	Out, std_ulogic_vector(1 DOWNT0 0)	Signal to indicate to which filter the packet match filtermatch(0) => filter 0 filtermatch(1) => filter 1
WriteCH	Out, std_ulogic	Indicate the writing of the CH
End_writeCH	Out, std_ulogic	Indicate the end of writing the CH
Not_free_Ram	In, std_ulogic	Indicate that the filterbuffer is full
Packet_error	Out, std_ulogic	Indicate an error. This signal goes also to the Arbiter. That he knows that the buffers are not used anymore.
To/from TSC		
Timestamp	In, std_ulogic_vector(95 DOWNT0 0)	Timestamp

Fig. 6.13 Table I/O signals receiver

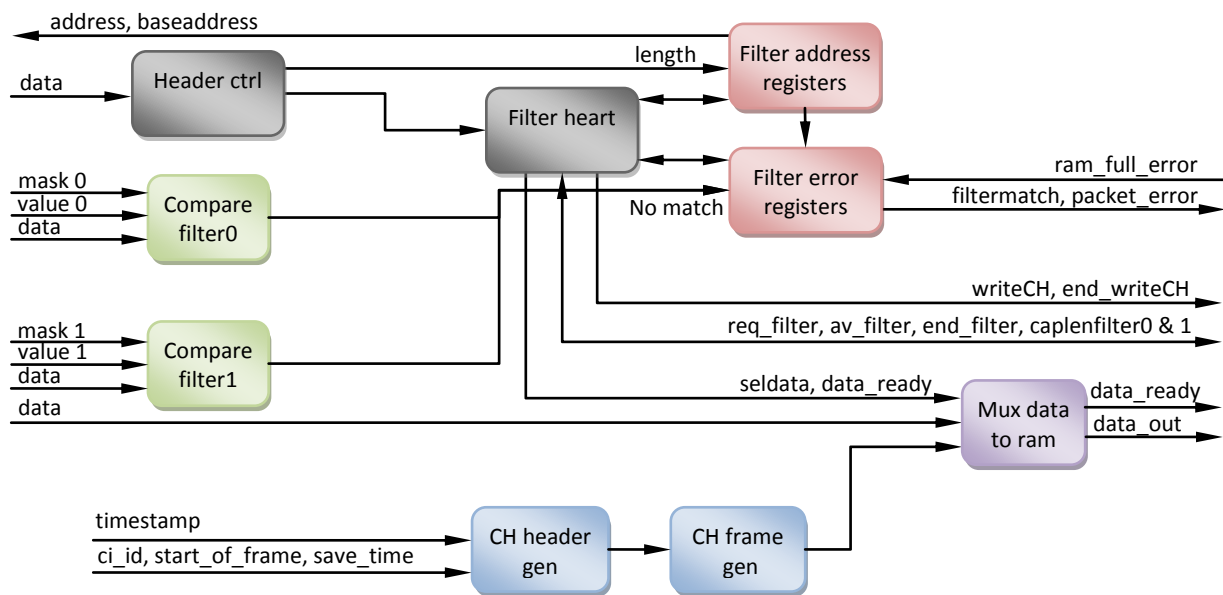
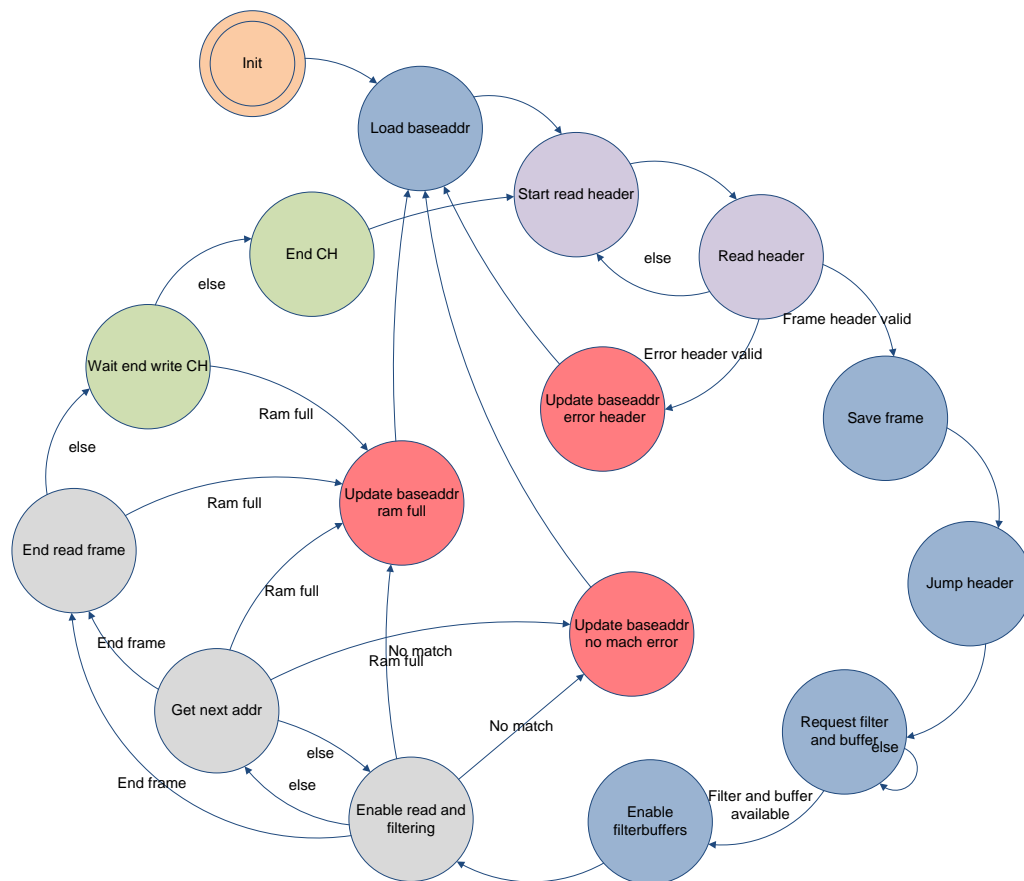


Fig. 6.14 Block schema of filter

- **Header ctrl:** Reads the data and control, if the frame- or error-header is valid. If it is a frameheader, it also reads the length of the packet and gives it to the filter-address reg and heart.
- **Compare filter:** This block performs the filtering, he takes the data, put the mask on it and compares with the value. There are two blocks, one for each filter.
- **CH header gen:** Generates the values for the CH frame gen bloc. For the timestamping is an array of 31 `std_u logic_vector(64 DOWNT0 0)` implemented, who works like an endless queue. If the SOF signal arrive he store the actual time in the temporary signal, after that if the `save_time` signal arrive I will be finally stored into the array to use in the appropriate CH. The array has 31 entrances because there can be max. 31 frames in the receiver buffer.
- **CH frame gen:** If the heart indicate to write the CH, he write the whole CH with the values from the CH header gen.
- **Filter addr reg:** Because we do not have registers inside the statemachine, for all the addresses the filter addr reg was created. He manages the actual addr and the baseaddr for the read access on the buffer inside the receiver. The heart can perform some actions on the addresses, like increment, decrement, jumpCH, save_begin, save_end, load_begin, load_end.
- **Filter error reg:** This registers stores all errors, if the filterbuffer is full, the `ram_full_error` signal will indicate it. If we have no filtermatch, the `no_match` signal will be set. In each case, if an error occurred, the `packet_error` signal will indicate it to the other blocs.
- **Mux data to ram:** The filter-heart can select, if the data or the CH_data should go to the ram.
- **Filter heart:** In the filter heart is a statemachine, which controls all the other blocs inside the filter. He waits for a valid frame, takes it, perform the filtering and send it to the CI-demux, at the end a CH will be added. Before he writes something into the filterbuffer(s) he asks for permission to the Arbiter. In the following picture you can see the statemachine.



☞ Fig. 6.15 Statemachine filter_heart

6.2.5 Conversion filterrules

This bloc converts the given filterrules into a mask and a value, which will be stored in the FDB. To see the ram structure of this buffer, go to chapter 6.2.1.4 Filterdatabase.

Because the first 42 bytes of a packet will be filtered, each filter contains a mask and a value of 42 bytes. The mask says which bits should be compared with the value. At the moment all the filters are static and cannot be changed during the runtime of the MP. To change a filter, the FPGA has to be reprogrammed.

6.2.6 Arbiter

In the FPGA are 2 CI implemented which are using the same filters and filterbuffer's. Therefore it is necessary to have a system that choose which CI can be active and have the permission, to write into the filterbuffer's. In this case in the arbiter-bloc is a round-robin algorithm implemented. When just one CI request for a filter, the arbiter gives him the permission, but if both CI wants to use the filterbuffer's he choose the active CI with the round robin algorithm. We have an internal register, where is stored which CI got the permission before.

If a CI is active the arbiter reads the filterrules from the FDB and gives the masks and the values of both filters to that CI. To see the FDB buffer structure look at chapter 6.2.1.4 Filterdatabase.

On the figures below you can see the I/O signal table and the statemachine of the arbiter.

Signal	Type (direction, type)	Description
To/from Filterrule-buffer		
RD	In, std_ulogic_vector(15 DOWNT0 0)	Databus of the filter database
REN	Out, std_ulogic	Read enable signals of the filter database
RADDR	Out, std_ulogic_vector(9 DOWNT0 0)	Read addrbus of the filter database
From Conversion_filter		
Caplen_filter1	In, std_ulogic_vector(9 DOWNT0 0)	Capture length input of filter1
Caplen_filter2	In, std_ulogic_vector(9 DOWNT0 0)	Capture length input of filter2
To/from Capture interface 0 & 1		
Req_filter_ci0	In, std_ulogic	With this signal the CI0 can request the filterbuffers
Av_filter_ci0	Out, std_ulogic	To give the write permission for the filterbuffers to CI0
End_filter_ci0	Out, std_ulogic	To indicate the end of the filtering for CI0
Req_filter_ci1	In, std_ulogic	With this signal the CI1 can request the filterbuffers
Av_filter_ci1	Out, std_ulogic	To give the write permission for the filterbuffers to CI1
End_filter_ci1	Out, std_ulogic	To indicate the end of the filtering for CI1
Mask1	Out, std_ulogic_vector(15 DOWNT0 0)	Mask of filter1
Mask2	Out, std_ulogic_vector(15 DOWNT0 0)	Mask of filter2
Value1	Out, std_ulogic_vector(15 DOWNT0 0)	Value of filter1
Value2	Out, std_ulogic_vector(15 DOWNT0 0)	Value of filter2
Caplength_filter1	Out, std_ulogic_vector(9 DOWNT0 0)	Capture length of filter1
Caplength_filter2	Out, std_ulogic_vector(9 DOWNT0 0)	Capture length of filter2
Packet_error_ci0	In, std_ulogic	Indicate an error during filtering in CI0
Packet_error_ci1	In, std_ulogic	Indicate an error during filtering in CI1
To/from CI-Demux		
Buffer_ready	In, std_ulogic	This signal indicate that the filterbuffers are not used by any CI.
Seldata	Out, std_ulogic_vector(1 DOWNT0 0)	These 2 bit indicate the active CI

Fig. 6.16 Table I/O signals arbiter

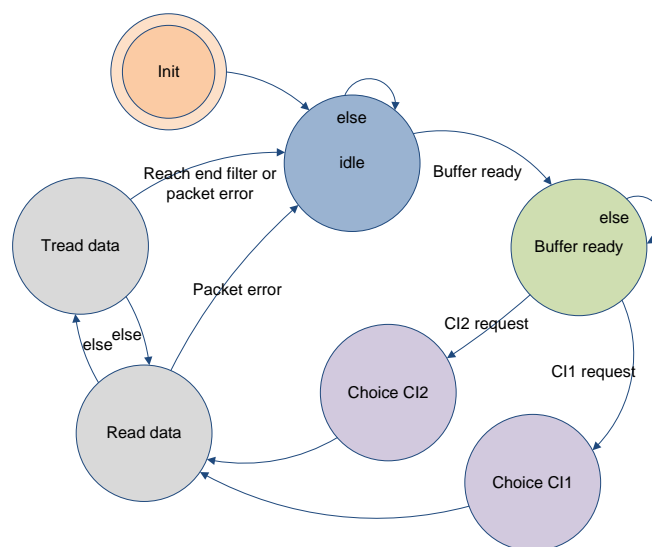


Fig. 6.17 Statemachine arbiter

6.2.7 CI-demux

If a CI is active, the data and the CH will be sent to the CI-demux. Which forward it to the appropriate filterbuffer(s). After a successful copy in the filterbuffer, he checks, if there is still enough space for another frame of a caplength size with the belonging CH. If one or both filterbuffer's are full, he adds the MH at the beginning and sends the flush_buffer signal to the sender. He will take the data and sends a flush_ok back. After that the buffer is ready for new packets from the CI. In the next pictures, you can see the I/O table and an overview about the sub bloc's in the CI demux, they will be explained in detail below.

Because two filterbuffer's are implemented in the system, the CI-demux has for each filterbuffer a separate address register and an error register to store the different addresses and errors

Signal	Type (direction, type)	Description
To/from Arbiter		
Buffer_ready	Out, std_ulogic	This signal will be set if the filterbuffer are not used
Seldata_arbiter	In, std_ulogic_vector(1 DOWNT0 0)	Indicate which CI is active bit0 for CI0 and bit 1 for CI1
Caplen_filter0	In, std_ulogic_vector(9 DOWNT0 0)	Capture length of filter0
Caplen_filter1	In, std_ulogic_vector(9 DOWNT0 0)	Capture length of filter1
From Capture interface 0		
Logger_data_ci0	In, std_ulogic_vector(15 DOWNT0 0)	Data from CI0
Data_ready_ci0	In, std_ulogic	Say when the data is ready to read
writeCH_ci0	In, std_ulogic	To write the CH
endCH_ci0	In, std_ulogic	End of writing CH
Filtermatch_ci0	In, std_ulogic_vector(1 DOWNT0 0)	That that CI-demux knows in which buffer the data has to be forwarded
Packet_error_ci0	In, std_ulogic	If an error occurred this signal will be set
From Capture interface 1		
Logger_data_ci1	In, std_ulogic_vector(15 DOWNT0 0)	Data from CI1
Data_ready_ci1	In, std_ulogic	Say when the data is ready to read
writeCH_ci1	In, std_ulogic	To write the CH
endCH_ci1	In, std_ulogic	End of writing CH
Filtermatch_ci1	In, std_ulogic_vector(1 DOWNT0 0)	That that CI-demux knows in which buffer the data has to be forwarded
Packet_error_ci1	In, std_ulogic	If an error occurred this signal will be set
To Filterbuffer 0		
WEN_filter0	Out, std_ulogic	Write enable of filterbuffer0
WADDR_filter0	Out, std_ulogic_vector(9 DOWNT0 0)	Actual write address of filterbuffer0
WD_filter0	Out, std_ulogic_vector(15 DOWNT0 0)	Data lines of filterbuffer0
To Filterbuffer 1		
WEN_filter1	Out, std_ulogic	Write enable of filterbuffer1
WADDR_filter1	Out, std_ulogic_vector(9 DOWNT0 0)	Actual write address of filterbuffer1
WD_filter1	Out, std_ulogic_vector(15 DOWNT0 0)	Data lines of filterbuffer1
To/from Output Choice (forward to sender)		
Flush_buffer	Out, std_ulogic_vector(1 DOWNT0 0)	Indicate which buffers has to be flushed by the sender
Flush_ok	In, std_ulogic_vector(1 DOWNT0 0)	Confirmation of buffer-flush
Not_free_ram	Out, std_ulogic	Error signal if buffer in sender is full
End_addr_filter0	Out, std_ulogic_vector(9 DOWNT0 0)	End of the data in filterbuffer0
End_addr_filter1	Out, std_ulogic_vector(9 DOWNT0 0)	End of the data in filterbuffer1

Fig. 6.18 Table I/O signals ci-demux

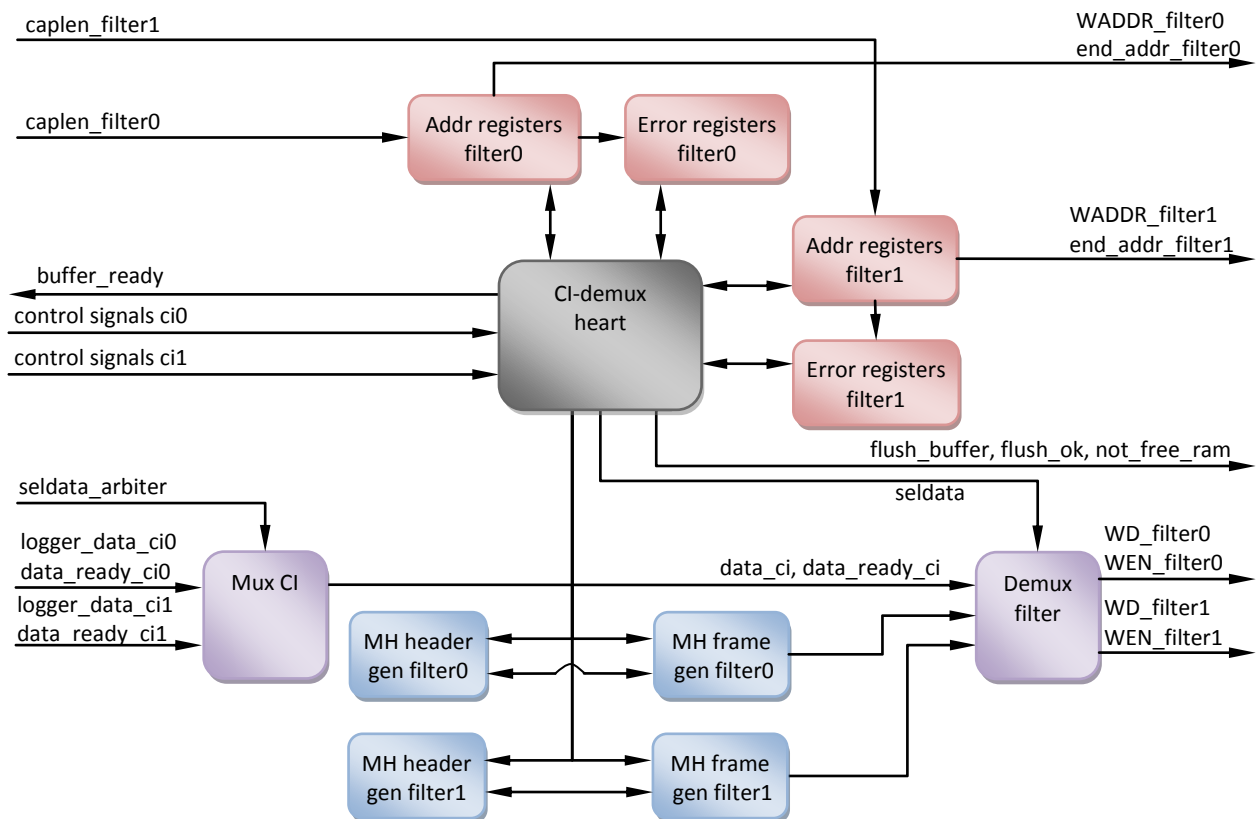
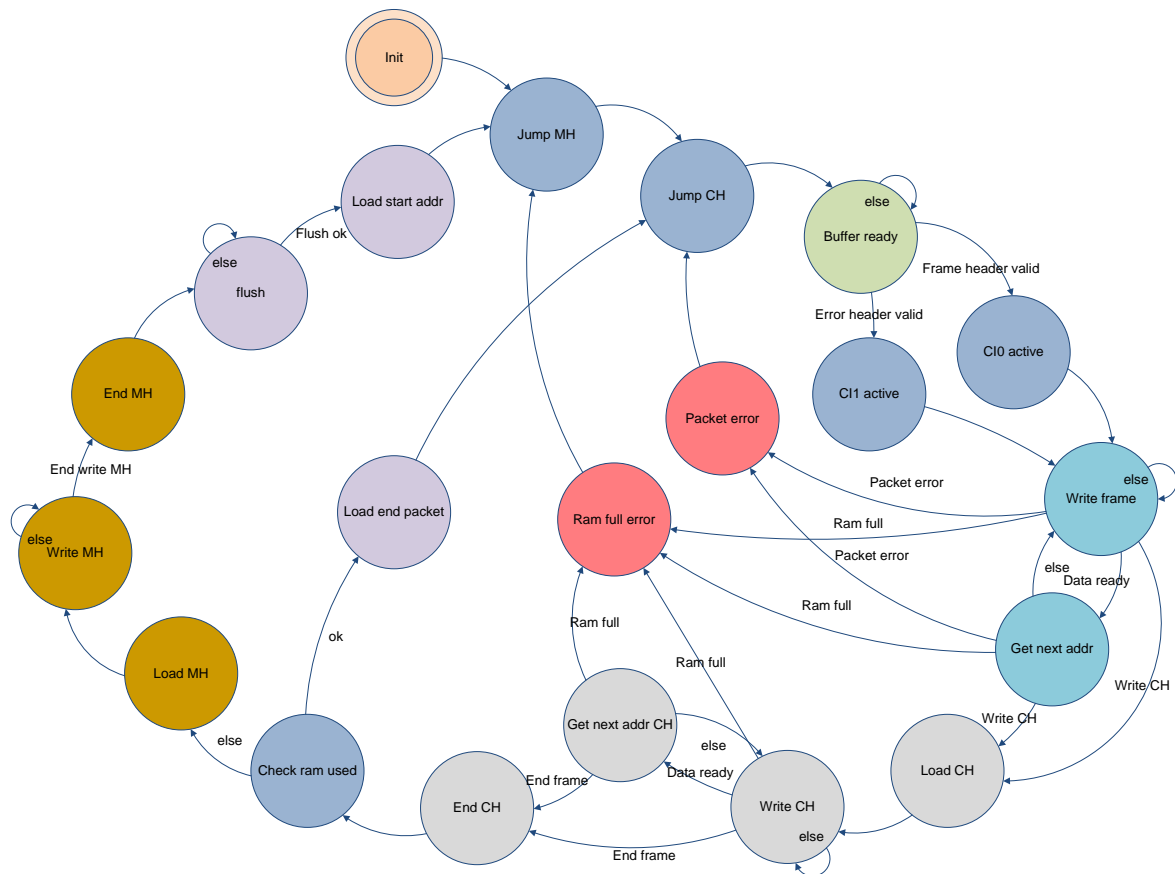


Fig. 6.19 Block schema of CI-demux

- Mux CI: Because there are 2 CI implemented, the Arbiter choose, which one should be active, he set also the seldata_arbiter to forward the right data lines to the buffer.
- Demux filter: The heart choose with help of the filtermatch signal, in which buffers the data should go, it can be just one or both filterbuffer's at the same time. With this bloc we choose also which input we want to store. It can be the data from the CI or one of the MH-data.
- MH headergen filter0 & 1: In this bloc all the values of the MH will be generated. If a new Measurement frame will be created the MH_frame_gen bloc indicate with a signal to increment the sequence-number, and the CI-demux_heart set a signal to indicate to increment the number of packets.
- MH framegen filter0 & 1: If a filterbuffer is full, the heart set the writeMH signal. This bloc take all the values from the MH headergen bloc and write it in the filterbuffers, after finishing, the end_writeMH signal will be set, so that the heart can go on and flush the buffer.
- Addr registers filter 0 & 1: Due to not have registers inside the statemachine, for all the addresses, the addr registers were created. It will save the begin-, end- and actual addresses, it is also responsible to check if there is enough space in the filterbuffer for another packet of the caplength size. For each filterbuffer we have one address registers. The heart can perform some actions on the addresses, like increment, decrement, jumpCH, jumpMH, save_begin, save_end, load_begin, load_end, load_MH and en_check_space.
- Error registers filter 0 & 1: If the filterbuffer is full, it receives the ram_full_error signal from the address register and if the error is enabled by the heart, it will be stored and sent to the heart.

- CI-demux heart: The heart contains a statemachine, which controls all the other bloc's in the CI-demux, he will fill the filterbuffer's, add the MH if it is full and send the flush signal to the sender. In the following picture you can see the statemachine.



👉 Fig. 6.20 Statemachine CI-demux_heart

6.2.8 Output-choice

To have an easier debug possibility in the MP, the Output-choice bloc was created. By pushing the button0 on the board, the output format can be changed from Ethernetframes to UART. It has to be mentioned, that the UART is not that fast as the Ethernet output, therefore the MP will not work on fullspeed, the performance will be much lower and some frames can be lost, due to the UART transmission which takes more than 800 times longer than the 100Mbps Ethernet. The output-choice button should just pressed in case of a restart of the system, it may gives some errors, if the button is pressed during the MP is running. In the table below you can see the I/O signal table of this bloc.

Signal	Type (direction, type)	Description
From Button		
switch	In, std_logic	To change between UART- and Ethernet-output 1 => Ethernet (default) 0 => UART
To /from CI-demux		
Flush_buffer	In, std_ulogic_vector(1 DOWNT0 0)	Indicate which buffers should be flushed
Flush_ok	Out, std_ulogic_vector(1 DOWNT0 0)	If the flush is done, flush_ok <= flush_buffer
To Filterbuffer0		
RADDR_filter0	Out, std_ulogic_vector(9 DOWNT0 0)	Read address of filterbuffer0
REN_filter0	Out, std_ulogic	Read enable signal of filterbuffer0

To Filterbuffer1		
RADDR_filter0	Out, std_ulogic_vector(9 DOWNT0 0)	Read address of filterbuffer1
REN_filter0	Out, std_ulogic	Read enable signal of filterbuffer1
To/from UART transmitter		
Flush_buffer_UART	Out, std_ulogic_vector(1 DOWNT0 0)	Indicate which buffers should be flushed
Flush_ok_UART	In, std_ulogic_vector(1 DOWNT0 0)	If the flush is done, flush_ok <= flush_buffer
RADDR_filter0_UART	In, std_ulogic_vector(9 DOWNT0 0)	Read addr of filterbuffer0 from UART transmitter
REN_filter0_UART	In, std_ulogic	Read enable of filterbuffer0 from UART transmitter
RADDR_filter1_UART	In, std_ulogic_vector(9 DOWNT0 0)	Read addr of filterbuffer1 from UART transmitter
REN_filter1_UART	In, std_ulogic	Read enable of filterbuffer1 from UART transmitter
To/from Sender		
Flush_buffer_sender	Out, std_ulogic_vector(1 DOWNT0 0)	Indicate which buffers should be flushed
Flush_ok_sender	In, std_ulogic_vector(1 DOWNT0 0)	If the flush is done, flush_ok <= flush_buffer
RADDR_filter0_sender	In, std_ulogic_vector(9 DOWNT0 0)	Read addr of filterbuffer0 from sender
REN_filter0_sender	In, std_ulogic	Read enable of filterbuffer0 from sender
RADDR_filter1_sender	In, std_ulogic_vector(9 DOWNT0 0)	Read addr of filterbuffer1 from sender
REN_filter1_sender	In, std_ulogic	Read enable of filterbuffer1 from sender

⇒ Fig. 6.21 Table I/O signals output-choice

6.2.9 Sender

The sender flushes the filterbuffer's if the signal flush_buffer is set, all the data from the filterbuffer will be copied in the transmitter-buffer as well as the Ethernetheader (EthH) will be added. The EthH is statically and cannot be changed during runtime of the MP. After that, the flush_ok signal will be set, to indicate to the CI-Demux, that the filterbuffer's are empty and can used for new packets. In the figures below, you can see the I/O signal table and a schematic overview about the different subbloks in the sender. The sender has for each accessed filter an address register, for the transmitterbuffer is also an errorregister added to store the ramfull error.

Signal	Type (direction, type)	Description
From CI-demux		
End_addr_filter0	In, std_ulogic_vector(9 DOWNT0 0)	Last used address in the filterbuffer0
End_addr_filter1	In, std_ulogic_vector(9 DOWNT0 0)	Last used address in the filterbuffer1
Incr_matched	In, std_ulogic	Increment number of frames who passed a filter
To/from Phy (Toplevel)		
Tx_error	In, std_logic	If an error occurred during transmitting the phy will set this signal
Tx_clk	In, std_logic	25MHz transmitting clock from the phy
Tx_reset	In, std_logic	Reset signal who is synchronized with the tx_clk
Tx_enable	Out, std_logic	To enable the phy for transmitting a packet
Tx_data	Out, std_logic_vector(3 DOWNT0 0)	Data lines for the phy
To/from Output choice		
RADDR_filter0	Out, std_ulogic_vector(9 DOWNT0 0)	Read address of filterbuffer0
REN_filter0	Out, std_ulogic	Read enable signal of filterbuffer0
RADDR_filter1	Out, std_ulogic_vector(9 DOWNT0 0)	Read address of filterbuffer1
REN_filter1	Out, std_ulogic	Read enable signal of filterbuffer1
Flush_buffer	In, std_ulogic_vector(1 DOWNT0 0)	Indicate which buffers should be flushed
Flush_ok	Out, std_ulogic_vector(1 DOWNT0 0)	If the flush is done, flush_ok <= flush_buffer
From Filterbuffer0		
RD_filter0	Out, std_ulogic_vector(15 DOWNT0 0)	Data lines of filterbuffer0
From Filterbuffer1		
RD_filter1	Out, std_ulogic_vector(15 DOWNT0 0)	Data lines of filterbuffer1

⇒ Fig. 6.22 Table I/O signals sender

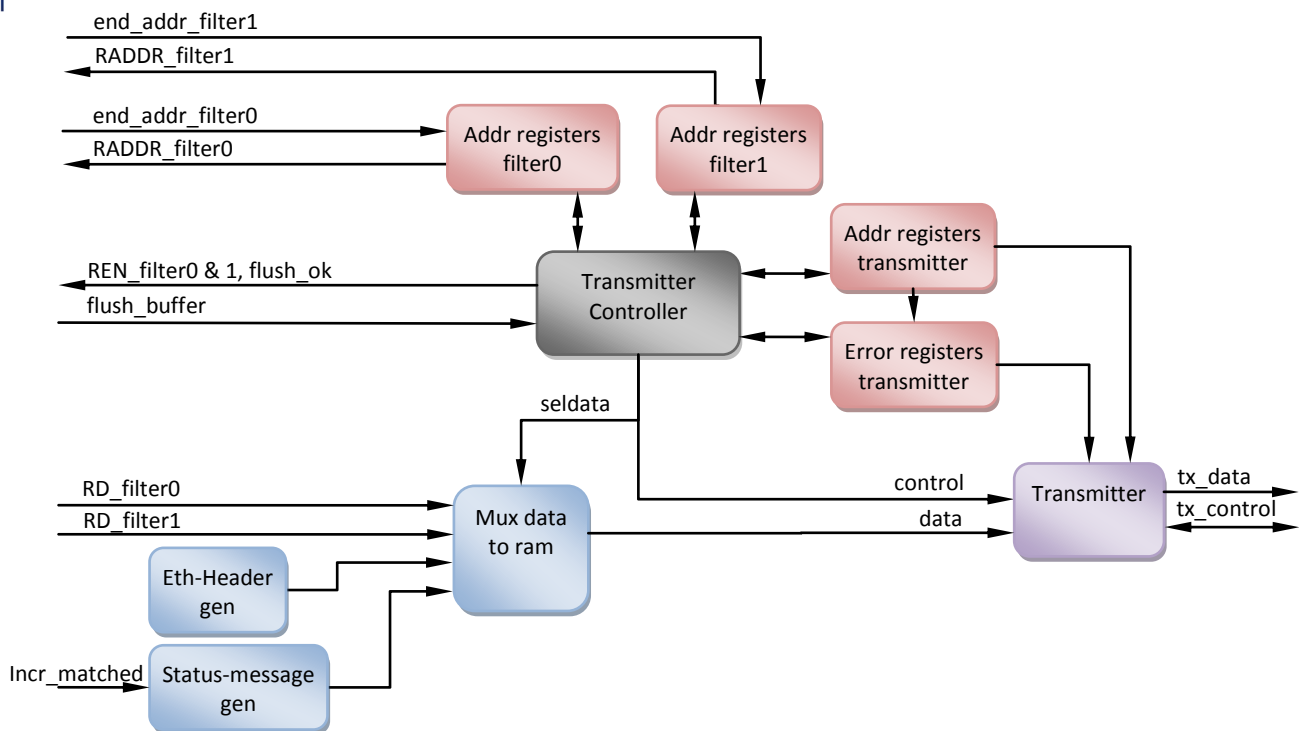


Fig. 6.23 Block schema of sender

- **Addr registers filter0 & 1:** For each filterbuffer an address register, for the read access was created. The transmitter controller can perform the increment or reset of the address registers.
- **Addr registers transmitter:** For the Tx-buffer in the transmitter, an address register was created to store the actual-, begin- and end-address of a frame. The transmitter controller can perform some actions on the addresses, like increment, decrement, save_begin, save_end, load_begin, load_end.
- **Error register transmitter:** If the Tx-buffer is full, it receives the **ram_full_error** signal from the address register and if the error is enabled by the controller, it will be stored and sent to the controller.
- **Eth-header gen:** This block generate the Ethernet header. The Ethernet destination and source addresses are fixed. The value of the Ethernet type depends on which frame will be sent. For a measurement frame 0x0810 and for a status frame 0x0800.
- **Status-message gen:** This block generates the UDP/IP headers fields and the data for the statusmessage, each second it will indicate to the controller that he will write the message into the Tx-buffer. The IP and UDP checksum will dynamically calculated, if something changes in the message.
- **Mux data to ram:** The transmitter controller can choose which data should go to the Tx-buffer. It can be the Ethernet data, the filter 1 data, the filter 2 data or the status message data.
- **Transmitter:** Further information see Chapter 6.2.9.1 below.
- **Transmitter controller:** Inside the transmitter controller a statemachine is implemented, which controls all the blocks inside the sender. First, the Ethernet header will be written into the Tx-buffer then he will wait for a message to send. This can be either a measurement frame or a status message. In the following picture you can see the statemachine of the transmitter controller.

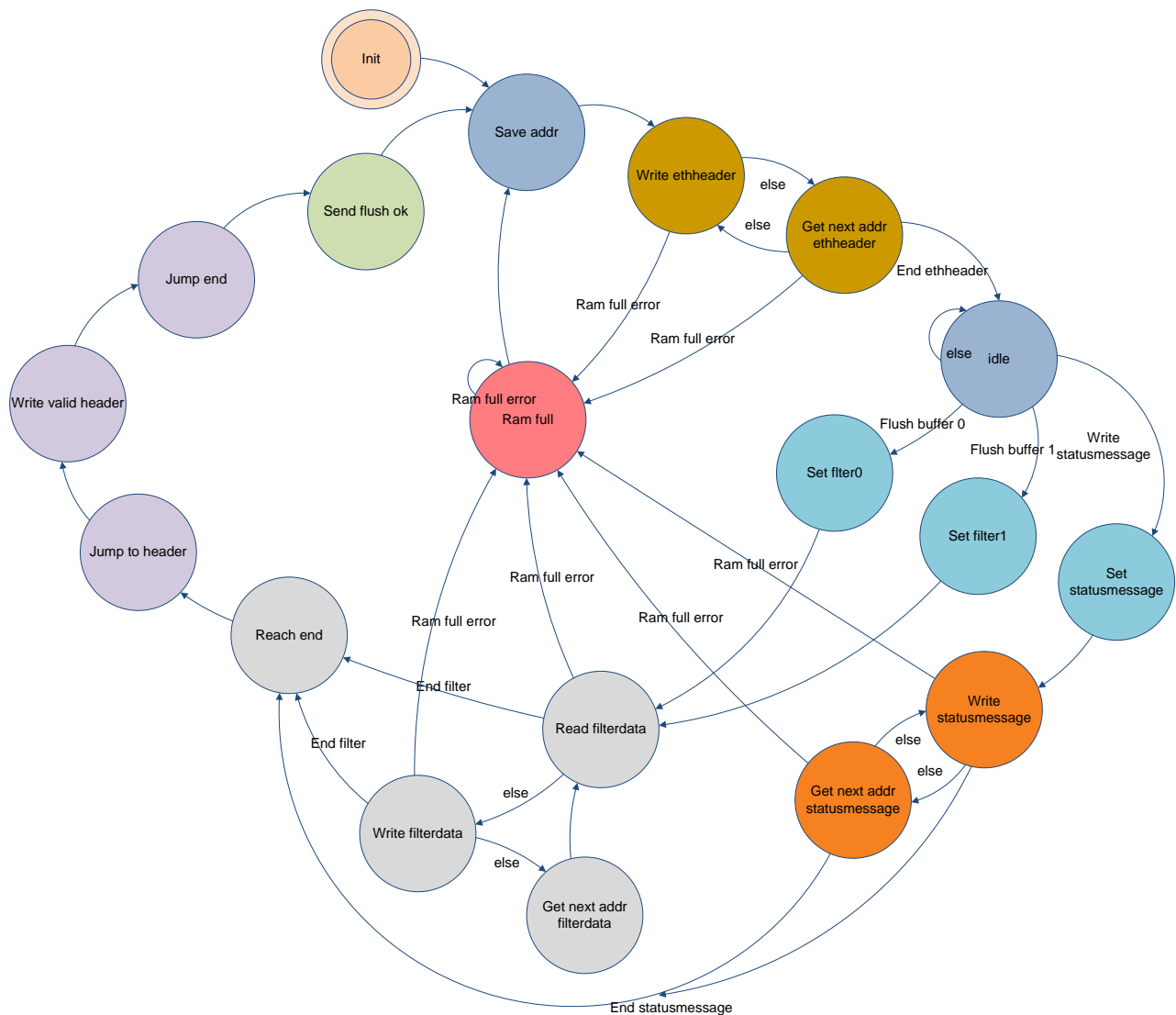


Fig. 6.24 State machine transmitter_controller

6.2.9.1 Transmitter

The transmitterblock is made by the school in Switzerland, like in the receiver, I made just some adaptations for my design. For additional information's about the transmitter, have a look at appendix 7. Inside the transmitter is a dualport buffer, which stores all the Ethernet frames to send. To see the structure of this buffer, have a look at Chapter 6.2.1 Intermediate frames.

Appendix 7 SimAP Design report

The changes include the following blocs:

- The reset synchronization bloc was moved into the resetgen bloc in the top-level to combine all the reset synchronization's in one bloc.
- Replaced the Xilinx buffer by an Actel dual port buffer.

In the next table shows you the I/O signal of this bloc

Signal	Type (direction, type)	Description
To/from Phy (Toplevel)		
Tx_error	In, std_logic	If an error occurred during transmitting the phy will set this signal
Tx_clk	In, std_logic	25MHz transmitting clock from the phy
Tx_reset	In, std_logic	Reset signal from the phy
Tx_enable	Out, std_logic	To enable the phy for transmitting a packet
Tx_data	Out, std_logic_vector(3 DOWNT0 0)	Data lines for the phy
To/from transmitter controller		
End_of_frame	Out, std_uologic	Indicate the end of transmitting of a frame
Not_set_ram	Out, std_uologic	If the data should not taken from the buffer but from the datalines directly
Write_ram	In, std_uologic	Enable write signal for the transmitter-buffer
To/from address registers		
Address	In, std_uologic_vector(9 DOWNT0 0)	Actual write address of the transmitter-buffer
Base_addr	Out, std_uologic_vector(9 DOWNT0 0)	Address till the transmitter controller can write new packets to send
To/from muxram to data		
Data	In, std_uologic_vector(15 DOWNT0 0)	Write data-lines for the transmitter-buffer

Fig. 6.25 Table I/O signals transmitter

A schematic overview about the bloc inside the receiver is given below.

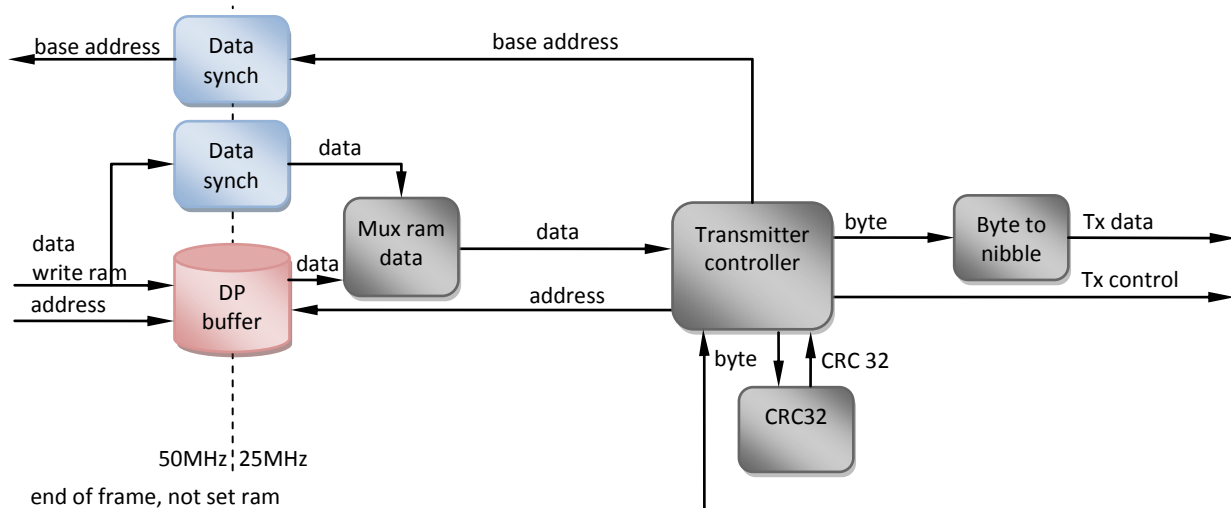


Fig. 6.26 Block schema of transmitter

The sent packets from the transmitter was captured by a PC with the program Wireshark

0000	ff ff ff ff ff ff 22 22 22 22 22 08 10 00 00" ".....	
0010	00 09 00 0d 00 00 00 00 00 00 00 00 05-FPGAM	EthH
0020	00 00 43 49 31 20 52 58 20 20 2d 46 50 47 41 4d	..CII RX)B.u.....	
0030	50 30 00 00 00 00 00 00 29 42 e3 75 f9 a0 00 00	..2.....	
0040	00 00 32 00 00 00 ff ff ff ff ff ff 00 19 b9 20	..?..E..9d.v.	MH
0050	cc 3f 08 00 45 00 00 39 13 88 00 00 64 11 76 ca5%.0	
0060	c0 a8 a5 11 c0 a8 a5 ff 13 88 00 35 00 25 fe 4f	Ethernet testfra	
0070	45 74 68 65 72 6e 65 74 20 74 65 73 74 66 72 61	me / 57 bytes...	
0080	6d 65 20 2f 20 35 37 20 62 79 74 65 73 ed 1d 0b	..CII RX -FPGAM	CH
0090	e3 ee 43 49 31 20 52 58 20 20 2d 46 50 47 41 4d	P0.....)...j...	
00a0	50 30 00 00 00 00 00 00 29 b3 b1 fc 6a e0 00 00	..2.....	
00b0	00 00 32 00 00 00 ff ff ff ff ff ff 00 19 b9 20	..?..E..9d.v.	
00c0	cc 3f 08 00 45 00 00 39 13 88 00 00 64 11 76 ca5%.0	
00d0	c0 a8 a5 11 c0 a8 a5 ff 13 88 00 35 00 25 fe 4f	Ethernet testfra	
00e0	45 74 68 65 72 6e 65 74 20 74 65 73 74 66 72 61	me / 57 bytes...	
00f0	6d 65 20 2f 20 35 37 20 62 79 74 65 73 ed 1d 0b	..CII RX -FPGAM	
0100	e3 ee 43 49 31 20 52 58 20 20 2d 46 50 47 41 4d	P0..... *\$y.....	
0110	50 30 00 00 00 00 00 00 2a 24 79 18 d5 e0 00 00	..2.....	
0120	00 00 32 00 00 00 ff ff ff ff ff ff 00 19 b9 20	..?..E..9d.v.	
0130	cc 3f 08 00 45 00 00 39 13 88 00 00 64 11 76 ca5%.0	
0140	c0 a8 a5 11 c0 a8 a5 ff 13 88 00 35 00 25 fe 4f	Ethernet testfra	
0150	45 74 68 65 72 6e 65 74 20 74 65 73 74 66 72 61	me / 57 bytes...	
0160	6d 65 20 2f 20 35 37 20 62 79 74 65 73 ed 1d 0b	..CII RX -FPGAM	
0170	e3 ee 43 49 31 20 52 58 20 20 2d 46 50 47 41 4d	P0..... *.B0. ...	
0180	50 30 00 00 00 00 00 00 2a 95 42 30 ad 20 00 00	..2.....	
0190	00 00 32 00 00 00 ff ff ff ff ff ff 00 19 b9 20	..?..E..9d.v.	
01a0	cc 3f 08 00 45 00 00 39 13 88 00 00 64 11 76 ca5%.0	
01b0	c0 a8 a5 11 c0 a8 a5 ff 13 88 00 35 00 25 fe 4f	Ethernet testfra	
01c0	45 74 68 65 72 6e 65 74 20 74 65 73 74 66 72 61	me / 57 bytes...	
01d0	6d 65 20 2f 20 35 37 20 62 79 74 65 73 ed 1d 0b		

Fig. 6.27 Received filterbuffer

6.2.10 UART

By pushing the button0 on the Controller-board during restart, the output format will be changed from Ethernet to UART. In the UART case, all the data from the filterbuffer, who has to be flushed, will be send through the RS232 Interface, no additional data will be added. In the following pictures, you can see a schematic overview, about the sub bloc's inside the UART, the I/O signal table and a flushed filterbuffer with a MH, a CH and 1 packets received by a PC, with a serial port terminal.

It has to be mentioned that the UART output is just used as debug output, because the speed is limited to 115Kbps, during the flush of the filterbuffer's, some packets will be lost if the traffic on the CI's is too high.

Signal	Type (direction, type)	Description
From CI-demux		
End_addr_filter0	In, std_logic_vector(9 DOWNT0 0)	Last used address in the filterbuffer0
End_addr_filter1	In, std_logic_vector(9 DOWNT0 0)	Last used address in the filterbuffer1
To/from Phy (Toplevel)		
TX1	In, std_logic	If an error occurred during transmitting the phy will set this signal
To/from Output choice		
RADDR_filter0	Out, std_ulogic_vector(9 DOWNT0 0)	Read address of filterbuffer0
REN_filter0	Out, std_ulogic	Read enable signal of filterbuffer0
RADDR_filter1	Out, std_ulogic_vector(9 DOWNT0 0)	Read address of filterbuffer1
REN_filter1	Out, std_ulogic	Read enable signal of filterbuffer1
Flush_buffer	In, std_ulogic_vector(1 DOWNT0 0)	Indicate which buffers should be flushed
Flush_ok	Out, std_ulogic_vector(1 DOWNT0 0)	If the flush is done, flush_ok <= flush_buffer
From Filterbuffer0		
RD_filter0	Out, std_ulogic_vector(15 DOWNT0 0)	Data lines of filterbuffer0
From Filterbuffer1		
RD_filter1	Out, std_ulogic_vector(15 DOWNT0 0)	Data lines of filterbuffer1

Fig. 6.28 Table I/O signals UART

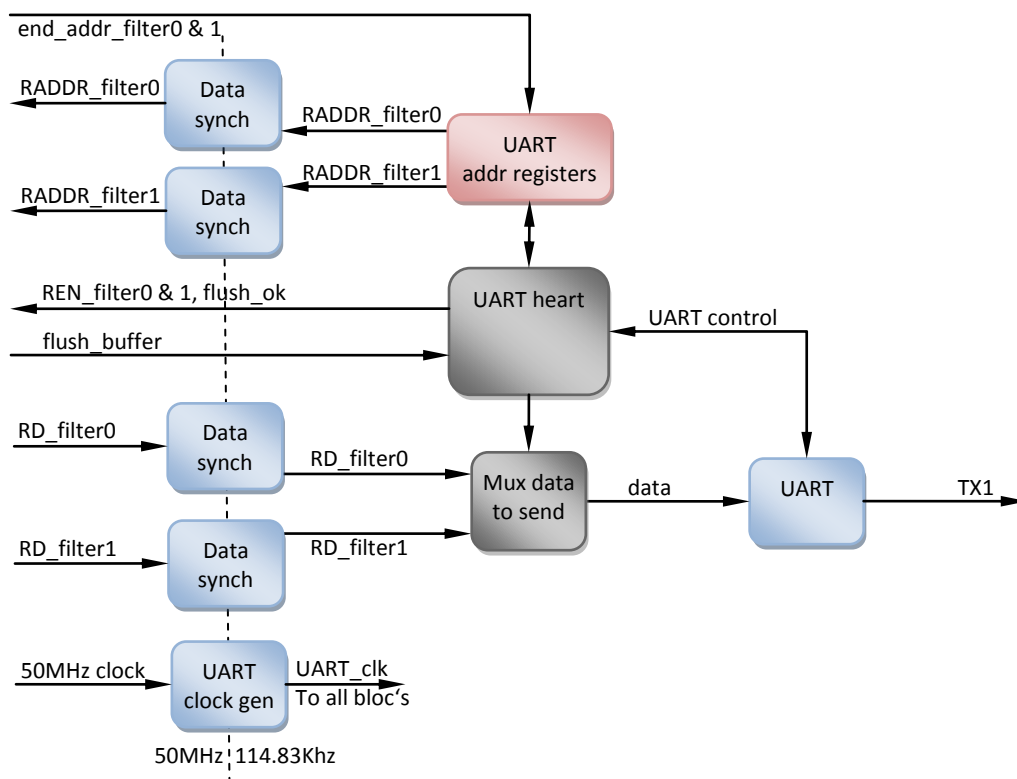


Fig. 6.29 Block schema of UART

☞ Fig. 6.30 Received filterbuffer

-
- ```

graph TD
 Init((Init)) --> Wait_flush((Wait flush))
 Wait_flush --> Wait_flush
 Wait_flush -- "flush filter buffer 0 or 1" --> read((read))
 read --> read
 read -- "C12 request" --> send1((send1))
 send1 --> send1
 send1 -- "else" --> idle1((idle1))
 idle1 -- "Tx empty" --> send2((send2))
 send2 --> send2
 send2 -- "else" --> idle2((idle2))
 idle2 -- "Tx empty" --> Get_next_addr((Get next addr))
 Get_next_addr --> Get_next_addr
 Get_next_addr -- "End flush" --> flushed((flushed))
 flushed --> flushed
 flushed -- "End flush" --> Wait_flush
 flushed -- "End flush" --> send1
 flushed -- "End flush" --> idle1
 flushed -- "End flush" --> idle2

```

- 42/49 -



The configuration of the UART is given in the table below.

| UART configuration |           |
|--------------------|-----------|
| Baud rate          | 115200bps |
| Data bits          | 8         |
| Parity type        | None      |
| Stop bits          | 1         |
| Flow control       | Off       |

Fig. 6.32 Table UART configuration

### 6.2.11 TSC

The Time Synchronization Client provides us the possibility to have a more accurate timestamping. In the actual version of the MP, the timestamp is a relative value, by pressing the reset button, the time starts from 0 in picoseconds with an accuracy of 20ns.

A previous diploma work was already made in this topic, it is called Time and frequency synchronization, see appendix 8. Unfortunately the hardware that was already made for the timesynchronization was not ready to use. Therefore there is no absolute timestamping and not all of the features could be tested.

In following section you have an explanation how the time synchronization works.

Appendix 8 Time and frequency synchronization

The device structure for the time synchronization network, is a Master – Slave – Client architecture. The MP made in this project, is used as client for the RS422 Network, this is obvious in the following picture.

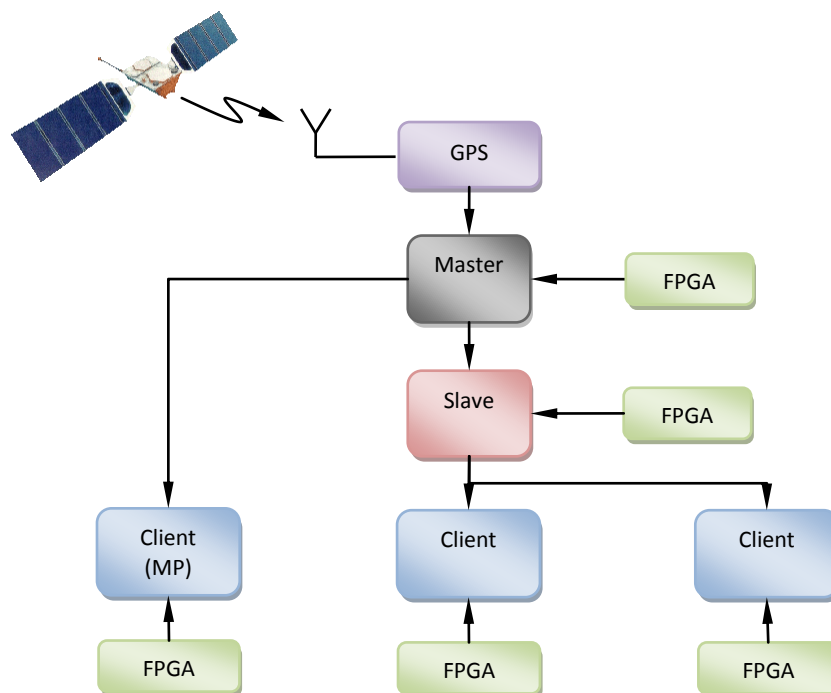


Fig. 6.33 Time synchronisation network

To have an exact time, 3 Points have to be considered.

- Prevent the internal clock from drifting
- Calculate the delay of the time transmission
- Take the actual time from the GPS antenna (not implemented)

There were 4 signals from the RS422 transceiver to implement this feature.



| Signal                       | Type (direction, type)             | Description                                                                                                      |
|------------------------------|------------------------------------|------------------------------------------------------------------------------------------------------------------|
| <b>To CI 1&amp;2</b>         |                                    |                                                                                                                  |
| <b>timestamp</b>             | Out, std_logic_vector(96 DOWNTO 0) | Relative time in picoseconds                                                                                     |
| <b>To/from Output choice</b> |                                    |                                                                                                                  |
| <b>PPS</b>                   | In, std_logic                      | The PPS signal is a 1Hz clock from the GPS antenna, with this clock you prevent the internal clock from drifting |
| <b>Delay_in</b>              | In, std_logic                      | With the delay in and out signal it is possible to calculate the delay between Client-Slave or Client-Master     |
| <b>Delay_out</b>             | Out, std_logic                     | See above                                                                                                        |
| <b>Time</b>                  | In, std_logic                      | This signal should give the MP the actual time but unfortunately this could not implemented                      |

Fig. 6.34 Table I/O signals TSC

In the following figure you can see a schematic overview about the sub blocks in the TSC.

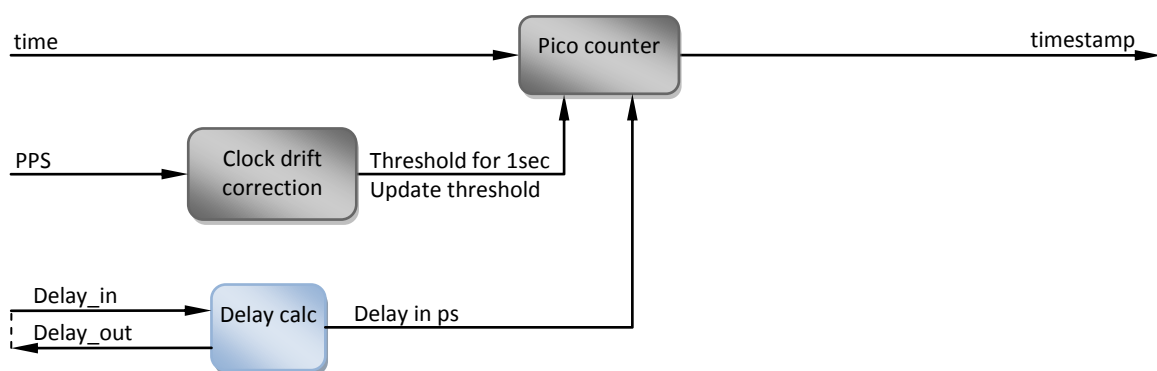


Fig. 6.35 Block schema of TSC

- Clock drift correction: If a rising edge of the PPS-signal arrives, a counter is started till the next rising edge, to know the number of systemclock pulses for 1 sec.
- Pico counter: He has a 96bit counter to count the time in picoseconds for the CH. The threshold from the clock\_drift\_correction-block gives the number of counts, till 1sec is passed.
- Delay calc: To calculate the delay, a small statemachine is used. He sends a 1 on delay\_out and starts a counter, till the 1 appears on the signal delay\_in. The counter has than the time of the delay. It has to be divided by 2, to know the one way delay. The statemachine is given below.

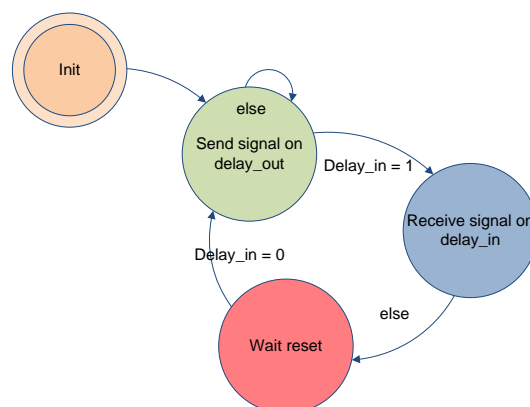


Fig. 6.36 Statemachine delay\_calc

## 7 Tests

In this part, the hardware tests I made, are explained in detail. Before the FPGA could be programmed, the design has to be verified in the simulations. But because the two boards was from different manufactures, one with a CPLD from Xilinx the other with a FPGA from Actel, it was not possible to simulate the behavior between the two boards, therefore, I could test just partially in the simulation. There were a lot of problems with the synchronization of both boards. After that was solved, the hardware tests could be made.

The implementation was tested with a Frame generator program where it is possible to define the content of a packet and the time between 2 packets. A constant frame rate could be generated. In the last week we also made together with Arlos Patrik, a test run where the custom MP could be compared with a real MP.

The following issues were tested:

- Receiving: The packets received properly on both CI. At the end also the CRC-check went fine and worked as expected. The receiving works with a 100Mbps and also with the 10Mbps connection.
- Filtering: The filtering was successful tested. In the actual version it is possible to filter the first 42 bytes of a packet.
- CH / MH: The capture- and measurement-header will be added with the right values. There is a problem with the bit ordering of some fields, in the CH. It should be easy to solve.
- Timestamping: The time of the frame is taken when the start of frame signal is detected, that is what we expected. Also here the bit order was not compatible with version 0.6 of the DPPI interface.
- Statusmessage: The status messages are sending every second with the right content.
- Switch UART-Eth output: As long as the button is pressed before restart, the switch between the two outputs worked well. But if it is pressed during the MP is running, it may be able to lose some packets. In each case the UART output has not a good performance, some packets will be lost if the traffic is too high.

## 8 Actual state

The MP was developed, but the system is not ready for the market, there are some implementation left, like the Control messages, to and from the DPPI interface. But since the last version of Gubler Olivier, some improvements could be made.

### Interface

The Interface was programmed and tested in the 32bit and 4bit version. In the last week in Sweden the Interface board was damaged (VCC and GND are connected), and could not further used. I implemented a demonstration MP with the 2 Ethernetplugs of the Dev-Board. One acts as CI and the other as connection to the MArN.

### Controller

Not all of the goals were reached. We made some simplifications, that I could finish the project. One problem was, that the Arm Core could not use as indented. The actual version has two CI with two working filters. The capture length is adaptable from none to fullframe catch and also the filterlength can be changed from 42bytes to 512bytes per filter. The communication with the DPPI Interface worked, but not all the control messages could be implemented. The receiving part of these messages like add/remove filter, flush buffer are missing.

## 9 Further work

I would like to make suggestions for the further development of the project:

- Finish further development of the Interface-board
  - Mount the optical receivers on the Interface
  - Test the Interface
- Build of a Interface with other Input Sources, like 1000Base-TX
- Build a proper RS422 transmitter for a real timestamping
- Build of the Controller Board with an Actel Core MP7 FPGA
  - In this project was a prefabricated board used as Controller, the next step would be to make the own board with just the necessary functions.
- Better use of the ARM Core
  - Interface between ARM and FPGA logic
  - In this Project the ARM Core was not used as intended, in the further development it could be used more intensive as now, this could be done by an interface between the Arm-core and the FPGA logic.
- Implement new functions in the Controller
  - Optical capture interface
  - MP with more than 2 capture interfaces
  - Optical ⇔ Twisted pair converter
  - Implement Control frames (flush buffer, add/remove filter)

## 10 Conclusion and remarks

During the semester project and the diploma work, I could see how a prototype is growing from the beginning. At first, the conception of the board, choose the right components and testing. After that, I could start with the programming of the FPGA and the CPLD. Because the chips on the boards were from different manufacturer, Actel and Xilinx, I had to work with the corresponding programs it was really interesting to see the advantages and disadvantages of each program.

In the project I improved my acknowledge about VHDL, hardware conception and project management. It was a challenging project, a lot of problems occurred during the development of each step. Now at the end, I would handle some things in a different way, so I look forward to my next project.

I had to work together with many people, without those, it would not have been possible, to progress so far in this project. Thanks to everyone, who was helping me. Your support, ideas and comments were very helpful.

Unfortunately, I did not reach all goals of the project, because of the lack of time. In further projects, it should be possible to have closer look at those points.

These 5 months opened my horizon and gave me the chance to get in contact with new people, a new culture and an interesting country. The BTH school is very international, there work peoples from everywhere of the world. I hope that it will always be possible to do such exchanges for students, to show them new fields in science and to open their mind for other countries and cultures. To make new connections in the world.

Zahno Silvan:



## 11 Glossary

|            |                                                                                                                                                                                                                                                                                                                                                                                     |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MP         | <b>Measurement Point</b>                                                                                                                                                                                                                                                                                                                                                            |
| FPGA       | <b>Field Programmable Gate Array</b> , is a semiconductor device containing programmable logic components and programmable interconnects.                                                                                                                                                                                                                                           |
| CPLD       | <b>Complex Programmable Logic Device</b> is a programmable logic device. The building block of a CPLD is the macro cell, which contains logic implementing disjunctive normal form expressions and more specialized logic operations.                                                                                                                                               |
| Dev-Board  | <b>Development Board</b> from Actel, this is a Board which allows to develop different kind of application. In this paper it is also called Controller-board.                                                                                                                                                                                                                       |
| Phy        | <b>Physical</b> , this IC representing the first layer of the OSI-model.                                                                                                                                                                                                                                                                                                            |
| MARn       | <b>Measurement Area Network</b> , network which analyzes and tread the sent Measurement frames.                                                                                                                                                                                                                                                                                     |
| MARc       | <b>Measurement Area Controller</b> , this device controls the MP.                                                                                                                                                                                                                                                                                                                   |
| Eth        | <b>Ethernet</b>                                                                                                                                                                                                                                                                                                                                                                     |
| CI         | <b>Captured Interface</b> , a MP can have different CI, each can capture one line, in this project there exist 2 CI                                                                                                                                                                                                                                                                 |
| MH         | <b>Measurement Header</b> , on each sended packet are a MH added to give the DPMI some additional informations                                                                                                                                                                                                                                                                      |
| CH         | <b>Capture Header</b> , on each captured frame is a CH added, which provide some information about the captured frame                                                                                                                                                                                                                                                               |
| SOF        | <b>Start Of Frame</b> , a Ethernet packet starts with the preamble (0x5) and after that the sof delimiter (0x5D), it indicates the begin of a Ethernet packet.                                                                                                                                                                                                                      |
| TP         | <b>Twisted Pair</b> , is a form of wiring, this cables are used to connect the MP to the MARn                                                                                                                                                                                                                                                                                       |
| VHDL       | <b>Very High Speed Integrated Circuit Hardware Description Language</b> , is a programming language to describe hardware                                                                                                                                                                                                                                                            |
| JTAG       | <b>Joint Test Action Group</b> , is the usual name used for the IEEE 1149.1 standard entitled Standard Test Access Port and Boundary-Scan Architecture for test access ports used for testing printed circuit boards using boundary scan                                                                                                                                            |
| 10BaseT    | <b>10Base-T</b> is a form of medium speed Ethernet, providing 10Mbit/s                                                                                                                                                                                                                                                                                                              |
| 100BaseTX  | <b>100Base-TX</b> is the predominant form of Fast Ethernet, providing 100 Mbit/s Ethernet. It introduces an additional, medium dependent sublayer, which employs MLT-3 as a final encoding of the data stream before transmission                                                                                                                                                   |
| 100Base-FX | <b>100Base-FX</b> is a version of Fast Ethernet over optical fiber. It uses two strands of multi-mode optical fiber for receive and transmit                                                                                                                                                                                                                                        |
| RJ45       | <b>Registered Jack (RJ)</b> is a standardized physical interface for connecting telecommunications equipment or computer networking equipment. The standard designs for these connectors and their wiring are named RJ11, RJ14, RJ45, etc.                                                                                                                                          |
| PCB        | <b>Printed Circuit Boards</b> are used to mechanically support and electrically connect electronic components using conductive pathways, or traces, etched from copper sheets laminated onto a non-conductive substrate.                                                                                                                                                            |
| CRC        | <b>Cyclic Redundancy Check (CRC)</b> is a type of function that takes as input a data stream of unlimited length and produces as output a value of a certain fixed size. The term CRC is often used to denote either the function or the function's output. A CRC can be used in the same way as a checksum to detect accidental alteration of data during transmission or storage. |
| RTC        | <b>Real-Time Clock</b> is a computer clock that keeps track of the current time. Although the term often refers to the devices in personal computers, servers and embedded systems, RTCs are present in most any electronic device which needs to keep accurate time.                                                                                                               |
| UART       | <b>Universal Asynchronous Receiver/Transmitter</b> , is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers.                                                                                                                              |

## 12 References

Books:

“On The Quality Of Computer Network Measurements”

Patrik Arlos

Bleckinge Institute of Technology

Doctoral Dissertation Series No. 2005:05

“VHDL-Synthese. Entwurf digitaler Schaltungen und Systeme“

Jürgen Reichardt, Bernd Schwarz

Oldenbourg, 2007

Manufacturer:

[www.actel.com](http://www.actel.com)

[www.xilinx.com](http://www.xilinx.com)

[www.intel.com](http://www.intel.com)

[www.belfuse.com](http://www.belfuse.com)

[www.magjack.com](http://www.magjack.com)

[www.nationalsemiconductor.com](http://www.nationalsemiconductor.com)

Distributors:

[www.elca.se](http://www.elca.se)

[www.distrelec.com](http://www.distrelec.com)

[www.silica.com](http://www.silica.com)

[www.farnell.com](http://www.farnell.com)

[www.reselec.ch](http://www.reselec.ch)

[www.msc.ch](http://www.msc.ch)

Informations:

[www.wikipedia.com](http://www.wikipedia.com)

[www.opencores.org](http://www.opencores.org)

[www.fpga4fun.com](http://www.fpga4fun.com)

[www.wireshark.org](http://www.wireshark.org)

and much other sites...

## 13 Appendix

- Appendix 1: Report Semester project
- Appendix 2: Schematic Interface
- Appendix 3: Schematic Converter
- Appendix 4: CoreMP7 development kit users guide
- Appendix 5: Datasheet CoreMP7
- Appendix 6: Passive measurement infrastructure
- Appendix 7: SimAP Design report
- Appendix 8: Time and frequency synchronization

## **Appendix 1: Report Semesterproject**



**HEVs**

haute école valaisanne  
hochschule wallis



Systems Engineering: Infotronic

# Semester project

Part I of the Diploma Work

## - Frame Capturing and Sending in FPGA -

Author

Under guidance of

Version

Sion, 13.01.08

Zahno Silvan

Corthay Francois and Gubler Olivier

v 1.0



## Table of contents

|         |                                                                     |    |
|---------|---------------------------------------------------------------------|----|
| 1       | Author .....                                                        | 4  |
| 2       | Preface .....                                                       | 4  |
| 2.1     | Introduction .....                                                  | 4  |
| 2.2     | Appendix .....                                                      | 4  |
| 2.3     | Structure of this project .....                                     | 4  |
| 2.3.1   | Planification .....                                                 | 4  |
| 3       | Overview .....                                                      | 5  |
| 4       | Hardware .....                                                      | 6  |
| 4.1     | Problem analysis and solution approaches .....                      | 6  |
| 4.2     | Interface .....                                                     | 6  |
| 4.2.1   | Active Ethernet tap .....                                           | 8  |
| 4.2.2   | Passive Ethernet tap .....                                          | 8  |
| 4.2.3   | Sending the Data from the plug to the Phy .....                     | 11 |
| 4.2.4   | Physical Interface .....                                            | 13 |
| 4.2.4.1 | Strap options .....                                                 | 13 |
| 4.2.5   | Sending of the packets to the Controller .....                      | 15 |
| 4.2.5.1 | Choice of the CPLD .....                                            | 15 |
| 4.2.6   | RS422 interface .....                                               | 16 |
| 4.2.7   | Connector .....                                                     | 17 |
| 4.3     | Controller .....                                                    | 18 |
| 4.3.1   | Choice of the FPGA .....                                            | 18 |
| 4.3.2   | Design flow .....                                                   | 20 |
| 4.4     | Converter-board .....                                               | 21 |
| 4.4.1   | Pin assignment on the Controller .....                              | 21 |
| 5       | Construction of the hardware .....                                  | 22 |
| 5.1     | Interface PCB .....                                                 | 22 |
| 5.2     | Converter PCB .....                                                 | 22 |
| 5.3     | List of order for Project Frame Capturing and sending in FPGA ..... | 22 |
| 6       | Actual state .....                                                  | 24 |
| 7       | Further work .....                                                  | 24 |
| 8       | Conclusion and remarks .....                                        | 25 |
| 9       | Glossary .....                                                      | 26 |
| 10      | Bibliography .....                                                  | 27 |
| 11      | Appendix .....                                                      | 27 |

## Table of figures

|                                                                               |    |
|-------------------------------------------------------------------------------|----|
| Fig. 2.1 Time table diagram of the Project .....                              | 4  |
| Fig. 4.1 Bloc diagram of the Interface .....                                  | 7  |
| Fig. 4.2 Active tap bloc diagram of the Interface .....                       | 8  |
| Fig. 4.3 High-impedance-bloc .....                                            | 9  |
| Fig. 4.4 Offset circuit .....                                                 | 9  |
| Fig. 4.5 Passive tap bloc diagram of the Interface .....                      | 10 |
| Fig. 4.6 Worse passive tap of the fiber-network.....                          | 10 |
| Fig. 4.7 Termination circuit for RJ45 and construction of the RJ45-plug ..... | 11 |
| Fig. 4.8 Termination circuit for fiber-plug.....                              | 11 |
| Fig. 4.9 ZTool to calculate the line impedance .....                          | 12 |
| Fig. 4.10 Application of the Phy .....                                        | 13 |
| Fig. 4.11 Table of mode options .....                                         | 14 |
| Fig. 4.12 Table of MAC interface options .....                                | 14 |
| Fig. 4.13 Table of Led mode select.....                                       | 14 |
| Fig. 4.14 CPLD-Pin table .....                                                | 15 |
| Fig. 4.15 RS422 network overview.....                                         | 16 |
| Fig. 4.16 Bloc diagram of the RS422 Interface.....                            | 17 |
| Fig. 4.17 CPLD-Pin table .....                                                | 17 |
| Fig. 4.18 FPGA choice table .....                                             | 18 |
| Fig. 4.19 MP7 FPGA-Core .....                                                 | 18 |
| Fig. 4.20 Actel CoreMP7 Developement Kit .....                                | 19 |
| Fig. 4.21 Schema of the design flow .....                                     | 20 |
| Fig. 4.22 Blocdiagramm of Converter-board .....                               | 21 |
| Fig. 4.23 Pin assignment table .....                                          | 21 |
| Fig. 5.1 List of order .....                                                  | 23 |

# 1 Author

Zahno Silvan

## 2 Preface

### 2.1 Introduction

This Semester project should be the first part of my diploma work, which I will continue in Sweden, Blekinge.

The subject of the Diploma Work will be "Frame capturing and sending in FPGA". In this part of the project, I will develop the Hardware, or a part of it, for the work in Sweden. It is the sequel of the Diploma work of Oliver Gubler, who also made his work in Sweden.

In fact, in this part, I will develop a more powerful Hardware than I usually need. For my specific diploma work, I don't need Hardware with so many I/O Interfaces, but to extend the project for future students, it's good to have some expansion capabilities in the Hardware.

In consideration of this reason, my board should contains the following I/O Interfaces:

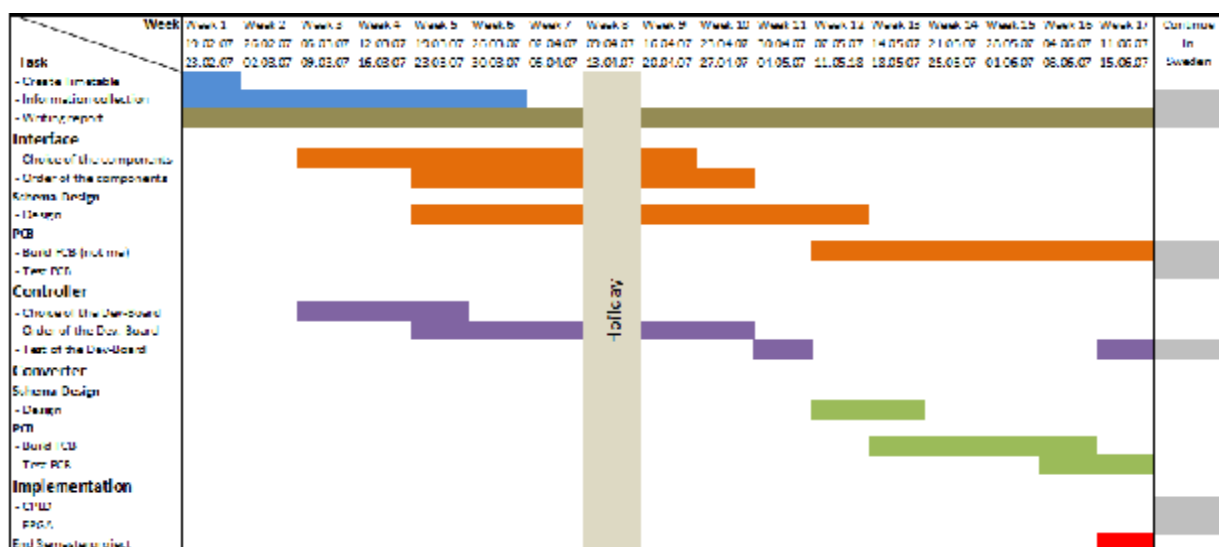
- 10/100Base-TX Copper Ethernet RJ45
- Fiber Ethernet 100Base-FX
- RS 422 for time synchronization
- LCD display
- USB 2.0 Interface
- SD-Card-Reader
- JTag Interface
- Led's

### 2.2 Appendix

Some articles and parts of the work are given in appendix. They can be found at the end of the report. The appendixes 2-10 are just given on the enclosed CD.

### 2.3 Structure of this project

#### 2.3.1 Planification



C Fig. 2.1 Time table diagram of the Project

### 3 Overview

The goal is to build a hardware, called *Measurement Point (MP)*. Such a device does packet capturing, then the captured data will be filtered and a time stamp will be added on each packet. The new packet will be buffered. After all, this packet will be sending in data packets in a measurement frame with defined sizes. I will realize this MP with a FPGA, which can implement a microprocessor as an IP soft-core or a hard-wired-core.

The Ethernet input connections are:

- 10/100 Copper Ethernet RJ45
- fiber Ethernet 100Base-FX

Because a MP is a passive device to the Ethernet input, it must be invisible to the Measurement Area Network (MArN). To do a serious time stamp, an exact time synchronization is needed. Therefore a RS422 Interface is added to synchronize with a master's clock.

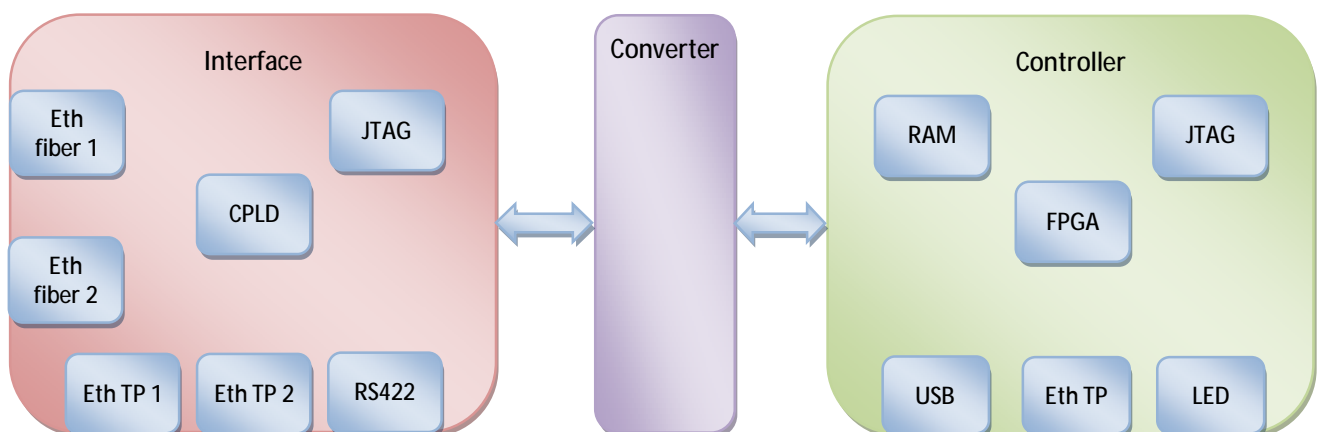
The captured data will also be sent with a 10/100 Copper Ethernet RJ45. The programming interface of the board will be done via JTAG interface and it is possible to store the program in the FPGA as well as in a Smardcard-reader (SD), which is also integrated in the board. Other additional I/O's for further work are:

- USB Interface
- LCD Display
- Led's
- Optional optical input

It should also be possible to exchange the Interface with a different one, so that the input sources can be enlarged again, for example, to implement Giga Ethernet, or other Transmission mediums. For this project, it is sufficient to have a 10/100Base –TX or a FX support.

The functional specifications were discussed with Mr F.Corthay and O.Gubler. The meeting documents are given in the appendix.

#### C Appendix 1 Meeting document



## 4 Hardware

The Hardware is divided into three main parts:

- Interface to the MArN
- Converter between Interface and Controller
- Controller to generate the captured packets

### 4.1 Problem analysis and solution approaches

The tasks of the MP to be developed can, be summarized in the following central problems.

Interface:

- Passive Ethernet tap
- Active Ethernet tap
- Sending the Data from plug to the Physical Layer (Phy)
- Physical Interface for 100Base-TX, 10Base-T and 100BaseFX
- Sending of the packets to the Controller
- RS422-Interface
- Oscillator
- Connector

Converter:

- Connector

Controller:

- FPGA
- RAM module
- Power over Ethernet
- Synchronization with a masters-clock
- JTag Interface to store in SD-Card or directly in the FPGA
- Additional I/O Interfaces

These points must be analyzed, so that they can be converted in practicable solutions.

### 4.2 Interface

This part of the hardware submits the captured packages of the MArN-Network to the Controller, without changements of the content. The network entrance can be carried out via a 100Base-TX twisted pair, a 10Base-T twisted pair or a 100Base-FX fiber.

This Interface can be used for the following tasks:

- Active tap : Both input-lines go directly to the physical Interface (Phy) without any influence.
- Passive tap: Both Rx-Channels are connected through the high-impedance tap to the Phy, because we are just listening, we don't need the Tx-Channels

In the active case it's possible to attach either the twisted pair (TP) or the fiber. There are the following possibilities:

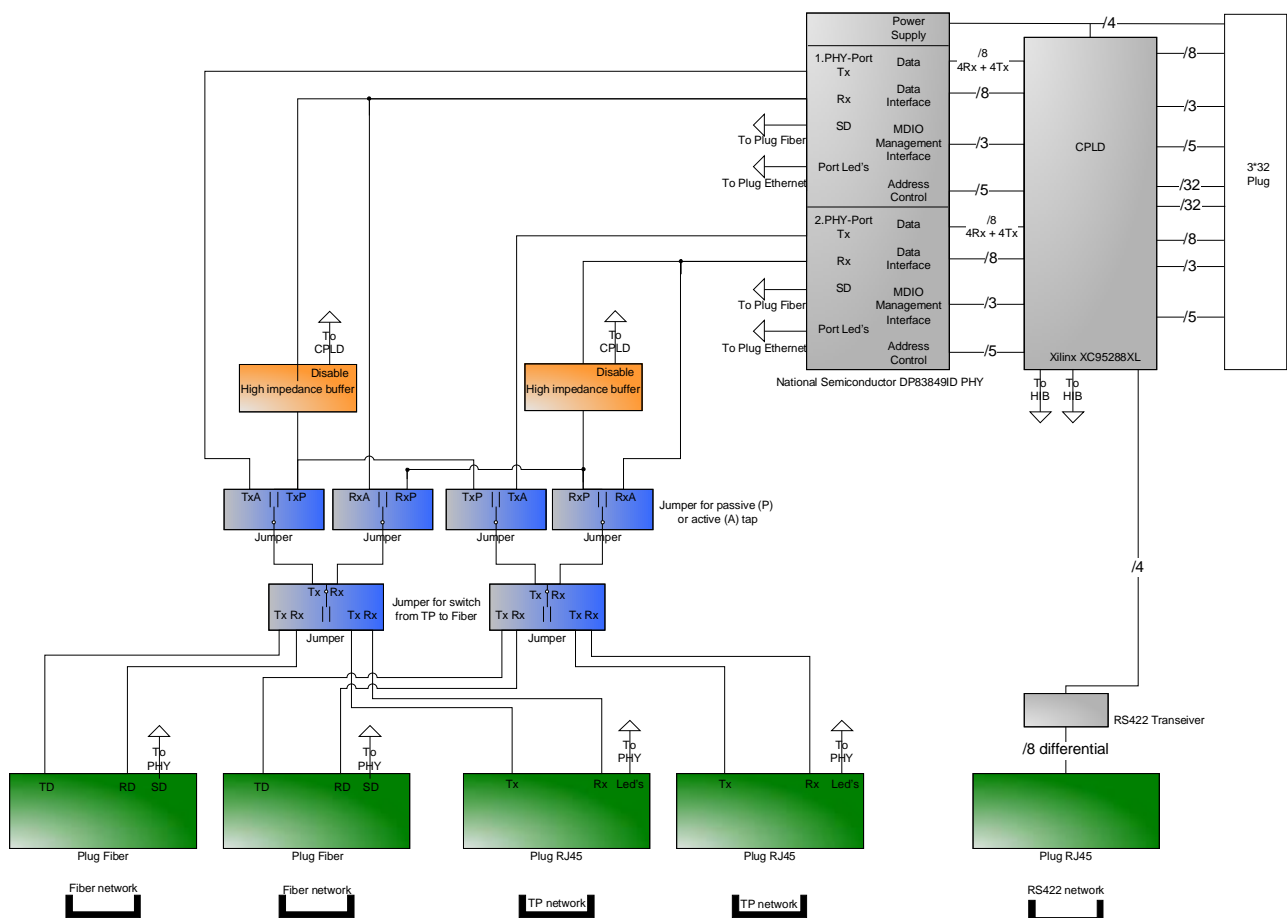
- TP  $\rightarrow$  TP: Both twisted pair inputs are used
- TP  $\rightarrow$  fiber: One twisted pair and one fiber are used (TP – FX conversion)
- fiber  $\rightarrow$  fiber: Both fiber inputs are used

In the passive case it's only possible to attach the twisted pair. It is not possible to listen complete passively to the fiber-network.

- **TP to TP:** Both twisted pair inputs are used

With Jumpers it is possible to switch the input-line between the RJ45 and the optical fiber input, as well as for the switch from the active to the passive tap.

The routing of the conducting paths was realized by a specialist of the school. The construction of the printed circuit board (PCB) couldn't be realized at school because the board owns 4 different layers. Therefore, the production of the PCB was passed on to an external manufacturer by a quantity of 1. Unfortunately, at the end of the semesterproject, the board wasn't completed, therefore it was impossible to mount the components on the board and to test it.



C Fig. 4.1 Bloc diagram of the Interface

Remark: We have moved the RS422 Interface for the synchronization with the master's clock of the Controller board to the Interface, so that we have the possibility to attach a controller without a RS422-Interface. Because the time for the completion of the Controller is not sufficient, an Actel Development-Board without a RS422 Interface is used as Controller (C see 4.3 Controller).

### 4.2.1 Active Ethernet tap

The active Ethernet tap gives us the possibility to use the MP also as converter between the TP and the optical fiber. This is also very useful to "turn off" the MP and to connect the 2 MarN lines with each other. Through this, no cables have to be changed. The Rx and Tx lines go directly to the Phy without going through the high-impedance buffer. Of course, we can also put TP to TP or fiber to fiber together. For this feature, we have to send the packets to the CPLD by the Phy and mix up Rx to Tx there. After, the packets will be sent back into the other Phy-port again.

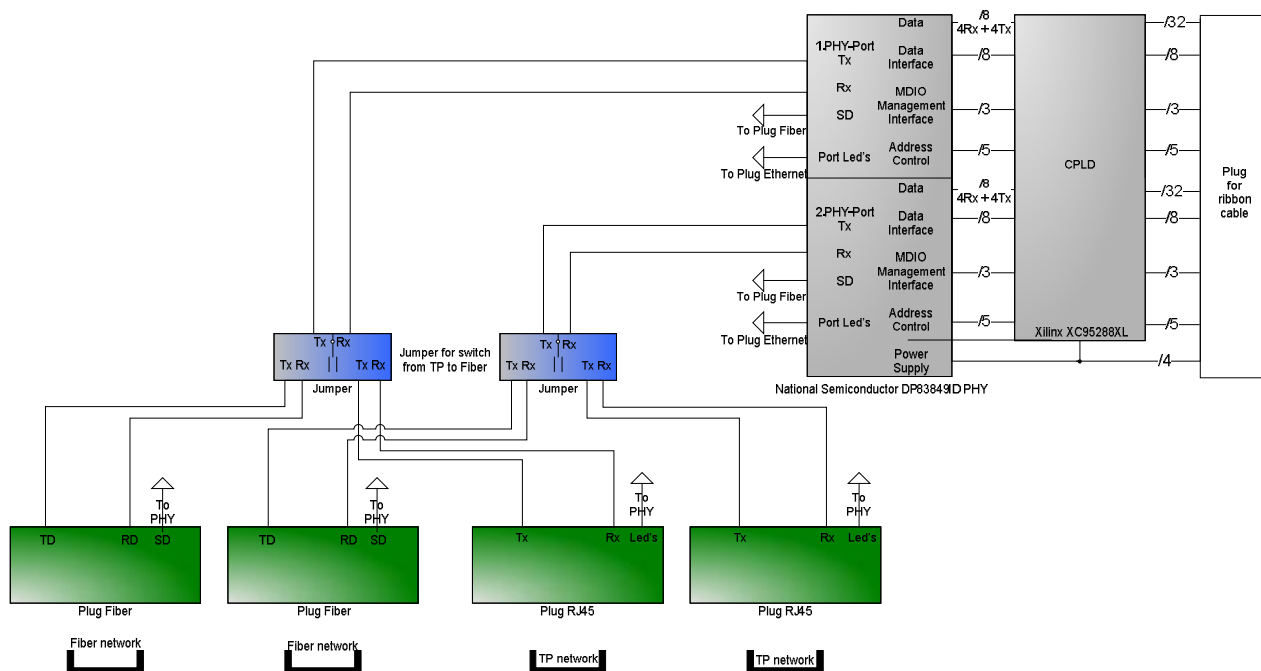


Fig. 4.2 Active tap bloc diagram of the Interface

### 4.2.2 Passive Ethernet tap

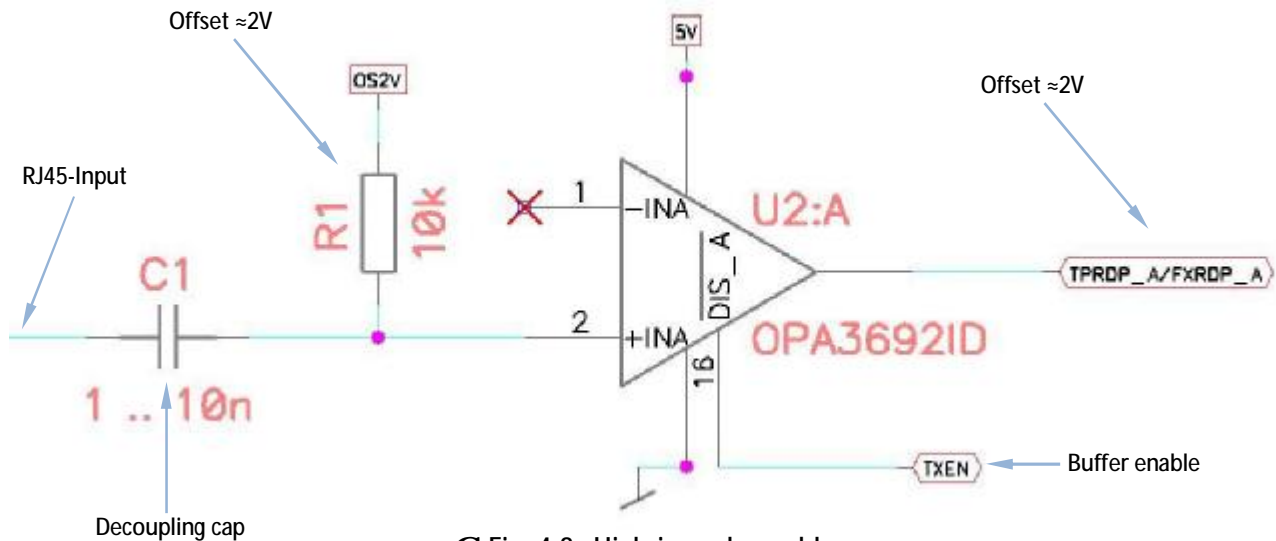
To capture the data traffic on an Ethernet connection, the signals must be tapped off by the lines. Not to influence the line and the connection to the next device, the tap has to be carried out with high-impedance. So the impedance of the line is not changed fundamentally. A high-impedance buffer is added between the RJ45 plug and the Phy (see Fig. 4.3 High impedance Buffer).

This problem was already solved in an earlier diploma work and will serve us as a template. (Decoding of 100Base-TX signals).

They have used a Voltage follower, so that the seen impedance from the MarN-Network is very high and therefore, we won't disturb it. In reality, we still have some capacitive disturbances from the lines and from the operation amplifier (OP)-input ( $\approx 2\text{pF}$ ), in addition, the OP has an input resistor of  $100\text{k}\Omega$  and a resistor to set an operating point.

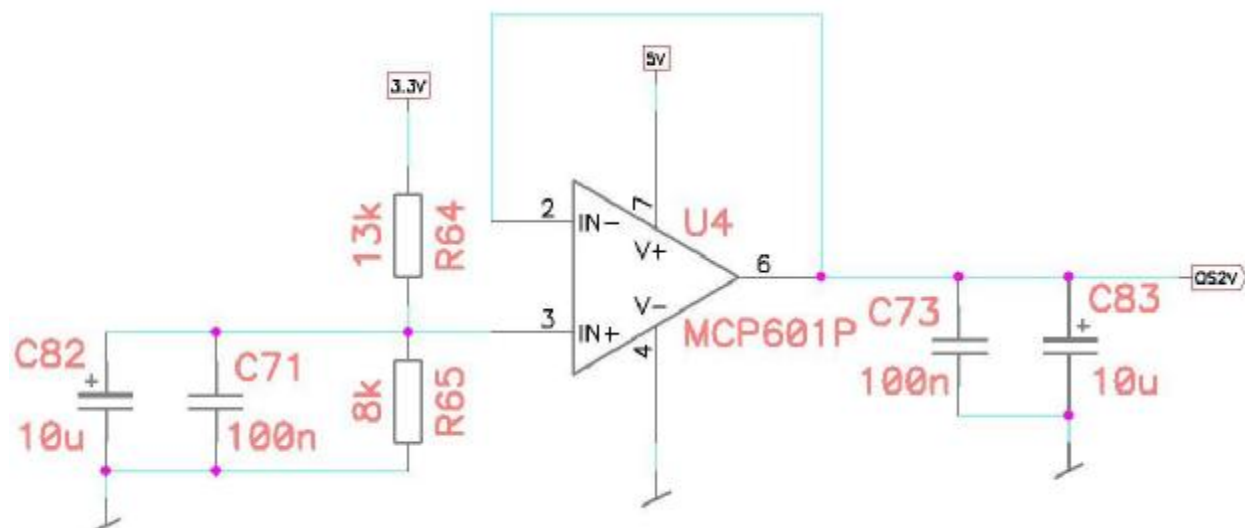
The lines to the OP have no DC-offset of  $1.65\text{V}$  ( $\frac{1}{2}$ ), the decoupling will be done by a capacitor, therefore, a resistor of  $10\text{k}\Omega$  adds a DC-offset of  $2.05\text{V}$  to the line. These resistors should, however, influence the line only insignificantly opposite the capacities.

The OP, which is used for the DC-offset, is internally already connected with resistors, with which reinforcements of 1, 2, -1 can be caused. We only use them as buffer with reinforcements of 1. Since the amplifiers as RGB drivers are conceived, they can deliver sufficient current and reject a frequency range of over  $200\text{MHz}$ . Moreover, they can be activated or deactivated with an additional line (TXEN) (see Fig. 4.4 Offset circuit).



C Fig. 4.3 High-impedance-bloc

C Appendix 2 Datasheet OPA3692ID (only on CD)

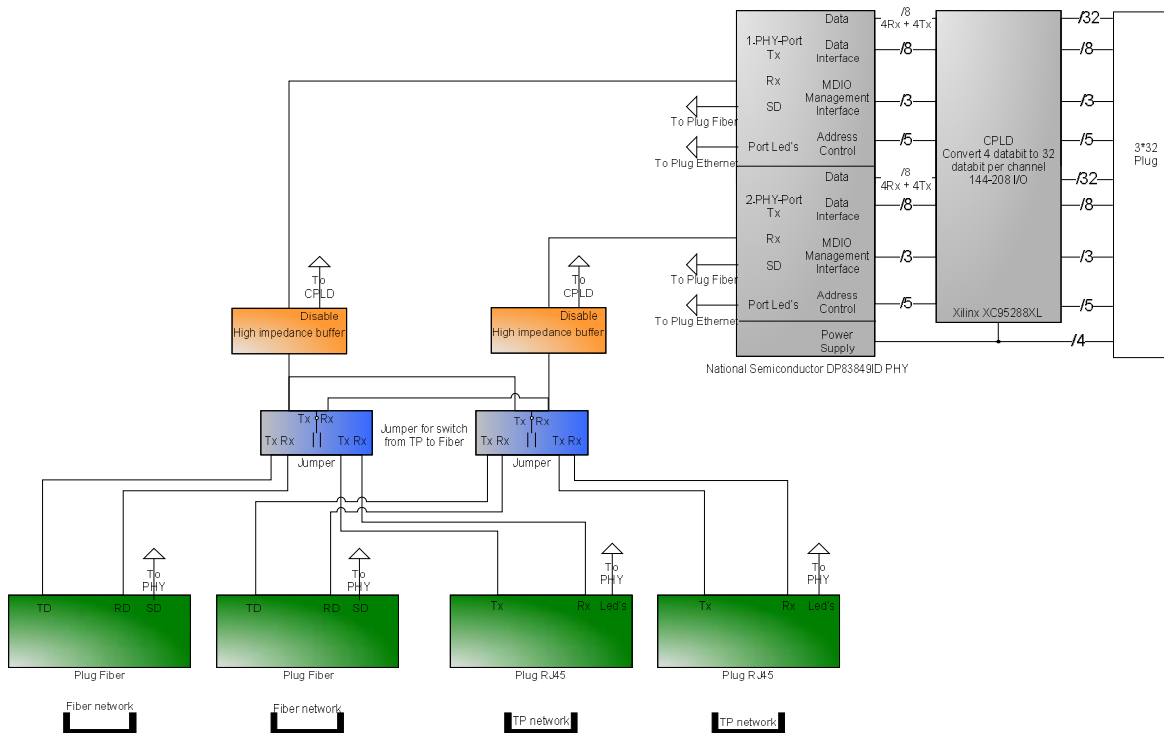


C Fig. 4.4 Offset circuit

Because the MP only listens to the Ethernet, it knows no difference between the Rx and the Tx line. To be able to select the line to be measured, the physical interface must be able to capture both lines at the same time or change between the two lines.

If we judge both solutions, it is obvious, that a 2-port Phy is the best choice. Because the MP has 2 independent passive Ethernet taps disposes, both lines can be captured when required. Fortunately, there already exist some good Phys on the market, like the National Semiconductor DP83849ID Transceiver, which offer a 2-Port Phy with fiber and twisted-pair support.

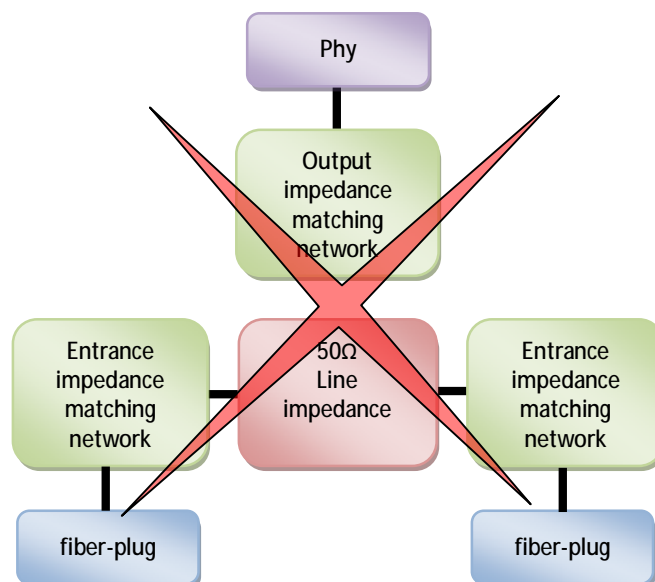




C Fig. 4.5 Passive tap bloc diagram of the Interface

Because the fiber-plugs do already transform the light pulses into electrical signals, we cannot listen passively to the fiber-network. The lines between the connector and the Phy have their own impedance network: The signals first go through an entrance impedance, match the network near the plug and after, through an output impedance and match the network near the Phy. The transmission lines between the 2 networks have an impedance of  $50\Omega$ .

The passive tap of the fiber-network doesn't work, because the 2 fiber-plugs are connected through 2 entrance impedance matching networks. The following figure illustrates this case.



C Fig. 4.6 Worse passive tap of the fiber-network

### 4.2.3 Sending the Data from the plug to the Phy

Between the RJ45 and the fiber-plugs, the packets are still available in an analogous form. It can cause some disturbances or reflections on the line, therefore, we have to add a termination circuit, which includes an entrance- and output-resistances-circuit and in addition, the lines can have a line impedance of  $50\Omega$  in case of the fiber termination circuit. Hereinafter I would like to explain the sending for the 2 possible entrance inputs.

#### Ethernet RJ45-plug

In the following figure, you can see the required resistors. We also need two RJ45 plugs with a 1:1 transformer to achieve a DC-decoupling. RJ45 plugs, with integrated magnetic, are available at school and fit perfectly to this application.

In addition, it still has to be mentioned that these RJ45 plugs have some short-circuited pin's (4, 5, 7, 8) and are directly connected to the mass.

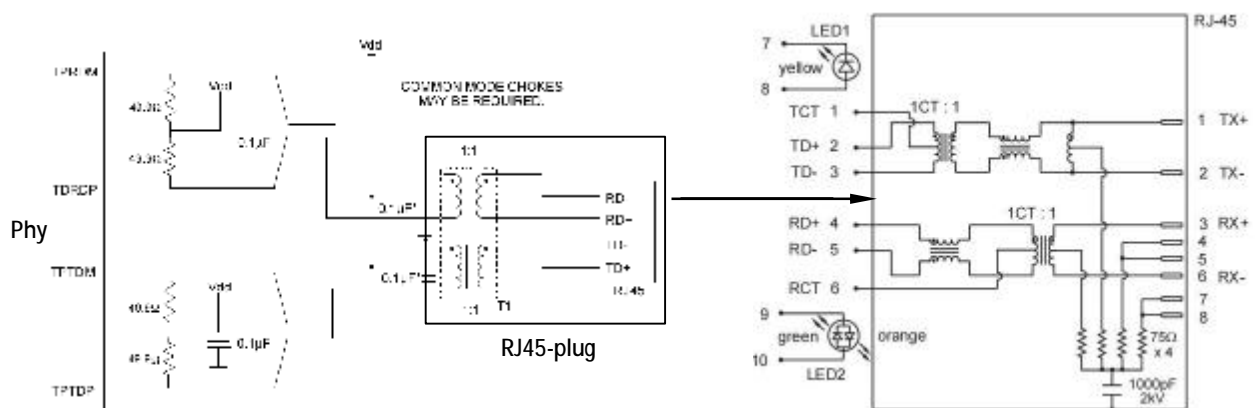


Fig. 4.7 Termination circuit for RJ45 and construction of the RJ45-plug

#### Appendix 3 Datasheet MagJack 0801-1X1T-03 (only on CD)

#### Ethernet fiber-plug

For the fiber connection we must have a termination circuit. That means, entrance-, output-resistors and a line impedance of  $50\Omega$ . The line is therefore well-defined and no disturbances can appear on it.



Fig. 4.8 Termination circuit for fiber-plug

Because the fiber plug is designed for future diploma works, we will neither order nor put together these plugs at the board.

#### Appendix 4 Datasheet Agilent AFBR-5903Z (only on CD)

Calculation of the  $50\Omega$  line-impedance:

The Calculation of the impedance of the line can be done via the program ZTool (see CD).

In the following figure, you can see an example of  $50\Omega$  line impedance.

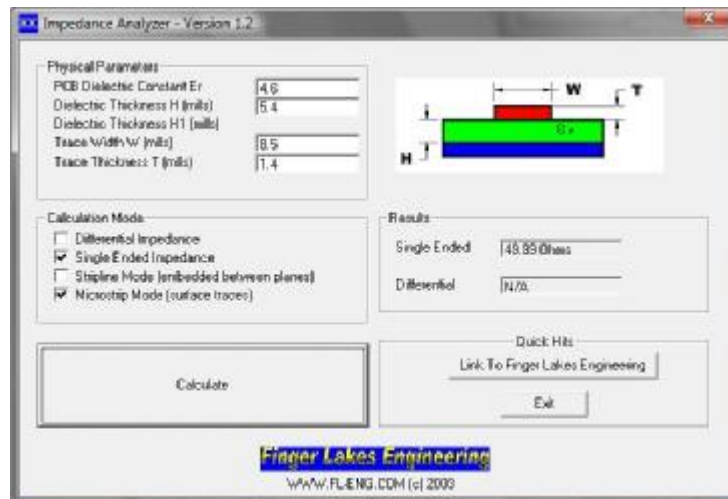


Fig. 4.9 ZTool to calculate the line impedance

National Semiconductor offers a scheme of a Development board that exactly is adapted to the Phy. On this scheme, neither the fiber-network nor the TP-network has a line impedance of  $50\Omega$ . Because this board is done by specialists, we decided to take it as a template and we didn't adapt the line with  $50\Omega$ . However, we had to place the Phy as close as possible to the plugs, to minimize the reflections on the lines between the plug and the Phy.

Appendix 5 DP83849 Dual Phyter Demo II Stump Jumper scheme (only on CD)

#### 4.2.4 Physical Interface

The decoding of the physical layer of the 100Base-TX and also of the 100Base-FX signals are very complex. With a 100Base-TX hardware the raw bits go through a 4B5B binary encoding to generate a series of 0 and 1 bits, clocked at 125 MHz; the 4B5B encoding provides DC equalization and spectrum shaping. Just as in the 100Base-FX case, the bits are then transferred to the physical medium attachment layer, using NRZ encoding. However, the 100Base-TX introduces an additional, medium dependent sublayer, which employs MLT-3 as a final encoding of the data stream before transmission.

But as already mentioned above, there exist IC's, which offer these functionalities. The chosen 2-Port Phy is the National Semiconductor DP83849ID. It offers a Dual Port 10/100Mbit/s Ethernet Layer Transceiver with FX support. One of the reasons why I chose this Phy, is, that mostly the packing unit is up to 400 pieces per order. But by National Semiconductor, there is the possibility to order samples in a quantity of 1-5 pieces. I would like to thank to National Semiconductor for giving me 4 examples of this Phy for free to use in this project.

This is more than sufficient for this work.

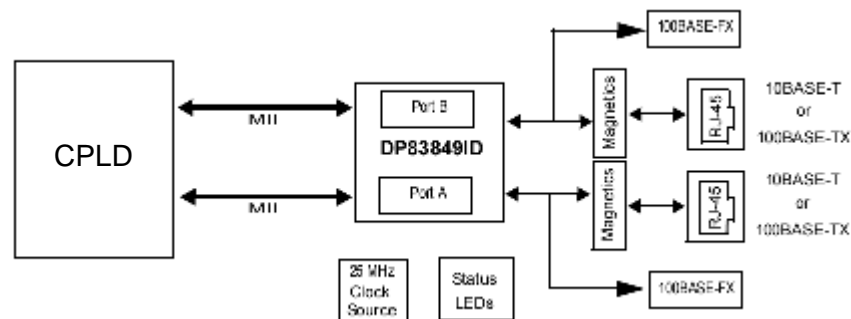


Fig. 4.10 Application of the Phy

#### Appendix 6 Datasheet National Semiconductor DP83849ID (only on CD)

##### 4.2.4.1 Strap options

The Phy uses many of the functional pins as strap options. The values of these pins are sampled during reset and used to strap the device into specific modes of operation.

A 2.2kΩ resistor should be used to pull-down or pull-up to change the default strap option.

All strap options can also be set through a register access.

In the following part I will give you an overview of the possible options:

- Mode options: With the 4 pins (FX\_EN, AN\_EN, AN1 and AN0) we choose the mode of our Phy for port 1 and 2.
  - FX\_EN → fiber enable
  - AN\_EN → Auto-Negotiation enable
  - AN1/AN0 → Control the forced or advertised operating mode

The default is 0111 since FX\_EN pin has an internal pull-down and the Auto-Negotiation pins have internal pull-ups.

To choose the different options, jumpers are added to these pins to force 0 or 1.

So we have 3 configuration possibilities (see in following table).

| FX_EN | AN_EN | AN1 | AN0 | Forced Mode                                                |
|-------|-------|-----|-----|------------------------------------------------------------|
| 0     | 0     | 0   | 0   | 10BASE-T, Half-Duplex                                      |
| 0     | 0     | 0   | 1   | 10BASE-T, Full-Duplex                                      |
| 0     | 0     | 1   | 0   | 100BASE-TX, Half-Duplex                                    |
| 0     | 0     | 1   | 1   | 100BASE-TX, Full-Duplex                                    |
| 1     | X     | X   | 0   | 100BASE-FX, Half-Duplex                                    |
| 1     | X     | X   | 1   | 100BASE-FX, Full-Duplex                                    |
| FX_EN | AN_EN | AN1 | AN0 | Advertised Mode                                            |
| 0     | 1     | 0   | 0   | 10BASE-T, Half/Full-Duplex                                 |
| 0     | 1     | 0   | 1   | 100BASE-TX, Half/Full-Duplex                               |
| 0     | 1     | 1   | 0   | 10BASE-T Half-Duplex<br>100BASE-TX, Half-Duplex            |
| 0     | 1     | 1   | 1   | 10BASE-T, Half/Full-Duplex<br>100BASE-TX, Half/Full-Duplex |

C Fig. 4.11 Table of mode options

- MAC Interface mode: With the 2 pins (MII\_MODE and SNI\_MODE) we determines the operating mode of the MAC Data Interface
  - MII\_MODE à disable MMI mode
  - SNI\_MODE à enable SNI mode

The default operation is 0 (MII mode) because the MII\_MODE pin has an internal pull-down resistor

| MII_MODE | SNI_MODE | MAC Interface Mode |
|----------|----------|--------------------|
| 0        | X        | MII Mode           |
| 1        | 0        | RMII Mode          |
| 1        | 1        | 10 Mb SNI Mode     |

C Fig. 4.12 Table of MAC Interface options

- LED configuration: With the 2 pins (CRS\_DV and CRS) several functions can be multiplexed onto the three LED's using three different modes of operation. In our case we don't need the LED\_SPEED

The default mode is "mode1" the LED\_CFG[1] register is only controllable through the register access and cannot be set by a strap pin.

| Mode | LED_CFG[1] | LED_CFG[0] | LED_LINK                               | LED_SPEED                        | LED_ACT/LED_COL                           |
|------|------------|------------|----------------------------------------|----------------------------------|-------------------------------------------|
| 1    | don't care | 1          | ON for Good Link<br>OFF for No Link    | ON in 100 Mb/s<br>OFF in 10 Mb/s | ON for Activity<br>OFF for No Activity    |
| 2    | 0          | 0          | ON for Good Link<br>BLINK for Activity | ON in 100 Mb/s<br>OFF in 10 Mb/s | ON for Collision<br>OFF for No Collision  |
| 3    | 1          | 0          | ON for Good Link<br>BLINK for Activity | ON in 100 Mb/s<br>OFF in 10 Mb/s | ON for Full Duplex<br>OFF for Half Duplex |

C Fig. 4.13 Table of Led mode select

#### 4.2.5 Sending of the packets to the Controller

One of the problems of the ancient Diploma work was, that the sending of the packets from the Interface to the Controller Board caused some errors in the CRC-Check. Because of this, some frames were lost. To prevent the lost of a package, we have to reduce the speed of the data between the Interface and the Controller.

The maximal speed of a 1000Base-TX Ethernet on the 4 Phy-Output lines is:

$$F_{max} = \frac{D}{nbrOfLines} \approx \frac{1Gb/s}{4lines} = 250MHz$$

We have chosen to use 32 lines/channel, this gives a maximal speed of:

$$F_{max} = \frac{D}{nbrOfLines} \approx \frac{1Gb/s}{32lines} = 31.25MHz$$

For the assignment about a ribbon cable or hard plug, 31.25MHz does not represent a problem.

Therefore, we add a complex programmable logic device (CPLD), which has the following tasks:

- Enlarges the number of data lines
- Adapts the control signals
- Realizes the transformation TPB à fiber
- Disables the high-impedance buffer
- Transmits the RS422 master's clock to the Controller

##### 4.2.5.1 Choice of the CPLD

The choice of the CPLD is carried out with the required pin. In the following table, all needed I/O are listed with the number of pin required:

| Pin from Interface              |                |           |
|---------------------------------|----------------|-----------|
| Description                     | Number of pins | Direction |
| Rx-Data lines<br>(Port A and B) | 8              | Input     |
| Tx-Data lines<br>(Port A and B) | 8              | Input     |
| Control-signals                 | 18             | I/O       |
| RS422 Interface                 | 4              | I/O       |
| High-impedance buffer disable   | 2              | Output    |
| Pin to Controller               |                |           |
| Description                     | Number of pins | Direction |
| Rx-Data lines                   | 64             | Output    |
| Control-signals                 | 18             | I/O       |
| RS422 Interface                 | 4              | I/O       |
| High-impedance buffer disable   | 2              | Input     |
| Reserve                         | 5              | I/O       |
| Total Nbr of Pins               | 129            |           |

C Fig. 4.14 CPLD-Pin table

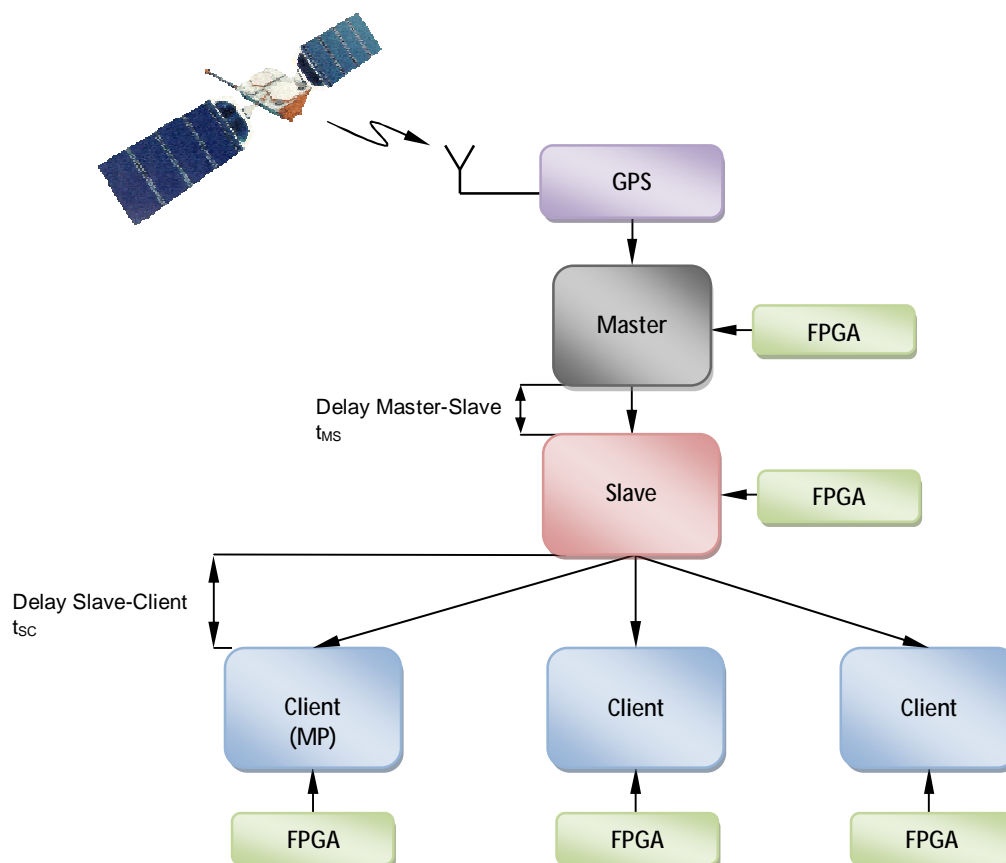
With this table it is obvious that we need a CPLD with 129 pin supported. We considered to use the Xilinx XC9500XL family. This CPLD was already used often and it is a low energy device with up to 5V I/O capabilities. The numbers of the available user I/O can be selected between 34 – 196 Pin's. In this case we took the CPLD with 168 User pin in a PQ208 package.

C Appendix 7 Datasheet Xilinx XC95288XL (only on CD)

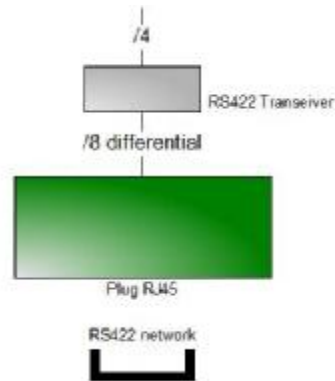
#### 4.2.6 RS422 Interface

The RS422 Interface is needed for exact time synchronization with a master's clock and for the time stamp, which we add in each captured packet. This problem has already been solved in an earlier diploma work (C Time and frequency synchronization). The MP made in this project, is used as client for the RS422 Network, this is obvious in the following picture.

The slave calculates the delay from master to slave ( $t_{MS}$ ) and add this to the time, which was sent by the master. The client will now calculate the delay from slave to itself ( $t_{SC}$ ) and adds this to the time, that was sent by the slave too. After this process, the result has the exact time and can be used further for the time stamp of the packets in the Controller.



C Fig. 4.15 RS422 network overview



C Fig. 4.16 Bloc diagram of the RS422 Interface

### 4.2.7 Connector

The decision, which connector we take, depends on the number of required pins. In our case, it is better to install a hard Plug and not to work with a ribbon cable. One of the advantages is, that the disturbances will be minimized. At the completion of all parts (Interface and Controller), the components should be connected fix through the connector without using a Converter-board.

We decided to take a 96 pin (3\*32) male-female connector.

Which signals are connected to the Connector is obvious in the following table

| Pin from Interface              |                |           |
|---------------------------------|----------------|-----------|
| Description                     | Number of pins | Direction |
| Rx-Data lines<br>(Port A and B) | 64             | Input     |
| Control-signals                 | 18             | I/O       |
| RS422 Interface                 | 4              | I/O       |
| High-impedance buffer disable   | 2              | Output    |
| Power signals                   | 3              | Input     |
| Reserved                        | 5              | -         |
| Total Nbr of Pins               | 96             |           |

C Fig. 4.17 CPLD-Pin table



### 4.3 Controller

The Controller part will receive all the Rx packets. These packets will be filtered and a time stamp will be added. After this, the packets will be reorganized and saved in RAM-Modules. To send the filtered packets to a Computer, a Ethernet connection is used. On the Computer we have a DPMI-Interface which is developed by Arloos Patrik, this interface will analyse the captured packets who was sended by the Controller. This board contains the additional I/O interfaces for the further diploma works.

Unfortunately, the Controller can't be realized in this semester project, because time is restricted and not sufficient.

It is inevitable, to take an already prefabricated development board, which is used as Controller.

The development board should contain the following features:

- Connector to the Interface
- RJ45-Ethernet plug
- Power supply (also for the interface)
- RAM-modules
- FPGA which can implement a IP soft-core or hard-wired-core
- Other additional Interfaces

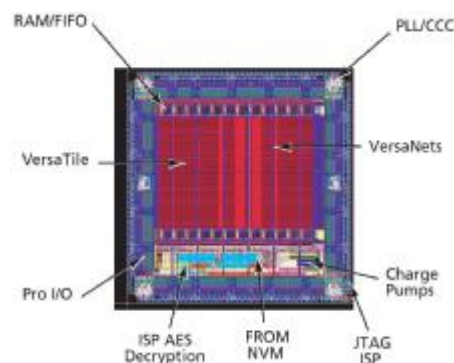
#### 4.3.1 Choice of the FPGA

First of all, it is necessary to choose the FPGA to use in the Controller. The features that can be implemented are a microprocessor, such as an IP soft-core or a hard-wired-core. It should also support enough I/O's for all the interface signals. In addition, the same FPGA should also be available as low power FPGA and as one time programmable (OTP). The following table shows the chosen possibilities:

| Items  |                | Integrated Prozessor                                                        |                                    | FPGA                                                                    | Licence                             | Code / Example                                                                   | Avail. | DevBoard                                                                                                                                                                       | DevTools                                                                                          |
|--------|----------------|-----------------------------------------------------------------------------|------------------------------------|-------------------------------------------------------------------------|-------------------------------------|----------------------------------------------------------------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
|        |                | SoftCore                                                                    | HardCore                           |                                                                         |                                     |                                                                                  |        |                                                                                                                                                                                |                                                                                                   |
| device | Xilinx Spartan | - 32-bit MicroBlaze™ soft processor<br>It use 800-2600 LUTs                 | No                                 | - Spartan 3 XC3S1000-4FG456C \$60<br>- Spartan 3E XC3S100E-4CP132C \$30 | SoftCore needs EDK                  | - There exist some example<br>- Microblaze codes are available                   | Yes    | - Spartan-3A Starter Kit (HW-SPAR3A-SK-UNI-G) \$199<br>- Spartan-3E 1600E MicroBlaze Development Kit (DO-SP3E1600E-DK-UNI-G) \$599                                             | - ISE™ Foundation DS-ISE-FND \$2500<br>- The Embedded Dev Kit (EDK)<br>- Real View Dev Kit \$2000 |
|        | Xilinx Virtex  |                                                                             | Embedded PowerPC 405 (PPC405) core | Virtex-4 FX FX not available                                            | HardCore Free<br>SoftCore needs EDK |                                                                                  | No     | PowerPC & Microblaze Starter Kit DO-ML403-EDK-ISE-USB-EC \$895                                                                                                                 |                                                                                                   |
|        | Actel MP7      |                                                                             | No                                 | The Virtex™-II Pro                                                      | HardCore Free<br>SoftCore need EDK  |                                                                                  | Yes    | PowerPC StarterKit XC2VP20-5FF896C \$360                                                                                                                                       |                                                                                                   |
|        |                | - MP7 soft core, impl. ARM7TDMI-S.<br>Available for all M7 devices for free |                                    | - M7 ProASIC3 \$30-\$150<br>- M7 Fusion<br>- M7 IGL00                   | Free for M7 devices                 | - 1 example<br># Core MP7 - WebServer<br>- SoftCore codes are available for free | Yes    | The CoreMP7-1000 are available on May COREMP7-1000-DEV-KIT-FP3 \$600 with Programmer FP3 (\$100).<br># Fusion Starter Kit<br># ProASIC3 Starter Kit<br># ProASIC3E Starter Kit | - Design Software Libero® DIE Free<br>- CoreConsole for configuration<br>- SoftConsole free       |

C Fig. 4.18 FPGA choice table

In this Project, I decided to take the Actel MP7 FPGA. In this FPGA we have the possibility to implement one (or more) ARM7TDMI-S microprocessor as IP soft-core. The ARM7 Core is the most widely used architecture in 32-Bit microprocessors. We chose the Actel Development board with a MP7A3P1000 FPGA chip. With this chip, we have sufficient user I/O's for our Interface board.



C Fig. 4.19 MP7 FPGA-Core

C Appendix 8 Datasheet ProASIC3E Flash Family (only on CD)

The board consists of the following:

- Wall-mount power supply connector with switch and LED indicator
- Switches to select from 1.5 V, 2.5 V, and 3.3 V (I/O Bank) voltages on banks 3–4
- 10-pin, 0.1"-pitch programming connector compatible with Altera connections
- 48 MHz oscillator and 32kHz oscillator for real-time clock (RTC) calculations
- Eight LEDs driven by outputs from the device
- Jumpers allowing disconnection of all external circuitry from the FPGA
- One monostable pulse generator switch
- Eight switches providing input to the device
- Two RS-232 serial interfaces
- Two 10/100 ethernet interfaces
- One Controller Area Network (CAN) 2.0B serial interface
- One USB 1.1 serial interface

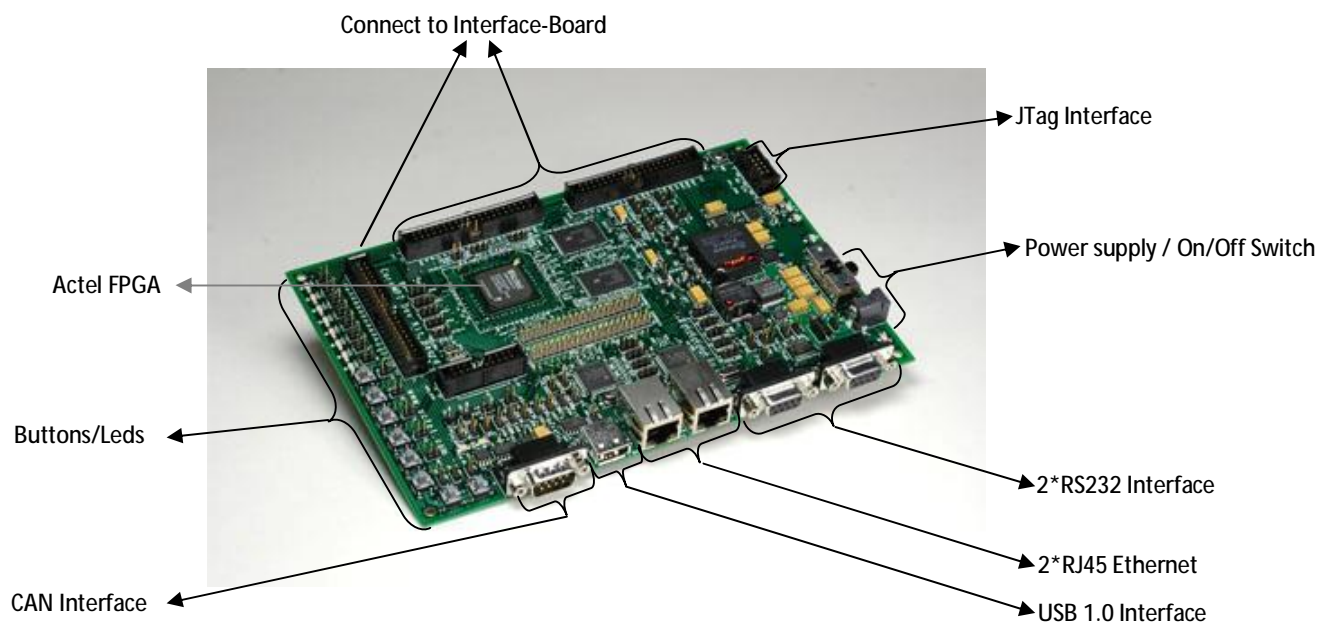
#### C Appendix 9 CoreMP7 development kit users guide (only on CD)

CoreMP7 is a soft IP-core implementation of the popular ARM7TDMI-S microprocessor. The CoreMP7 microprocessor has the following features:

- 32-bit ARM instruction set for maximum performance and flexibility
- 16-bit Thumb instruction set for increase code density
- Inified bus interface
- 3-stage pipeline
- 32-bit ALU
- Fully static operation

#### C Appendix 10 Datasheet CoreMP7 (only on CD)

Remark: A reason why I chose this Dev-Board is, that the FPGA works with a 50MHz oscillator. The packets leave the Phy with a speed of 25Mhz, so we have a certain surplus and can work without problems on the data in the FPGA, otherwise, we could have an anti-aliasing effect.

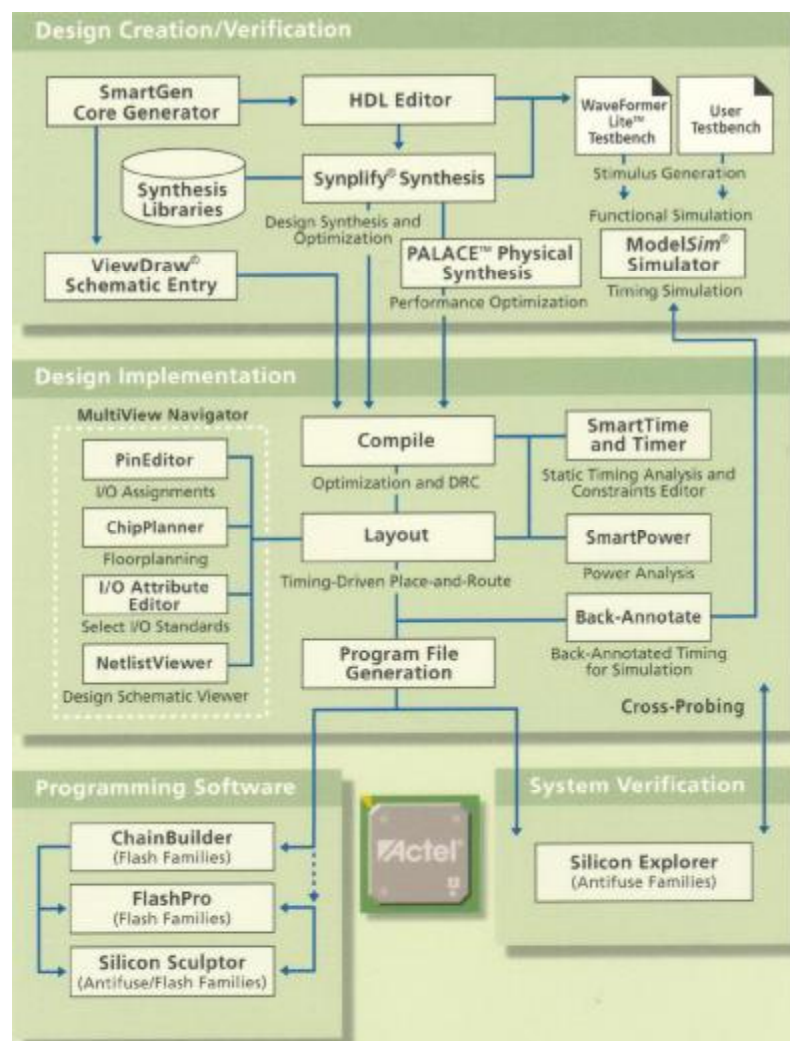


C Fig. 4.20 Actel CoreMP7 Development Kit

### 4.3.2 Design flow

In the following picture you can see the Design flow of a program, the individual evolutionary steps can be subdivided into 3 groups:

- Design creation/verification  
In this step, the VHDL code will be generated (HDL-Editor) and simulated (ModelSim). The VHDL code of the MP7-Core will also be generated out of the SmartGen Core Generator. After this, the synthesis will be made with PALACE.
- Design implementation  
In this step, the code will be compiled and to conclude, the program files will be generated.
- Programming software  
The FlashPro 3 programmer will now program the FPGA with the generated program files.



C Fig. 4.21 Schema of the design flow

In the last week of the project, I began to program the dev-board with a tutorial project, which was given with the development kit. I tried to implement the ARM7TDMI-S IP soft-core in the FPGA. During the step "place & route", I had a compile error which I couldn't eliminate until the end of the semesterproject.

Error: CMP073 power found on external GND pin NET: 'GND'

I have contacted Actel, to inform about this error in the tutorial project. Unfortunately, my request was not solved till now.

## 4.4 Converter-board

We need a Converter-board so that our Interface-board is able to connect to the Controller-board.

This converter has on one side a 96Pin connector for the connection with the Interface and on the other side three 40Pin connectors to attach the Controller. The connectors on the Controller-board also offer several powerlines, like 5V, 3.3V and GND, therefore, we don't need the external 5V connector in the Interface, the power can be directly taken from the Controller.

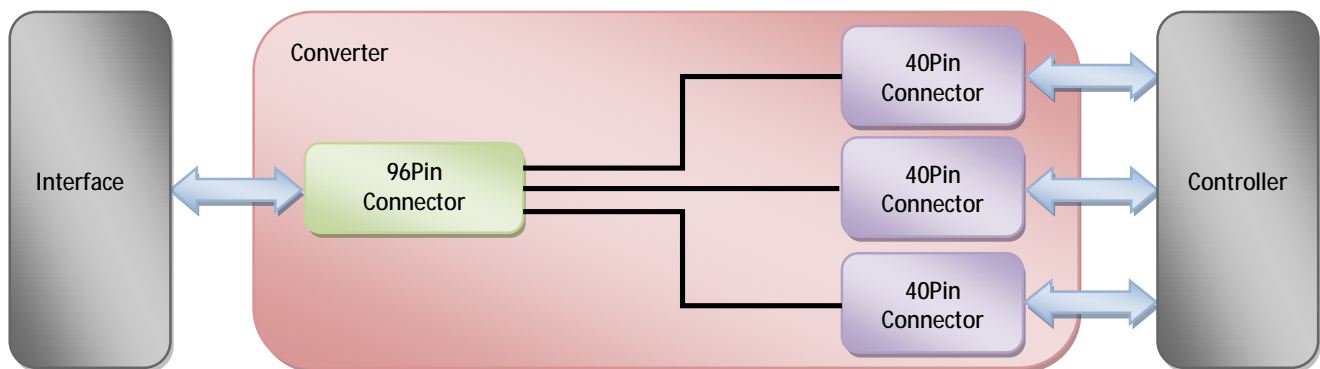


Fig. 4.22 Blocdiagramm of Converter-board

### 4.4.1 Pin assignment on the Controller

With the help of the Converter-board, the signals of the Interface, which belong together, can be summarized on one or several I/O Banks of the Controller. Later, the different signals can simply be accessed and mistakes are avoided.

The pin assignment of the different signals are shown in the following table

| Pin from Interface                       |                |                   |                                                       |
|------------------------------------------|----------------|-------------------|-------------------------------------------------------|
| Signal description                       | Number of pins | Pin of the Header | I/O Bank of the Controller                            |
| Header J11 (control)                     |                |                   |                                                       |
| RES 1-5                                  | 5              | [4..8]            | Bank 0 [3..5] & Bank 1 [0..1]                         |
| RS422 Interface                          | 4              | [11..14]          | Bank 1 [4..5] & Bank 2 [0..1]                         |
| Control signals of port B                | 8              | [17..24]          | Bank 2 [4..6] & Bank 3 [0..4]                         |
| Control signals of port A                | 8              | [27..34]          | Bank 3 [7..11] & Bank 4 [0..2]                        |
| MDIO Interface                           | 2              | [37..38]          | Bank 4 [5..6]                                         |
| High impedance buffer disable            | 2              | [39..40]          | Bank 4 [7..8]                                         |
| Header J12 (data port B)                 |                |                   |                                                       |
| Data signals of port B                   | 32             | [1..32]           | Bank 4 [9..23] &<br>Bank 5 [0..5] &<br>Bank 6 [0..10] |
| Header J13 (data port A & power signals) |                |                   |                                                       |
| Data signals of port A                   | 32             | [5..36]           | Bank 7 [0..31]                                        |
| 5V                                       | 1              | [38]              | -                                                     |
| 3.3V                                     | 1              | [37]              | -                                                     |
| GND                                      | 1              | [39..40]          | -                                                     |
| Total Nbr of Pins                        | 129            |                   |                                                       |

Fig. 4.23 Pin assignment table

## 5 Construction of the hardware

### 5.1 Interface PCB

The construction of the Interface PCB cannot be realized at school, because the board owns 4 different layers. Therefore, the production of the PCB was passed on to an extern manufacturer.

Because the routage of the schema will be very complicated, it will be done by Steve Gallay of the electronic section. The final electrical scheme and PCB layout are given in the Appendix 10. I also want to thank Steve, for his time spent on this project.

**C** Appendix 11 Electrical and PCB layout of the Interface board

### 5.2 Converter PCB

The Converter board contains 2 layers. The routage was done by me, because the different lines were, with some exceptions, directly connected to the Controller. Therefore it was relatively simple to realize the PCB layout. The final electrical scheme and PCB layout are given in the Appendix 11.

**C** Appendix 12 Electrical and PCB layout of the Converter board

### 5.3 List of order for Project Frame Capturing and sending in FPGA

In the following table, you can see a complete list of all components I ordered. It includes all components for the Interface-board, the Converter-board and the Development-board. The total costs of my project were 1150SFr. One of the reason, why this amount is so high, is, that we had not the time to build the Controller-board, therefore we had to buy an Actel Development-board, which has a price of more than 780SFr.

| Quantity        | Order number | Name              | Material                                      | Price/unit<br>CHF | Price total<br>CHF | Contact                                                                                                                                  |
|-----------------|--------------|-------------------|-----------------------------------------------|-------------------|--------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| No Distrelec    |              |                   |                                               |                   |                    |                                                                                                                                          |
| 2               | 644127       | VX3MH-5000        | Oszillator 50Mhz 50PPM                        | SFr. 7.00         | SFr. 14.00         | Distrelec<br><a href="http://www.distrelec.ch">www.distrelec.ch</a>                                                                      |
| 1               | 640916       | XC 95288-20PQ208C | Xilinx CPLD XC9500XL                          | SFr. 84.00        | SFr. 84.00         |                                                                                                                                          |
| 1               | 121776       | 154963 3 x 32     | Female 3X32Pin Connector                      | SFr. 7.10         | SFr. 7.10          |                                                                                                                                          |
| 6               | 110831       | NFE31PT222Z1E9L   | Filter                                        | SFr. 1.72         | SFr. 10.32         |                                                                                                                                          |
| 2               | 513096       | 3801-40           | Ribbon cabel 40-Pol                           | SFr. 10.11        | SFr. 20.22         |                                                                                                                                          |
| 10              | 121674       | 0918 540 6813     | Female Plug 2*20Pol                           | SFr. 4.52         | SFr. 45.20         |                                                                                                                                          |
| Total Distrelec |              |                   |                                               |                   | SFr. 180.84        |                                                                                                                                          |
|                 |              |                   |                                               |                   | SFr. 180.84        |                                                                                                                                          |
| No Farnell      |              |                   |                                               |                   |                    |                                                                                                                                          |
| 3               | 1207055      | OPA3692ID-G4      | Texas Instrument OPAMP                        | SFr. 7.95         | SFr. 23.85         | Farnell AG<br>Brandschenkestr.<br>178<br>Postfach 1703<br>ch-8027 Zürich<br><a href="http://www.farnellinone.ch">www.farnellinone.ch</a> |
| 2               | 1106047      | TPS54316PWP       | Texas Instument Buck-Boost                    | SFr. 10.10        | SFr. 20.20         |                                                                                                                                          |
| 2               | 1188060      | MAX811LEUS+T      | Maxim Dallas Manual Reset                     | SFr. 4.60         | SFr. 9.20          |                                                                                                                                          |
| 2               | 9487956      | DS26LS32ACN       | National Semiconductor<br>RS422 quad Receiver | SFr. 2.50         | SFr. 5.00          |                                                                                                                                          |
| 2               | 9724516      | MAX483ECPA+       | Maxim Dallas<br>RS422 single transmitter      | SFr. 6.70         | SFr. 13.40         |                                                                                                                                          |
| Total Farnell   |              |                   |                                               |                   | SFr. 71.65         |                                                                                                                                          |
|                 |              |                   |                                               |                   | SFr. 252.49        |                                                                                                                                          |

| Quantity                     | Order number | Name                   | Material                                         | Price/unit<br>CHF | Price total<br>CHF | Contact                                                                          |
|------------------------------|--------------|------------------------|--------------------------------------------------|-------------------|--------------------|----------------------------------------------------------------------------------|
| MSC                          |              |                        |                                                  |                   |                    |                                                                                  |
| 1                            | -            | CORE-MP7_1000-DEV-KIT  | Developement Kit Actel<br>+ Flashpro3 Programmer | \$600.00          | \$600.00           | MSC Suisse SA<br>Avenue Nestle 14<br>CH-1820 Montreux<br>Montreux@msc-<br>ge.com |
| 1                            | -            | M7A3P1000-FG484        | Actel FPGA 1M gates                              | \$85.00           | \$85.00            |                                                                                  |
| Total MSC                    |              |                        |                                                  |                   | \$685.00           |                                                                                  |
|                              |              |                        |                                                  |                   | SFr. 1'013         |                                                                                  |
| National Semiconductors      |              |                        |                                                  |                   |                    |                                                                                  |
| 1                            | -            | DP83849CVS             | Dual-Port Ethernet<br>Transceiver                | SFr. 0.00         | SFr. 0.00          | <a href="http://www.national.com">www.national.com</a><br>Sample order           |
| Total National Semiconductor |              |                        |                                                  |                   | SFr. 0.00          |                                                                                  |
|                              |              |                        |                                                  |                   | SFr. 1'013         |                                                                                  |
| Various resistors            |              |                        |                                                  |                   |                    |                                                                                  |
| Various condenser            |              |                        |                                                  |                   |                    |                                                                                  |
| Various inductances          |              |                        |                                                  |                   |                    |                                                                                  |
| 16                           |              | Jumpers                |                                                  |                   |                    | Available material<br>in school                                                  |
| 1                            |              | JTAG Connector         |                                                  |                   |                    |                                                                                  |
| 1                            |              | 3*32Pin Male Connector |                                                  |                   |                    |                                                                                  |
| 1                            |              | RJ45 Connector         | without magnetics                                |                   |                    |                                                                                  |
| 2                            |              | MagJack 0810-1X1T-03   | RJ45 with magnetics                              |                   |                    |                                                                                  |
| 3                            |              | SMD-LED                |                                                  |                   |                    |                                                                                  |
| 1                            |              | 5V-Power Plug          |                                                  |                   |                    |                                                                                  |
| 3                            |              | 2X20Pin Connector      |                                                  |                   |                    |                                                                                  |
| 1                            |              | Reset-Button           |                                                  |                   |                    |                                                                                  |
| Total school                 |              |                        |                                                  |                   | SFr. 0.00          |                                                                                  |
| Total price                  |              |                        |                                                  |                   | SFr. 1'013.24      |                                                                                  |

C Fig. 5.1 List of order

## 6 Actual state

### Interface

The Interface electrical scheme and also the routage is done. Within the last weeks of school, the PCB scheme was submitted to the external manufacturer. Until the end of the project, the board couldn't be completed. Therefore, it was not possible, to put the ordered components all together and test the interface board.

### Converter

The Converter was completed on time and the components were put together. Because the Interface couldn't be finished completely, we could test this board only conditionally.

### Controller

The Development board was ordered. The enclosed programmes were installed properly. The development-kit include a tutorial project, I have tried to implement this project on the dev-board. But we had an error while place & route. I would have needed a bit more time to eliminate this error. Nevertheless, I could still read some documentation and get an overview of how the board works.

## 7 Further work

I would like to make suggestions for the further development of the project:

- Finish and test the Interface-board
  - o Mount the components on the Interface
  - o Test the Interface
- Build of a Interface with other Input Sources, like 1000Base-TX
- Build of the Controller Board with an Actel Core MP7 FPGA
  - o See points I listed on 4.1 Problem analysis and solution approaches
- Implement the CPLD-functions
  - o Split the 4 Data-signals per port to 32 Data-signals per port
  - o Adapt the control signals
  - o Create the 25Mhz for the Phy
  - o Transmit the RS422 Signals
  - o Transmit or control the high-impedance buffer disable
- Understanding how the Development board works
  - o Implement the tutorial
- Programming of the Development board
  - o Implement the controller functions
  - o Implement the time synchronization for the time stamp

## 8 Conclusion and remarks

One interest of this project was to realize a work, which should be a simplified diploma work, a kind of a small introduction. Because this project took also place in cooperation with Patrik Arloos, it should be possible for him to read this report. This was also one of the reasons, why I wrote this report in English.

I also improved my knowledge in project management and project planning. In advance, I wrote a time schedule, which I tried to follow, as much as possible. It was also necessary to correct it. I had to work together with many people, without those, it would not have been possible to progress so far in this project.

I would like to thank everyone, who tried to help me in this project. Your support, ideas and comments were very helpful. Special thanks go to:

- Corthay Francois, supervisor of this project
- Gubler Olivier, for giving me his Diploma work as a help
- Steve Gallay, for designing the PCB of the Interface and all the library components I needed
- Sartoretti Pascal, for helping me with the Pcad and to order all the needed components
- Pignat Marc, for helping me to drawn the electrical scheme
- Biner Hans-Peter, for giving me useful tips for the high impedance buffer and high frequency transmissions
- All of my student colleagues

This project was very challenging for me. I learned a lot in electrical engineering and how to build a PCB. An important point is also to choose the right components. The first weeks were just gathering of the required information and looking what I had to do, after that, I could really start working seriously on the project.

During this semester I could work in many different topics: Hardware, Software

Unfortunately, I didn't succeed in finishing the whole project, because of the lack of time. I still must complete and test the unit till the Begin of the diploma work in Sweden.

Zahno Silvan:



## 9 Glossary

|                |                                                                                                                                                                                                                                                                                                                                                                              |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MP             | Measurement Point                                                                                                                                                                                                                                                                                                                                                            |
| FPGA           | Field Programmable Gate Array, is a semiconductor device containing programmable logic components and programmable interconnects.                                                                                                                                                                                                                                            |
| Phy            | Physical, this IC representing the first layer of the OSI-model                                                                                                                                                                                                                                                                                                              |
| MArN           | Measurement Area Network                                                                                                                                                                                                                                                                                                                                                     |
| Eth            | Ethernet                                                                                                                                                                                                                                                                                                                                                                     |
| TP             | Twisted Pair, is a form of wiring, this cables are used to connect the MP to the MArN                                                                                                                                                                                                                                                                                        |
| VHDL           | Very High Speed Integrated Circuit Hardware Description Language, is a programming language to describe hardware                                                                                                                                                                                                                                                             |
| JTAG           | Joint Test Action Group, is the usual name used for the IEEE 1149.1 standard entitled Standard Test Access Port and Boundary-Scan Architecture for test access ports used for testing printed circuit boards using boundary scan                                                                                                                                             |
| SD Memory Card | Secure Digital Memory Card, is a digital storage medium, which is based on the princip of the flash technologie                                                                                                                                                                                                                                                              |
| 10BaseT        | 10Base-T is a form of medium speed Ethernet, providing 10Mbit/s                                                                                                                                                                                                                                                                                                              |
| 100BaseTX      | 100Base-TX is the predominant form of Fast Ethernet, providing 100 Mbit/s Ethernet. It introduces an additional, medium dependent sublayer, which employs MLT-3 as a final encoding of the data stream before transmission                                                                                                                                                   |
| 100Base-FX     | 100Base-FX is a version of Fast Ethernet over optical fiber. It uses two strands of multi-mode optical fiber for receive and transmit                                                                                                                                                                                                                                        |
| 4B5B           | 4B5B is a form of data communications line code. It works by mapping groups of four bits onto groups of 5 bits                                                                                                                                                                                                                                                               |
| MLT3           | MLT-3 encoding is a line code (a signaling method used in a telecommunication system for transmission purposes) that uses three voltage levels. An MLT-3 interface emits less electromagnetic interference and requires less bandwidth than most other binary or ternary interfaces that operate at the same data rate                                                       |
| OTP            | One Time Programmable are devices which have a memory which can be programmed just ones.                                                                                                                                                                                                                                                                                     |
| OP             | An Operational amplifier, usually referred to as an op for brevity, is a DC-coupled high-gain electronic voltage amplifier with differential inputs and, usually, a single output.                                                                                                                                                                                           |
| RJ45           | Registered Jack (RJ) is a standardized physical interface for connecting telecommunications equipment or computer networking equipment. The standard designs for these connectors and their wiring are named RJ11, RJ14, RJ45, etc.                                                                                                                                          |
| PCB            | Printed Circuit Boards are used to mechanically support and electrically connect electronic components using conductive pathways, or traces, etched from copper sheets laminated onto a non-conductive substrate.                                                                                                                                                            |
| NRZ-encoding   | Non-Return-to-Zero (NRZ) line code is a binary code in which can represent just two conditions, "0" and "1"                                                                                                                                                                                                                                                                  |
| CPLD           | Complex Programmable Logic Device is a programmable logic device. The building block of a CPLD is the macro cell, which contains logic implementing disjunctive normal form expressions and more specialized logic operations.                                                                                                                                               |
| CRC            | Cyclic Redundancy Check (CRC) is a type of function that takes as input a data stream of unlimited length and produces as output a value of a certain fixed size. The term CRC is often used to denote either the function or the function's output. A CRC can be used in the same way as a checksum to detect accidental alteration of data during transmission or storage. |
| RTC            | Real-Time Clock is a computer clock that keeps track of the current time. Although the term often refers to the devices in personal computers, servers and embedded systems, RTCs are present in most any electronic device which needs to keep accurate time.                                                                                                               |

## 10 Bibliography

Manufacturer:

[www.actel.com](http://www.actel.com)

[www.xilinx.com](http://www.xilinx.com)

[www.intel.com](http://www.intel.com)

[www.belfuse.com](http://www.belfuse.com)

[www.magjack.com](http://www.magjack.com)

[www.nationalsemiconductor.com](http://www.nationalsemiconductor.com)

Distributors:

[www.distrelec.com](http://www.distrelec.com)

[www.silica.com](http://www.silica.com)

[www.farnell.com](http://www.farnell.com)

[www.reselec.ch](http://www.reselec.ch)

[www.msc.ch](http://www.msc.ch)

Informations:

[www.wikipedia.com](http://www.wikipedia.com)

and much other sites...

## 11 Appendix

- Appendix 1: Meeting Document
- Appendix 2: Datasheet OPA3692ID
- Appendix 3: Datasheet MagJack 0801-1X1T-03
- Appendix 4: Datasheet Agilent AFBR-5903Z
- Appendix 5: DP83849 Dual Phyter Demo II Stump Jumper scheme
- Appendix 6: Datasheet National Semiconductor DP83849ID
- Appendix 7: Datasheet Xilinx XC95288XL
- Appendix 8: Datasheet ProASIC3E Flash Family
- Appendix 9: CoreMP7 Development Kit Users Guide
- Appendix 10: Datasheet CoreMP7
- Appendix 11: Electrical and PCB scheme of the Interface board
- Appendix 12: Electrical and PCB scheme of the Converter board

## Appendix 1: Meeting Document

## HEVs

|    |    |    |    |     |
|----|----|----|----|-----|
| SI | TV | EE | IG | EST |
| X  | X  |    |    |     |

# Echéancier du projet de semestre Semesterprojekt : Termine

FO 1.2.02.04.DB  
mam/31/03/2006

|                                                                             |                                                  |                                                       |
|-----------------------------------------------------------------------------|--------------------------------------------------|-------------------------------------------------------|
| Filière / Studiengang<br><b>Systèmes Industriels – it<br/>Systemtechnik</b> | Année scolaire / Schuljahr<br><b>2006/07</b>     | No PS / Nr. PS<br><b>SI/2007/13</b>                   |
| Mandant / Auftraggeber                                                      | Mandataire / Beauftragter<br><b>Silvan Zahno</b> | Lieu d'exécution / Ausführungsort<br><b>HEVs, DSI</b> |
| Resp.d'unité / Leiter Einheit<br><b>Pierre Pomplii</b>                      | Enseignant / Dozent<br><b>François Corthay</b>   |                                                       |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Titre / Titel<br><p style="text-align: center;"><b>Carte FPGA pour mesures de trames ethernet</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Objectif / Ziel<br><p>Le but de ce travail est de réaliser une carte servant de point de mesure pour des trames ethernet. Il s'inscrit dans la collaboration avec Pr. Patrik Arloos, Blekinge Institute of Technology, Karlskrona. Les travaux du Pr. Patrik Arloos permettent de mettre en oeuvre un système capable de mesurer le trafic sur un réseau ethernet.</p> <p>Le circuit se divise en 3 blocs :</p> <ul style="list-style-type: none"> <li>• Le «wire-tap» qui sert de rallonge ethernet et copie les signaux vers une autre sortie</li> <li>• La carte d'interface physique qui transforme les signaux ethernet en signaux logiques indépendants de la ligne ethernet</li> <li>• La carte contrôleur avec une FPGA qui reçoit les trames et une information de temps pour les estampiller. Cette carte transmet le résultat des mesures sur une ligne ethernet.</li> </ul> <p>Le travail de semestre consistera principalement à faire le schéma et le circuit imprimé de ces cartes. Une partie de la logique programmable sera développée en parallèle avec la réalisation des circuits imprimés.</p> |

Visa du responsable de l'unité / Visum Leiter der Einheit

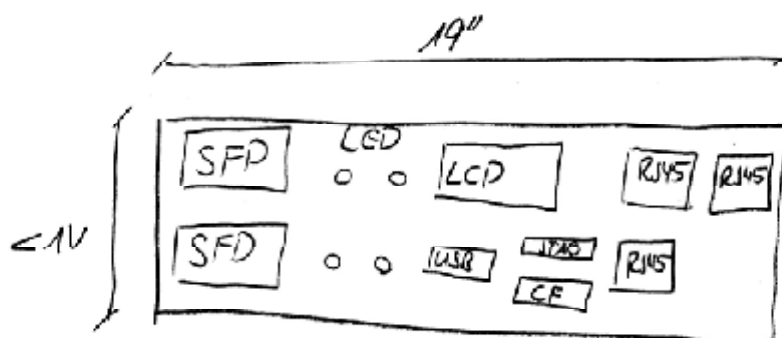
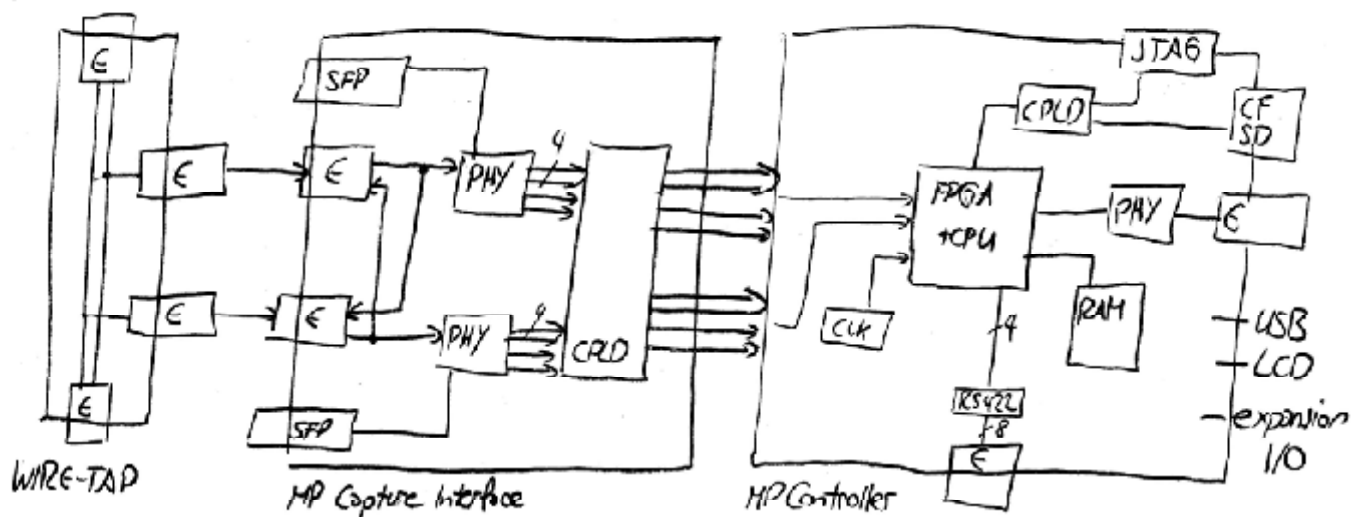
*Pomplii*

|                  |                                                                      |                                                             |
|------------------|----------------------------------------------------------------------|-------------------------------------------------------------|
| Délais / Termine | Attribution du thème<br><i>Ausgabe des Auftrags</i>                  | 21.02.2007                                                  |
|                  | Remise du rapport de l'étudiant<br><i>Abgabe des Schlussberichts</i> | 08.06.2007 – 17.00                                          |
|                  | Défense orale<br><i>Mündliche Verfechtung</i>                        | 13 – 15.06.2007<br>selon programme / <i>gemäss Programm</i> |

Rapport final reçu le  
*Schlussbericht erhalten am* .....

Signature de l'enseignant  
*Unterschrift des Dozenten*

# Measurement Point (MP)

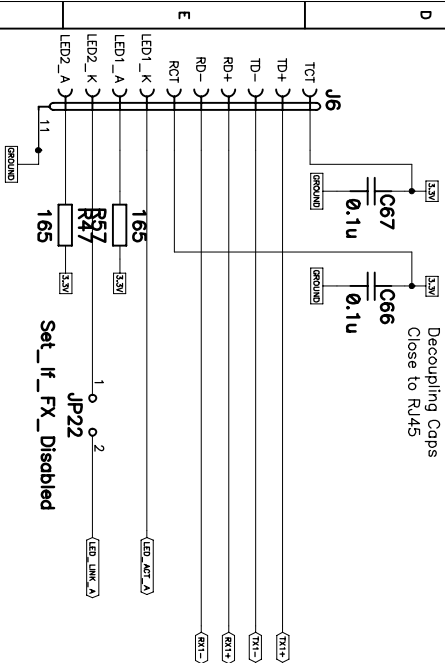
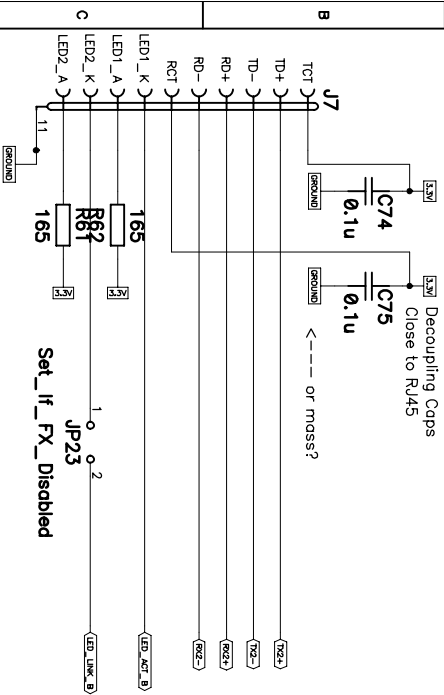


## Appendix 11: Electrical and PCB scheme of the Interface board

|                        |                                                     |                 |                                      |   |   |       |                                                        |                      |          |     |
|------------------------|-----------------------------------------------------|-----------------|--------------------------------------|---|---|-------|--------------------------------------------------------|----------------------|----------|-----|
|                        | 1                                                   | 2               | 3                                    | 4 | 5 | 6     | 7                                                      | 8                    | 9        | 10  |
| A                      |                                                     |                 |                                      |   |   |       |                                                        |                      |          |     |
| B                      | INDEX OF PAGES FOR THE INTERFACE (RJ45 and Optical) |                 |                                      |   |   |       |                                                        |                      |          |     |
|                        | NUMBER                                              | NAME            | DESCRIPTION                          |   |   |       | COMMENTS                                               |                      |          |     |
|                        | 1                                                   | Index           | Description of sheets                |   |   |       | This is this page                                      |                      |          |     |
|                        | 2                                                   | Ethernet Input  | Input connectors                     |   |   |       | Contains the RJ45—plugs and fiber—plugs                |                      |          |     |
|                        | 3                                                   | Input switching | Input switching from RJ45 to Optical |   |   |       | This is the Jumper page to switch fiber<—>RJ45 ...     |                      |          |     |
|                        | 4                                                   | Buffer / PHY    | High Impedace Buffer and PHY         |   |   |       | Contains the Ethernet Phy                              |                      |          |     |
|                        | 5                                                   | CPLD Connector  | CPLD and Connector 3*32 Pin          |   |   |       | Contains the connection to other board and the CPLD    |                      |          |     |
|                        | 6                                                   | Power Supply    | Power Supply Parts                   |   |   |       | Contains the power parts 5V, 3,3V                      |                      |          |     |
| C                      | 7                                                   | Divers          | Clock, JTAG, RS422                   |   |   |       | Contains Clock generation, JTag, RS422 and Board Reset |                      |          |     |
| D                      |                                                     |                 |                                      |   |   |       |                                                        |                      |          |     |
| E                      |                                                     |                 |                                      |   |   |       |                                                        |                      |          |     |
| F                      |                                                     |                 |                                      |   |   |       |                                                        |                      |          |     |
|                        | 1                                                   | 2               | 3                                    | 4 | 5 | 6     | 7                                                      | 8                    | 9        | 10  |
| Frame capture board    |                                                     |                 |                                      |   |   | Index |                                                        | DES                  | 28.03.07 | zos |
| Interface              |                                                     |                 |                                      |   |   |       |                                                        | REV                  | V1.0     |     |
| HAUTE ECOLE VALAISANNE |                                                     |                 |                                      |   |   | 1/7   |                                                        | {Path} Interface.sch |          |     |

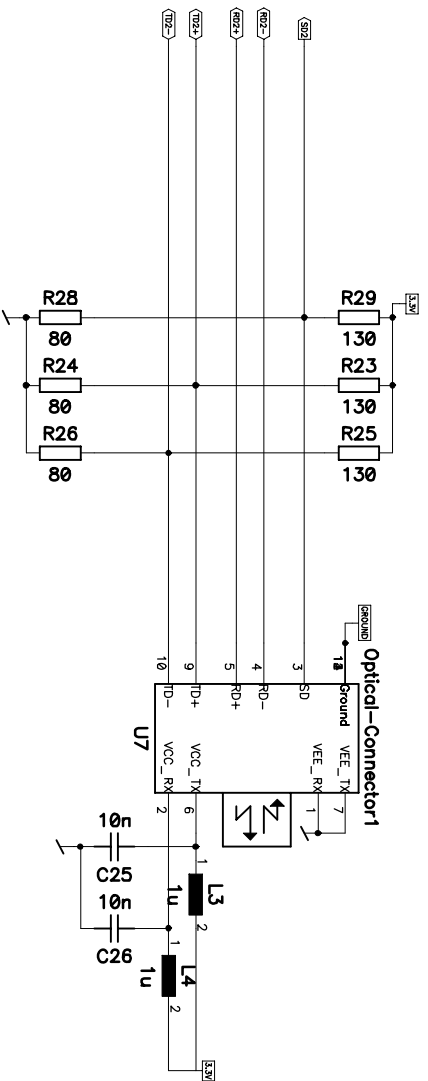
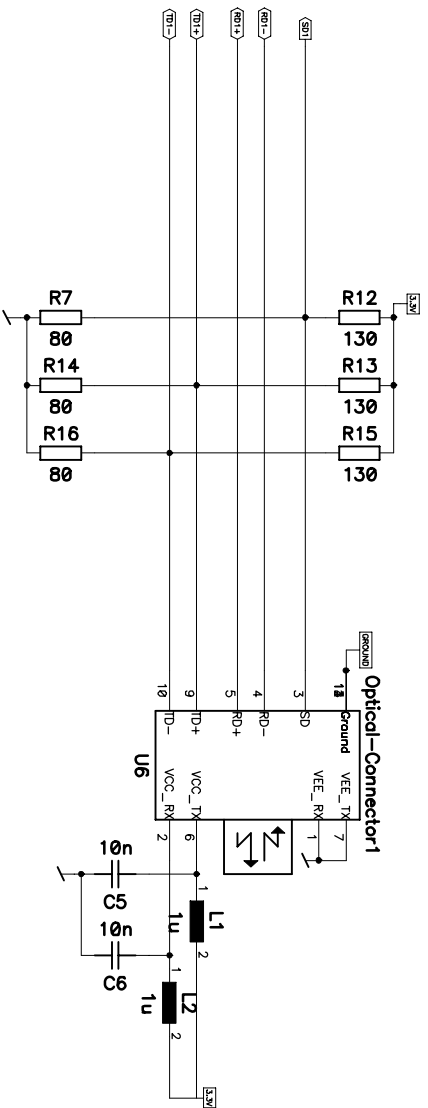
## Ethernet RJ45 input

1 == Port A  
2 == Port B  
Caution, all blue lines should have a  
line impedance of 100/50 Ohm.



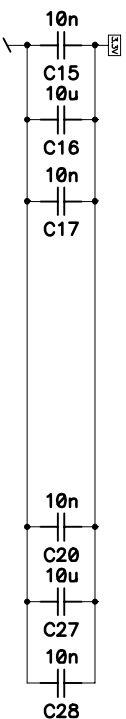
## Ethernet optical fiber input

Optical fiber input  
Caution, all Capacitors and Resistances close to  
the Connectors



## Decoupling caps

The decoupling capacitors are placed on  
the bottom layer and  
close to the fiber-plug's --->



Frame capture board  
Interface  
HAUTE ECOLE VALAISANNE

Ethernet Input  
DES 28.03.07 zos  
REV V1.0  
2/7 {Path}  
Interface.sch



## Ethernet input switching

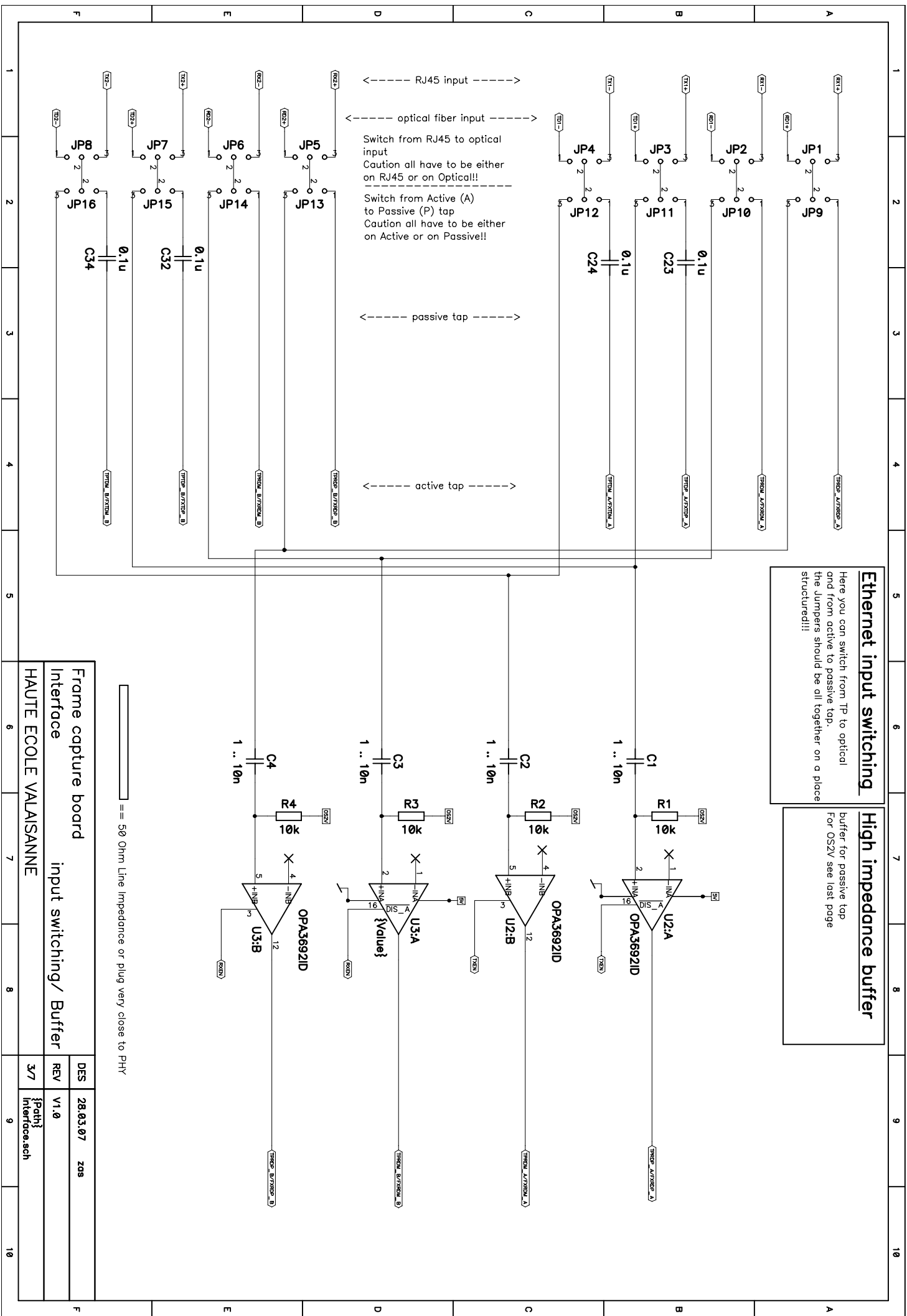
Here you can switch from JP to optical and from active to passive tap.

For OS2V see last page

## High impedance buffer

buffer for passive tap

For OS2V see last page

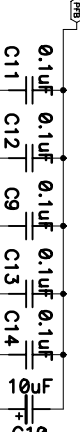


## Ethernet transceiver (PHY)

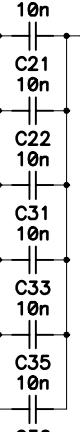
TX\_CLK\_A & B and RX\_CLK\_A & B must be longer than other signals on Mill and routing of the 25MHz line is critical.

## Power Feedback Circuit

Countion these Caps should be placed close to the correspond Phy Pin (PFOUT, PFB), see Datasheet p15 & 38



## Decoupling caps for PHY, please controll!!!



## Decoupling caps

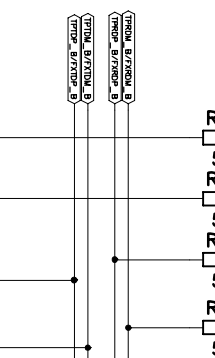
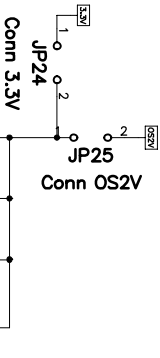
The decoupling capacitors are placed on the bottom layer. Close to Phy

For RJ45, close to PHY

For fiber, close to PHY

## Phy Matching Network

This resistors should be close to the PHY



4.87k

R6

4.87k

R27

2.2k FX\_B\_B\_HD

R27

2.2k FX\_B\_B\_ON

R30

2.2k FX\_B\_ON

R36

2.2k FX\_A\_ON

R38

2.2k CLK2MAC\_DIS

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

R37

2.2k FX\_A\_HD

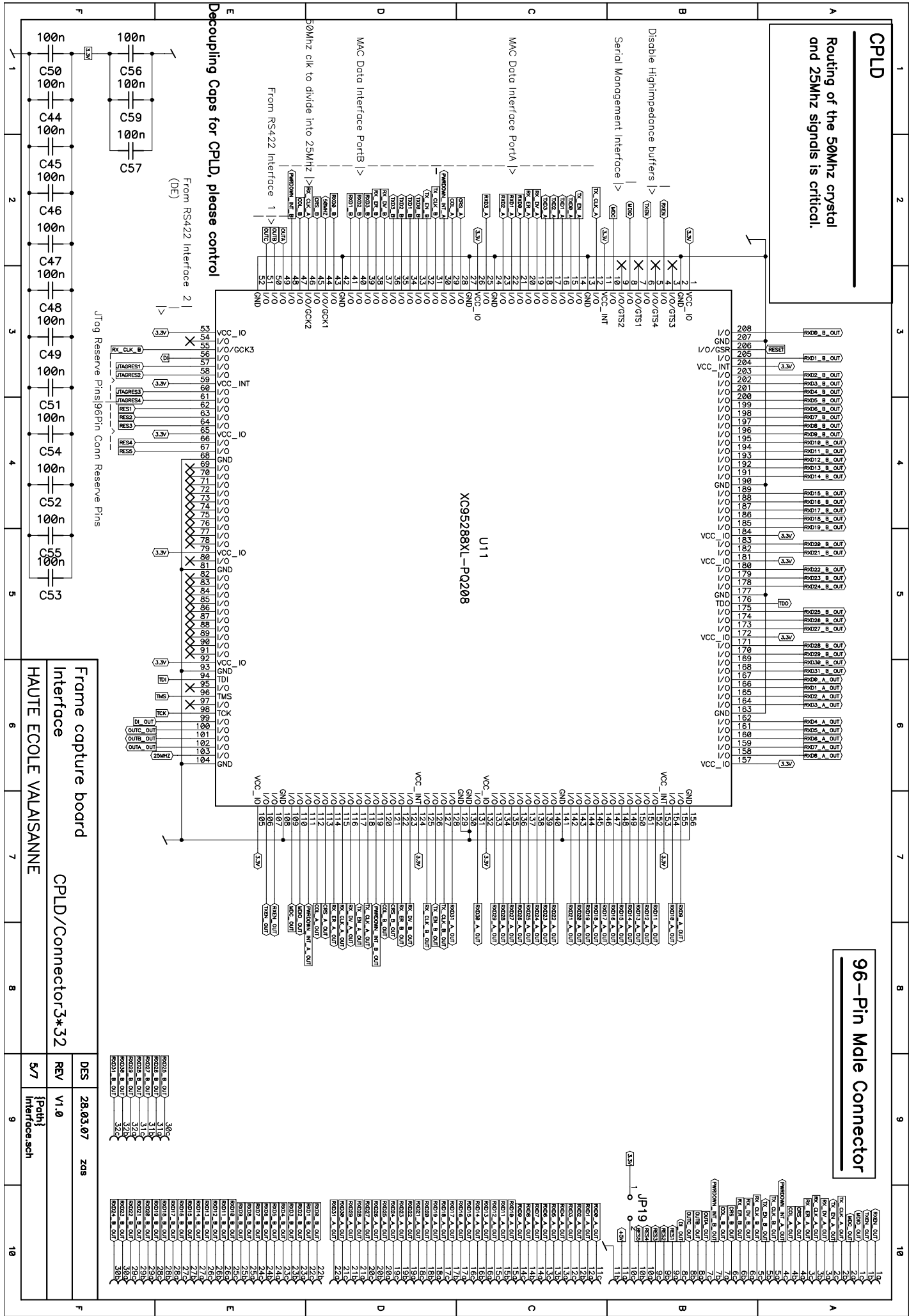
R37

2.2k FX\_A

CPLD

Routing of the 50Mhz crystal and 25Mhz signals is critical.

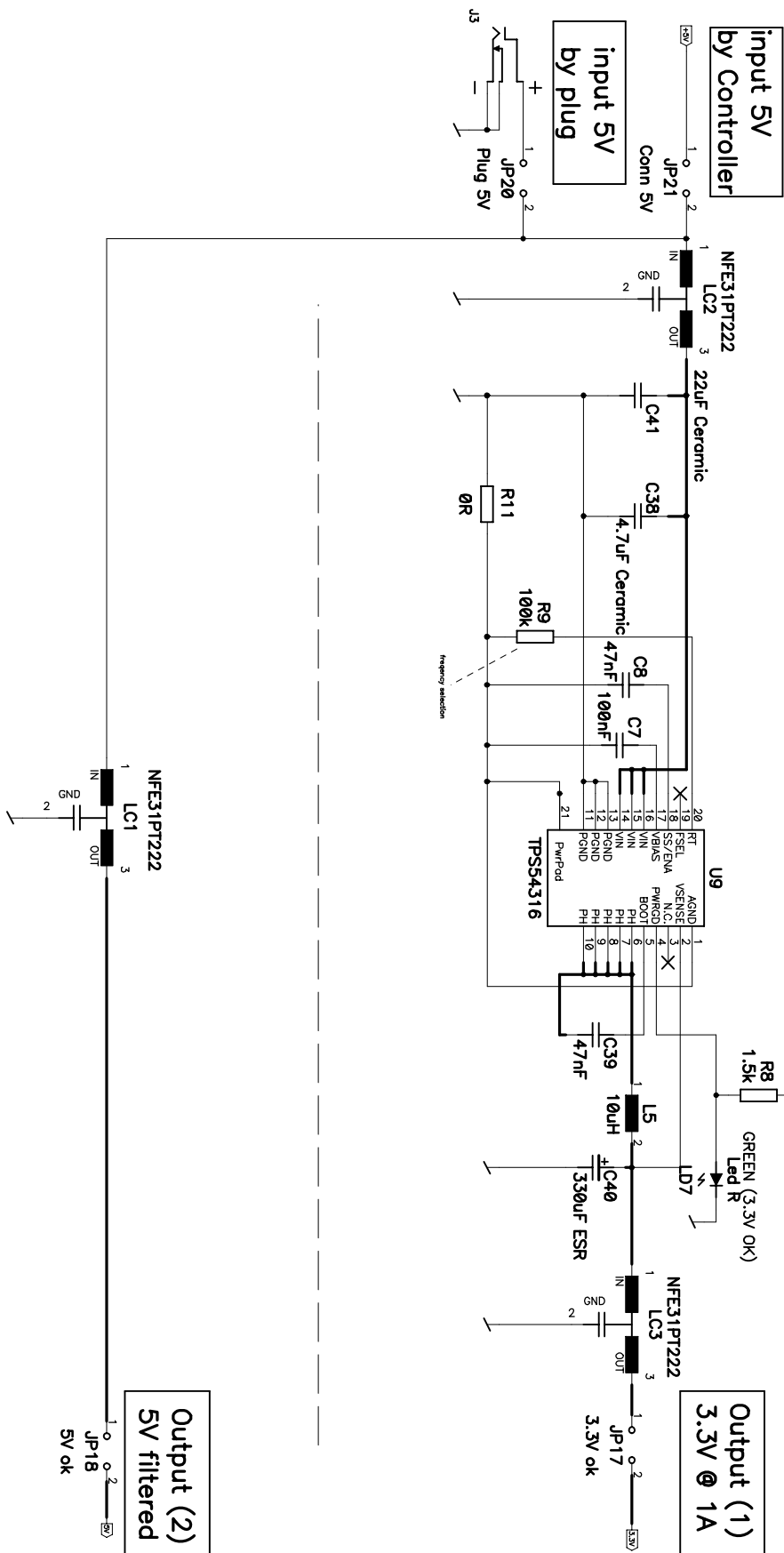
96-Pin Male Connector



|                        |  |                    |  |     |          |               |
|------------------------|--|--------------------|--|-----|----------|---------------|
| Frame capture board    |  | CPLD/Connector3*52 |  | DES | 28.03.07 | Z08           |
| Interface              |  |                    |  | REV | V1.0     |               |
| HAUTE ECOLE VALAISANNE |  |                    |  | 5/7 | {Path}   | Interface.sch |

Power Supply Parts

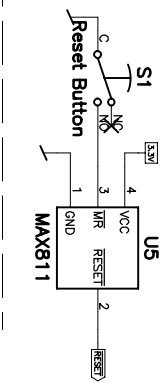
Is this R sufficient?  
PWRGD = 0      V < 90%  
PWRGD = HI-Z    V >= 90%



|                        |  |     |                         |     |
|------------------------|--|-----|-------------------------|-----|
| Frame capture board    |  | DES | 28.03.07                | zos |
| Interface              |  | REV | V1.0                    |     |
| HAUTE ECOLE VALAISANNE |  | 6/7 | {Path}<br>Interface.sch |     |

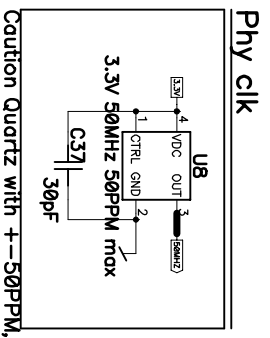
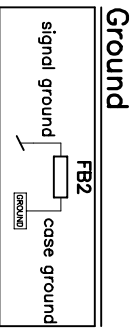
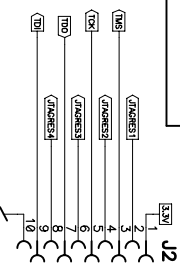
**Board reset**

This IC is NOT decoupled by a capacitor, to detect small power fault. The reset button resets the board and the JTAG.



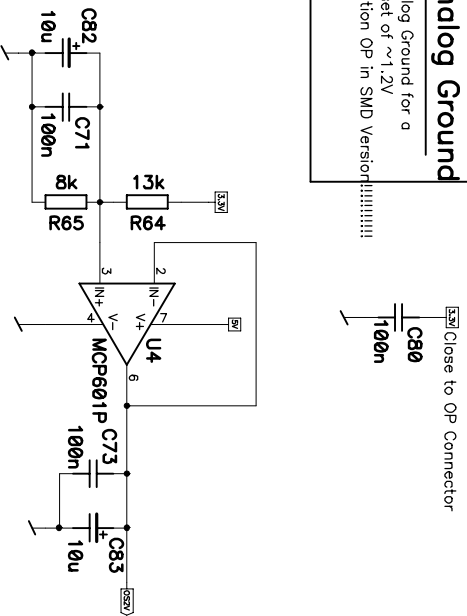
**JTAG Connector**

Caution, are the right pin's connected???



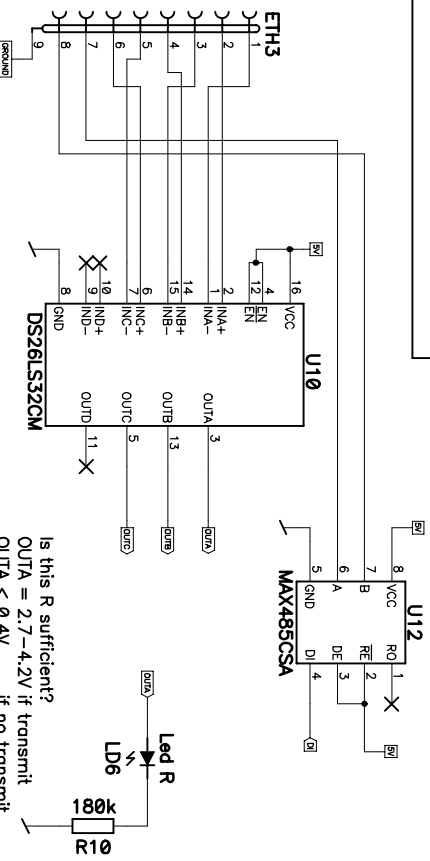
**Analog Ground**

Analog Ground for a offset of ~1.2V  
Caution OP in SMD Version!!!!!!



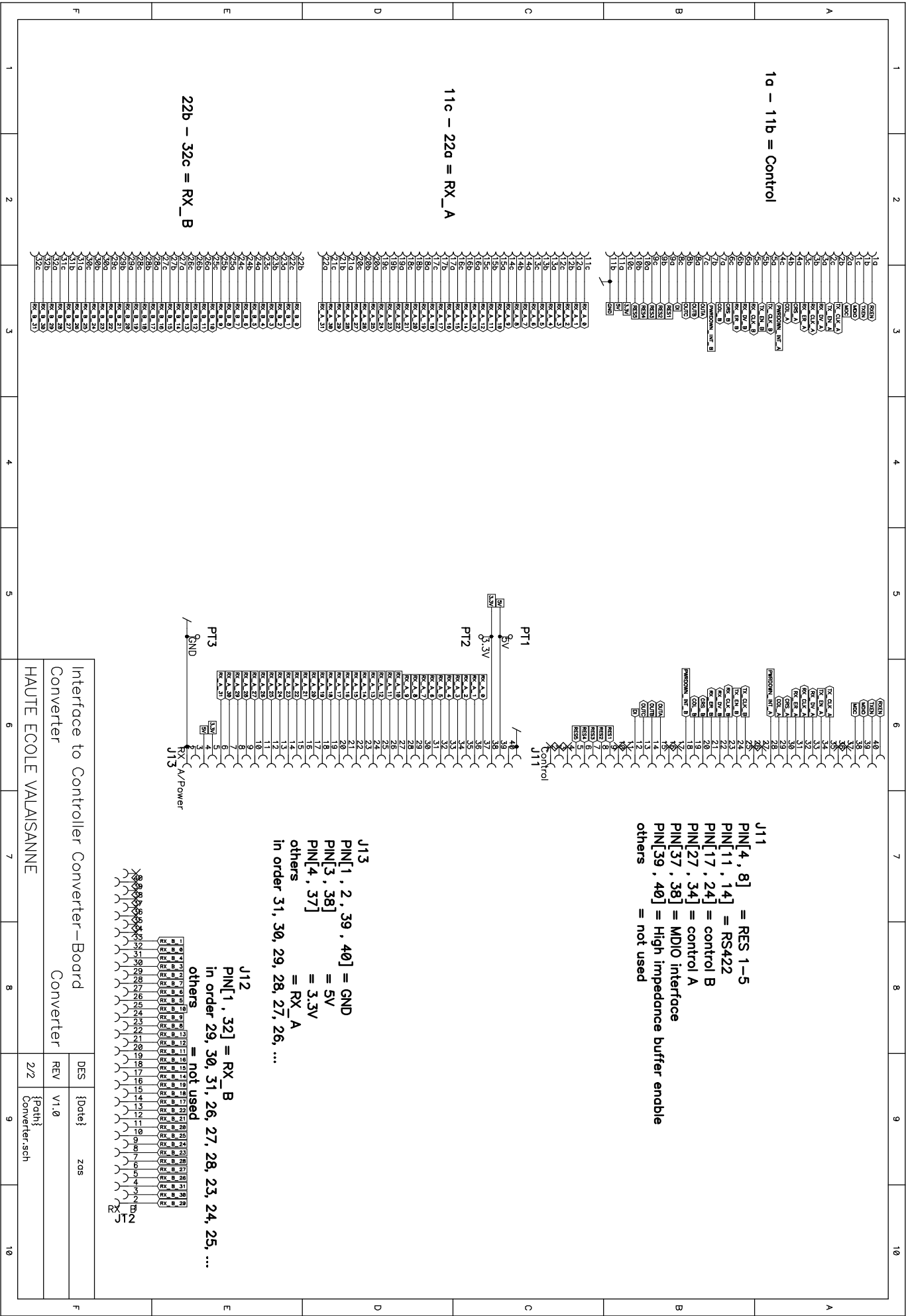
**RS422 Client bloc**

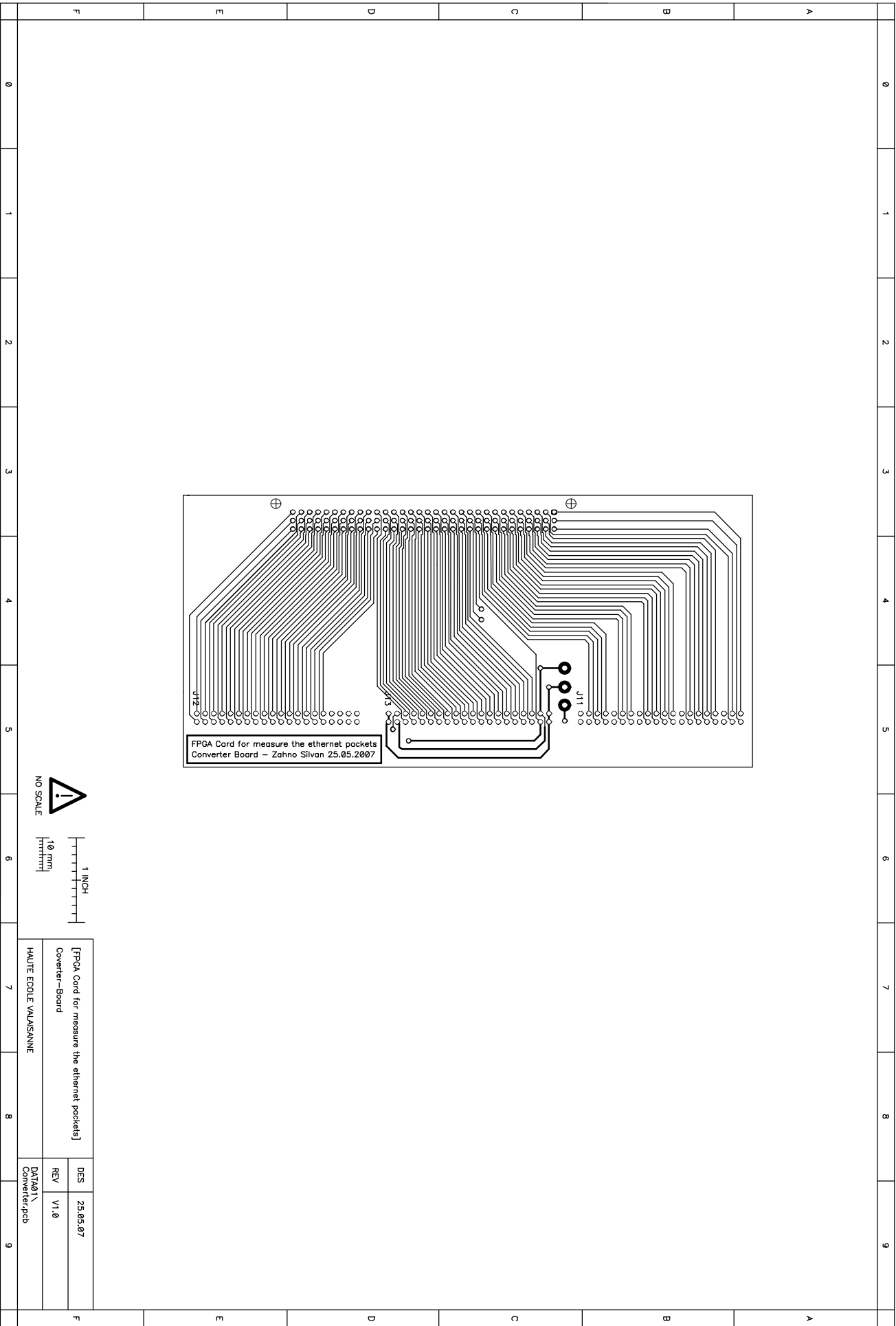
The led shows the transmit and receive actions.  
Normal: 1-2 Port C; 3-4 Port D; 5-6 Port A  
Done: 1-2 Port A; 3-4 Port B; 5-6 Port C



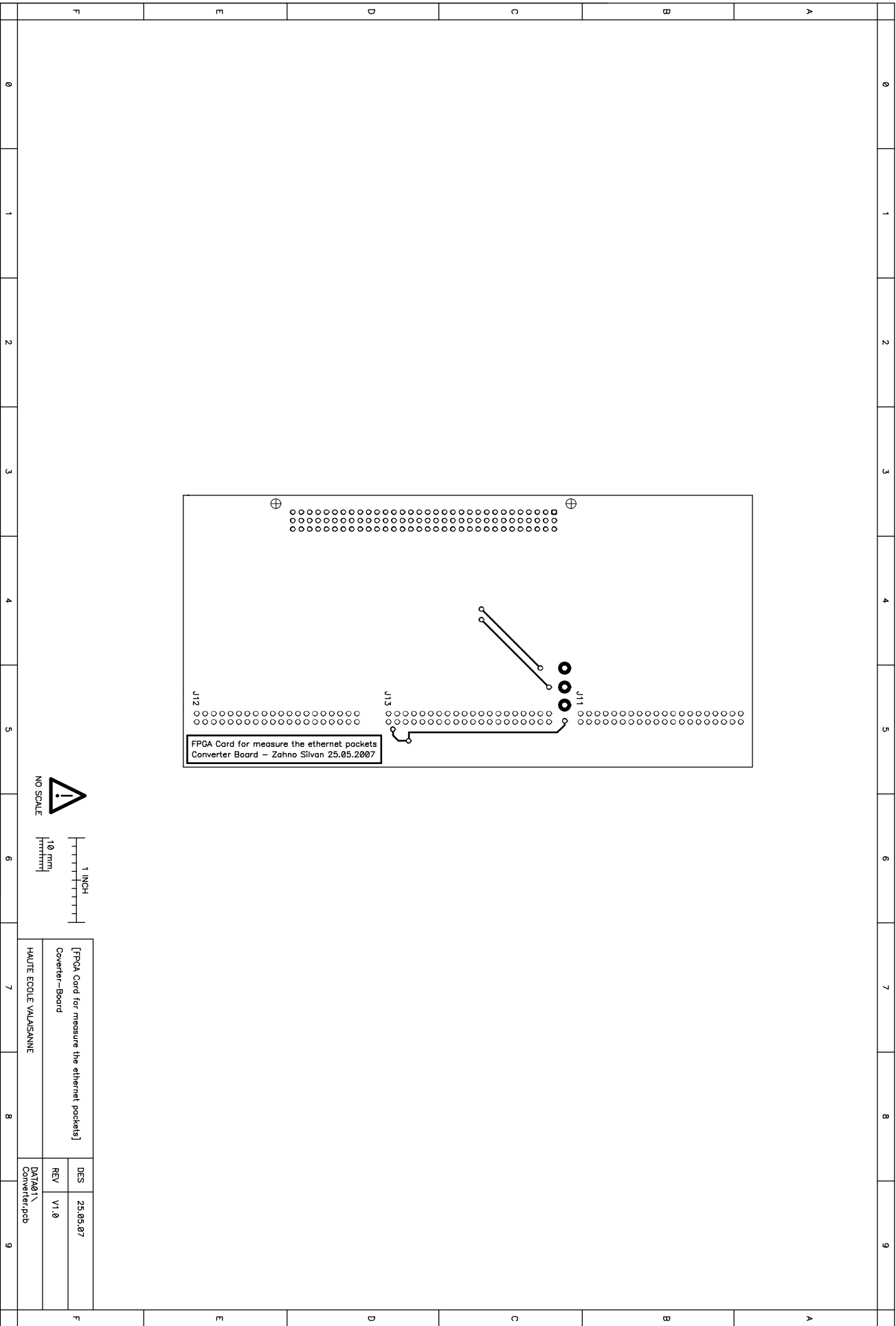
| Frame capture board    |  |        |                         |
|------------------------|--|--------|-------------------------|
| Interface              |  | Divers |                         |
| HAUTE ECOLE VALAISANNE |  | DES    | 28.03.07 zos            |
|                        |  | REV    | V1.0                    |
|                        |  | 7/7    | {Path}<br>Interface.sch |

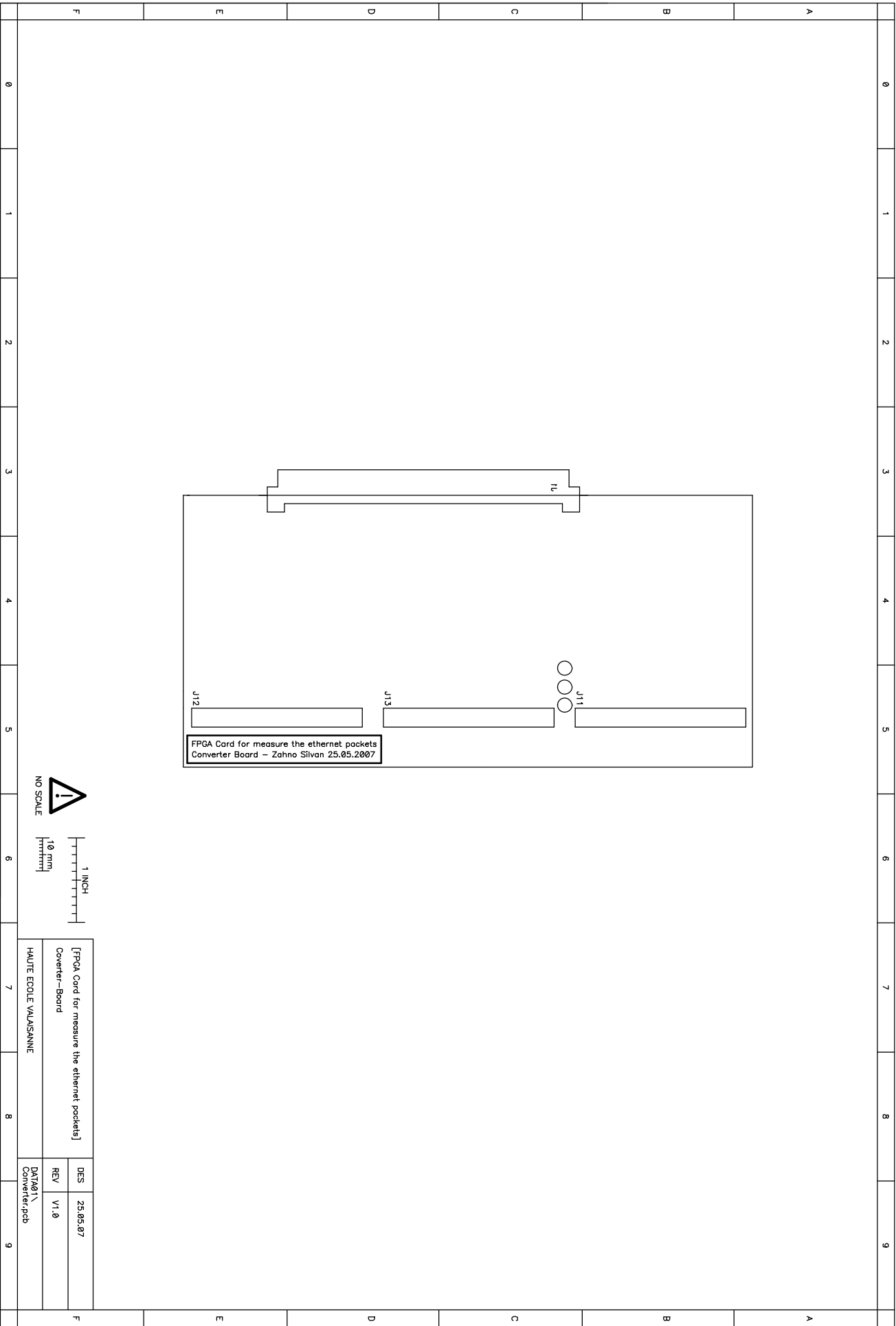
## Appendix 12: Electrical and PCB scheme of the Converter board

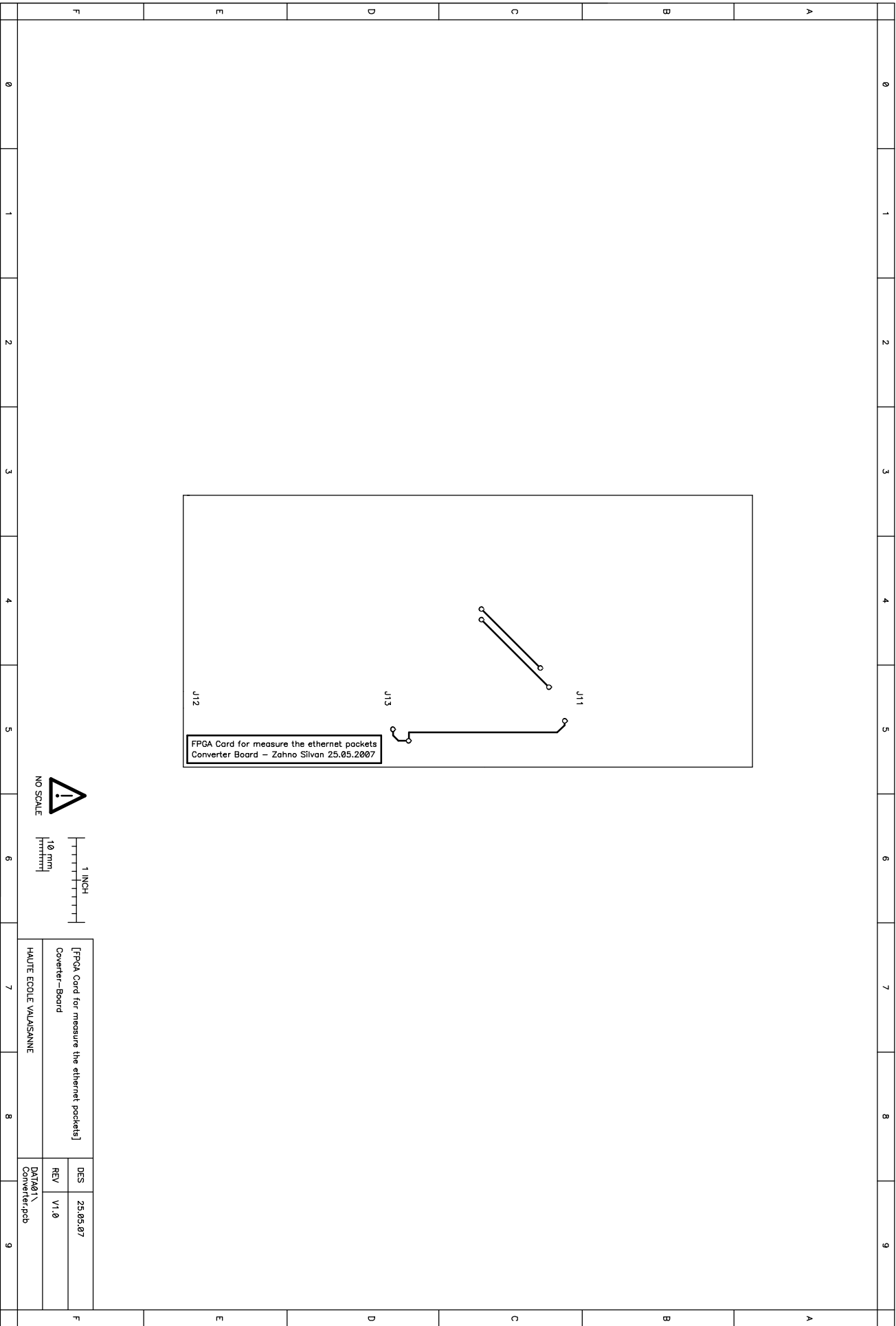










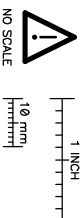
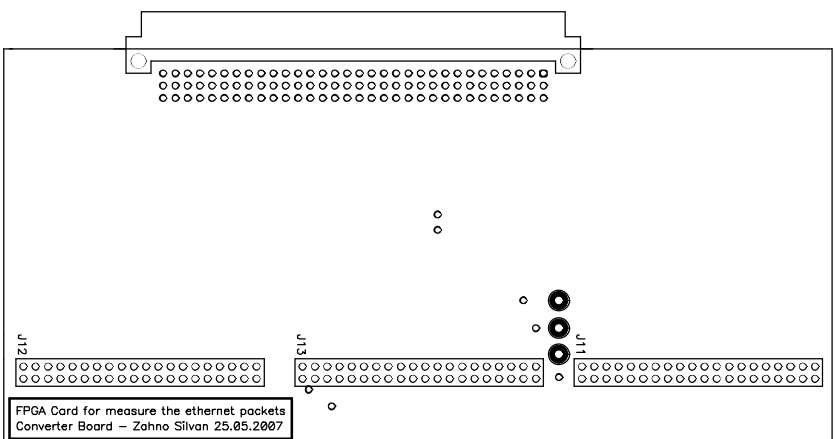


FPGA Card for measure the ethernet packets  
Converter Board - Zahno Silvan 25.05.2007

J13

J11

J12



|                                                                 |         |              |
|-----------------------------------------------------------------|---------|--------------|
| [FPGA Card for measure the ethernet packets]<br>Converter-Board | DES     | 25.05.07     |
|                                                                 | REV     | v1.0         |
| HAUTE ECOLE VILVAISANNE                                         | DATA01\ | Converterpcb |

## **Appendix 2: Schematic Interface**

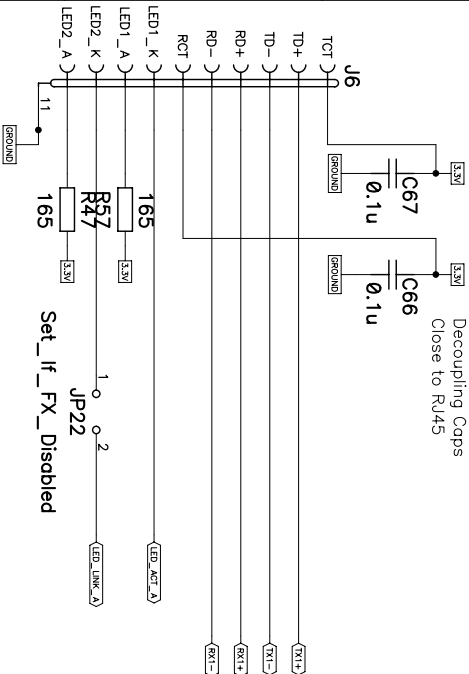
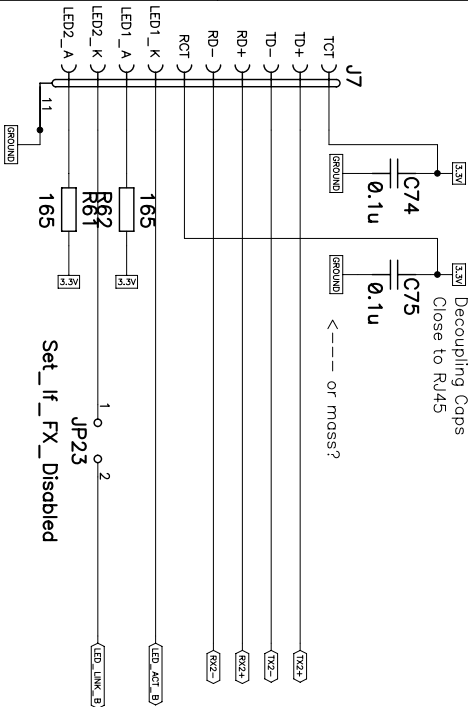
| INDEX OF PAGES FOR THE INTERFACE (RJ45 and Optical) |                 |                                      |                                                     |
|-----------------------------------------------------|-----------------|--------------------------------------|-----------------------------------------------------|
| NUMBER                                              | NAME            | DESCRIPTION                          | COMMENTS                                            |
| 1                                                   | Index           | Description of sheets                | This is this page                                   |
| 2                                                   | Ethernet Input  | Input connectors                     | Contains the RJ45—plugs and fiber—plugs             |
| 3                                                   | Input switching | Input switching from RJ45 to Optical | This is the Jumper page to switch fiber<—>RJ45 ...  |
| 4                                                   | Buffer / PHY    | High Impedace Buffer and PHY         | Contains the Ethernet Phy                           |
| 5                                                   | CPLD Connector  | CPLD and Connector 3*32 Pin          | Contains the connection to other board and the CPLD |
| 6                                                   | Power Supply    | Power Supply Parts                   | Contains the power parts 5V, 3,3V                   |
| 7                                                   | Divers          | Clock, JTAG, RS422                   | Contains Clock generation, JTag, RS422 and Board    |

|                        |  |  |  |     |                         |     |
|------------------------|--|--|--|-----|-------------------------|-----|
| Frame capture board    |  |  |  | DES | 28.03.07                | zas |
| Interface              |  |  |  | REV | V1.0                    |     |
| HAUTE ECOLE VALAISANNE |  |  |  | 1/7 | {Path}<br>Interface.sch |     |

**Ethernet RJ45 input**

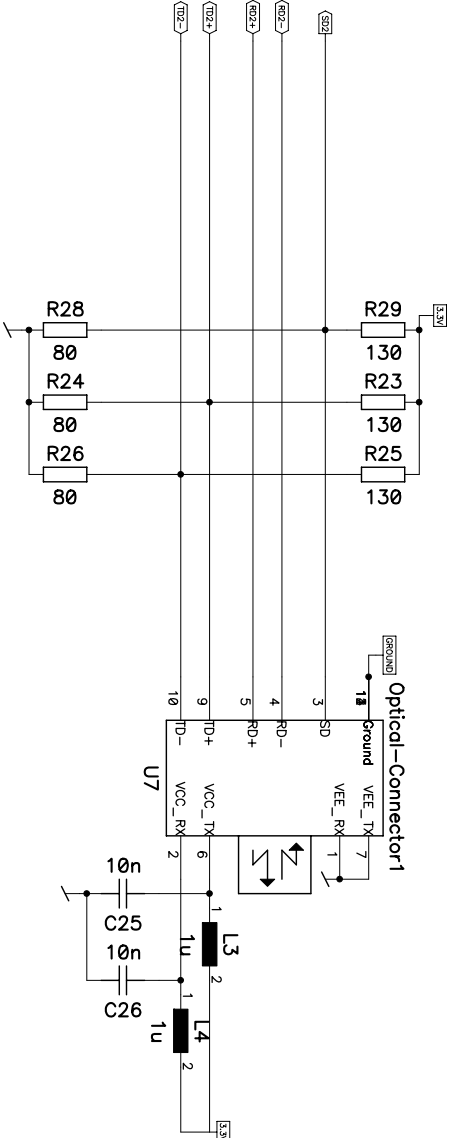
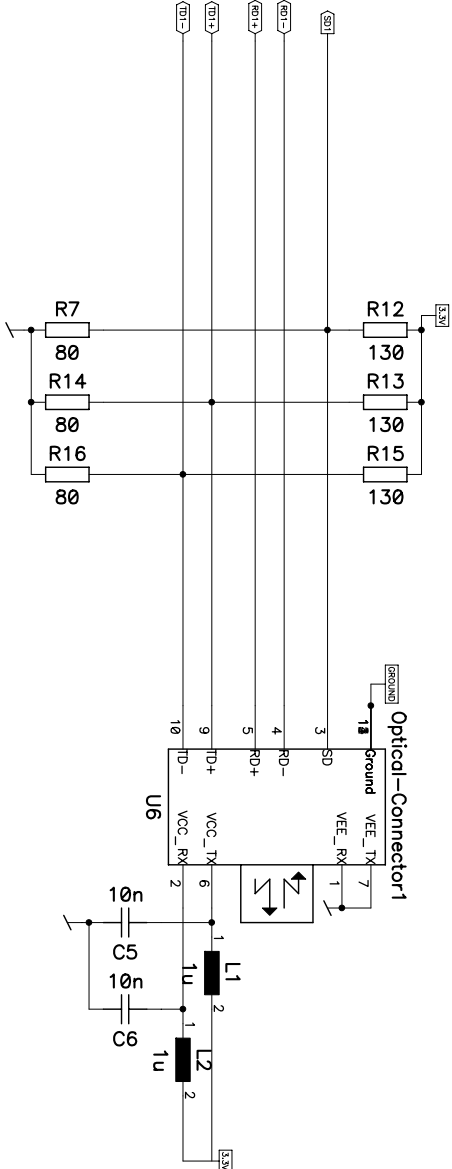
1 == Port A  
2 == Port B

Caution, all blue lines should have a line impedance of 100/50 Ohm.



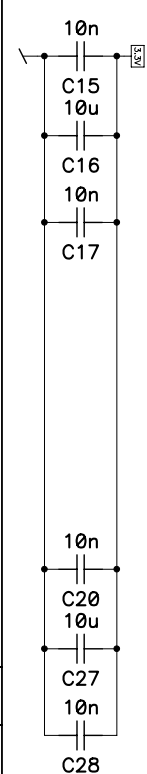
**Ethernet optical fiber input**

Optical fiber input  
Caution, all Capacitors and Resistances close to the Connectors

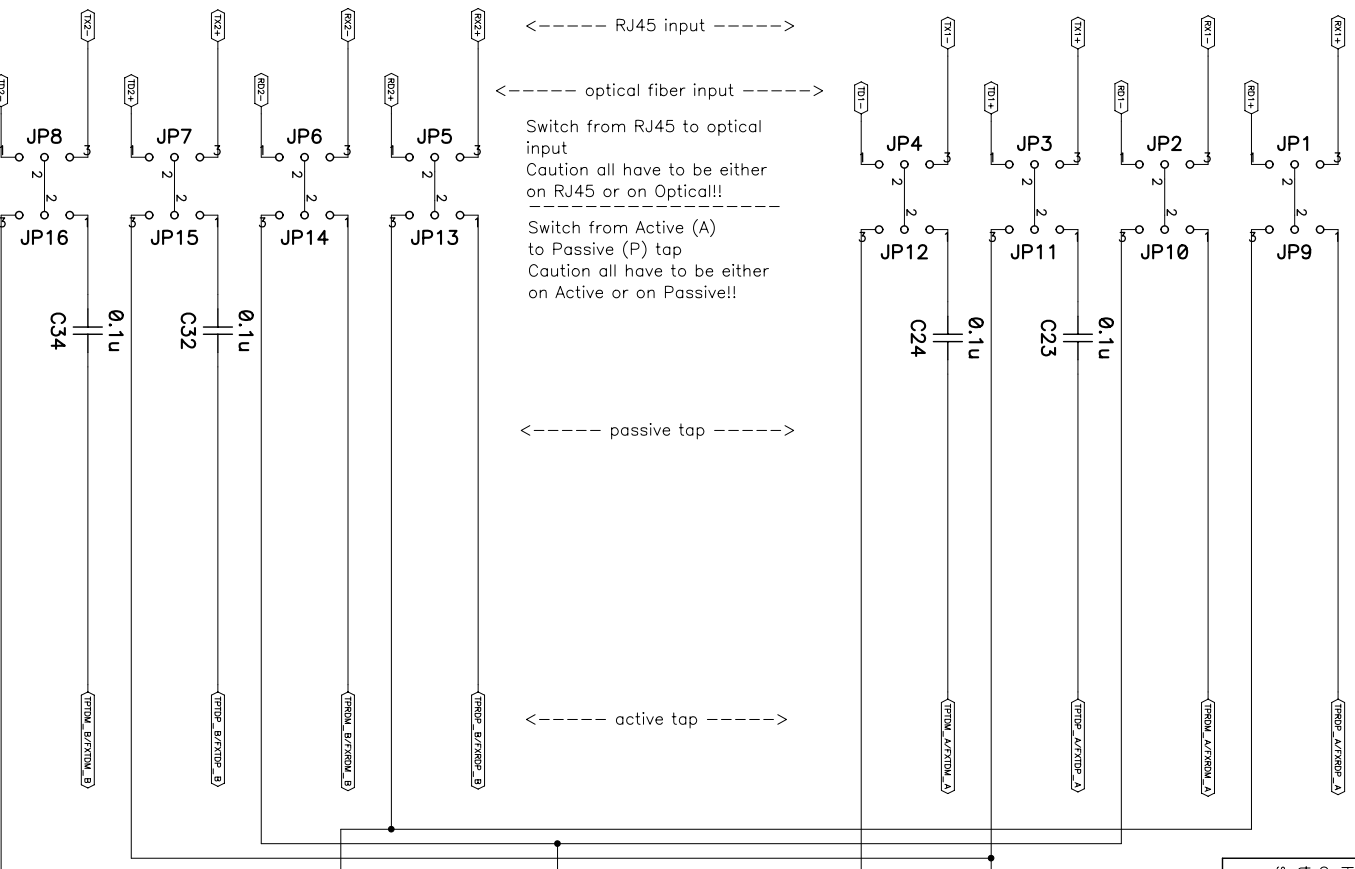


**Decoupling caps**

The decoupling capacitors are placed on the bottom layer and close to the fiber-plug's --->



|                        |  |  |  |                |  |     |          |               |
|------------------------|--|--|--|----------------|--|-----|----------|---------------|
| Frame capture board    |  |  |  | Ethernet Input |  | DES | 28.03.07 | zos           |
| Interface              |  |  |  |                |  | REV | V1.0     |               |
| HAUTE ECOLE VALAISANNE |  |  |  |                |  | 2/7 | {Path}   | Interface.sch |

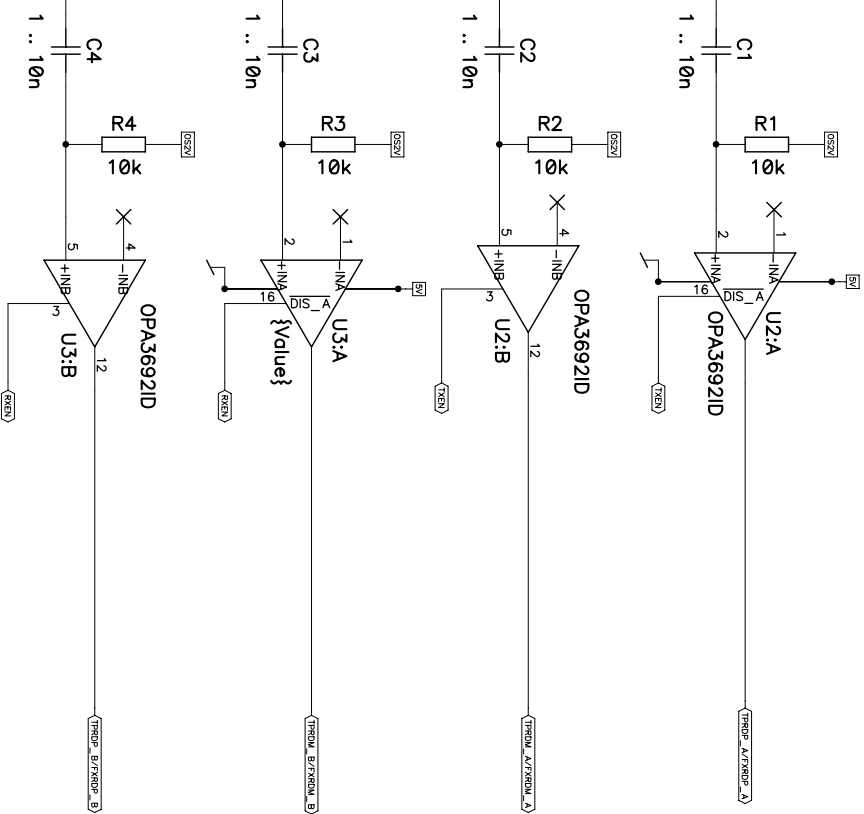


## Ethernet input switching

Here you can switch from TP to optical and from active to passive tap.  
the Jumpers should be all together on a place structured!!!

### High impedance buffer

buffer for passive tap  
For OS2V see last page



== 50 Ohm Line Impedance or plug very close to PHY

|                                  |     |                           |     |
|----------------------------------|-----|---------------------------|-----|
| Frame capture board<br>Interface | DES | 28.03.07                  | zas |
|                                  | REV | V1.0                      |     |
| HAUTE ECOLE VALAISANNE           | 3/7 | {{Path}}<br>interface.sch |     |

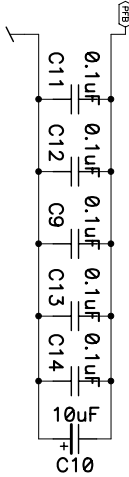


# Ethernet transceiver (PHY)

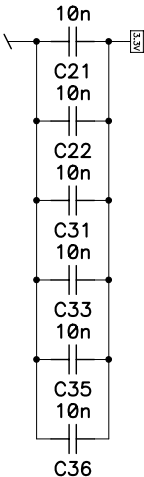
TX\_CLK\_A & B and RX\_CLK\_A & B must be longer than other signals on MI and routing of the 25MHz line is critical.

## Power Feedback Circuit

Countion these Caps should be placed close to the correspond Phy Pin (PFOUT, PFB), see Datasheet p15 & 38

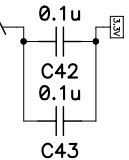


Decoupling caps for PHY, please controll!!!



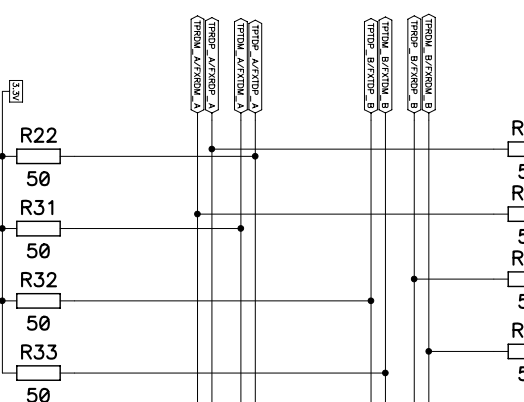
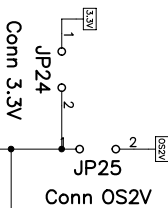
## Decoupling caps

The decoupling capacitors are placed on the bottom layer. Close to Phy

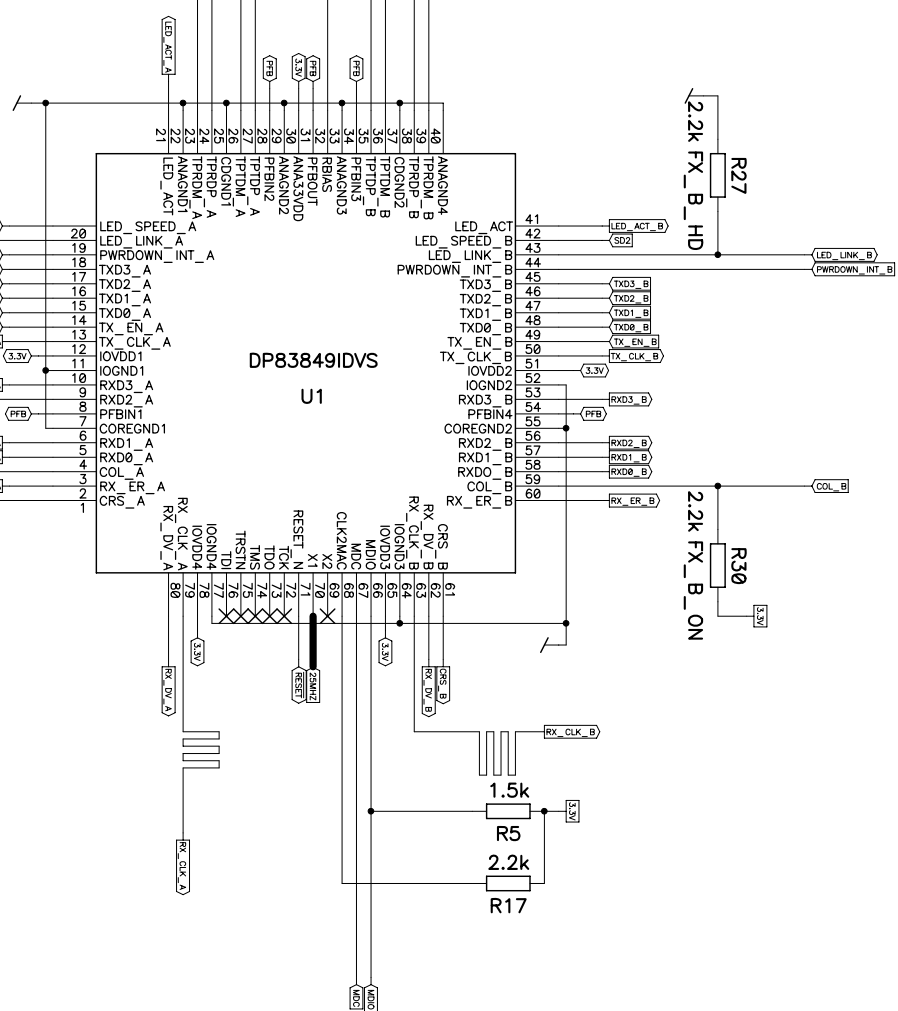


## Phy Matching Network

This resistors should be close to the PHY



4.87k



2.2k FX\_A\_HD

2.2k FX\_A\_ON  
2.2k CLK2MAC\_DIS

Frame capture board  
Interface  
HAUTE ECOLE VALAISANNE

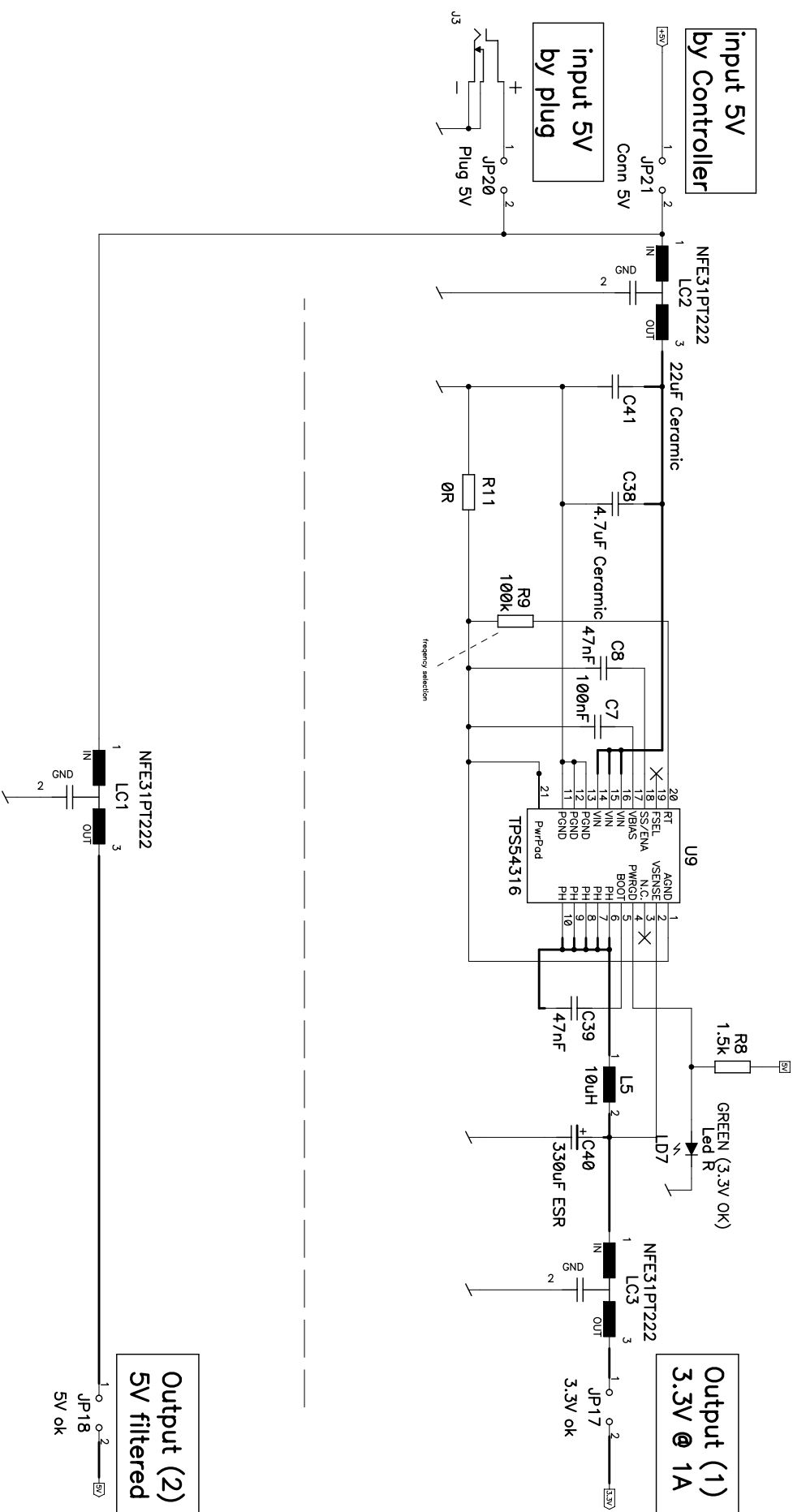
| DES | 28.03.07             | zss |
|-----|----------------------|-----|
| REV | V1.0                 |     |
| PHY | {Path} Interface.sch |     |
| 4/7 |                      |     |



## Power Supply Parts

Is this R sufficient?

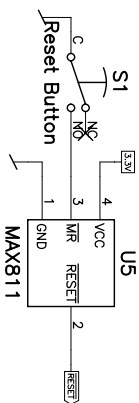
|              |          |
|--------------|----------|
| PWRGD = 0    | V < 90%  |
| PWRGD = HI-Z | V >= 90% |



|                                  |     |          |               |
|----------------------------------|-----|----------|---------------|
| Frame capture board<br>Interface | DES | 28.03.07 | zos           |
|                                  | REV | V1.0     |               |
| HAUTE ECOLE VALAISANNE           | 6/7 | {{Path}} | Interface.sch |

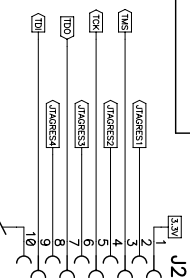
## Board reset

This IC is NOT decoupled by a capacitor, to detect small power fault.  
The reset button resets the board and the JTAG.

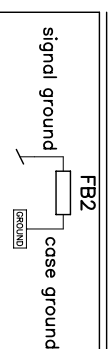


## JTAG Connector

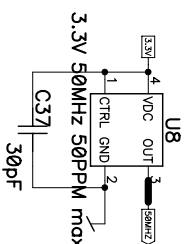
Caution, are the right pin's connected???



## Ground



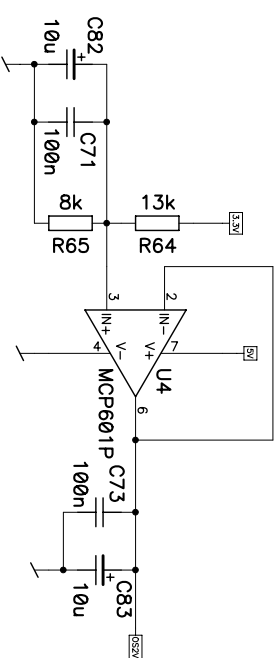
## Phy clk



Caution Quartz with +−50PPM,

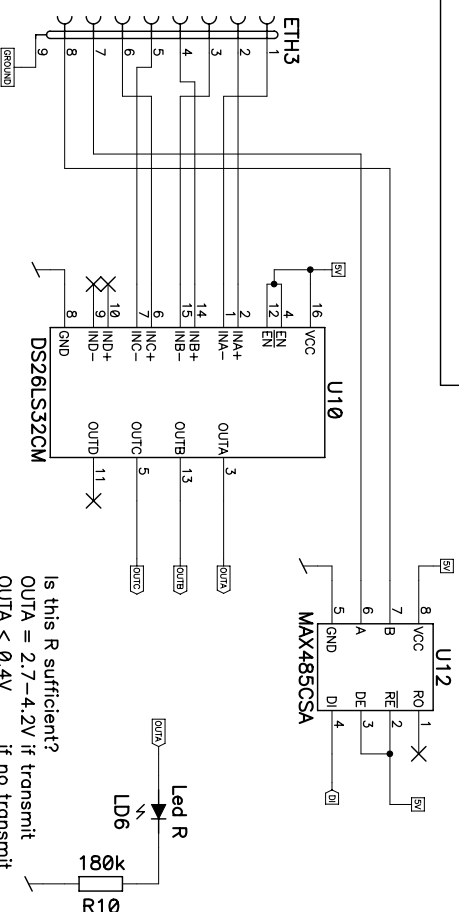
## Analog Ground

Analog Ground for a offset of ~1.2V  
Caution OP in SMD Version!!!!!!!



## RS422 Client bloc

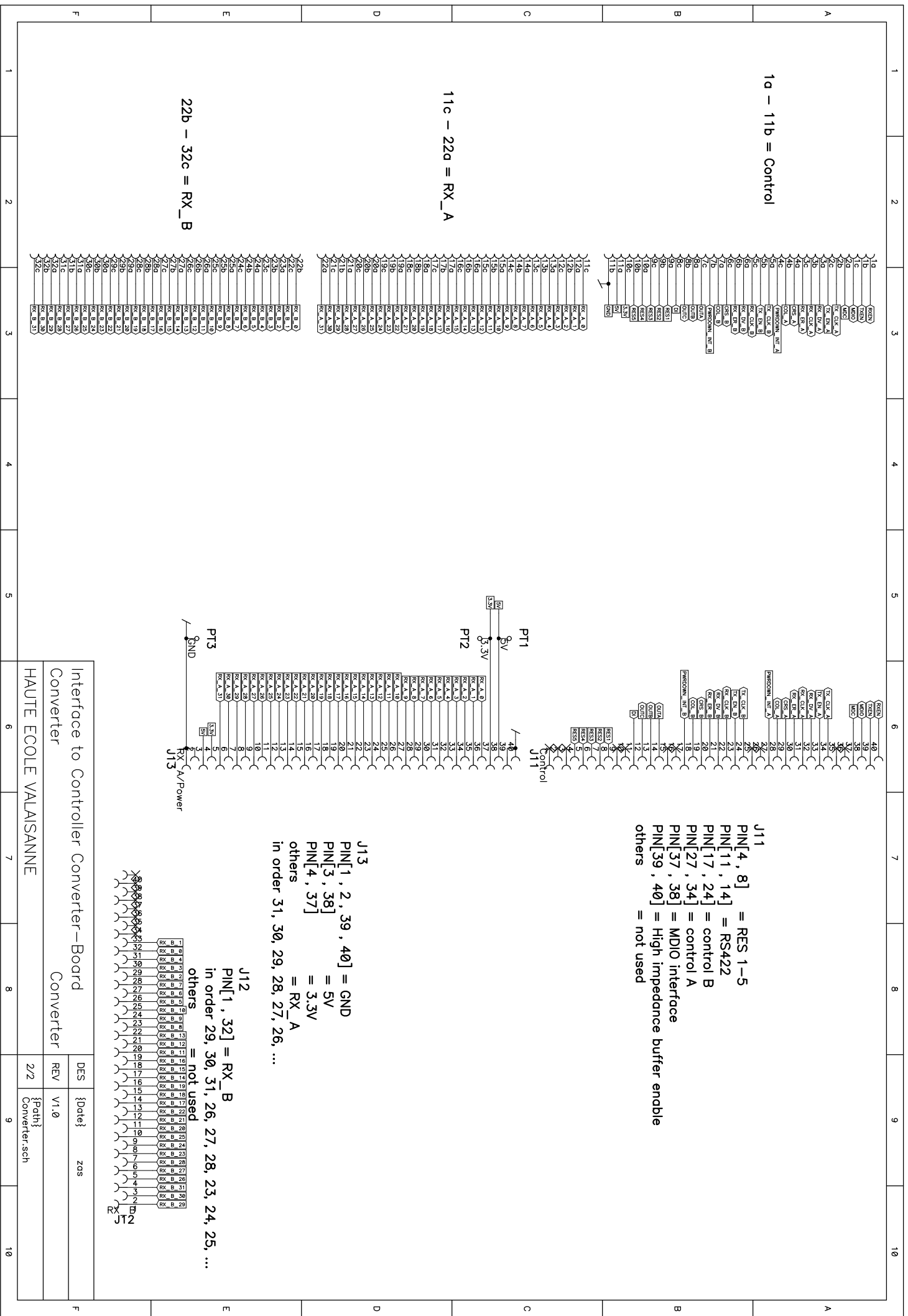
The led shows the transmit and receive actions.  
Normal: 1–2 Port C; 3–4 Port D; 5–6 Port A  
Done: 1–2 Port A; 3–4 Port B; 5–6 Port C



Is this R sufficient?  
OUTA = 2.7–4.2V if transmit  
OUTA < 0.4V if no transmit

| Frame capture board    |  |  | Divers |                         |
|------------------------|--|--|--------|-------------------------|
| Interface              |  |  | REV    | V1.0                    |
| HAUTE ECOLE VALAISANNE |  |  | 7/7    | {Path}<br>Interface.sch |
|                        |  |  | DES    | 28.03.07 zos            |
|                        |  |  | REV    |                         |

## **Appendix 3: Schematic Converter**



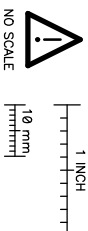
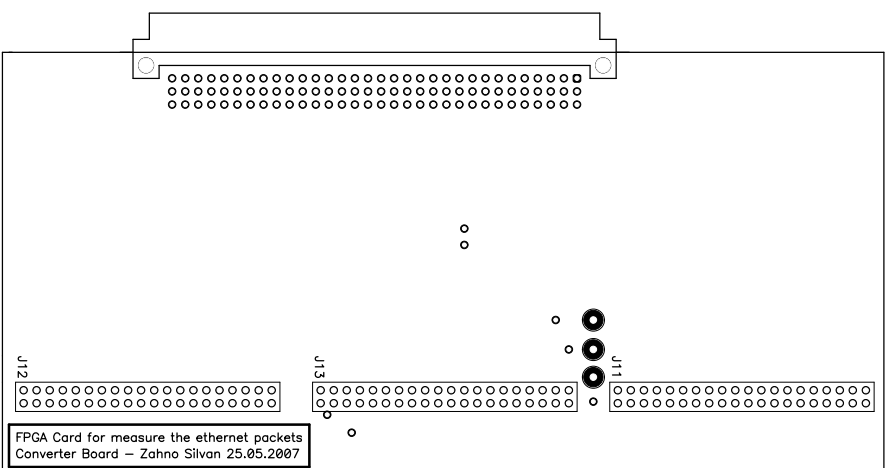












|                                                                 |     |          |
|-----------------------------------------------------------------|-----|----------|
| [FPGA Card for measure the ethernet packets]<br>Converter-Board | DES | 25.05.07 |
|                                                                 | REV | V1.0     |

## **Appendix 4: CoreMP7 development kit users guide**

---

# ***CoreMP7 Development Kit***

*User's Guide*



---

## **Actel Corporation, Mountain View, CA 94043**

© 2006 Actel Corporation. All rights reserved.

Printed in the United States of America

Part Number: XXXXXXXX.X

Release: July 2006

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Actel.

Actel makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability or fitness for a particular purpose. Information in this document is subject to change without notice. Actel assumes no responsibility for any errors that may appear in this document.

This document contains confidential proprietary information that is not to be disclosed to any unauthorized person without prior written consent of Actel Corporation.

### **Trademarks**

Actel and the Actel logo are registered trademarks of Actel Corporation.

Adobe and Acrobat Reader are registered trademarks of Adobe Systems, Inc.

All other products or brand names mentioned are trademarks or registered trademarks of their respective holders.

---

# Table of Contents

|   |                                                                       |     |
|---|-----------------------------------------------------------------------|-----|
|   | Introduction . . . . .                                                | 5   |
|   | Document Contents . . . . .                                           | 5   |
|   | Document Assumptions . . . . .                                        | 5   |
| 1 | Contents and System Requirements . . . . .                            | 7   |
|   | Development Kit Contents . . . . .                                    | 7   |
|   | System Requirements . . . . .                                         | 7   |
| 2 | Hardware Components . . . . .                                         | 9   |
|   | CoreMP7 Evaluation Board . . . . .                                    | 9   |
|   | Detailed Board Description and Usage . . . . .                        | 9   |
|   | PLL Parts/Usage on M7A3P/E . . . . .                                  | 11  |
|   | Programming the Development Kit with a FlashPro3 Programmer . . . . . | 14  |
| 3 | Setup and Self Test . . . . .                                         | 25  |
|   | Software Installation . . . . .                                       | 25  |
|   | Hardware Installation . . . . .                                       | 25  |
|   | Programming the Test File . . . . .                                   | 25  |
| 4 | Actel CoreMP7 Design Flow . . . . .                                   | 27  |
|   | CoreMP7 System Creation . . . . .                                     | 27  |
|   | FPGA Design Creation and Verification . . . . .                       | 29  |
|   | FPGA Design Implementation . . . . .                                  | 30  |
|   | FPGA Programming Software . . . . .                                   | 31  |
|   | Microprocessor Design Creation and Programming . . . . .              | 31  |
| 5 | Quickstart Tutorial . . . . .                                         | 33  |
|   | Actel CoreConsole 1.1 . . . . .                                       | 34  |
|   | Actel Libero IDE v7.1 . . . . .                                       | 49  |
|   | ARM RealView Developer Kit – Actel Edition . . . . .                  | 86  |
|   | Running the Reversi Game via the On-Chip Debugger . . . . .           | 104 |
| A | M7A3PE600 and M7A3P1000 FG484 Package Connections . . . . .           | 105 |
|   | 484-Pin FGBGA Package . . . . .                                       | 106 |

---

## Table of Contents

|          |                                                            |            |
|----------|------------------------------------------------------------|------------|
| <b>B</b> | <b>Board Schematics . . . . .</b>                          | <b>129</b> |
|          | Top-Level View . . . . .                                   | 129        |
|          | CoreMP7 Schematics . . . . .                               | 129        |
| <b>C</b> | <b>Signal Layers . . . . .</b>                             | <b>143</b> |
| <b>D</b> | <b>Product Support . . . . .</b>                           | <b>151</b> |
|          | Customer Service . . . . .                                 | 151        |
|          | Actel Customer Technical Support Center . . . . .          | 151        |
|          | Actel Technical Support . . . . .                          | 151        |
|          | Website . . . . .                                          | 151        |
|          | Contacting the Customer Technical Support Center . . . . . | 152        |
|          | <b>Index . . . . .</b>                                     | <b>153</b> |



---

# Introduction

Thank you for purchasing the Actel CoreMP7 Development Kit.

This guide provides the information required to easily evaluate the CoreMP7 intellectual property (IP) core and M7A3P/E devices.

The CoreMP7 Development Kit software includes a base set of common IP for use in your embedded system. The CoreMP7 Evaluation Board also includes additional hardware to facilitate your system development; however, additional purchases may be required to use certain hardware found on the development board, such as the 10/100 Ethernet, USB 1.1, or CAN 2.0A/B interfaces.

## Document Contents

[Chapter 1 – Contents and System Requirements](#) describes the contents of the CoreMP7 Development Kit.

[Chapter 2 – Hardware Components](#) describes the components of the CoreMP7 Evaluation Board.

[Chapter 3 – Setup and Self Test](#) describes how to set up the CoreMP7 Evaluation Board and how to perform a self test.

[Chapter 4 – Actel CoreMP7 Design Flow](#) introduces the design flow for CoreMP7 using Actel CoreConsole®, Actel Libero® Integrated Development Environment (IDE), and ARM® RealView Developer Kit.

[Chapter 5 – Quickstart Tutorial](#) illustrates a sample Verilog design for the CoreMP7 Evaluation Board.

[Appendix A – M7A3PE600 and M7A3P1000 FG484 Package Connections](#) provides a table listing the board connections.

[Appendix B – Board Schematics](#) provides illustrations of the CoreMP7 Evaluation Board.

[Appendix C – Signal Layers](#) provides illustrations of the six signal layers of the CoreMP7 Evaluation Board.

[Appendix D – Product Support](#) describes Actel support services.

## Document Assumptions

This user's guide assumes the following:

- You intend to use Actel Libero IDE and ARM RealView Developer Kit.
- You have installed and are familiar with Actel Libero IDE v7.0 and ARM RealView Developer Kit v2.2, or later versions of either suite.
- You are familiar with Verilog.
- You are familiar with PCs and the Windows operating system.



---

# Contents and System Requirements

This chapter details the contents of the CoreMP7 Development Kit and lists the power supply and software system requirements.

## Development Kit Contents

The CoreMP7 Development Kit includes the following:

- CoreMP7 Evaluation Board
- Actel Libero IDE Gold
- Actel CoreConsole IP Deployment Platform
- Actel SoftConsole GNU-based C compiler with basic debugger and FlashPro3 JTAG support
- CoreMP7 User's Guide and Tutorial
- CD-ROM with design examples
- Universal 9 V DC power supply providing output up to 2 A  
CUI, Inc. Part Number: DTS090220U-P5P-SZ
- Actel FlashPro3 Programmer (optional, depending on kit ordered)

For the CD-ROM contents, review the *ReadMe.doc* file at the top level of the CD-ROM.

## System Requirements

The system requirements for Actel Libero IDE and ARM RealView Developer Kit are as follows:

- 1.0 GHz Pentium-class processor
- 750 MB hard disk space
- 256 MB RAM
- CD-ROM drive
- USB 1.1 (USB 2.0 recommended)
- Windows 2000 SP4 or Windows XP SP2



---

# Hardware Components

This chapter describes the hardware components of the CoreMP7 Evaluation Board.

## CoreMP7 Evaluation Board

[Figure 2-1 on page 10](#) shows a top-level view of the CoreMP7 Evaluation Board. The board consists of the following:

- Wall-mount power supply connector with switch and LED indicator
- Switches to select from 1.5 V, 2.5 V, and 3.3 V  $V_{CCI}$  (I/O Bank) voltages on banks 4–7 (for the M7A3PE600) or banks 3–4 (for the M7A3P1000)
- 10-pin, 0.1"-pitch programming connector compatible with Altera connections
- 48 MHz oscillator and 32 kHz oscillator for real-time clock (RTC) calculations
- Eight LEDs driven by outputs from the device
- Jumpers allowing disconnection of all external circuitry from the FPGA
- One monostable pulse generator switch
- Eight switches providing input to the device
- Two RS-232 serial interfaces
- Two 10/100 Ethernet interfaces (only populated on boards with M7A3P1000)
- One Controller Area Network (CAN) 2.0B serial interface
- One USB 1.1 serial interface

For further information, refer to [“M7A3PE600 and M7A3P1000 FG484 Package Connections” on page 105](#) and [“Board Schematics” on page 129](#).

## Detailed Board Description and Usage

The CoreMP7 Evaluation Board has various advanced features that are covered in later sections of this chapter. The Development Kit version can be identified as the one that has the FPGA soldered directly to the board.

A block diagram of the CoreMP7 Evaluation Board is shown in [Figure 2-1 on page 10](#) and will facilitate understanding of the more detailed schematics shown in [“Board Schematics” on page 129](#).

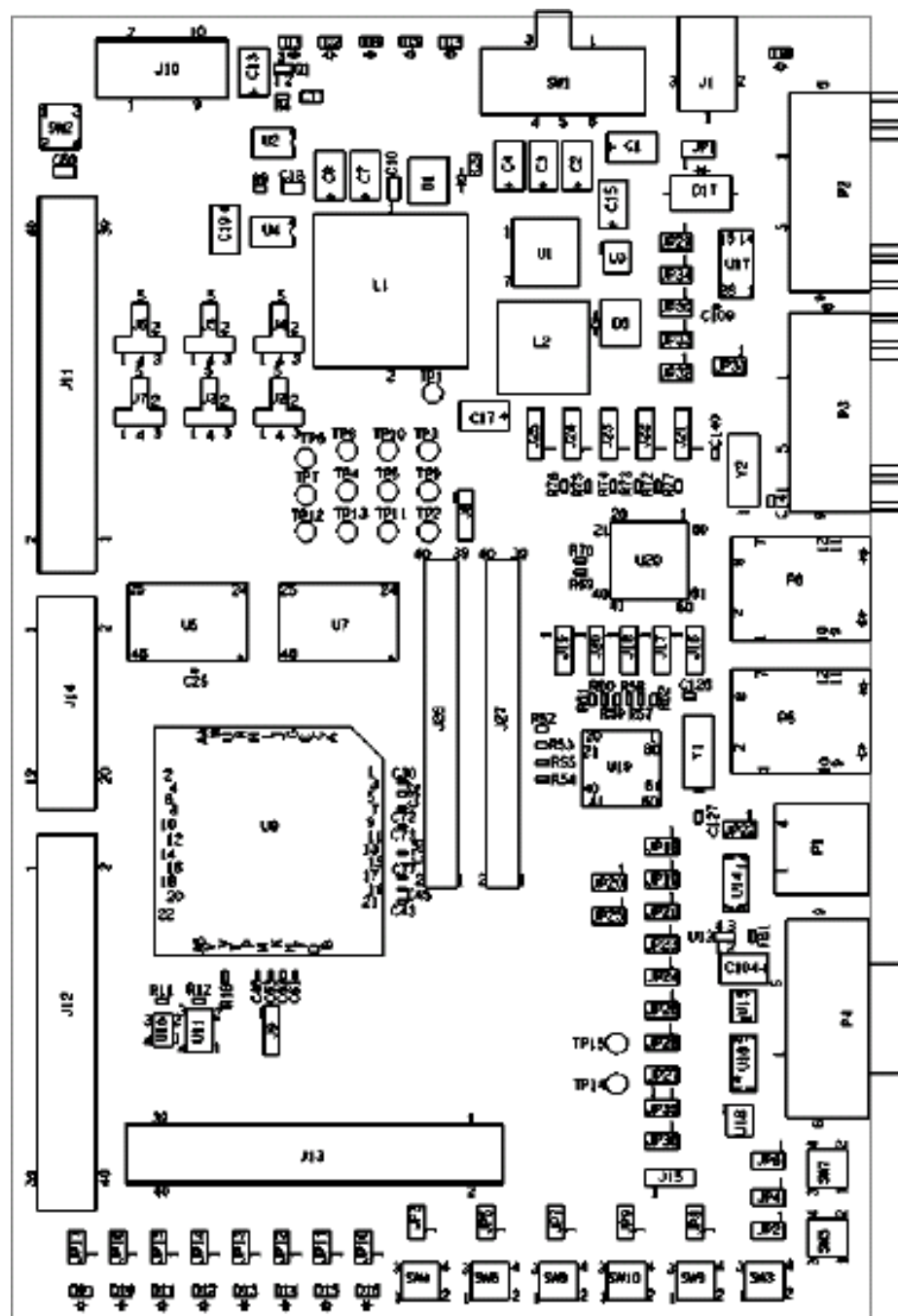


Figure 2-1. CoreMP7 Evaluation Board Top-Level View

Full schematics are available on the Development Kit tutorial CD-ROM supplied with the kit. The schematics are also available for download from the [Actel website](#). The dedicated electronic version of the schematics can be enlarged to a far greater degree than can be shown in the printed version of this manual; hence, the interested reader is referred to the dedicated schematics for the appropriate level of detail.

## PLL Parts/Usage on M7A3P/E

### Instructions for PLL Activation on the CoreMP7 Evaluation Board

To use the PLLs on the CoreMP7 Evaluation Board, power must be applied to their respective analog supply rails. For the west side middle PLL, known as PLF, the VCCPLF line must be connected to VCC, which is held at 1.5 V. The same is true of VCCPLC for the PLL on the east side, known as PLC. These voltages are not connected by default on the board for three reasons:

- The PLC analog voltage rails are not available on M7A3P devices, only in the M7A3PE family; only the west side PLL, namely PLF, is available on M7A3P devices. On M7A3P devices, the remaining pins are used as general purpose I/Os. The same board is used for M7A3PE and M7A3P devices.
- The aim is to demonstrate the lowest possible power consumption for the part. Perpetually powering the PLL lines would not achieve the lowest power.
- It is easy to connect the appropriate pins together when desired. This is why the pins are available on the jumper-based headers.

A variety of valid connections is possible. Three examples are as follows:

- For PLF, connect pin M6 (VCCPLF) to VCC via jumper JP42.
- For PLC, connect pin M18 (VCCPLC) to VCC via jumper JP44.
- For PLA, connect pin F7 (VCCPLA) to VCC via jumper JP40

**Note:** PLA, PLB, PLD, and PLE are only available on M7A3PE devices.

To facilitate use, Actel supplies jumpers with selected production versions of the kit to allow users to quickly connect and disconnect these voltage supply rails. If a user has lost the jumpers or has a kit without jumpers, it is a simple matter of soldering short, insulated connecting wire to the appropriate header pins on the corresponding PLL jumper block.

## Power Supplies

A 9 V power supply is provided with the Development Kit (Figure 2-2). There are many power supply components on the Evaluation Board to illustrate the many ways that differing voltage banks may be used with M7A3P and M7A3PE technology. These voltage banks are not all required for general use of the M7A3P silicon. They are provided for illustrative purposes only.

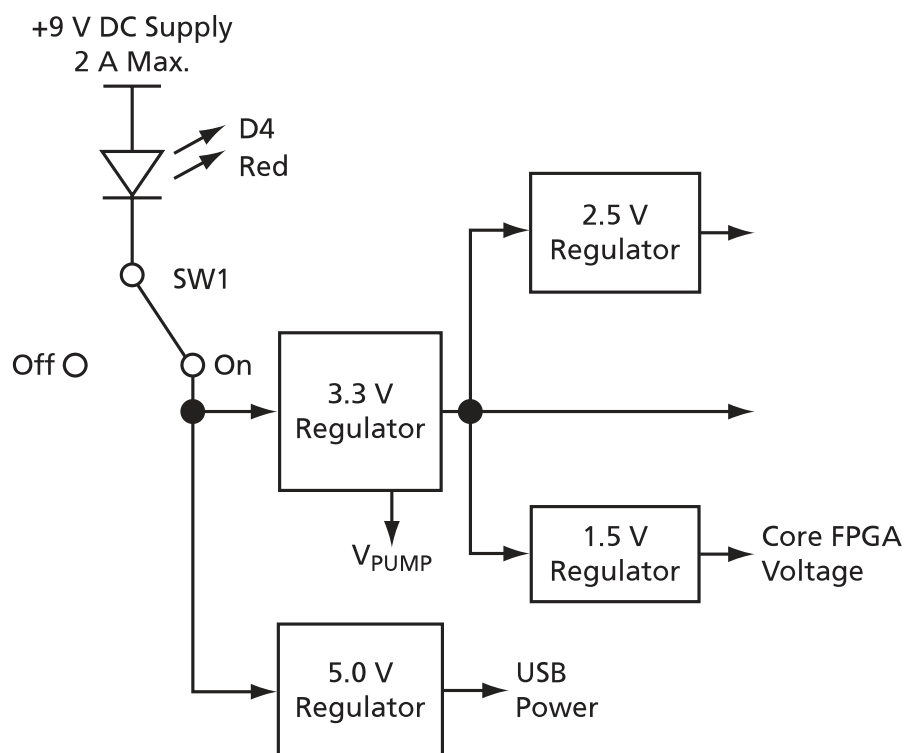


Figure 2-2. Power Supply Block Diagram

To use the CoreMP7 Evaluation Board with a wall-mount power supply, use the switching brick power supply provided with the kit.

The external +9 V center-positive power supply provided to the board via connector J1 goes to a voltage regulator chip, U1. As soon as the external voltage is connected to the board, the red “power applied” LED, D4, illuminates to indicate that an external supply has been connected. As soon as switch SW1 is moved to the ON position, the disabling ground signal is removed from pin 7 of U1, and the regulator begins to provide power at its output.

The switching voltage regulator (U1) provides a dedicated 3.3 V supply at its output. The board’s 3.3 V supply is used to feed separate regulators that deliver 1.5 V (via U2) and 2.5 V (via U4). The 1.5 V supply is required for the core voltage of the M7A3P/E family, and the 2.5 V supply is required for demonstrating LVDS extended I/O bank capability.



The presence of these voltages is indicated by the illumination of three green LEDs (D2, D3, and D7) at the top middle of the board. Each LED is labeled with the voltage it represents and its component identifier. All three voltages are selectable on I/O banks 4–7 on the M7A3PE device.

**Note:** Only M7 ProASIC3E devices have eight I/O banks. M7 ProASIC3 devices have four I/O banks—one per side of the FG484 package.

The 3.3 V supply can also be used to provide the  $V_{PUMP}$  programming voltage.  $V_{PUMP}$  can be provided to the chip during programming by applying a FlashPro3 programmer to the J10 interface and selecting  $V_{PUMP}$  from the FlashPro v4.0 (or later) programming software. The  $V_{PUMP}$  voltage can also be provided directly to the chip from the board. Leave the JP39 jumper in place to apply the 3.3 V supply to the  $V_{PUMP}$  pin (U17 on the FG484 package).

**Note:** If both FlashPro3 and the board are selected to provide  $V_{PUMP}$ , the connection on the board will override, as FlashPro3 will detect that a voltage is available, issue an information message in the programming software, and then tristate the  $V_{PUMP}$  output pin, allowing the board to provide all the power.

The board must be powered up during programming, as the chip needs its core voltages provided, and  $V_{JTAG}$  must be detected by the FlashPro3 programmer before it can set its JTAG signal voltages to the correct level.

USB has its own dedicated 5 V power supply, all components of which (including the regulator U3) are marked on the circuit board in a boxed area to indicate which components on the PCB are associated with which tasks. A green LED (D5) representing 5 V supply availability is located at the top middle of the board.

The external +9 V power supply is rated at 2 A maximum. In the first of the schematics shown in “Board Schematics” on page 129, it can be seen that the 3.3 V supply is rated at 5 A maximum. The derived power supplies of 1.5 V and 2.5 V are each rated at 2 A maximum, and the USB 5 V power supply is rated at 500 mA, as shown in Figure B-3 on page 132. As such, the derived supplies cannot all be working at their maximum current outputs simultaneously. The maximum ratings are given for the individual regulator ICs and cannot be added together.

Both U1 (LM2678S-3.3) and U3 (LM2674M-5.0) are rated for an input voltage range of +8 V to +40 V, so a wide range of power supplies can be used with the board with no concern about over-voltage conditions occurring from inadvertent usage of the wrong power supply. However, the user should take care to ensure that the voltage provided is positive at the center pin of the J16 connector and grounded on the outside.

**Note:** Greater heating of the regulator chips will be observed with higher voltages. It is therefore recommended that only the included power supply or an equivalent substitute be used with the Development Kit. The included power supply has been rated for this board, including any Actel daughter cards that may be attached to the board.

## Programming the Development Kit with a FlashPro3 Programmer

The same board is used for all CoreMP7 Development Kits. The COREMP7-E600-DEV-KIT board is fitted with a M7A3PE600-FG484 device, and the COREMP7-1000-DEV-KIT board is fitted with a M7A3P1000-FG484 device. Further, there are two additional variations of the CoreMP7 Development Kit: COREMP7-E600-DEV-KIT-FP3 and COREMP7-1000-DEV-KIT-FP3. The only difference between these two is the designator -FP3, which indicates that the kit includes the FlashPro3 programmer.

### Connecting the FlashPro3 Programmer to the Board

**To connect the FlashPro3 programmer to the board:**

1. Connect the FlashPro3 programmer to your computer via the USB cable.
2. Follow the instructions in the [FlashPro User's Guide](#) (software v4.0 or later) for installing the software and connecting to FlashPro3. The amber (yellow) power LED on the FlashPro3 should be illuminated at this stage. If it is not, recheck the procedure given in the [FlashPro User's Guide](#) until you obtain steady illumination of the amber power LED.
3. Make sure the board power switch SW1 is in the OFF position and only the red external power LED is illuminated on the board.
4. Connect the FlashPro3 programmer to the board via the 10-pin programming cable supplied with the FlashPro3 programmer. The connector to use on the board is labeled FP3 JTAG (J10) and has a keyed header. The pin 1 location on the cable, indicated by the red ribbon running along the side of the cable, will be on the left side as it enters the board.

After connecting the FlashPro3 programmer, you can verify communication by checking **Device Info** in the FlashPro software. The M7A3P/E details will be shown in the software log window. If you suspect a JTAG communication problem, try changing the  $V_{JTAG}$  voltage. To overcome noise, higher values usually work better, but all values should work with the supplied programming cable (6" in length) connected to just one board.

### Programming or Reprogramming the Example Design

On the Development Kit CD, you will find a *Designer* directory containing a STAPL file for programming the target design. Select the *TOP M7A3PE6.STP* file (for M7A3PE600 parts) or the *TOP M7A3P1K.STP* file (for M7A3P1000 parts) from the CD and use that as the STAPL file in the FlashPro software. Selecting **Program** will erase, program, and verify the part.

## **Jumpers for Isolating Switches, LEDs, and Other Components from the FPGA**

Many jumpers are provided on the board to allow the user to disconnect various switch combinations and LEDs from the FPGA I/O banks. All such jumpers are shown in the schematic in [Figure B-8 on page 137](#) and are labeled on the top-layer silkscreen as JP\*, where \* is a number. All jumpers are also labeled with the FPGA I/O pin number to which they are connected; e.g., JP29, for the TX0 connection of the RS-232 transmitter to the FPGA, is labeled “F18,” which indicates that it is connected to pin F18. Similarly, SW4 has a jumper above it, JP3, that is labeled “T5,” indicating that SW4 is connected to pin T5 of the FPGA when the jumper is in place.

Disconnecting jumpers JP2–JP9 causes the push button switches (SW3–SW10, respectively) to be disconnected from the FPGA so that I/O pins T4, T5, R6, R5, U2, U3, P6, and P7 can be used for other purposes. Disconnecting the eight jumpers, JP10–JP17, causes the eight LEDs (D9–D16) to be disconnected from FPGA I/O pins R4, P5, R2, T2, P2, N2, N6, and N7, respectively.

The push-button switch SW2 (labeled RESET#), meant for applying a reset pulse, is connected to pin W15, a chip-wide global. Again, all labeling is clearly shown on the silkscreen. This flexibility is useful for experimentation with designs of your own choosing and in connecting other external equipment to the board for development purposes.

## LED Connections

Eight LEDs are connected to the device via jumpers. If the jumpers are in place, the device I/O can drive the LEDs. The LEDs change based on the output as follows:

- A '1' on the output of the device lights the LED.
- A '0' on the output of the device switches off the LED.
- An unprogrammed or tristated output may show a faintly lit LED.

**Note:** If the I/O voltage of Bank 5 (on A3PE, set by J6) or Bank 2 (A3P, set by J6) is not at least 2.5 V, the LEDs will not illuminate. A setting of 1.8 V on the voltage bank will cause extremely faint illumination.

Table 2-1 lists the jumper and device connection associated with each LED.

Table 2-1. LED Device Connections

| LED | Jumper | Device Connection |
|-----|--------|-------------------|
| D9  | JP17   | U9 pin N7         |
| D10 | JP16   | U9 pin N6         |
| D11 | JP15   | U9 pin N2         |
| D12 | JP14   | U9 pin P2         |
| D13 | JP13   | U9 pin T2         |
| D14 | JP12   | U9 pin R2         |
| D15 | JP11   | U9 pin P5         |
| D16 | JP10   | U9 pin P4         |

To use the device I/O for other purposes, remove the jumpers.

## Switch Connections

Eight switches are connected to the device via jumpers. If the jumpers are in place, the device I/O can be driven by the switches listed in [Table 2-2](#).

- Pressing a switch drives a '1' onto the associated device I/O pin. The '1' continues to be driven while the switch is in place.
- Releasing a switch drives a zero onto the device I/O pin.

[Table 2-2](#) lists the jumper and device connection associated with each switch.

Table 2-2. Switch Device Connections

| Switch | Jumper | Device Connection |
|--------|--------|-------------------|
| SW3    | JP2    | U9 pin T4         |
| SW4    | JP3    | U9 pin T5         |
| SW5    | JP4    | U9 pin R6         |
| SW6    | JP5    | U9 pin R5         |
| SW7    | JP6    | U9 pin U2         |
| SW8    | JP7    | U9 pin U3         |
| SW9    | JP8    | U9 pin P6         |
| SW10   | JP9    | U9 pin P7         |

## CoreUARTapb RS-232 Implementation

The CoreMP7 Development Kit includes two RS-232 ports that can be used for communication between the embedded microprocessor and a common serial port, as found on a PC or other RS-232-compatible device. To use either of the RS-232 ports, jumpers must be in place to connect the FPGA to the on-board RS-232 transceiver. The jumpers used for the RS-232 connections can be found in [Table 2-3](#).

The primary RS-232 port (P2) has lines to support RTS/CTS flow control in addition to TXD and RXD.

**Note:** Currently, CoreUARTapb does not support hardware RTS/CTS handshaking. If this functionality is needed, it must be implemented in software. The default configuration has a jumper shorting RTS0 and CTS0 together, disconnecting them from the FPGA I/Os and creating a loopback connection, similar to the implementation found on the secondary RS-232 port.

Table 2-3. RS-232 Connections

| Signal | Jumper | Device Connection |
|--------|--------|-------------------|
| TX0    | JP29   | U9 pin B11        |
| RX0    | JP32   | U9 pin G21        |
| RTS0   | JP30   | U9 pin K17        |
| CTS0   | JP33   | U9 pin J19        |
| TX1    | JP31   | U9 pin C11        |
| RX1    | JP34   | U9 pin K18        |

## Core10/100 Ethernet Implementation

Core10/100 is an Ethernet Media Access Controller (MAC) that connects Local Area Networks (LANs) at data rates of 10 or 100 Mbps (see [Figure 2-3](#)). It has a Media Independent Interface (MII) for physical connection and implements Carrier Sense Multiple Access with Collision Detection (CSMA/CD) algorithms, per IEEE 802.3. Ethernet is a common standard used in computer, communications, industrial, and other applications.

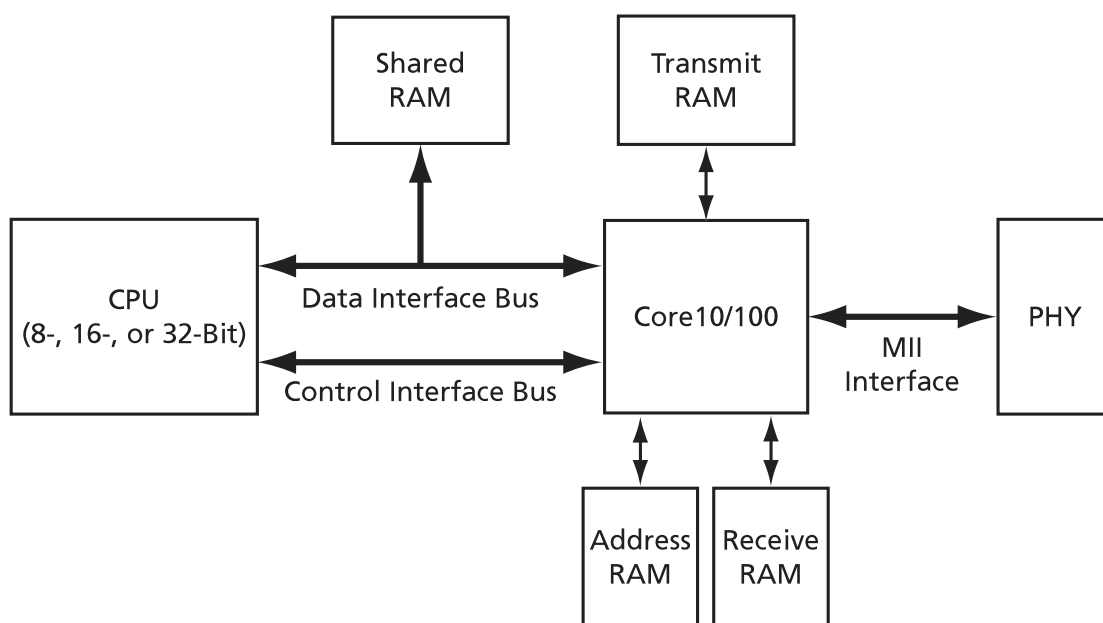


Figure 2-3. Overview of a Typical Core10/100 System

Detailed Core10/100 information is available in the *Ethernet Media Access Controller Core10/100* datasheet at [http://www.actel.com/ipdocs/Core10100\\_DS.pdf](http://www.actel.com/ipdocs/Core10100_DS.pdf).

The MII interface to Core10/100 works with most Ethernet PHY chips. Due to the analog requirements of an Ethernet PHY, such cannot be implemented in an Actel FPGA.

The CoreMP7 Evaluation Board supports dual Ethernet connections. An AM79C874VI from Advanced Micro Devices (AMD) is used for each PHY (U19 and U20). See [Table 2-4 on page 20](#) for details on the connections between the FPGA and each PHY via J26 and J27, which are connection/disconnection points for PHY0 and PHY1, respectively.

**Note:** The dual Core10/100 Ethernet interfaces are only populated on boards based on the M7A3P1000 device.

Table 2-4. 10/100 Ethernet Connections

| PHY Signal   | Jumper | Device Connection (J26/J27) |
|--------------|--------|-----------------------------|
| MDIO         | 1      | U12/U12                     |
| MDC          | 3      | T12/T12                     |
| TXD0         | 5      | V10/V12                     |
| TXD1         | 7      | U9/V11                      |
| TXD2         | 9      | U10/R12                     |
| TXD3         | 11     | T10/R11                     |
| TX EN        | 13     | AB7/AA9                     |
| TX ER        | 15     | AB6/AA10                    |
| EXT IN CLK 0 | 17     | Y7/AA8                      |
| RXD0         | 19     | Y6/AA7                      |
| RXD1         | 21     | U9/AB9                      |
| RXD2         | 23     | V8/AB8                      |
| RXD3         | 25     | AA6/W9                      |
| RX DV        | 27     | AA5/W8                      |
| RX ER        | 29     | AB5/Y10                     |
| EXT IN CLK 1 | 31     | AB4/W10                     |
| COL          | 33     | AA4/U11                     |
| CRS          | 35     | Y4/T11                      |
| RST#         | 37     | W15/W15                     |
| N/C          | 39     | N/C                         |

## USB

The CoreMP7 Evaluation Board includes a Fairchild Semiconductor USB1T11AM USB transceiver. The USB standard specifies support for multiple device connections, allowing up to 127 unique devices. Further, the Fairchild transceiver supports the transmitting and receiving of serial data at both full-speed (12 Mbps) and low-speed (1.5 Mbps) data rates. Implementation of the Serial Interface Engine (SIE) is required to use the USB interface present on the Evaluation Board. Information on the SIE can be found on the USB Implementers Forum at <http://www.usb.org>.



## CompanionCore CAN 2.0B Implementation

The CAN bus is a communication standard with multi-master capability, error detection and correction, and broad industry acceptance. The CAN bus was designed for the automobile industry, but CAN has recently been appearing in non-traditional applications. The CAN bus comprises two signals to which all networked devices are connected, thus allowing communication between multiple devices. The reliability and error detection is handled by a series of arbitrations, (not) acknowledges, and CRC checks.

Table 2-5 details the connections between the M7A3P/E FPGA and the onboard CAN transceiver (U18). The CoreMP7 Evaluation Board is also equipped with LEDs (D18 and D19) connected to the TXD and RXD lines of the CAN bus. Therefore, when data is being transmitted or received, the respective LED will blink.

If network termination is needed (typically for CAN baud rates greater than 100 kbps), shorting JP38 (CAN TERM) inserts a 120  $\Omega$  resistor between CAN-H and CAN-L.

Table 2-5. CAN Device Connections

| CAN Signal | Jumper | Function / Device Connection |
|------------|--------|------------------------------|
| CAN TXD    | JP35   | U9 pin K20                   |
| CAN RXD    | JP36   | U9 pin J22                   |
| CAN TERM   | JP37   | Enables termination          |
| CAN SHLD   | JP38   | Enables shield ground        |

## Clock Circuits

The CoreMP7 Evaluation Board has two clock circuits: a 48 MHz oscillator and a 32 kHz oscillator.

### 48 MHz Oscillator

The 48 MHz oscillator on the board is a 30 ppm–stability crystal module that provides more than adequate performance and can be connected to a general purpose I/O (pin W12) or a chip-wide global (pin W17) using a jumper.

### 32 kHz Oscillator

The 32 kHz oscillator on the board is a 30 ppm–stability crystal module that will provide enough accuracy to perform RTC calculations and is hardwired to a chip-wide global (pin V16).

## Memory

### Flash

The CoreMP7 Evaluation Board includes two STMicroelectronics M29W800DT Flash memory chips, totaling 2 MB, which can be arranged in either a  $1\text{M} \times 16$  or a  $512\text{K} \times 32$  configuration. The Flash memory is intended for use as executable program storage for the embedded microprocessor; however, it can also be used as nonvolatile memory for the storage of system constants and parameters.

### SRAM

The CoreMP7 Evaluation Board includes two GSI Technology GS8001BT Synchronous SRAM modules, totaling 2 MB, which can be arranged in either a  $1\text{M} \times 16$  or a  $512\text{K} \times 32$  configuration. The SRAM memory is used for the embedded microprocessor stacks (both hardware and software) and for dynamic system data.

## Headers

There are three headers (J11, J12, and J13) present on the CoreMP7 Evaluation Board intended for use as general purpose I/O. These pins are tied to the various chip-wide global signals in the I/O banks as well as dedicated general purpose I/Os. See the schematics in [“Board Schematics” on page 129](#) for further information.

## Test Points

All test points on the board are fitted with small test loops. These test points are labeled on the silkscreen as TP1, TP2, etc. All such test points are also labeled on the silkscreen with the voltage expected to be observed at that test point or the I/O pin to which the test point is connected. Each voltage will be either 3.3 V, 2.5 V, 1.5 V, or GND. When measuring the voltage at a test point with a DVM (digital voltage multimeter), the ground lead should be connected to a test point labeled GND, and the voltage lead should be connected to the voltage to be tested. All voltage labels on the board are relative to a 0 V ground reference (GND).

## Board Layers

The complete board design and manufacturing files are included on the Development Kit CD. The board file is in Allegro format, which will allow a user to create the appropriate Gerbers and other board views as needed. Pictures of the board layers are also included in [“Signal Layers” on page 143](#). For your convenience, high-resolution PDFs of these layers are also provided on the Development Kit CD.

The board is fabricated with six copper layers. The layers are arranged as follows, from top to bottom:

- Layer 1 – Top signal layer
- Layer 2 – Ground plane
- Layer 3 – Signal layer 3
- Layer 4 – Signal layer 4
- Layer 5 – Power plane
- Layer 6 – Bottom signal layer

Refer to the diagrams in [“Signal Layers”](#) on page 143.



---

# Setup and Self Test

This chapter outlines how to set up and test the CoreMP7 Evaluation Board.

## Software Installation

The CoreMP7 Development Kit includes the Libero IDE software suite (version 7.0). For Libero IDE software installation instructions, refer to the *Actel Libero IDE / Designer Installation and Licensing Guide for Software v6.1* at <http://www.actel.com/documents/install.pdf>.

The CoreMP7 Development Kit also includes the Actel SoftConsole GNU-based C compiler and debugger, which can be used to program and debug the CoreMP7 program memory through the FlashPro3.

## Hardware Installation

FlashPro3 is required to use the CoreMP7 Development Kit. For software and hardware installation instructions, refer to the *FlashPro v3.3 User's Guide* at <http://www.actel.com/documents/flashproUG.pdf>. FlashPro3 is also used with SoftConsole to program and debug the Flash program memory on the CoreMP7 Evaluation Board.

If you are using the ARM RealView Developer Kit, you will need to use the ARM RealView ICE Micro Edition (RVI-ME) supplied with it to program and debug the Flash program memory on the CoreMP7 Evaluation Board. For software and hardware installation instructions, refer to the documentation included on the ARM RealView installation CDs.

## Programming the Test File

To retest the evaluation board at any time, use the test program to reprogram the board. Use the *TEST M7A3PE6.stp* file with an M7A3PE600-FG484 fitted on the board. Use *TEST M7A3P1K.stp* with an M7A3P1000-FG484 fitted on the board.

The test design is currently implemented for the M7A3PE600 die size. It is possible to recompile the design for other device sizes. For information about retargeting the device, refer to the *Designer User's Guide* at <http://www.actel.com/documents/designerUG.pdf>. The design files are available under *SelfTest* on the Development Kit CD.

For instructions on programming the device using FlashPro3, refer to the *FlashPro User's Guide* at <http://www.actel.com/documents/flashproUG.pdf>.

The Flash memory on the board can be programmed using either FlashPro3 (if you are using SoftConsole) or the RVI-ME (if you are using the RealView Developer Kit). For information on programming the memory with RealView and the RVI-ME refer to *ARM Application Note #110: Flash Programming with RealView Debugger* at <http://www.arm.com/pdfs/AN110.zip>.



---

# Actel CoreMP7 Design Flow

The CoreMP7 design flow consists of the two paths, shown in [Figure 4-1 on page 28](#):

- FPGA development – the creation of the CoreMP7 system based on the Actel M7 FPGAs
- Executable code development – the creation of software programs that will execute on the embedded microprocessor core

The CoreMP7 design flow has five main components:

- CoreMP7 system creation
- FPGA design creation and verification
- FPGA design implementation
- FPGA programming
- Microprocessor design creation and programming

## CoreMP7 System Creation

CoreConsole is a system-level development tool and IP deployment platform that greatly simplifies the task of assembling and connecting IP for implementation in Actel FPGAs. It enables you to select IP components from a database supplied by Actel and graphically “stitch” them together to build a processor-based System-Level Integration (SLI) design. When the design is complete, the RTL (and other files needed to implement the design) can be generated and imported into the familiar and proven design flow of the Actel Libero IDE software. CoreConsole also generates a testbench for the SLI design that you can build to assist in verification.

Refer to the [CoreConsole User's Guide](http://www.actel.com/documents/CoreConsole_UG.pdf) at [http://www.actel.com/documents/CoreConsole\\_UG.pdf](http://www.actel.com/documents/CoreConsole_UG.pdf) for more information on using CoreConsole.

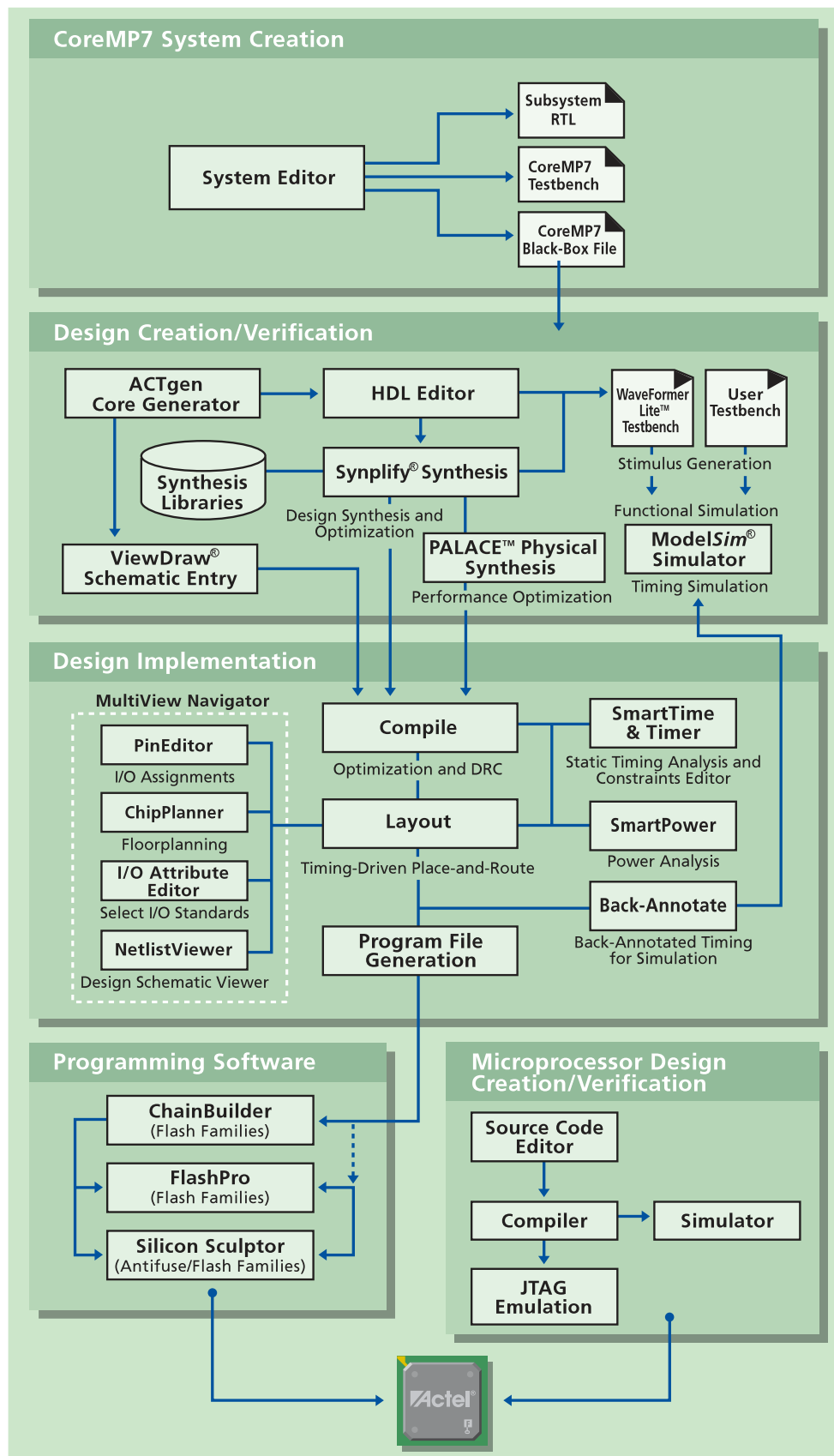


Figure 4-1. Design Flow Paths



## FPGA Design Creation and Verification

Design entry consists of writing HDL or capturing a schematic representation of the design and performing functional simulations with a testbench.

### Design Capture

For schematic capture, Libero IDE uses ViewDraw® for Actel, which includes a schematic editor. The schematic editor provides a graphical entry method to capture designs. ViewDraw for Actel is the Libero IDE integrated schematic entry vehicle, supporting mixed-mode entry, in which HDL blocks and schematic symbols can be mixed.

The ViewDraw WIR file is automatically created after using the **Save + Check** command. This file is used to create the structural HDL netlist.

For more information on using ViewDraw for Actel, refer to the *Libero User's Guide* at <http://www.actel.com/documents/liberoUG.pdf>.

### Adding SmartGen Macros

Use the SmartGen Macro Builder to instantly create customized macros, then use ViewDraw to add these macros to a schematic. Alternatively, add the SmartGen macros in the HDL file.

### Creating and Adding Symbols for HDL Files

Schematic users can encapsulate an HDL block within a block symbol.

#### **To create a symbol:**

1. Right-click the block in the Design Hierarchy window of Libero IDE.
2. Click **Create Symbol**. Libero IDE generates a symbol for the selected HDL block.

The macro is accessible from the components list in ViewDraw for Actel.

### Testbench Generation

To run a simulation, it is necessary to create a testbench and associate it with a project. WaveFormer Lite™ from SynaptiCAD™ is the Libero IDE integrated testbench generator. WaveFormer Lite fits perfectly into Libero IDE, automatically extracting signal information from HDL design files and producing HDL testbench code that can be used with any standard VHDL or Verilog simulator.

WaveFormer Lite generates VHDL and Verilog testbenches from drawn waveforms.

## Pre-Synthesis Simulation

Functional simulation verifies that the logic of a design is functionally correct. Simulation is performed using the Libero IDE integrated simulator, ModelSim® for Actel, which is a custom edition of ModelSim PE integrated into Libero IDE.

ModelSim for Actel is an OEM edition of the Model Technology™ Incorporated (MTI) tools. ModelSim for Actel supports VHDL or Verilog, but it can only simulate one language at a time. It only works with Actel libraries and is supported by Actel.

## Synthesis and Netlist Generation

After entering the design source, synthesize it to generate a netlist. Synthesis transforms the behavioral HDL source into a gate-level netlist and optimizes the design for a target technology. For more detailed information on the above topics, refer to the *Libero IDE v7.0 User's Guide* at <http://www.actel.com/documents/liberoUG.pdf>.

## FPGA Design Implementation

During design implementation, Actel Designer performs place-and-route on the design.

### Place-and-Route

Start Designer from Libero IDE to place-and-route the design.

### Timing Simulation

Perform timing simulation on the design after place-and-route in Designer. Timing simulation requires information extracted and back-annotated from Designer.

### Optional Tools

The tools listed in [Table 4-1](#) provide optional functions that are not required in a basic design. Use these tools to perform static timing analysis and power analysis, customize I/O placements and attributes, and view the netlist. Perform the post-layout (timing) simulation after place-and-route.

Table 4-1. Designer User Tools

| Designer User Tool | Function                                |
|--------------------|-----------------------------------------|
| SmartTime          | Static timing analysis                  |
| SmartPower         | Power analysis                          |
| ChipEdit           | Customize I/O and logic macro placement |
| PinEdit            | Customize I/O placements and attributes |
| Netlist Viewer     | View your netlist and trace paths       |

For more information on the tools described above, refer to the *Designer User's Guide* at <http://www.actel.com/documents/designerUG.pdf>.

## FPGA Programming Software

Program the device with programming software and hardware from Actel or with a supported third-party programming system. Refer to the *Designer User's Guide*, *Silicon Sculptor User's Guide*, and *FlashPro User's Guide* for information about programming an Actel device.

These guides can be found at <http://www.actel.com/techdocs/manuals/default.asp>.

## Microprocessor Design Creation and Programming

There are a large number of third party ARM7 program development tools that can be used with CoreMP7 for the development of software programs that run on the processor. Actel offers several, including the SoftConsole tools (included with the CoreMP7 Development Kit) and the RealView Developer Kit (RVDK). SoftConsole is available for free, and the RVDK can be licensed from Actel for an annual license fee. Although the RVDK has an annual license fee, the RealView C compiler generates significantly more efficient code for CoreMP7 than the SoftConsole GCC compiler.

ARM RealView Developer Kit provides a fully integrated software solution with leading-edge tools for creating efficient software to run on any ARM processor. Servicing all major market segments, RealView Developer Kit provides flexible software tools to meet present and future requirements.



---

# Quickstart Tutorial

This tutorial illustrates a Verilog CoreMP7 design for the CoreMP7 Evaluation Board. This design is created in Actel CoreConsole 1.1, Libero IDE v7.1, and ARM RealView Developer Kit. The steps involved are as follows:

## Actel CoreConsole 1.1

- “Step 1 – Creating the Basic CoreConsole Project”
- “Step 2 – Building the Subsystem within CoreConsole”
- “Step 3 – Reviewing and Generating the CoreConsole Design”

## Actel Libero IDE v7.1

- “Step 1 – Create a New Project”
- “Step 2 – Perform Pre-Synthesis Simulation”
- “Step 3 – Synthesize the Design in Synplify”
- “Step 4 – Perform Post-Synthesis Simulation”
- “Step 5 – Implementing the Design with Actel Designer”
- “Step 6 – Perform Timing Simulation with Back-Annotated Timing”
- “Step 7 – Generating the Programming File”
- “Step 8 – Programming the Device”

## ARM RealView Developer Kit

- “Step 1 – Creating a RealView Project”
- “Step 2 – Compiling the Source Files”
- “Step 3 – Debugging: Simulating/Executing the Compilation”

## Actel CoreConsole 1.1

This tutorial provides step-by-step instructions on how to create a CoreConsole project and generate a CoreConsole design. The tutorial consists of three steps: “[Step 1 – Creating the Basic CoreConsole Project](#)”, “[Step 2 – Building the Subsystem within CoreConsole](#)” on page 43, and “[Step 3 – Reviewing and Generating the CoreConsole Design](#)” on page 46.

**Note:** Before you begin this tutorial, make sure the CoreConsole software is installed.

### Step 1 – Creating the Basic CoreConsole Project

In Step 1, you learn the basic features of CoreConsole by creating a basic CoreConsole project. You will use the Actel CoreConsole IP Deployment Platform tool to develop a skeleton CoreMP7 system. This system can be simulated and synthesized; however, it is too basic for practical use and will be extended in “[Step 2 – Building the Subsystem within CoreConsole](#)”.

#### **To create the CoreConsole project:**

1. Double-click the **Actel CoreConsole 1.1** icon on your desktop to start the program, or select **Start > Programs > CoreConsole > Actel CoreConsole 1.1**.
2. From the **File** menu, select **New**. The New Design window displays, as shown in [Figure 5-1](#).

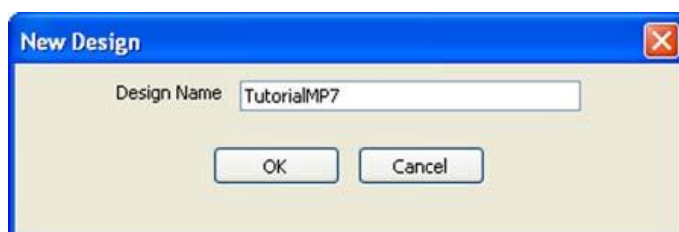


Figure 5-1. New Design Window in CoreConsole

3. Enter your **Design Name**. For this tutorial, name your design “TutorialMP7”.
4. Click **OK** to create your design project.

#### **To add components to your CoreConsole project:**

1. Under the **Components** tab, in the “Components available for selection” section, click **CoreMP7**.
2. Click the **Add** button in the “Selected Component's Details” section. The CoreMP7 component appears in your design.
3. In the “Components available for selection” section, click **CoreMP7Bridge**.
4. Click the **Add** button in the “Selected Component's Details” section.
5. Following the same process as in steps 4–5, add the component **CoreAHB**.

6. Once all three components have been added to the design, it should resemble [Figure 5-2](#).

**Note:** Some of the components are overlapping. To arrange the components neatly, select **Auto Layout** from the **Actions** menu.

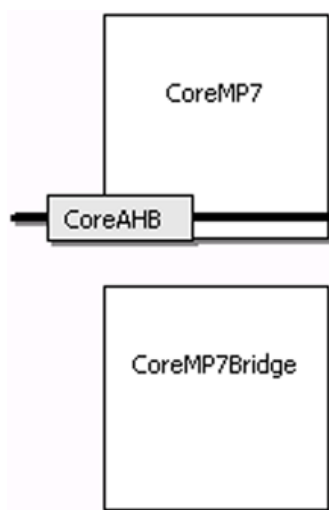


Figure 5-2. CoreConsole Schematic Window before Auto Layout

7. Following the process in steps 4–5, add the **CoreMemCtrl** component. When it appears in the schematic window, you can drag it to the right of CoreMP7 for neater appearance.

**To connect components within your CoreConsole project:**

1. From the **Actions** menu, select **Auto Stitch**. This displays the Auto Stitching window, as shown in Figure 5-3.

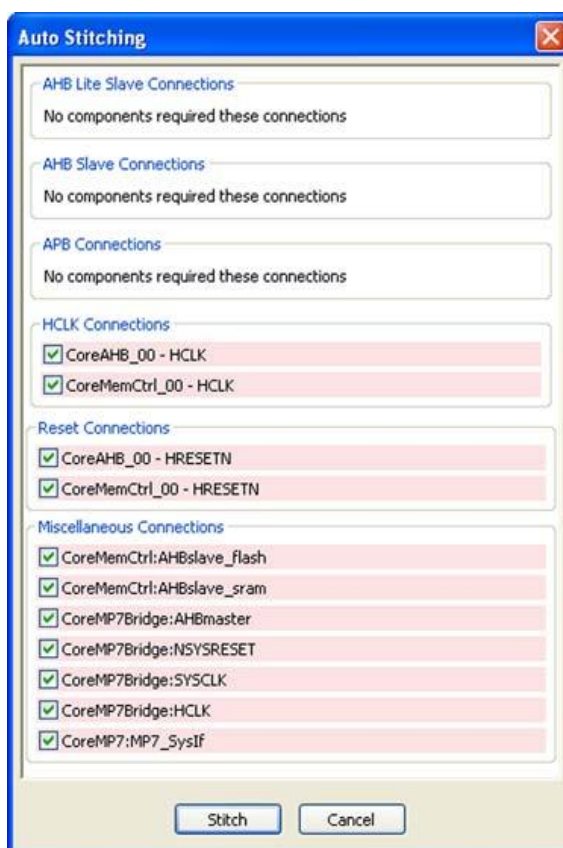


Figure 5-3. CoreConsole Auto Stitching Window

Auto Stitching connects the critical components of the system together. However, you will still need to connect most of the top-level signals manually. Manually connecting signals gives you finite control over the microprocessor's memory map.

2. Confirm that stitching has been enabled for **CoreMP7**, **CoreMP7Bridge**, **CoreAHB**, and **CoreMemCtrl**, as shown in Figure 5-3. Then click the **Stitch** button.



3. Once Auto Stitching is complete, your design should resemble [Figure 5-4](#).

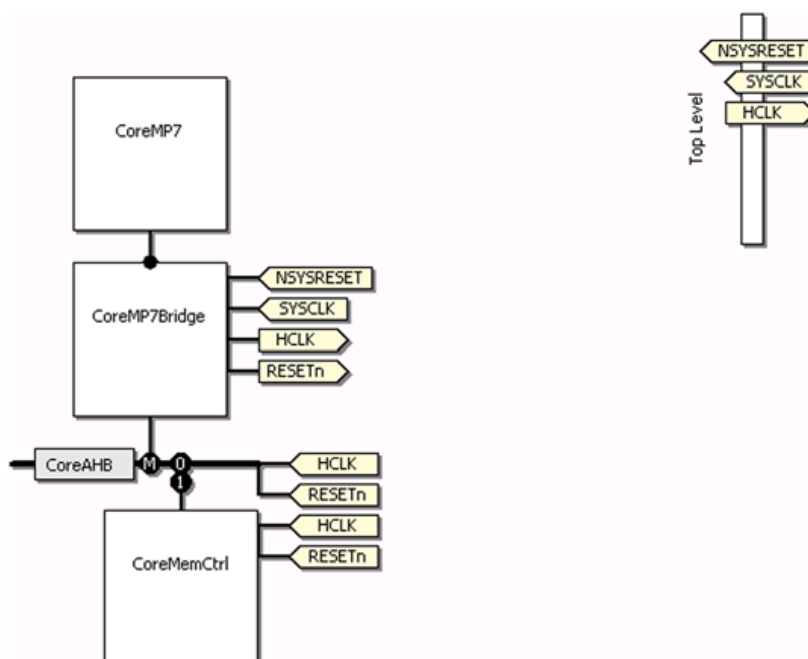


Figure 5-4. CoreConsole after Auto Stitching

The steps in the next section show you how to manually connect signals to the system's top level. The first set of instructions walks you through connecting the ARM7 JTAG interface to the top level. The second set of instructions brings the memory bus (both data and address) to the top level to interface with the Flash and SRAM modules.

4. To make additional connections, float your mouse over one of the components, such as **CoreMP7Bridge**. An options toolbar appears underneath the selected component, as shown in [Figure 5-5](#).



Figure 5-5. Component Options Toolbar

5. Click the **Connect** icon, the first icon from the left, which resembles a power plug. When you have done this, the Configuring Connection dialog box appears, as shown in Figure 5-6.

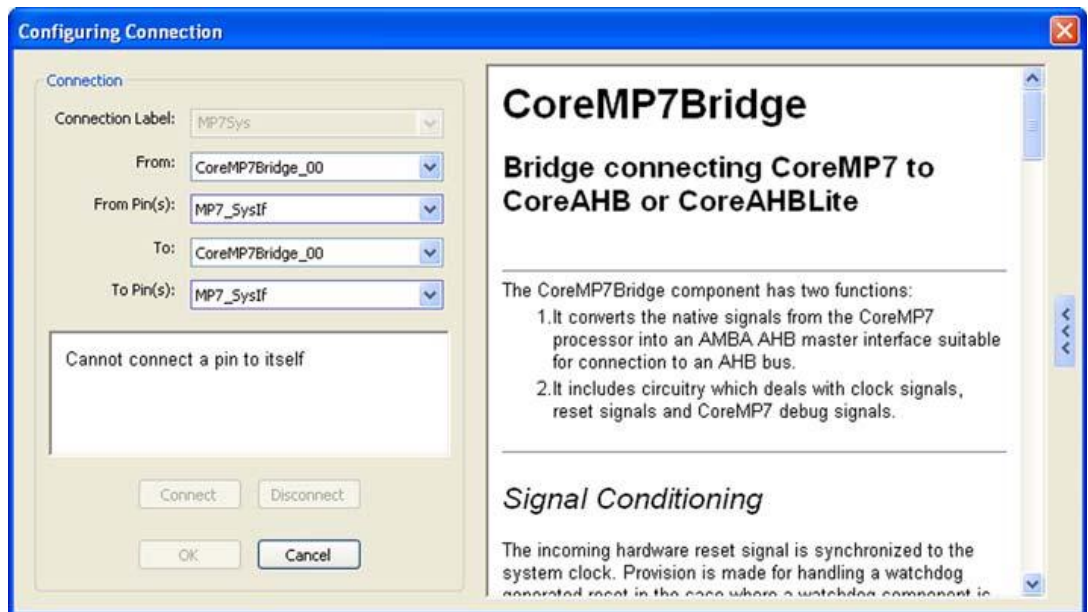


Figure 5-6. CoreConsole Configuring Connection Dialog Box

6. **CoreMP7Bridge** should automatically be selected in the **From** field. If it is not selected, select it from the drop-down menu. Then select **RV ICE If** from the **From Pin(s)** drop-down menu, as shown in Figure 5-7.

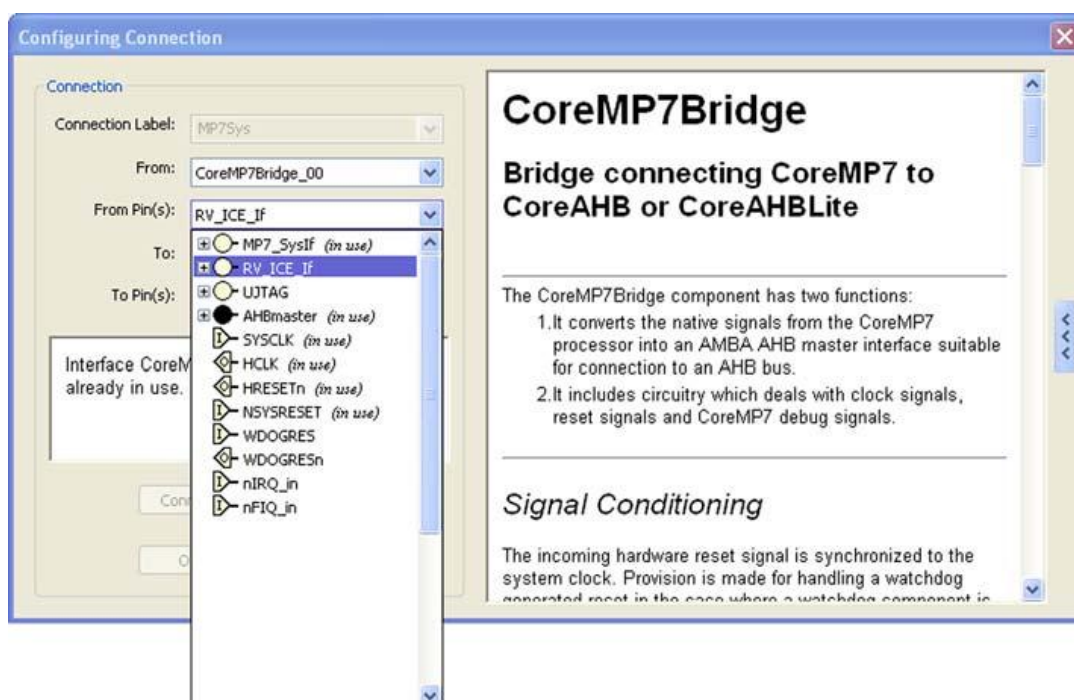


Figure 5-7. Selecting the CoreMP7Bridge RV ICE If Pins

7. Select **Top Level** in the **To** drop-down menu.
8. Enter the signal name “RV” for **Connection Label** and click **Connect**.

- Click **OK**. Your schematic should resemble the one in [Figure 5-8](#).

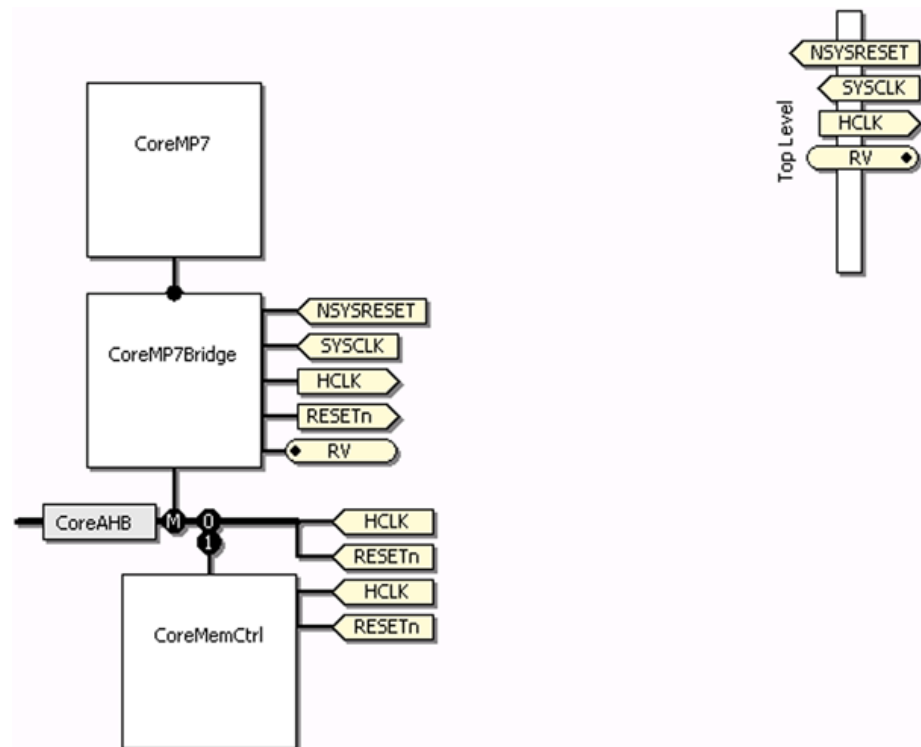


Figure 5-8. CoreConsole Schematic after Connecting ARM JTAG Interface

- Click the **Connect** icon on the floating options menu for the **CoreMemCtrl** component.

11. From the Configuring Connection dialog box, select **ExternalMemoryInterface** in the **From Pin(s)** drop-down menu, as shown in Figure 5-9.

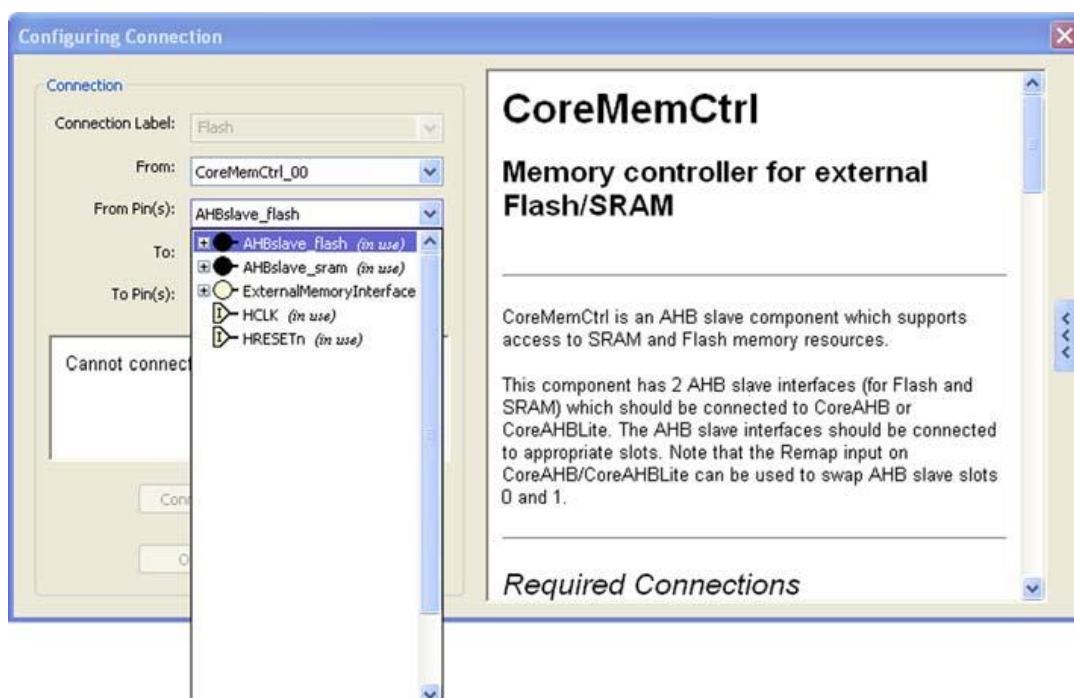


Figure 5-9. Selecting the CoreMemCtrl ExternalMemoryInterface Pins

12. Select **Top Level** in the **To** drop-down menu.
13. Enter the signal name "Mem" for **Connection Label** and click **Connect**.

14. Click **OK**. Your schematic should resemble the one in [Figure 5-10](#).

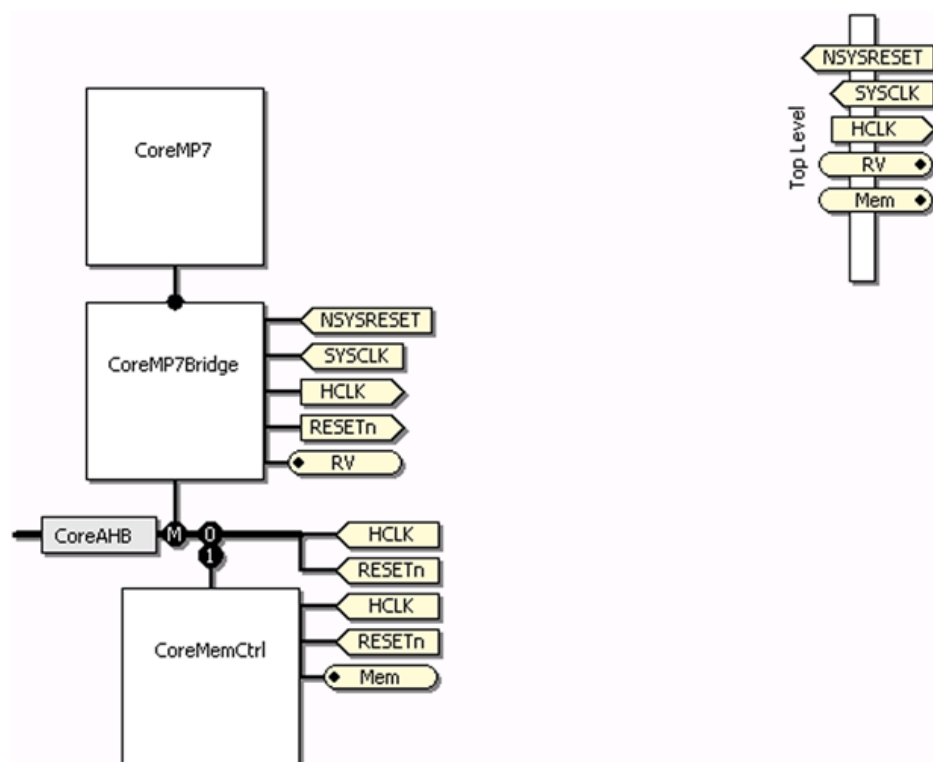


Figure 5-10. CoreConsole Schematic after Connecting the External Memory Interface

## Step 2 – Building the Subsystem within CoreConsole

Before you begin Step 2, you should know how to stitch components together, as taught in Step 1. For more information on stitching components together, review the instructions in “[Step 1 – Creating the Basic CoreConsole Project](#)” on page 34 or the *CoreConsole User Guide*.

### Adding CoreUARTapb to the System

In this section, you will add a common CoreUARTapb component to the subsystem. CoreUARTapb has an APB interface (as opposed to the high-speed AHB interface); therefore, a CoreAPB component is required and will be implemented through a bridge.

1. Add the **CoreAHB2APB**, **CoreAPB**, and **CoreUARTapb** components. For neater appearance, move CoreAHB2APB to the right of CoreMP7Bridge, with CoreAPB below the bridges and CoreUARTapb below CoreAPB.
2. Connect the components as follows:
  - Connect CoreAHB2APB through its AHBslave interface to CoreAHB via the AHBmslave12 interface.
  - Connect CoreUARTapb through its APBslave interface to CoreAPB via the APBmslave3 interface.
  - Connect the CoreUARTapb TX signal to Top Level with the connection name UART TX.
  - Connect the CoreUARTapb RX signal to Top Level with the connection name UART RX.
3. Once completed, select **Auto Stitch** from the **Actions** menu. Confirm that Auto Stitching is configured to operate on CoreAHB2APB and CoreUARTapb, then click the **Stitch** button. CoreConsole will then connect the HCLK and nRESET pins to the components you added, and connect CoreAHB2APB to CoreAPB as a master.

- Once completed, your CoreConsole schematic should look similar to Figure 5-11.

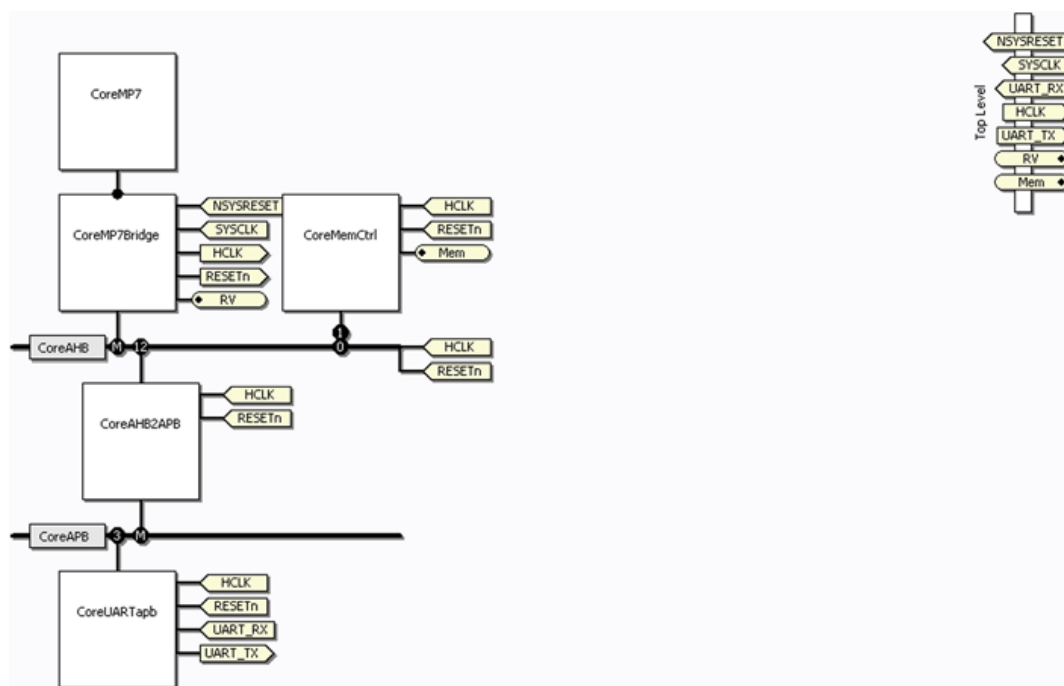


Figure 5-11. CoreConsole Schematic after Connecting CoreUARTapb

**To add the I/O block and system control registers to the system:**

- Add the **CoreGPIO** and **CoreRemap** components. For neater appearance, move CoreGPIO and CoreRemap to the right of CoreUARTapb.
- Connect the components as follows:
  - Connect CoreGPIO through its APBslave interface to CoreAPB via the APBmslave2 interface.
  - Connect CoreRemap through its APBslave interface to CoreAPB via the APBmslave15 interface.
  - Connect the CoreRemap CoreRemapDef pin to Top Level using the signal name ReMapDef.
  - Connect the CoreGPIO dataIn pin to Top Level using the signal name keyPadIn.
  - Connect the CoreGPIO dataOut pin to Top Level using the signal name ledOut.
- Once completed, select **Auto Stitch** from the **Actions** menu.



4. Confirm that Auto Stitching is configured to operate on CoreGPIO and CoreRemap, then click the **Stitch** button.  
CoreConsole will then connect the HCLK and nRESET pins to the components you added.
5. Once completed, your CoreConsole schematic should look similar [Figure 5-12](#).

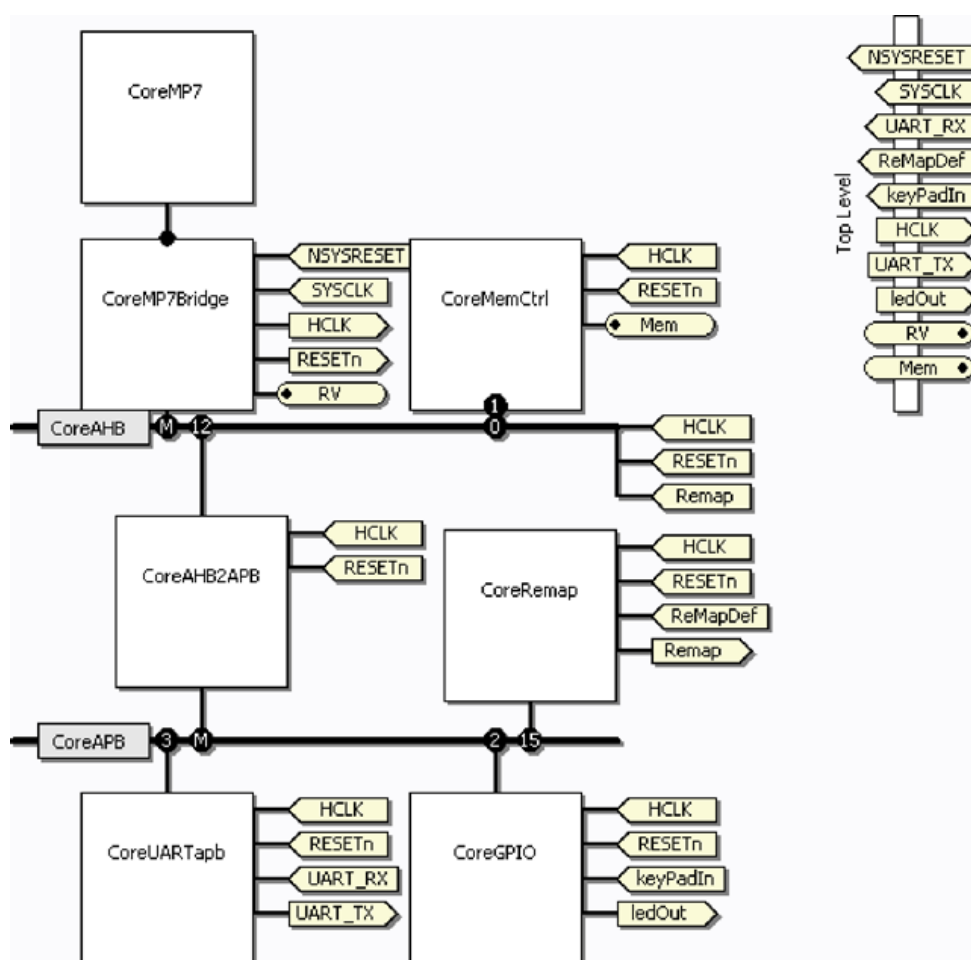


Figure 5-12. CoreConsole Schematic after Connecting CoreGPIO and CoreRemap

## Step 3 – Reviewing and Generating the CoreConsole Design

Even though you can manually view the design connections in a system of this size, this section explores features in CoreConsole for reviewing your design connections.

### **To review the CoreConsole design connections:**

1. From the **View** menu, select **Connections**. The Connections dialog box appears.
2. Examine the connections within the system for accuracy.
3. Click a connection. The appropriate Connection dialog box displays. If necessary, you can make modifications to the connection.
4. Click **OK** to close the Connections window.

### **Displaying Linked Connections**

Another useful tool is **Show Linked Connections**, which enables you to see all of your linked connections at once. This feature is enabled by default, but you can change this from **System Options > Options**.

In the schematic window, position your mouse cursor over HCLK on the Top Level bar and notice all the highlighted linked connections. You can perform this action with all of the system signals.

## Modifying Configuration Settings

Prior to generating the source code necessary for Actel Libero IDE, you must modify the configuration settings.

1. Position your mouse cursor over the **CoreMP7** component and click the **Configure** button (the second button from the left, with the binary digits).

The Configuring CoreMP7 dialog box displays, as shown in Figure 5-13.

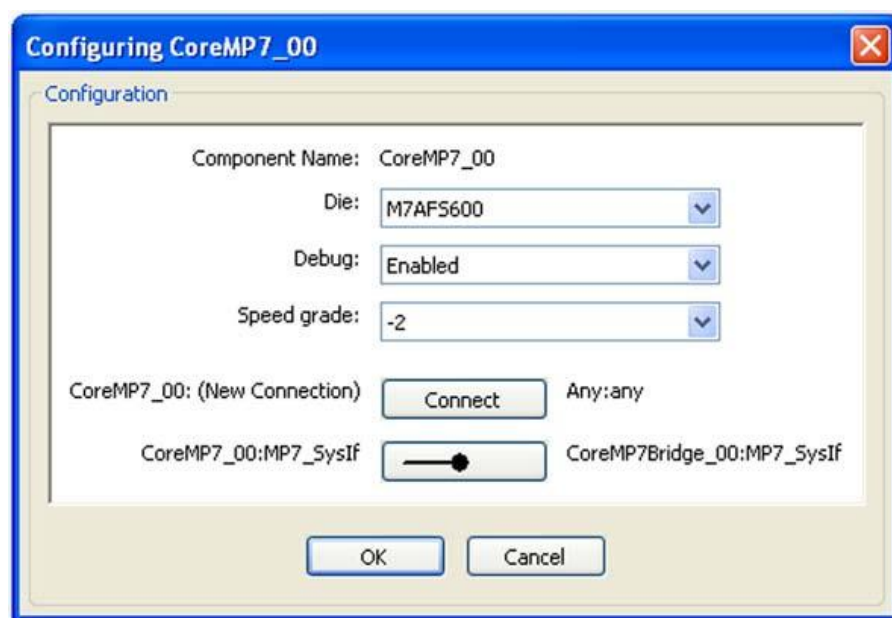


Figure 5-13. Configuring CoreMP7 Dialog Box

2. From the drop-down menu for **Die**, select **M7A3PE600**. This is the device populated on the CoreMP7 Evaluation Board.
3. Click **OK**.

Within the Configuring CoreMP7 dialog box, you can disable the JTAG debug interface, which allows you to select the "fast" version of CoreMP7.

4. Invoke the Configuration dialog boxes for the **CoreMP7Bridge** and **CoreUARTapb** components.
5. In the **Device family** field, select **ProASIC3E** from the drop-down menu and click **OK**.
6. Select the **Generate** tab within the design manager.
7. Select the HDL language preference. This tutorial is based on Verilog, so make sure **Verilog** is selected.
8. Click the **Save & Generate** button.

Before exiting CoreConsole, wait until both progress bars have reached 100% (Figure 5-14). This process can take up to 45 seconds, depending on system complexity and PC resources.

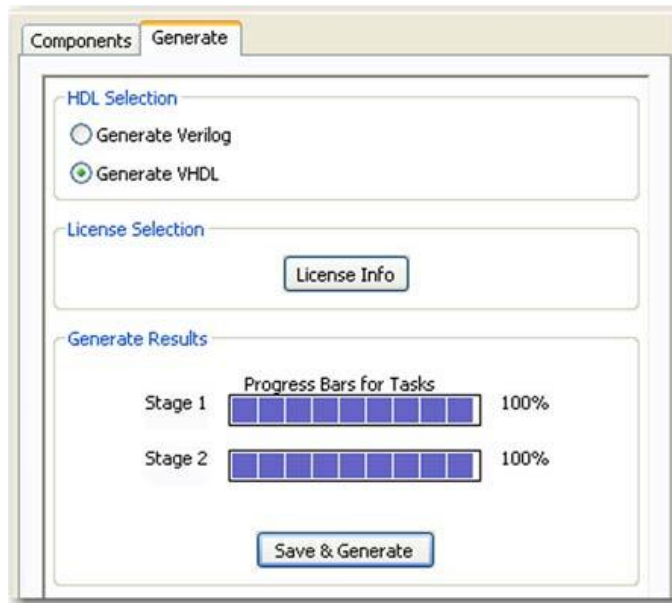


Figure 5-14. CoreConsole Generate Tab

Once the generation phase has successfully completed, you can import your CoreConsole project (i.e., HDL source files and the ARM7 black box) into Actel Libero IDE.

## Actel Libero IDE v7.1

### Step 1 – Create a New Project

This step uses the Libero IDE HDL Editor to enter an Actel CoreMP7 Verilog design.

#### To create the Libero IDE Verilog project:

1. Double-click the **Libero IDE** icon on your desktop to start the program.
2. From the **File** menu, select **New Project**. This displays the New Project Wizard, shown in Figure 5-15.



Figure 5-15. New Project Wizard in Libero IDE

3. Enter your **Project name**. For this tutorial, name your project “ReversiTutorial”.
4. Select your **HDL type**. For this tutorial select **Verilog**.
5. If necessary, in the **Project location** field, click **Browse** to navigate to *C:\Actelprj*. Click **Next** to continue.

6. Select your project **Family**, **Die**, and **Package**. For this tutorial, select **ProASIC3E**, the **M7A3PE600** die, and **484 FBGA** for the package (Figure 5-16).

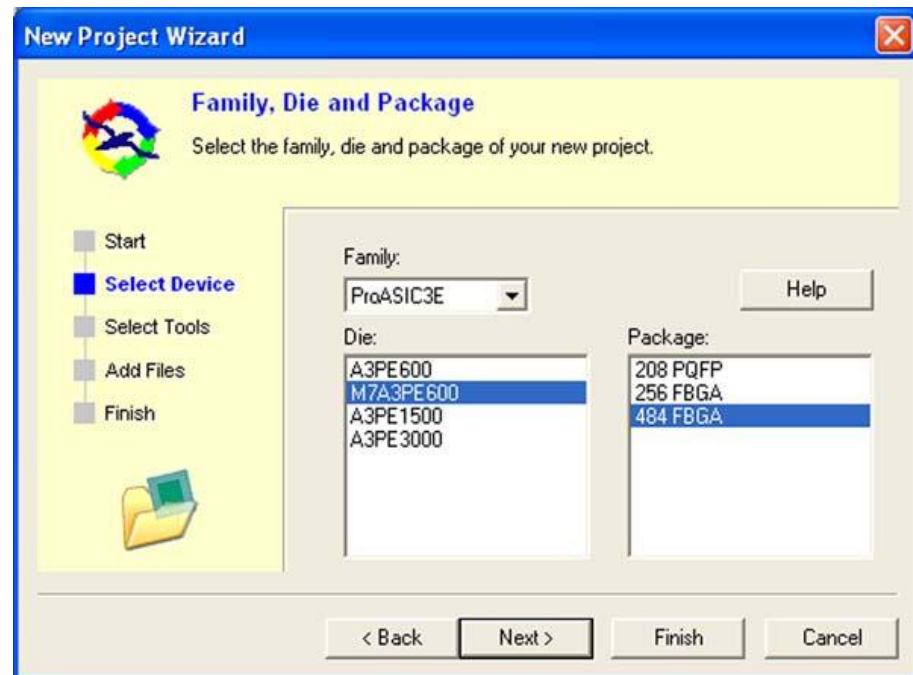


Figure 5-16. Select ProASIC3E, M7A3PE600, and 484 FBGA

7. Click **Next** to select integrated tools in the New Project Wizard (Figure 5-17).



Figure 5-17. Selecting Integrated Tools in the Libero IDE New Project Wizard

8. Click the **Restore Defaults** button to use the default tools included with Libero IDE.

9. Click the **Add** button to add a different Synthesis, Simulation, or Stimulus tool. If you wish to add a tool, Libero IDE opens the **Add Profile** dialog box (Figure 5-18).

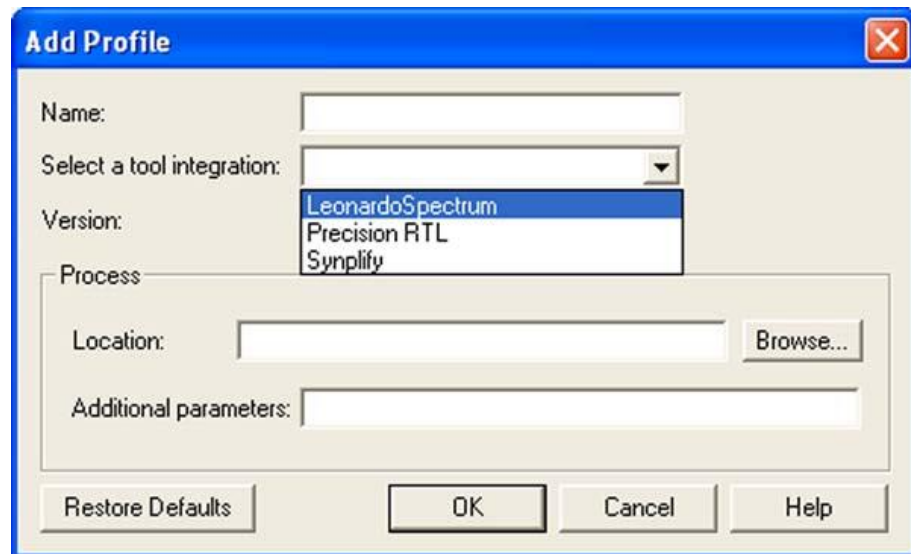


Figure 5-18. Add Profile Dialog Box in Libero IDE

10. Name your profile, select a tool from the list of Libero IDE supported tools, and **Browse** to the location of your tool. Click **OK** to return to the New Project Wizard.
11. After you have selected your tools, click **Next** to continue.



12. Click **Add Files** in the New Project Wizard to add existing project design files. Include any ACTgen cores, CoreConsole Projects, Block Symbol, Schematic, Verilog Source, Implementation, or Stimulus files (Figure 5-19).

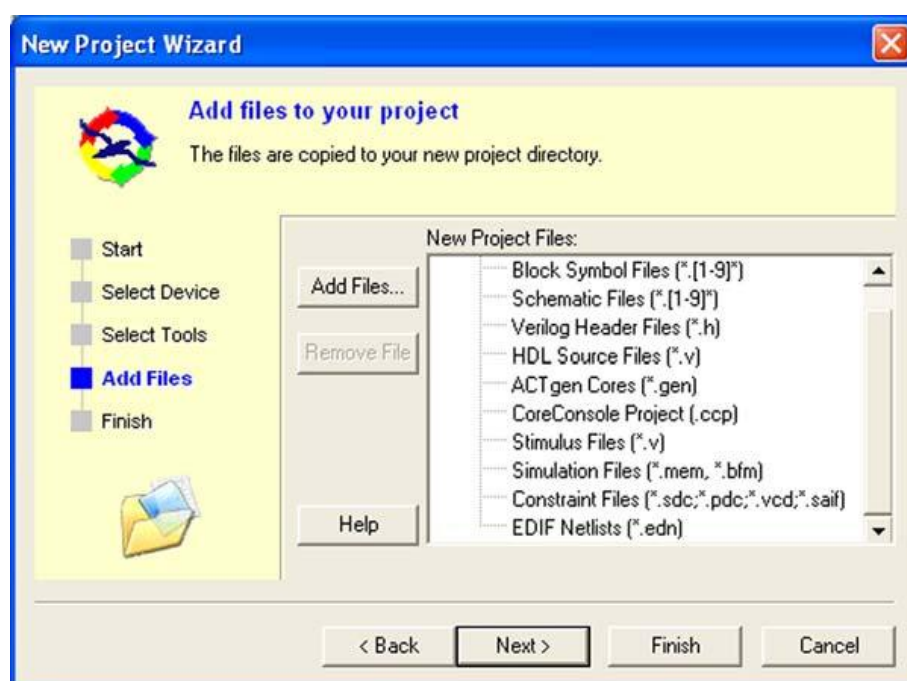


Figure 5-19. Add Files in the Libero IDE New Project Wizard

13. Select the CoreConsole Project file type and click **Add Files**. Browse to your CoreConsole Project created earlier (assuming default installation it will be located in the *C:\CoreConsole\Libero IDE Export\TutorialMP7\* directory) and select the *TutorialMP7.ccp* file, then click **Add**.

The CoreConsole project will be displayed in the project wizard (Figure 5-20). You can add as many files as you like this way. For this project, only the CoreConsole file will be imported with this method.

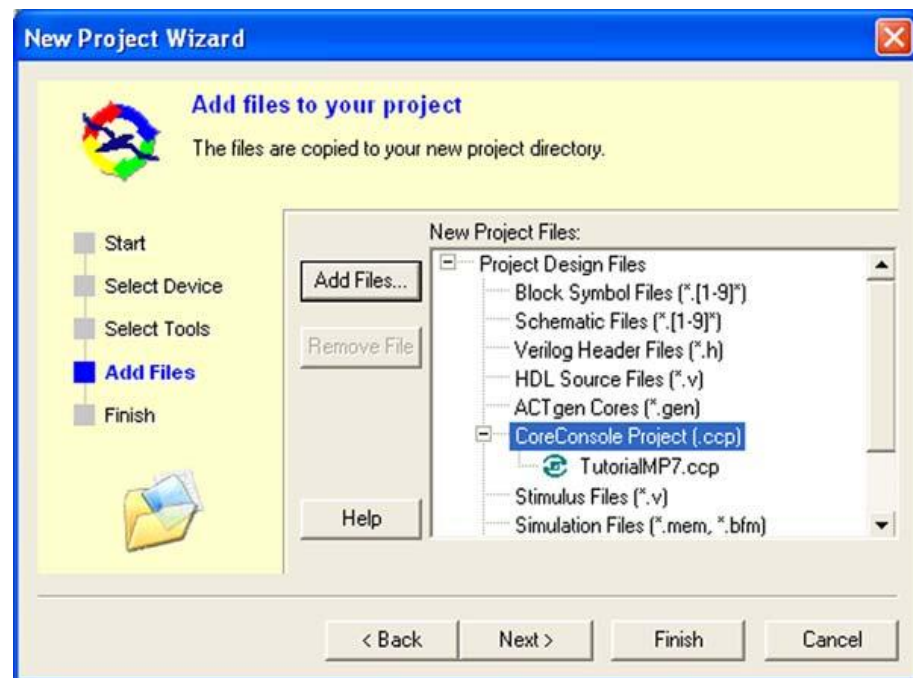


Figure 5-20. CoreConsole Project Added to the New Libero IDE Project

14. Review your project information. Click **Finish** to close the Wizard and create your new project (Figure 5-21). Click **Back** to return to any step of the Wizard and correct information in your project.

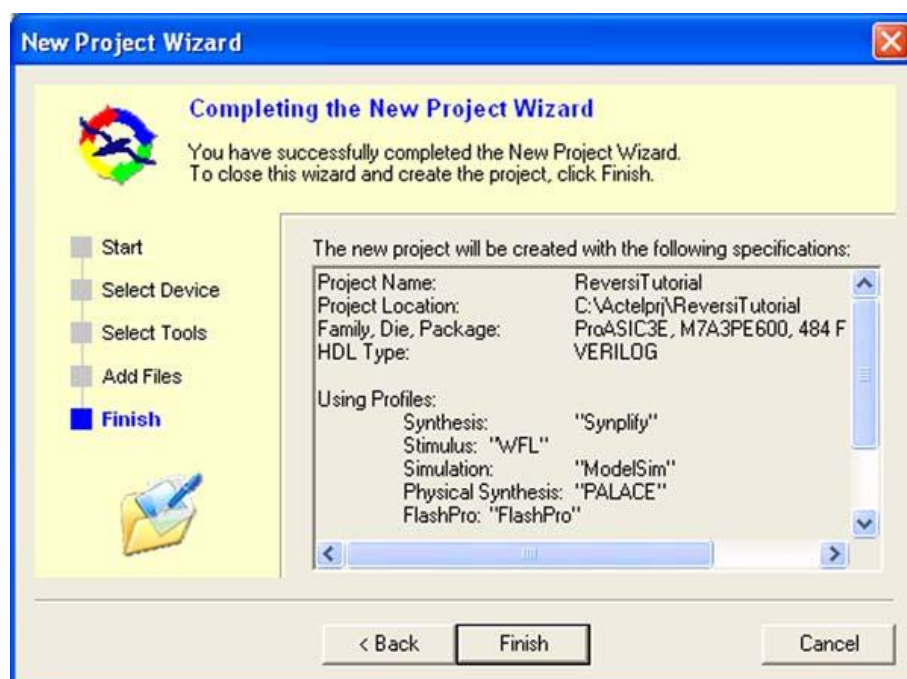


Figure 5-21. Summary in New Project Wizard

Your Libero IDE project exists, but you must add some top-level code or source to the project—such as a schematic, SmartGen core, or Verilog module—before you can run synthesis.

**To add a Verilog top-level HDL file to the project:**

1. From the **File** menu, select **Import Files**. Navigate to the `\Tutorial\FPGA` directory on the CD-ROM included with the CoreMP7 Development Kit and select the `TutorialTop.v` file. Click **Import**.
2. Click on the **Design Hierarchy** tab located at the bottom of the Libero IDE Design File Manager, as shown in Figure 5-22.

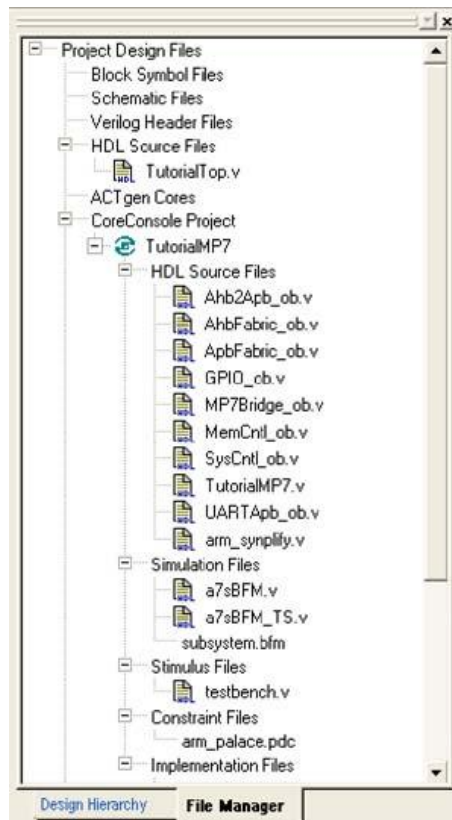


Figure 5-22. Libero IDE Design File Manager

3. From the **Design Hierarchy**, right-click the *TutorialTop.v* file and select **Set as Root** (see [Figure 5-23](#)). This sets the project's top-level file to the module contained in the source file just imported.

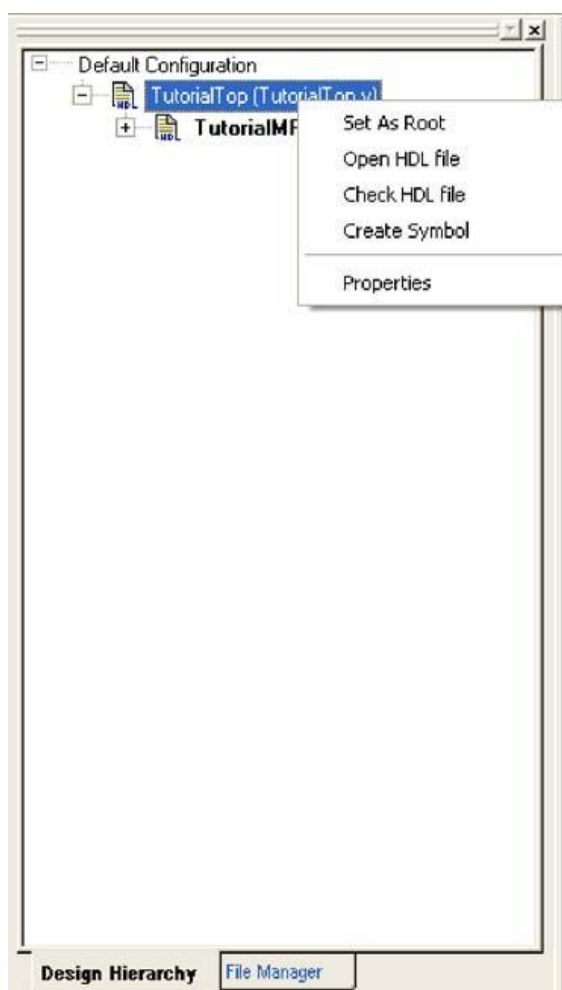


Figure 5-23. Libero IDE Design Hierarchy Viewer

4. Click the **File Manager** tab to return to the Libero IDE Design File Manager.

## Step 2 – Perform Pre-Synthesis Simulation

The next step is simulating the RTL description of the design. First, you must create a top-level testbench to provide a stimulus for the design. Keep in mind you will not be able to simulate CoreMP7 but will rely on the Bus Functional Model (BFM), which is a cycle-accurate model of the embedded ARM7TDMI-S processor.

### To create the top-level testbench:

1. From the **File** menu, select **File** and then **New**. This opens the New dialog box, shown in Figure 5-24.

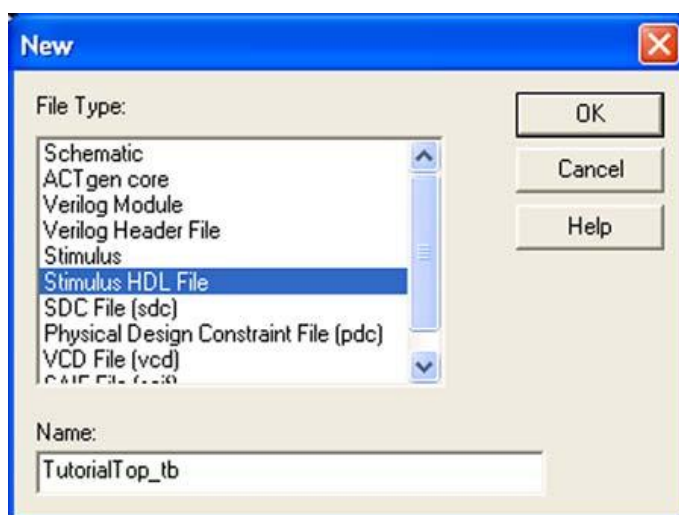


Figure 5-24. New File Dialog Box

2. Select **Stimulus HDL File** in the File Type field, enter “TutorialTop\_tb” in the **Name** field, and click **OK**. The HDL Editor opens. Enter the following text in a Verilog file, or if this document is open in an electronic form, copy and paste it from here. Alternatively, this file is provided in the \Tutorial\FPGA folder on the CD-ROM, and you can import the testbench as you did with the top-level source file.

```
`timescale 1ns/100ps

module testbench();
 parameter SYSCLK PERIOD = 100; // 10MHz

 reg SYSCLK;
 reg NSYSRESET;

 wire ICE nSRST;
```

```
pullup (weak1) p1 (ICE nSRST);

initial
begin
 SYSCLK = 1'b0;
 NSYSRESET = 1'b0;

 // Release system reset
 #(SYSCLK PERIOD * 4)
 NSYSRESET = 1'b1;

 #(SYSCLK PERIOD * 100000);

 $stop;
end

// SYSCLK signal
always @(SYSCLK)
 #(SYSCLK PERIOD / 2)
 SYSCLK <= !SYSCLK;

// Instantiate module to test
TutorialTop TutorialTop 0 (
 .SYSCLK(SYSCLK),
 .NSYSRESET(NSYSRESET),
 .RemapDefault(1'b0),
 .HIGH(),
 .LOW(),
 .FLASH BYTEN(),
 .FLASH CSN(),
 .FLASH OEN(),
 .FLASH RPN(),
 .FLASH WEN(),
 .SRAM ADSC(),
 .SRAM ADSP(),
 .SRAM ADV(),
 .SRAM BYTEN(),
```

```
.SRAM BYTE WEN() ,
.SRAM CLK() ,
.SRAM CSN() ,
.SRAM GLOBAL WEN() ,
.SRAM OEN() ,
.SRAM PWRDWN() ,
.MEM ADDR() ,
.MEM DATA() ,
.SW(8'b0) ,
.LED() ,
.RX0(1'b0) ,
.TX0() ,
.ICE nTRST() ,
.ICE TCK(1'b0) ,
.ICE TDI(1'b0) ,
.ICE TMS(1'b1) ,
.ICE VTref() ,
.ICE TDO() ,
.ICE RTCK() ,
.ICE nSRST(ICE nSRST) ,
.ICE DBGACK() ,
.ICE DBGRQ()
);

endmodule
```



3. From the **File** menu, click **Save**. The testbench file now appears in the Libero IDE Design Manager. Libero IDE lists *TutorialTop tb.v* under Stimulus Files, as shown in [Figure 5-25](#).

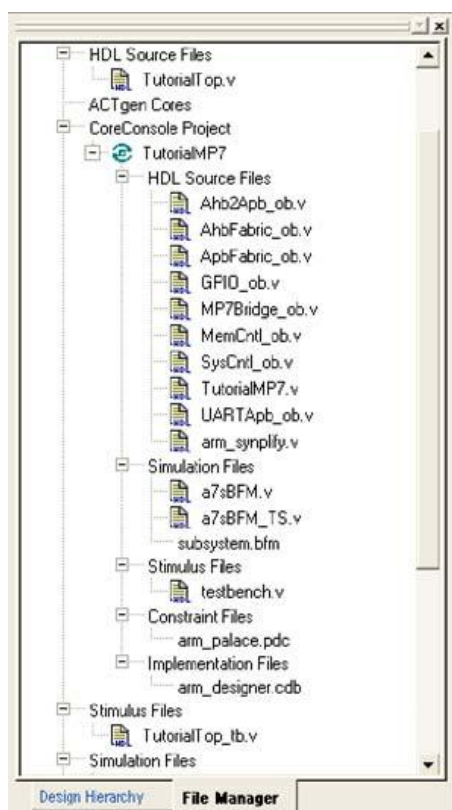


Figure 5-25. Libero IDE File Manager with Stimulus

4. Check the HDL in the file before you continue. Under the File Manager tab (Figure 5-26), right-click *TutorialTop\_tb.v* and select **Check HDL**. This checks the syntax of *TutorialTop\_tb.v*. Before moving to the next section, modify the code if you find any errors.

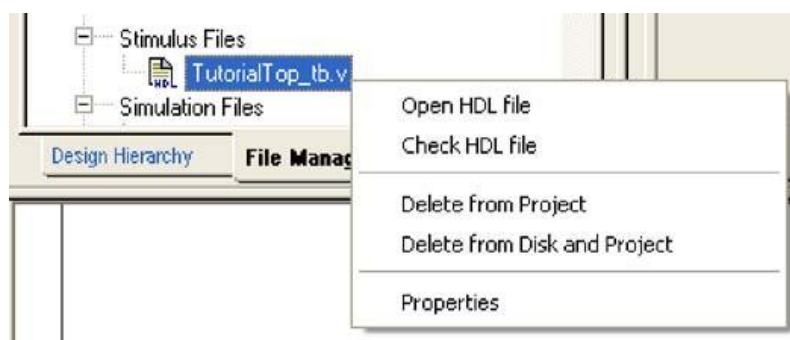


Figure 5-26. Check HDL Option from File Manager

**To perform a pre-synthesis simulation:**

1. From the **Design Hierarchy** tab, right-click the *TutorialTop.v* file and select **Organize Stimulus**, as shown in Figure 5-27.

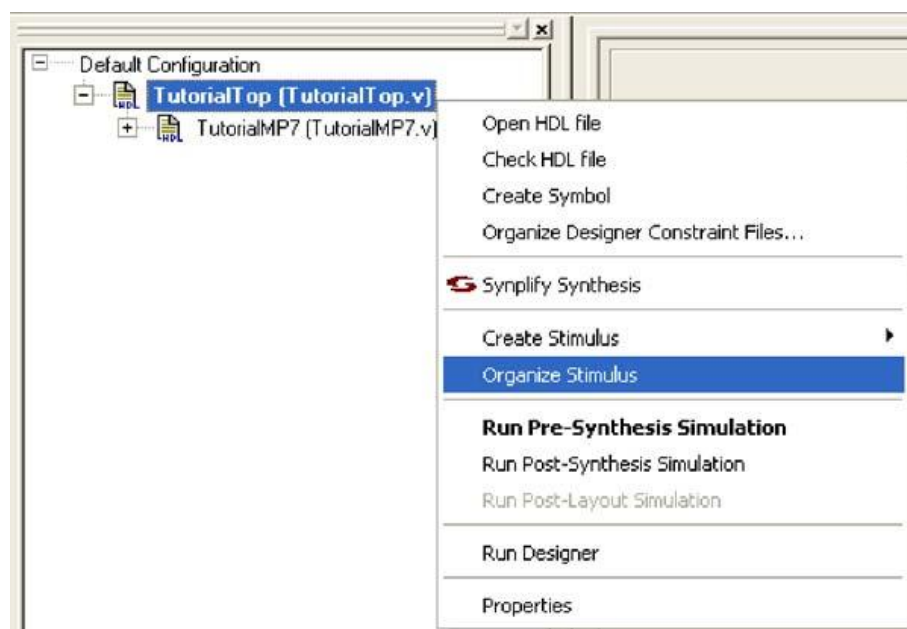


Figure 5-27. Design Hierarchy Context Menu

The Organize Stimulus dialog box appears, as shown in Figure 5-28.

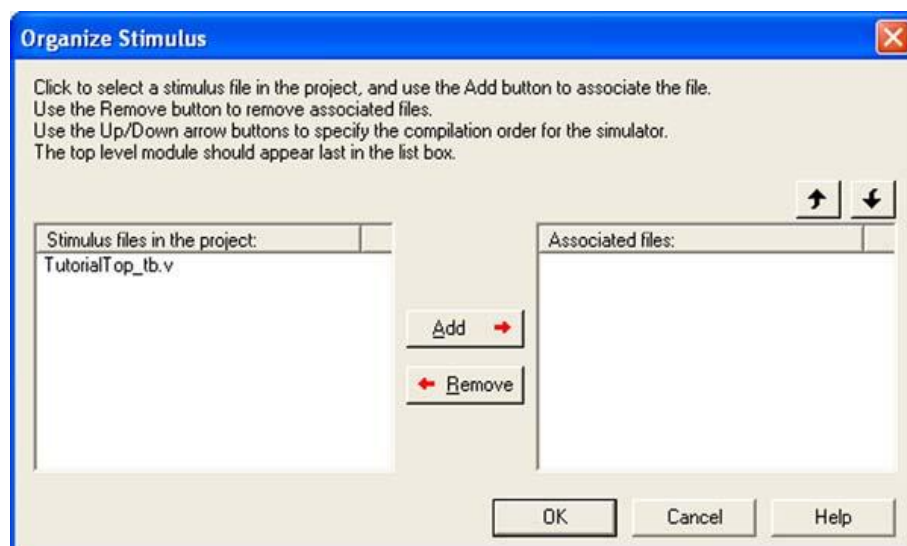


Figure 5-28. Organize Stimulus Dialog Box

2. Select *TutorialTop\_tb.v* from the Stimulus files in the project list box and click **Add** to add the file to the Associated files list.
3. Click **OK**. Stimulus icons in the Design Flow window turn green to notify you that there is a testbench file associated with the project.
4. Right-click the **Simulation** icon in the Libero IDE Design Flow window and select **Options**, as shown in Figure 5-29.

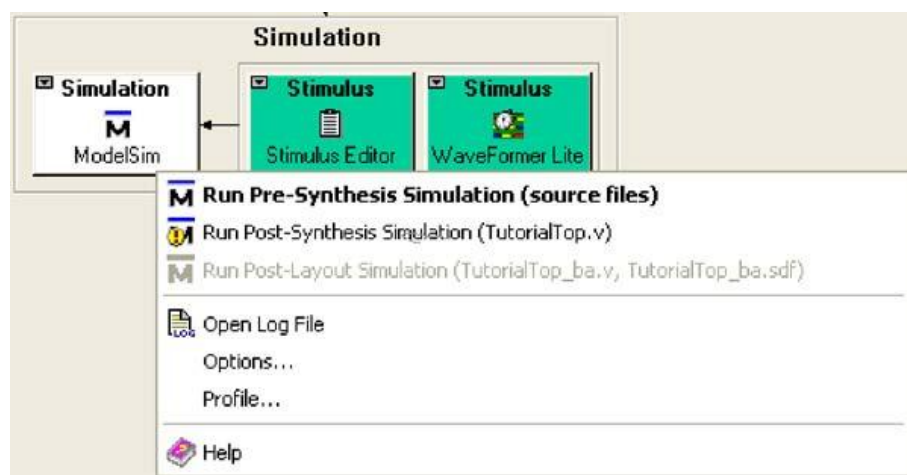


Figure 5-29. Simulation Context Menu

The Project Settings: Simulation options window appears (Figure 5-30).

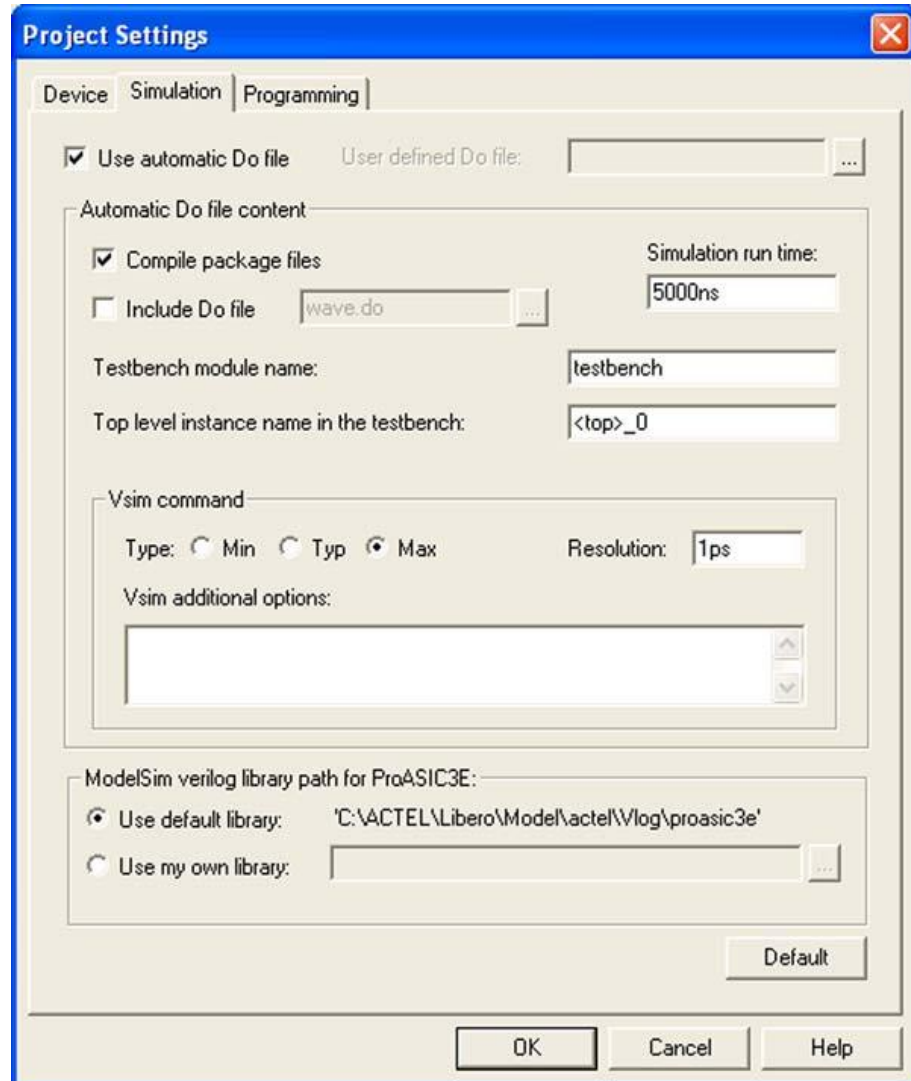


Figure 5-30. Simulation Project Settings

5. Change the **Simulation run time** from **1000ns** to **5000ns**, as shown in Figure 5-30, and click **OK**. Changing the run time allows the default BFM test scripts to complete without having to invoke additional run time from within the ModelSim simulator.

6. Click the **Simulation** icon in the Design Flow window, or right-click *TutorialTop.v* in the **Design Hierarchy** and select **Run Pre-Synthesis Simulation**, as shown in Figure 5-31.

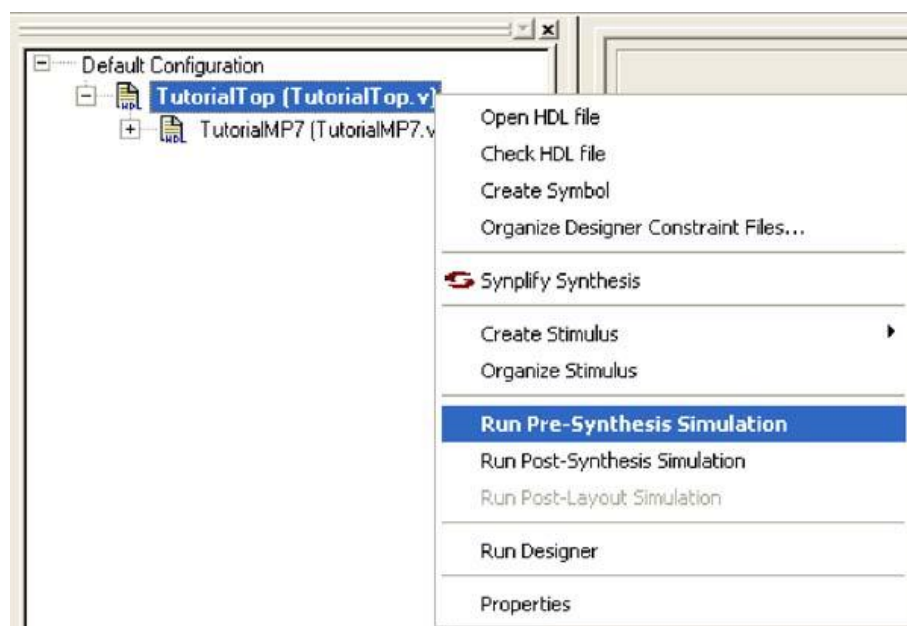


Figure 5-31. Running Pre-Synthesis Simulation from the *TutorialTop.v* Context Menu

## Quickstart Tutorial

The ModelSim simulator opens and compiles the source files, as shown in Figure 5-32.

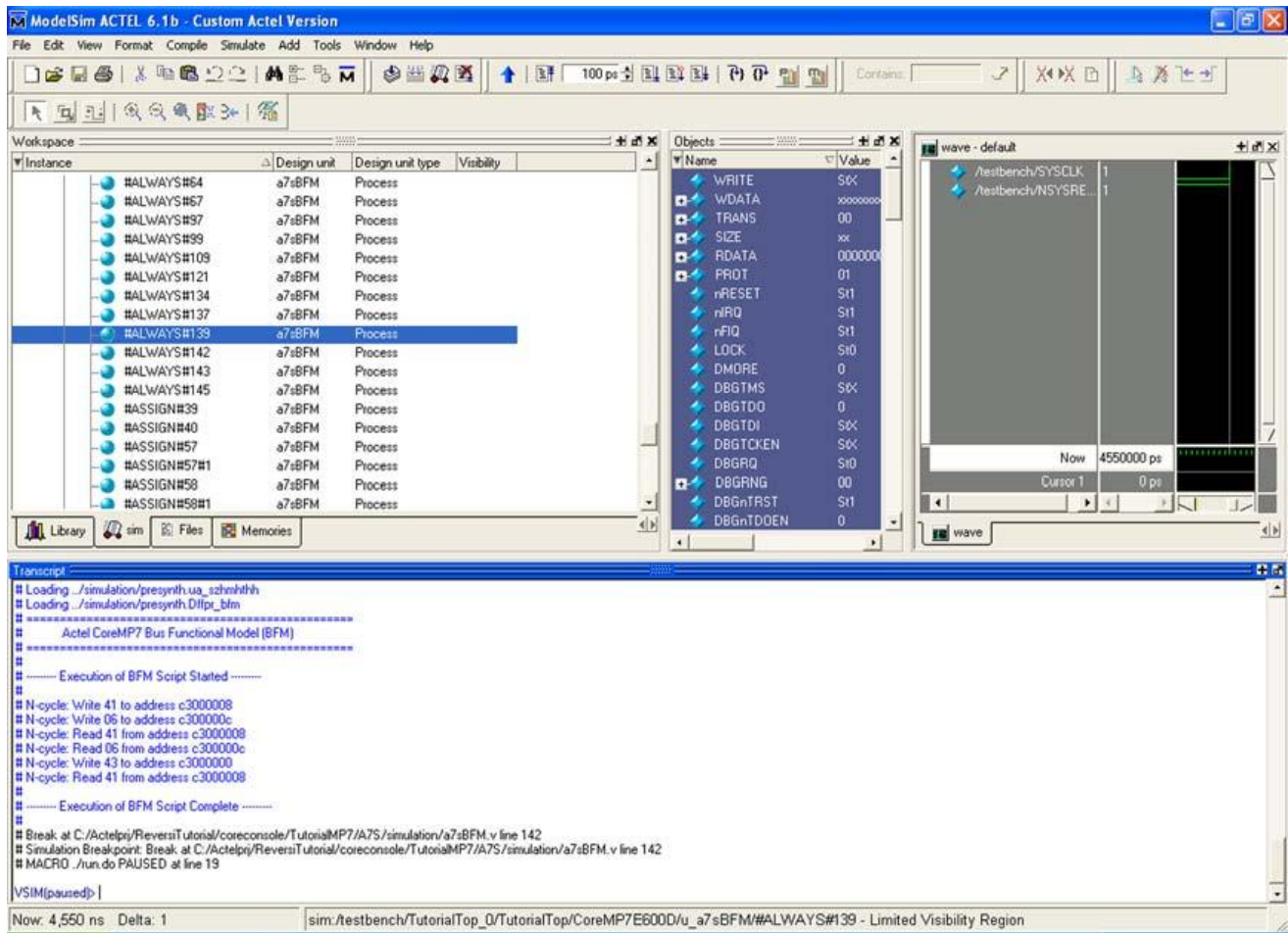


Figure 5-32. ModelSim Main Window

Once the compilation completes, the simulator simulates for the default time period of 5000 ns, and a wave window, shown in [Figure 5-33](#), opens to display the simulation results. The default wave window currently contains only the SYSCLK and NSYSRESET signals. You will expand this shortly. The results of the default BFM scripts can also be viewed in the ModelSim log window (as shown in [Figure 5-32 on page 66](#)). The successful reads and writes confirm that CoreMP7 is connected properly from the top level down to the various busses.

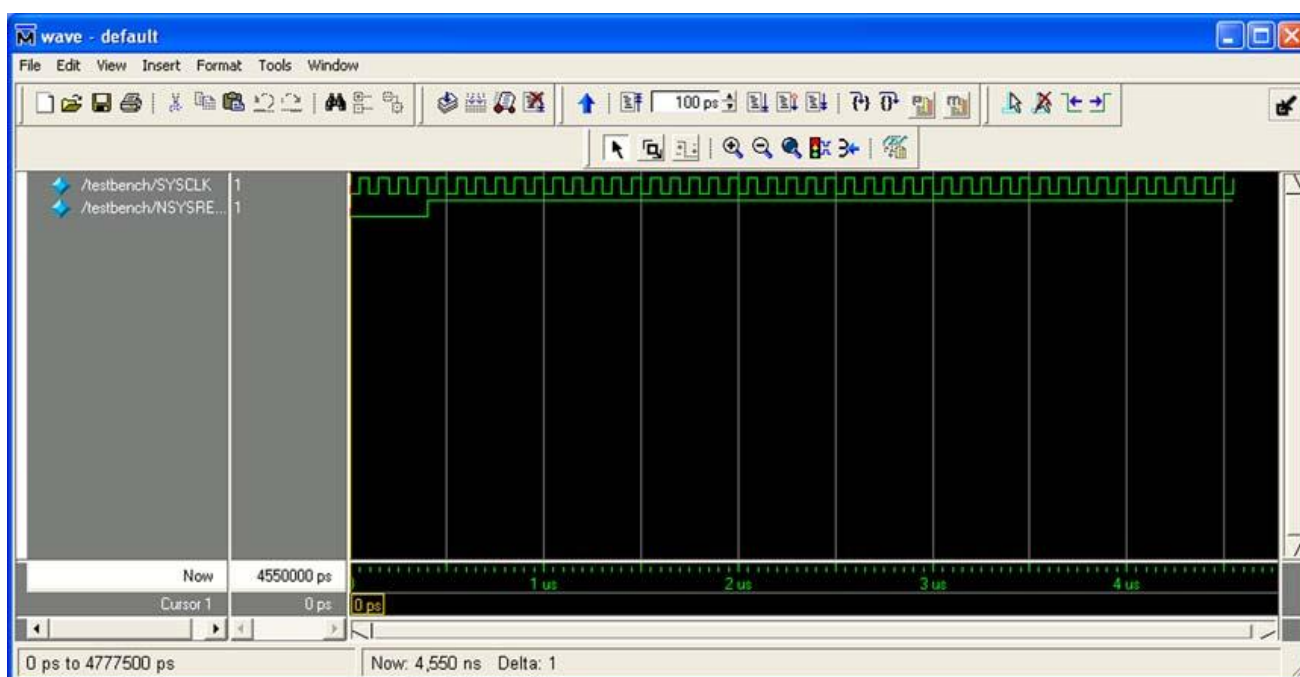


Figure 5-33. ModelSim Wave Window

7. To add the CoreMP7 signals to the ModelSim Wave window, navigate to the **CoreMP7E600D** instance in the ModelSim workspace, which can be found under the following hierarchy (shown in Figure 5-34):

TutorialTop\_0 > TutorialTop > CoreMP7E600D

Drag the CoreMP7E600D instantiation to the ModelSim wave window. The instantiation's signals will appear.

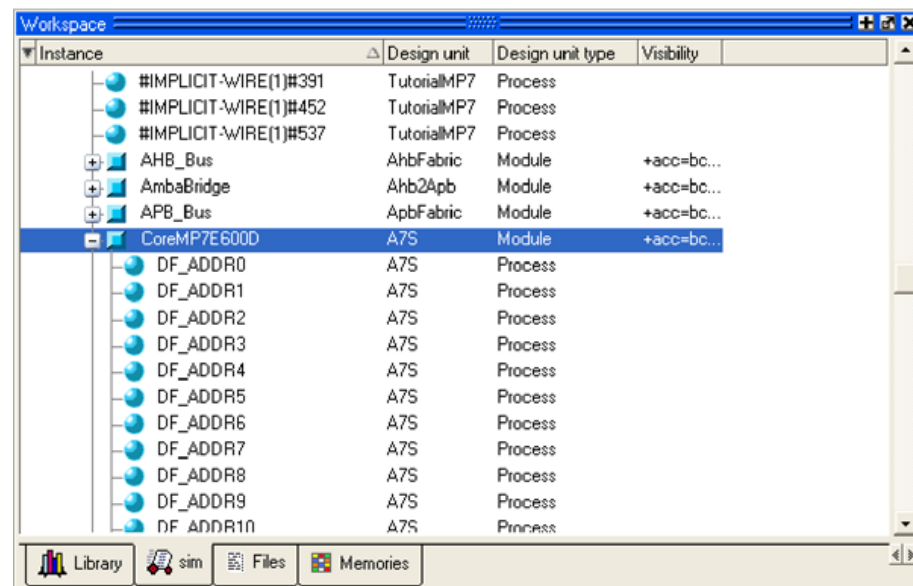


Figure 5-34. ModelSim Workspace Window



8. In the ModelSim Transcript (Log) window, type **restart**. This will bring up the Restart dialog box (shown in Figure 5-35). Click the **Restart** button. This will reset the simulation to the beginning so that logging of the CoreMP7 signals occurs.

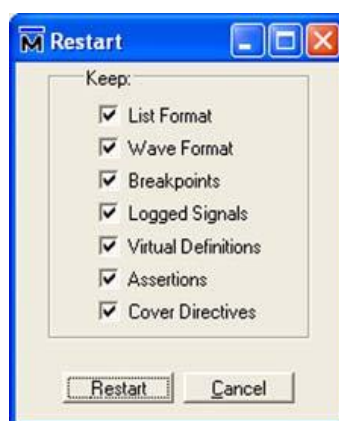


Figure 5-35. ModelSim Restart Dialog Box

9. Within the ModelSim Transcript window, type **run-all**. This will re-run the default testbench and BFM scripts. The results should be the same as the previous run. Notice the ModelSim Wave window—the CoreMP7 signals now have waveforms associated with them.
10. Undock the ModelSim Wave window and maximize it, then select **Zoom Full** from the **View > Zoom** menu. Examining the ADDR, RDATA, WDATA, WRITE, SIZE, SYSCLK, nRESET, and NSYSRESET signals allows the re-creation of the BFM scripts and validates the results. The corresponding signals have been grouped together and are shown in Figure 5-36.

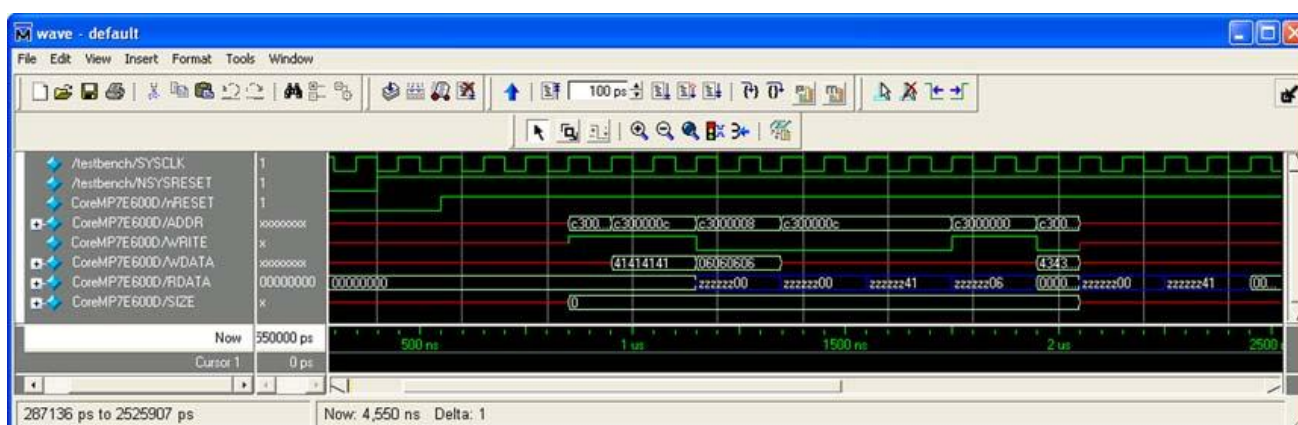


Figure 5-36. ModelSim Wave Window with CoreMP7 Signals

11. In the ModelSim window, select **File** then **Quit** to close the window.

## Step 3 – Synthesize the Design in Synplify

The next step is to generate an EDIF netlist by synthesizing the design in Synplify. For HDL designs, Libero IDE launches and loads the Synplify Synplicity synthesizer with the appropriate design files.

### To create an EDIF netlist for the design using Synplify:

1. In Libero IDE, click the **Synplify Synthesis** icon in the Design Flow window, or right-click the *TutorialTop.v* file in the Design Hierarchy and select **Synplify Synthesis**. This launches the Synplify synthesis tool with the appropriate design files, as shown in Figure 5-37.

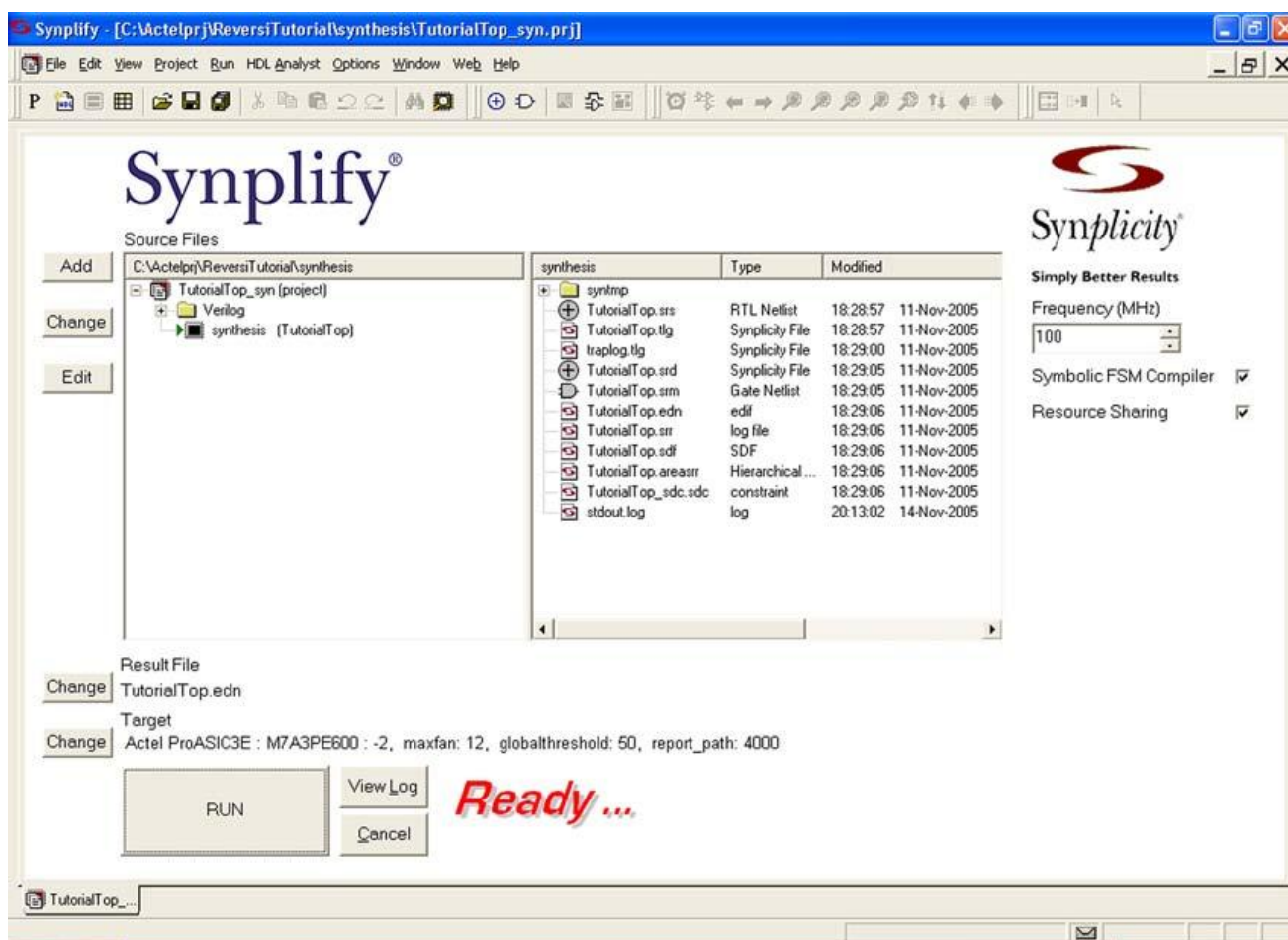


Figure 5-37. Synplify Synthesis Main Window

2. From the **Project** menu, select **Implementation Options**. This displays the options for the Implementation dialog box, as shown in Figure 5-38 for the M7A3PE600.

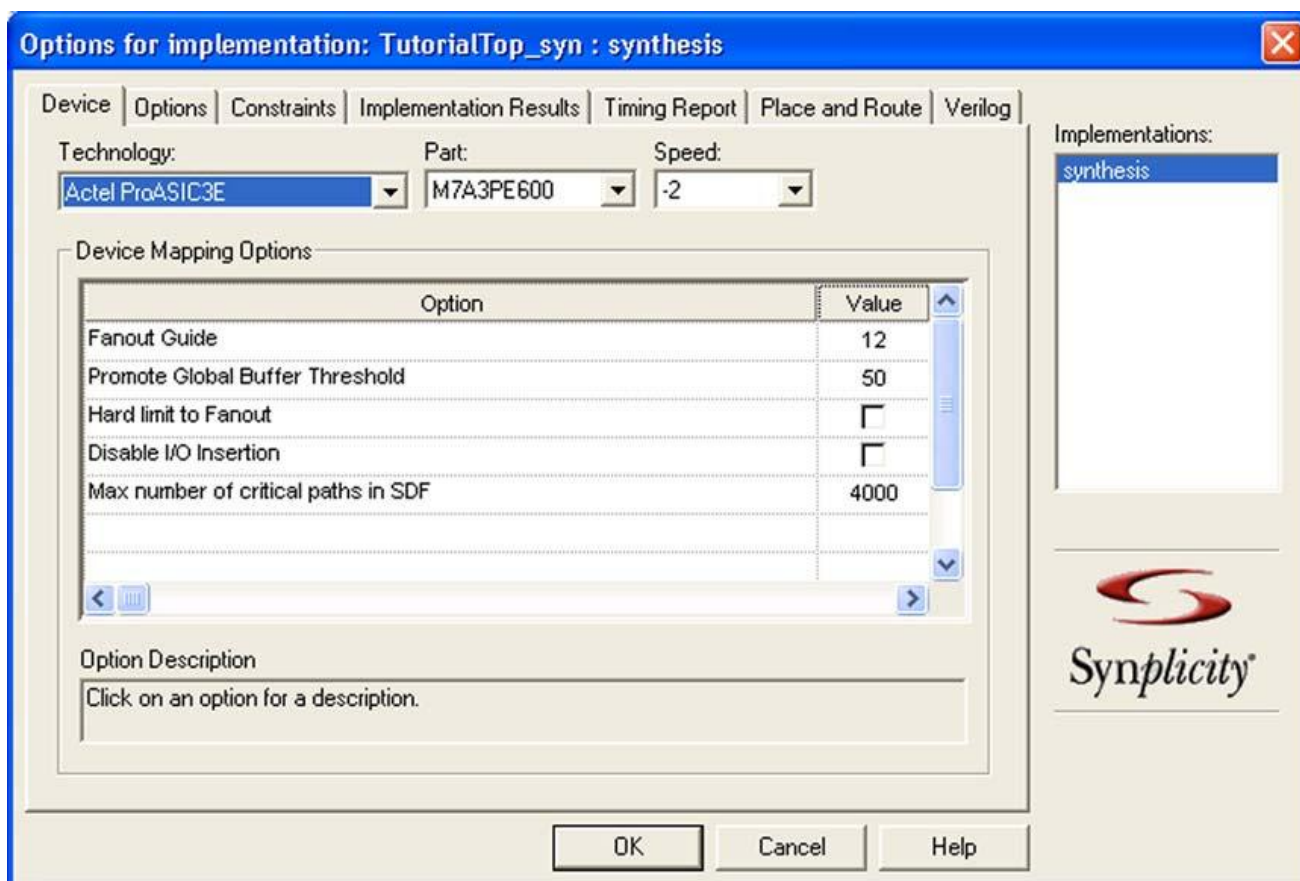


Figure 5-38. Implementation Options Dialog Box for the M7A3PE600

3. Set (confirm) the following in the dialog box:
  - **Technology:** Actel ProASIC3E (set automatically by Libero IDE)
  - **Part:** M7A3PE600
  - **Fanout Guide:** 12 (default)
  - **Hard limit to Fanout:** Off (default)
4. Accept the default values for each of the other tabs in the Options for Implementation dialog box and click **OK**.
5. In the Synplify main window, click **Run**. Synplify compiles and synthesizes the design into a netlist called *TutorialTop.edn*. This netlist is then automatically translated by Libero IDE into a Verilog netlist called *TutorialTop.v*.

The resulting EDIF and Verilog files are displayed under Implementation Files in the Libero IDE File Manager.

6. If any errors appear after you click the **Run** button, edit the file using the Synplify editor. To edit the file, double-click the file name in the Synplicity window. Any changes made here are saved to the original design file in Libero IDE.
7. Save and close Synplify. From the **File** menu, click **Exit** to close Synplify. Click **Yes** to save any settings made to the *TutorialTop syn.prj* file in Synplify.

## Step 4 – Perform Post-Synthesis Simulation

The next step is simulating the Verilog netlist of the design using the Verilog testbench created in “Step 2 – Perform Pre-Synthesis Simulation” on page 58.

1. Click the **Simulation** icon in the Libero IDE Design Flow window, or right-click the *TutorialTop.v* file in the Design Hierarchy tab and select **Run Post-Synthesis Simulation**. This launches the ModelSim simulator, which compiles the source files and testbench.

Once the compilation completes, the simulator runs for 5000 ns and the Wave window displays the simulation results. Verify that the read/write results of the executed BFM scripts are correct.

2. Follow the same sequence as in “Step 2 – Perform Pre-Synthesis Simulation” on page 58, beginning with step 7, to add and verify the internal CoreMP7 signals of the BFM.
3. Scroll in the Wave window to verify that the CoreMP7 system works correctly. Use the zoom buttons to zoom in and out as necessary.

## Step 5 – Implementing the Design with Actel Designer

After creating and simulating the design, the next phase is implementing the design using the Actel Designer software (performing place-and-route).

1. From the Libero IDE **File** menu, select **Import Files**. Navigate to the \Tutorial\FPGA directory on the Development Kit CD-ROM and select the *TutorialTop PinConstraints.pdc* file. It might be necessary to change the file type to PDC to view this file. Click **Import**. The file will now be listed under Constraint Files in the Libero IDE File Manager.

2. Click the Designer **Place & Route** button in the Libero IDE Design Flow window, or right-click *TutorialTop.v* in the **Design Hierarchy** tab and select **Run Designer**. Designer reads in the design file (Figure 5-39).

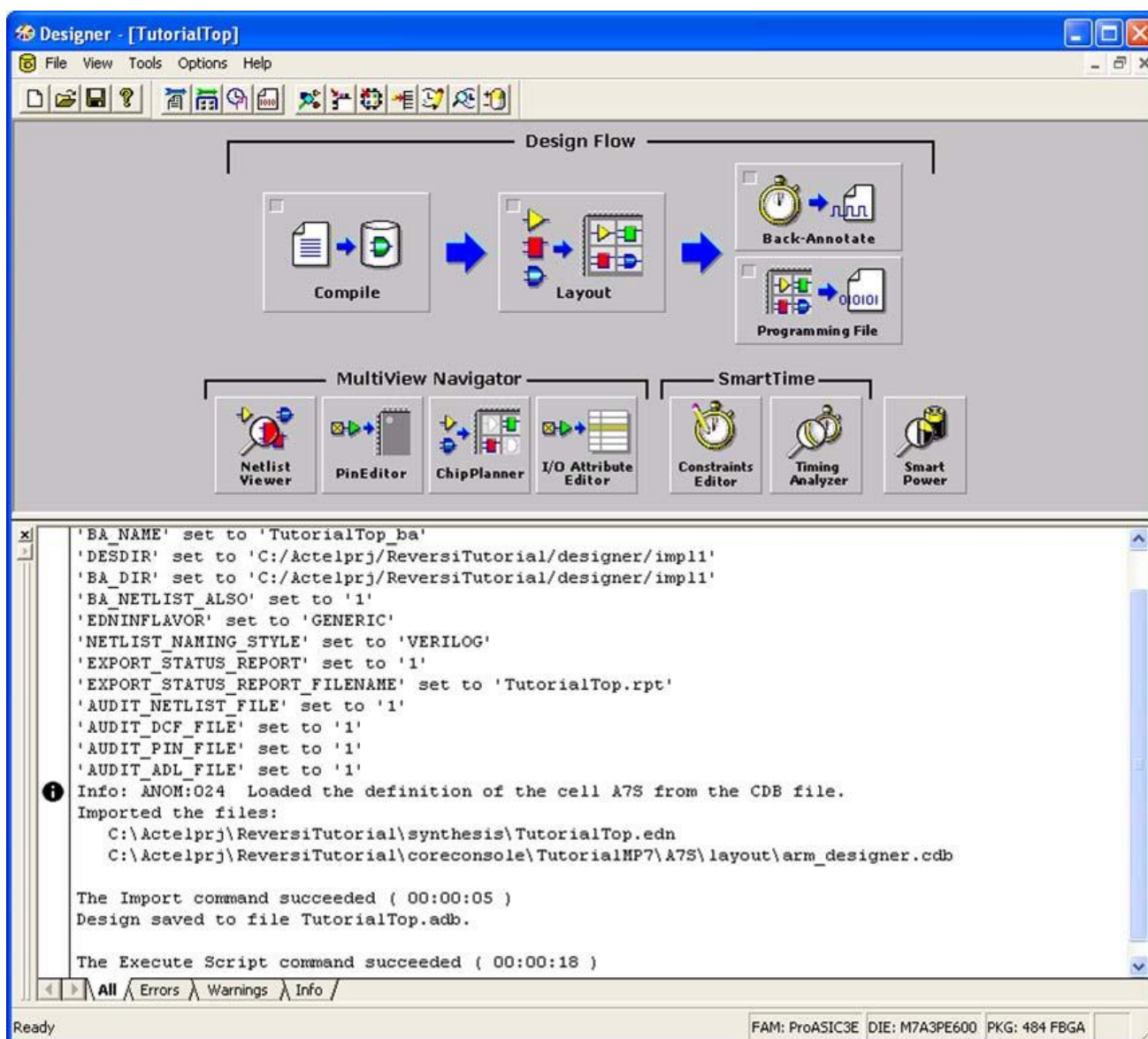


Figure 5-39. Actel Designer GUI

The Device Selection Wizard opens (Figure 5-40).

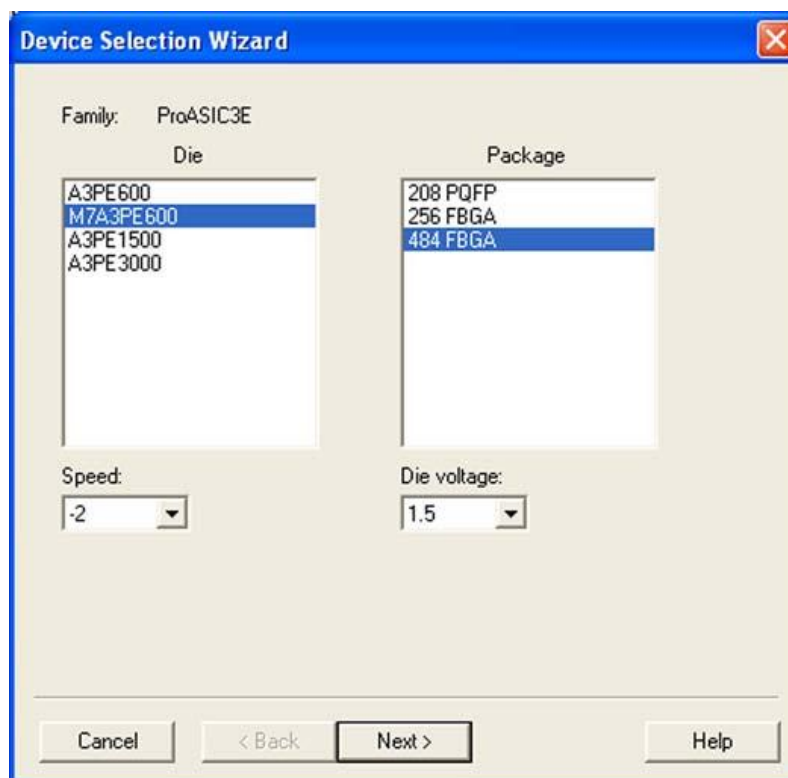


Figure 5-40. Device Selection Wizard for M7A3PE600

3. Select **M7A3PE600** in the **Die** field and **484 FBGA** in the **Package** field. Accept the default **Speed** grade and **Die voltage** and click **Next**.
4. Use the default I/O settings and click **Next**.
5. Use the default **Junction Temperature** and **Voltage** setup and click **Finish**.
6. From the Designer **File** menu, select **Import Source Files**.



This displays the Import Source Files dialog box (Figure 5-41). Click the **Add** button, navigate to the Libero IDE project's `\constraint` directory, and add the *TutorialTop\_PinConstraints.pdc* file (it may be necessary to change the file type to view the PDC file). Once the file has been added, click **OK**.

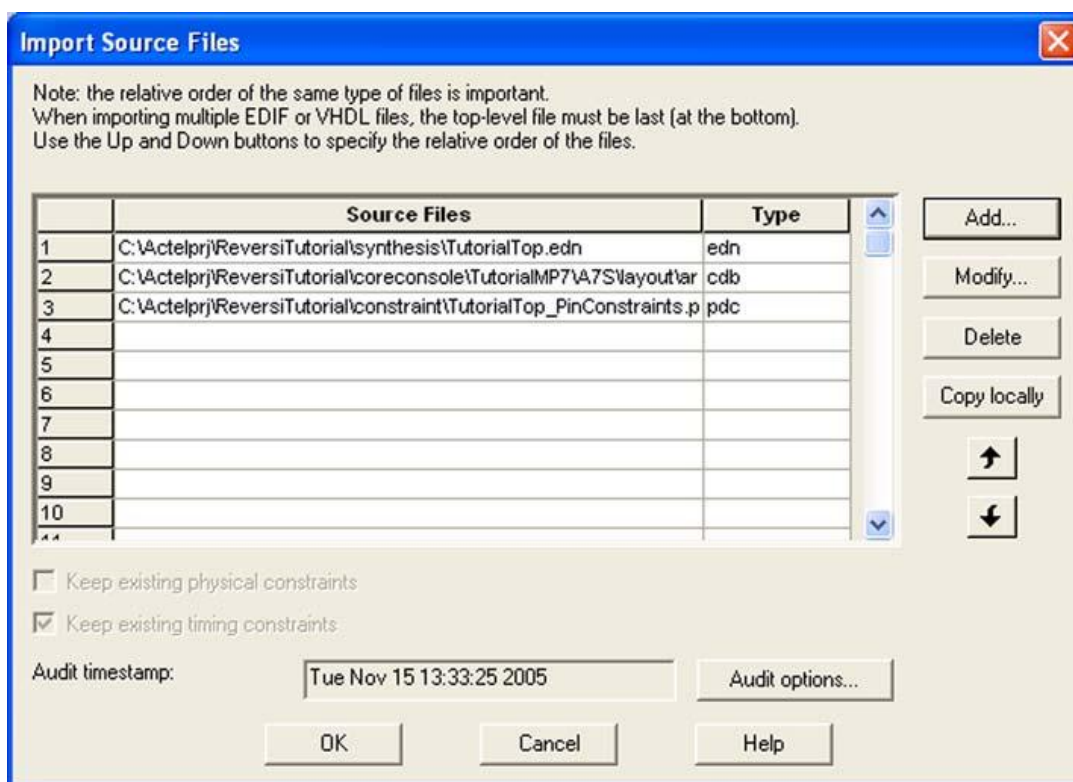


Figure 5-41. Import Source Files Dialog Box in Designer

- When the EDIF Import Options dialog box appears, as in [Figure 5-42](#), click **OK**. This will re-import the source files (all three of them) into Designer.



Figure 5-42. EDIF Import Options Dialog in Designer

- Click the **Compile** icon. Leave the default Compile settings ([Figure 5-43](#)) and click **OK**.

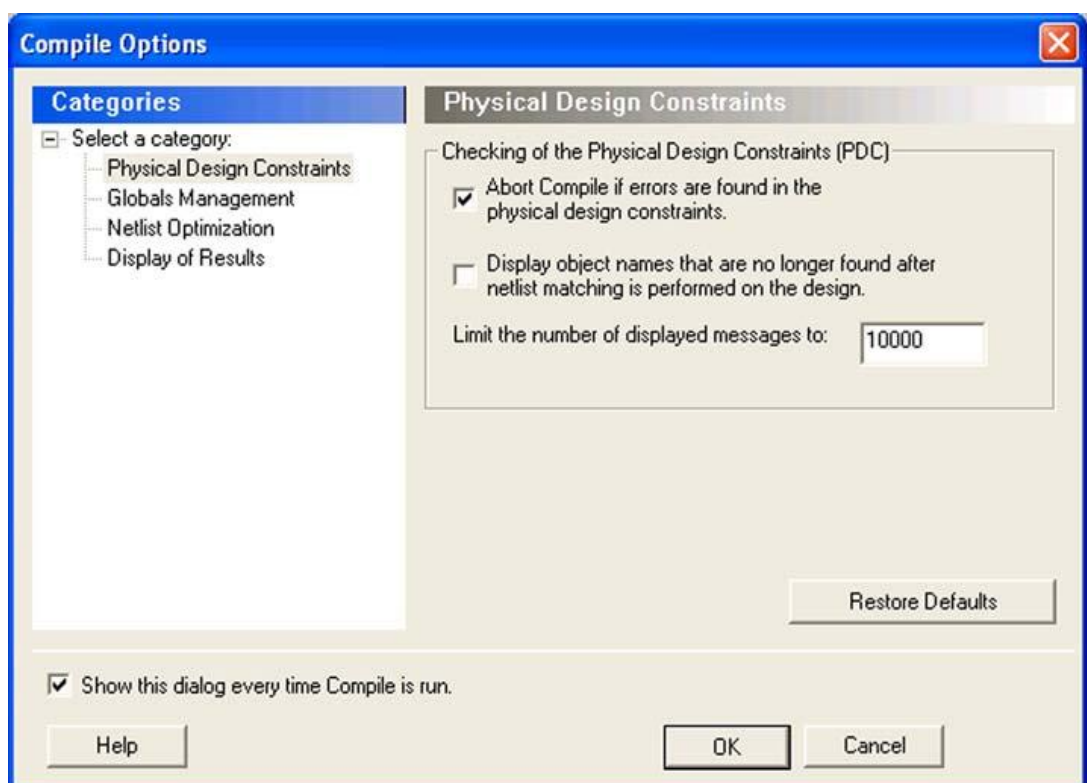


Figure 5-43. Compile Options Window



Designer compiles the design and shows the utilization of the selected device. Also, note that the Compile icon in Designer turns green once the compile has successfully completed.

9. Once the design compiles successfully, use the I/O Attribute Editor tool to verify the pin assignments imported from the pin constraints file. Alternatively, the I/O Attribute Editor can be used to create pin assignments. Click the **I/O Attribute Editor** to open the tool. It opens in the MultiView Navigator user interface, as shown in Figure 5-44.

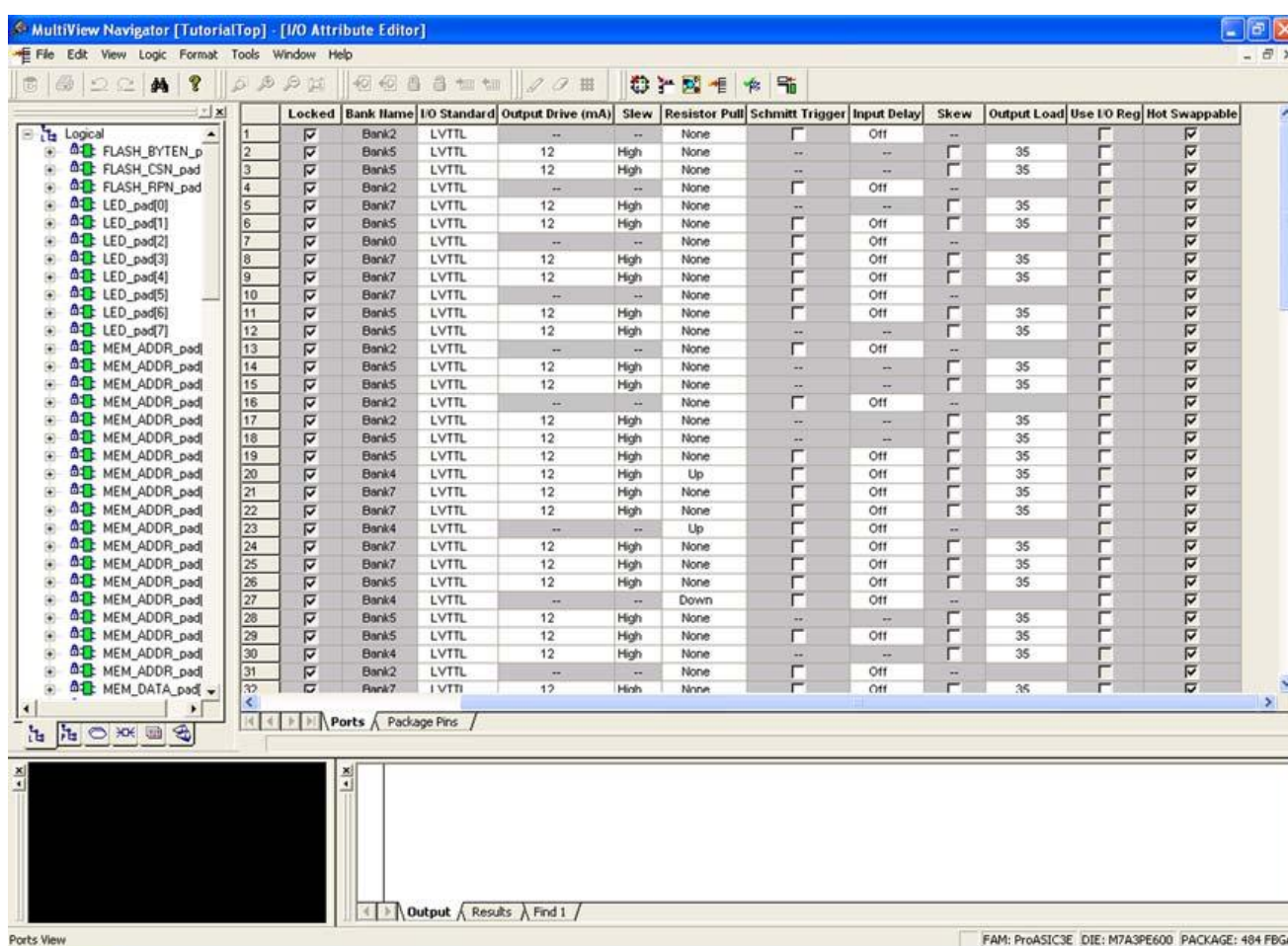


Figure 5-44. I/O Attribute Editor in MultiView Navigator

10. Verify that the correct pins have been assigned to all of the signals. If changes are made, select **Commit** from the **File** menu and then close the I/O Attribute Editor.

*Optional:* After successfully compiling the design, use the Designer Tools to view the pre-layout static timing analysis with SmartTime, set the timing constraints in SmartTime, analyze the

static and dynamic power with SmartPower, and use the ChipPlanner to assign modules. Click the appropriate icons to access these tools.

For more information on these functions, refer to the Designer or Libero IDE online help.

11. In Designer, click **Layout**. This opens the Layout Options dialog box, shown in [Figure 5-45](#).

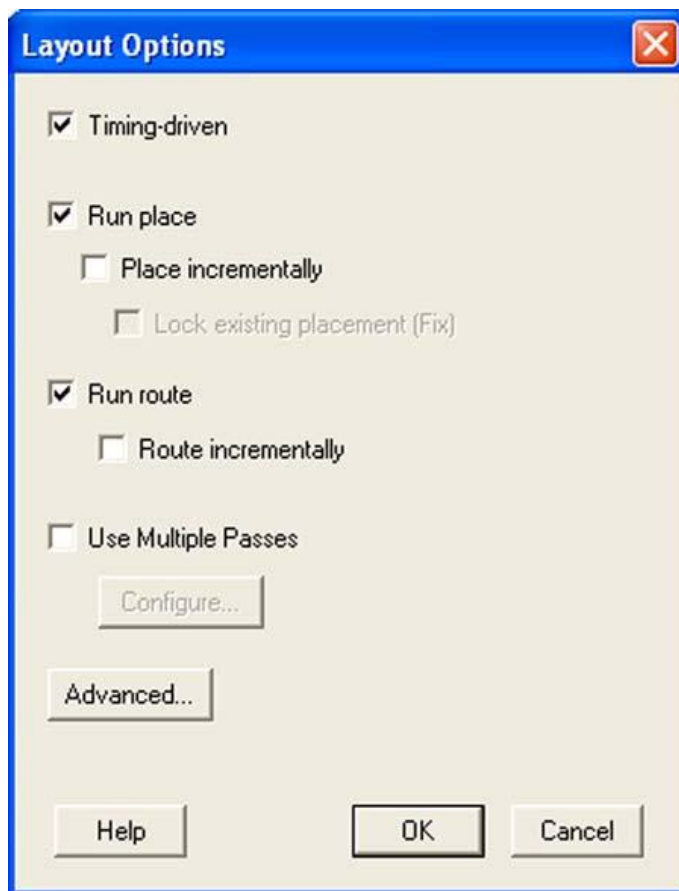


Figure 5-45. Layout Options Dialog Box

12. Click **OK** to accept the default layout options. This runs place-and-route on the design. The Layout icon turns green to indicate that the layout has successfully completed.

13. From Designer, click **Back-Annotate** in the Design Flow window. This opens the Back-Annotate dialog box, shown in Figure 5-46.

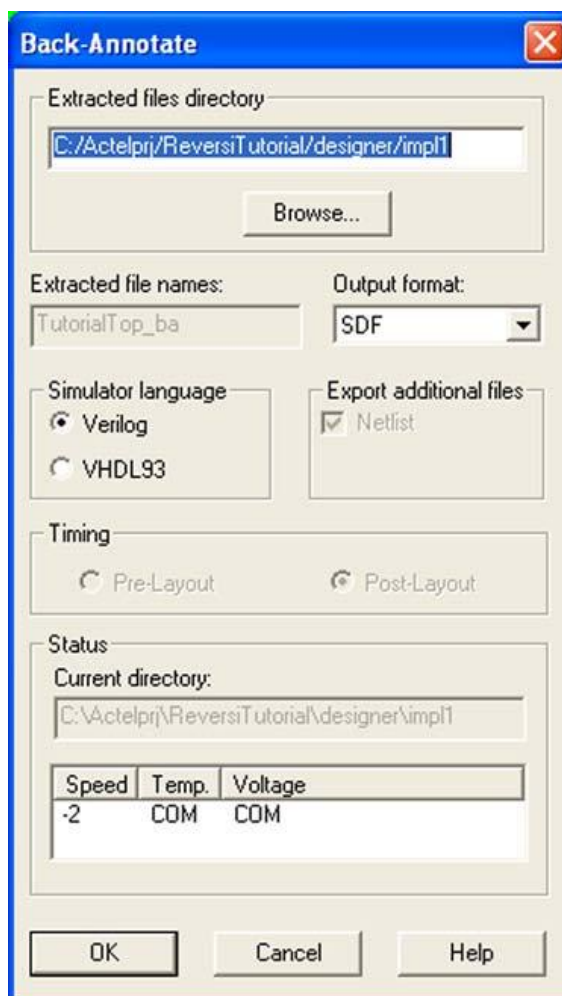


Figure 5-46. Back-Annotate Dialog Box

14. Accept the default settings and click **OK**. The Back-Annotate icon turns green.
15. Save and close Designer. From the **File** menu, click **Exit**. Click **Yes** to save the design before closing Designer. Designer saves all the design information in an ADB file.

The file *TutorialTop.adb* appears under the Designer Files tab of the File Manager. To re-open the file, right-click it and select **Open in Designer**.

## Step 6 – Perform Timing Simulation with Back-Annotated Timing

After completing place-and-route and back-annotation of the design, perform a timing simulation with the ModelSim HDL simulator.

### **To perform a timing simulation:**

1. Click the **Simulation** icon in the Libero IDE Design Flow window, or right-click the *TutorialTop.v* file under the **Design Hierarchy** tab and select **Run Post-Layout Simulation**.
2. This launches the ModelSim Simulator, which compiles the back-annotated Verilog netlist file and testbench.  
Once the compilation completes, the simulator runs for 5000 ns and the Wave window displays the simulation results. Verify that the read/write results of the executed BFM scripts are correct.
3. Follow the same sequence as in “[Step 2 – Perform Pre-Synthesis Simulation](#)” on page 58, beginning with step 7, to add and verify the internal CoreMP7 signals of the BFM.
4. Scroll in the Wave window to verify that the CoreMP7 system works correctly. Use the zoom buttons to zoom in and out as necessary.

## Step 7 – Generating the Programming File

1. Open the *TutorialTop.adb* file in Designer and click the **Programming File** button in the Design Flow window, which opens the FlashPoint window (Figure 5-47).

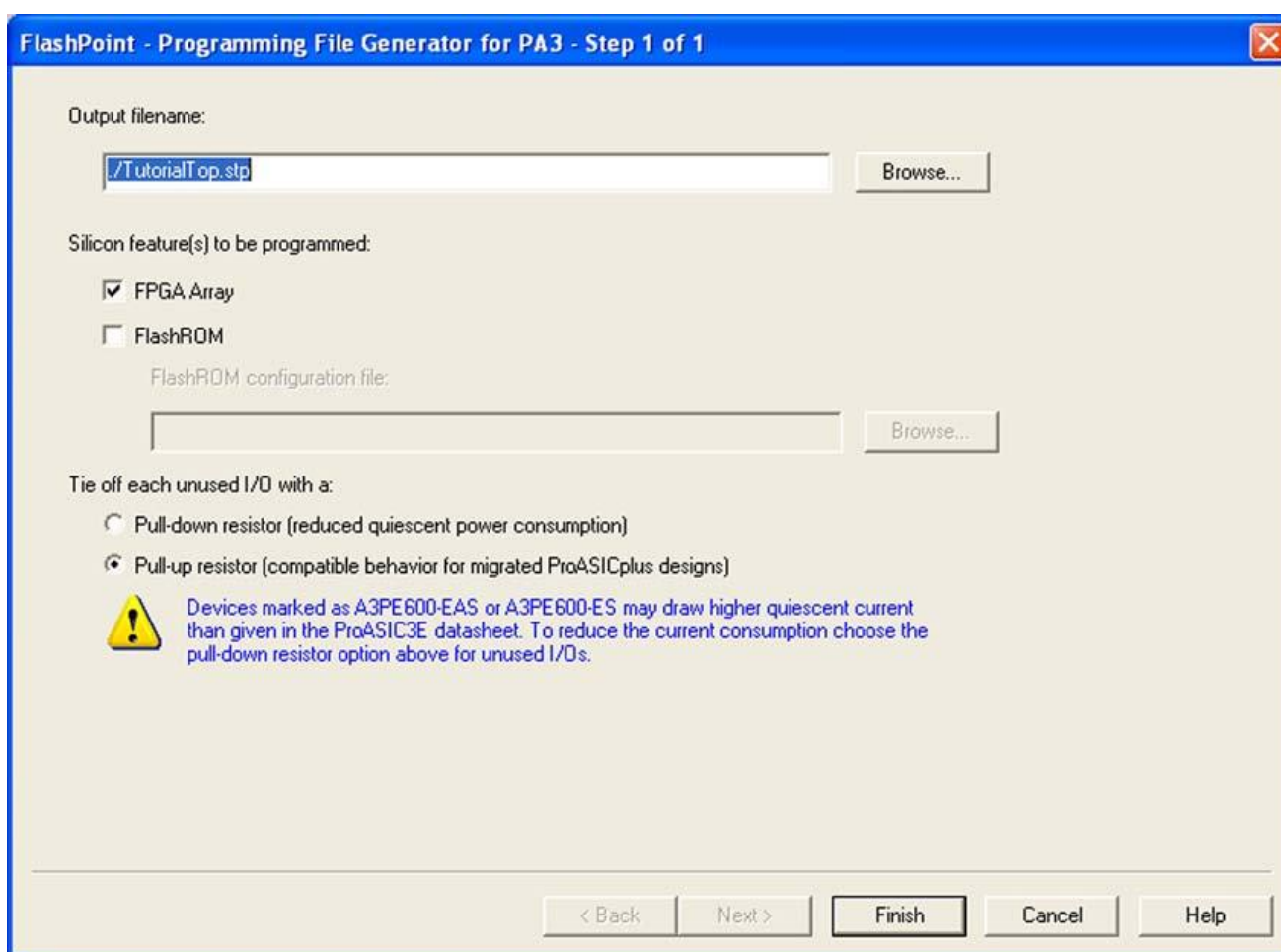


Figure 5-47. Flash Point Dialog Box

2. Click **Finish**. The programming file is generated and saved in the *\designer\impl1* folder. The **Programming File** icon in the Designer Design Flow window should now be green, indicating that programming file generation has been successfully completed.
3. Save and close Designer. From the **File** menu, click **Exit**. Click **Yes** to save the design before closing Designer.

## Step 8 – Programming the Device

After generating the programming file, program the device using an Actel FlashPro3 programmer.

Before performing any action with the FlashPro3 programmer, it must be properly set up. Connect the FlashPro3 USB cable to your PC USB port, connect the ribbon cable to the programming header on the target board, and turn on the power switch on the board.

1. Click the **FlashPro Programming** button in the Libero IDE Design Flow window, or right-click *TutorialTop.v* under the Design Hierarchy tab and select **Run FlashPro**.

FlashPro opens (Figure 5-48).

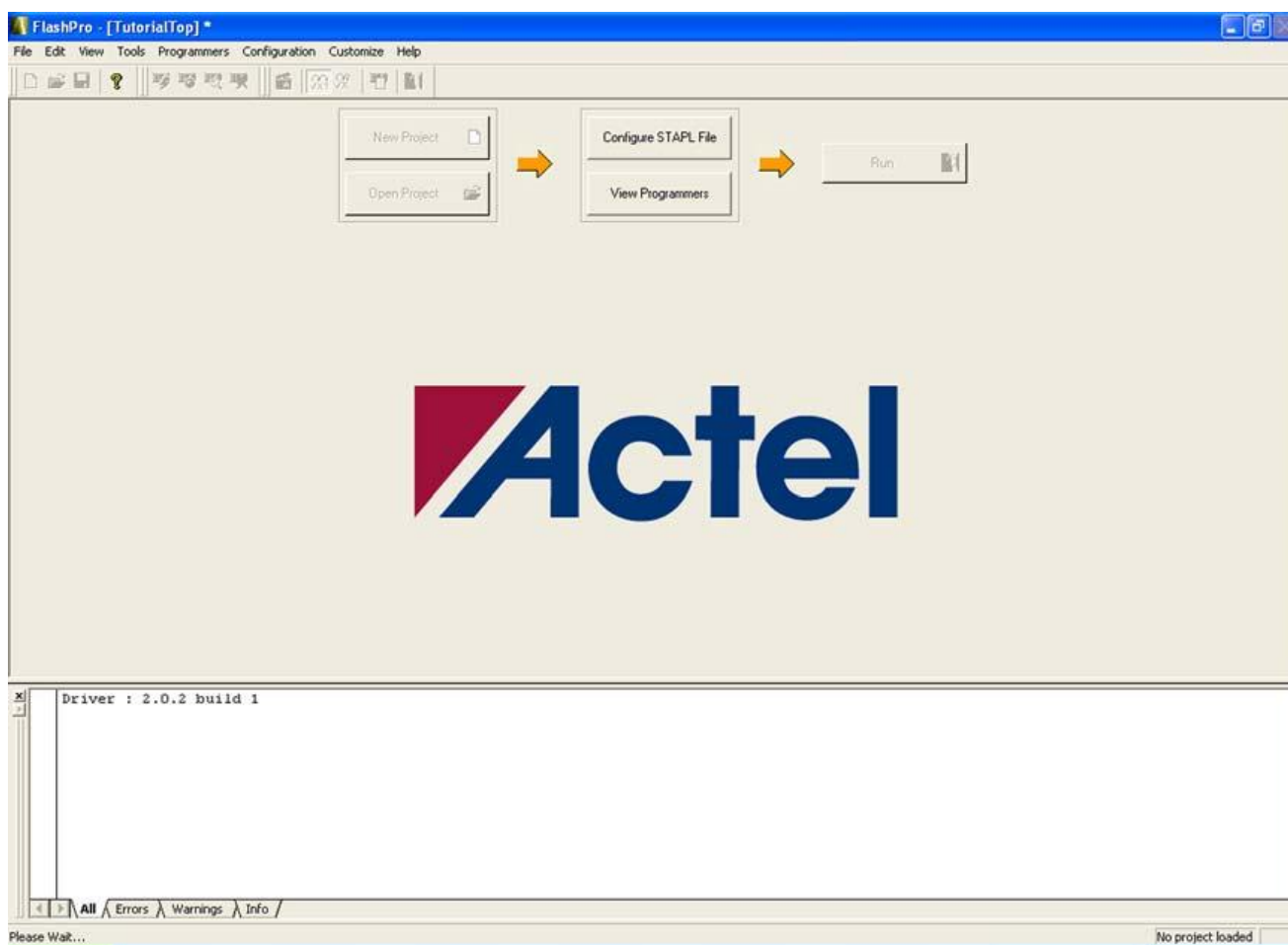


Figure 5-48. FlashPro Desktop – Prior to Locating Programmer(s)

FlashPro establishes a communication channel with the FlashPro3 programmer(s) attached to the PC (Figure 5-49).

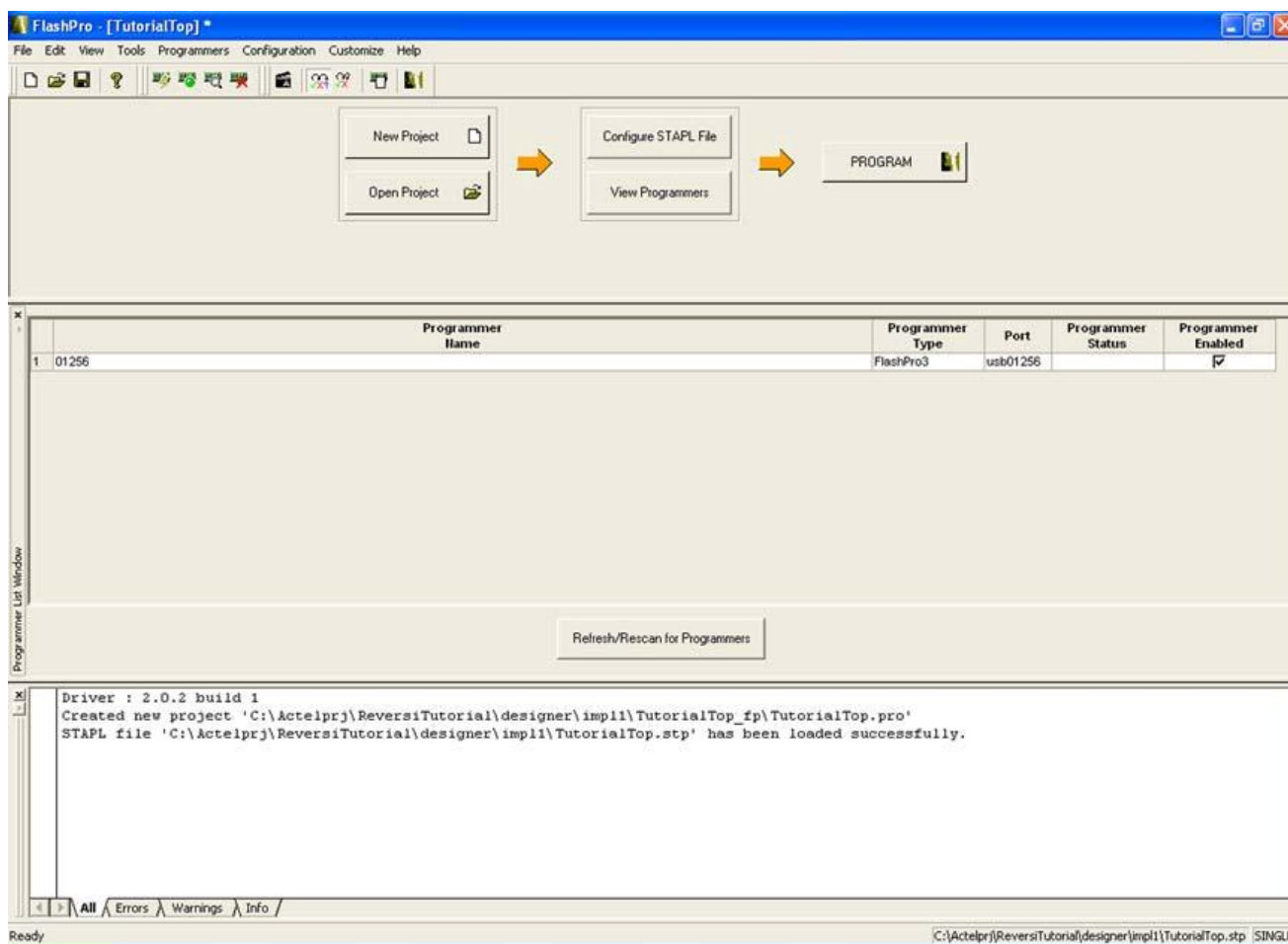


Figure 5-49. FlashPro Desktop – After Locating Programmer(s)

2. When launching FlashPro from within Libero IDE, the project STAPL file is automatically loaded and configured. To verify the STAPL file being used to program the device, click the **Configure STAPL File** button.

The STAPL Configuration window (Figure 5-50) appears.

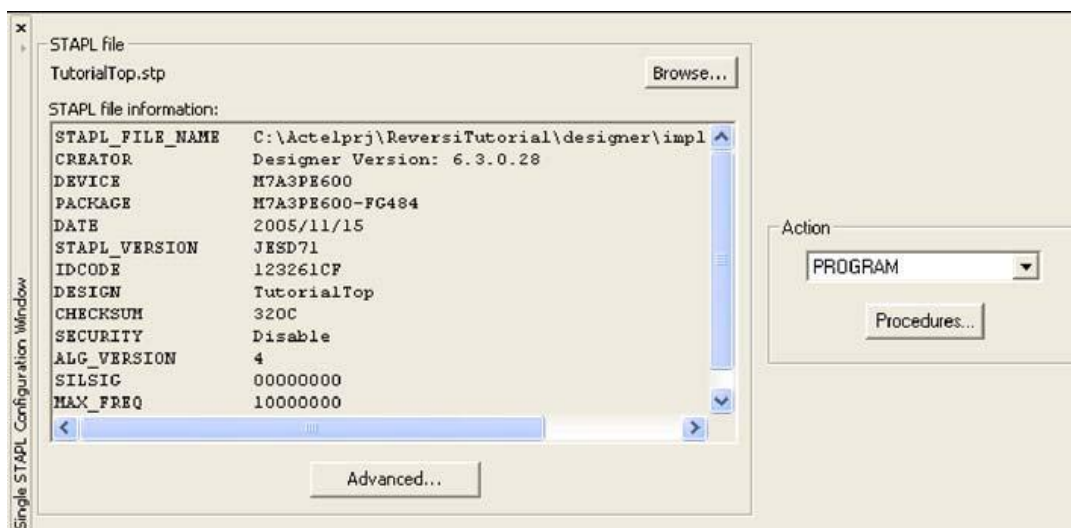


Figure 5-50. STAPL Configuration Window

- Here the programming file may be changed by clicking the **Browse** button and navigating to the new programming file. Various **Actions** may be performed using the drop-down selections. For this tutorial, leave it set to **PROGRAM** (default).

For more information on these functions, refer to the FlashPro online help.

To return to the Programmer List window (Figure 5-51), click the **View Programmers** button.



Figure 5-51. Programmers List Window



4. Verify that the attached programmer's check box is selected under the **Programmer Enabled** heading, then click the **Program** button.
5. Programming will take approximately two and a half to three minutes to complete. Under the **Programming Status** heading, a progress bar will appear. Alternatively, the log window may also be viewed (Figure 5-52).

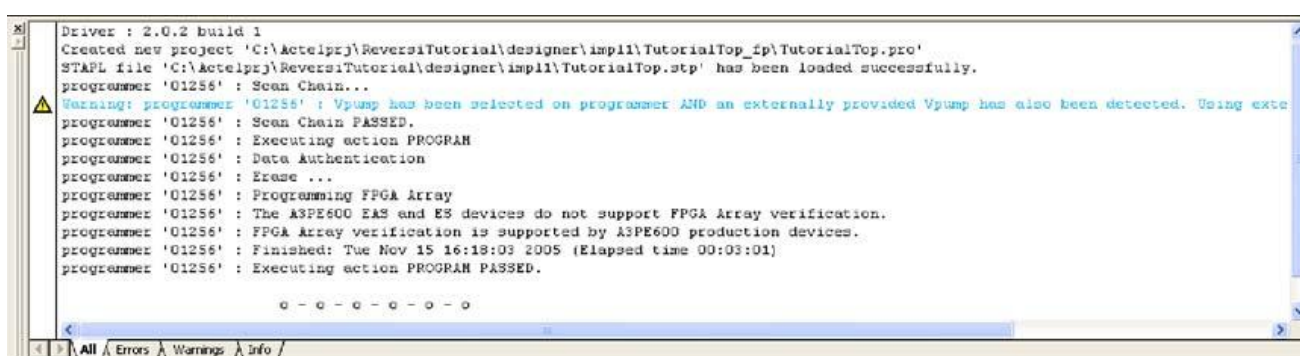


Figure 5-52. FlashPro Log Window

6. Once programming has completed, select **Exit** from the **File** menu. Answer **Yes** when prompted to save the project. This will return you to Libero IDE.
7. From the Libero IDE **File** menu, select **Exit**. If prompted to save your project, answer **Yes**, as this completes the Libero IDE FPGA portion of this tutorial.

## ARM RealView Developer Kit – Actel Edition

The RealView Developer Kit, available from Actel (separately from the CoreMP7 Development Kit), contains an integrated project manager and file editor suitable for creating and developing embedded projects. In this section, we'll create the executable source code to be run on CoreMP7. No coding will be required in this section, as we'll use source code included with the CoreMP7 Development Kit.

### Step 1 – Creating a RealView Project

#### **To create a RealView project:**

1. Copy the contents from the `\Tutorial\MPU` directory on the CoreMP7 Development Kit CD-ROM to the `C:\CoreMP7\Tutorial\Source` directory, which must be created.
2. Launch the **RealView Debugger 1.8** program located under **Start > ARM > RealView Developer Suite 2.2**.

Prior to launching RealView Debugger, the RealView ICE Micro Edition must be connected to the PC via the USB port. For information on installing and configuring the drivers for the RealView ICE Micro Edition, see the *RealView ICE User Guide* on the RealView Developer Kit – Actel Edition CD-ROM.

3. From the **Project** menu, select **New Project**. This displays the Create New Project dialog box, as shown in Figure 5-53.

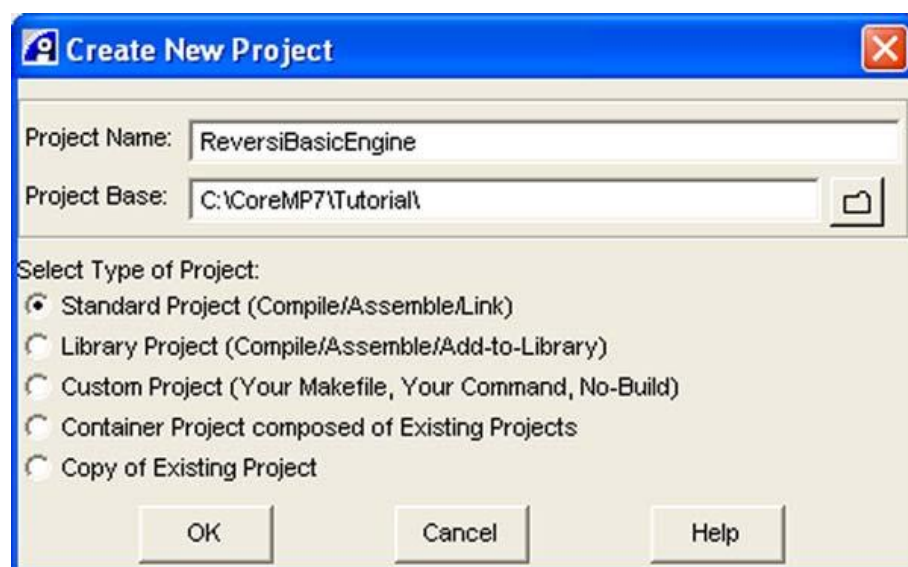


Figure 5-53. RealView Create New Project Dialog Box

4. Click the **Navigate** button (the folder to the right of **Project Base**), select <**Select Dir...**> from the context menu, and browse to the project directory: *C:\CoreMP7\Tutorial*. Click **Select**.
5. Ensure the **Standard Project** radio button is selected, and enter “ReversiBasicEngine” in the **Project Name** field. Click **OK**. This displays the Create Standard Project dialog box, shown in Figure 5-54.

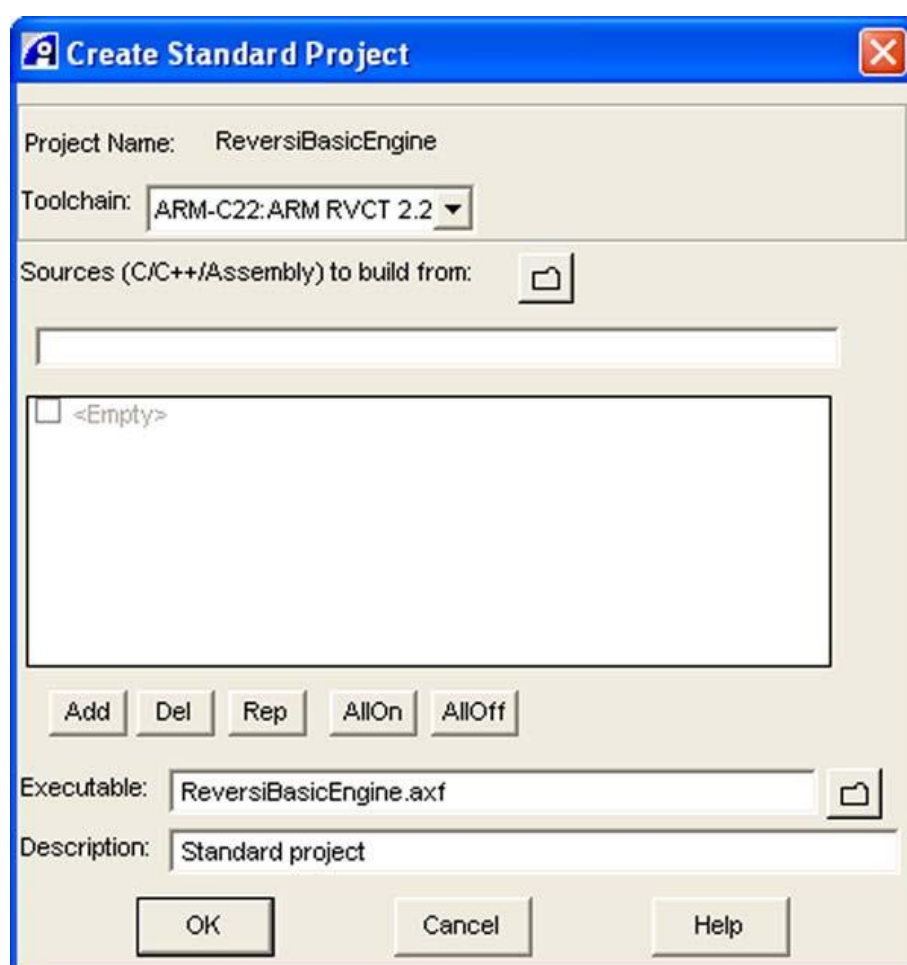


Figure 5-54. RealView Create Standard Project Dialog Box

6. Again, click the navigate button (to the right of **Sources to build from**) to open the Select Source Files for Project dialog box (Figure 5-55).

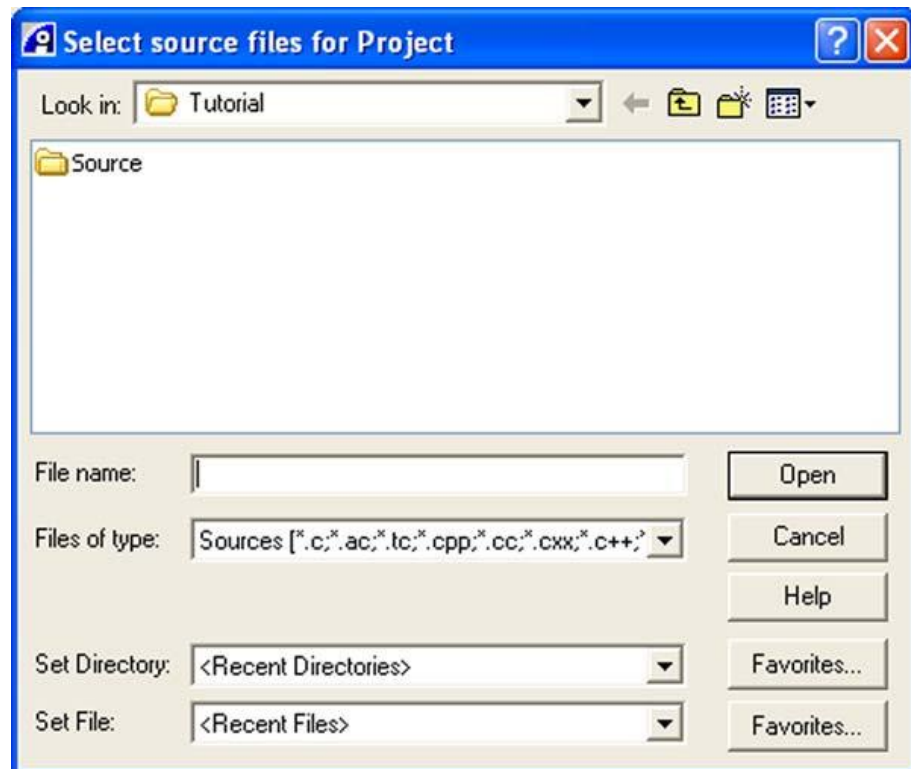


Figure 5-55. RealView Select Source Files for Project Dialog

7. Navigate to the `C:\CoreMP7\Tutorial\Source` directory and press CTRL + A to select all the files within the directory.
8. Click **Open**.

9. Click **OK** in the Create Standard Project dialog box. The Project Properties window now appears (Figure 5-56).

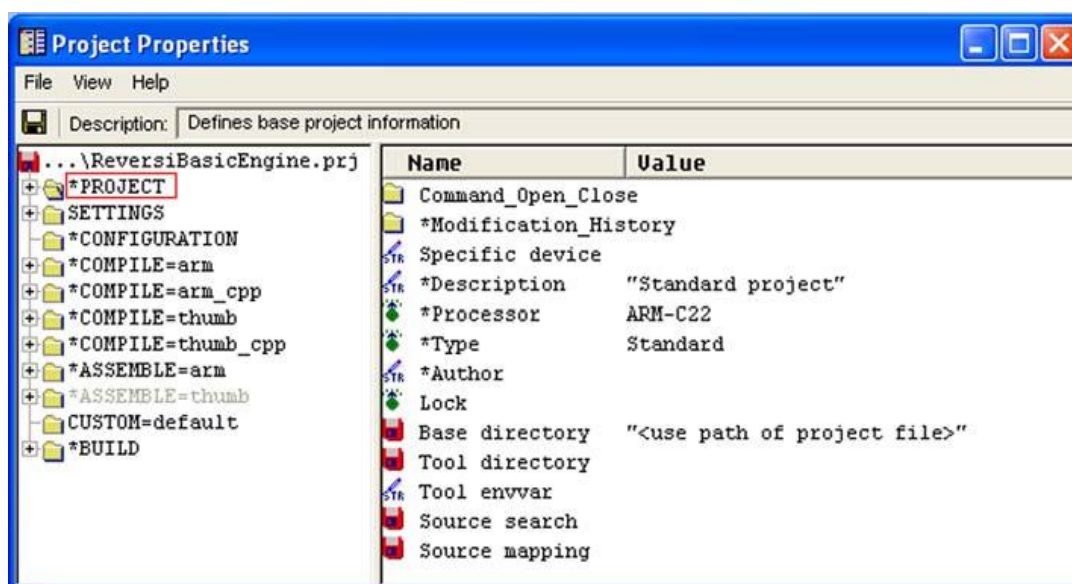


Figure 5-56. RealView Project Properties

10. Click the **CONFIGURATION** entry in the left-hand window pane to view the available project build variants.

11. Right-click the **Active config** entry in the **CONFIGURATION** pane and select **DebugRel** from the context menu, as shown in Figure 5-57.

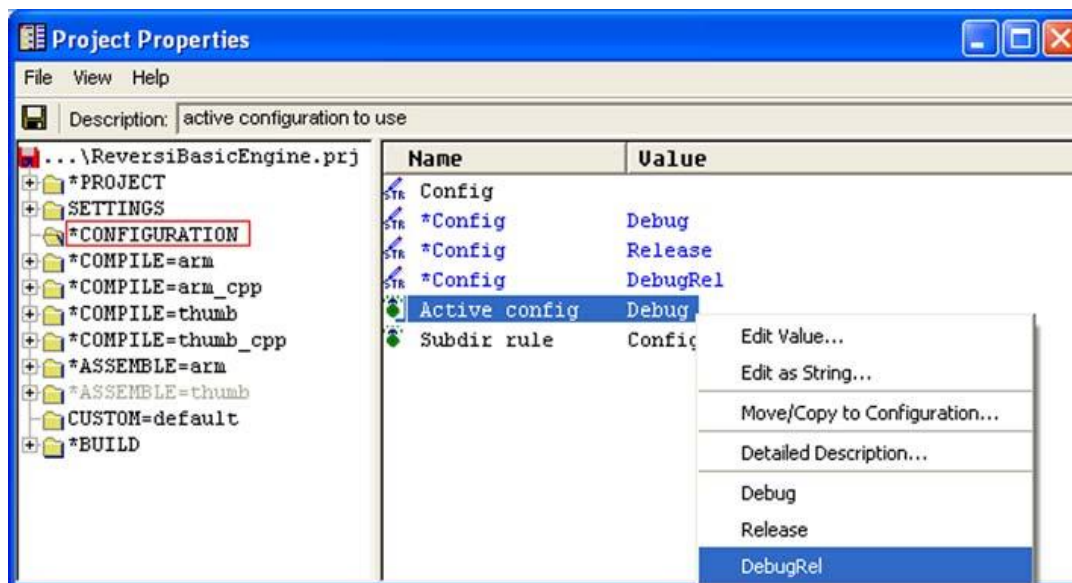


Figure 5-57. RealView Project Properties Configuration Options

The Configuration settings enable you to build your application program in different ways. They define the target configurations used in the build model. The most common target configurations are a Debug build, with debug information and no code optimization, and a Release build, with less debug information and high optimization.

This group can also be used to set up different optimization levels—for example, a DebugRel configuration with higher optimization than Debug but lower than Release. Another example is multiple variants of your application using different device drivers.

See the RealView documentation available at [http://www.arm.com/documentation/Trace\\_Debug/index.html](http://www.arm.com/documentation/Trace_Debug/index.html) or refer to online help for further information on this subject.

12. Expand the **BUILD** entry by clicking the plus sign to the left of its folder, and then select the **Link Advanced** sub-menu component.

13. Right-click the **Scatter file** entry in the **Link Advanced** pane and select **Edit as Filename** from the context menu, as shown in Figure 5-58.

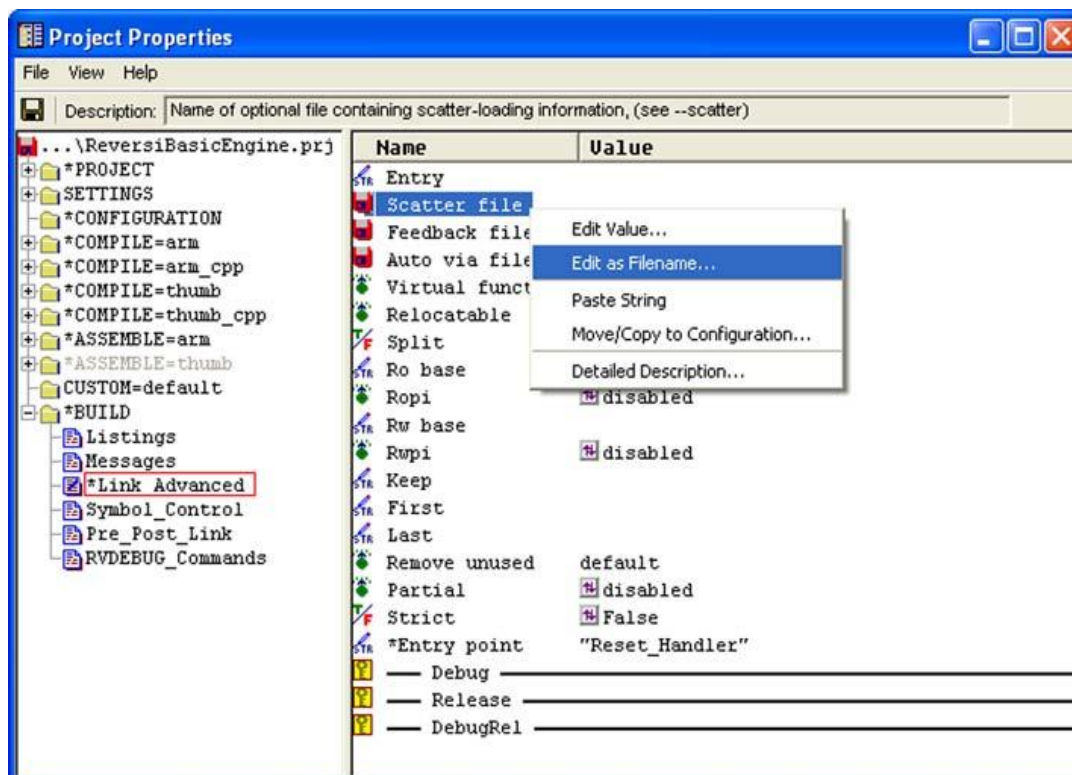


Figure 5-58. RealView Project Properties Build Options

14. Browse to the `..\CoreMP7\Tutorial\Source` directory and select the `CoreMP7DevKit.scf` scatter map description file.

The scatter file is used to tell the linker where to load files or objects residing in memory. For detailed reference information on the linker and scatter-loading, refer to the [ARM Developer Suite Linker and Utilities Guide](#).

The benefit of using a scatter description file is that all the (target-specific) absolute addresses chosen for your devices, code, and data are located in one file, making maintenance easy. Furthermore, if you decide to change your memory map (e.g., if peripherals are moved), you do not need to rebuild your entire project. You only need to re-link the existing objects.

15. Also within the **Link Advanced** menu, set the **Entry point** to **Reset Handler**.
16. From the **File** menu, select **Save and Close**. This will return you to the RealView Debugger desktop where you might see the compiler attempt to compile the source files with a makefile, which fails due to missing files. Ignore this error/warning.



## Step 2 – Compiling the Source Files

## Compiling Source Files

1. From the **Build** menu, select **Build**. Select **Yes** if you are asked if you would like to Rebuild All.

The output of the Build pane appears with several messages generated as a result of the attempted build, as shown in [Figure 5-59](#). If errors were present in your source code, they would be listed with the corresponding filename, line number, and a brief description of the error.

If an error is found, the Code pane of the RealView Debugger opens the relevant source file with an arrow pointing to and highlighting the line of source code the first error message is referencing. Correct the error, save the modified source file, and then rebuild the project, and continue this process until no errors are present upon Compile.



Figure 5-59. RealView Debugger Build Pane

2. If a project is already up-to-date, building will not occur when it is requested. If you wish to do a forced rebuild of all source files, select **Clean** from the **Build** menu, which deletes the relevant object files, and then select **Rebuild All** from the **Build** menu to rebuild the entire project.

### Step 3 – Debugging: Simulating/Executing the Compilation

## Simulator versus Emulator

A simulator attempts to model the entire behavior of a processor in software running on your personal computer. No matter the speed of your PC, there is no simulator which can simulate the microprocessor's real-time behavior. Further, there is no external world communication between the simulator and your target system. Stimulus files must be created and used to simulate external events.

On the other hand, emulators typically replace the processor on the target board and interface directly with the external world. The emulator provides the user with all the features of simulation plus the capabilities of interfacing with the external world and running at full system speed. Emulators exist in two forms: Debug Modules and In-Circuit Emulation (ICE).

The Debug Module approach combines all of the emulator electronics and the actual emulation chip into a single PCB, which connects to the target by ribbon cables, providing a connector that can plug into an actual chip package of the target processor. All signals for the emulated microprocessor pass through the ribbon cable that connects to the target system.



The ICE approach is slightly different. The ICE interfaces directly to the On-Chip Debug system within the actual processor. This provides an interface for complete control of the target processor—typically JTAG, but sometimes a proprietary interface, already embedded into the target processor. The exact electrical and timing characteristics of the target system are achieved when using the On-Chip Debug system, whereas the Debug Module approach may provide additional features and access to internals of the target. For that reason, simulators are best suited for the testing of algorithms.

## Configuring the Simulator / On-Chip Debugger

The frontend tools for performing emulation are exactly the same. The only differences between simulation and emulation are the initial setup steps, specifically steps 3–6. The alternative steps for performing emulation are discussed in “Alternative Steps for Using the On-Chip Debugger” on page 101.

1. Click the **Src** tab in the RealView Debugger Debug window. The Code pane shows that there is currently not a target connected to the debugger.
2. Click the **Connect to Target** link to launch the Connection Control dialog.
3. Expand the **Server > localhost** branches in the name tree, then right-click **new arm** and select **Configure Device Info**, as shown in Figure 5-60.

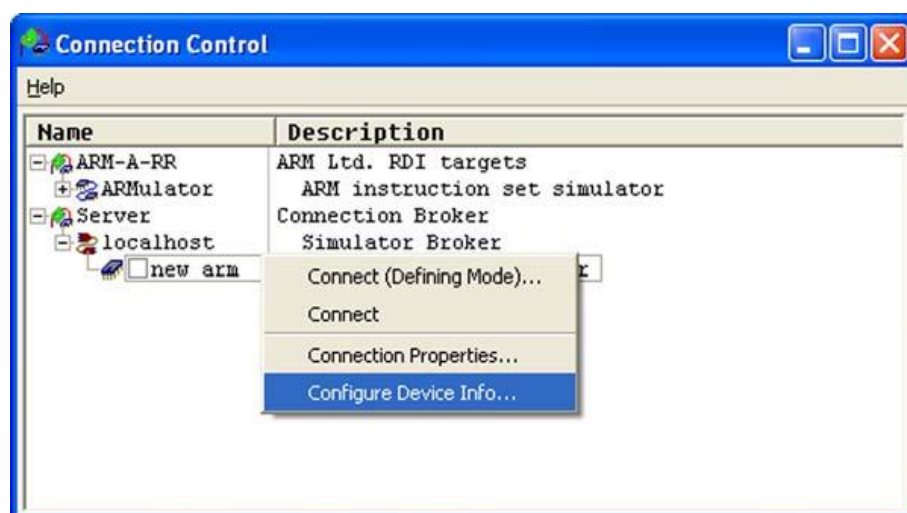


Figure 5-60. RealView Debugger Connection Control Dialog Box

4. The ARMulator Configuration dialog box appears. Select the **ARM7TDMI-S** processor, as shown in Figure 5-61. Click **OK** to return to the Connection Control dialog box.

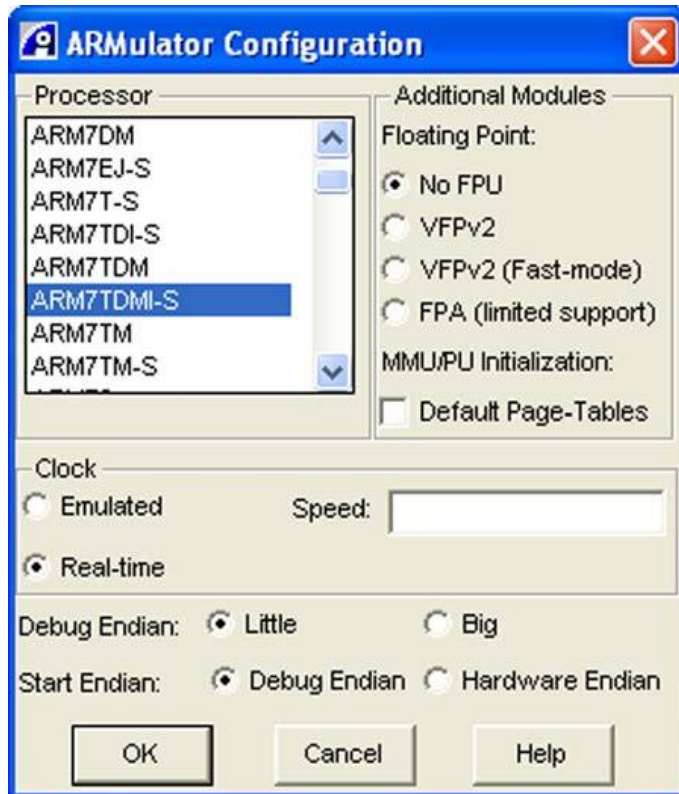


Figure 5-61. ARMulator Configuration Dialog Box

5. Select the **new arm** check box under the name tree. A new simulation object, Simarm 1, will be instantiated, as shown in Figure 5-62.

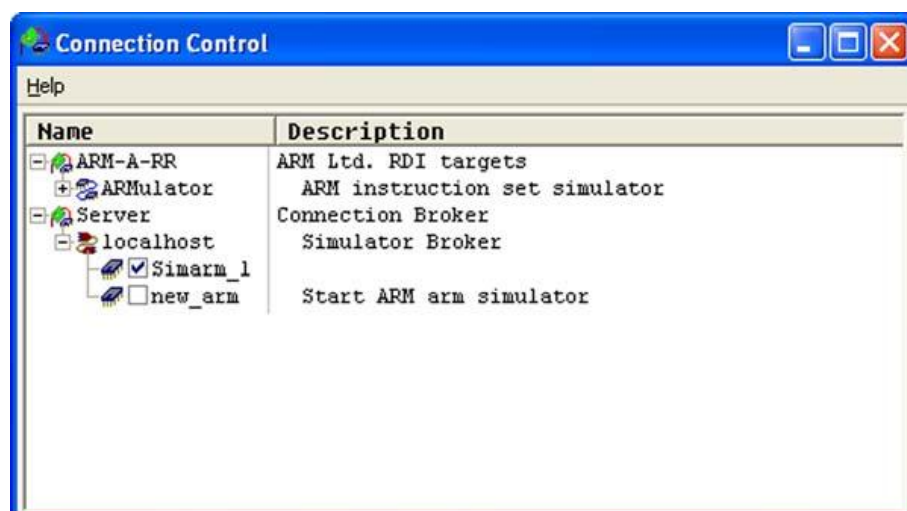
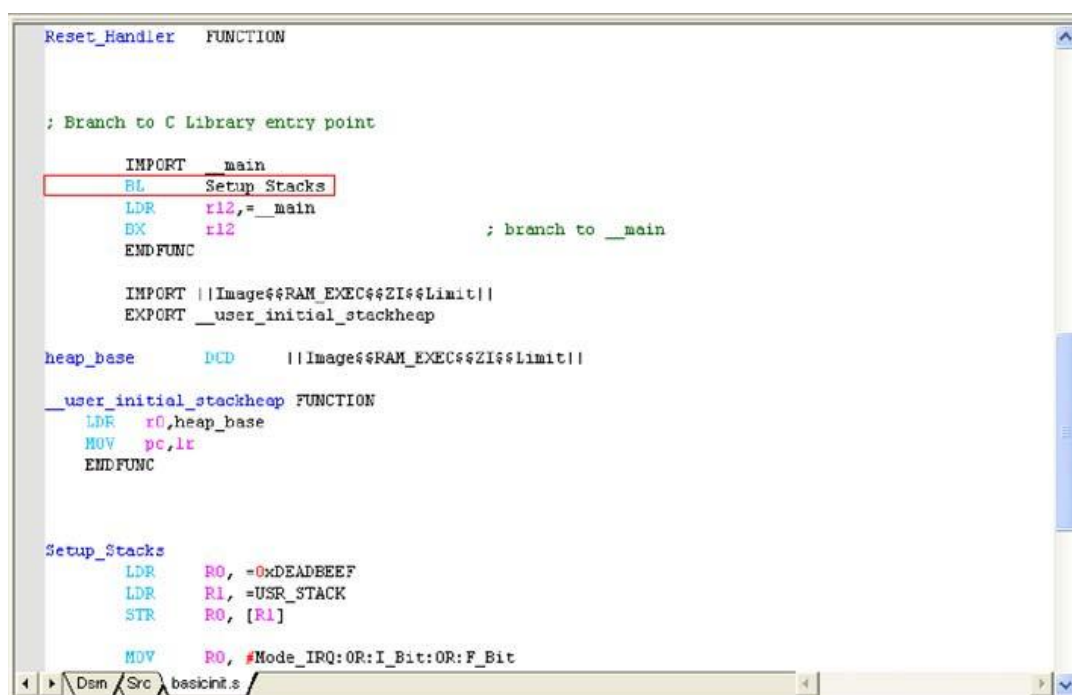


Figure 5-62. RealView Connection Control with Simulation Object

6. The RealView Debugger is now connected to the ARM7TDMI-S Instruction Set Simulator target. You may now close or minimize the Connection Control dialog box.
7. The RealView Debugger Code pane prompts for the loading of the recently built image to the target. Click the **Load** link to load the image. If the Code pane is not prompting you to load an image, click the **Code** tab at the bottom of the Code pane.

- The code is now loaded into the target. The current point of execution is identified by the red box, as shown in Figure 5-63. The code currently being displayed is the basic initialization (or bootloader) code, which is typically written in assembly language.



```

Reset_Handler FUNCTION

; Branch to C Library entry point

IMPORT __main
BL Setup_Stacks
LDR r12,=__main
BX r12 ; branch to __main
ENDFUNC

IMPORT ||Image$$RAM_EXEC$$ZI$Limit||
EXPORT __user_initial_stackheap

heap_base DCD ||Image$$RAM_EXEC$$ZI$Limit||

__user_initial_stackheap FUNCTION
LDR r0,heap_base
MOV pc,r0
ENDFUNC

Setup_Stacks
LDR R0, =0xDEADBEEF
LDR R1, =USR_STACK
STR R0, [R1]

MOV R0, #Mode_IRQ:0R:I_Bit:0R:F_Bit

```

Figure 5-63. RealView Debugger Code Pane with Loaded Target

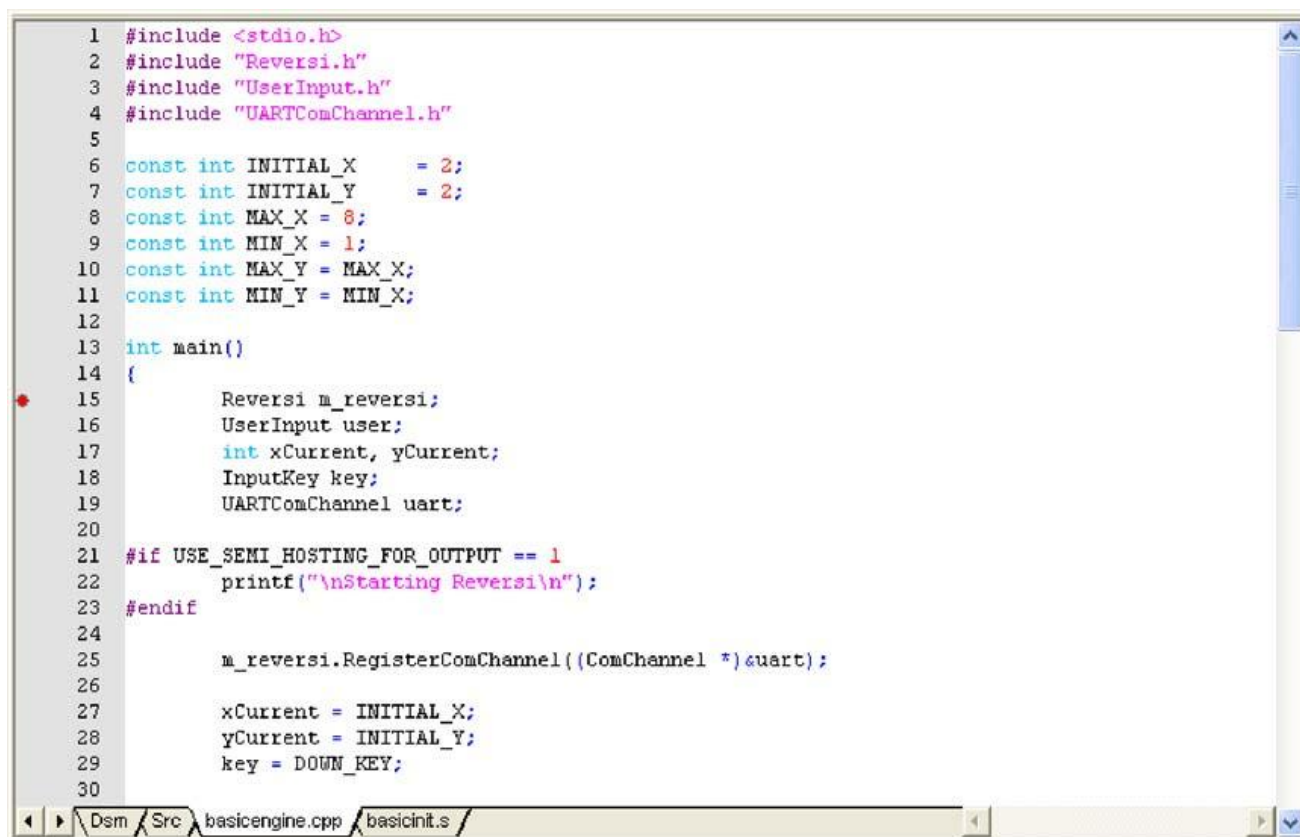
## Debugging the Design

This section will touch on the basics of debugging from the simulation point of view. The primary goal of this section is to learn the very basic features of the debugger, as their application in the following example is rudimentary.

- Open the *basicengine.cpp* source file found in the *..\CoreMP7\Tutorial\Source* directory using the **Open** option under the **File** menu.
- Select **Show Line Numbers** from the **Advanced** sub-menu of the **Edit** menu to display line numbers in the margins of the source code.
- Set a breakpoint on line 15, the “*Reversi m reversi;*” statement, by double-clicking to the left of the line number. A red breakpoint marker will appear to the left of the line number, as shown in Figure 5-64 on page 97.

A *breakpoint* is a user-defined stopping point in a program that is inserted for debugging purposes. Breakpoints are a method embedded developers use to gain information about a

program during its execution. During the *break*, the developer can examine the internal contents of the processor, memory, registers, etc. to ensure proper operation.



```

1 #include <stdio.h>
2 #include "Reversi.h"
3 #include "UserInput.h"
4 #include "UARTComChannel.h"
5
6 const int INITIAL_X = 2;
7 const int INITIAL_Y = 2;
8 const int MAX_X = 8;
9 const int MIN_X = 1;
10 const int MAX_Y = MAX_X;
11 const int MIN_Y = MIN_X;
12
13 int main()
14 {
15 Reversi m_reversi;
16 UserInput user;
17 int xCurrent, yCurrent;
18 InputKey key;
19 UARTComChannel uart;
20
21 #if USE_SEMI_HOSTING_FOR_OUTPUT == 1
22 printf("\nStarting Reversi\n");
23 #endif
24
25 m_reversi.RegisterComChannel((ComChannel *)&uart);
26
27 xCurrent = INITIAL_X;
28 yCurrent = INITIAL_Y;
29 key = DOWN_KEY;
30

```

Figure 5-64. RealView Code Pane with Breakpoint

4. Select **Run** from the **Debug** menu (or use the F5 shortcut key). The cursor (red box) will now be present on line 15 (where the breakpoint was set).
5. Open the *position.cpp* source file and set a breakpoint on line 125, the “m\_board[4][4] = White;” statement.

6. Right-click **m\_board** and select **Watch** from the context menu. The **m\_board** array will be added to the watch window, as shown in [Figure 5-65](#).

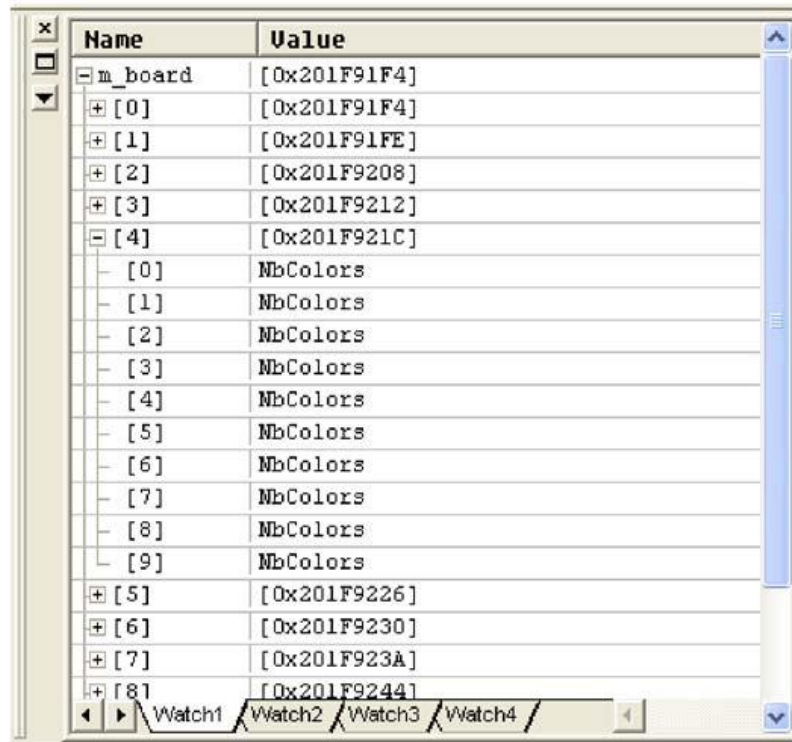


Figure 5-65. RealView Debugger Watch Window

A *watch* is a variable or expression that you require the debugger to display at every step or breakpoint so that you can see how its value changes. The Watch pane is part of the RealView Debugger Code window and displays the watches you have defined.

7. Within the watch window, expand the **m board** variable and then expand the **[4]** sub-component (a dimension of the *m board* array).
8. Select **Step Into** from the **Debug** menu (or use the F11 shortcut key). The cursor (red box) will now be present on line 126. Looking at the watch window, you can see that the *m board*[4][4] contents have changed from the default “nbColors” to the value “White.”
9. Continuing to single-step (or **Step Into**) three more times, you will see changes in the [5][4], [5][5], and [4][5] components of the *m board* multi-dimensional array.
10. Select **Registers** from the **View** menu. The Register window is displayed; you may choose to “dock” it into the RealView Debugger, as shown in [Figure 5-66](#).

The *Register window* displays the contents of the internal processor registers at every step or breakpoint so that you can see how its value changes. The register contents may also be modified through this interface by simply double-clicking the value and modifying the field.

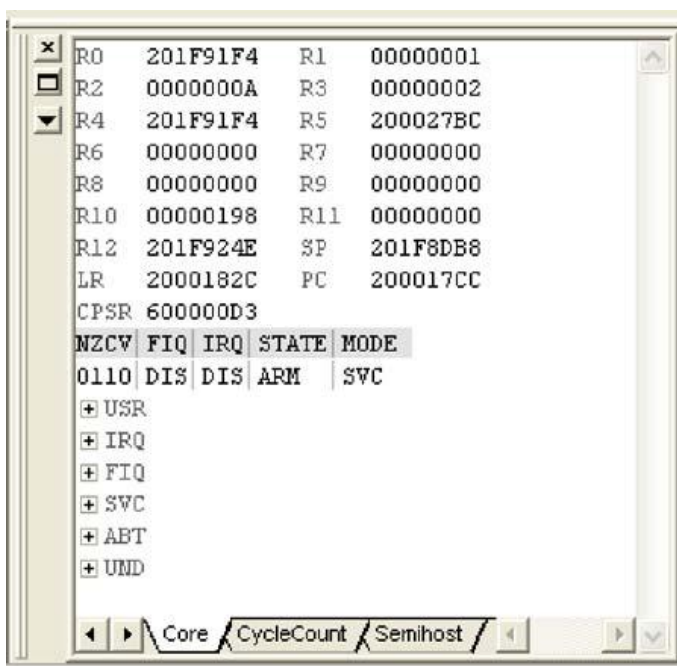


Figure 5-66. RealView Debugger Register Window



11. Right-click one of the <<NoAddr>> values in the RealView Debugger Memory window and select **Set Start Address**. Enter the value **0xC2000000**. The Memory window will now display the memory segment contents beginning with the aforementioned address, as shown in [Figure 5-67](#).

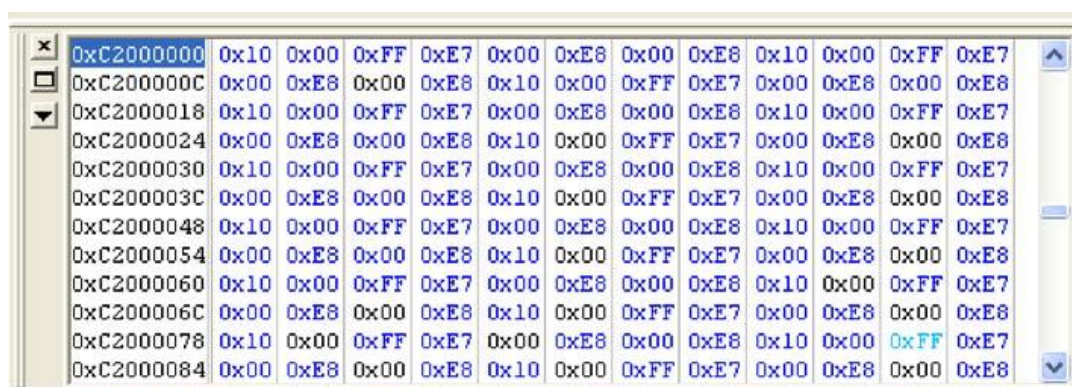


Figure 5-67. RealView Debugger Memory Window

The *Memory window* displays the contents of the memories (both Flash and SRAM), as well as the memory-mapped peripheral configuration registers, at every step or breakpoint so that you can see how its value changes. The memory contents may also be modified through this interface by simply double-clicking the value and modifying the field.

12. If you are debugging with the emulator, you can modify the memory contents at 0xC2000000 and twiddle the LEDs on the CoreMP7 Evaluation Board, thus showing that you have direct access to the on-chip peripherals.
13. If you wish to run the complete Reversi (also known as Othello) game on the CoreMP7 Development Kit, continue with [“Running the Reversi Game via the On-Chip Debugger” on page 104](#). Otherwise, end the debugging session by selecting **Disconnect** from the **Target** menu.



## Alternative Steps for Using the On-Chip Debugger

1. Connect the RVI-ME to the CoreMP7 Evaluation Board through the ARM JTAG header and power up the board.
2. Expand the **ARM-ARM-USB** branch to reveal **RVI-ME** in the name tree. Right-click **RVI-ME** and select **Configure Device Info**, as shown in Figure 5-68. This launches the RVConfig utility.

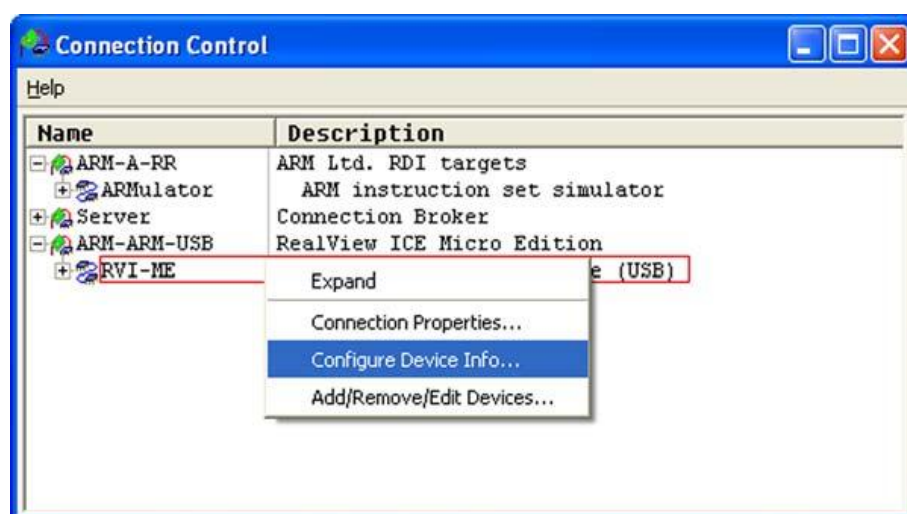


Figure 5-68. RealView Debugger RVI-ME Configure Device

3. Click the **Auto Configure Scan Chain** button in the RVConfig utility. After a few seconds, the ARM7TDMI-S processor will be identified, as shown in [Figure 5-69](#).

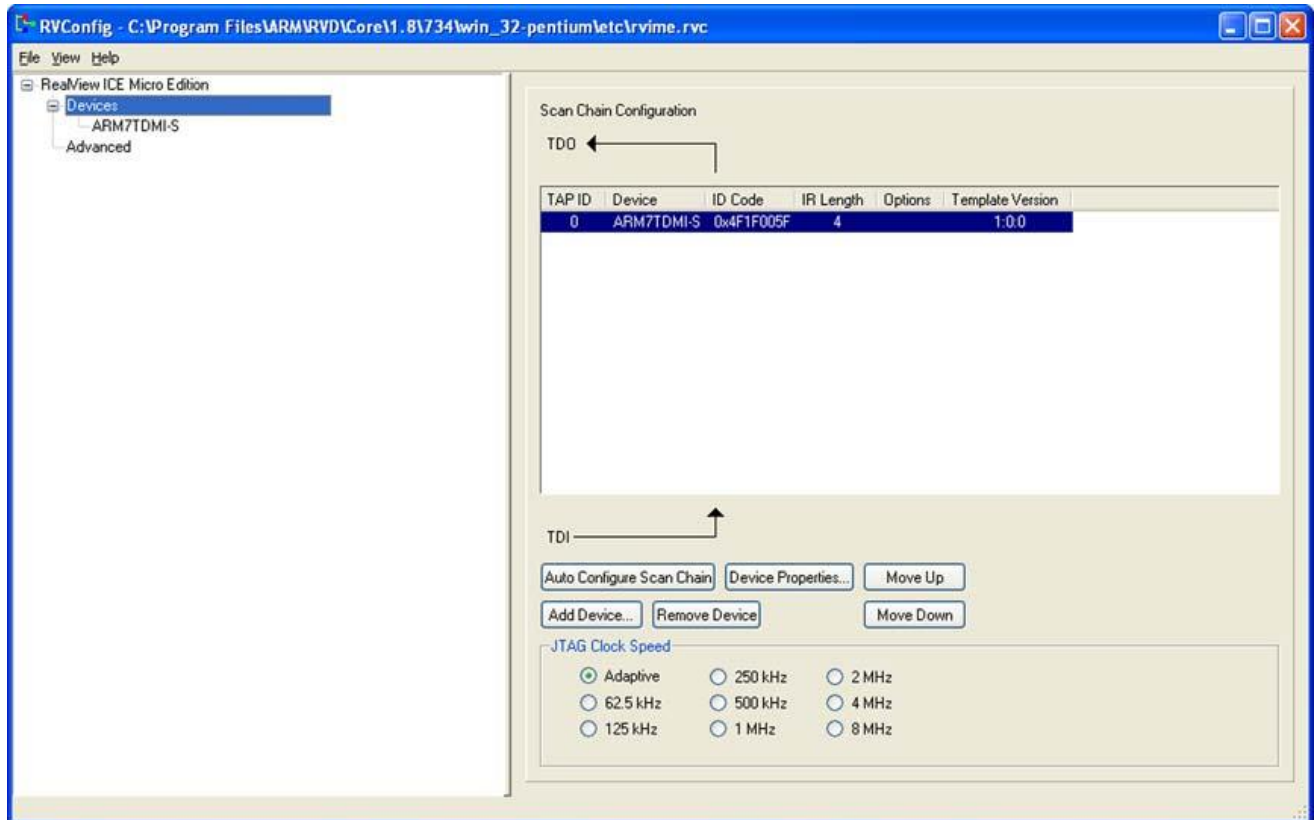


Figure 5-69. RealView Debugger RVConfig Dialog Box

4. From the **File** menu, select **Save**. Then select **File > Exit** and return to the Configuration Control dialog box.
5. Select the **ARM7TDMI-S** check box under the Name tree. It may be necessary to expand the **RVI-ME** branch first, as shown in [Figure 5-70](#).

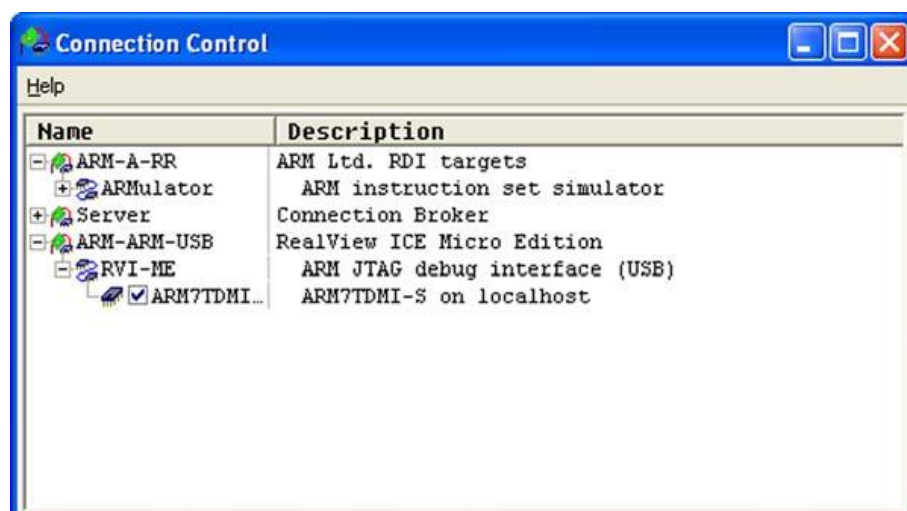


Figure 5-70. RealView Debugger Connection Control with On-Chip Debugger Target

6. The RealView Debugger is now connected to the ARM7TDMI-S via the On-Chip Debugger. You may now close or minimize the Connection Control dialog box.
7. Continue with the Quickstart Tutorial with step 7 on [page 95](#).

## Running the Reversi Game via the On-Chip Debugger

1. Reload the *ReversiBasicEngine.axf* file by selecting **Reload Image to Target** from the **Debug** menu.
2. Connect the RS-232 connector P2 to the COM1 port of your PC using a standard straight-through serial cable.
3. From the **Debug** menu, select **Run** (or use the F11 shortcut key). There will be an indicator in the RealView Debugger status bar which indicates that the target is *Running* with a small progress bar.
4. Minimize the RealView Debugger tool.
5. Navigate to the `..\Tutorial\Demo` directory on the Tutorial CD-ROM and double-click the *ReversiGUI.exe* executable.
6. Pressing SW6, SW7, SW8, SW9, or SW10 will commence the game.
7. To end the game, select **Stop Execution** from the **Debug** menu (or use the SHIFT + F5 shortcut key).
8. Disconnect from the target by selecting **Disconnect** from the **Target** menu. It is now safe to close the RealView Debugger.

## Reversi Background

The object of the game is to own more pieces than your opponent when the game is over. The game is over when neither player has a move. Usually, this means the board is full. On your turn, you place one piece on the board with your color facing up. You must place the piece so that your opponent's piece, or a row of your opponent's pieces, is flanked by your pieces. All of the opponent's pieces between your pieces are then turned over to become your color.

## Game Controls

- SW8: Places a piece
- SW9: Moves the placement box to the left one unit
- SW10: Moves the placement box to the right one unit
- SW7: Moves the placement box down one unit
- SW6: Moves the placement box up one unit

# M7A3PE600 and M7A3P1000 FG484

## Package Connections

Due to the comprehensive and flexible nature of M7 ProASIC3/E device user I/Os, a naming scheme is used to show the details of each I/O. The name identifies the I/O bank to which the I/O belongs as well as the pairing and pin polarity for differential I/Os.

I/O nomenclature: Gmn or IOuxwBy

Gmn is only used for I/Os that also have CCC access, i.e., global pins.

- G: Global
- m: Global pin location associated with each CCC on the device: A (northwest corner), B (northeast corner), C (east middle), D (southeast corner), E (southwest corner), and F (west middle)
- n: Global input MUX and pin number of the associated global location m: either A0, A1, A2, B0, B1, B2, C0, C1, or C2
- u: I/O pair number in the bank, starting at 00 from the northwest I/O bank and proceeding in a clockwise direction
- x: P (positive) or N (negative) for differential pairs, or R (regular – single-ended) for I/Os that support single-ended and voltage-referenced I/O standards only. U (positive-LVDS only) or V (negative-LVDS only) restricts the I/O differential pair from being selected as an LVPECL pair.
- w: D (differential pair), P (pair) or S (single-ended). D if both members of the pair are bonded out to adjacent pins or are separated only by one GND or NC pin, P if both members of the pair are bonded out but do not meet the adjacency requirement, or S if the I/O pair is not bonded out. For differential pairs, adjacency for ball grid packages means only vertical or horizontal. Diagonal adjacency does not meet the requirements for a true differential pair.
- B: Bank
- y: Bank number [0..3] for M7 ProASIC3 and [0..7] for M7 ProASIC3E. The bank number starts at 0 from the northwest I/O bank and proceeds in a clockwise direction.

Figure A-1 on page 106 and Table A-1 on page 107 are extracted from the *ProASIC3* and *ProASIC3E* datasheets and provide package connections for the M7A3PE600 and M7A3P1000 devices.

Pinouts for other devices in the FG484 family may be found on the Actel website:

*ProASIC3 Flash Family FPGAs datasheet* at [www.actel.com/documents/PA3\\_DS.pdf](http://www.actel.com/documents/PA3_DS.pdf)

*ProASIC3E Flash Family FPGAs datasheet* at [www.actel.com/documents/PA3E\\_DS.pdf](http://www.actel.com/documents/PA3E_DS.pdf)

These datasheets are included on the CoreMP7 Development Kit CD. However, always refer to the website for the most recent datasheet.

## 484-Pin FGBGA Package

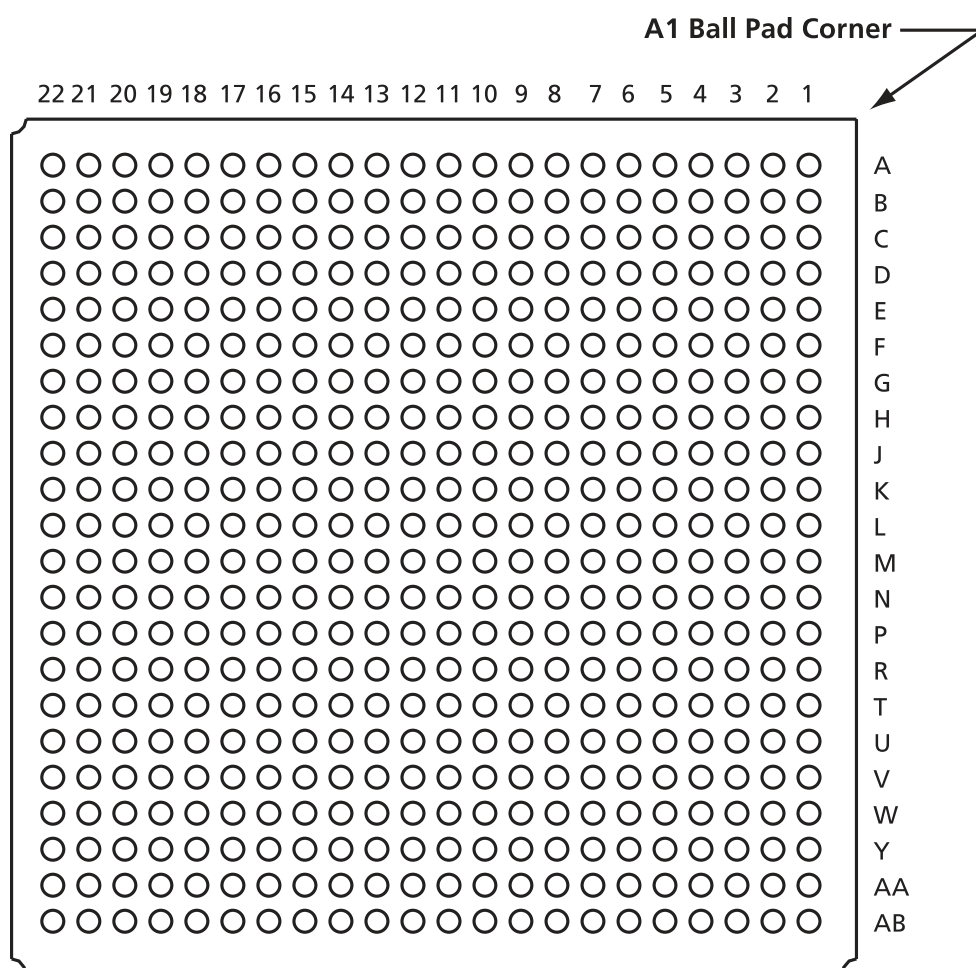


Figure A-1. 484-Pin FGBGA Package Layout

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function  |
|------------|---------------------|---------------------|
| A1         | GND                 | GND                 |
| A2         | GND                 | GND                 |
| A3         | V <sub>CCI</sub> B0 | V <sub>CCI</sub> B0 |
| A4         | IO07RSB0            | IO06NDB0V1          |
| A5         | IO09RSB0            | IO06PDB0V1          |
| A6         | IO13RSB0            | IO08NDB0V1          |
| A7         | IO18RSB0            | IO08PDB0V1          |
| A8         | IO20RSB0            | IO11PDB0V1          |
| A9         | IO26RSB0            | IO17PDB0V2          |
| A10        | IO32RSB0            | IO18NDB0V2          |
| A11        | IO40RSB0            | IO18PDB0V2          |
| A12        | IO41RSB0            | IO22PDB1V0          |
| A13        | IO53RSB0            | IO26PDB1V0          |
| A14        | IO59RSB0            | IO29NDB1V1          |
| A15        | IO64RSB0            | IO29PDB1V1          |
| A16        | IO65RSB0            | IO31NDB1V1          |
| A17        | IO67RSB0            | IO31PDB1V1          |
| A18        | IO69RSB0            | IO32NDB1V1          |
| A19        | NC                  | NC                  |
| A20        | V <sub>CCI</sub> B0 | V <sub>CCI</sub> B1 |
| A21        | GND                 | GND                 |
| A22        | GND                 | GND                 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function  |
|------------|---------------------|---------------------|
| AA1        | GND                 | GND                 |
| AA2        | V <sub>CCI</sub> B3 | V <sub>CCI</sub> B7 |
| AA3        | NC                  | NC                  |
| AA4        | IO181RSB2           | IO03NDB0V0          |
| AA5        | IO178RSB2           | IO03PDB0V0          |
| AA6        | IO175RSB2           | IO07NDB0V1          |
| AA7        | IO169RSB2           | IO07PDB0V1          |
| AA8        | IO166RSB2           | IO11NDB0V1          |
| AA9        | IO160RSB2           | IO17NDB0V2          |
| AA10       | IO152RSB2           | IO14PDB0V2          |
| AA11       | IO146RSB2           | IO19PDB0V2          |
| AA12       | IO139RSB2           | IO22NDB1V0          |
| AA13       | IO133RSB2           | IO26NDB1V0          |
| AA14       | NC                  | NC                  |
| AA15       | NC                  | NC                  |
| AA16       | IO122RSB2           | IO30NDB1V1          |
| AA17       | IO119RSB2           | IO30PDB1V1          |
| AA18       | IO117RSB2           | IO32PDB1V1          |
| AA19       | NC                  | NC                  |
| AA20       | NC                  | NC                  |
| AA21       | V <sub>CCI</sub> B1 | V <sub>CCI</sub> B2 |
| AA22       | GND                 | GND                 |



Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function  |
|------------|---------------------|---------------------|
| AB1        | GND                 | V <sub>CCI</sub> B7 |
| AB2        | GND                 | NC                  |
| AB3        | V <sub>CCI</sub> B2 | NC                  |
| AB4        | IO180RSB2           | NC                  |
| AB5        | IO176RSB2           | GND                 |
| AB6        | IO173RSB2           | IO04NDB0V0          |
| AB7        | IO167RSB2           | IO04PDB0V0          |
| AB8        | IO162RSB2           | V <sub>CC</sub>     |
| AB9        | IO156RSB2           | V <sub>CC</sub>     |
| AB10       | IO150RSB2           | IO14NDB0V2          |
| AB11       | IO145RSB2           | IO19NDB0V2          |
| AB12       | IO144RSB2           | NC                  |
| AB13       | IO132RSB2           | NC                  |
| AB14       | IO127RSB2           | V <sub>CC</sub>     |
| AB15       | IO126RSB2           | V <sub>CC</sub>     |
| AB16       | IO123RSB2           | NC                  |
| AB17       | IO121RSB2           | NC                  |
| AB18       | IO118RSB2           | GND                 |
| AB19       | NC                  | NC                  |
| AB20       | V <sub>CCI</sub> B2 | NC                  |
| AB21       | GND                 | NC                  |
| AB22       | GND                 | V <sub>CCI</sub> B2 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function |
|------------|---------------------|--------------------|
| B1         | GND                 | NC                 |
| B2         | V <sub>CCI</sub> B3 | NC                 |
| B3         | NC                  | NC                 |
| B4         | IO06RSB0            | GND                |
| B5         | IO08RSB0            | GAA0/IO00NDB0V0    |
| B6         | IO12RSB0            | GAA1/IO00PDB0V0    |
| B7         | IO15RSB0            | GAB0/IO01NDB0V0    |
| B8         | IO19RSB0            | IO05PDB0V0         |
| B9         | IO24RSB0            | IO10PDB0V1         |
| B10        | IO31RSB0            | IO12PDB0V2         |
| B11        | IO39RSB0            | IO16NDB0V2         |
| B12        | IO48RSB0            | IO23NDB1V0         |
| B13        | IO54RSB0            | IO23PDB1V0         |
| B14        | IO58RSB0            | IO28NDB1V1         |
| B15        | IO63RSB0            | IO28PDB1V1         |
| B16        | IO66RSB0            | GBB1/IO34PDB1V1    |
| B17        | IO68RSB0            | GBA0/IO35NDB1V1    |
| B18        | IO70RSB0            | GBA1/IO35PDB1V1    |
| B19        | NC                  | GND                |
| B20        | NC                  | NC                 |
| B21        | V <sub>CCI</sub> B1 | NC                 |
| B22        | GND                 | NC                 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function |
|------------|---------------------|--------------------|
| C1         | V <sub>CCI</sub> B3 | NC                 |
| C2         | IO220PDB3           | NC                 |
| C3         | NC                  | GND                |
| C4         | NC                  | GAB2/IO133PDB7V1   |
| C5         | GND                 | GAA2/IO134PDB7V1   |
| C6         | IO10RSB0            | GNDQ               |
| C7         | IO14RSB0            | GAB1/IO01PDB0V0    |
| C8         | V <sub>CC</sub>     | IO05NDB0V0         |
| C9         | V <sub>CC</sub>     | IO10NDB0V1         |
| C10        | IO30RSB0            | IO12NDB0V2         |
| C11        | IO37RSB0            | IO16PDB0V2         |
| C12        | IO43RSB0            | IO20NDB1V0         |
| C13        | NC                  | IO24NDB1V0         |
| C14        | V <sub>CC</sub>     | IO24PDB1V0         |
| C15        | V <sub>CC</sub>     | GBC1/IO33PDB1V1    |
| C16        | NC                  | GBB0/IO34NDB1V1    |
| C17        | NC                  | GNDQ               |
| C18        | GND                 | GBA2/IO36PDB2V0    |
| C19        | NC                  | IO42NDB2V0         |
| C20        | NC                  | GND                |
| C21        | NC                  | NC                 |
| C22        | V <sub>CCI</sub> B1 | NC                 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function |
|------------|--------------------|--------------------|
| D1         | IO219PDB3          | NC                 |
| D2         | IO220NDB3          | IO131NDB7V1        |
| D3         | NC                 | IO131PDB7V1        |
| D4         | GND                | IO133NDB7V1        |
| D5         | GAA0/IO00RSB0      | IO134NDB7V1        |
| D6         | GAA1/IO01RSB0      | VMV7               |
| D7         | GAB0/IO02RSB0      | V <sub>CCPLA</sub> |
| D8         | IO16RSB0           | GAC0/IO02NDB0V0    |
| D9         | IO22RSB0           | GAC1/IO02PDB0V0    |
| D10        | IO28RSB0           | IO15NDB0V2         |
| D11        | IO35RSB0           | IO15PDB0V2         |
| D12        | IO45RSB0           | IO20PDB1V0         |
| D13        | IO50RSB0           | IO25NDB1V0         |
| D14        | IO55RSB0           | IO27PDB1V0         |
| D15        | IO61RSB0           | GBC0/IO33NDB1V1    |
| D16        | GBB1/IO75RSB0      | V <sub>CCPLB</sub> |
| D17        | GBA0/IO76RSB0      | VMV2               |
| D18        | GBA1/IO77RSB0      | IO36NDB2V0         |
| D19        | GND                | IO42PDB2V0         |
| D20        | NC                 | NC                 |
| D21        | NC                 | NC                 |
| D22        | NC                 | NC                 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function  |
|------------|--------------------|---------------------|
| E1         | IO219NDB3          | IO127NDB7V1         |
| E2         | NC                 | IO127PDB7V1         |
| E3         | GND                | NC                  |
| E4         | GAB2/IO224PDB3     | IO128PDB7V1         |
| E5         | GAA2/IO225PDB3     | IO129PDB7V1         |
| E6         | GNDQ               | GAC2/IO132PDB7V1    |
| E7         | GAB1/IO03RSB0      | V <sub>COMPLA</sub> |
| E8         | IO17RSB0           | GNDQ                |
| E9         | IO21RSB0           | IO09NDB0V1          |
| E10        | IO27RSB0           | IO09PDB0V1          |
| E11        | IO34RSB0           | IO13PDB0V2          |
| E12        | IO44RSB0           | IO21PDB1V0          |
| E13        | IO51RSB0           | IO25PDB1V0          |
| E14        | IO57RSB0           | IO27NDB1V0          |
| E15        | GBC1/IO73RSB0      | GNDQ                |
| E16        | GBB0/IO74RSB0      | V <sub>COMPLB</sub> |
| E17        | IO71RSB0           | GBB2/IO37PDB2V0     |
| E18        | GBA2/IO78PDB1      | IO39PDB2V0          |
| E19        | IO81PDB1           | IO39NDB2V0          |
| E20        | GND                | IO43PDB2V0          |
| E21        | NC                 | IO43NDB2V0          |
| E22        | IO84PDB1           | NC                  |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function  |
|------------|--------------------|---------------------|
| F1         | NC                 | NC                  |
| F2         | IO215PDB3          | NC                  |
| F3         | IO215NDB3          | V <sub>CC</sub>     |
| F4         | IO224NDB3          | IO128NDB7V1         |
| F5         | IO225NDB3          | IO129NDB7V1         |
| F6         | VMV3               | IO132NDB7V1         |
| F7         | IO11RSB0           | IO130PDB7V1         |
| F8         | GAC0/IO04RSB0      | VMV0                |
| F9         | GAC1/IO05RSB0      | V <sub>CCI</sub> B0 |
| F10        | IO25RSB0           | V <sub>CCI</sub> B0 |
| F11        | IO36RSB0           | IO13NDB0V2          |
| F12        | IO42RSB0           | IO21NDB1V0          |
| F13        | IO49RSB0           | V <sub>CCI</sub> B1 |
| F14        | IO56RSB0           | V <sub>CCI</sub> B1 |
| F15        | GBC0/IO72RSB0      | VMV1                |
| F16        | IO62RSB0           | GBC2/IO38PDB2V0     |
| F17        | VMV0               | IO37NDB2V0          |
| F18        | IO78NDB1           | IO41NDB2V0          |
| F19        | IO81NDB1           | IO41PDB2V0          |
| F20        | IO82PPB1           | V <sub>CC</sub>     |
| F21        | NC                 | NC                  |
| F22        | IO84NDB1           | NC                  |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function  |
|------------|--------------------|---------------------|
| G1         | IO214NDB3          | IO123NDB7V0         |
| G2         | IO214PDB3          | IO123PDB7V0         |
| G3         | NC                 | NC                  |
| G4         | IO222NDB3          | IO124PDB7V0         |
| G5         | IO222PDB3          | IO125PDB7V0         |
| G6         | GAC2/IO223PDB3     | IO126PDB7V0         |
| G7         | IO223NDB3          | IO130NDB7V1         |
| G8         | GNDQ               | V <sub>CCI</sub> B7 |
| G9         | IO23RSB0           | GND                 |
| G10        | IO29RSB0           | V <sub>CC</sub>     |
| G11        | IO33RSB0           | V <sub>CC</sub>     |
| G12        | IO46RSB0           | V <sub>CC</sub>     |
| G13        | IO52RSB0           | V <sub>CC</sub>     |
| G14        | IO60RSB0           | GND                 |
| G15        | GNDQ               | V <sub>CCI</sub> B2 |
| G16        | IO80NDB1           | IO38NDB2V0          |
| G17        | GBB2/IO79PDB1      | IO40NDB2V0          |
| G18        | IO79NDB1           | IO40PDB2V0          |
| G19        | IO82NPB1           | IO45PDB2V1          |
| G20        | IO85PDB1           | NC                  |
| G21        | IO85NDB1           | IO48PDB2V1          |
| G22        | NC                 | IO46PDB2V1          |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function  |
|------------|---------------------|---------------------|
| H1         | NC                  | IO121NDB7V0         |
| H2         | NC                  | IO121PDB7V0         |
| H3         | V <sub>CC</sub>     | NC                  |
| H4         | IO217PDB3           | IO124NDB7V0         |
| H5         | IO218PDB3           | IO125NDB7V0         |
| H6         | IO221NDB3           | IO126NDB7V0         |
| H7         | IO221PDB3           | GFC1/IO120PPB7V0    |
| H8         | VMV0                | V <sub>CCI</sub> B7 |
| H9         | V <sub>CCI</sub> B0 | V <sub>CC</sub>     |
| H10        | V <sub>CCI</sub> B0 | GND                 |
| H11        | IO38RSB0            | GND                 |
| H12        | IO47RSB0            | GND                 |
| H13        | V <sub>CCI</sub> B0 | GND                 |
| H14        | V <sub>CCI</sub> B0 | V <sub>CC</sub>     |
| H15        | VMV1                | V <sub>CCI</sub> B2 |
| H16        | GBC2/IO80PDB1       | GCC1/IO50PPB2V1     |
| H17        | IO83PPB1            | IO44NDB2V1          |
| H18        | IO86PPB1            | IO44PDB2V1          |
| H19        | IO87PDB1            | IO49NPB2V1          |
| H20        | V <sub>CC</sub>     | IO45NDB2V1          |
| H21        | NC                  | IO48NDB2V1          |
| H22        | NC                  | IO46NDB2V1          |



Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function  |
|------------|---------------------|---------------------|
| J1         | IO212NDB3           | NC                  |
| J2         | IO212PDB3           | IO122PDB7V0         |
| J3         | NC                  | IO122NDB7V0         |
| J4         | IO217NDB3           | GFB0/IO119NPB7V0    |
| J5         | IO218NDB3           | GFA0/IO118NDB6V1    |
| J6         | IO216PDB3           | GFB1/IO119PPB7V0    |
| J7         | IO216NDB3           | V <sub>COMPLF</sub> |
| J8         | V <sub>CCI</sub> B3 | GFC0/IO120NPB7V0    |
| J9         | GND                 | V <sub>CC</sub>     |
| J10        | V <sub>CC</sub>     | GND                 |
| J11        | V <sub>CC</sub>     | GND                 |
| J12        | V <sub>CC</sub>     | GND                 |
| J13        | V <sub>CC</sub>     | GND                 |
| J14        | GND                 | V <sub>CC</sub>     |
| J15        | V <sub>CCI</sub> B1 | GCC0/IO50NPB2V1     |
| J16        | IO83NPB1            | GCB1/IO51PPB2V1     |
| J17        | IO86NPB1            | GCA0/IO52NPB3V0     |
| J18        | IO90PPB1            | V <sub>COMPLC</sub> |
| J19        | IO87NDB1            | GCB0/IO51NPB2V1     |
| J20        | NC                  | IO49PPB2V1          |
| J21        | IO89PDB1            | IO47NDB2V1          |
| J22        | IO89NDB1            | IO47PDB2V1          |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function |
|------------|---------------------|--------------------|
| K1         | IO211PDB3           | NC                 |
| K2         | IO211NDB3           | IO114NDB6V1        |
| K3         | NC                  | IO117NDB6V1        |
| K4         | IO210PPB3           | GFA2/IO117PDB6V1   |
| K5         | IO213NDB3           | GFA1/IO118PDB6V1   |
| K6         | IO213PDB3           | V <sub>CCPLF</sub> |
| K7         | GFC1/IO209PPB3      | IO116NDB6V1        |
| K8         | V <sub>CCI</sub> B3 | GFB2/IO116PDB6V1   |
| K9         | V <sub>CC</sub>     | V <sub>CC</sub>    |
| K10        | GND                 | GND                |
| K11        | GND                 | GND                |
| K12        | GND                 | GND                |
| K13        | GND                 | GND                |
| K14        | V <sub>CC</sub>     | V <sub>CC</sub>    |
| K15        | V <sub>CCI</sub> B1 | GCB2/IO54PPB3V0    |
| K16        | GCC1/IO91PPB1       | GCA1/IO52PPB3V0    |
| K17        | IO90NPB1            | GCC2/IO55PPB3V0    |
| K18        | IO88PDB1            | V <sub>CCPLC</sub> |
| K19        | IO88NDB1            | GCA2/IO53PDB3V0    |
| K20        | IO94NPB1            | IO53NDB3V0         |
| K21        | IO98NDB1            | IO56PDB3V0         |
| K22        | IO98PDB1            | NC                 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function  |
|------------|---------------------|---------------------|
| L1         | NC                  | IO114PDB6V1         |
| L2         | IO200PDB3           | IO111NDB6V1         |
| L3         | IO210NPB3           | NC                  |
| L4         | GFB0/IO208NPB3      | GFC2/IO115PDB6V1    |
| L5         | GFA0/IO207NDB3      | IO113PPB6V1         |
| L6         | GFB1/IO208PPB3      | IO112PDB6V1         |
| L7         | V <sub>COMPLF</sub> | IO112NDB6V1         |
| L8         | GFC0/IO209NPB3      | V <sub>CCI</sub> B6 |
| L9         | V <sub>CC</sub>     | V <sub>CC</sub>     |
| L10        | GND                 | GND                 |
| L11        | GND                 | GND                 |
| L12        | GND                 | GND                 |
| L13        | GND                 | GND                 |
| L14        | V <sub>CC</sub>     | V <sub>CC</sub>     |
| L15        | GCC0/IO91NPB1       | V <sub>CCI</sub> B3 |
| L16        | GCB1/IO92PPB1       | IO54NPB3V0          |
| L17        | GCA0/IO93NPB1       | IO57NPB3V0          |
| L18        | IO96NPB1            | IO55NPB3V0          |
| L19        | GCB0/IO92NPB1       | IO57PPB3V0          |
| L20        | IO97PDB1            | NC                  |
| L21        | IO97NDB1            | IO56NDB3V0          |
| L22        | IO99NPB1            | IO58PDB3V0          |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function  |
|------------|--------------------|---------------------|
| M1         | NC                 | NC                  |
| M2         | IO200NDB3          | IO111PDB6V1         |
| M3         | IO206NDB3          | IO115NDB6V1         |
| M4         | GFA2/IO206PDB3     | IO113NPB6V1         |
| M5         | GFA1/IO207PDB3     | IO109PPB6V0         |
| M6         | V <sub>CCPLF</sub> | IO108PDB6V0         |
| M7         | IO205NDB3          | IO108NDB6V0         |
| M8         | GFB2/IO205PDB3     | V <sub>CCI</sub> B6 |
| M9         | V <sub>CC</sub>    | GND                 |
| M10        | GND                | V <sub>CC</sub>     |
| M11        | GND                | V <sub>CC</sub>     |
| M12        | GND                | V <sub>CC</sub>     |
| M13        | GND                | V <sub>CC</sub>     |
| M14        | V <sub>CC</sub>    | GND                 |
| M15        | GCB2/IO95PPB1      | V <sub>CCI</sub> B3 |
| M16        | GCA1/IO93PPB1      | GDB0/IO66NPB3V1     |
| M17        | GCC2/IO96PPB1      | IO60NDB3V1          |
| M18        | IO100PPB1          | IO60PDB3V1          |
| M19        | GCA2/IO94PPB1      | IO61PDB3V1          |
| M20        | IO101PPB1          | NC                  |
| M21        | IO99PPB1           | IO59PDB3V0          |
| M22        | NC                 | IO58NDB3V0          |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function  |
|------------|---------------------|---------------------|
| N1         | IO201NDB3           | NC                  |
| N2         | IO201PDB3           | IO110PDB6V0         |
| N3         | NC                  | V <sub>CC</sub>     |
| N4         | GFC2/IO204PDB3      | IO109NPB6V0         |
| N5         | IO204NDB3           | IO106NDB6V0         |
| N6         | IO203NDB3           | IO106PDB6V0         |
| N7         | IO203PDB3           | GEC0/IO104NPB6V0    |
| N8         | V <sub>CCI</sub> B3 | VMV5                |
| N9         | V <sub>CC</sub>     | V <sub>CCI</sub> B5 |
| N10        | GND                 | V <sub>CCI</sub> B5 |
| N11        | GND                 | IO84NDB5V0          |
| N12        | GND                 | IO84PDB5V0          |
| N13        | GND                 | V <sub>CCI</sub> B4 |
| N14        | V <sub>CC</sub>     | V <sub>CCI</sub> B4 |
| N15        | V <sub>CCI</sub> B1 | VMV3                |
| N16        | IO95NPB1            | V <sub>CC</sub> PLD |
| N17        | IO100NPB1           | GDB1/IO66PPB3V1     |
| N18        | IO102NDB1           | GDC1/IO65PDB3V1     |
| N19        | IO102PDB1           | IO61NDB3V1          |
| N20        | NC                  | V <sub>CC</sub>     |
| N21        | IO101NPB1           | IO59NDB3V0          |
| N22        | IO103PDB1           | IO62PDB3V1          |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function               | M7A3PE600 Function  |
|------------|----------------------------------|---------------------|
| P1         | NC                               | NC                  |
| P2         | IO199PDB3                        | IO110NDB6V0         |
| P3         | IO199NDB3                        | NC                  |
| P4         | IO202NDB3                        | IO105PDB6V0         |
| P5         | IO202PDB3                        | IO105NDB6V0         |
| P6         | IO196PPB3                        | GEC1/IO104PPB6V0    |
| P7         | IO193PPB3                        | V <sub>COMPLE</sub> |
| P8         | V <sub>CC</sub> I <sub>B</sub> 3 | GNDQ                |
| P9         | GND                              | GEA2/IO101PPB5V2    |
| P10        | V <sub>CC</sub>                  | IO92NDB5V1          |
| P11        | V <sub>CC</sub>                  | IO90NDB5V1          |
| P12        | V <sub>CC</sub>                  | IO82NDB5V0          |
| P13        | V <sub>CC</sub>                  | IO74NDB4V1          |
| P14        | GND                              | IO74PDB4V1          |
| P15        | V <sub>CC</sub> I <sub>B</sub> 1 | GNDQ                |
| P16        | GDB0/IO112NPB1                   | V <sub>COMPLD</sub> |
| P17        | IO106NDB1                        | V <sub>JTAG</sub>   |
| P18        | IO106PDB1                        | GDC0/IO65NDB3V1     |
| P19        | IO107PDB1                        | GDA1/IO67PDB3V1     |
| P20        | NC                               | NC                  |
| P21        | IO104PDB1                        | IO64PDB3V1          |
| P22        | IO103NDB1                        | IO62NDB3V1          |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function |
|------------|---------------------|--------------------|
| R1         | NC                  | NC                 |
| R2         | IO197PPB3           | IO107PDB6V0        |
| R3         | V <sub>CC</sub>     | IO107NDB6V0        |
| R4         | IO197NPB3           | GEB1/IO103PDB6V0   |
| R5         | IO196NPB3           | GEB0/IO103NDB6V0   |
| R6         | IO193NPB3           | VMV6               |
| R7         | GEC0/IO190NPB3      | V <sub>CCPLE</sub> |
| R8         | VMV3                | IO101NPB5V2        |
| R9         | V <sub>CCI</sub> B2 | IO95PPB5V1         |
| R10        | V <sub>CCI</sub> B2 | IO92PDB5V1         |
| R11        | IO147RSB2           | IO90PDB5V1         |
| R12        | IO136RSB2           | IO82PDB5V0         |
| R13        | V <sub>CCI</sub> B2 | IO76NDB4V1         |
| R14        | V <sub>CCI</sub> B2 | IO76PDB4V1         |
| R15        | VMV2                | VMV4               |
| R16        | IO110NDB1           | TCK                |
| R17        | GDB1/IO112PPB1      | V <sub>PUMP</sub>  |
| R18        | GDC1/IO111PDB1      | TRST               |
| R19        | IO107NDB1           | GDA0/IO67NDB3V1    |
| R20        | V <sub>CC</sub>     | NC                 |
| R21        | IO104NDB1           | IO64NDB3V1         |
| R22        | IO105PDB1           | IO63PDB3V1         |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function |
|------------|--------------------|--------------------|
| T1         | IO198PDB3          | NC                 |
| T2         | IO198NDB3          | NC                 |
| T3         | NC                 | GND                |
| T4         | IO194PPB3          | GEA1/IO102PDB6V0   |
| T5         | IO192PPB3          | GEA0/IO102NDB6V0   |
| T6         | GEC1/IO190PPB3     | GNDQ               |
| T7         | IO192NPB3          | GEC2/IO99PDB5V2    |
| T8         | GNDQ               | IO95NPB5V1         |
| T9         | GEA2/IO187RSB2     | IO91NDB5V1         |
| T10        | IO161RSB2          | IO91PDB5V1         |
| T11        | IO155RSB2          | IO83NDB5V0         |
| T12        | IO141RSB2          | IO83PDB5V0         |
| T13        | IO129RSB2          | IO77NDB4V1         |
| T14        | IO124RSB2          | IO77PDB4V1         |
| T15        | GNDQ               | IO69NDB4V0         |
| T16        | IO110PDB1          | GDB2/IO69PDB4V0    |
| T17        | V <sub>JTAG</sub>  | TDI                |
| T18        | GDC0/IO111NDB1     | GNDQ               |
| T19        | GDA1/IO113PDB1     | TDO                |
| T20        | NC                 | GND                |
| T21        | IO108PDB1          | NC                 |
| T22        | IO105NDB1          | IO63NDB3V1         |



Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function |
|------------|--------------------|--------------------|
| U1         | IO195PDB3          | NC                 |
| U2         | IO195NDB3          | NC                 |
| U3         | IO194NPB3          | NC                 |
| U4         | GEB1/IO189PDB3     | GND                |
| U5         | GEB0/IO189NDB3     | IO100NDB5V2        |
| U6         | VMV2               | GEB2/IO100PDB5V2   |
| U7         | IO179RSB2          | IO99NDB5V2         |
| U8         | IO171RSB2          | IO88NDB5V0         |
| U9         | IO165RSB2          | IO88PDB5V0         |
| U10        | IO159RSB2          | IO89NDB5V0         |
| U11        | IO151RSB2          | IO80NDB4V1         |
| U12        | IO137RSB2          | IO81NDB4V1         |
| U13        | IO134RSB2          | IO81PDB4V1         |
| U14        | IO128RSB2          | IO70NDB4V0         |
| U15        | VMV1               | GDC2/IO70PDB4V0    |
| U16        | TCK                | IO68NDB4V0         |
| U17        | V <sub>PUMP</sub>  | GDA2/IO68PDB4V0    |
| U18        | TRST               | TMS                |
| U19        | GDA0/IO113NDB1     | GND                |
| U20        | NC                 | NC                 |
| U21        | IO108NDB1          | NC                 |
| U22        | IO109PDB1          | NC                 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function  |
|------------|--------------------|---------------------|
| V1         | NC                 | V <sub>CCI</sub> B6 |
| V2         | NC                 | NC                  |
| V3         | GND                | NC                  |
| V4         | GEA1/IO188PDB3     | IO98NDB5V2          |
| V5         | GEA0/IO188NDB3     | GND                 |
| V6         | IO184RSB2          | IO94NDB5V1          |
| V7         | GEC2/IO185RSB2     | IO94PDB5V1          |
| V8         | IO168RSB2          | V <sub>CC</sub>     |
| V9         | IO163RSB2          | V <sub>CC</sub>     |
| V10        | IO157RSB2          | IO89PDB5V0          |
| V11        | IO149RSB2          | IO80PDB4V1          |
| V12        | IO143RSB2          | IO78NPB4V1          |
| V13        | IO138RSB2          | NC                  |
| V14        | IO131RSB2          | V <sub>CC</sub>     |
| V15        | IO125RSB2          | V <sub>CC</sub>     |
| V16        | GDB2/IO115RSB2     | NC                  |
| V17        | TDI                | NC                  |
| V18        | GNDQ               | GND                 |
| V19        | TDO                | NC                  |
| V20        | GND                | NC                  |
| V21        | NC                 | NC                  |
| V22        | IO109NDB1          | V <sub>CCI</sub> B3 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function | M7A3PE600 Function  |
|------------|--------------------|---------------------|
| W1         | NC                 | GND                 |
| W2         | IO191PDB3          | V <sub>CC1</sub> B6 |
| W3         | NC                 | NC                  |
| W4         | GND                | IO98PDB5V2          |
| W5         | IO183RSB2          | IO96NDB5V2          |
| W6         | GEB2/IO186RSB2     | IO96PDB5V2          |
| W7         | IO172RSB2          | IO86NDB5V0          |
| W8         | IO170RSB2          | IO86PDB5V0          |
| W9         | IO164RSB2          | IO85PDB5V0          |
| W10        | IO158RSB2          | IO85NDB5V0          |
| W11        | IO153RSB2          | IO78PPB4V1          |
| W12        | IO142RSB2          | IO79NDB4V1          |
| W13        | IO135RSB2          | IO79PDB4V1          |
| W14        | IO130RSB2          | NC                  |
| W15        | GDC2/IO116RSB2     | NC                  |
| W16        | IO120RSB2          | IO71NDB4V0          |
| W17        | GDA2/IO114RSB2     | IO71PDB4V0          |
| W18        | TMS                | NC                  |
| W19        | GND                | NC                  |
| W20        | NC                 | NC                  |
| W21        | NC                 | V <sub>CC1</sub> B3 |
| W22        | NC                 | GND                 |

Table A-1. M7A3PE600 and M7A3P1000 FG484 Package Connections (Continued)

| Pin Number | M7A3P1000 Function  | M7A3PE600 Function  |
|------------|---------------------|---------------------|
| Y1         | V <sub>CCI</sub> B3 | GND                 |
| Y2         | IO191NDB3           | GND                 |
| Y3         | NC                  | V <sub>CCI</sub> B5 |
| Y4         | IO182RSB2           | IO97NDB5V2          |
| Y5         | GND                 | IO97PDB5V2          |
| Y6         | IO177RSB2           | IO93NDB5V1          |
| Y7         | IO174RSB2           | IO93PDB5V1          |
| Y8         | V <sub>CC</sub>     | IO87NDB5V0          |
| Y9         | V <sub>CC</sub>     | IO87PDB5V0          |
| Y10        | IO154RSB2           | NC                  |
| Y11        | IO148RSB2           | NC                  |
| Y12        | IO140RSB2           | IO75NDB4V1          |
| Y13        | NC                  | IO75PDB4V1          |
| Y14        | V <sub>CC</sub>     | IO72NDB4V0          |
| Y15        | V <sub>CC</sub>     | IO72PDB4V0          |
| Y16        | NC                  | IO73NDB4V0          |
| Y17        | NC                  | IO73PDB4V0          |
| Y18        | GND                 | NC                  |
| Y19        | NC                  | NC                  |
| Y20        | NC                  | V <sub>CCI</sub> B4 |
| Y21        | NC                  | GND                 |
| Y22        | V <sub>CCI</sub> B1 | GND                 |

---

## Board Schematics

This appendix provides illustrations of the CoreMP7 Evaluation Board.

### Top-Level View

Figure B-1 on page 130 illustrates a top-level view of the CoreMP7 Evaluation Board. Figure B-2 on page 131 shows a bottom-level view of CoreMP7 Evaluation Board.

### CoreMP7 Schematics

The rest of this appendix shows the following illustrations of the CoreMP7 Evaluation Board:

Figure B-3 on page 132: Main 1.5 V, 2.5 V, and 3.3 V Power Supplies

Figure B-4 on page 133: Flash and Synchronous SRAM Memories

Figure B-5 on page 134: M7A3PE600 FPGA – I/O Banks 0–2

Figure B-6 on page 135: M7A3PE600 FPGA – I/O Banks 3–5

Figure B-7 on page 136: M7A3PE600 FPGA – I/O Banks 6–7

Figure B-8 on page 137: LEDs and Push-Button Switches

Figure B-9 on page 138: FPGA I/O Expansion Headers

Figure B-10 on page 139: USB Interface

Figure B-11 on page 140: RS-232 and CAN Interfaces

Figure B-12 on page 141: Ethernet 0 Interface

Figure B-13 on page 142: Ethernet 1 Interface



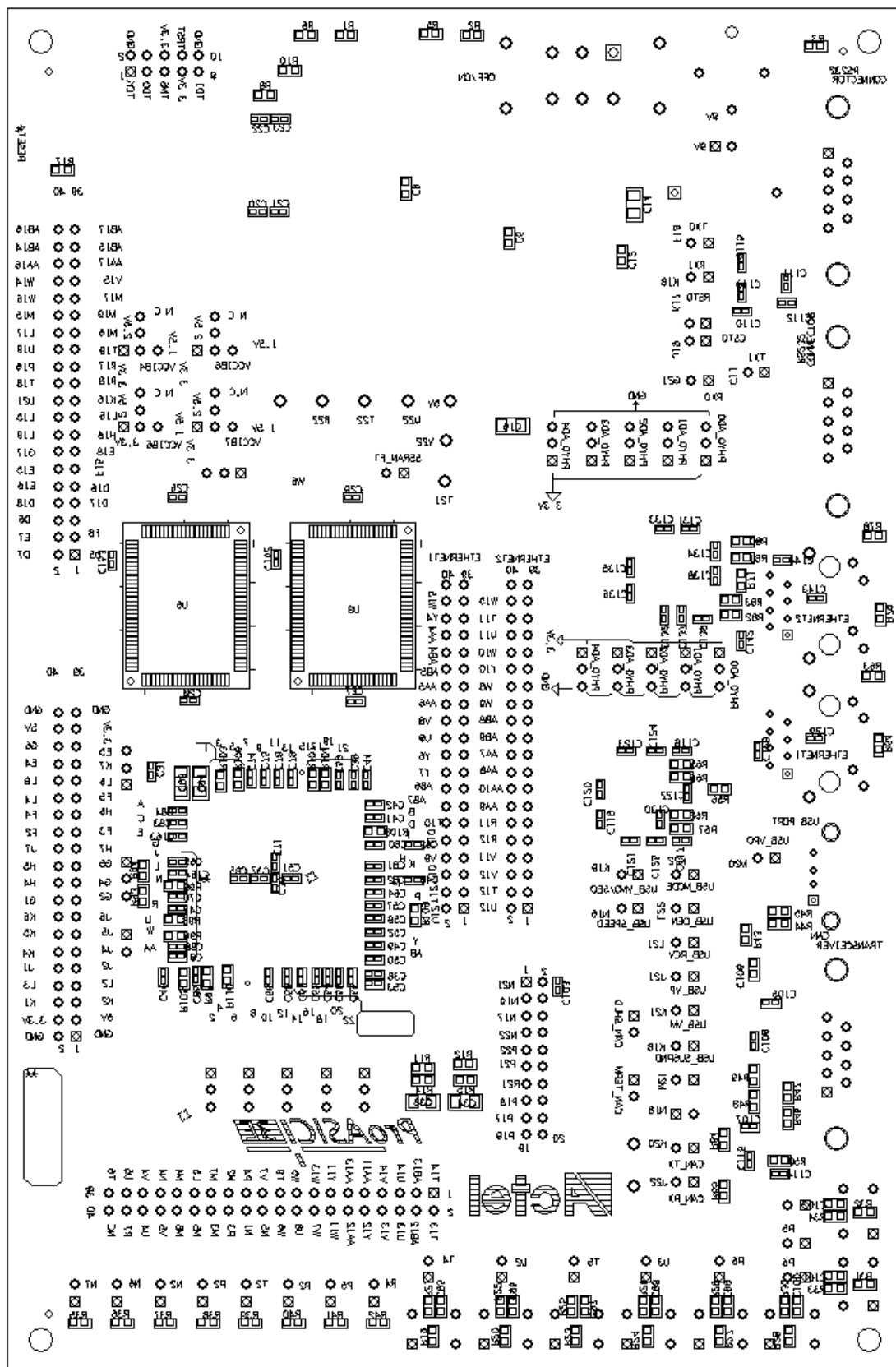
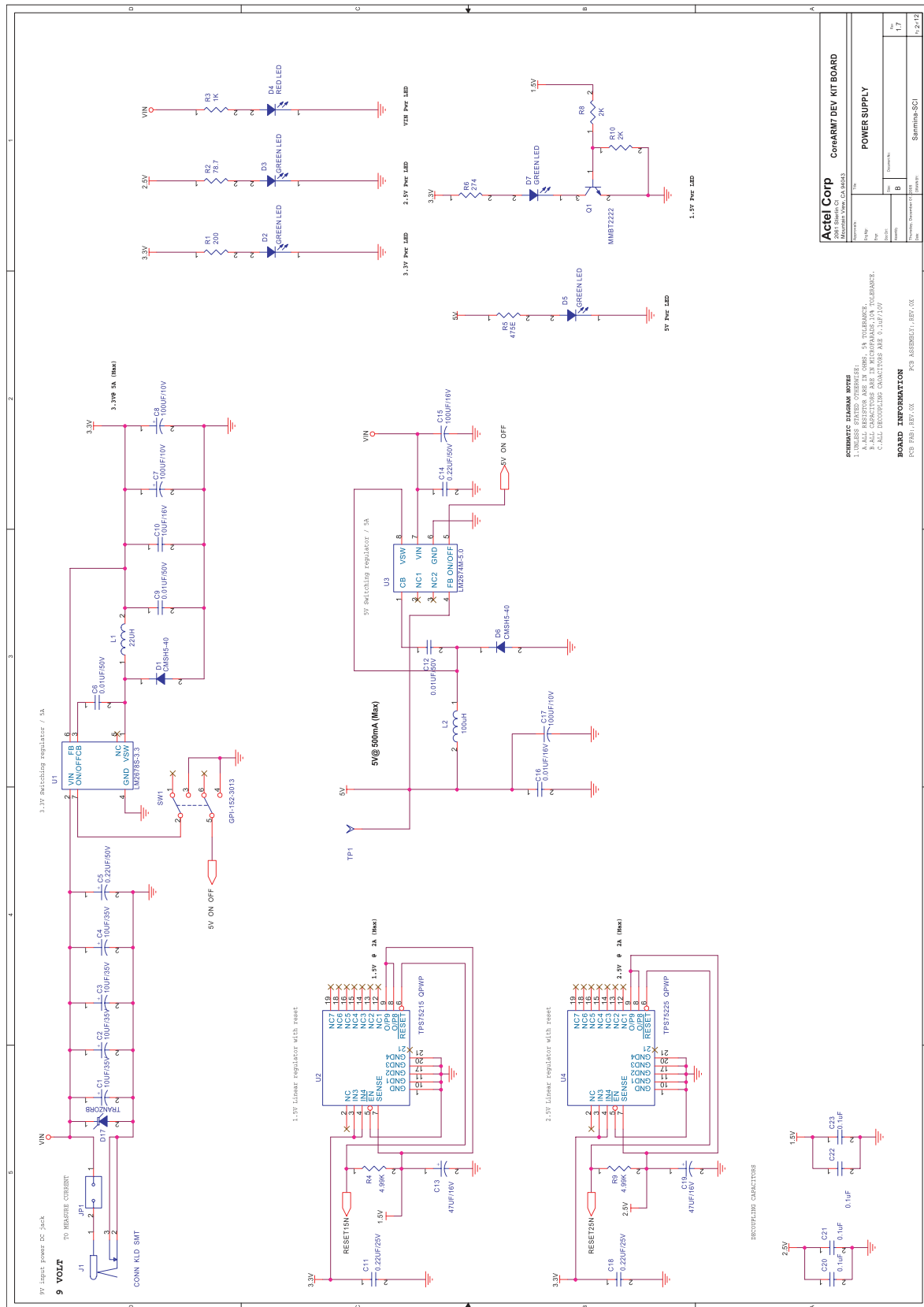


Figure B-2. Bottom-Level View of CoreMP7 Development Board



### Figure B-3. Main 1.5 V, 2.5 V, and 3.3 V Power Supplies



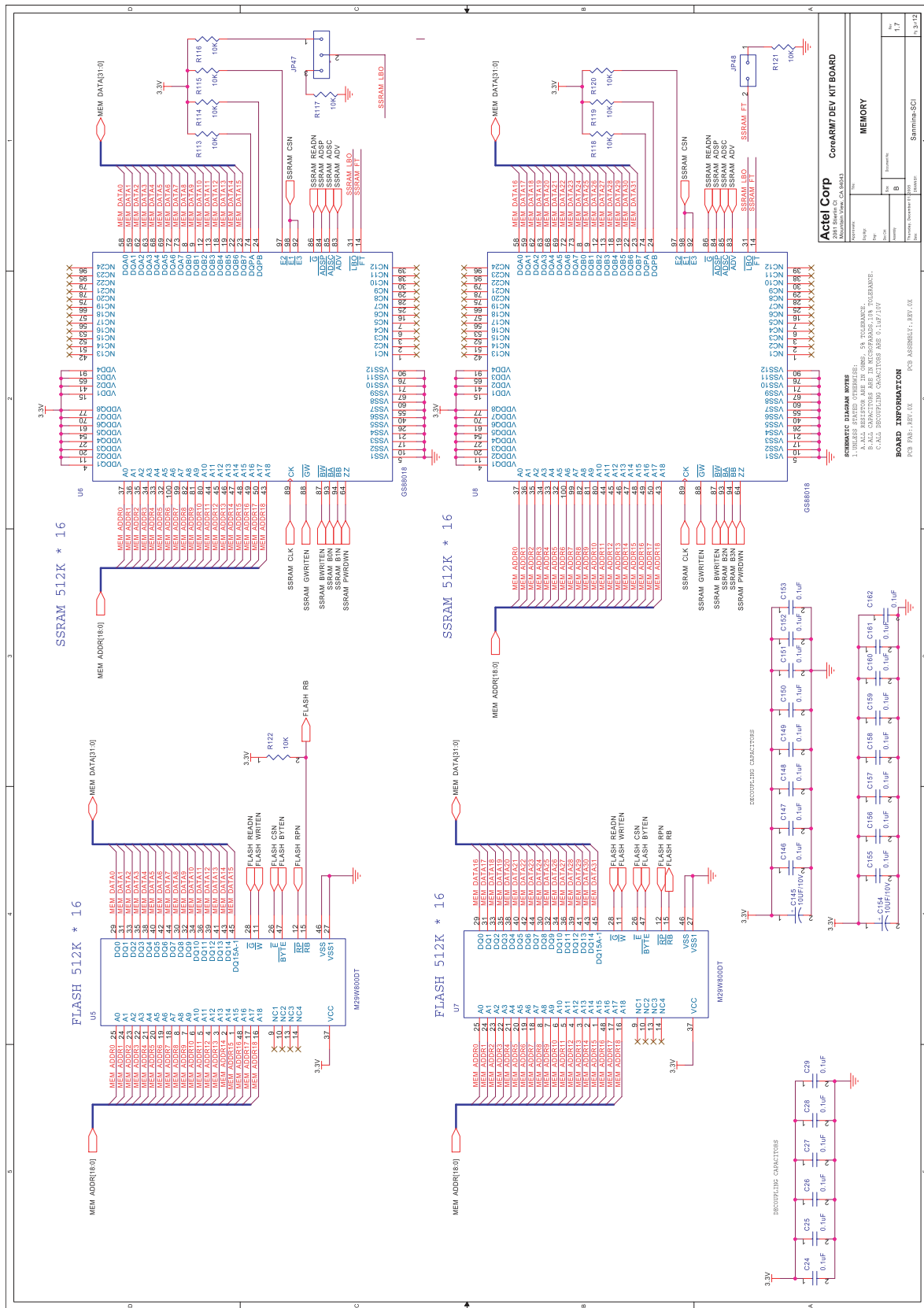


Figure B-4. Flash and Synchronous SRAM Memories

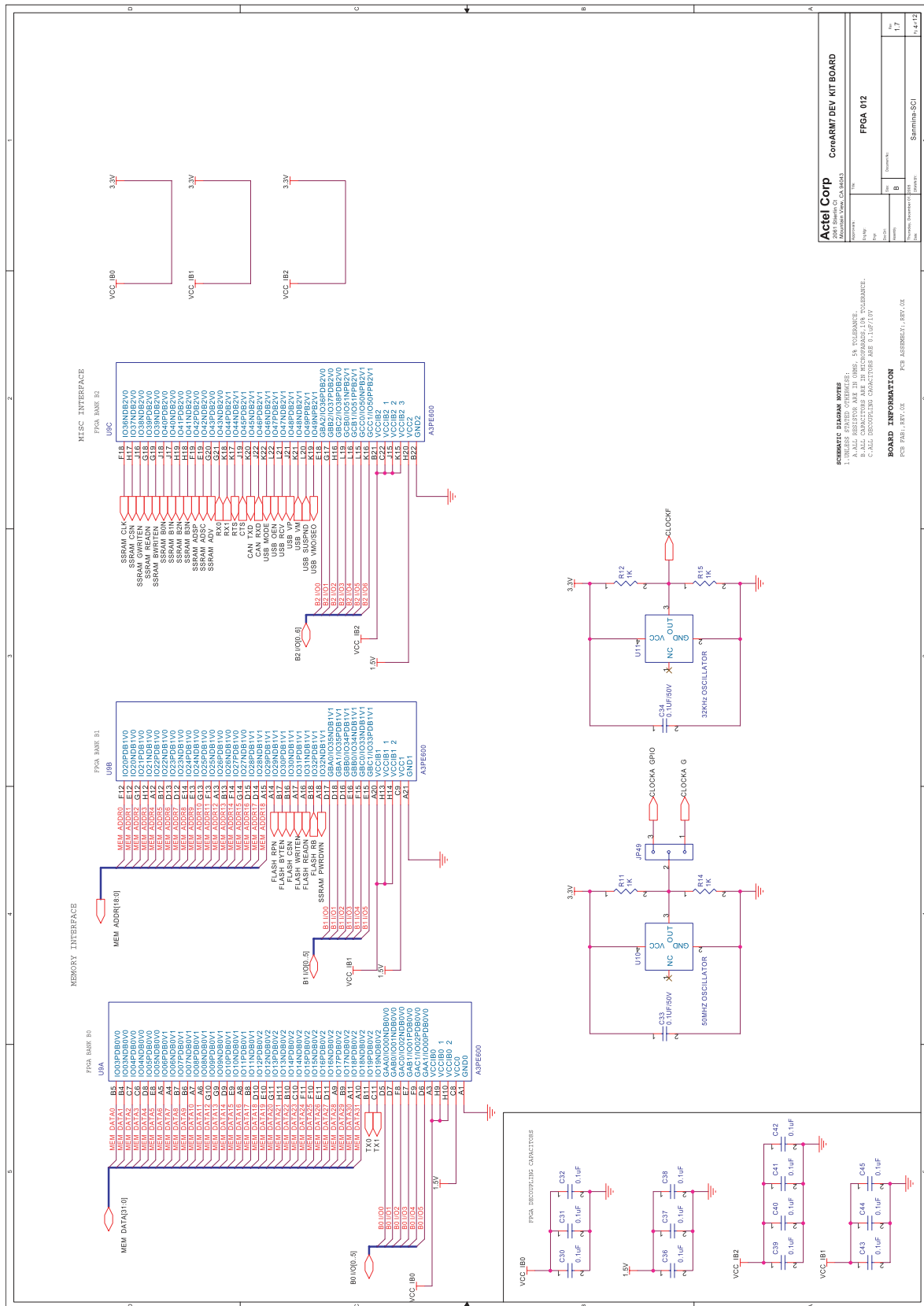


Figure B-5. M7A3PE600 FPGA – I/O Banks 0–2

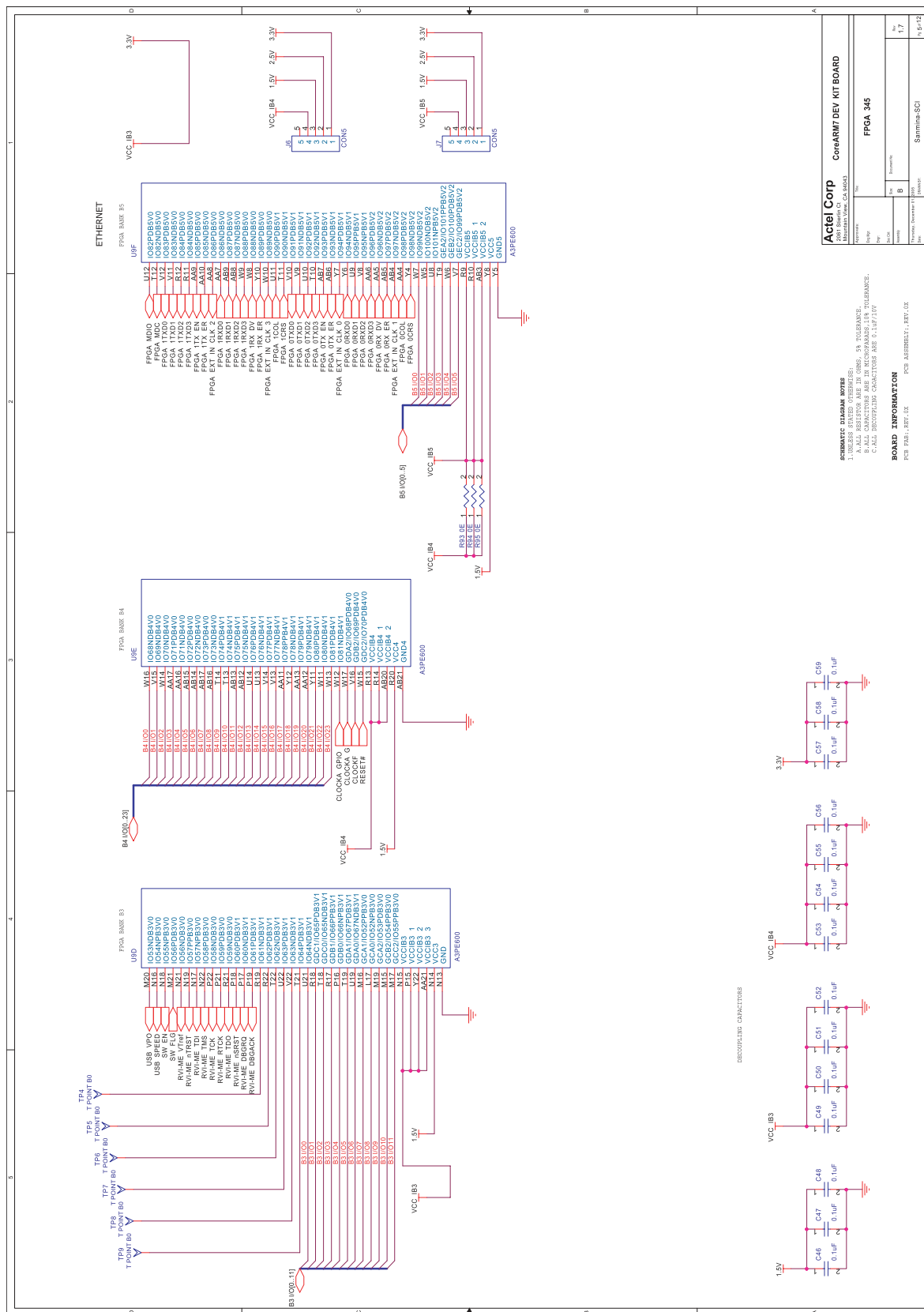


Figure B-6. M7A3PE600 FPGA – I/O Banks 3–5



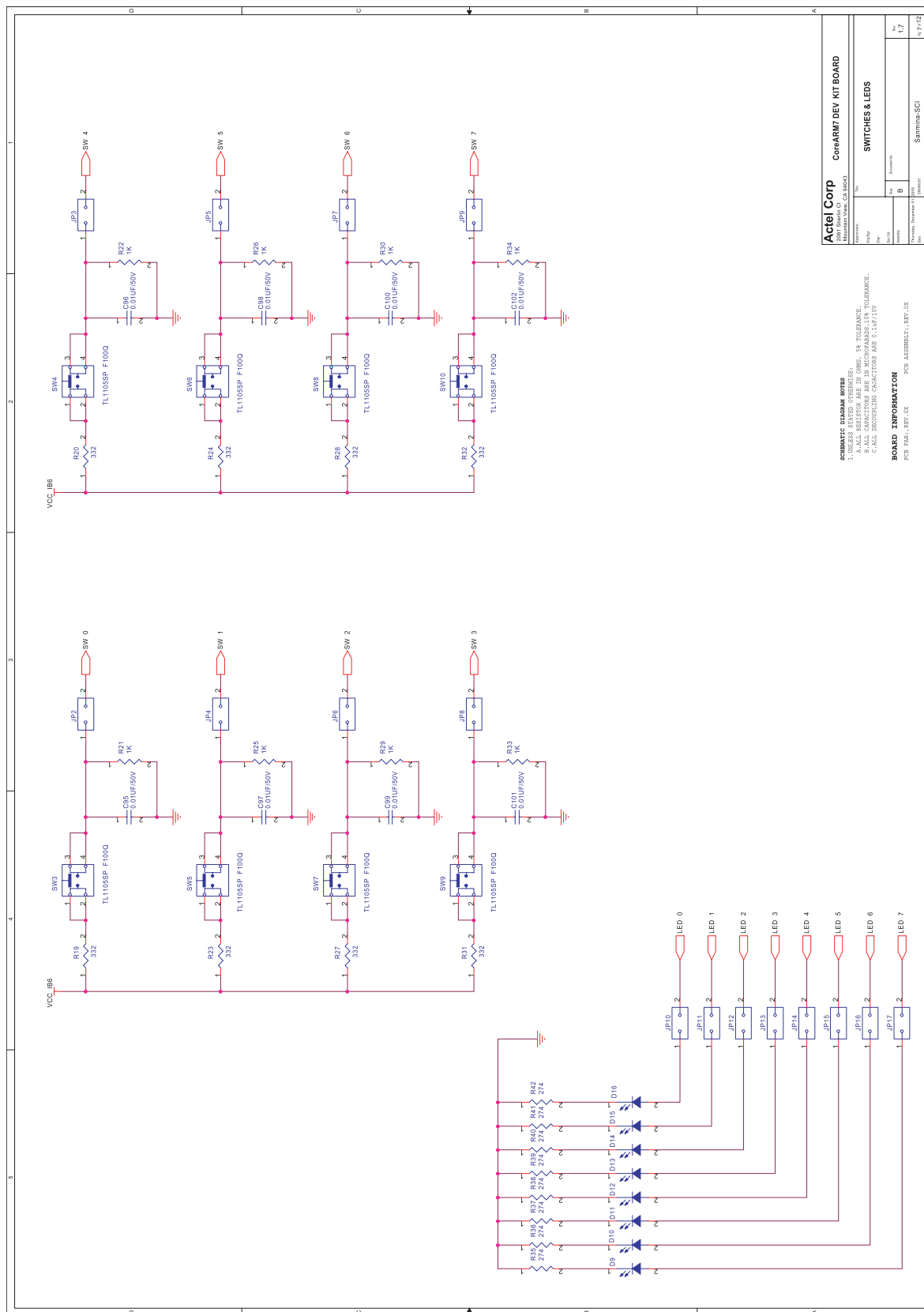


Figure B-8. LEDs and Push-Button Switches

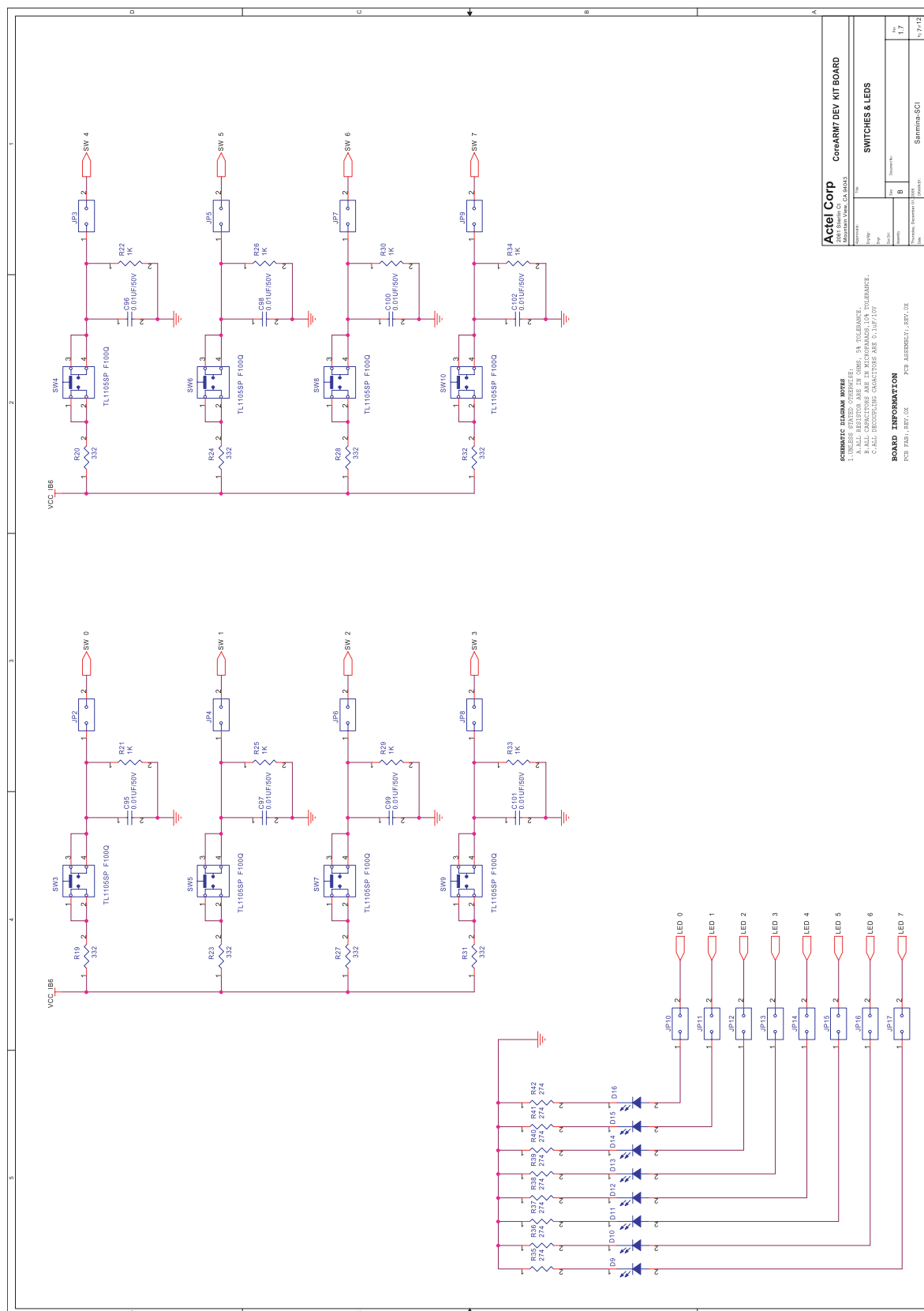
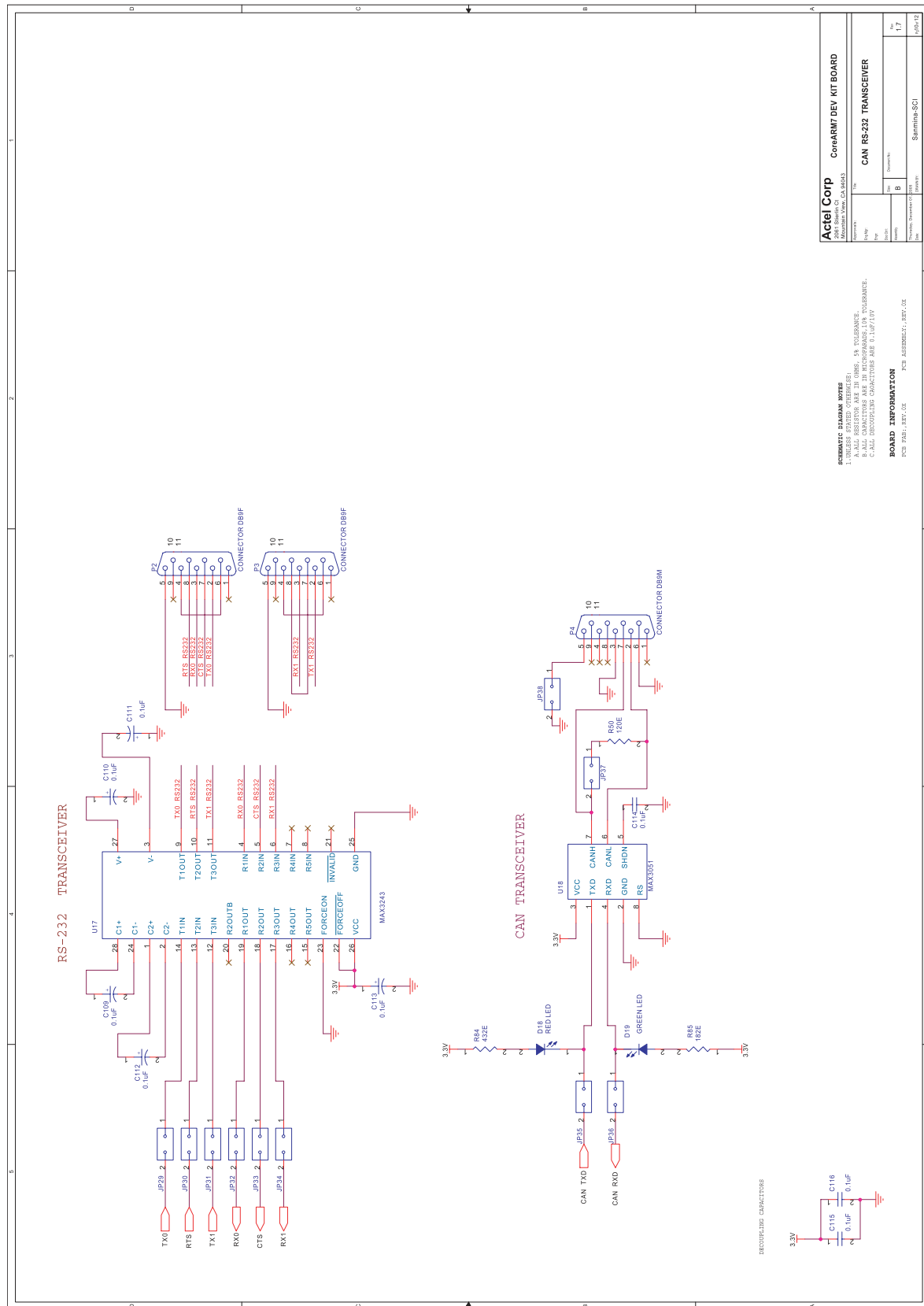


Figure B-9. FPGA I/O Expansion Headers





### Figure B-11. RS-232 and CAN Interfaces



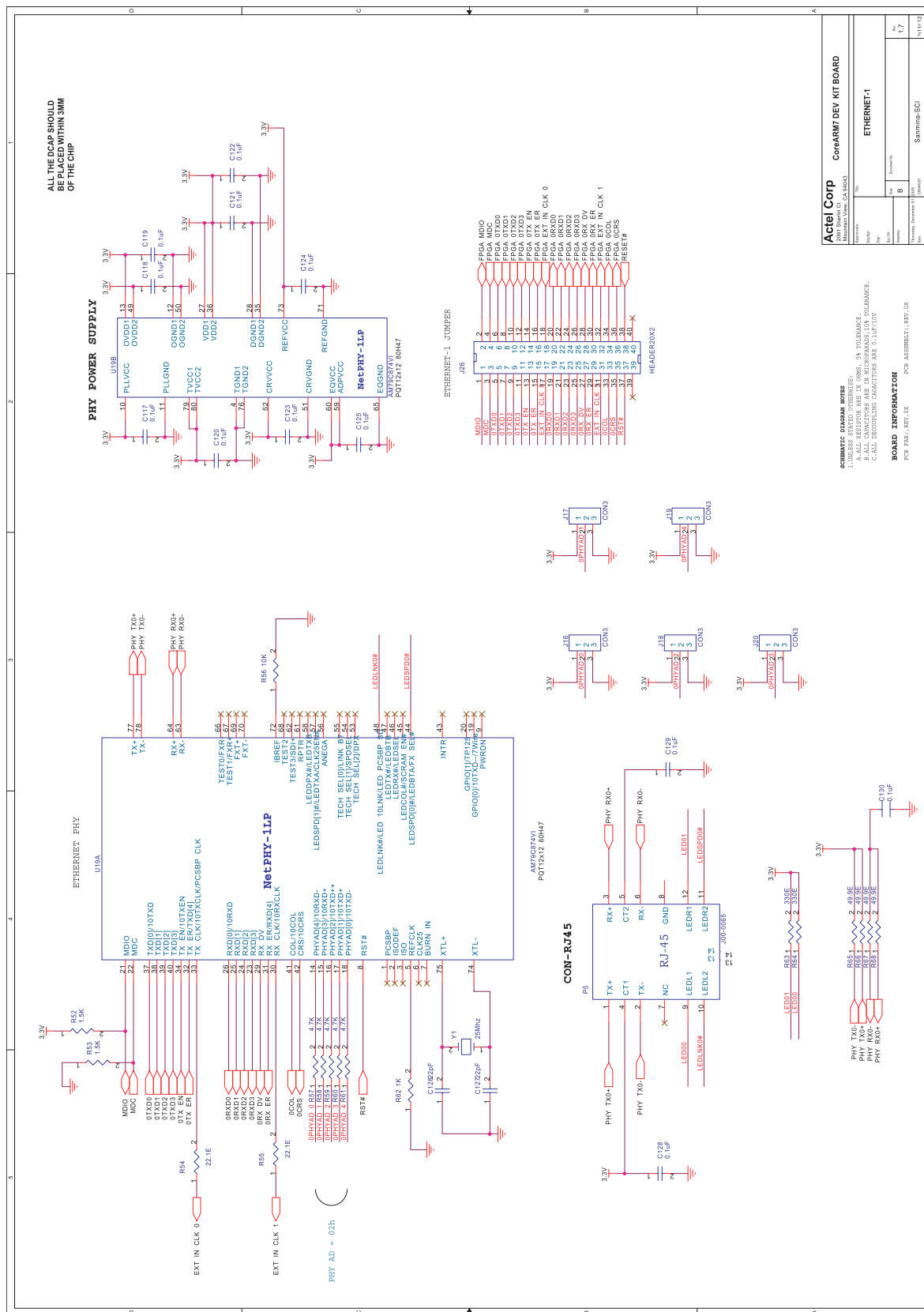
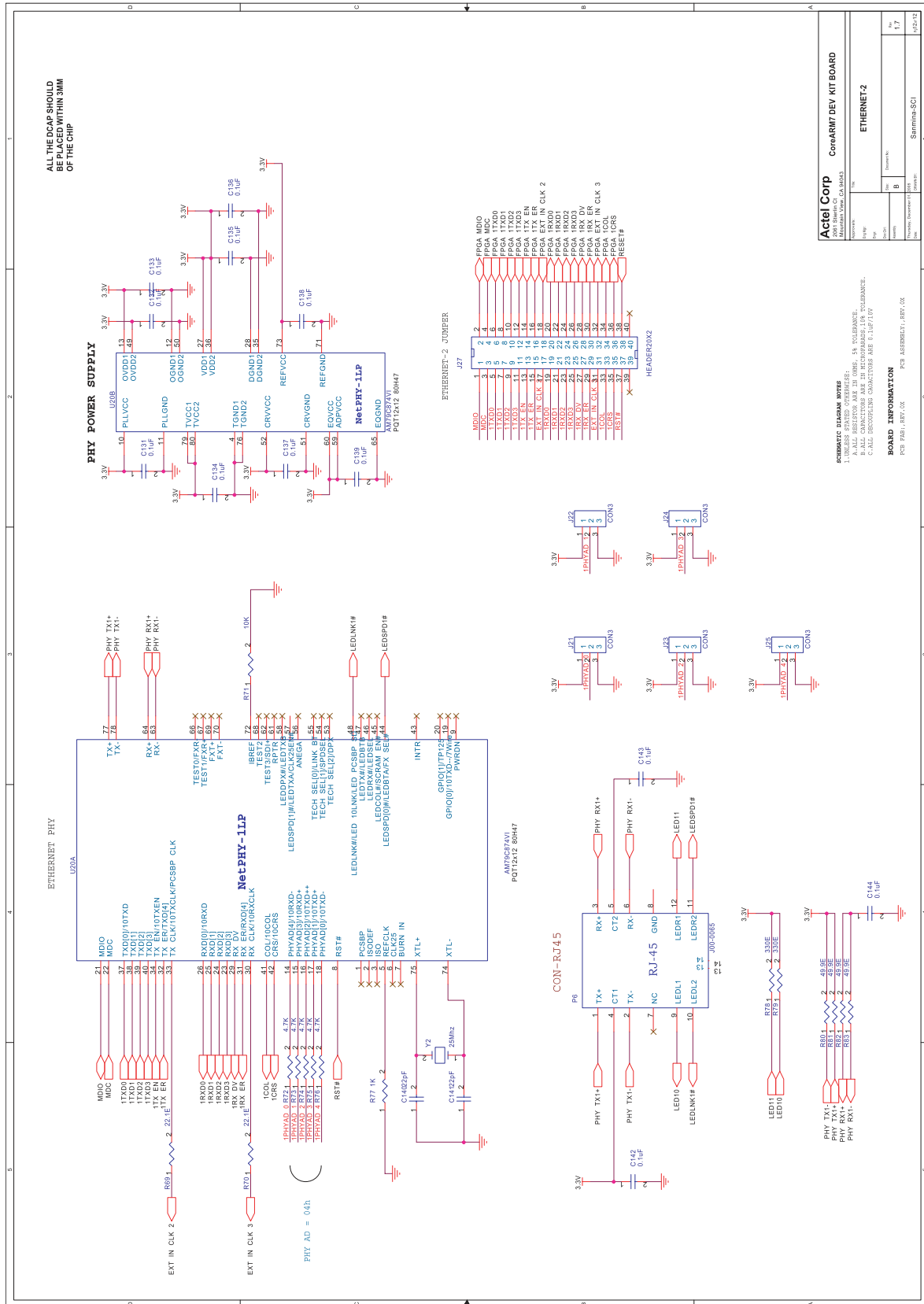


Figure B-12. Ethernet 0 Interface



---

## Signal Layers

The CoreMP7 Evaluation Board is a six-layer board. The board has the following copper layers:

[Figure C-1 on page 144](#): Layer 1 – Top Signal Layer

[Figure C-2 on page 145](#): Layer 2 – Ground Plane

[Figure C-3 on page 146](#): Layer 3 – Signal Layer 3

[Figure C-4 on page 147](#): Layer 4 – Signal Layer 4

[Figure C-5 on page 148](#): Layer 5 – Power Plane

[Figure C-6 on page 149](#): Layer 6 – Bottom Signal Layer

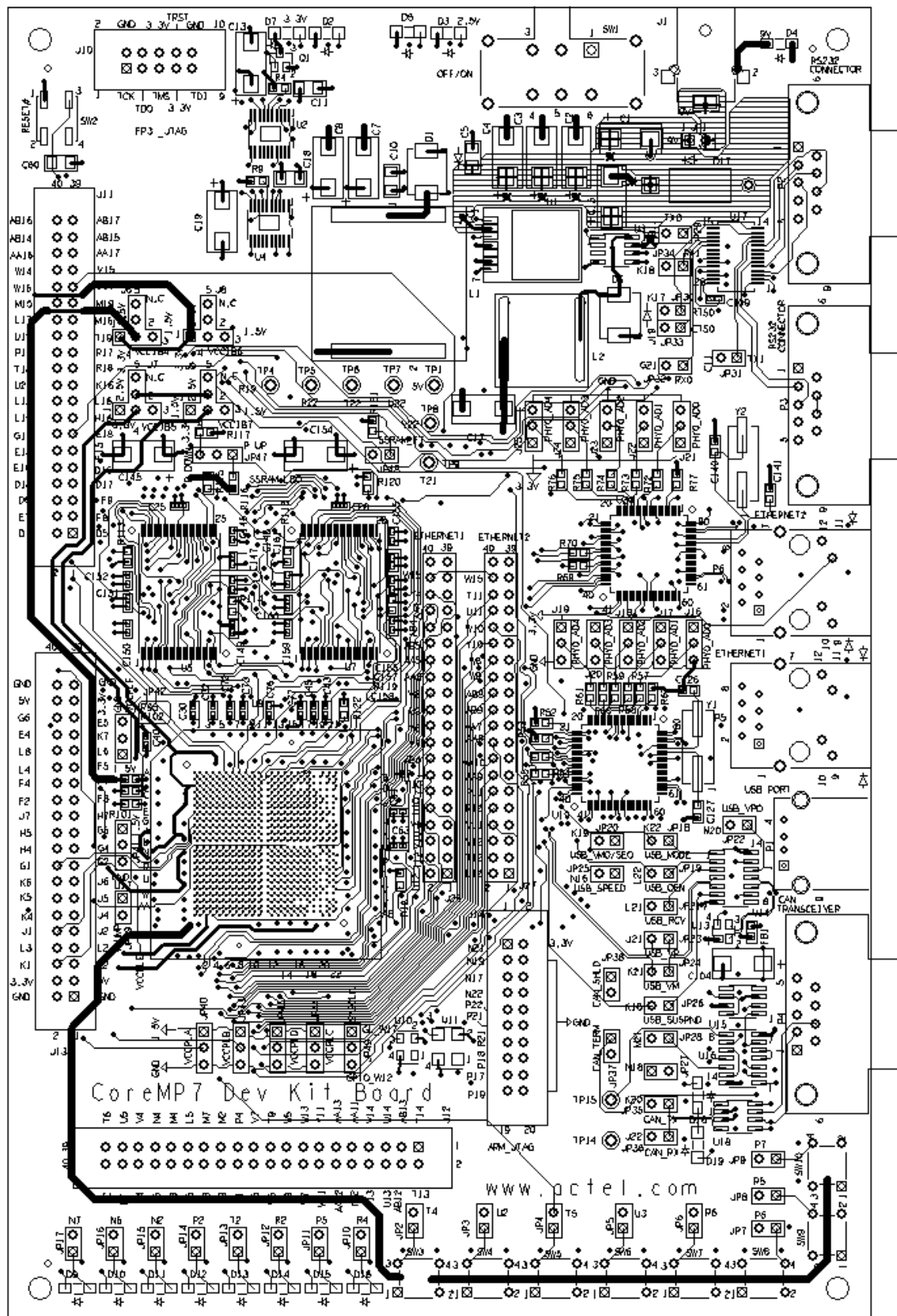


Figure C-2. Layer 2 – Ground Plane (Blank)

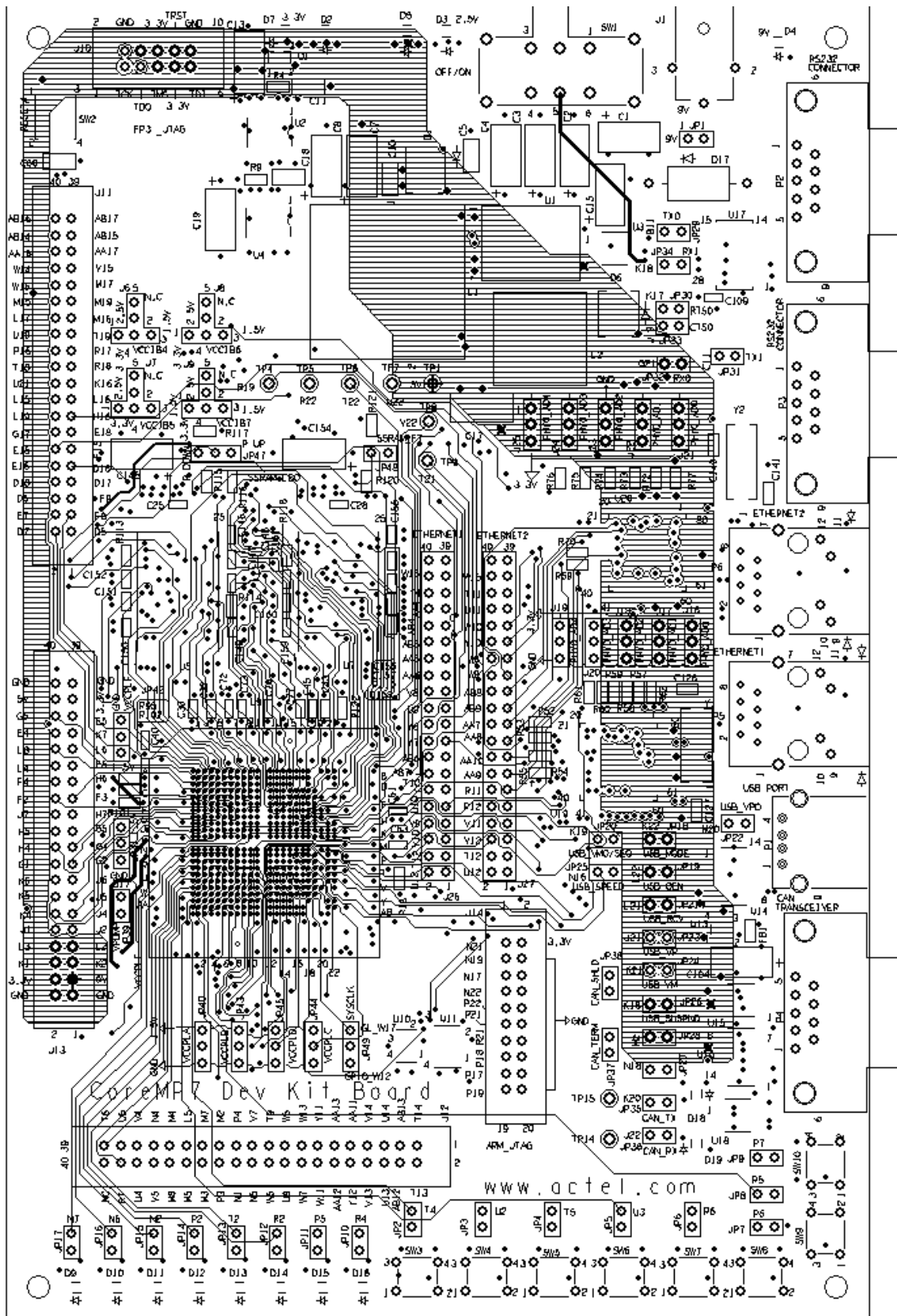


Figure C-3. Layer 3 – Signal Layer 3



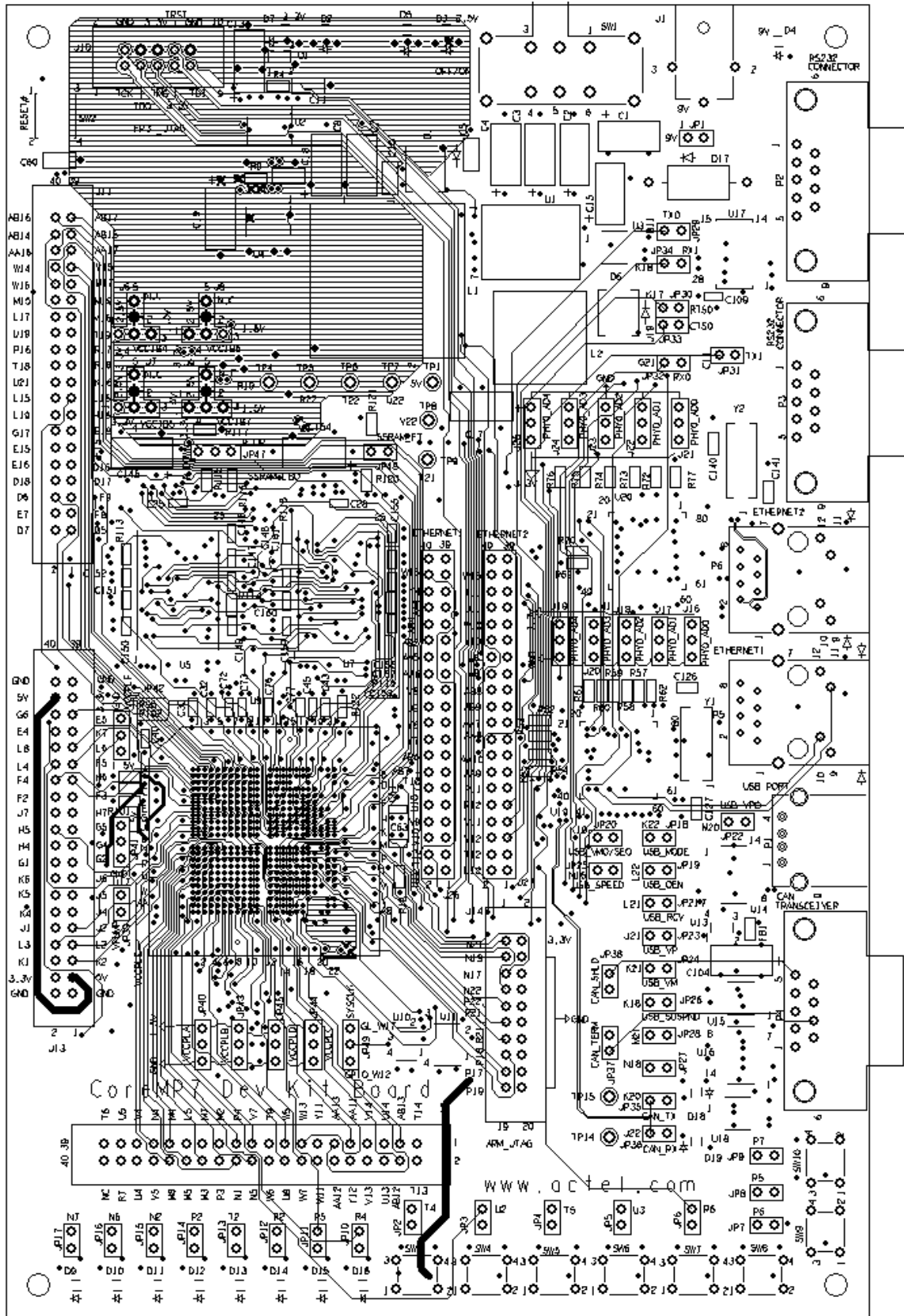
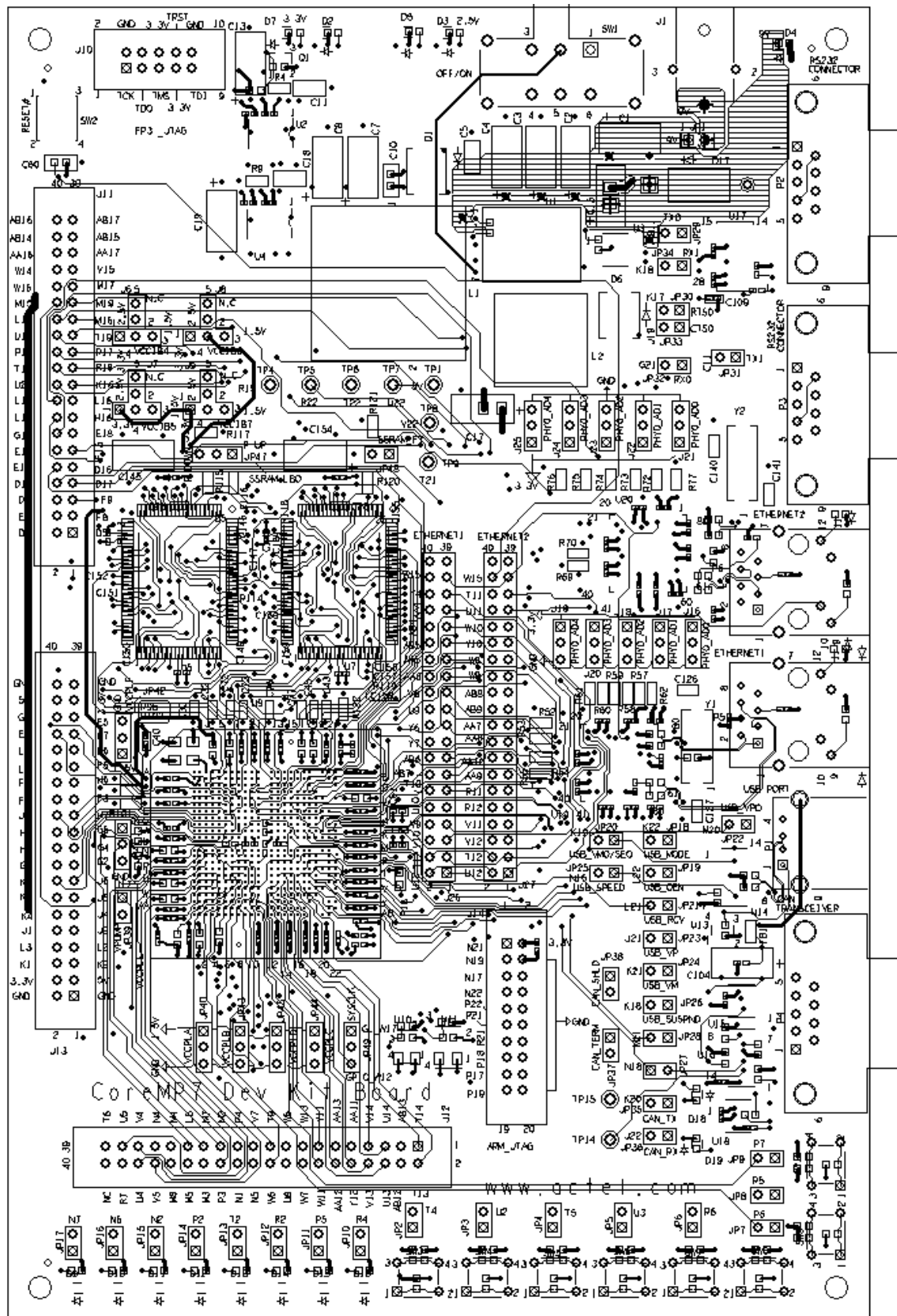


Figure C-4. Layer 4 – Signal Layer 4

Figure C-5. Layer 5 – Power Plane (Blank)







---

## Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

### Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**

From Southeast and Southwest U.S.A., call **650.318.4480**

From South Central U.S.A., call **650.318.4434**

From Northwest U.S.A., call **650.318.4434**

From Canada, call **650.318.4480**

From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**

From Japan, call **650.318.4743**

From the rest of the world, call **650.318.4743**

Fax, from anywhere in the world **650.318.8044**

### Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

### Actel Technical Support

Visit the [Actel Customer Support website \(www.actel.com/custsup/search.html\)](http://www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

### Website

You can browse a variety of technical and non-technical information on Actel's [home page](http://www.actel.com), at [www.actel.com](http://www.actel.com).

## Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

### Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is [tech@actel.com](mailto:tech@actel.com).

### Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

**650.318.4460**

**800.262.1060**

Customers needing assistance outside the US time zones can either contact technical support via email ([tech@actel.com](mailto:tech@actel.com)) or contact a local sales office. [Sales office listings](#) can be found at [www.actel.com/contact/offices/index.html](http://www.actel.com/contact/offices/index.html).

---

# Index

10/100 Ethernet 19

## A

Actel

- electronic mail 152

- telephone 152

- web-based technical support 151

- website 151

assumptions 5

## B

board

- CAN 21

- clocks 21

- description 9

- Ethernet 19

- headers 22

- jumpers 15

- layers 22

- LEDs 16

- memory 22

- PLLs 11

- power supplies 12

  - block diagram 12

- programming 14

- RS-232 18

- schematics 129

- self test 25

  - programming 25

- switches 17

- test points 22

- testing 25

  - programming 25

- top-level view 10

- usage 9

- USB 20

## C

CAN 21

clocks 21

CompanionCore 21

contacting Actel

- customer service 151

- electronic mail 152

- telephone 152

- web-based technical support 151

Core10/100 19

CoreMP7 evaluation board 9

CoreUART 18

customer service 151

## D

design flow 27

- design creation 29

- implementation 30

- microprocessor 31

- programming 31

- system creation 27

- verification 29

development kit contents 7

## E

Ethernet 19

example design 14

## F

FPGA package connections 105

## H

hardware 9

- description 9

- installation 25

headers 22

## **J**

jumpers 15

## **K**

kit contents 7

## **L**

LEDs 16

## **M**

memory 22

## **P**

PLLs 11

power supplies 12

    block diagram 12

product support 151–152

    customer service 151

    electronic mail 152

    technical support 151

    telephone 152

    website 151

programming 14

## **R**

RS-232 18

## **S**

schematics 129

self test 25

    programming 25

setup 25

software

    installation 25

switches 17

system requirements 7

## **T**

technical support 151

test points 22

testing 25

    programming 25

## **U**

USB 20

## **W**

web-based technical support 151



***For more information about Actel's products, visit our website at  
<http://www.actel.com>***

***Actel Corporation*** • 2061 Stierlin Court • Mountain View, CA 94043 USA  
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

***Actel Europe Ltd.*** • Dunlop House, Riverside Way • Camberley, Surrey GU15 3YL • United Kingdom  
Phone +44 (0) 1276 401 450 • Fax +44 (0) 1276 401 490

***Actel Japan*** • EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan  
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • [www.jp.actel.com](http://www.jp.actel.com)

***Actel Hong Kong*** • Suite 2114, Two Pacific Place • 88 Queensway, Admiralty Hong Kong  
Phone +852 2185 6460 • Fax +852 2185 6488 • [www.actel.com.cn](http://www.actel.com.cn)

XXXXXXXXX.X/7.06





## **Appendix 5:     Datasheet CoreMP7**

## Product Summary

- Personal Audio (MP3, WMA, and AAC Players)
- Personal Digital Assistants
- Wireless Handset
- Pagers
- Digital Still Camera
- Inkjet/Bubble-Jet Printer
- Monitors

## Key Features

- FPGA Optimized ARM7™ Family Processor
- Compatible with ARM7TDMI-S™
- 32/16-Bit RISC Architecture (ARMv4T)
- 32-Bit ARM® Instruction Set
- 16-Bit Thumb® Instruction Set
- 32-Bit Unified Bus Interface
- 3-Stage Pipeline
- 32-Bit ALU
- 32-Bit Memory Addressing Range
- Static Operation
- EmbeddedICE-RT™ Real-Time Debug Unit
- JTAG Interface Unit

## Benefits

- Fully Implemented in FPGA Fabric
- All Microprocessor I/Os Available to User
- Unified Bus Interface Simplifies SoC Design
- ARM and Thumb Instruction Sets Can Be Mixed

## ARM Supported Families

- ProASIC®3 (M7A3P)
- ProASIC3E (M7A3PE)
- Fusion (M7AFS)

## Synthesis and Simulation Support

- Directly Supported within the Actel Libero® Integrated Design Environment (IDE)
- Synthesis: Synplify® and Design Compiler®
- Simulation: Vital-Compliant VHDL Simulators and OVI-Compliant Verilog Simulators

## Verification and Compliance

- Compliant with ARMv4T ISA
- Compatible with ARM7TDMI-S

## Core Version

- This Datasheet Defines the Functionality for CoreMP7 v1.0.

## Contents

|                               |    |
|-------------------------------|----|
| Introduction .....            | 1  |
| Device Utilization .....      | 2  |
| General Description .....     | 3  |
| Programmer's Model .....      | 7  |
| AHB Wrapper .....             | 12 |
| CoreMP7 Variants .....        | 13 |
| Delivery and Deployment ..... | 14 |
| Bus Functional Mode .....     | 14 |
| AC Parameters .....           | 19 |
| Debug .....                   | 23 |
| Ordering Information .....    | 28 |
| List of Changes .....         | 28 |
| Datasheet Categories .....    | 28 |

## Introduction

The CoreMP7 soft IP core is an ARM7 family processor optimized for use in Actel ARM-ready FPGAs and is compatible with the ARM7TDMI-S. Users should refer to the [ARM7TDMI-S Technical Reference Manual](#) (DDI0234A-7TMI5-R4.pdf), published by the ARM Corporation, for detailed information on the ARM7. The ARM7 TRM is available for download from the ARM website at [www.arm.com](http://www.arm.com).

CoreMP7 is supplied with an Advanced Microcontroller Bus Architecture (AMBA) Advanced High-Performance Bus (AHB) compliant wrapper for inclusion in an AMBA-based processor system such as the one generated by the Actel CoreConsole IP Deployment Platform (IDP).

## ARM7 Family Processor

CoreMP7 is a general purpose, 32-bit, ARM7 family microprocessor that offers high performance and low power consumption. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles. The simplicity of RISC results in a high instruction throughput and fast real-time interrupt response from a small and cost-effective processor core. Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory. The CoreMP7 processor also implements the Thumb instruction set, which makes it ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue.

The 16-bit Thumb instruction set approaches twice the density of standard ARM code while retaining most of the ARM performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because Thumb code operates on the same 32-bit register set as ARM code. Thumb code is able to reduce

up to 65% of the code size compared to 32-bit ARM instructions, and offers 160% of the performance of an equivalent ARM processor connected to a 16-bit memory system.

## Device Utilization

The CoreMP7 has been implemented in M7 ProASIC3/E and M7 Fusion devices. A summary of the implementation data is listed in [Table 1](#).

### CoreMP7S

This variant of the CoreMP7 is optimized for maximum speed and minimum size and does not include debug.

### CoreMP7Sd

This variant of the CoreMP7 is optimized for minimum size and includes debug.

Table 1 • CoreMP7 Utilization and Performance

| Device Variant   | Performance (MHz) | Tiles | RAM Block | Utilization (%) |
|------------------|-------------------|-------|-----------|-----------------|
| <b>M7A3P1000</b> |                   |       |           |                 |
| CoreMP7S         | 28.548            | 6,397 | 4         | 26.0            |
| CoreMP7Sd        | 22.714            | 8,522 | 4         | 34.7            |
| <b>M7A3PE600</b> |                   |       |           |                 |
| CoreMP7S         | 29.699            | 6,324 | 4         | 45.7            |
| CoreMP7Sd        | 23.646            | 8,587 | 4         | 62.1            |
| <b>M7A3P250</b>  |                   |       |           |                 |
| CoreMP7S         | 23.904            | 6,104 | 4         | 99.3            |
| <b>M7AFS600</b>  |                   |       |           |                 |
| CoreMP7S         | 26.499            | 6,350 | 4         | 45.9            |
| CoreMP7Sd        | 23.165            | 8,243 | 4         | 59.6            |

## General Description

The CoreMP7 processor architecture, core, and functional diagrams are illustrated in the following figures:

- The CoreMP7 block diagram is shown in [Figure 1](#).
- The CoreMP7 core is shown in [Figure 2 on page 4](#).
- The CoreMP7 functional diagram is shown in [Figure 3 on page 5](#).

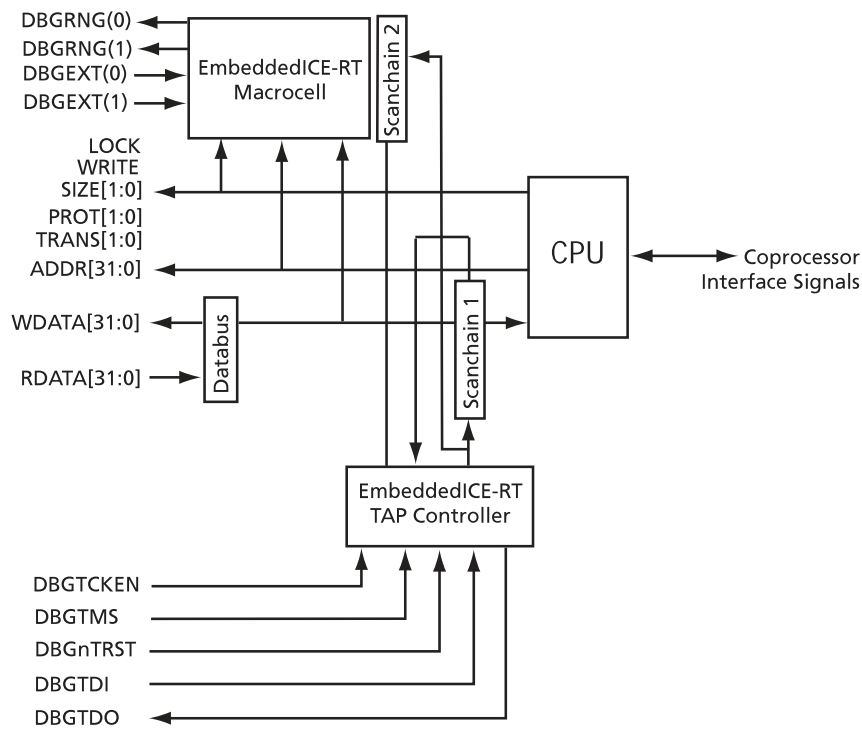


Figure 1 • CoreMP7 Top-Level Block Diagram

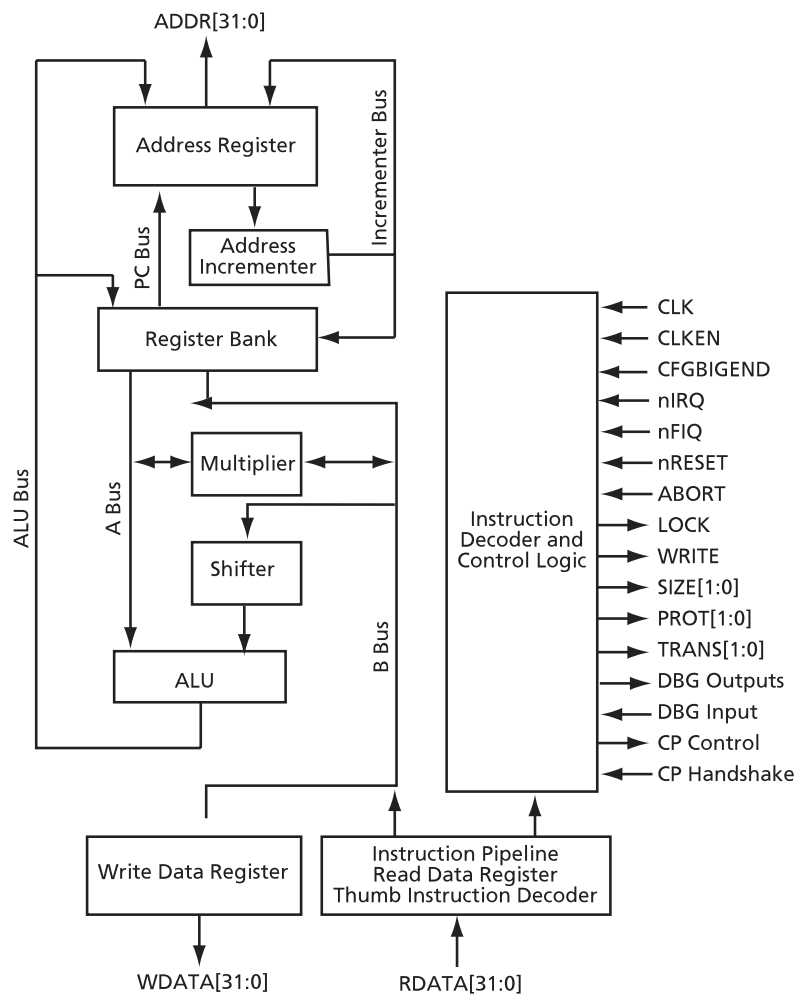


Figure 2 • CoreMP7 CPU Block Diagram

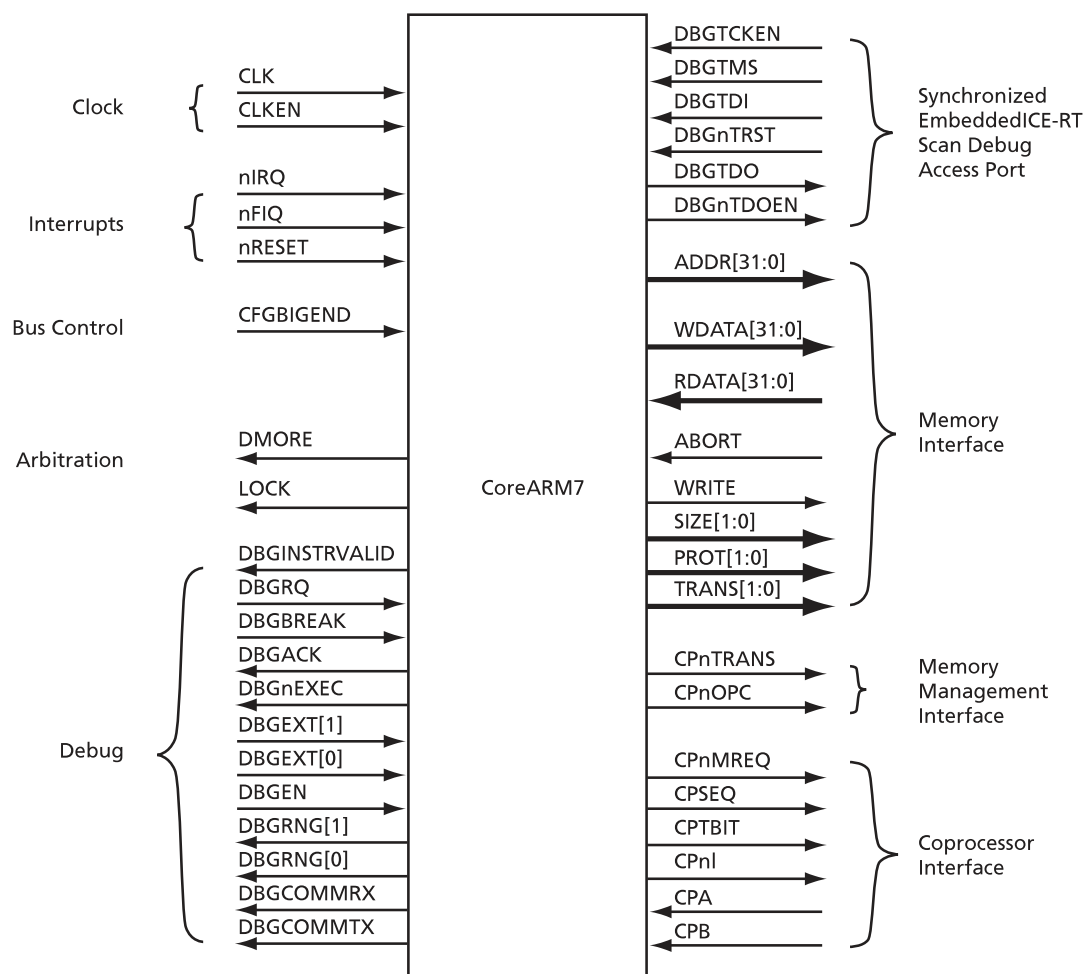


Figure 3 • CoreMP7 Functional Diagram

The signals of the CoreMP7 are listed in Table 2.

Table 2 • Signal Descriptions

| Name        | Type  | Description                         |
|-------------|-------|-------------------------------------|
| ABORT       | Input | Memory abort or bus error           |
| CFGBIGEND   | Input | Big/Little Endian configuration     |
| CLK         | Input | Clock                               |
| CLKEN       | Input | Clock enable                        |
| CPA         | Input | Coprocessor absent                  |
| CPB         | Input | Coprocessor busy                    |
| DBGBREAK    | Input | EICE breakpoint/watchband indicator |
| DBGEN       | Input | Debug enable                        |
| DBGEXT[1:0] | Input | EICE external input 0               |

**Note:** The CoreMP7 is available with either the native ARM7 bus interface or with an AHB wrapper. The use of the AHB wrapper changes or transforms some of the signals in Table 2. This is discussed in detail later in this document.

Table 2 • Signal Descriptions (Continued)

| Name           | Type   | Description                                                                     |
|----------------|--------|---------------------------------------------------------------------------------|
| DBGnTRST       | Input  | Test reset                                                                      |
| DBGnRQ         | Input  | Debug request                                                                   |
| DBGnTCKEN      | Input  | Test clock enable                                                               |
| DBGnTDI        | Input  | EICE data in                                                                    |
| DBGnTMS        | Input  | EICE mode select                                                                |
| nFIQ           | Input  | Interrupt request                                                               |
| nIRQ           | Input  | Fast interrupt request                                                          |
| nRESET         | Input  | Reset                                                                           |
| RDATA[31:0]    | Input  | Read data bus                                                                   |
| ADDR[31:0]     | Output | Address bus                                                                     |
| CPnI           | Output | Coprocessor instruction (asserted low)                                          |
| CPnMREQ        | Output | Memory request (asserted low)                                                   |
| CPnOPC         | Output | Opcode fetch (asserted low)                                                     |
| CPnTRANS       | Output | Memory translate (asserted low)                                                 |
| CPSEQ          | Output | Sequential address                                                              |
| CPTBIT         | Output | Processor in Thumb mode                                                         |
| DBGACK         | Output | Debug acknowledge                                                               |
| DBGnCOMMRX     | Output | EICE communication channel receive                                              |
| DBGnCOMMTX     | Output | EICE communication channel transmit                                             |
| DBGnEXEC       | Output | Executed (asserted low)                                                         |
| DBGnTDOEN      | Output | TDO enable (asserted low)                                                       |
| DBGnRNG[1:0]   | Output | EICE rangeout                                                                   |
| DBGnTDO        | Output | EICE data out                                                                   |
| DBGnINSTRVALID | Output | ETM Instruction valid indicator                                                 |
| DMORE          | Output | Set when next data memory access is followed by a sequential data memory access |
| LOCK           | Output | Locked transaction operation                                                    |
| PROT[1:0]      | Output | Indicates code, data, or privilege level                                        |
| SIZE[1:0]      | Output | Memory access width                                                             |
| TRANS          | Output | Next transaction type (i, n, s)                                                 |
| WDATA[31:0]    | Output | Write data bus                                                                  |
| WRITE          | Output | Indicates write access                                                          |

**Note:** The CoreMP7 is available with either the native ARM7 bus interface or with an AHB wrapper. The use of the AHB wrapper changes or transforms some of the signals in Table 2. This is discussed in detail later in this document.

## Programmer's Model

This section summarizes the programmer's model of the CoreMP7. Supporting detail is available in the *ARM7TDMI-S Technical Reference Manual* (available for download at [www.arm.com](http://www.arm.com)) and the *ARM Architecture Reference Manual*, which can be purchased at [www.amazon.com](http://www.amazon.com).

The CoreMP7 processor implements the ARMv4T architecture and includes both the 32-bit ARM instruction set and the 16-bit Thumb instruction set.

## Processor Operating States

The CoreMP7 processor has two operating states:

**ARM state:** 32-bit, word-aligned ARM instructions are executed in this state.

**Thumb state:** 16-bit, halfword-aligned Thumb instructions are executed in this state.

In Thumb state, the Program Counter (PC) uses bit 1 to select between alternate halfwords.

**Note:** Transition between ARM and Thumb states does not affect the processor mode or the register contents.

## Switching State

You can switch the operating state of the CoreMP7 between ARM state and Thumb state using the BX instruction. This is described fully in the *ARM Architecture Reference Manual*.

All exception handling is performed in ARM state. If an exception occurs in Thumb state, the processor reverts to ARM state. The transition back to Thumb state occurs automatically on return.

## Memory Formats

The CoreMP7 processor views memory as a linear collection of bytes, numbered in ascending order from zero:

- Bytes 0 to 3 hold the first stored word.
- Bytes 4 to 7 hold the second stored word.
- Bytes 8 to 11 hold the third stored word.

Although both Little Endian and Big Endian memory formats are supported, it is recommended that you use Little Endian format.

## Data Types

The CoreMP7 processor supports the following data types:

- Word (32-bit)
- Halfword (16-bit)
- Byte (8-bit)

You must align these as follows:

- Word quantities must be aligned to four-byte boundaries.
- Halfword quantities must be aligned to two-byte boundaries.
- Byte quantities can be placed on any byte boundary.

## Operating Modes

The CoreMP7 processor has seven operating modes:

- User mode is the usual ARM program execution state, and is used for executing most application programs.
- Fast interrupt (FIQ) mode supports a data transfer or channel process.
- Interrupt (IRQ) mode is used for general-purpose interrupt handling.
- Supervisor mode is a protected mode for the operating system.
- Abort mode is entered after a data or instruction prefetch abort.
- System mode is a privileged user mode for the operating system.
- Undefined mode is entered when an undefined instruction is executed.

Modes other than User mode are collectively known as privileged modes. Privileged modes are used to service interrupts or exceptions, or to access protected resources.

## Registers

The CoreMP7 processor has a total of 37 registers:

- 31 general-purpose 32-bit registers
- 6 status registers

These registers are not all accessible at the same time. The processor state and operating mode determine which registers are available to the programmer.

## The ARM State Register Set

In ARM state, 16 general registers and one or two status registers are accessible at any one time. In privileged modes, mode-specific banked registers become available. [Figure 4 on page 8](#) shows which registers are available in each mode.

The ARM state register set contains 16 directly accessible registers, r0 to r15. An additional register, the Current Program Status Register (CPSR), contains condition code flags, and the current mode bits. Registers r0 to r13 are general-purpose registers used to hold either data or address values. Registers r14 and r15 have special functions as the Link Register and Program Counter.



### Link Register

Register 14 is used as the subroutine Link Register (LR).

Register 14 (r14) receives a copy of r15 when a Branch with Link (BL) instruction is executed.

At all other times, you can treat r14 as a general-purpose register.

The corresponding banked registers—r14 svc, r14 irq, r14 fiq, r14 abt, and r14 und—are similarly used to hold the return values of r15 when interrupts and exceptions arise, or when BL instructions are executed within interrupt or exception routines.

### Program Counter

Register 15 holds the Program Counter (PC).
















In ARM state, bits [1:0] of r15 are zero. Bits [31:2] contain the PC.

In Thumb state, bit [0] is zero. Bits [31:1] contain the PC.

In privileged modes, another register, the Saved Program Status Register (SPSR), is accessible. This contains the condition code flags, and the mode bits saved as a result of the exception that caused entry to the current mode.

Figure 4 shows the ARM state registers.

ARM State General Registers and Program Counter

| System and User | FIQ                                                                                         | Supervisor                                                                                  | Abort                                                                                       | IRQ                                                                                          | Undefined                                                                                     |
|-----------------|---------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| r0              | r0                                                                                          | r0                                                                                          | r0                                                                                          | r0                                                                                           | r0                                                                                            |
| r1              | r1                                                                                          | r1                                                                                          | r1                                                                                          | r1                                                                                           | r1                                                                                            |
| r2              | r2                                                                                          | r2                                                                                          | r2                                                                                          | r2                                                                                           | r2                                                                                            |
| r3              | r3                                                                                          | r3                                                                                          | r3                                                                                          | r3                                                                                           | r3                                                                                            |
| r4              | r4                                                                                          | r4                                                                                          | r4                                                                                          | r4                                                                                           | r4                                                                                            |
| r5              | r5                                                                                          | r5                                                                                          | r5                                                                                          | r5                                                                                           | r5                                                                                            |
| r6              | r6                                                                                          | r6                                                                                          | r6                                                                                          | r6                                                                                           | r6                                                                                            |
| r7              | r7                                                                                          | r7                                                                                          | r7                                                                                          | r7                                                                                           | r7                                                                                            |
| r8              |  r8 fiq  | r8                                                                                          | r8                                                                                          | r8                                                                                           | r8                                                                                            |
| r9              |  r9 fiq  | r9                                                                                          | r9                                                                                          | r9                                                                                           | r9                                                                                            |
| r10             |  r10 fiq | r10                                                                                         | r10                                                                                         | r10                                                                                          | r10                                                                                           |
| r11             |  r11 fiq | r11                                                                                         | r11                                                                                         | r11                                                                                          | r11                                                                                           |
| r12             |  r12 fiq | r12                                                                                         | r12                                                                                         | r12                                                                                          | r12                                                                                           |
| r13             |  r13 fiq |  r13 svc |  r13 abt |  r13 irq |  r13 und |
| r14             |  r14 fiq |  r14 svc |  r14 abt |  r14 irq |  r14 und |
| r15 (PC)        | r15 (PC)                                                                                    | r15 (PC)                                                                                    | r15 (PC)                                                                                    | r15 (PC)                                                                                     | r15 (PC)                                                                                      |

ARM State Program Status Registers

|      |                                                                                              |                                                                                              |                                                                                              |                                                                                               |                                                                                                |
|------|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| CPSR | CPSR                                                                                         | CPSR                                                                                         | CPSR                                                                                         | CPSR                                                                                          | CPSR                                                                                           |
|      |  SPSR fiq |  SPSR svc |  SPSR abt |  SPSR irq |  SPSR und |

 = banked register

Figure 4 • CoreMP7 Register Organization in the ARM State

## The Thumb State Register Set

The Thumb state register set is a subset of the ARM state set. The programmer has direct access to the following:

- Eight general registers, r0–r7
- The PC
- A Stack Pointer (SP)
- A Link Register (LR)
- The CPSR

There are banked SPs, LRs, and SPSRs for each privileged mode. The Thumb state register set is shown in [Figure 5](#).

**Thumb State General Registers and Program Counter**

| System and User | FIQ    | Supervisor | Abort  | IRQ    | Undefined |
|-----------------|--------|------------|--------|--------|-----------|
| r0              | r0     | r0         | r0     | r0     | r0        |
| r1              | r1     | r1         | r1     | r1     | r1        |
| r2              | r2     | r2         | r2     | r2     | r2        |
| r3              | r3     | r3         | r3     | r3     | r3        |
| r4              | r4     | r4         | r4     | r4     | r4        |
| r5              | r5     | r5         | r5     | r5     | r5        |
| r6              | r6     | r6         | r6     | r6     | r6        |
| r7              | r7     | r7         | r7     | r7     | r7        |
| SP              | SP fiq | SP svc     | SP abt | SP irq | SP und    |
| LR              | LR fiq | LR svc     | LR abt | LR irq | LR und    |
| PC              | PC     | PC         | PC     | PC     | PC        |

**Thumb State Program Status Registers**

|      |          |          |          |          |          |
|------|----------|----------|----------|----------|----------|
| CPSR | CPSR     | CPSR     | CPSR     | CPSR     | CPSR     |
|      | SPSR fiq | SPSR svc | SPSR abt | SPSR irq | SPSR und |

 = banked register

Figure 5 • CoreMP7 Thumb State Registers

## The Relationship Between ARM State and Thumb State Registers

The Thumb state registers relate to the ARM state registers in the following way:

- Thumb state r0–r7 and ARM state r0–r7 are identical.
- Thumb state CPSR and SPSR, and ARM state CPSR and SPSR are identical.
- Thumb state SP maps onto ARM state r13.
- Thumb state LR maps onto ARM state r14.
- The Thumb state PC maps onto the ARM state PC (r15).

These relationships are shown in [Figure 6](#).

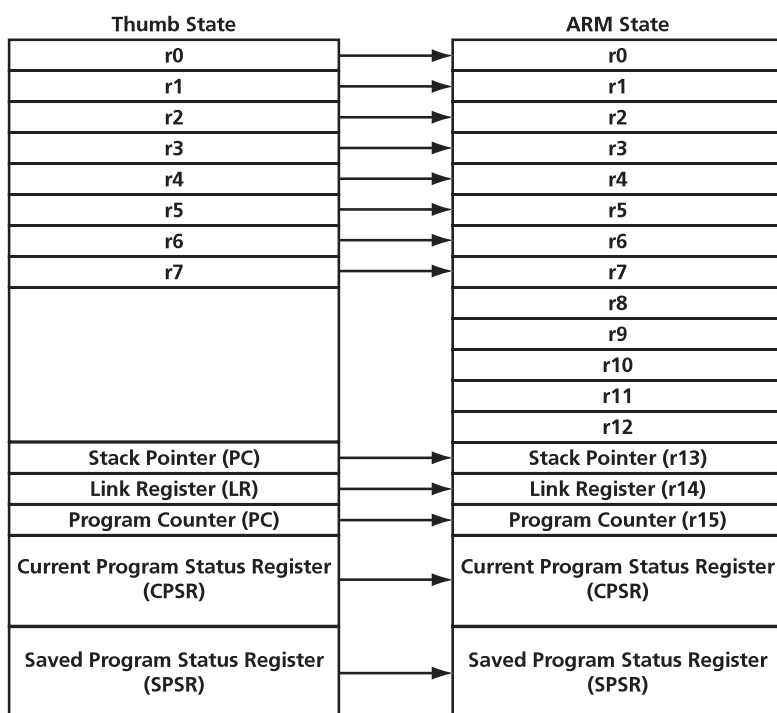


Figure 6 • Mapping of Thumb State Registers to ARM State Registers

**Note:** Registers r0–r7 are known as the low registers. Registers r8–r15 are known as the high registers.

## The Program Status Registers

The CoreMP7 core contains a CPSR and five SPSRs for exception handlers to use. The program status registers the following:

- Hold the condition code flags
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The arrangement of bits is shown in [Figure 7](#).

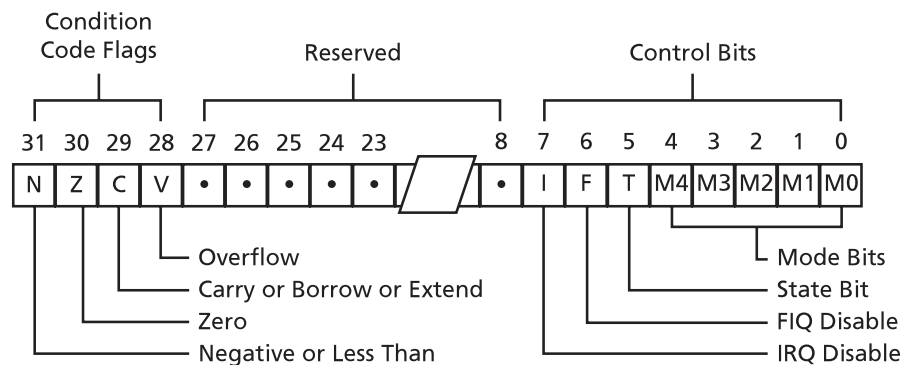


Figure 7 • Program Status Register Format

### The Condition Code Flags

The N, Z, C, and V bits are the condition code flags. You can set these bits by arithmetic and logical operations. The flags can also be set by MSR and LDM instructions. The CoreMP7 processor tests these flags to determine whether to execute an instruction.

All instructions can be executed conditionally in ARM state. In Thumb state, only the Branch instruction can be executed conditionally. For more information about conditional execution, see the *ARM Architecture Reference Manual*.

## AHB Wrapper

The AHB wrapper interfaces between the CoreMP7 native ARM7 interface and the AHB bus. The module translates access from the core to AHB accesses when the core is the current master. The external interface signals from the wrapper are described in [Table 3](#).

Table 3 • AHB Wrapper External Interface

| Signal<br>External Master I/F | Direction | Description                                                                                                                                                                                                                                                                   |
|-------------------------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HCLK                          | Input     | Bus clock. This clock times all bus transfers. All signal timings are related to the rising edge of <b>HCLK</b> .                                                                                                                                                             |
| HRESETn                       | Input     | Reset. The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW AHB signal.                                                                                                                                                |
| HREADY                        | Input     | Transfer done. When HIGH the <b>HREADY</b> signal indicates that a transfer has finished on the bus. This signal can be driven LOW to extend a transfer.                                                                                                                      |
| HRESP[1:0]                    | Input     | Transfer response. Indicates an OKAY, ERROR, RETRY, or SPLIT response.                                                                                                                                                                                                        |
| HGRANT                        | Input     | Bus grant. Indicates that the CoreMP7 is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when <b>HREADY</b> is HIGH, so a master gains access to the bus when both <b>HREADY</b> and <b>HGRANT</b> are HIGH. |
| HADDR[31:0]                   | Output    | This is the 32-bit system address bus.                                                                                                                                                                                                                                        |
| HTRANS[1:0]                   | Output    | Transfer type. Indicates the type of the current transfer.                                                                                                                                                                                                                    |
| HWRITE                        | Output    | Transfer direction. When HIGH this signal indicates a write transfer and when LOW a read transfer.                                                                                                                                                                            |
| HSIZE[2:0]                    | Output    | Transfer size. Indicates the size of the transfer, which can be byte (8-bit), halfword (16-bit), or word (32-bit).                                                                                                                                                            |
| HBURST[2:0]                   | Output    | Burst type. Indicates if the transfer forms part of a burst. The CoreMP7 performs incrementing bursts of type INCR.                                                                                                                                                           |
| HPROT[3:0]                    | Output    | Protection control. These signals indicate if the transfer is an opcode fetch or data access, and if the transfer is a Supervisor mode access or User mode access.                                                                                                            |
| HWDATA[31:0]                  | Output    | 32-bit data from the MASTER.                                                                                                                                                                                                                                                  |
| HRDATA[31:0]                  | Input     | 32-bit data written back to the MASTER.                                                                                                                                                                                                                                       |

## CoreMP7 Variants

There are two implementations of CoreMP7 for all M7 devices except for the M7A3P250 which only supports the CoreMP7S variant. The utilization and performance of the variants are shown in [Table 1 on page 2](#).

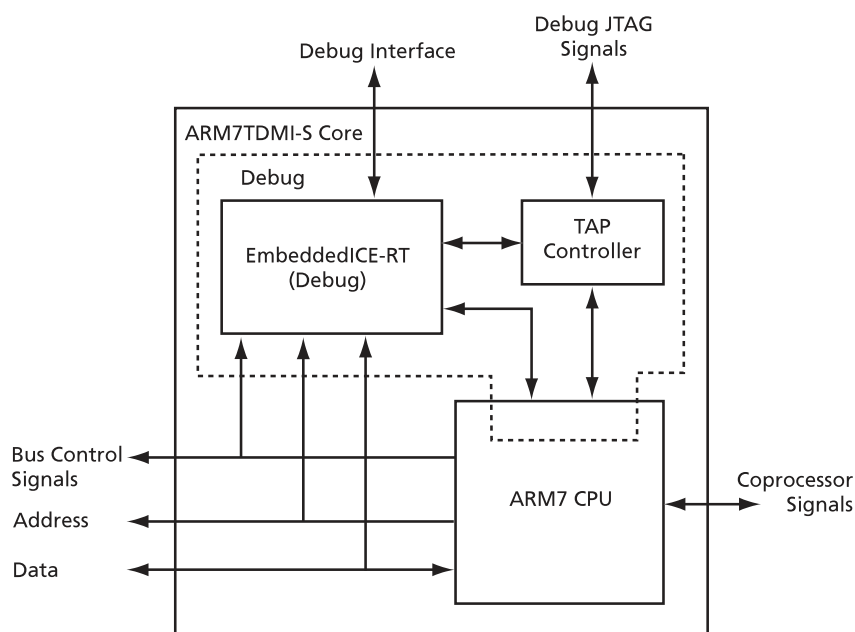


Figure 8 • **ARM7 TDMI-S Core**

### CoreMP7Sd

The CoreMP7Sd is configured with all features of the ARM7TDMI-S. The variant incorporates the full debug functionality of the ARM7TDMI-S and is fully compliant with RealView RVDS, RVDK, and other ARM software debug tools.

### CoreMP7S

The CoreMP7S is optimized for maximum speed and minimum area, and has the same features as CoreMP7Sd except that it does not include the ICE-RT debug block, the TAP controller, or the coprocessor interface and is Little Endian only, to reduce the size of the core. This means that the standard debug tools cannot be used with this variant of CoreMP7.

### No Debug

This is the most obvious characteristic of the CoreMP7S variant. To reduce area, the Debug EmbeddedICE-RT macrocell, the EmbeddedICE-RT TAP controller, and the scan logic have been optimized out. This means that standard software debug tools cannot be used when these variants of the CoreMP7 are instantiated. Users of these variants can use the comparable variant that includes debug for development, and when the application has been fully tested and debugged, one of

these variants can be instantiated to reduce area in the final shipping product.

### No Coprocessor Interface

The coprocessor interface is a rarely used feature in ARM7 family microprocessors and has been removed from the CoreMP7S variant to minimize area.

### Little Endian Only

Most microprocessor-based systems use Little Endian byte ordering. The option of selecting Big Endian has been removed from the CoreMP7S variant to minimize area.

### On-Chip RAM Consumed by Register Block

To minimize the area, the CoreMP7 variants map the processor register block into on-chip RAM. RAM blocks used to implement CoreMP7 registers are no longer available for use in user designs.

## Delivery and Deployment

The CoreMP7 is delivered as a series of files from the Actel CoreConsole IP Deployment Platform (IDP) development tool, and these files are directly imported into the Design and Simulation folders for use in the Actel Libero IDE FPGA tool suite. The CoreMP7 files consist of the BFM files and test wrapper, AHB wrapper, and the A7S secured CDB file, which is instantiated on the user device at programming.

## Bus Functional Model

During the development of an FPGA-based SoC, a number of stages of testing may be undertaken. This can involve some, or all, of the following approaches:

- Hardware simulation, using Verilog or VHDL
- Software simulation, using a host-based instruction set simulator (ISS) of the SoC's processor
- Hardware and software co-verification, using a fully functional model of the processor in Verilog, VHDL, or SWIFT form, or using a tool such as Seamless

Due to the rapid prototyping capability of FPGAs, however, integration of hardware and software often occurs earlier in the SoC development cycle for FPGA targets than it would for ASIC targets. Therefore, hardware and software co-verification, which can be very slow, is not a critical issue except in the most complex FPGA-based SoCs.

The planned availability of ARM-based SoC solutions to Actel FPGA customers necessitates that Actel provide support for the test approaches described above. In particular, there should be an emphasis on providing solutions for hardware simulation and for software simulation.

A software simulation solution is already available to customers as part of the proposed ARM package. This package contains the RealView Instruction Set Simulator, which provides ARM7 instruction accurate simulation, as well as powerful features, such as integration with the RealView debugger.

Support for hardware simulation is also proposed. The CoreConsole SoC configuration utility provides a means for the developer to stitch together IP blocks using a bus fabric of choice. It generates a system testbench, controlled by a script-driven, bus functional model (BFM) of the ARM7 processor. The ARM7 BFM allows the developer to model low-level bus transactions, which allow verification of connectivity of the various IP blocks and the system memory map presented to the ARM7 by the rest of the hardware.

This document specifies the following aspects of the ARM7 BFM:

- Functionality
- BFM usage flow
- BFM script language
- Platforms
- Supported simulation tools
- Example BFM use case

## BFM Usage Flow

As the BFM is part of an overall system test strategy, it is helpful to look at the context in which it is used. [Figure 9 on page 15](#) shows the various components within an example system-level testbench that can be generated by CoreConsole.

In [Figure 9 on page 15](#), it is assumed that the developer specifies an SoC subsystem by selecting the processor, bus fabric, IP blocks, and memory subsystem in CoreConsole. In this example, the user selects the following:

- ARM7 processor,
- AMBA AHB bus fabric
- MAC 10/100 IP core
- CoreUART IP core
- External SSRAM and Flash memory

The user also specifies the memory map of the system. Based on this information, CoreConsole generates the following outputs, amongst others:

- Verilog/VHDL model of SoC subsystem
- Verilog/VHDL models of IP cores
- Verilog/VHDL model of ARM7 BFM
- BFM test script
- System-level skeleton testbench

The BFM acts as a pin-for-pin replacement of the ARM7TDMI-S in the SoC subsystem. It initiates bus transactions on the native ARM7 bus, which are cycle-accurate with real bus cycles that the ARM7TDMI-S would produce. It has no knowledge, however, of real ARM7 instructions.

At this point, the BFM may be used to run a basic test of the SoC subsystem using the skeleton system testbench. The BFM is fully integrated into the CoreConsole user flow. In particular, if the user has an AHB-based CoreMP7 subsystem, CoreConsole automatically derives the memory map of the user's subsystem. CoreConsole uses this information to generate an overall BFM test script, which includes customized "scriptlets" for each resource attached to the AHB or APB buses.

The developer may edit the SoC Verilog/VHDL to add new design blocks, such as the VideoCodec in the above diagram. The system-level testbench may also be filled out by the developer to include tasks that test any newly

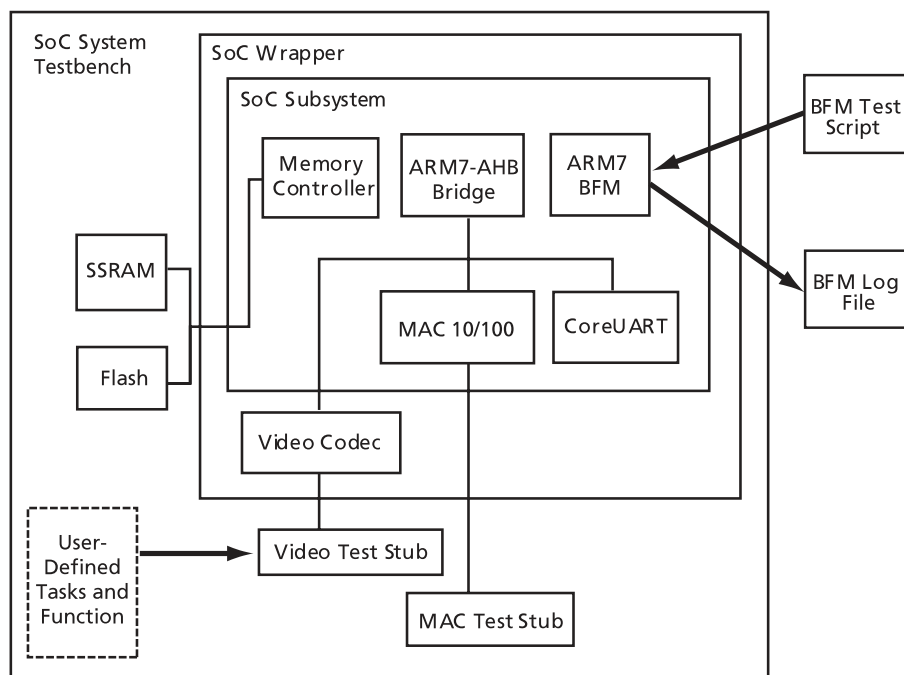


Figure 9 • SoC System-Level Testbench Example

added functionality or additional stubs that allow more complex system testing involving the IP cores. The BFM input scripts may also be manually enhanced to allow the user to test access to register locations in newly added logic. In this way, the user can provide stimuli to the

system from the inside (via the ARM7 BFM), as well as from the outside (via testbench tasks).

Figure 10 shows the design flow into which the BFM fits.

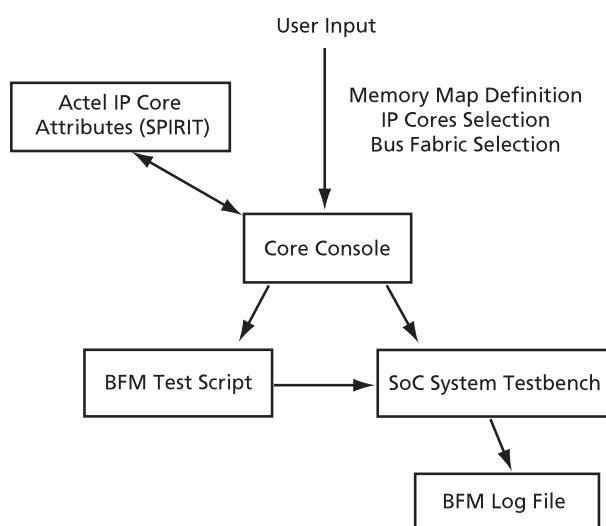


Figure 10 • BFM Flow Diagram



## Functionality

This section describes the specific functionality of the ARM7 BFM. The BFM models the ARM7 native bus. Specifically, this models the following bus signals:

- ADDR, // address bus
- WDATA, // write data bus
- RDATA, // read data bus
- TRANS, // next transaction type (i, n, or s)
- WRITE, // indicates write access
- CLKEN, // clock enable

The BFM also models the following control signals:

- CFGBIGEND, // big/little endian configuration
- CLK, // clock
- nFIQ, // interrupt request
- nIRQ, // fast interrupt request
- SIZE, // memory access width

## CoreConsole v1.1

### ARM7 Pin Compatibility

The BFM model is pin-for-pin compatible with the ARM7TDMI-S. This allows the model to be dropped into the space that would be occupied by the ARM in the Verilog/VHDL system testbench.

### ARM7 Bus Cycle Accuracy

The bus cycle timings for the ARM7 native bus signals are specified in the [ARM7TDMI Technical Reference Manual](#). The BFM models these bus cycles exactly.

### Scripting

In order to provide a simple and extensible mechanism for providing stimuli to the BFM, a BFM scripting language is defined (see the ["BFM Script Language" section](#)). The scripting language can initiate writes to system resources, reads from system resources (with or without checking of expected data), and to wait for events.

### Self-Checking

The BFM gives a pass/fail indication at the end of a test run. This is based on whether any of the expected data read checks failed or not.

### Endianess

The BFM supports both big and little-endian memory configurations. For byte and halfword transfers, it reads and writes data from/to the appropriate data lanes.

### Interrupt Support

The BFM has the ability to wait for either of the two ARM7 interrupt lines to be triggered, before proceeding with the remainder of the test script.

## Log File Generation

The BFM generates output messages to the console of the simulation tool and also generates a plain text log file.

## BFM Script Language

The following script commands are defined for use by the BFM:

### memmap

This command is used to associate a label, representing a system resource, with a memory map location. The other BFM script commands may perform accesses to locations within this resource by referencing this label and a register offset relative to this base address.

#### Syntax

memmap resource name base address;

#### resource name

This is a string containing the user-friendly instance name of the resource being accessed. For BFM scripts generated automatically by CoreConsole, this name corresponds to the instance name of the associated core in the generated subsystem Verilog or VHDL.

#### base address

This is the base address of the resource, in hexadecimal.

### write

This command causes the BFM to perform a write to a specified offset, within the memory map range of a specified system resource.

#### Syntax

write width resource name byte offset data;

#### width

This takes on the enumerated values of W, H, or B, for word, halfword, or byte.

#### resource name

This is a string containing the user-friendly instance name of the resource being accessed, as defined by the user in the memory map (when input to CoreConsole).

#### byte offset

This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.

#### data

This is the data to be written. It is specified as a hexadecimal value.

### Example

write W videoCodec 20 11223344;

## read

This command causes the BFM to perform a read of a specified offset, within the memory map range of a specified system resource.

### Syntax

read width resource name byte offset;

#### width

This takes on the enumerated values of W, H, or B, for word, halfword, or byte.

#### resource name

This is a string containing the user-friendly instance name of the resource being accessed, as defined by the user in the memory map (when input to CoreConsole).

#### byte offset

This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.

### Example

read W videoCodec 20;

## readcheck

This command causes the BFM to perform a read of a specified offset, within the memory map range of a specified system resource, and to compare the read value with the expected value provided.

### Syntax

readcheck width resource name byte offset data;

#### width

This takes on the enumerated values of W, H, or B, for word, halfword, or byte.

#### resource name

This is a string containing the user-friendly instance name of the resource being accessed, as defined by the user in the memory map (when input to CoreConsole).

#### byte offset

This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.

#### data

This is the expected read data. It is specified as a hexadecimal value.

### Example

readcheck W videoCodec 20 11223344;

## poll

This command continuously reads a specified location until a requested value is obtained. This command allows one or more bits of the read data to be masked out. This allows, for example, poll waiting for a ready bit to be set, while ignoring the values of the other bits in the location being read.

### Syntax

poll width resource name byte offset data bitmask;

#### width

This takes on the enumerated values of W, H, or B, for word, halfword, or byte.

#### resource name

This is a string containing the user-friendly instance name of the resource being accessed.

#### byte offset

This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.

#### bitmask

The bitmask is ANDed with the read data and the result is then compared to the bitmask itself. If equal, then all the bits of interest are at their required value and the poll command is complete. If not equal, then the polling continues.

## wait

This command causes the BFM script to stall for a specified number of clock periods.

### Syntax

wait num clock ticks;

#### num clock ticks

This is the number of CoreMP7 clock periods, during which the BFM stalls (doesn't initiate any bus transactions).

## waitfiq

This command causes the BFM to wait until an interrupt event (high to low transition) is seen on the nFIQ pin before proceeding with the execution of the remainder of the script.

### Syntax

waitfiq;

## waitirq

This command causes the BFM to wait until an interrupt event (high to low transition) is seen on the nIRQ pin before proceeding with the execution of the remainder of the script.

### Syntax

waitirq;

## Supported Simulation Tools

BFM is delivered to the user as both a Verilog and VHDL model.

## Timing Shell

The BFM incorporates a timing shell, which performs setup/hold checks on inputs and delays outputs by the appropriate amount from the rising clock edge.

## Example BFM Use Case

This provides an example use case of the ARM7 BFM. The example SoC used in this section is the same as that shown in [Figure 9 on page 15](#). In this system, the developer requires two Actel IP cores: the MAC 10/100 and the CoreUART.

### SPIRIT Attributes

CoreConsole has access to a database of Actel IP cores and a list of attributes for each core. These attributes are organized according to the SPIRIT specification, in XML. For example, in the case of the CoreUART, the attributes would indicate that there are three registers, as in [Table 4](#).

Table 4 • CoreUART Attributes

| Offset | Register             | Read/Write | Width |
|--------|----------------------|------------|-------|
| 0      | Uart Status Register | R          | Byte  |
| 1      | Uart Tx data         | W          | Byte  |
| 2      | Uart Rx data         | R          | Byte  |

Based on these attributes, CoreConsole can determine that when generating the BFM script, there are three locations corresponding to the UART that can be accessed. In this case, none of the registers are RW, so there will not be any self-checking that can be performed for the UART. Nevertheless, the bus transactions do take place and the cycles may be viewed in a waveform of the simulator.

## Memory Map

The designer must feed in the memory map of the SoC to CoreConsole. During this stage, the absolute address ranges of the various system resources in the ARM7 memory map are fed in. Also, user-friendly instance names of these resources are fed in.

For example, the user could feed the memory map information into CoreConsole that is given in [Table 5](#).

Based on the information in [Table 5](#), CoreConsole generates the SoC subsystem corresponding to the Actel IP cores present. It also generates a BFM script, which accesses all the registers in the Actel IP cores.

Table 5 • Memory Map Information

| Resource   | Actel IP Core | Address Range |
|------------|---------------|---------------|
| ssram      | N             | 0-3fffff      |
| flash      | N             | 400000-7fffff |
| uart       | Y             | c00000-c0000b |
| mac        | Y             | d00000-d00040 |
| videocodec | N             | e00000-e000ff |

### Processor Choice

In this example, the user selects an ARM7 as the processor of choice in CoreConsole. The BFM in this specification only relates to ARM7.

### Bus Fabric Selection

The user may select one of a number of bus fabrics in CoreConsole. For example, the user could select AMBA AHB-Lite. However, this selection is irrelevant for the ARM7 BFM, as it is concerned only with generating native ARM7 bus based transactions.

### Automatic BFM Scriptlet

At this point, having run CoreConsole to completion, a BFM scriptlet is available. This would look something like the following:

```
read B uart 0;
write B uart 4 bb;
read B uart 8;
write B mac 30 11;
readcheck B mac 11;
```

### Run BFM

The developer can run the BFM with the automatic script or edit the script to put in bus transactions to/from any new logic that has been added to the SoC. For example, transactions to/from the registers in the new VideoCodec block could be added.

The skeleton system-level testbench, generated by CoreConsole, could also be modified, to add some external resources (e.g., models of SSRAM and Flash) and some high-level tasks.

Upon running the system simulation, messages appear in the console window of the simulation tool.

## AC Parameters

This section gives the AC timing parameters of the CoreMP7 processor.

### Timing Diagrams

Timing diagrams are shown in Figure 11, Figure 12 on page 20, Figure 13 on page 20, Figure 14 on page 21, and Figure 15 on page 21.

### Data Access Timing

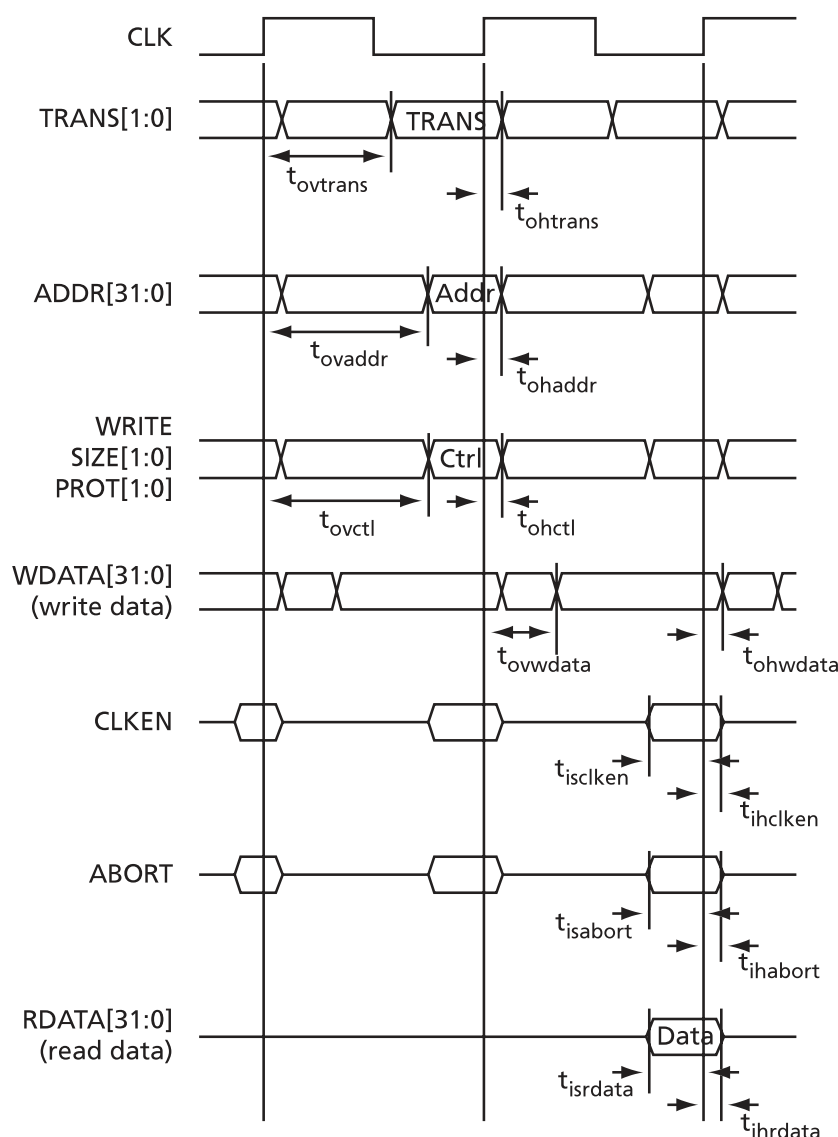


Figure 11 • Data Access Timing

## Coprocessor Timing

The Coprocessor timing is included for completeness although it is expected that the Coprocessor interface is omitted in most deployments of the CoreMP7.

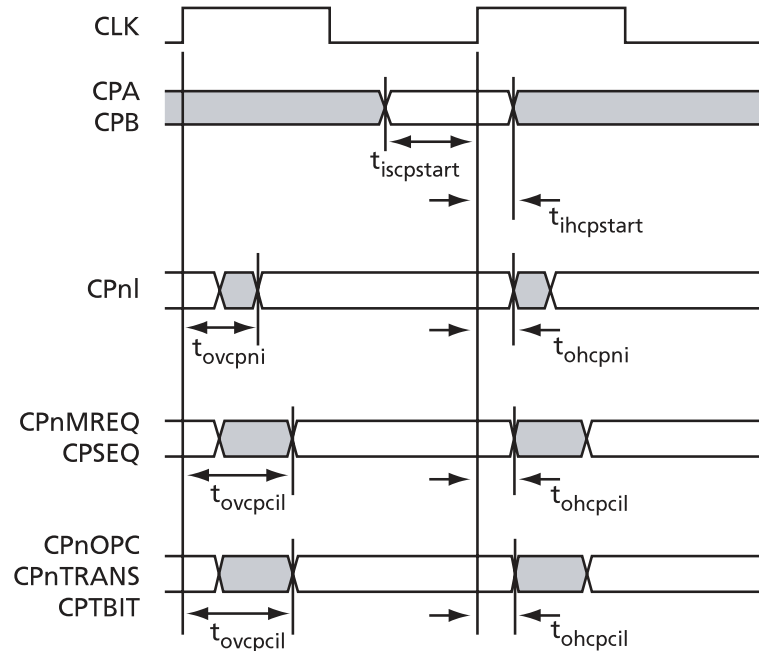


Figure 12 • Coprocessor Timing

## Exception Timing

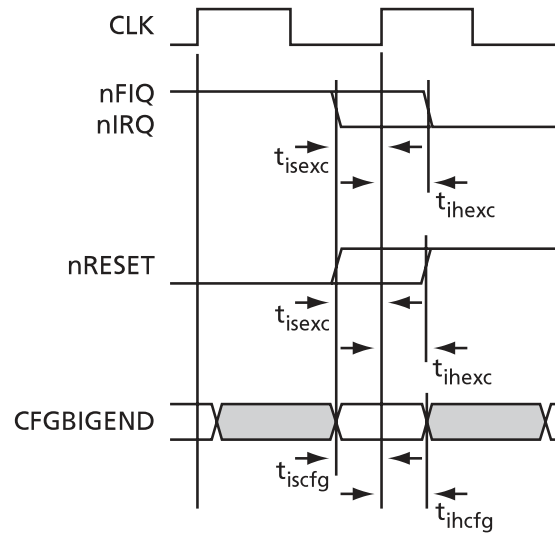


Figure 13 • Exception Timing

## Debug Timing

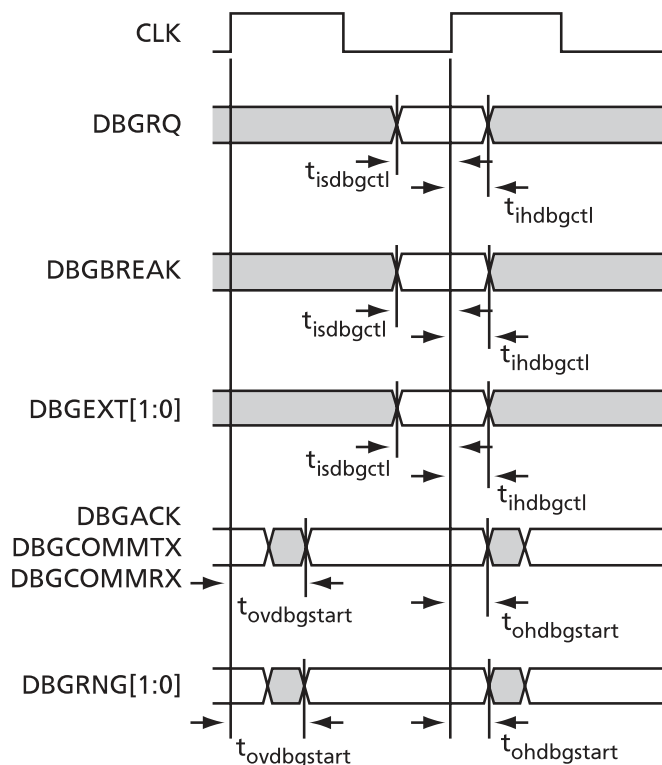


Figure 14 • Debug Timing

## Scan Timing

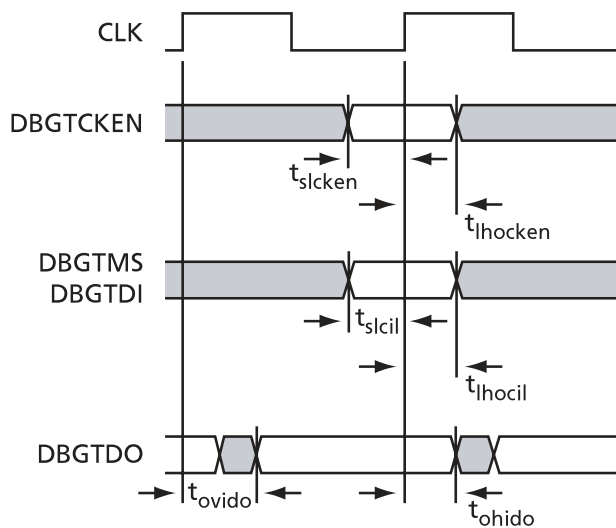


Figure 15 • Scan Timing

## AC Timing Parameter Definitions

AC Timing Parameters shows target AC parameters. All figures are expressed as percentages of the **CLK** period at maximum operating frequency.

**Note:** Where 0% is shown, this indicates the hold time to clock edge plus the maximum clock skew for internal clock buffering.

Table 6 • AC Timing Parameters

| Symbol          | Parameter                                                                        | Min  | Max |
|-----------------|----------------------------------------------------------------------------------|------|-----|
| $t_{CYC}$       | <b>CLK</b> cycle time                                                            | 100% | –   |
| $t_{ISCLKEN}$   | <b>CLKEN</b> input setup to rising <b>CLK</b>                                    | 60%  | –   |
| $t_{IHCLKEN}$   | <b>CLKEN</b> input hold from rising <b>CLK</b>                                   | –    | 0%  |
| $t_{ISABORT}$   | <b>ABORT</b> input setup to rising <b>CLK</b>                                    | 40%  | –   |
| $t_{IHABORT}$   | <b>ABORT</b> input hold from rising <b>CLK</b>                                   | –    | 0%  |
| $t_{ISRDATA}$   | <b>RDATA</b> input setup to rising <b>CLK</b>                                    | 10%  | –   |
| $t_{ISRST}$     | <b>nRESET</b> input setup to rising <b>CLK</b>                                   | 90%  | –   |
| $t_{ISTRST}$    | <b>DBGnTRST</b> input setup to rising <b>CLK</b>                                 | 25%  | –   |
| $t_{IHRDATA}$   | <b>RDATA</b> input hold from rising <b>CLK</b>                                   | –    | 0%  |
| $t_{OCPTBIT}$   | Rising <b>CLK</b> to <b>CPTBIT</b> valid                                         | –    | 90% |
| $t_{ODBG}$      | Rising <b>CLK</b> to <b>DBGnEXEC</b> , <b>DBGINSTRVALID</b> valid                | –    | 40% |
| $t_{OLOMO}$     | Rising <b>CLK</b> to <b>DMORE</b> , <b>LOCK</b> valid                            | –    | 90% |
| $t_{OVADDR}$    | Rising <b>CLK</b> to <b>ADDR</b> valid                                           | –    | 90% |
| $t_{OHADDR}$    | <b>ADDR</b> hold time from rising <b>CLK</b>                                     | >0%  | –   |
| $t_{OVCTL}$     | Rising <b>CLK</b> to control valid                                               | –    | 90% |
| $t_{OHCTL}$     | Control hold time from rising <b>CLK</b>                                         | >0%  | –   |
| $t_{OVTRANS}$   | Rising <b>CLK</b> to transaction type valid                                      | –    | 50% |
| $t_{OHTRANS}$   | Transaction type hold time from rising <b>CLK</b>                                | >0%  | –   |
| $t_{OVWDATA}$   | Rising <b>CLK</b> to <b>WDATA</b> valid                                          | –    | 40% |
| $t_{OHWDATA}$   | <b>WDATA</b> hold time from rising <b>CLK</b>                                    | >0%  | –   |
| $t_{ISCPSTAT}$  | <b>CPA</b> , <b>CPB</b> input setup to rising <b>CLK</b>                         | 20%  | –   |
| $t_{IHCPSTAT}$  | <b>CPA</b> , <b>CPB</b> input hold from rising <b>CLK</b>                        | –    | 0%  |
| $t_{OVCPCTL}$   | Rising <b>CLK</b> to coprocessor control valid                                   | –    | 80% |
| $t_{OHCPCTL}$   | Coprocessor control hold time from rising <b>CLK</b>                             | >0%  | –   |
| $t_{OVCPNI}$    | Rising <b>CLK</b> to coprocessor <b>CPnI</b> valid                               | –    | 40% |
| $t_{OHCPNI}$    | Coprocessor <b>CPnI</b> hold time from rising <b>CLK</b>                         | >0%  | –   |
| $t_{ISEXC}$     | <b>nFIQ</b> , <b>nIRQ</b> , input setup to rising <b>CLK</b>                     | 10%  | –   |
| $t_{IHEXC}$     | <b>nFIQ</b> , <b>nIRQ</b> , <b>nRESET</b> hold from rising <b>CLK</b>            | –    | 0%  |
| $t_{ISCFG}$     | <b>CFGBIGEND</b> setup to rising <b>CLK</b>                                      | 10%  | –   |
| $t_{IHCFG}$     | <b>CFGBIGEND</b> hold from rising <b>CLK</b>                                     | –    | 0%  |
| $t_{ISDBGCTL}$  | <b>DBGBREAK</b> , <b>DBGEXT</b> , <b>DBGREQ</b> input setup to rising <b>CLK</b> | 10%  | –   |
| $t_{ISDBGSTAT}$ | Debug status inputs setup to rising <b>CLK</b>                                   | 10%  | –   |

Table 6 • AC Timing Parameters (Continued)

| Symbol          | Parameter                                                       | Min  | Max |
|-----------------|-----------------------------------------------------------------|------|-----|
| $t_{IHDBGSTAT}$ | Debug status inputs hold from rising <b>CLK</b>                 | –    | 0%  |
| $t_{OVDBGCTL}$  | Rising <b>CLK</b> to debug control valid                        | –    | 40% |
| $t_{OHDBCTL}$   | Debug control hold time from rising <b>CLK</b>                  | >0%  | –   |
| $t_{ISTCKEN}$   | <b>DBGTCKEN</b> input setup to rising <b>CLK</b>                | 60%' | –   |
| $t_{IHTCKEN}$   | <b>DBGTCKEN</b> input hold from rising <b>CLK</b>               | –    | 0%  |
| $t_{ISTCTL}$    | <b>DBGTDI</b> , <b>DBGTMS</b> input setup to rising <b>CLK</b>  | 35%  | –   |
| $t_{IHTCTL}$    | <b>DBGTDI</b> , <b>DBGTMS</b> input hold from rising <b>CLK</b> | –    | 0%  |
| $t_{OVTDO}$     | Rising <b>CLK</b> to <b>DBGTDO</b> valid                        | –    | 20% |
| $t_{OHTDO}$     | <b>DBGTDO</b> hold time from rising <b>CLK</b>                  | >0%  | –   |
| $t_{OVDBGSTAT}$ | Rising <b>CLK</b> to debug status valid                         | 40%  | –   |
| $t_{OHDBGSTAT}$ | Debug status hold time                                          | >0%  | –   |

## Debug

The ARM Debug Architecture uses a protocol converter box to allow the debugger to talk via a Joint Test Action Group (JTAG) port directly to the core. In effect, the scan chains in the core that are required for test are re-used for debugging.

The architecture uses the scan chains to insert instructions directly in to the ARM core. The instructions are executed on the core and, depending on the type of instruction that has been inserted, the core or the system state can be examined, saved, or changed. The architecture has the ability to execute instructions at a slow debug speed or to execute instructions at system speed (for example, if access to an external memory was required).

The fact that the debugger is actually using the JTAG scan chains to access the core is of no importance to the user, as the front end debugger remains exactly the same. The user could still use the debugger with a monitor program running on the target system or with an instruction set simulator that runs on the debugger host. In each case the debugging environment is the same.

The advantages of using the JTAG port are:

- Hardware access required by a system for test is re-used for debug.
- Core state and system state can be examined via the JTAG port.
- The target system does not have to be running in order to start debug.

A monitor program for example requires that some target resources are running in order for the monitor program to run.

- Traditional breakpoints and watchpoints are available.
- On-chip resources can be supplemented.
- For example, the ARM Debug Architecture uses an on-chip macro-cell to enhance the debugging facilities available.
- A separate UART to communicate with the monitor program is not required.

The debugging of the target system requires the following:

- A PC host computer running Windows to run the debugger software
- An EmbeddedICE Protocol Converter, a separate box which converts the serial interface to signals compatible with the JTAG interface and a target system with a JTAG interface and an ARM Debug Architecture compliant core.



Once the system is connected, the debugger can start communicating with the target system via the RVI-ME (which is an EmbeddedICE Interface Converter).

The debug extensions consist of several scan chains around the processor core, and some additional signals that are used to control the behavior of the core for debug purposes. The most significant of these additional signals are as follows:

**BREAKPT:** This core signal enables external hardware to halt processor execution for debug purposes. When HIGH during an instruction fetch, the instruction is tagged as breakpointed, and the core stops if this instruction reaches execute.

**DBGRQ:** This core signal is a level-sensitive input that causes the CPU core to enter debug state when the current instruction has completed.

**DBGACK:** This core signal is an output from the CPU core that goes HIGH when the core is in debug state so that external devices can determine the current state of the core.

RealView ICE uses these, and other signals, through the debug interface of the processor core, for example by writing to the control register of the EmbeddedICE logic. For more details, refer to the debug interface section of the ARM datasheet or technical reference manual for your core.

JTAG Debug Interface

The RVI-ME ICE run control unit is supplied with a short ribbon cable. These both terminate in a 20-way 2.54 mm pitch IDC connector. You can use the cable to mate with a keyed box header on the target. The pinout is shown in Figure 16.

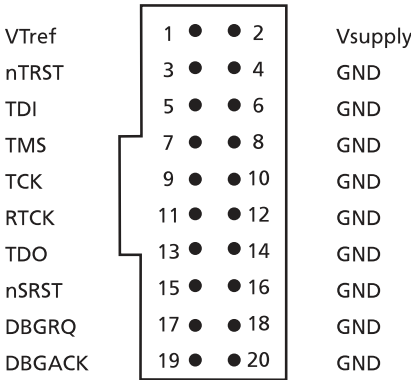


Figure 16 • JTAG Interface Pinout

The signals on the JTAG interface are shown in [Table 7](#).

Table 7 • JTAG Signals

| Signal  | I/O              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DBGACK  | –                | This pin is connected in the RealView ICE run control unit, but is not supported in the current release of the software. It is reserved for compatibility with other equipment to be used as a debug acknowledge signal from the target system. It is recommended that this signal is pulled LOW on the target.                                                                                                                                                                         |
| DBGREQ  | –                | This pin is connected in the RealView ICE run control unit, but is not supported in the current release of the software. It is reserved for compatibility with other equipment to be used as a debug request signal to the target system. This signal is tied LOW. When applicable, RealView ICE uses the core's schanchain 2 to put the core in debug state. It is recommended that this signal is pulled LOW on the target.                                                           |
| GND     | –                | Ground                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| nSRST   | Input/<br>Output | Open collector output from RealView ICE to the target system reset. This is also an input to RealView ICE so that a reset initiated on the target can be reported to the debugger. This pin must be pulled HIGH on the target to avoid unintentional resets when there is no connection.                                                                                                                                                                                                |
| nTRST   | Output           | Open collector output from RealView ICE to the Reset signal on the target JTAG port. This pin must be pulled HIGH on the target to avoid unintentional resets when there is no connection.                                                                                                                                                                                                                                                                                              |
| RTCK    | Input            | Return Test Clock signal from the target JTAG port to RealView ICE. Some targets must synchronize the JTAG inputs to internal clocks. To assist in meeting this requirement, you can use a returned, and retimed, <b>TCK</b> to dynamically control the <b>TCK</b> rate. RealView ICE provides Adaptive Clock Timing that waits for <b>TCK</b> changes to be echoed correctly before making further changes. Targets that do not have to process <b>TCK</b> can simply ground this pin. |
| TCK     | Output           | Test Clock signal from RealView ICE to the target JTAG port. It is recommended that this pin is pulled LOW on the target.                                                                                                                                                                                                                                                                                                                                                               |
| TDI     | Output           | Test Data In signal from RealView ICE to the target JTAG port. It is recommended that this pin is pulled HIGH on the target.                                                                                                                                                                                                                                                                                                                                                            |
| TDO     | Input            | Test Data Out from the target JTAG port to RealView ICE. It is recommended that this pin is pulled HIGH on the target.                                                                                                                                                                                                                                                                                                                                                                  |
| TMS     | Output           | Test Mode signal from RealView ICE to the target JTAG port. This pin must be pulled HIGH on the target so that the effect of any spurious <b>TCKs</b> when there is no connection is benign.                                                                                                                                                                                                                                                                                            |
| Vsupply | Input            | This pin is not connected in the RealView ICE run control unit. It is reserved for compatibility with other equipment to be used as a power feed from the target system.                                                                                                                                                                                                                                                                                                                |
| VTref   | Input            | This is the target reference voltage. It indicates that the target has power, and it must be at least 0.628 V. <b>VTref</b> is normally fed from <b>Vdd</b> on the target hardware and might have a series resistor (though this is not recommended). There is a 10 k pull-down resistor on <b>VTref</b> in RealView ICE.                                                                                                                                                               |

The EmbeddedICE logic which implements the on-chip debug function in the CoreMP7 debug architecture is described in detail in the [ARM7TDMI-S \(rev 4\) Technical Reference Manual](#) (ARM DDI0234A), published by ARM Limited, and is available via Internet at [www.arm.com](http://www.arm.com).

The CoreMP7 debug architecture uses a JTAG port as a method of accessing the core. The debug architecture uses EmbeddedICE logic which resides on chip with the CoreMP7 core. The EmbeddedICE has its own scan chain that is used to insert watchpoints and breakpoints for the CoreMP7. The EmbeddedICE logic consists of two real-time watchpoint registers, together with a control and status register. One or both of the watchpoint registers can be programmed to halt the CoreMP7 core. Execution is halted when a match occurs between the values programmed into the EmbeddedICE logic and the values currently appearing on the address bus, databus, and some control signals. Any bit can be masked so that its value does not affect the comparison. Either watchpoint register can be configured as a watchpoint (i.e., on a data access) or a break point (i.e., on an instruction fetch). The watchpoints and breakpoints can be combined such that:

- The conditions on both watchpoints must be satisfied before the CoreMP7 is stopped. The CHAIN functionality requires two consecutive conditions to be satisfied before the core is halted.

An example of this would be to set the first breakpoint to trigger on an access to a peripheral and the second to trigger on the code segment that performs the task switching. Therefore the breakpoints trigger the information regarding which task has switched out that will be ready for examination.

- The watchpoints can be configured such that a range of addresses are enabled for the watchpoints to be active. The RANGE function allows the breakpoints to be combined such that a breakpoint is to occur if an access occurs in the bottom 256 bytes of memory but not in the bottom 32 bytes.

The CoreMP7 core has a Debug Communication Channel function in-built. The debug communication channel allows a program running on the target to communicate with the host debugger or another separate host without stopping the program flow or even entering the debug state. The debug communication channel is accessed as coprocessor 14 by the program running on the CoreMP7 core. The debug communication channel allows the JTAG port to be used for sending and receiving data without affecting the normal program flow. The debug communication channel data and control registers are mapped in to addresses in the EmbeddedICE logic.

Table 8 • Debug Communication Channel Signals

| Signal Name | Type   | Description                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TMS         | Input  | <b>Test Mode Select.</b> The TMS pin selects the next state in the TAP state machine.                                                                                                                                                                                                                                                                                                         |
| TCK         | Input  | <b>Test Clock.</b> This allows shifting of the data in, on the TMS and TDI pins. It is a positive edge triggered clock with the TMS and TCK signals that define the internal state of the device.                                                                                                                                                                                             |
| TDI         | Input  | <b>Test Data In.</b> This is the serial data input for the shift register.                                                                                                                                                                                                                                                                                                                    |
| TDO         | Output | <b>Test Data Output.</b> This is the serial data output from the shift register. Data is shifted out of the device on the negative edge of the TCK signal.                                                                                                                                                                                                                                    |
| nTRST       | Input  | <b>Test Reset.</b> The nTRST pin can be used to reset the test logic within the EmbeddedICE logic.                                                                                                                                                                                                                                                                                            |
| RTCK        | Output | <b>Returned Test Clock.</b> Extra signal added to the JTAG port. Required for designs based on COREMP7 processor core. Multi-ICE (development system from ARM) uses this signal to maintain synchronization with targets having slow or widely varying clock frequency. For details, refer to the <a href="#">Multi-ICE System Design Considerations Application Note 72</a> (ARM DAI 0072A). |

The EmbeddedICE logic contains 16 registers, as shown in [Table 9](#). The CoreMP7 debug architecture is described in detail in [ARM7TDMI-S \(rev 4\) Technical Reference Manual](#) (ARM DDI0234A), published by ARM Limited, and is available via Internet at [www.arm.com](http://www.arm.com).

Table 9 • **EmbeddedICE Logic Registers**

| <b>Name</b>                  | <b>Width</b> | <b>Description</b>                    | <b>Address</b> |
|------------------------------|--------------|---------------------------------------|----------------|
| Debug Control                | 6            | Force debug state, disable interrupts | 00000          |
| Debug Status                 | 5            | Status of debug                       | 00001          |
| Debug Comms Control Register | 32           | Debug communication control register  | 00100          |
| Debug Comms Data Register    | 32           | Debug communication data register     | 00101          |
| Watchpoint 0 Address Value   | 32           | Holds watchpoint 0 address value      | 01000          |
| Watchpoint 0 Address Mask    | 32           | Holds watchpoint 0 address mask       | 01001          |
| Watchpoint 0 Data Value      | 32           | Holds watchpoint 0 data value         | 01010          |
| Watchpoint 0 Data Mask       | 32           | Holds watchpoint 0 data mask          | 01011          |
| Watchpoint 0 Control Value   | 9            | Holds watchpoint 0 control value      | 01100          |
| Watchpoint 0 Control Mask    | 8            | Holds watchpoint 0 control mask       | 01101          |
| Watchpoint 1 Address Value   | 32           | Holds watchpoint 1 address value      | 10000          |
| Watchpoint 1 Address Mask    | 32           | Holds watchpoint 1 address mask       | 10001          |
| Watchpoint 1 Data Value      | 32           | Holds watchpoint 1 data value         | 10010          |
| Watchpoint 1 Data Mask       | 32           | Holds watchpoint 1 data mask          | 10011          |
| Watchpoint 1 Control Value   | 9            | Holds watchpoint 1 control value      | 10100          |
| Watchpoint 1 Control Mask    | 8            | Holds watchpoint 1 control mask       | 10101          |

## Ordering Information

All variants of the CoreMP7 soft IP core are included in the CoreConsole IDP. To use CoreMP7, you need to download CoreConsole, which is available for free at:

<http://www.actel.com/custsup/updates/coreconsole/>.

You can also request that a CoreConsole CD (which includes CoreMP7) be mailed to you.

## List of Changes

The following table lists critical changes that were made in the current version of the document.

| Previous Version | Changes in Current Version (v2.4)                    | Page |
|------------------|------------------------------------------------------|------|
| v2.3             | Table 1 was updated with AFS600 information.         | 2    |
|                  | The "Bus Functional Model" section was updated.      | 14   |
| v2.2             | The datasheet was updated to include Fusion devices. | NA   |
|                  | Table 1 was updated.                                 | 2    |
|                  | The "CoreMP7 Variants" section was updated.          | 13   |
| v2.1             | Table 1 was updated.                                 | 2    |
| v2.0             | The "No Coprocessor Interface" section was updated.  | 13   |
|                  | The "Little Endian Only" section was updated.        | 13   |

## Datasheet Categories

In order to provide the latest information to designers, some datasheets are published before data has been fully characterized. Datasheets are designated as "Product Brief," "Advanced," and "Production." The definitions of these categories are as follows:

### Product Brief

The product brief is a summarized version of an advanced or production datasheet containing general product information. This brief summarizes specific device and family information for unreleased products.

### Advanced

This datasheet version contains initial estimated information based on simulation, other products, devices, or speed grades. This information can be used as estimates, but not for production.

### Unmarked (production)

This datasheet version contains information that is considered to be final.

Actel and the Actel logo are registered trademarks of Actel Corporation.  
All other trademarks are the property of their owners.



[www.actel.com](http://www.actel.com)

**Actel Corporation**

2061 Stierlin Court  
Mountain View, CA  
94043-4655 USA

**Phone** 650.318.4200  
**Fax** 650.318.4600

**Actel Europe Ltd.**

Dunlop House, Riverside Way  
Camberley, Surrey GU15 3YL  
United Kingdom

**Phone** +44 (0) 1276 401 450  
**Fax** +44 (0) 1276 401 490

**Actel Japan**

[www.jp.actel.com](http://www.jp.actel.com)

EXOS Ebisu Bldg. 4F  
1-24-14 Ebisu Shibuya-ku  
Tokyo 150 Japan

**Phone** +81.03.3445.7671  
**Fax** +81.03.3445.7668

**Actel Hong Kong**

[www.actel.com.cn](http://www.actel.com.cn)

Suite 2114, Two Pacific Place  
88 Queensway, Admiralty  
Hong Kong

**Phone** +852 2185 6460  
**Fax** +852 2185 6488

## **Appendix 6:     Passive measurement infrastructure**

# A Distributed Passive Measurement Infrastructure

Patrik Arlos, Markus Fiedler, and Arne A. Nilsson

Blekinge Institute of Technology, School of Engineering,  
Karlskrona, Sweden  
`{patrik.arlos,markus.fiedler,arne.nilsson}@bth.se`

**Abstract.** In this paper we describe a distributed passive measurement infrastructure. Its goals are to reduce the cost and configuration effort per measurement. The infrastructure is scalable with regards to link speeds and measurement locations. A prototype is currently deployed at our university and a demo is online at <http://inga.its.bth.se/projects/dpmi>. The infrastructure differentiates between measurements and the analysis of measurements, this way the actual measurement equipment can focus on the practical issues of packet measurements. By using a modular approach the infrastructure can handle many different capturing devices. The infrastructure can also deal with the security and privacy aspects that might arise during measurements.

## 1 Introduction

Having access to relevant and up-to-date measurement data is a key issue for network analysis in order to allow for efficient Internet performance monitoring, evaluation and management. New applications keep appearing; user and protocol behaviour keep evolving; traffic mixes and characteristics are continuously changing, which implies that traffic traces may have a short span of relevance and new traces have to be collected quite regularly.

In order to give a holistic view of what is going on in the network, passive measurements have to be carried out at different places simultaneously. On this background, this paper proposes a passive measurement infrastructure, consisting of coordinated measurement points, arranged in measurement areas.

This structure allows for a efficient use of passive monitoring equipment in order to supply researchers and network managers with up-to-date and relevant data. The infrastructure is generic with regards to the capturing equipment, ranging from simple PCAP-based devices to high-end DAG cards and dedicated ASICs, in order to promote a large-scale deployment of measurement points.

The infrastructure, which currently is under deployment at our university, was designed with the following requirements in mind:

1. *Cost.* Access to measurement equipment should be shared among users, primarily for two reasons: First, as measurements get longer (for instance for detecting long-range dependent behaviour) a single measurement can tie



up a resource for days (possibly weeks). Second, high quality measurement equipment is expensive and should hence have a high rate of utilization.

2. *Ease of use.* The setup and control of measurements should be easy from the user's point of view. As the complexity of measurements grows, we should hide this complexity from the users as far as possible.
3. *Modularity.* The system should be modular, this to allow independent development of separate modules. With separate modules handling security, privacy and scalability (w.r.t. different link speeds as well as locations). Since we cannot predict all possible uses of the system, the system should be flexible to support different measurements as well as different measurement equipment.
4. *Safety and Security.* Measurement data should be distributed in a safe and secure manner, i.e. loss of measurement data should be avoided and access to the data restricted.

To solve these requirements we came up with an infrastructure consisting of three main components, *Measurement Point* (MP), *Consumer* and *Measurement Area* (MAr). The task of the MP is to do packet capturing, packet filtering, and distribute measurement data. The approach to the second design requirement was to use a system with a web interface. Through this interface users can add and remove their desired measurements. The MAr then handles the communication with the MPs. The cost for implementing this architecture is not very high, compared to a normal measurement setup you need two additional computers and an Ethernet switch of suitable speed, and this basic setup can grow as the requirements change.

There are several other monitoring and capturing systems available, here we describe only a few.

CoralReef [1] is a set of software components for passive network monitoring, it is available for many network technologies and computer architectures. The major difference between CoralReef and our infrastructure is that CoralReef does not separate the packet capturing and analysis as we do. Furthermore, the CoralReef trace format does not include location information as our does.

IPMON [2] is a general purpose measurement system for IP networks. IPMON is implemented and deployed by Sprint. IPMON separates capturing from analysis, similar to our infrastructure. On the other hand, the IPMONs store traces locally and transfer them over a dedicated link to a common data repository. The repository is then accessed by analyzers.

Gigascope [3] uses a similar approach as IPMON, by storing captured data locally at the capturer. This data is then copied, either in real time or during off-peak hour, to a data warehouse for analysis. It uses GSQL as an interface to access the data.

The IETF has (at least) two work groups that are relevant for this work; Packet Sampling (PSAMP) [4] and IP Flow Information Export (IPFIX) [5]. PSAMP works on defining a standard set of capabilities for network elements to sample subsets of packets by statistical and other methods. Recently an Internet draft was published [6], which describes a system at a higher level than our in-

infrastructure, but they are very similar and our system could benefit by adjusting somewhat to the PSAMP notation. The IPFIX group is interesting since they deal with how to export measurement data from A to B, thus it is interesting with regards to consumers.

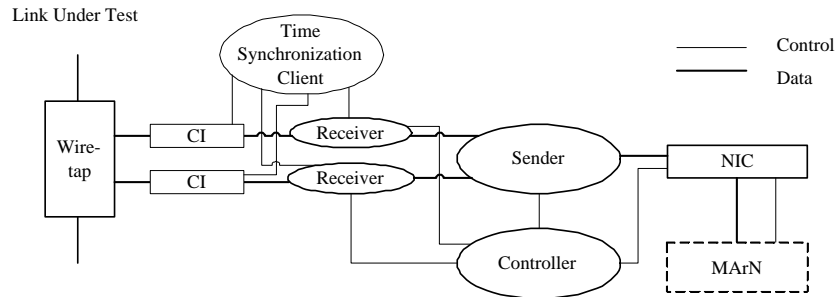
In Section 2 we will discuss the components and how they interact. This is followed by Section 3 where we describe how the system handles rules and filters. In Section 4 we discuss privacy and security related to the infrastructure. In Section 5 we describe two cases where the system has been deployed. In Section 6 we describe some of the ongoing and future work. And in Section 7 we conclude the paper.

## 2 Components

The three main components in the infrastructure will be described in the following subsections.

### 2.1 Measurement Point

In Figure 1 the components of a schematic MP are shown. This is the device that does the actual packet capturing. It is managed from a Measurement Area Controller (MArC) and transfers the captured data to consumers attached to the Measurement Area Network (MArN). The MP can either be a logical or a physical device. A logical MP is simply a program running on a host, whereas a physical MP could either use a dedicated computer or custom hardware in order to create high-speed high-performance MPs.



**Fig. 1.** Schematic overview of a MP.

A MP can tap one or more links; each link is tapped via a wiretap. For full-duplex Ethernets, a wiretap has two outputs, one for each direction. These are connected to separate capture interfaces (CI). A receiver listens to a CI and filters the packets according to the filter rules stated by the MArC. If the CI

hasn't timestamped the packet the receiver will do so. The packets are then delivered to the sender, which is responsible for sending the captured packets to the appropriate consumers. Such a measurement frame can contain several packets, where the number of packets is controlled by the maximum transfer unit (MTU) of the MArN. Each MP also has a controller that is responsible for the configuration of the MP and the communication with the MArC. A time synchronization client (TSC) is used to keep all the MPs with in a MAr synchronized, which can be done using a dedicated device or a simple NTP server.

The filter rules used by the receiver specify, in addition to packet properties, a consumer and the amount of the packet to be captured (currently the upper limit is 96 bytes). For each frame that passes the filter, the MP attaches a capture header (Figure 2). In this header, we store a CI identifier, a MP identifier, a timestamp when the packet was captured (supporting an accuracy of picoseconds), the packet length, and the number of bytes that actually were captured. The filters are supplied to the MP from the MArC, and they will be discussed in Section 3. Once a packet matches a filter, it is stored in a buffer pending transmission. Once the buffer contents reaches a certain threshold the buffer is transmitted using Ethernet multicast. This way, it is simple to distribute frames to several consumers in one transmission. The duplication of data is done by the MArN. This approach will also reduce the probability of overloading the MArN, and hence preventing loss of measurement frames as far as possible. However, in order to detect frame loss each measurement frame is equipped with a sequence number that is checked by the consumer upon reception. If a measurement frame is lost it is up to the consumer to handle this particular loss and notify the MArC. Given this information the MArC can take actions to prevent future losses. Actions can be to alter filters as well as requesting additional switching resources inbetween the MPs and the Consumers. The current implementation only notifies the consumer "user", who has to take appropriate actions.

|        |        |
|--------|--------|
| CI     |        |
| MAMPid |        |
| Time   |        |
| Time   | Length |
| CapLen |        |

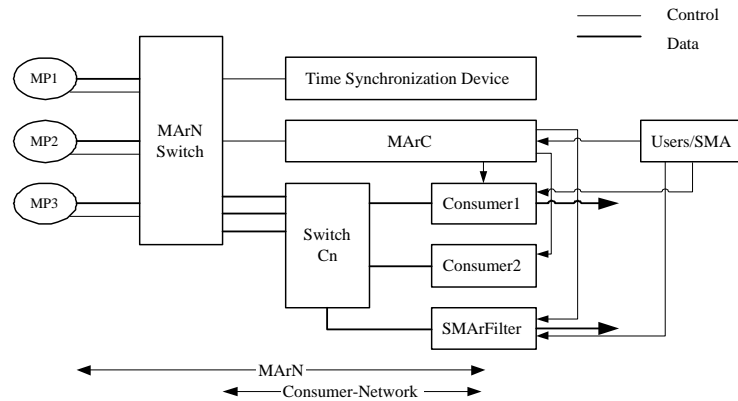
**Fig. 2.** Capture Header.

The capture header enables us to exactly pinpoint by which MP and on what link the frame was captured, which is vital information when trying to obtain spatial information about the network's behaviour. This also enables us to use several MPs to measure a single link, which is interesting when the measurement

task of a link speed becomes too great for a single MP to handle. This would require a device that is capable of distributing the packets such that the wiretap feeds different MPs in a round robin approach.

## 2.2 Measurement Area

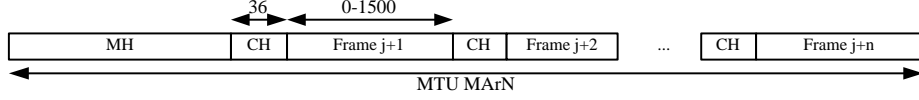
In Figure 3 an example of a MA is shown. The MA provides a common point of control for one or more MPs. It uses a dedicated network in between MPs and the MA subsystems for reasons of performance and security. A MA consists of the following subsystems: a MArc, a time synchronization device (TSD), a MArN and at least one consumer and one MP. The MArc is the central subsystem in a MA. It supplies the users with a GUI for setting up and controlling their measurements. It also manages the MPs by supplying filters and by keeping track of their status. The TSD supplies all the MPs in the MA with a common time and synchronization signal. It can utilize the existing Ethernet structure to the MPs, or it can utilize some other network to distribute the time signal.



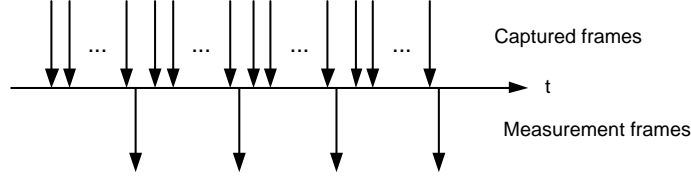
**Fig. 3.** Simple overview of a MA with three MPs, four consumers, one MArc and a time synchronization unit.

The capacity of the MArN should be such that it can handle the peak rate of the measured traffic. Assume that a MP monitors a 10Base-T link, with a frame rate of 800 fps where each frame is 1500 bytes long ( $\approx 9.6$  Mbps). From each frame we collect 96 bytes, add a capture header of 36 bytes and store the data in a measurement frame, see Figure 4. Given a MArN MTU of 1500, a measurement frame can contain 1480 bytes of measurement data, consisting of capture headers and frames, the remaining 20 bytes are used by a measurement header (MH). In the current example we can store 11 frames in each measurement frame ( $11 * (36 + 96) = 1452 \leq 1480$  bytes), causing the MP to send only  $800/11 \approx 72$  fps into the MArN, see Figure 5. If the monitored link would have a

frame rate of 14000 fps, each frame would only be 85 bytes long ( $\approx 9.6$  Mbps), the measurement frame would contain 12 frames ( $12 \times (36 + 85) = 1452 \leq 1480$  bytes), yielding a frame rate of  $14000/12 \approx 1167$  fps. However, if the MArN MTU was 9000, the measurement frame could contain 74 frames, yielding a frame rate of 189 fps.



**Fig. 4.** Measurement frame encapsulation.

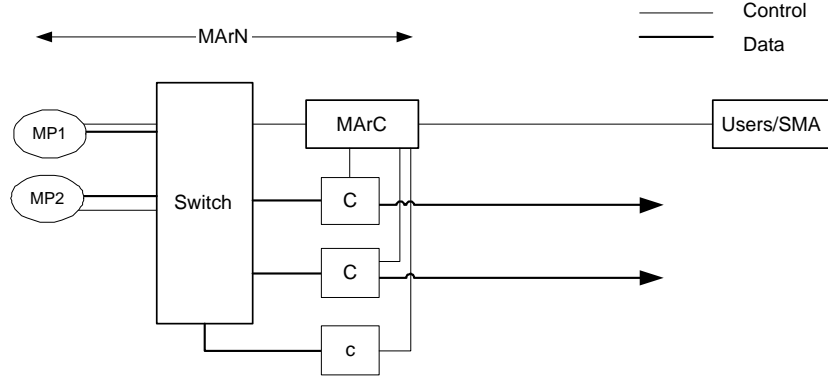


**Fig. 5.** After capturing  $N$  frames one measurement frame is sent from the MP.

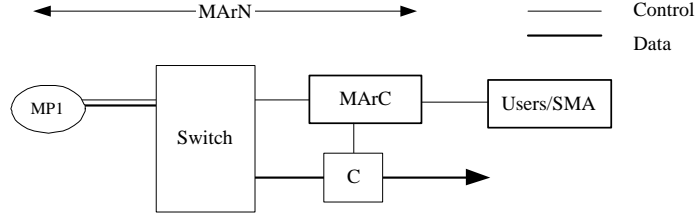
A consumer that attaches to the MArN should not request more data than the link that it is attached to can handle. For instance a consumer C1 is the recipient of two measurement streams, S1 and S2, each generating 1272 measurement frames per second. As long as the total frame rate of S1 and S2 is less or equal to the capacity offered by link and switch there should be no problems, but if the consumer desires to get full frames it might run into problems quite fast, since the MP adds a capture header to each captured frame potentially generating more traffic than it captures. The current implementation addresses this problem by having a maximum capture size of 96 bytes. The MArC also provides the user with an estimation of the frame rate on the links that the MPs are monitoring, giving the user an indication of the amount of traffic that his consumer might receive.

The example in Figure 3 contains a consumer network (CN). It is placed on a separate switch to minimize processing required by the MArN, thus enabling additional consumers to be easily connected to the MArN, for instance new probes, analyzers etc. to be evaluated in parallel. If the number of consumers is low, the MArN switch might handle them directly, and no CN switch is necessary. This would be the normal setup, see Figure 6. In Figure 7 a minimal MAr is

shown. In both cases the MPs are using a separate network for the time signal distribution.



**Fig. 6.** Normal MAr



**Fig. 7.** Minimal MAr

### 2.3 Consumer

A consumer is a user-controlled device that accepts packets according to the format specified by the system. A consumer should filter the content of the measurement frame that it receives, since the MP merges multiple user requests some filters will capture packets that match several requests. Such a joint filter might not perfectly match the desired frame description; this is discussed in the following section.

### 3 Filters and Rules

A user supplies rules to the MArC. These rules describe *what* data the user desires to collect, *where* the data should be collected, *when* the data should be collected and *where to send* the data. The MArC uses this information to create filters that the MPs understand. The filters that the MP uses are a combination of all the user supplied rules, combined in such a manner that all requests are met in a best effort style. The MArC keeps track of the MPs and their capabilities, thus it knows how many filters a MP can handle before it runs into performance problems. The MArC also monitors the performance of the MArN and reject user rules that could cause performance problems within the MArN. If a MP is to obtain a filter list that would push it into a region of potential performance problems, the MArC will alter the filters in order to minimize the number of filters. By doing this the load on the MP is kept at a reasonable level, but this approach requires the consumers to do some filtering of their own. Hence, it is up to the user to supply the desired Consumer with a filter. The filters within a MP are arranged in such a manner that no packet is reported twice by the MP.

Let's give a simple example, we have one MP and two consumers C1 and C2. Initially we have two rules (using BPF syntax):

R1  $\{tcp\ host\ A.a\}$  which sends its data to C1.

R2  $\{ip\ net\ A\}$  which targets C2

Here two approaches are possible; the first during low load would have the following filters sent to the MP:

F1  $\{tcp\ host\ A.a\} \rightarrow M1$

F2  $\{ip\ net\ A\} \rightarrow C2$

Here M1 is a multicast address that C1 and C2 listens to. If the load on the MP approaches a high level then only one filter would be sent to the MP

F1  $\{ip\ net\ A\} \rightarrow M1$

In this case the C1 consumer would need to perform filtering in order to select the TCP segments of host A.a. By default a consumer should always filter the measurement data that it receives, ensuring that it passes a correct stream to the analysis/storage entity.

### 4 Privacy & Security Issues

A MP will see all the traffic passing on a link that it is tapping, which can be viewed as a intrusion of privacy. Furthermore, since the majority of the network protocols used today were not designed with security in mind, user credentials might pass on the link and be clearly visible to the MP. This can be an intrusion of privacy and should require special care on behalf of the measurement system and its users. If the data collected from the system is only intended for internal

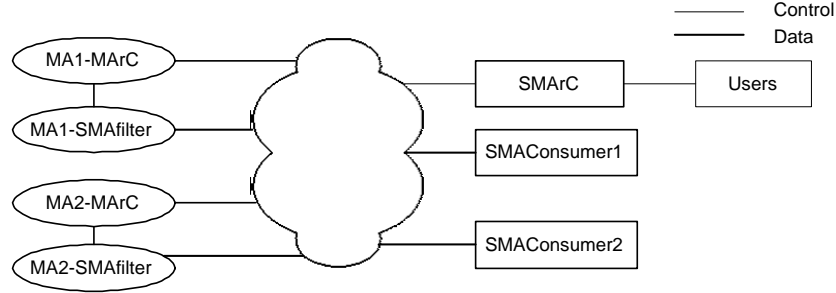
use, it might be enough that all users and the network-owner have agreed to that their traffic can be monitored to allow for measurements. However, if the data is to be shared with researchers in other organizations, the data should be deprivatized. Deprivatization [7] can be done on various levels, from the removal of parts in the application data to the removal of all network data. We believe that the system should minimize the alternation of the captured data and leave the anonymization to the consumers. If the MP would anonymize the data, e.g. through scrambling of addresses [8], some consumers such as intrusion detection systems or charging systems might not be able to operate anymore. However, if the system does deprivatization by default, this should be done in the MPs. If address scrambling is utilized, this causes problems when the user specifies the measurement rules. If the unscrambled address was used, the user will obtain scrambled addresses matching his requirement and then it is possible to reverse-engineer the scrambling system. If the scrambled address was used, the user would need to know how to create that scrambled address. Probably, the first method should be chosen. In that case, the only person that is capable of reverse-engineering the packet trace is the user requesting the trace, since he knows both scrambled and unscrambled address. Now, if the packet trace is stolen, the thief cannot match packets to individual hosts/users unless he has access to a descrambler and the scrambling key.

Privacy issues will probably have to be addressed by specialized consumers. For instance, we have two consumers, a intrusion detection system (IDS) and a link utilization estimator (LUE). The IDS needs undistorted information. The LUE could on the other hand use deprivatized data, but since the MP will not send two copies of the same packet there is a problem. It is probable that a network owner would like to have control of the information that leaves his network, so it would be easier for the network owner to supply an export consumer that deprivatizes the data according to his own policies, which might not meet the particular desires of the user. For our own measurements, the agreement we made with the system owner was the following: The MPs are only allowed to capture headers, not user payload. Furthermore, the data leaving a consumer may only be in statistical form, or deprivatized in such a manner that it is impossible to reverse-engineer the data to obtain information that allows you to identify a particular individual.

From a security point of view, all components in the system should be protected from unauthorized access. The simplest way to do this is to have the system operating on a separate network, with no connection to any other networks. This would however be expensive and unpractical in measurements distributed over a wide area. The solution to this is to utilize Super Measurement Areas (SMAr), see Figure 8. SMAr's are used to connect to MAr's at different locations using existing infrastructure. A SMAr can be seen as a MAr at a higher level, the MAr's MP becomes SMArFilters (specialized consumers that attach to the MArN), the MAr's consumers are called SMArConsumers. Between the SMArFilters and SMArConsumers TCP is used to provide reliable communication. The MPs and the MArN need to be protected from unauthorized access,



both physical and logically. Physical protection of the MAr subsystems is the first requirement in giving logical protection; the consumers and the MArC need to be protected from intrusions via their connection to the users.



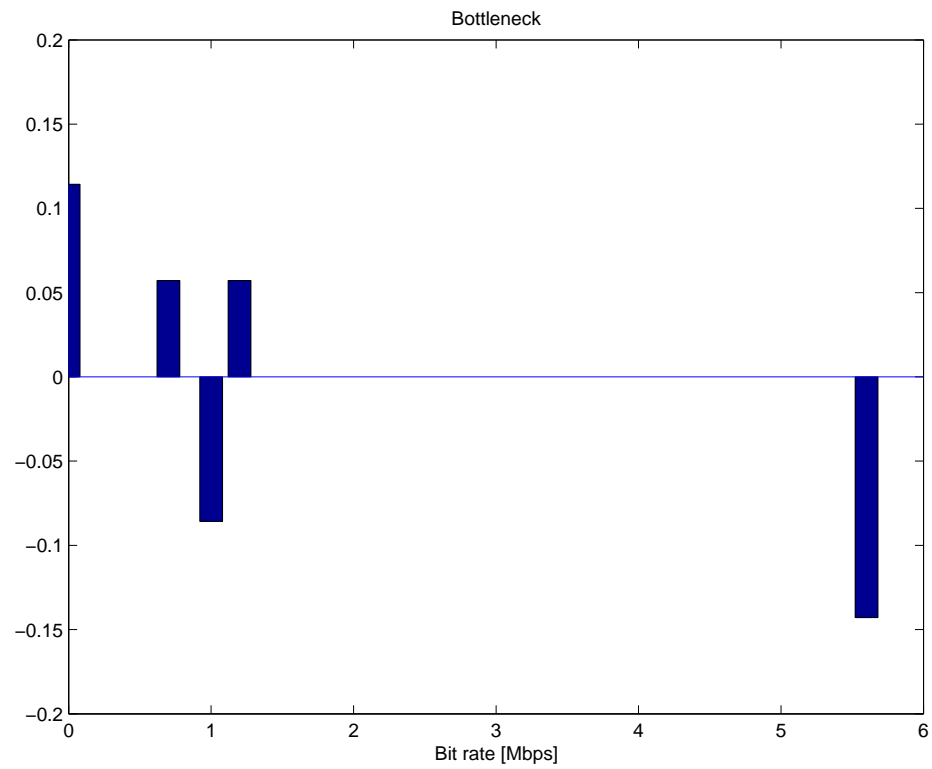
**Fig. 8.** Example of a SMAR.

## 5 Examples of Use

As of writing two MAr have been implemented and used. One is available online via <http://inga.its.bth.se/projects/dpmi> and is mainly used in a controlled environment. The second MAr consisted of two measurement points each monitoring a gigabit link on a campus network. In both cases only one physical consumer was used, but it was sufficient to handle up to eight logical consumers. Examples of consumers are: estimation of traffic distribution (at link, network, transport and application level); link utilization; packet inter arrival time; communication identification; and bottleneck identification [9]. At the time of writing we are preparing a third MAr to be deployed in an ISP network, where it will initially be used for bottleneck identification. In Figure 9 we visualize the result from a analyzer that identifies bottlenecks. It uses two consumers to estimate the link bit rate over a given time intervall, these are then transferred to a database which is accessed by the visualizer that estimates the bottleneck.

In Figure 10 the MArC (prototype) interface for adding a rule is shown. In this implementation all tasks are done manually, the goal was to develop the MP not the MArC. The following filtering options are available, the MASK fields are used to mask the packet value.

- CI: Physical interface identifier.
- VLAN\_TCI: VLAN number and priority.
- ETH.TYPE: Ethernet type.
- ETH.SRC/DST: Ethernet source/destination address.
- IP.PROTO: IP payload type.
- IP.SRC/DST: IP source/destination address.



**Fig. 9.** Example of a consumer: Visualization of a bottleneck through bitrate histogram difference plots (c.f. [9]).

- SRC/DST.PORT: Transport protocol source/destination port numbers (if applicable).
- DESTADDR: What Ethernet address should receive the measurement data?
- TYPE: Which type of transport should the MP use? Ethernet, UDP or TCP.
- CAPLEN: How much of each captured frame should we store?

FilterID is a number that specifies in which order the MP should check its filters, starting with number zero. Index will indicate which fields that are used in the rule specification. For instance if we wish to collect all packets caught on a specific CI the index would be 512, and the CI field would hold a string identifying the CI. If we would like to capture IP packets caught on a specific CI, index would be 640, ETH.TYPE=2048 and CI a string specifying the interface.

**Add Filter - Mozilla Firefox**

Filter Specification (DO NOT EDIT INDEX!!!)(If form handles it for you, else you do the math..)

INDEX:

FILTER ID:

**Packet Specification**

|                              |          |                                           |               |                                                                                                                 |
|------------------------------|----------|-------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------|
| <input type="checkbox"/> 512 | CI       | <input type="text" value="null"/>         |               |                                                                                                                 |
| <input type="checkbox"/> 256 | VLAN_TCI | <input type="text" value="0"/>            | VLAN_TCI_MASK | <input type="text" value="0"/> Other ▾                                                                          |
| <input type="checkbox"/> 128 | ETH_TYPE | <input type="text" value="0"/>            | ETH_TYPE_MASK | <input type="text" value="0"/> Other ▾                                                                          |
| <input type="checkbox"/> 64  | ETH_SRC  | <input type="text" value="000000000000"/> | ETH_SRC_MASK  | <input type="text" value="000000000000"/> Other ▾                                                               |
| <input type="checkbox"/> 32  | ETH_DST  | <input type="text" value="000000000000"/> | ETH_DST_MASK  | <input type="text" value="000000000000"/> Other ▾                                                               |
| <input type="checkbox"/> 16  | IP_PROTO | <input type="text" value="0"/>            | Other ▾       |                                                                                                                 |
| <input type="checkbox"/> 8   | IP_SRC   | <input type="text" value="0"/>            | IP_SRC_MASK   | <input type="text" value="0"/> Other ▾                                                                          |
| <input type="checkbox"/> 4   | IP_DST   | <input type="text" value="0"/>            | IP_DST_MASK   | <input type="text" value="0"/> Other ▾                                                                          |
| <input type="checkbox"/> 2   | SRC_PORT | <input type="text" value="0"/>            | SRC_PORT_MASK | <input type="text" value="0"/> Other ▾                                                                          |
| <input type="checkbox"/> 1   | DST_PORT | <input type="text" value="0"/>            | DST_PORT_MASK | <input type="text" value="0"/> Other ▾                                                                          |
|                              | DESTADDR | <input type="text" value="010000000000"/> | TYPE          | <input type="text" value="1"/> Ethernet Multicast ▾<br><small>Note: TCP requires a running TCP consumer</small> |
|                              | CAPLEN   | <input type="text" value="54"/>           |               |                                                                                                                 |

**MP Receiving Filter**

|                                  | Name | Comment | Max filters |
|----------------------------------|------|---------|-------------|
| <input checked="" type="radio"/> | mp06 |         | 20          |
| <input type="radio"/>            | mp05 |         | 20          |

Find:

Fig. 10. User interface for adding rules.

## 6 Ongoing and Future Work

Initial experiences with the system are encouraging, and development of consumers is currently ongoing. The experience of the demo has indicated that the MP's software needs to be changed in such a manner that the MPs periodically flush their measurement buffers, in order to prevent consumers from waiting long times. We are considering a modification of the system so that the MArC supplies the consumers automatically with the information that they need with regards to filters and multicast addresses.

To handle the increased link speeds, new devices with better timestamping accuracy are needed. Even if we can obtain this accuracy, a single device will probably run into problems when measuring such a link. Hence another task would be to investigate how to distribute the measurement task of a link onto several MPs. Compression of frame data is also considered to be implemented, this would enable us to do full frame capturing without requiring a MArN that is more powerful than the observed link. We also need to evaluate the performance of a MArN.

The infrastructure is being considered as a part of the EuroNGI WP.JRA.4.3 [10] Measurement tool. This tool will support traffic generation, measurement, analysis and visualization.

## 7 Conclusions

In this paper we have presented a distributed passive measurement infrastructure, which has separate components for packet capturing, control and analysis. We discussed how the system deals with multiple users and their request for data. Since the infrastructure is passive we addressed the security and privacy issues associated with this. Furthermore, we gave examples of current usage and future work.

## References

1. CAIDA: CoralReef. (2005) <http://www.caida.org/tools/measurement/coralreef> (Verified in January 2005).
2. Sprint: IPMON (2005) <http://ipmon.sprint.com> (Verified in January 2005).
3. AT&T: Gigascope (2005) <http://www.research.att.com/info/Projects/Gigascope> (Verified in January 2005).
4. IETF: PSAMP Workgroup. (2005) <http://www.ietf.org/html.charters/psamp-charter.html> (Verified in January 2005).
5. IETF: IPFIX Workgroup. (2005) <http://www.ietf.org/html.charters/ipfix-charter.html> (Verified in January 2005).
6. IETF: A Framework for Packet Selection and Reporting. (2005) <http://www.ietf.org/internet-drafts/draft-ietf-psamp-framework-10.txt> (Verified in January 2005).

7. Pang, R., Paxson, V.: A high-level programming environment for packet trace anonymization and transformation. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, ACM Press (2003) 339–351
8. Xu, J., Fan, J., Ammar, M., Moon, S.B.: On the design and performance of prefix-preserving ip traffic trace anonymization. In: IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, ACM Press (2001) 263–266
9. Fiedler, M., Tutschku, K., Carlsson, P., Nilsson, A.A.: Identification of performance degradation in ip networks using throughput statistic. In: Proceedings of the 18th international Teletraffic Congress (ITC-18), ELSEVIER (2003) 399–408
10. EuroNGI: Homepage (2005) <http://www.eurongi.org> (Verified in January 2005).
11. TCPDUMP Public Repository: Homepage. (2005) <http://www.tcpdump.org> (Verified in January 2005).
12. Endace Measurement Systems: Homepage. (2005) <http://www.endace.com> (Verified in January 2005).

## **Appendix 7: SimAP Design report**

*All information contained in this document is the property of the Hes-so.  
Its contents cannot be reproduced or divulged without the school's approval.*

**PROJECT** : MAC Ethernet for specific application  
(MAC+)

**DOCUMENT TITLE** : IO-MAC design report

**DOCUMENT NUMBER** : SPAM-RPT-006-HEV

**ISSUE/ REVISION** : 2/-

|             | NAME         | FUNCTION          | SIGNATURE | DATE          |
|-------------|--------------|-------------------|-----------|---------------|
| PREPARED BY | F. SEBASTIEN | Project Engineer  |           | October, 2007 |
| CHECKED BY  | F. CORTHAY   | Technical Manager |           | October, 2007 |

## D I S T R I B U T I O N L I S T

| COMPANY / NAME  | QUANTITY |
|-----------------|----------|
| <b>HES-SO :</b> | 0        |
| F. Sébastien    | 0        |
| M. Clausen      | 0        |
| F. Corthay      | 0        |

## C H A N G E R E C O R D

| ISSUE / REVISION | DATE         | PAGES | MODIFICATION   |
|------------------|--------------|-------|----------------|
| 1/-              | July 2007    | All   | First Edition  |
| 2/-              | October 2007 | All   | Second Edition |



# CONTENTS

|                                       | <u>PAGE</u> |
|---------------------------------------|-------------|
| <b>CONTENTS.....</b>                  | <b>II</b>   |
| <b>PAGE.....</b>                      | <b>II</b>   |
| <b>1 INTRODUCTION.....</b>            | <b>5</b>    |
| <b>2 WORK.....</b>                    | <b>6</b>    |
| 2.1 OBJECTIVES.....                   | 6           |
| 2.2 PARSING.....                      | 6           |
| <b>3 THE USED FRAME.....</b>          | <b>7</b>    |
| 3.1 GLOBALITY .....                   | 7           |
| 3.2 ETHERNET FIELD .....              | 7           |
| 3.3 SIMAP FIELD .....                 | 7           |
| 3.4 OTHERS PARTS .....                | 8           |
| <b>4 GLOBAL FUNCTIONING.....</b>      | <b>9</b>    |
| <b>5 CONTENT OF DESIGN.....</b>       | <b>10</b>   |
| <b>6 INTERFACE ETHERNET .....</b>     | <b>11</b>   |
| 6.1 FUNCTIONALITY .....               | 11          |
| 6.2 CONTENT SUMMARY .....             | 11          |
| 6.3 CONTENT DESCRIPTION .....         | 12          |
| 6.3.1 receiver.....                   | 12          |
| 6.3.1.1 Functionality.....            | 12          |
| 6.3.1.2 Content summary.....          | 12          |
| 6.3.1.3 Content description .....     | 13          |
| 6.3.1.3.1 data to word.....           | 13          |
| 6.3.1.3.2 crc32 .....                 | 13          |
| 6.3.1.3.3 synchronizer .....          | 14          |
| 6.3.1.3.4 reset synch.....            | 14          |
| 6.3.1.3.5 ram dp.....                 | 14          |
| 6.3.1.3.6 mux ram data.....           | 14          |
| 6.3.1.3.7 receiver controller.....    | 15          |
| 6.3.2 transmitter.....                | 17          |
| 6.3.2.1 Functionality.....            | 17          |
| 6.3.2.2 Content summary.....          | 17          |
| 6.3.2.3 Content description .....     | 18          |
| 6.3.2.3.1 byte to nibble.....         | 18          |
| 6.3.2.3.2 crc32 .....                 | 18          |
| 6.3.2.3.3 synchronizer .....          | 18          |
| 6.3.2.3.4 reset synch.....            | 18          |
| 6.3.2.3.5 ram dp.....                 | 19          |
| 6.3.2.3.6 mux ram data.....           | 19          |
| 6.3.2.3.7 transmitter controller..... | 20          |

|          |                                                                                                                            |           |
|----------|----------------------------------------------------------------------------------------------------------------------------|-----------|
| <b>7</b> | <b>CONTROLLER SIMAP .....</b>                                                                                              | <b>22</b> |
| 7.1      | FUNCTIONALITY .....                                                                                                        | 22        |
| 7.2      | CONTENT SUMMARY .....                                                                                                      | 23        |
| 7.3      | CONTENT DESCRIPTION .....                                                                                                  | 24        |
| 7.3.1    | <i>controller</i> .....                                                                                                    | 24        |
| 7.3.1.1  | Functionality.....                                                                                                         | 24        |
| 7.3.1.2  | State machine summary.....                                                                                                 | 24        |
| 7.3.2    | <i>reader address registers</i> .....                                                                                      | 25        |
| 7.3.2.1  | Functionality.....                                                                                                         | 25        |
| 7.3.3    | <i>checker</i> .....                                                                                                       | 25        |
| 7.3.3.1  | Functionality.....                                                                                                         | 25        |
| 7.3.4    | <i>work register</i> .....                                                                                                 | 25        |
| 7.3.4.1  | Functionality.....                                                                                                         | 25        |
| 7.3.5    | <i>selector register</i> .....                                                                                             | 25        |
| 7.3.5.1  | Functionality.....                                                                                                         | 25        |
| 7.3.6    | <i>blocks: mux and demux</i> .....                                                                                         | 25        |
| 7.3.6.1  | Functionality.....                                                                                                         | 25        |
| 7.3.7    | <i>application address registers</i> .....                                                                                 | 25        |
| 7.3.7.1  | Functionality.....                                                                                                         | 25        |
| 7.3.8    | <i>group config addressing</i> .....                                                                                       | 26        |
| 7.3.8.1  | Functionality.....                                                                                                         | 26        |
| 7.3.9    | <i>group config memory</i> .....                                                                                           | 26        |
| 7.3.9.1  | Functionality.....                                                                                                         | 26        |
| 7.3.9.2  | Notice.....                                                                                                                | 26        |
| 7.3.10   | <i>writer address registers</i> .....                                                                                      | 26        |
| 7.3.10.1 | Functionality.....                                                                                                         | 26        |
| <b>8</b> | <b>APPLICATION.....</b>                                                                                                    | <b>27</b> |
| 8.1      | FUNCTIONALITY .....                                                                                                        | 27        |
| 8.2      | CONTENT SUMMARY .....                                                                                                      | 27        |
| 8.3      | CONTENT DESCRIPTION .....                                                                                                  | 28        |
| 8.3.1    | <i>blocks: mux and demux</i> .....                                                                                         | 28        |
| 8.3.1.1  | Functionality.....                                                                                                         | 28        |
| 8.3.2    | <i>blocks: ROM registers, params registers, program memory, groupe value registers, groupe description registers</i> ..... | 28        |
| 8.3.2.1  | Functionality.....                                                                                                         | 28        |
| 8.3.3    | <i>application test</i> .....                                                                                              | 28        |
| 8.3.3.1  | Functionality.....                                                                                                         | 28        |
| 8.3.4    | <i>event register</i> .....                                                                                                | 28        |
| 8.3.4.1  | Functionality.....                                                                                                         | 28        |
| <b>9</b> | <b>TEST BENCH .....</b>                                                                                                    | <b>29</b> |
| 9.1      | RECEIVER_TB .....                                                                                                          | 29        |
| 9.1.1    | <i>Info on the test</i> .....                                                                                              | 29        |
| 9.1.2    | <i>Test sequence</i> .....                                                                                                 | 29        |
| 9.1.2.1  | Part: Reset sequence.....                                                                                                  | 29        |
| 9.1.2.2  | Part: Detail of a valid frame without error .....                                                                          | 30        |
| 9.1.2.3  | Part: CRC error and multiple Rx errors.....                                                                                | 31        |
| 9.1.2.4  | Part: Fill the RAM and test the limit of free RAM.....                                                                     | 32        |
| 9.1.2.5  | Part: Fill the RAM and test the limit of free RAM.....                                                                     | 34        |
| 9.1.2.6  | Part: Test if reader can move correctly.....                                                                               | 36        |
| 9.1.3    | <i>Notice</i> .....                                                                                                        | 38        |
| 9.2      | TRANSMITTER_TB.....                                                                                                        | 39        |
| 9.3      | MAC_PLUS_TB .....                                                                                                          | 40        |
| 9.3.1    | <i>Little description of MAC_plus_tester_fas.vhd</i> .....                                                                 | 40        |
| 9.3.2    | <i>Test sequence</i> .....                                                                                                 | 40        |
| 9.3.2.1  | Part: Send only writing frames.....                                                                                        | 40        |

|            |                                                                                                                     |           |
|------------|---------------------------------------------------------------------------------------------------------------------|-----------|
| 9.3.2.2    | Part: Send only reading frames .....                                                                                | 41        |
| 9.3.2.3    | Part: Send only event .....                                                                                         | 41        |
| 9.3.2.4    | Part: Send all frames with not SimAP frame .....                                                                    | 41        |
| 9.3.2.5    | Part: Send with transmission error .....                                                                            | 42        |
| 9.3.2.6    | Part: Transmission loss.....                                                                                        | 42        |
| 9.3.2.7    | Part: Others.....                                                                                                   | 42        |
| <b>10</b>  | <b>TEST OF DEMO.....</b>                                                                                            | <b>43</b> |
| 10.1       | EQUIPMENT .....                                                                                                     | 43        |
| 10.2       | NOTICE.....                                                                                                         | 43        |
| <b>11</b>  | <b>IMPROVEMENTS TO MAKE.....</b>                                                                                    | <b>44</b> |
| 11.1       | RECEIVER.....                                                                                                       | 44        |
| 11.2       | TRANSMITTER .....                                                                                                   | 44        |
| 11.3       | CONTROLLER .....                                                                                                    | 45        |
| 11.4       | MEMORIES .....                                                                                                      | 45        |
| 11.5       | APPLICATION.....                                                                                                    | 45        |
| 11.6       | TEST .....                                                                                                          | 46        |
| 11.7       | OTHERS.....                                                                                                         | 46        |
| 11.8       | MISC .....                                                                                                          | 46        |
| <b>12</b>  | <b>WHY THESE CHOICES ? .....</b>                                                                                    | <b>47</b> |
| 12.1       | INTERFACE ETHERNET .....                                                                                            | 47        |
| 12.1.1     | receiver.....                                                                                                       | 47        |
| 12.1.1.1   | crc32 .....                                                                                                         | 47        |
| 12.1.1.2   | RAM dp .....                                                                                                        | 48        |
| 12.1.1.3   | receiver controller.....                                                                                            | 49        |
| 12.1.2     | transmitter .....                                                                                                   | 50        |
| 12.1.2.1   | transmitter controller.....                                                                                         | 50        |
| 12.2       | CONTROLLER SIMAP.....                                                                                               | 51        |
| 12.2.1     | controller.....                                                                                                     | 51        |
| 12.2.1.1   | parsing.....                                                                                                        | 52        |
| 12.2.1.1.1 | physical addressing mode.....                                                                                       | 54        |
| 12.2.1.1.2 | group addressing mode.....                                                                                          | 56        |
| 12.3       | APPLICATION.....                                                                                                    | 57        |
| 12.3.1     | mux and demux.....                                                                                                  | 57        |
| 12.3.2     | blocks: ROM registers, params registers, program memory, groupe value registers, groupe description registers ..... | 57        |
| 12.3.3     | application_test.....                                                                                               | 57        |
| 12.3.4     | event_registers.....                                                                                                | 57        |
| <b>13</b>  | <b>APPENDICES .....</b>                                                                                             | <b>58</b> |
| 13.1       | ABBREVIATION .....                                                                                                  | 58        |
| 13.2       | FILES AND FOLDERS FOR THE PROJECT .....                                                                             | 59        |
| 13.3       | RAM STRUCTURE .....                                                                                                 | 60        |
| 13.3.1     | Structure .....                                                                                                     | 60        |
| 13.3.2     | Header.....                                                                                                         | 60        |
| 13.3.2.1   | Empty header.....                                                                                                   | 60        |
| 13.3.2.2   | Error header.....                                                                                                   | 61        |
| 13.3.2.3   | Frame header.....                                                                                                   | 61        |
| 13.4       | ABBREVIATIONS TIRÉE DU 802.3.....                                                                                   | 62        |

## 1 INTRODUCTION

In this document, there is a lot of abbreviation. To know the signification of these, it must to go on: "appendices" and to see: "abbreviation".

The network topology is quite similar to a standard EIB infrastructure.  
The following illustration shows a typical SimAP installation:

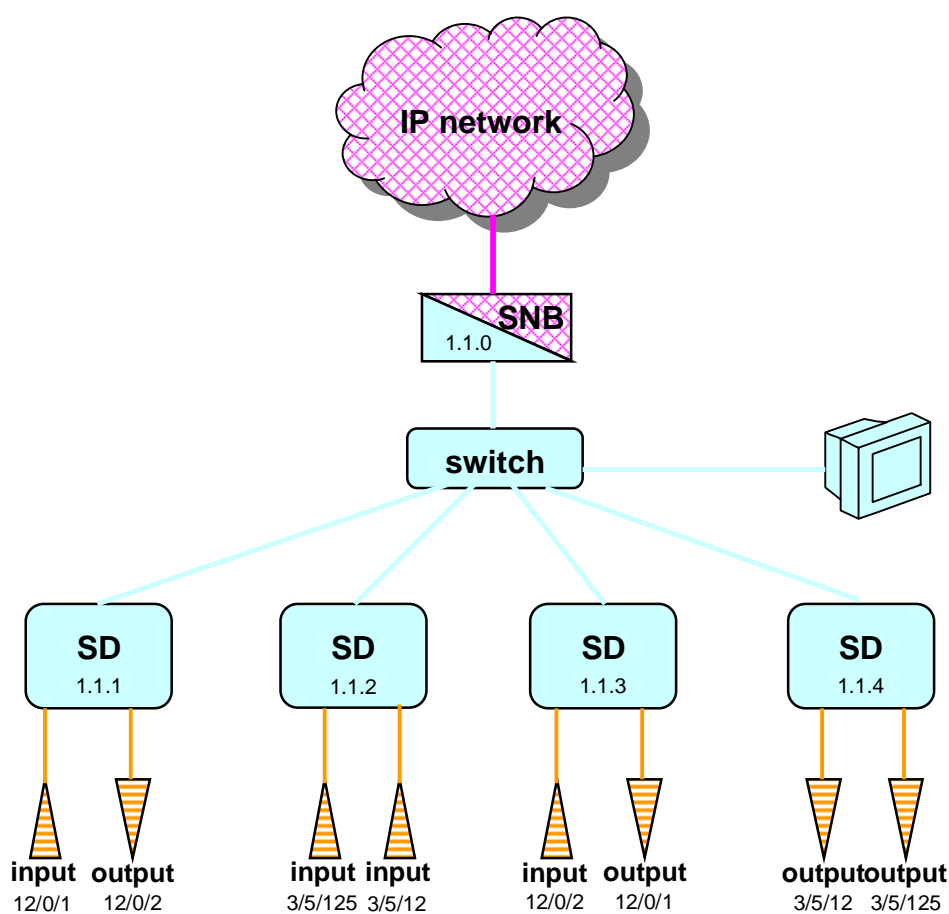


Figure 1: SAoE network topology

In this project, only the SD (SAoE Device) is realized.

## 2 WORK

To realize a design that can be downloaded in a FPGA of a SD so it can communicate with other SD.

All devices are connected with Ethernet.

The communication is based on SimAP.

So the design must to receive Ethernet frames, to work with these frames respecting the protocol SimAP and be able to send Ethernet frames.

### 2.1 Objectives

The device can realise a simple Ethernet node without processor. The IOs are directly connected on the device. The used protocol is SimAP (Simple Automation Protocol). The design of the device must be composed be three parts: an Ethernet interface (receiver and transmitter), a controller and a specific part to the application.

### 2.2 Parsing

#### Receiver

- To receive correctly the frames
- To be synchronized with Rx clock
- To store the frames
- To calculate CRC
- To check all errors on the frames

#### Transmitter

- To be synchronized with Tx clock
- To read the frames
- To calculate CRC
- To send correctly the frames

#### Controller

- To read the received frames
- To get the events
- To identify the frames
- To work with the registers or the memories
- To create frames
- To store the created frames

#### Application specific

- To work with the outputs or the inputs
- To work with the registers or the memories
- To store the events

### 3 THE USED FRAME

#### 3.1 Globality

The complete Ethernet frame that it is possible to see on the twisted pair Rx or Tx:

| SSD | Preamble | SFD | Ethernet field | Data     | PAD  | FCS | Extension | ESD | IFG |
|-----|----------|-----|----------------|----------|------|-----|-----------|-----|-----|
| 1   | 6        | 1   | 14             | 0 - 1500 | 46-0 | 4   | ?         | 1   | >11 |

On SimAP project, the used frame to work with the SimAP protocol and the PHY (Intel chip: LXT972A):

| Preamble | SFD | Ethernet field | SimAP field | Data     | PAD  | FCS |
|----------|-----|----------------|-------------|----------|------|-----|
| 7        | 1   | 14             | 10          | 0 – 1372 | 36-0 | 4   |

#### 3.2 Ethernet field

| Destination Address | Source Address | Type    |
|---------------------|----------------|---------|
| 6 bytes             | 6 bytes        | 2 bytes |

The Ethernet field is composed of 3 parts:

- Destination Address : it's the MAC address from recipient.
- Source Address : it's the MAC address from issuer.
- Type : it's to know the contents of data.

#### 3.3 SimAP field

| PCF    | GID     | NPCI    | TPCI   | APCI    |
|--------|---------|---------|--------|---------|
| 1 byte | 2 bytes | 2 bytes | 1 byte | 4 bytes |

The SimAP field is composed of 5 major parts:

- PCF : it's to have information about the frame.
- GID : it's the group identifier.
- NPCI : it's to have information about the data.
- TPCI : it's the transport protocol control information.
- APCI : it's to know how to use the data.

Notice:

The parts: PCF, GID, NPCI, TPCI and APCI are composed by small parts that depend from addressing mode.

(To know contents, see part: 13.2.1.1 parsing)

### 3.4 Others parts

The others parts:

- SSD : it's to indicate the start of stream of frame.
- Preamble : it's for physical medium stabilization and synchronization.
- SFD : it's the start frame delimiter.
- Data : it's data of the frame.
- PAD : it's used to complete the frame so as to have the min length.
- FCS : it's to check and to validate the frame sequence.
- Extension : it's an extension for the frame.
- ESD : it's to indicate the end of stream of frame.
- IFG : it's the min time to wait before to send another frame.

## 4 GLOBAL FUNCTIONING

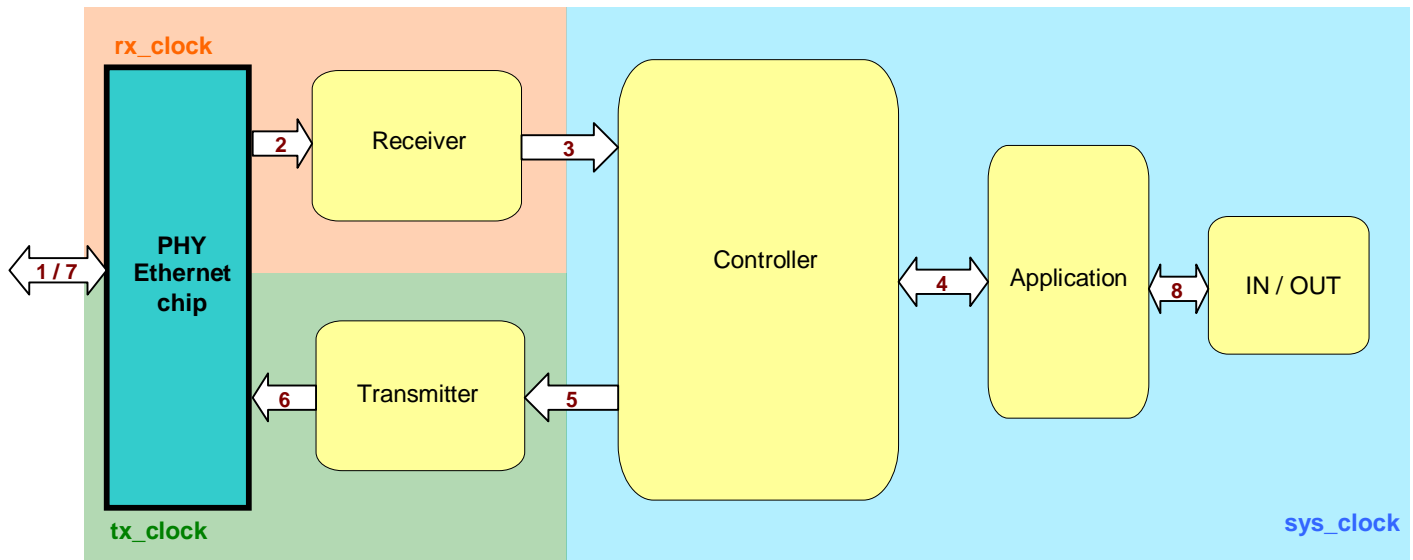


Figure 2: Global functioning

- 1: The PHY receives the stream of a frame from the twisted pair Rx.
- 2: The PHY transmits a frame with some signals drivers to the receiver.  
The receiver stores a frame with her frame header in a RAM.
- 3: The controller reads a stored frame.  
The controller parses a read frame.
- 4: The controller stores data in registers or memories.  
The application reads data in registers or memories.  
The application stores events.  
The controller gets the next event.  
The application stores data in registers or memories.  
The controller reads data in registers or memories.
- 5: The controller stores a created frame with her frame header in a RAM.
- 6: The transmitter reads a frame.  
The transmitter sends a frame with some signals drivers to the PHY.
- 7: The PHY transmits the stream of a frame on the twisted pair Tx.
- 8: The application sets outputs.  
The application reads inputs.



## 5 CONTENT OF DESIGN

The following figure shows the SimAP architecture. Each block will be described in the next chapters.

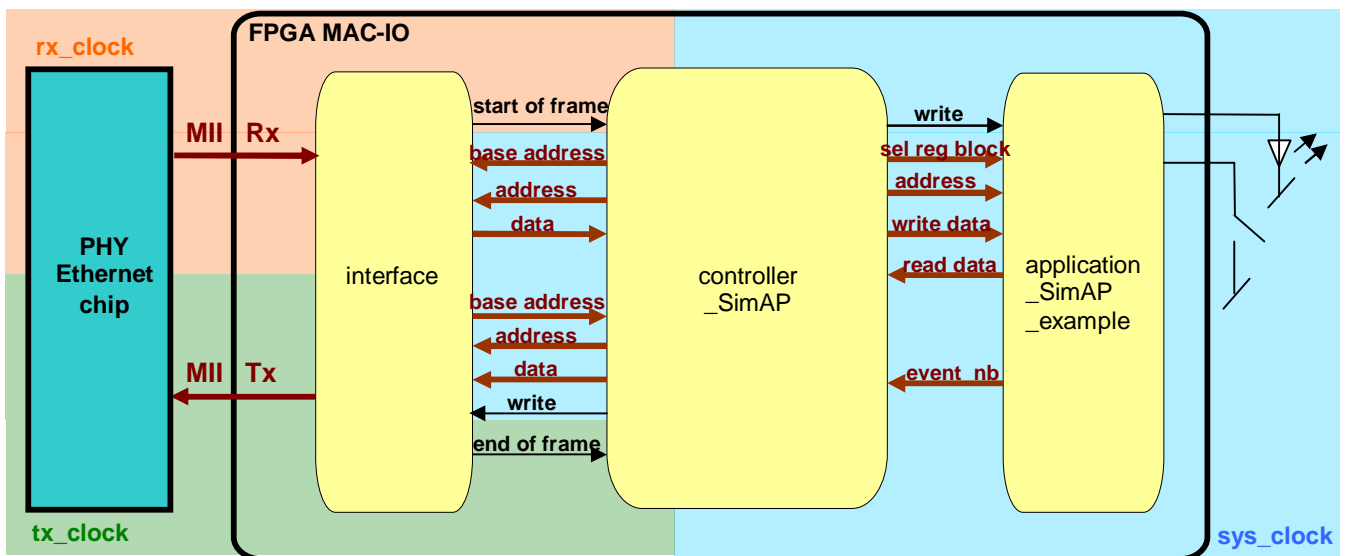


Figure 3: SimAP architecture

For this project, there are 3 important libraries:

- interface\_Ethernet\_lib (block: interface)
- controller\_SimAP\_lib (block: controller\_SimAP)
- application\_SimAP\_lib (block: application\_SimAP\_example)

In the library: interface\_Ethernet\_lib, there are all elements to make the reception of frames with the storage and to make the transmission of frames with the storage.

In the library: controller\_SimAP\_lib, there are all elements to communicate respecting the protocol: SimAP and to store data in registers or memories.

In the library: application\_SimAP\_lib, there are only the elements to make a little example comprising all registers or memories to be able to test elements describes in SimAP documentation.

## 6 INTERFACE ETHERNET

### 6.1 Functionality

The interface receives data from PHY Ethernet chip and saves them on a RAM. A structure is created to recognize what is saved (See part: 14.3 RAM structure). To read correct data, it's necessary to set the signals: base address and address.

A signal will be transmitted when the SFD (Start Frame Delimiter) is detected. (This signal, it's a pulse which depends of rx\_clock.)

And interface transmits data to PHY Ethernet chip.

### 6.2 Content summary

The following figure shows the interface architecture. Each block will be described in details in the next chapters.

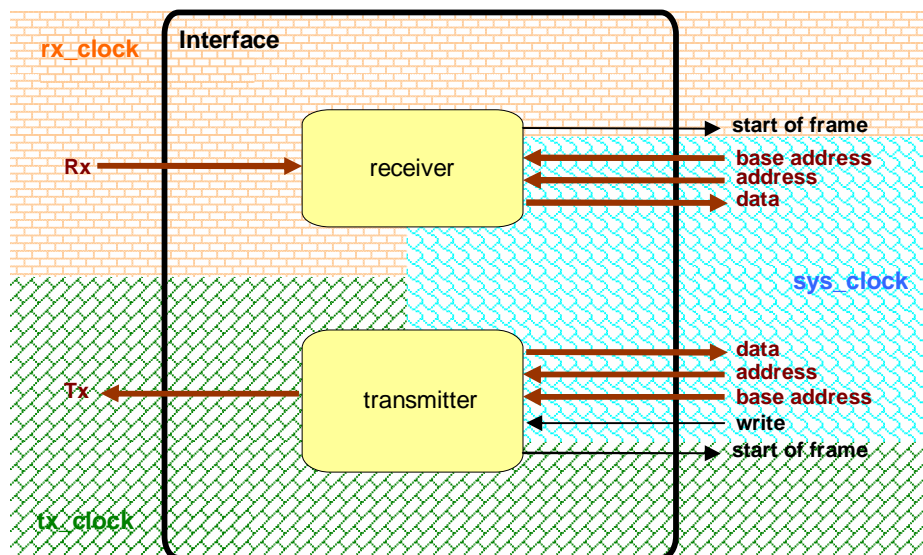


Figure 4: Interface architecture

## 6.3 Content description

### 6.3.1 receiver

#### 6.3.1.1 Functionality

The receiver receives data from PHY Ethernet chip.

When the SFD (Start Frame Delimiter) is detected, a pulse is sent.  
(This signal isn't synchronize.)

He checks the received data to detect a CRC error or the transmission error signal.  
With signal: base address, it's possible to check free memory.

He writes data in RAM with a specific format (see part: 14.3 RAM structure).

These data can be read by changing address.

#### 6.3.1.2 Content summary

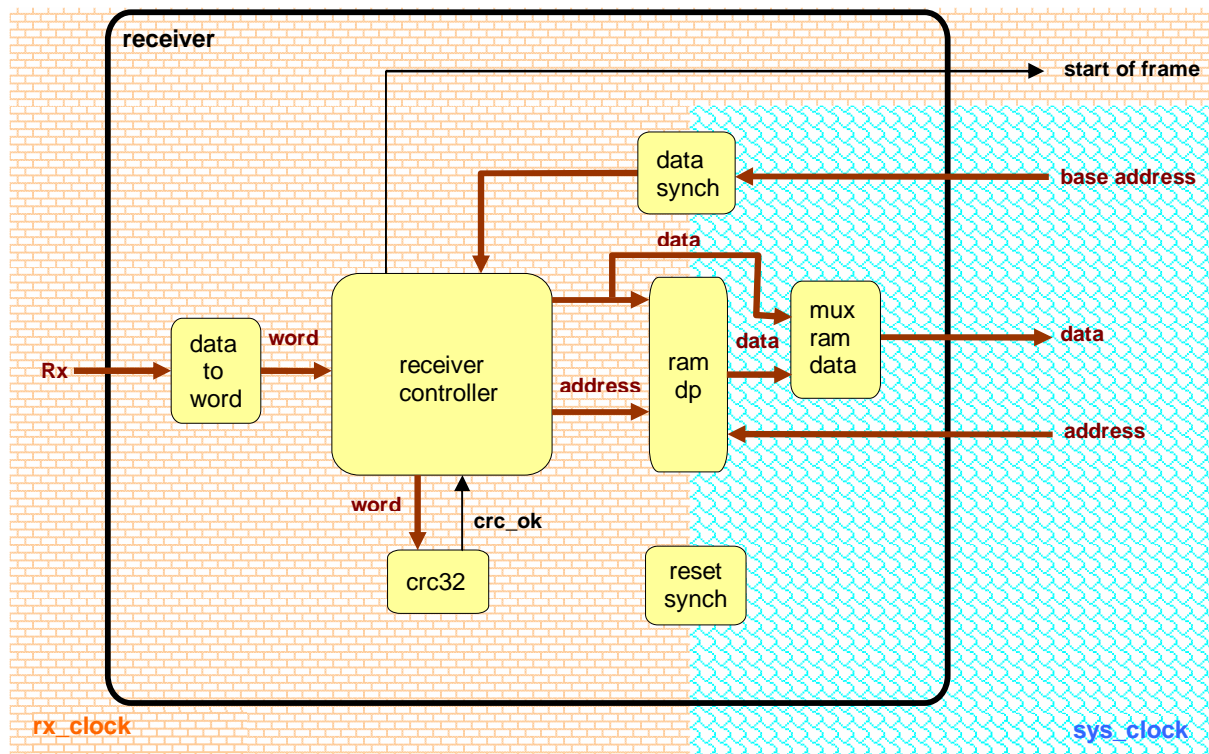


Figure 5: Receiver architecture

### **6.3.1.3 Content description**

#### **6.3.1.3.1 *data to word***

##### **6.3.1.3.1.1 Functionality**

He adapts the parallel bits number.

##### **6.3.1.3.1.2 Notice**

At this moment, it's only possible to have at entry a parallel bits number smaller than the parallel bits number at exit.

#### **6.3.1.3.2 *crc32***

##### **6.3.1.3.2.1 Functionality**

He calculates CRC as long as calculate is high and enable is high.  
He compares CRC with residue as long as enable is low.

##### **6.3.1.3.2.2 Notice**

crc32 is created with crcgen.pl (Version: 2.0)  
(2007 HES-SO // Valais – Wallis)

<http://hevs-cof.dynalias.net:8057/internet/CRC/>

Specifications of CRC 32

constant input\_width : positive := 8;

constant crc\_width : positive := 32;

### **6.3.1.3.3    *synchronizer***

#### **6.3.1.3.3.1    Functionality**

He synchronizes data input to rx clock.

### **6.3.1.3.4    *reset synch***

#### **6.3.1.3.4.1    Functionality**

He synchronizes reset input to rx clock.

### **6.3.1.3.5    *ram dp***

#### **6.3.1.3.5.1    Functionality**

It's a ram with dual port.

A port works with rx clock and he is used for the writing.

The other port works with system clock and he is used for the reading.

#### **6.3.1.3.5.2    Notice**

The implementation is based on XST User Guide (xst.pdf) page 147.

### **6.3.1.3.6    *mux ram data***

#### **6.3.1.3.6.1    Functionality**

He picks a data input to set the output.

#### **6.3.1.3.6.2    Notice**

In this case, he is used to make believe that the data read are that of an empty header while the memory is initialized.

### 6.3.1.3.7 receiver controller

#### 6.3.1.3.7.1 Functionality

He guides and drives all signals to have the receiver functionality.  
(He is the receiver brain.)

#### 6.3.1.3.7.2 Content summary

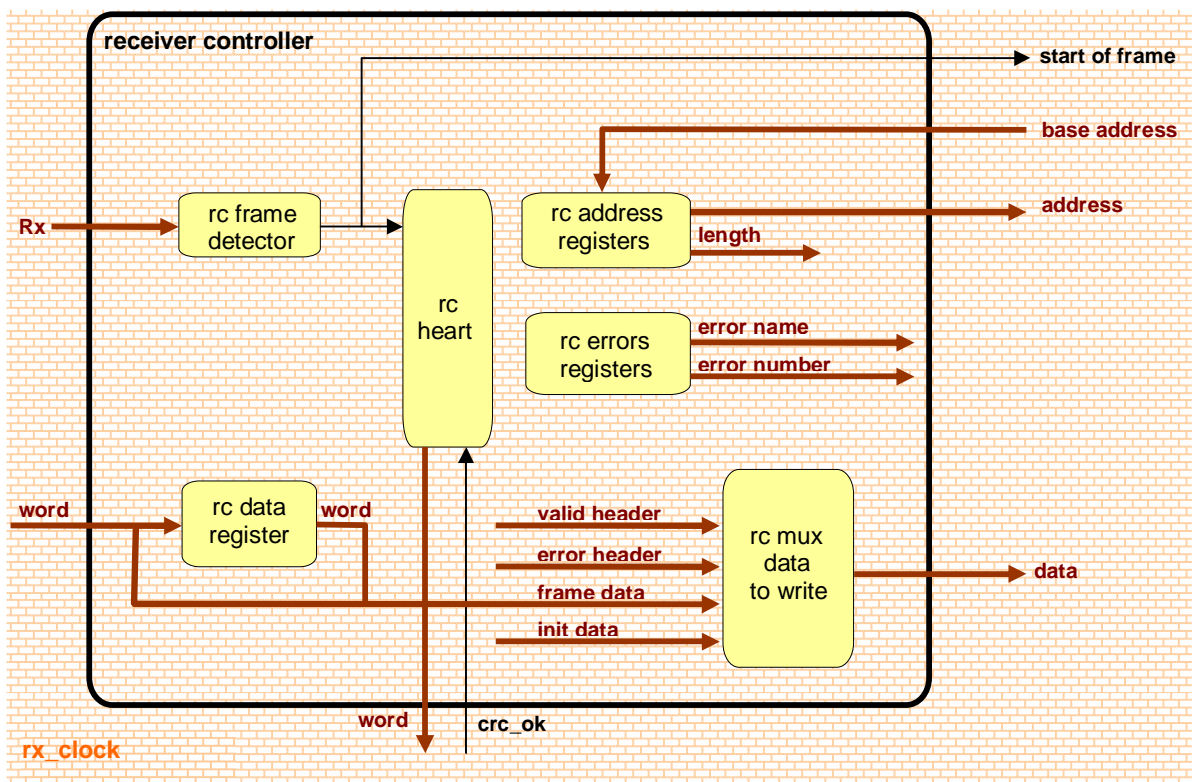


Figure 6: Receiver controller architecture

Notice:

To see content of all data type ( valid header, error header, frame data and init data ),  
open HDL designer and open receiver controller struct.

#### 6.3.1.3.7.3 Content description

##### 6.3.1.3.7.3.1 rc frame detector

###### 6.3.1.3.7.3.1.1 Functionality

He sends a pulse when the SFD (Start Frame Delimiter) is detected.  
(This pulse is asynchronous.)

#### 6.3.1.3.7.3.2 rc data register

##### *6.3.1.3.7.3.2.1 Functionality*

It's a register to delay the data frame signals because in this case it's necessary to have 16 bits to write data frame in RAM.

#### 6.3.1.3.7.3.3 rc address registers

##### *6.3.1.3.7.3.3.1 Functionality*

He is constituted 3 registers (for current address, begin address and end address) and process driven by rc heart to control the registers.

This is used to check and to send the state of the RAM.

#### 6.3.1.3.7.3.4 rc errors registers

##### *6.3.1.3.7.3.4.1 Functionality*

He is used to store, to check and to send the state of current error and previous error.

#### 6.3.1.3.7.3.5 rc mux data to write

##### *6.3.1.3.7.3.5.1 Functionality*

He picks a data input to set the output.

##### *6.3.1.3.7.3.5.2 Notice*

In this case, he is used to select data type which will be written in RAM.

#### 6.3.1.3.7.3.6 rc heart

##### *6.3.1.3.7.3.6.1 Functionality*

He directs all rc (receiver controller) internal blocks to realize a sequence of work.  
(He is the receiver controller heart.)



## 6.3.2 transmitter

### 6.3.2.1 Functionality

The transmitter reads a frame in RAM with a specific format (see part: 14.3 RAM structure).

He calculates the CRC.

He sends the stream of a frame with all signals control to the PHY.

When all data are sent, a pulse is created.  
(This signal: end of frame isn't synchronized.)

### 6.3.2.2 Content summary

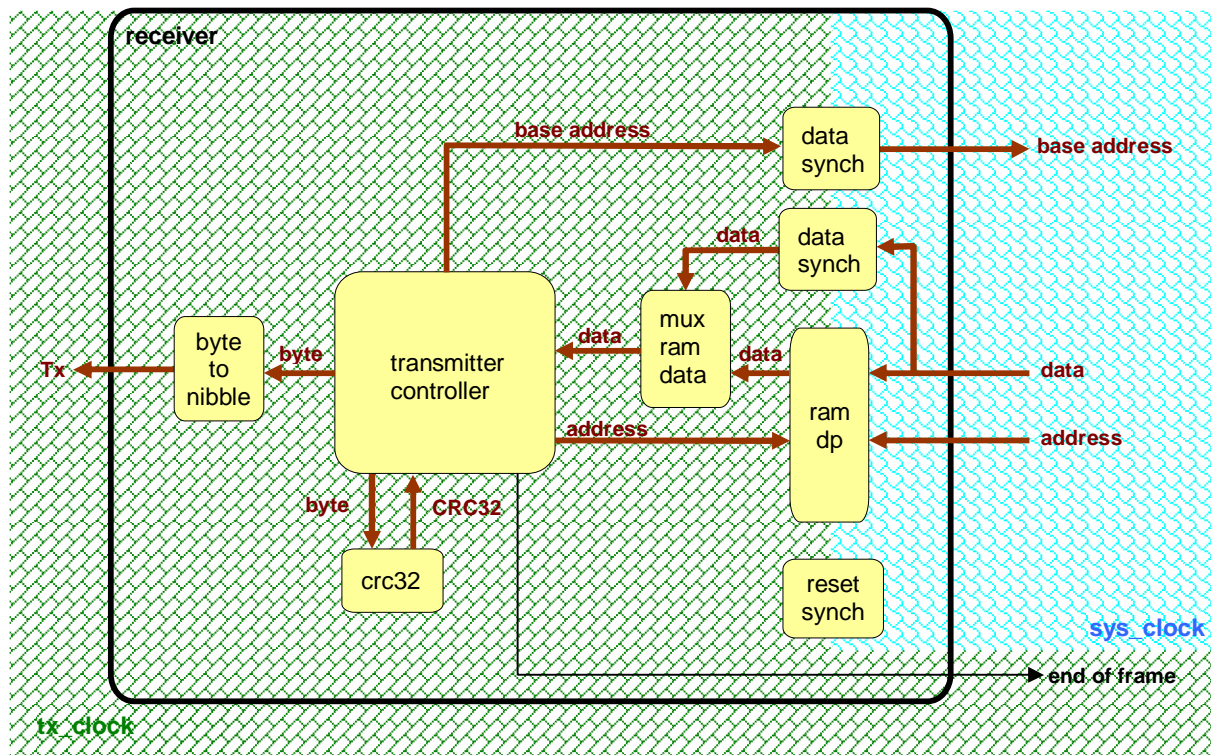


Figure 7: Transmitter architecture



### **6.3.2.3 Content description**

#### **6.3.2.3.1 *byte to nibble***

##### **6.3.2.3.1.1 Functionality**

He adapts the parallel bits number.

##### **6.3.2.3.1.2 Notice**

At this moment, it's usable only with nibble because the PHY gets 4 words by 4 bits.

#### **6.3.2.3.2 *crc32***

##### **6.3.2.3.2.1 Functionality**

He calculates CRC as long as calculate is high and enable is high.

##### **6.3.2.3.2.2 Notice**

crc32 is created with crcgen.pl (Version: 2.0)  
(2007 HES-SO // Valais – Wallis)

<http://hevs-cof.dynalias.net:8057/internet/CRC/>

Specifications of CRC 32

constant input\_width : positive := 8;  
constant crc\_width : positive := 32;

#### **6.3.2.3.3 *synchronizer***

##### **6.3.2.3.3.1 Functionality**

He synchronizes data input to tx clock.

#### **6.3.2.3.4 *reset synch***

##### **6.3.2.3.4.1 Functionality**

He synchronizes reset input to tx clock.

### **6.3.2.3.5 ram dp**

#### **6.3.2.3.5.1 Functionality**

It's a ram with dual port.

A port works with tx clock and he is used for the writing.

The other port works with system clock and he is used for the reading.

#### **6.3.2.3.5.2 Notice**

The implementation is based on XST User Guide (xst.pdf) page 147.

### **6.3.2.3.6 mux ram data**

#### **6.3.2.3.6.1 Functionality**

He picks a data input to set the output.

#### **6.3.2.3.6.2 Notice**

In this case, he is used to make believe that the data read are that of an empty header while the memory is initialized.

### 6.3.2.3.7 transmitter controller

#### 6.3.2.3.7.1 Functionality

He guides and drives all signals to have the transmitter functionality.  
(He is the transmitter brain.)

#### 6.3.2.3.7.2 Content summary

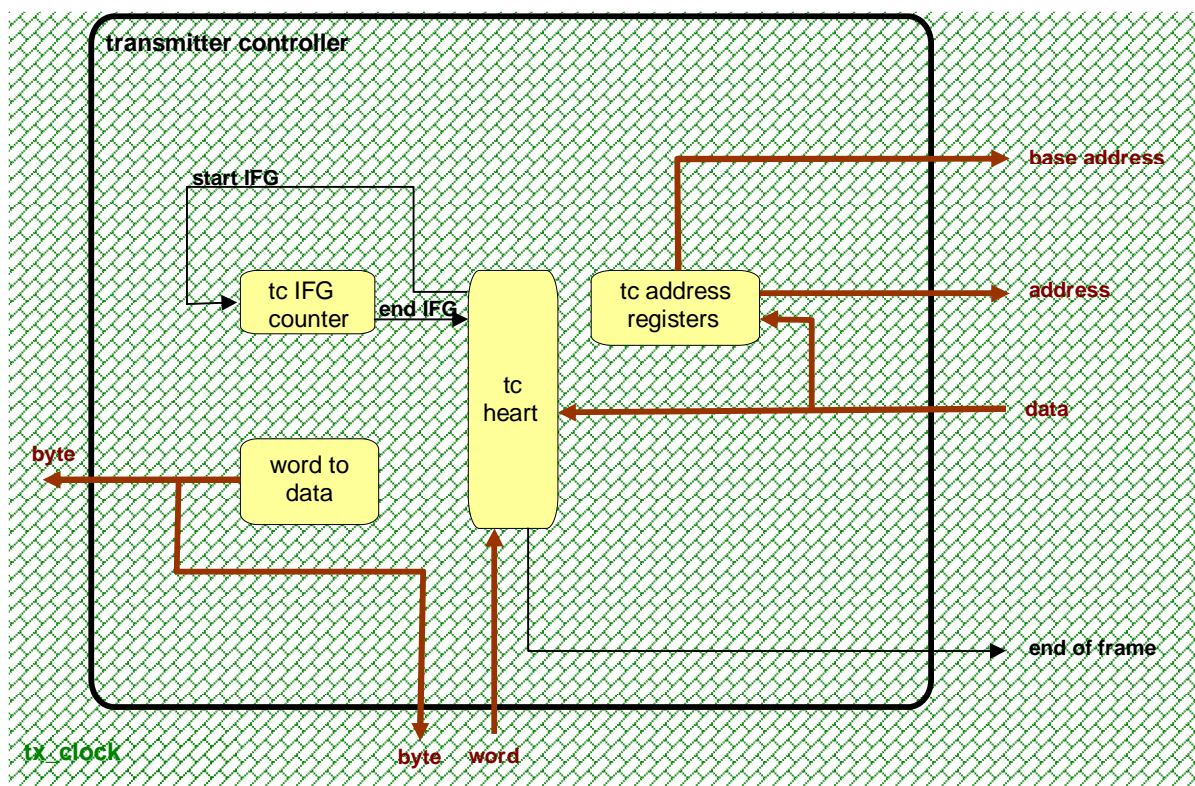


Figure 8: Transmitter controller architecture

#### 6.3.2.3.7.3 Content description

##### 6.3.2.3.7.3.1 tc address registers

###### 6.3.2.3.7.3.1.1 Functionality

He is constituted 3 registers (for current address, begin address and end address) and process driven by tc heart to control the registers.

This is used to check and to read the state of the RAM.

#### 6.3.2.3.7.3.2 tc IFG counter

##### *6.3.2.3.7.3.2.1 Functionality*

He is used to wait a minimum time before to send the next frame.

#### 6.3.2.3.7.3.3 word to data

##### *6.3.2.3.7.3.3.1 Functionality*

He adapts the parallel bits number.

#### 6.3.2.3.7.3.4 tc heart

##### *6.3.2.3.7.3.4.1 Functionality*

He directs all tc (transmitter controller) internal blocks to realize a sequence of work.  
(He is the transmitter controller heart.)

## 7 CONTROLLER SIMAP

### 7.1 Functionality

There are 2 information sources:

- Rx RAM (from Ethernet)
- event register (from inputs)

And there are 2 storing places:

- Tx RAM (for Ethernet)
- registers and memories (for outputs)

With that, it is possible to have various functionalities:

- an event leads a broadcasted frame on Ethernet.
- a frame with physical addressing can be write or read anything in registers or memories and returns a response on Ethernet.
- a frame with group addressing can be write or read anything in registers (not in memories) and returns a response on Ethernet.

So the controller reads the frames in the Rx RAM.

He gets the events.

He parses the frames using the structure (See part: 14.3 RAM structure).

He reads or writes in registers or memories.

He stores the created frames in the Tx RAM.

## 7.2 Content summary

The following figure shows the controller SimAP architecture. Each block will be described in details in the next chapters.

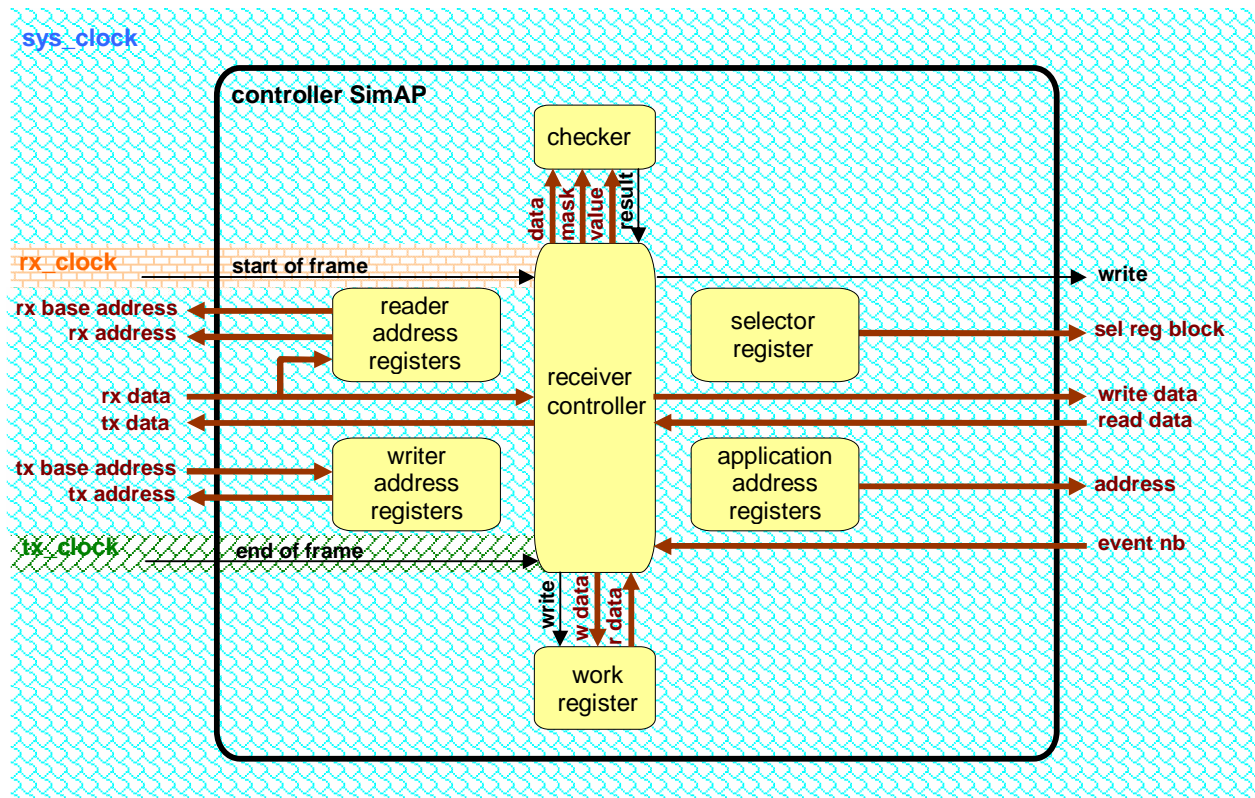


Figure 9: Controller SimAP architecture

Notice:

To see the exact contents, open HDL designer and open controller SimAP struct.

## 7.3 Content description

### 7.3.1 controller

#### 7.3.1.1 Functionality

The controller guides and drives all signals to have the controller SimAP functionality. (It's the brain of controller SimAP.)

#### 7.3.1.2 State machine summary

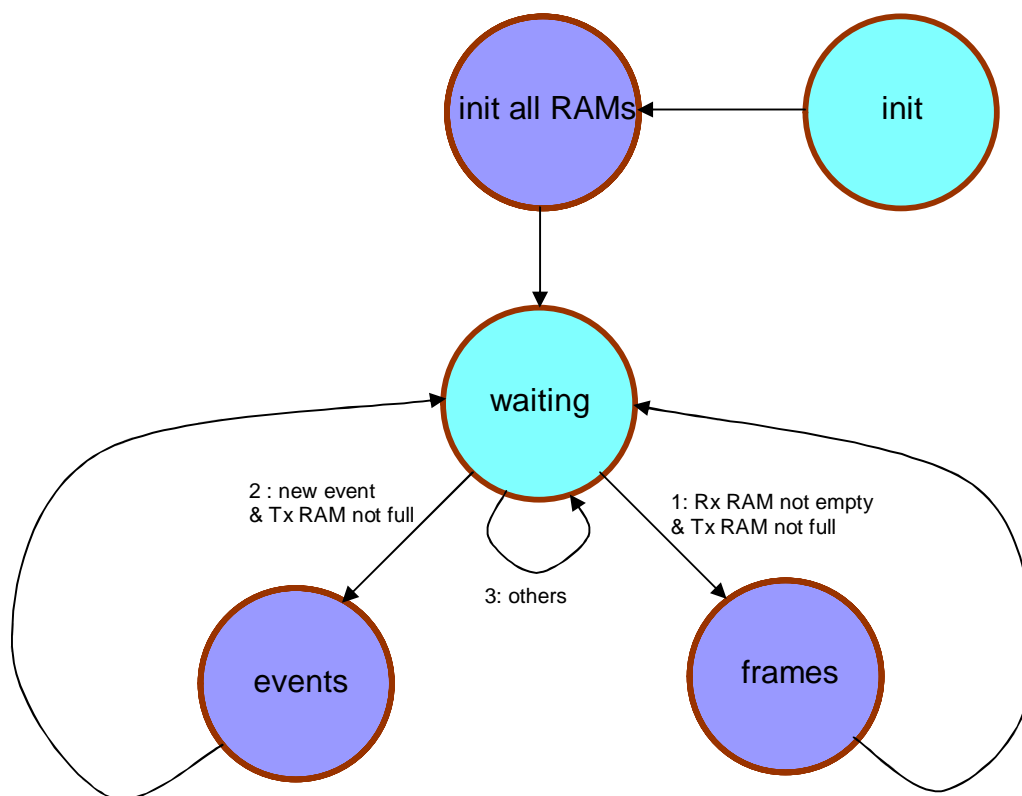


Figure 10: State machine of controller

## **7.3.2 reader address registers**

### **7.3.2.1 Functionality**

He is constituted 3 registers (for current address, begin address and end address) and process driven by controller to control the registers.

This is used to check and to read the state of the RAM.

## **7.3.3 checker**

### **7.3.3.1 Functionality**

This block compares data combined with a mask to a value.

## **7.3.4 work register**

### **7.3.4.1 Functionality**

The work register stores data.

## **7.3.5 selector register**

### **7.3.5.1 Functionality**

The selector register selects a memory so as to work with.

## **7.3.6 blocks: mux and demux**

### **7.3.6.1 Functionality**

They are used to direct the data flow.

## **7.3.7 application address registers**

### **7.3.7.1 Functionality**

He is constituted 3 registers (for current address, begin address and end address) and process driven by controller to control the registers.

This is used to write or to read memory block.



## **7.3.8 group config addressing**

### **7.3.8.1 Functionality**

He adapts the address to read the correct part of config RAM.

## **7.3.9 group config memory**

### **7.3.9.1 Functionality**

It's a ram with dual port.

Where is stored the GID and the EIB flags.

### **7.3.9.2 Notice**

The implementation is based on XST User Guide (xst.pdf) page 147.

## **7.3.10 writer address registers**

### **7.3.10.1 Functionality**

He is constituted 3 registers (for current address, begin address and end address) and process driven by controller to control the registers.

This is used to write frame in the RAM.

## 8 APPLICATION

### 8.1 Functionality

The block: "application SimAP example" has been created so as to have a functional demo.

### 8.2 Content summary

The following figure shows the application SimAP example architecture. Each block will be described in details in the next chapters.

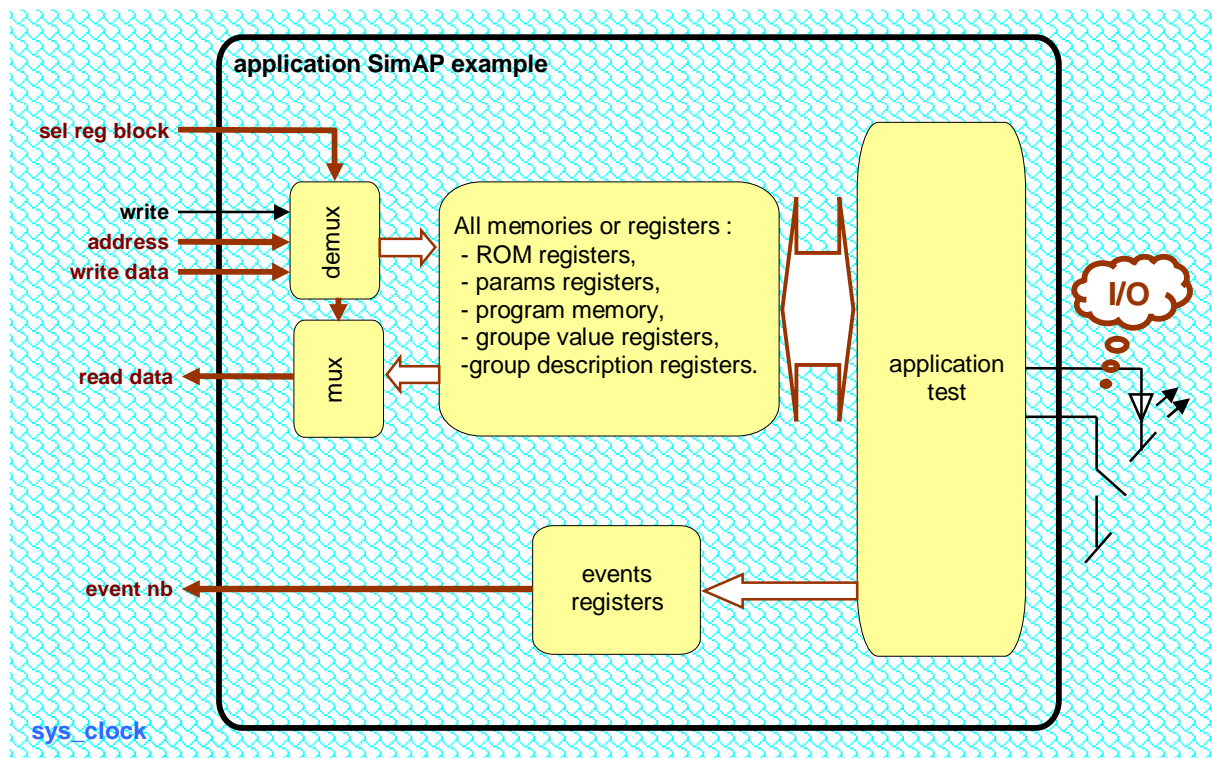


Figure 11: Application SimAP example architecture

Notice:

It's only an example of something that runs.

To see the exact contents, open HDL designer and open application SimAP example struct.

## **8.3 Content description**

### **8.3.1 blocks: mux and demux**

#### **8.3.1.1 Functionality**

They are used to direct the data flow.

### **8.3.2 blocks: ROM registers, params registers, program memory, groupe value registers, groupe description registers**

#### **8.3.2.1 Functionality**

It is necessary that they are readable or writable as a RAM from controller interface. And from application, it is possible to read or to write as a RAM or as a register.

### **8.3.3 application test**

#### **8.3.3.1 Functionality**

The application test is an example to use the registers with IOs.

### **8.3.4 event register**

#### **8.3.4.1 Functionality**

It is an example to store and to send events.

## 9 TEST BENCH

## 9.1 receiver\_tb

### 9.1.1 Info on the test

A table is used to describe the test of receiver.

## Table contents:

[illegible]

Regularly, a little sentence summarizes the part of test and why it's tested.

And some time, there is a graph of waves.

### 9.1.2 Test sequence

To start, the test of reader is locked to address: 0 hex and to base\_address : 0 hex.

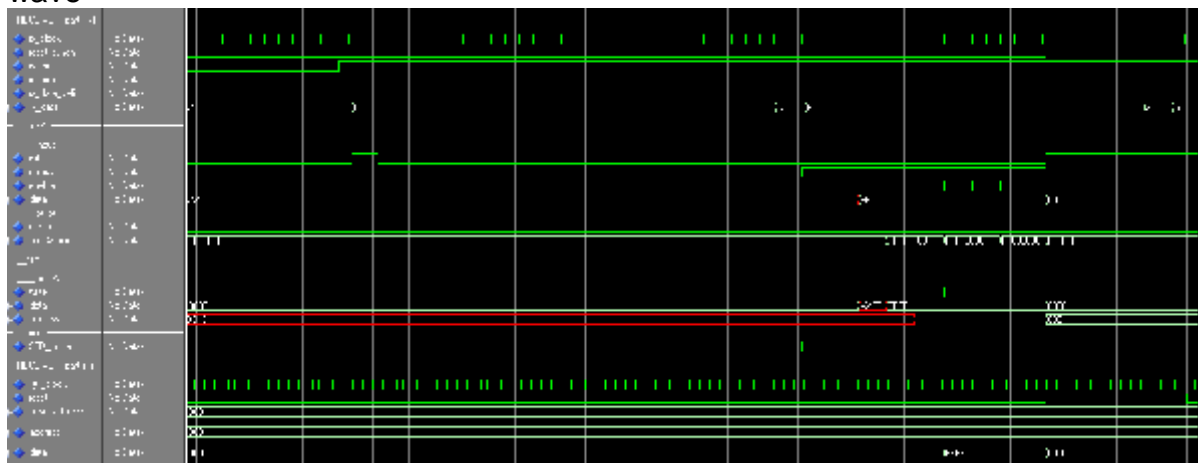
### 9.1.2.1 Part: Reset sequence

The test begins with a reset.

It's to check signals and state of RAM.

|           |         |                   |     |      |
|-----------|---------|-------------------|-----|------|
| 0 to 1999 |         | => don't know     | XXX | XXXX |
|           | 2000 ns | => begin of reset | XXX | XXXX |
| 2216      | 216 ns  | => end of reset   | 000 | 0000 |

wave



memory

[illegible]

### 9.1.2.2 Part: Detail of a valid frame without error

The writing of a valid frame.

It's to check if all data of frame is written in RAM.

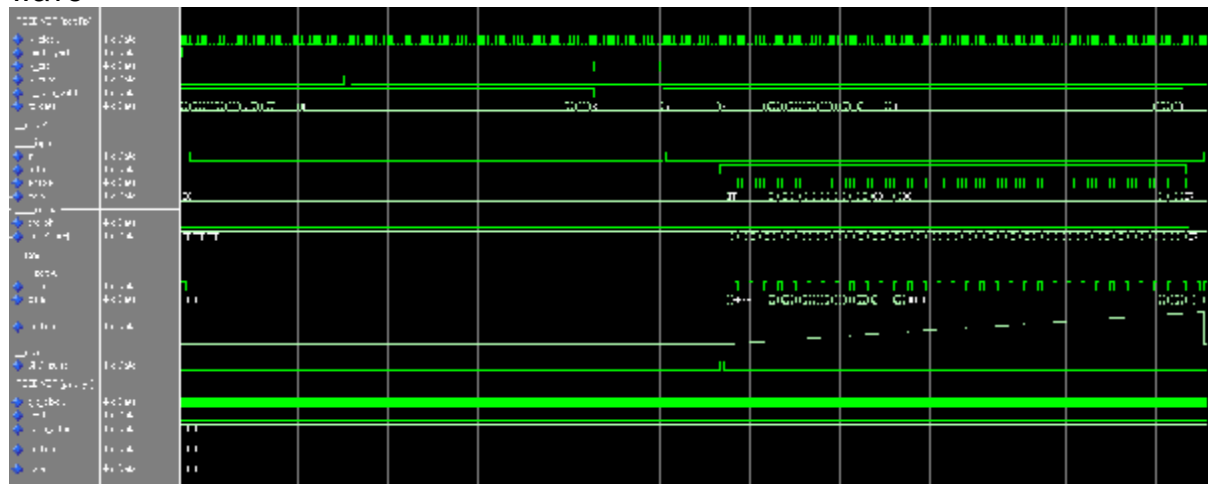
|       |         |                                  |        |      |
|-------|---------|----------------------------------|--------|------|
| 7540  | 5324 ns | => begin of frame                | NC     | NC   |
|       | 640 ns  | => SFD (Start Frame Delimiter)   | NC     | NC   |
|       | 196 ns  | => write first byte of the frame | 001    | FFFF |
|       | 160 ns  | => write next byte of the frame  | 002    | FFFF |
|       | 160 ns  | => write next byte of the frame  | 003    | FFFF |
|       | ...     |                                  | to 01E | ADF  |
| 13176 | 4480 ns | => write first byte of CRC       | 01F    | E625 |
|       | 160 ns  | => write second byte of CRC      | 020    | C4C7 |
|       | 160 ns  | => write empty header            | 021    | 0000 |
| 13576 | 80 ns   | => write frame header            | 000    | 8021 |

Rem:

NC : Not Change

ADF : All Data of Frame

wave



memory

|          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|----------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 00000000 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 |
| 00000010 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 |
| 00000020 | C4C7 | 0000 | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX |
| 00000030 | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX |

### 9.1.2.3 Part: CRC error and multiple Rx errors.

The writing of an invalid frame (CRC error).

It's to check if after the writing of frame, he places correctly the new empty header and correctly writes the error header.

|       |                                        |     |      |
|-------|----------------------------------------|-----|------|
| 20136 | 6480 ns => write empty header          | 022 | 0000 |
|       | 80 ns => write error header: CRC error | 021 | 1001 |

An Rx error during the writing of a valid frame.

It's to check the detection of Rx error and the validation of previous error header.

|       |                                                  |     |      |
|-------|--------------------------------------------------|-----|------|
|       | 5480 ns => write empty header                    | 023 | 0000 |
|       | 80 ns => validation of the previous error header | 021 | 9001 |
| 25736 | 40 ns => write error header: Rx error            | 022 | 2001 |

An Rx error during the writing of a valid frame.

It's to check the detection of a same error, the counter of error and if the error header is correctly places.

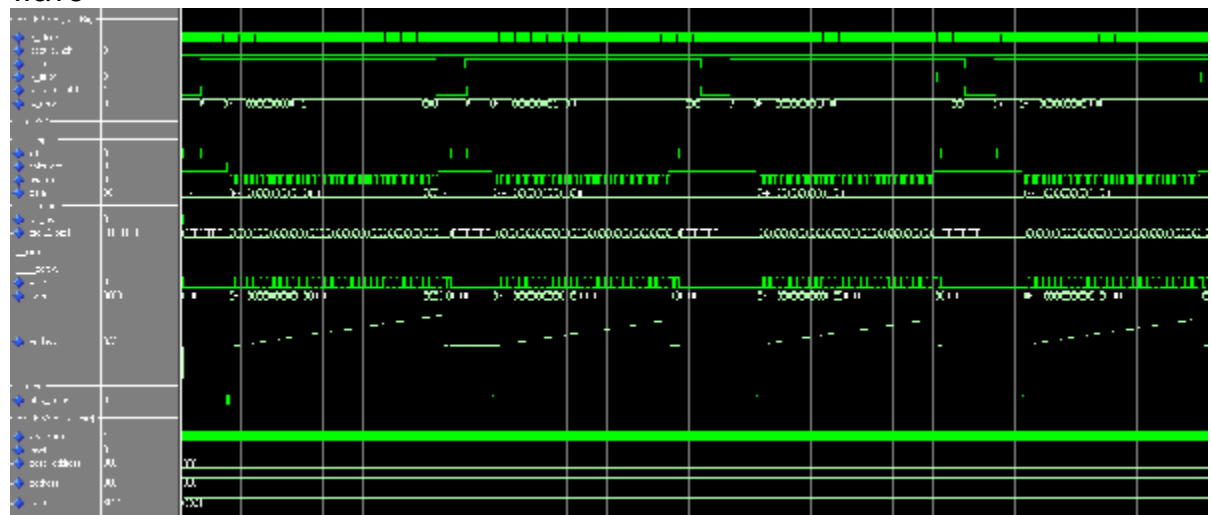
|       |                                               |     |      |
|-------|-----------------------------------------------|-----|------|
| 32216 | 6360 ns => write empty header                 | 023 | 0000 |
|       | 120 ns => write error header: second Rx error | 022 | 2002 |

An Rx error during the writing of a valid frame.

It's to check a second time.

|       |                                              |     |      |
|-------|----------------------------------------------|-----|------|
| 38736 | 6400 ns => write empty header                | 023 | 0000 |
|       | 120 ns => write error header: third Rx error | 022 | 2003 |

wave



memory

|          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|----------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 00000000 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 |
| 00000010 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 |
| 00000020 | C4C7 | 9001 | 2003 | 0000 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 |
| 00000030 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000040 | 1925 | C4C7 | 0000 | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX |

#### 9.1.2.4 Part: Fill the RAM and test the limit of free RAM.

A complete frame is written in RAM to easily see where the pointer of writing is in RAM.

The writing of a valid frame.

It's to check if all data of frame is written in RAM and previous error header are correctly validated.

|       |         |                                            |     |      |
|-------|---------|--------------------------------------------|-----|------|
|       | 7240 ns | => write empty header                      | 044 | 0000 |
|       | 120 ns  | => validation of the previous error header | 022 | A003 |
| 46136 | 40 ns   | => write frame header                      | 023 | 8021 |

At this moment, the test of reader stays locked but the address becomes 47 hex and the base\_address becomes 47 hex in order to fill quickly the buffer and to see if anything is write at base\_address.

The writing of a valid frame.

It's to check detection of full buffer and if the security of 1 RAM line is sufficient.

|       |         |                                                    |     |      |
|-------|---------|----------------------------------------------------|-----|------|
|       | 1480 ns | => write empty header                              | 045 | 0000 |
| 47696 | 80 ns   | => write error header: buffer is full (first time) | 044 | 4001 |

The writing of a valid frame.

It's to check a second time the security of 1 RAM line.

|       |         |                                                     |     |      |
|-------|---------|-----------------------------------------------------|-----|------|
|       | 6240 ns | => write empty header                               | 045 | 0000 |
| 54056 | 120 ns  | => write error header: buffer is full (second time) | 044 | 4002 |

The writing of a valid frame.

It's to check a second time the security of 1 RAM line.

|       |         |                                                    |     |      |
|-------|---------|----------------------------------------------------|-----|------|
|       | 6400 ns | => write empty header                              | 045 | 0000 |
| 60576 | 120 ns  | => write error header: buffer is full (third time) | 044 | 4003 |

The writing of a valid frame.

It's to only to increment error counter.

|       |         |                                                     |     |      |
|-------|---------|-----------------------------------------------------|-----|------|
|       | 6360 ns | => write empty header                               | 045 | 0000 |
| 67056 | 120 ns  | => write error header: buffer is full (fourth time) | 044 | 4004 |

The writing of a valid frame with an Rx error.

It's to check the priority when the buffer is full.

|       |         |                                                    |     |      |
|-------|---------|----------------------------------------------------|-----|------|
|       | 5640 ns | => write empty header                              | 045 | 0000 |
| 72816 | 120 ns  | => write error header: buffer is full (fifth time) | 044 | 4005 |

The writing of a valid frame.

It's to check if this error is considered as another error.

|       |         |                                                    |     |      |
|-------|---------|----------------------------------------------------|-----|------|
|       | 7120 ns | => write empty header                              | 045 | 0000 |
| 80056 | 120 ns  | => write error header: buffer is full (sixth time) | 044 | 4006 |

The writing of a valid frame.

It's to only to increment error counter.

|       |         |                                                      |     |      |
|-------|---------|------------------------------------------------------|-----|------|
| 86576 | 6400 ns | => write empty header                                | 045 | 0000 |
|       | 120 ns  | => write error header: buffer is full (seventh time) | 044 | 4007 |

The writing of a valid frame with an Rx error.

It's to check a second time the priority when the buffer is full.

|       |         |                                                     |     |      |
|-------|---------|-----------------------------------------------------|-----|------|
| 92296 | 5600 ns | => write empty header                               | 045 | 0000 |
|       | 120 ns  | => write error header: buffer is full (eighth time) | 044 | 4008 |

The writing of a valid frame with an Rx error.

It's to check when Rx error is sent 2 times.

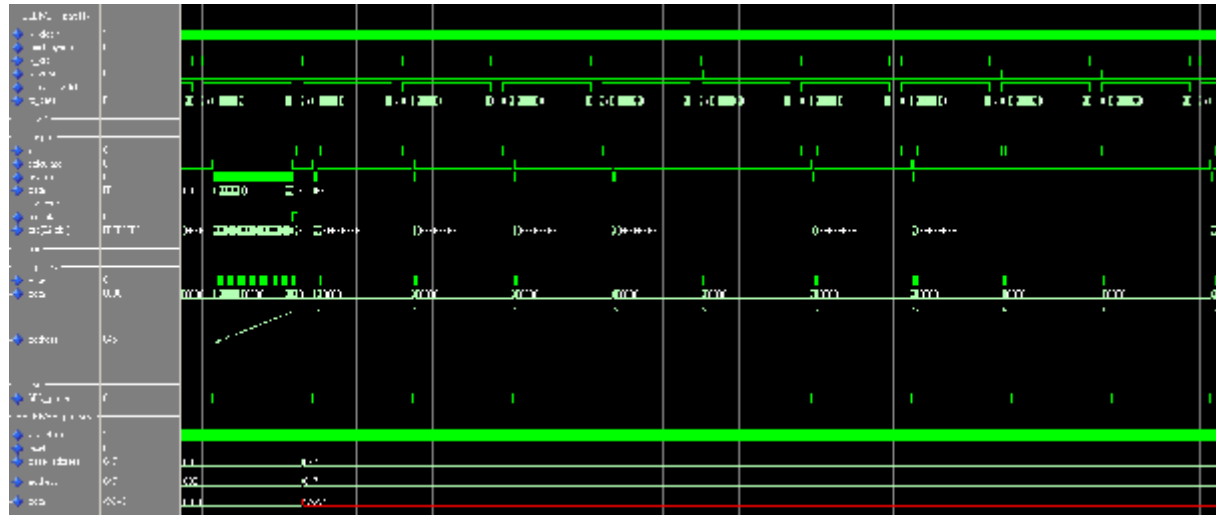
|       |         |                                                    |     |      |
|-------|---------|----------------------------------------------------|-----|------|
| 98816 | 6400 ns | => write empty header                              | 045 | 0000 |
|       | 120 ns  | => write error header: buffer is full (ninth time) | 044 | 4009 |

The writing of a valid frame.

It's to only to increment error counter.

|        |         |                                                    |     |      |
|--------|---------|----------------------------------------------------|-----|------|
| 106056 | 7120 ns | => write empty header                              | 045 | 0000 |
|        | 120 ns  | => write error header: buffer is full (tenth time) | 044 | 400A |

wave



memory

|          |                                                                                 |
|----------|---------------------------------------------------------------------------------|
| 00000000 | 8021 FFFF FFFF FFFF 5341 4501 0203 FA0E 8345 0681 0002 0000 0001 0000 0000 0000 |
| 00000010 | 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 E625 |
| 00000020 | C4C7 9001 A003 8021 FFFF FFFF FFFF 5341 4501 0203 FA0E 8345 0681 0002 0000 0001 |
| 00000030 | 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000      |
| 00000040 | 0000 0000 E625 C4C7 400A 0000 FFFF XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX |



### 9.1.2.5 Part: Fill the RAM and test the limit of free RAM.

This part is different of the previous part because it starts with Rx error and not buffer is full.

At this moment, the test of reader stays locked but the address becomes 49 hex and the base\_address becomes 49 hex in order to fill quickly the buffer and to see if anything is write at base\_address.

The writing of a valid frame with an Rx error.

|        |         |                                            |     |      |
|--------|---------|--------------------------------------------|-----|------|
|        | 5640 ns | => write empty header                      | 046 | 0000 |
|        | 80 ns   | => validation of the previous error header | 044 | C00A |
| 111816 | 40 ns   | => write error header: Rx error            | 045 | 2001 |

The writing of a valid frame with an Rx error.

|        |         |                                        |     |      |
|--------|---------|----------------------------------------|-----|------|
|        | 7080 ns | => write empty header                  | 046 | 0000 |
| 119016 | 120 ns  | => write error header: second Rx error | 045 | 2002 |

The writing of a valid frame.

It's to check detection of full buffer and if the security of 1 RAM line is sufficient.

|        |         |                                                    |     |      |
|--------|---------|----------------------------------------------------|-----|------|
|        | 6320 ns | => write first word of frame                       | 046 | 0000 |
|        | 240 ns  | => write empty header                              | 047 | 0000 |
|        | 80 ns   | => validation of the previous error header         | 045 | A002 |
| 125696 | 40 ns   | => write error header: buffer is full (first time) | 046 | 4001 |

The writing of a valid frame.

|        |         |                                                     |     |      |
|--------|---------|-----------------------------------------------------|-----|------|
|        | 6200 ns | => write empty header                               | 047 | 0000 |
| 132016 | 120 ns  | => write error header: buffer is full (second time) | 046 | 4002 |

The writing of a valid frame with an Rx error.

It's to check the priority when the buffer is full.

|        |         |                                                    |     |      |
|--------|---------|----------------------------------------------------|-----|------|
|        | 5680 ns | => write empty header                              | 047 | 0000 |
| 137816 | 120 ns  | => write error header: buffer is full (third time) | 046 | 4003 |

The writing of a valid frame.

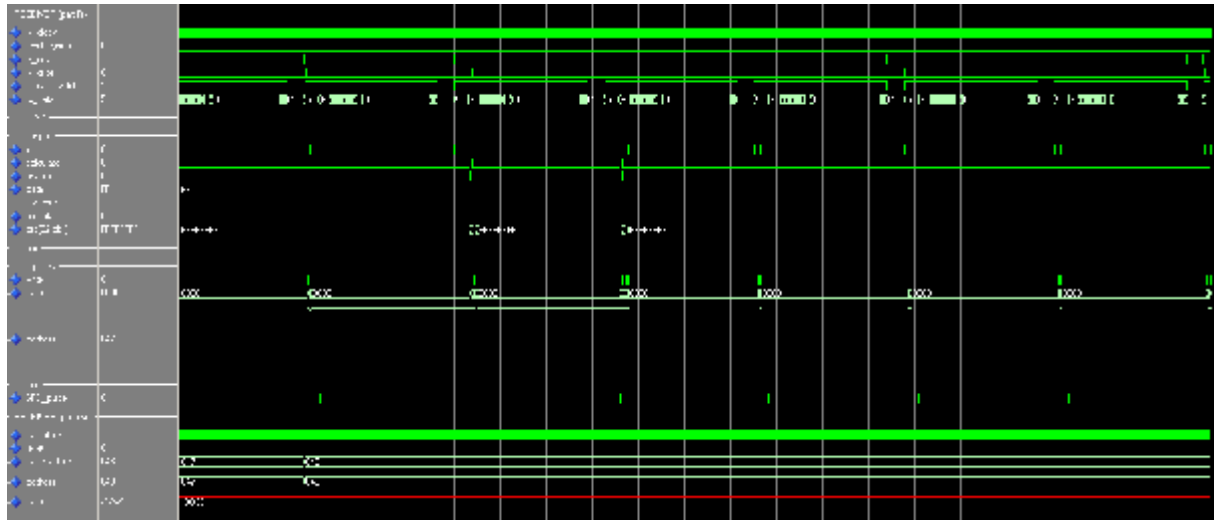
|        |         |                                                     |     |      |
|--------|---------|-----------------------------------------------------|-----|------|
|        | 7080 ns | => write empty header                               | 047 | 0000 |
| 145016 | 120 ns  | => write error header: buffer is full (fourth time) | 046 | 4004 |

The writing of a valid frame with an Rx error.

It's to check the priority when the buffer is full.

|        |         |                                                    |     |      |
|--------|---------|----------------------------------------------------|-----|------|
|        | 5680 ns | => write empty header                              | 047 | 0000 |
| 150816 | 120 ns  | => write error header: buffer is full (fifth time) | 046 | 4005 |

## wave



## memory

|          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|----------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 00000000 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 |
| 00000010 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 |
| 00000020 | C4C7 | 9001 | A003 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 |
| 00000030 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000040 | 0000 | 0000 | E625 | C4C7 | C00A | A002 | 4005 | 0000 | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX |
| 00000050 | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX | XXXX |

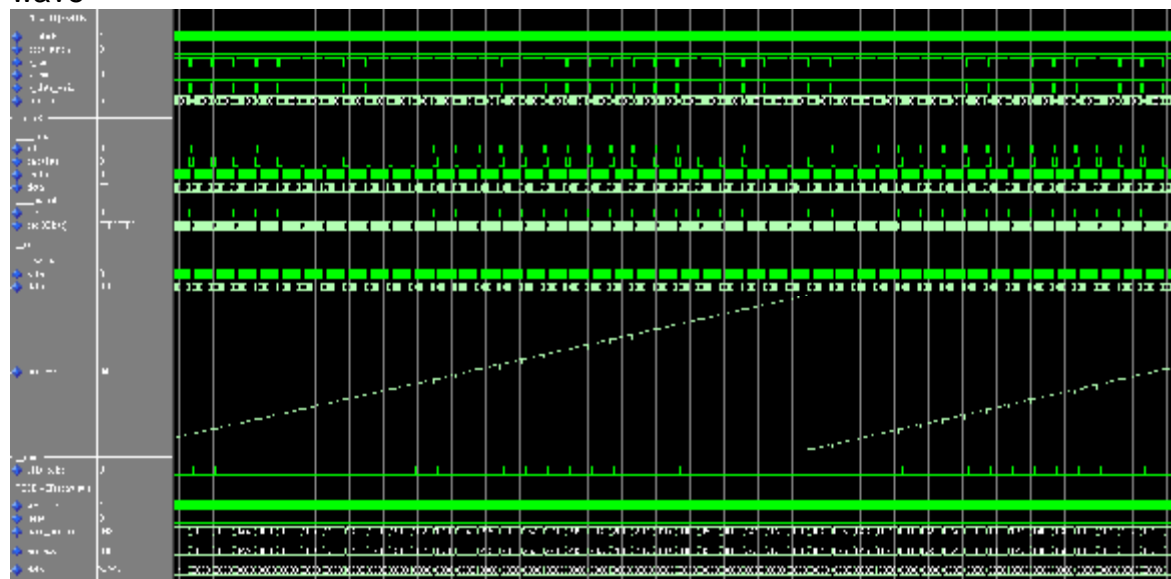
### 9.1.2.6 Part: Test if reader can move correctly.

At this moment, the test of reader becomes unlocked and starts checking of data to place correctly the base\_address at the next reading point.  
And, the base\_address and the address become 0 hex.

Only the same valid frame is sent infinitely.



wave



memory

|          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|----------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 00000000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 |
| 00000010 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000020 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 |
| 00000030 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000040 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 |
| 00000050 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000060 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 |
| 00000070 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000080 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E |
| 00000090 | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000000a0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 |
| 000000b0 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000000c0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 |      |
| 000000d0 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000000e0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 |      |
| 000000f0 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF |      |
| 00000110 | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000120 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF |      |
| 00000130 | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000140 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF |      |
| 00000150 | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000160 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 |      |
| 00000170 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 |
| 00000180 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 |      |
| 00000190 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 |
| 000001a0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 |      |
| 000001b0 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 |
| 000001c0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000001d0 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 |
| 000001e0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000001f0 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 |
| 00000200 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000210 | 0000 | 0000 | E625 | C4C7 | 0000 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 |
| 00000220 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000230 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 |
| 00000240 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000250 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 |
| 00000260 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000270 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E |
| 00000280 | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000290 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 |
| 000002a0 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000002b0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 |      |
| 000002c0 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000002d0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 |      |
| 000002e0 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000002f0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF |      |
| 00000300 | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000310 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF |      |
| 00000320 | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000330 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 | FFFF |      |
| 00000340 | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 00000350 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 | 8021 |      |      |
| 00000360 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 | 0000 |
| 00000370 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 | C4C7 |      |
| 00000380 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 | 0000 |
| 00000390 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | E625 |      |
| 000003a0 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 | 0000 |
| 000003b0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000003c0 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 | 0000 |
| 000003d0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 000003e0 | 0000 | E625 | C4C7 | 8021 | FFFF | FFFF | FFFF | 5341 | 4501 | 0203 | FA0E | 8345 | 0681 | 0002 | 0000 | 0001 |
| 000003f0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

### 9.1.3 Notice

In a first time, only frames of mode: group addressing are sent.  
(It's not necessary to have frames of type: physical addressing to test receiver. When the block: interface Ethernet is connected to the block: controller SimAP, it's necessary to have all possible frames.)

It's possible to have any little change in the timing.

## 9.2 transmitter\_tb

I would not describe this part because I haven't correctly made this part.  
(I have found more bugs when I have used the MAC\_plus\_tb.)

## 9.3 MAC\_plus\_tb

### 9.3.1 Little description of MAC\_plus\_tester\_fas.vhd

A small index : from line 1 to line 20

Definition of all signals : from line 27 to line 160

Clocks signals : from line 168 to line 187

Reset process : from line 191 to line 208

Process to create and to send an Ethernet frame: from line 218 to line 452

Test sequence : from line 456 to line 2363

### 9.3.2 Test sequence

The test sequence is composite by 7 parts:

- Send only writing frames => line: 467
- Send only reading frames => line: 608
- Send only event => line: 701
- Send all frames with not SimAP frame => line: 770
- Send with transmission error => line: 900
- Transmission loss => line: 1069
- Others => line: 1947

#### 9.3.2.1 Part: Send only writing frames

No event for this part (no buttons).

Reset and wait end of reset before send anything.

Sent frames:

- configure the device
- write dimming
- write led0 switching on
- write led0 switching off
- write led0 switching on
- write led0 switching on
- write led1 switching on

### 9.3.2.2 Part: Send only reading frames

No event for this part (no buttons).

No reset because this part uses the previous information.

Sent frames:

- read the configuration of the device
- read dimming
- read led0 switching
- read led1 switching

### 9.3.2.3 Part: Send only event

See the design of application\_test because, it's special.

No reset because this part uses the configuration of the device.

Buttons pressed:

- all in same time => 1 frame is transmitted
- all 1 by 1 => 3 frames are transmitted (two buttons have the same event number)
- button 0 a long time => 1 frame is transmitted
- button 2 => see effect of full Tx RAM

### 9.3.2.4 Part: Send all frames with not SimAP frame

Init buttons positions (all to low).

Reset and not wait end of reset before send anything.

Sent frames and buttons pressed:

- a frame with false SFD (Start Frame Delimiter)
- button 0
- configure the device with false SFD and with false destination address
- other Ethernet Type
- other address type
- button 2



#### **9.3.2.5 Part: Send with transmission error**

No event for this part (no buttons).

Reset and not wait end of reset before send anything.

Sent frames:

- configure the device
- write dimming
- read dimming
- write led0 switching on with wrong CRC
- write led0 switching on with rx error
- write led0 switching on with rx error
- write led0 switching on
- read led0 switching
- read led0 switching with rx error
- read led0 switching with a biggest size

#### **9.3.2.6 Part: Transmission loss**

Init buttons positions (all to low).

Reset and wait end of reset before send anything.

In a first time a lot of frames are sent with a lot of events.

In a second time a lot of frames are sent.

In two times, it's possible to change the length of the frame, the number of the frames and (only for the first time) the number of times that the button 2 stays at high.

#### **9.3.2.7 Part: Others**

Init buttons positions (all to low).

Reset and wait end of reset before send anything.

Sent frames:

- configure the device with an offset
- write anything in RAM of uP with offset
- send a fluid situation (from line 2056 to line 2193)
- an infinite loop (from line 2199 to line 2361)

## 10 TEST OF DEMO

### 10.1 Equipment

From Planet: [www.planet.com.tw](http://www.planet.com.tw)

A Ethernet switch PoE: FSD-804P

A PoE splitter 5V: POE-151S-5V

Software: Ethereal or WireShark (To parse all frames)

From school

2 boards: FPGA-EBS

2 boards: HEB\_ETHERNET\_PHY

2 boards: HEB\_LCD\_I2C

(It's 2 devices. When the design is loaded in FPGA, it's 2 SDs).

With software, frames are created and sent.

### 10.2 Notice

This test is made with a switch because it must work with mode: full duplex (no CRS and no COL).

## 11 IMPROVEMENTS TO MAKE

### 11.1 receiver

- to standardize the streaming (nibble -> byte -> word) so that this be more easy to use the GBits Ethernet.
- to add a system to know the number of valid byte at the last line of structure in Rx RAM (and that must be correct with generic RAM).
- to update to work with the mode: "half-duplex" => with HUB (CRS, COL, ...)
- to manage the signals: PRIO and ACK
- to not store the part: CRC in Rx RAM
- to add generic value so to know the limits of the used Ethernet frame and to add the checking of these limits.
- 

### 11.2 transmitter

- to standardize the streaming (nibble -> byte -> word) so that this be more easy to use the GBits Ethernet.
- to add a system to know the number of valid byte at the last line of structure in Rx RAM (and that must be correct with generic RAM).
- to update to work with the mode: "half-duplex" => with HUB
  - => to check CRS before to send a frame.
  - => to send JAM when a collision is detected.
  - => to wait the timing who be calculate with the rule: "Binary exponential Backoff" before to resend a frame.
  - => to update RTRY when a frame is resent.
  - => ...
- to manage the signals: PRIO and ACK
- to add generic value so to know the limits of the used Ethernet frame and to add the checking of these limits.
- to manage error header to use the signal: "Tx error".
- (to add the PAD when the frame is smaller as the min Ethernet frame
  - => I think:"This is not his job.".)
-

### 11.3 controller

- to use the MAC address stored in the parameters registers block.  
(Currently, the MAC address is wrote in design.)
- to add a system to know the number of valid byte at the last line of structure in Rx RAM (and that must be correct with generic RAM).
- to manage PRIO and ACK (and the management to resend a previous frame).
- to define if a device can have more one resister with the same GID.  
(Currently, only the first found GID is used.)
- to define the constants in lib to remove the values in design.  
=> code more readable.
- to check part to enable or disable (communication, read, write, transmit, update).
- to update the block to easily modify generics.
- to parse and to use a frame in physical addressing mode with EIB mode.
- to check all possible data in all field of an frame.
- 

### 11.4 memories

To make a discussion about all memories because I haven't correctly understood the documentation: SimAP.

- to choose which memories are in the block: "controller\_SimAP " or which are in the block: "application\_SimAP\_example".
- to define were are the information to define a GID (EIS, ECMD, PRIO, ACK, ...).
- to add a memory block to enable or to disable (communication, read, write, transmit, update).
- to define how use the program memory block.
- to develop a system to work with offset and various sizes of RAM.
- to update memories types => initializable RAM => EEPROM
- to define the result on memories block after a reset  
(to restore the default value => MAC address, ... ).
- 

### 11.5 application

- to add all registers to test all possible data define in SimAP.
- to modify the block: "application\_test" to use all possible data.
-

## 11.6 test

- to add at the receiver test bench all possible Ethernet frames.
- to make a new transmitter test bench.
- to make test for the reception and the transmission of a string.
- to make test for the reception and the transmission of a register > 8 bits.
- 

## 11.7 others

- to create a generic to enable or disable the debug mode.
- to update the doc: SimAP (before make that, see part: 11.4 memories).
- (to add an auto reset for all RAM to be sure to do not stay with a full RAM.)
- 

## 11.8 misc

The small frames are favoured by the checker of free space in RAM.

A GID has only a data type.

An event number must correspond at the address of all registers about the correspond GID.

## 12 WHY THESE CHOICES ?

The device doesn't work in mode: "half duplex".

The big problem is when the transmitter sends a frame and receives the signal of a collision.

He must to send JAM (This is 32 bits send for all devices can detect the collision.).  
He must to calculate the waiting time before to resend the frame (To calculate this waiting time, it is necessary to use the rule: "Binary exponential Backoff").

So the interface works in mode: "full duplex" to simplify the management.  
In this case, the signals: CRS and COL aren't used.

To have a simple management, this interface doesn't work with neither the priority of a frame and nor acknowledge.

### 12.1 INTERFACE ETHERNET

Currently, the interface only works correctly with frames that have an even number of bytes.

(Because at the beginning, I haven't created the header with an indication of the number of the valid bytes at the last line of the frame in the RAM.)

#### 12.1.1 receiver

The receiver doesn't check the length of received frame because I'm not sure that is only used with an Ethernet frame without extension.

##### 12.1.1.1 crc32

The Ethernet frame length is 64 to 1'518 bytes.

And because that, if one takes 16 bits to 16 bits (or more), it is necessary to have signals to indicate how many bits are valid.

So to have the signals minimum, the data length of the block: crc32 is 8 bits.

This is why, the block: data\_to\_word was created.

(He assembles two received nibbles (4 bits) to have a word of 8 bits).

### 12.1.1.2 RAM dp

There is a RAM dp for 2 reasons.

The first, as soon as the data are stored, it's possible to take the received data any time, in any order and many times.

The second, it is of more easily being able to coordinate the data between the Ethernet interface and the SimAP controller. The RAM makes it possible to synchronize the two clocks (rx\_clock and sys\_clock) and there is not many signals to drive communication, only a specific structure (see part: 14.3 RAM structure).

Because the maximum length of an Ethernet frame is 1'518 bytes (12'144 bits) and the RAM width is 16 bits (choice to have a sufficient gap in order to write the header data), the maximum space taken in RAM is 759 lines.

With the valid header and the empty header, there is 761 occupied lines.

So it's necessary to have 10 bits to write the length of a frame and 10 bits, it's too the minimum number of RAM address (with 10 bits, there is 1'024 lines in RAM).

In this case, the data length is 16 bits and the address length is 10 bits.

This is why, the block: rc\_data\_register was created.

(He delays a word of 8 bits so that it is assembled with current word of 8 bits to have a data of 16 bits).

As it's not possible (I haven't found) to init a RAM dual port for a FPGA: Spartan II, the block: mux\_ram\_data was created.

(He makes it possible to mislead the reader so that it reads an empty header during the RAM initialization.)

### 12.1.1.3 receiver controller

To not have a too complicated schema, all blocks to drive the receiver are in receiver controller.

The block: rc\_heart coordinates all others blocks in the block: receiver. So to not have memory elements in the heart of the receiver, the blocks: rc\_data\_registers, rc\_address\_registers and rc\_errors\_registers were created.

The block: rc\_heart has only time: interframe gap to build the RAM structure. Because it's the min time during which the data stream is stopped.

Notice: interframe gap (from wikipedia)

Ethernet devices must allow a minimum idle period between transmission of Ethernet frames known as the interframe gap (IFG) or interpacket gap (IPG). It provides a brief recovery time between frames to allow devices to prepare for reception of the next frame. The minimum interframe gap is 96 bit times (the time it takes to transmit 96 bits of raw data on the medium), which is 9.6  $\mu$ s for 10 Mbit/s Ethernet, 960 ns for 100 Mbit/s (fast) Ethernet, and 96 ns for 1 Gbit/s (gigabit) Ethernet.

In this specific case, the block: interface\_Ethernet is connected with the LXT972A that is an IEEE compliant Fast Ethernet PHY Transceiver.

He transmits nibble by nibble (4 bits by 4 bits) each clock blow. And because the minimum interframe gap is 96 bits, it's possible to build the RAM structure of the received frame during 24 clock bows.



### 12.1.2 transmitter

The transmitter is similar to the receiver (The main difference is the direction of data stream).

The transmitter doesn't create the padding of transmitted frame because I'm not sure that is only used with an Ethernet frame without extension.

#### 12.1.2.1 transmitter controller

To not have a too complicated schema, all blocks to drive the transmitter are in transmitter controller.

The block: tc\_heart coordinates all others blocks in the block: transmitter. So to not have memory elements in the heart of the transmitter, the block: tc\_address\_registers was created.

The block: tc\_IFG\_counter is used to create the interframe gap timing.

In this specific case, the block: interface\_Ethernet is connected with the LXT972A that is an IEEE compliant Fast Ethernet PHY Transceiver.

The transmitter sends nibble by nibble (4 bits by 4 bits) each clock blow. And he waits the minimum interframe gap (96 bits) before sends another frame.

## 12.2 CONTROLLER SIMAP

The controller SimAP is connected with others parts as RAMs.

To not have a too complicated schema, all blocks to drive the controller SimAP are in controller.

The block: controller coordinates all others blocks in the block: controller SimAP. So to not have memory elements in the controller of the controller SimAP, the blocks: reader\_address\_registers, writer\_address\_registers, application\_address\_registers, selector\_registers checker and work\_register were created.

For the first version, the config memory is included in the block: controller\_SimAP because this memory isn't used in the block: application\_SimAP\_example. But he doesn't used a part of config memory

### 12.2.1 controller

For the first version:

- the controller works only with the MAC address which is implemented in design.
- he works without constants.
- he checks only the minimum part of SimAP field to use or not the frame.  
(Look below, the part: 13.2.1.1parsing.)
- he doesn't work with a physical addressing frame when the EIB mode is selected.  
(Because, I don't know how use this frame.)
- he doesn't resend a frame even if she wasn't sent.
- he doesn't work with the part of config to enable or to disable (communication, read, write, transmit and update).

### 12.2.1.1 parsing

1) Read the line header in the Rx RAM.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

header

2) Wait on the flag: "header valid"

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

V

3) Check error name

a) if the frame hasn't an error then  
 to continue the parsing.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1 no error - - frame length +1

b) if the frame has an error then to  
 go to the next line to read the next  
 header

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1 error - - frame number with this error  
 next header

Ethernet field

Data

FCS

next header

#### 4) Check Ether Type

a) if the frame is a SAoE frame then to continue the parsing.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1 0 0 0 - - frame length +1  
 Destination MAC Address (part 2)  
 Destination MAC Address (part 1)  
 Destination MAC Address (part 0)  
 Source MAC Address (part 2)  
 Source MAC Address (part 1)  
 Source MAC Address (part 0)

0xFA0E

SAoE Field

b) if the frame is not a SAoE frame then to go to the next header

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1 0 0 0 - - frame length +1  
 Destination MAC Address (part 2)  
 Destination MAC Address (part 1)  
 Destination MAC Address (part 0)  
 Source MAC Address (part 2)  
 Source MAC Address (part 1)  
 Source MAC Address (part 0)

EtherType

Data

SAoE Data

FCS

next header

FCS

next header

## Contents of SAoE field.

|               |               |
|---------------|---------------|
| PCF           | GID (part 1)  |
| GID (part 0)  | NPCI (part 1) |
| NPCI (part 0) | TPCI          |
| APCI (part 1) |               |
| APCI (part 0) |               |

## 5) Check addressing mode in part PCF

AM - - - - R PRIO GID (part 1)  
 GID (part 0) NPCI (part 1)  
 NPCI (part 0) TPCI  
 APCI (part 1)  
 APCI (part 0)

If AM = 0 then the frame is in physical addressing mode  
 else (AM = 1) the frame is in group addressing mode.

At this moment, the parsing changes with the addressing mode.

### 12.2.1.1.1 *physical addressing mode*

#### 6) Check destination address

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1 0 0 0 - - frame length +1

|      |             |
|------|-------------|
| 0x53 | 0x41        |
| 0x45 | 0x00 area   |
| 0x00 | line device |

Source Address (part 2)

Source Address (part 1)

Source Address (part 0)

EtherType => SAoE

0 - - - - R PRIO GID (part 1)  
 GID (part 0) 0 PM ACK size (part 1)  
 size (part 0) TPCI  
 RBSEL RBOFF  
 RBOP RBSIZE

## 7) Check protocol mode

|   |   |   |   |   |   |      |               |               |
|---|---|---|---|---|---|------|---------------|---------------|
| 0 | - | - | - | - | R | PRI0 |               | GID (part 1)  |
|   |   |   |   |   |   |      | GID (part 0)  | 0             |
|   |   |   |   |   |   |      | size (part 0) | PM            |
|   |   |   |   |   |   |      |               | ACK           |
|   |   |   |   |   |   |      |               | size (part 1) |
|   |   |   |   |   |   |      |               | TPCI          |
|   |   |   |   |   |   |      | RBSEL         | RBOFF         |
|   |   |   |   |   |   |      | RBOP          | RBSIZE        |

### a) if PM = 1 then 8) check TPCI.

|   |   |   |   |   |   |      |               |               |
|---|---|---|---|---|---|------|---------------|---------------|
| 0 | - | - | - | - | R | PRI0 |               | GID (part 1)  |
|   |   |   |   |   |   |      | GID (part 0)  | 0             |
|   |   |   |   |   |   |      | size (part 0) | 1             |
|   |   |   |   |   |   |      |               | ACK           |
|   |   |   |   |   |   |      |               | size (part 1) |
|   |   |   |   |   |   |      |               | TPCI          |
|   |   |   |   |   |   |      | RBSEL         | RBOFF         |
|   |   |   |   |   |   |      | RBOP          | RBSIZE        |

### b) if PM = 0 then to go to the next header

|   |   |   |   |   |   |      |        |      |              |
|---|---|---|---|---|---|------|--------|------|--------------|
| 0 | - | - | - | - | R | PRI0 |        | main | intermediate |
|   |   |   |   |   |   |      | groupe | 0    | 0            |
|   |   |   |   |   |   |      |        | ACK  | -            |
|   |   |   |   |   |   |      |        | TPCI | -            |
|   |   |   |   |   |   |      | EIS    | -    | -            |
|   |   |   |   |   |   |      | ECMD   | -    | -            |

### a) if TPCI = 0x00 then to get size.

|   |   |   |   |   |   |      |               |               |
|---|---|---|---|---|---|------|---------------|---------------|
| 0 | - | - | - | - | R | PRI0 |               | GID (part 1)  |
|   |   |   |   |   |   |      | GID (part 0)  | 0             |
|   |   |   |   |   |   |      | size (part 0) | 1             |
|   |   |   |   |   |   |      |               | ACK           |
|   |   |   |   |   |   |      |               | size (part 1) |
|   |   |   |   |   |   |      |               | 0x00          |
|   |   |   |   |   |   |      | RBSEL         | RBOFF         |
|   |   |   |   |   |   |      | RBOP          | RBSIZE        |

### b) if TPCI != 0x00 then to go to the next header

|   |   |   |   |   |   |      |               |               |
|---|---|---|---|---|---|------|---------------|---------------|
| 0 | - | - | - | - | R | PRI0 |               | GID (part 1)  |
|   |   |   |   |   |   |      | GID (part 0)  | 0             |
|   |   |   |   |   |   |      | size (part 0) | 1             |
|   |   |   |   |   |   |      |               | ACK           |
|   |   |   |   |   |   |      |               | size (part 1) |
|   |   |   |   |   |   |      |               | TPCI          |
|   |   |   |   |   |   |      | RBSEL         | RBOFF         |
|   |   |   |   |   |   |      | RBOP          | RBSIZE        |

## 9) Check the selected register and get the offset of registers.

|   |   |   |   |   |   |      |               |               |
|---|---|---|---|---|---|------|---------------|---------------|
| 0 | - | - | - | - | R | PRI0 |               | GID (part 1)  |
|   |   |   |   |   |   |      | GID (part 0)  | 0             |
|   |   |   |   |   |   |      | size (part 0) | 1             |
|   |   |   |   |   |   |      |               | ACK           |
|   |   |   |   |   |   |      |               | size (part 1) |
|   |   |   |   |   |   |      |               | 0x00          |
|   |   |   |   |   |   |      | RBSEL         | RBOFF         |
|   |   |   |   |   |   |      | RBOP          | RBSIZE        |

## 10) Check the selected operation and get the size of data.

|   |   |   |   |   |   |      |               |               |
|---|---|---|---|---|---|------|---------------|---------------|
| 0 | - | - | - | - | R | PRI0 |               | GID (part 1)  |
|   |   |   |   |   |   |      | GID (part 0)  | 0             |
|   |   |   |   |   |   |      | size (part 0) | 1             |
|   |   |   |   |   |   |      |               | ACK           |
|   |   |   |   |   |   |      |               | size (part 1) |
|   |   |   |   |   |   |      |               | 0x00          |
|   |   |   |   |   |   |      | RBSEL         | RBOFF         |
|   |   |   |   |   |   |      | RBOP          | RBSIZE        |

End of parsing



## 12.3 APPLICATION

The application\_SimAP\_example receives and transmits data as a RAM with her selector.

He sends an event when the signal: "get\_next\_event" is high.

This part is not too complicate because it's only a basic example to make test and to have a small demo.

### 12.3.1 mux and demux

They are here so that the controller\_SimAP see only a RAM with her selector.

### 12.3.2 blocks: ROM registers, params registers, program memory, groupe value registers, groupe description registers

For this part, I have chosen blocks with registers or a RAM to display a part of memories that be can used.

Each of these blocks has only a writer (the controller\_SimAP or the application\_test).

### 12.3.3 application\_test

This block has a simple connectivity between registers and IOs.

He assigns an event number at a state of an input.

### 12.3.4 event\_registers

The tail is used to store the event until be called.

The tail\_contents is used to not write a second time the same event if it has not been again called.



## 13 APPENDICES

### 13.1 Abbreviation

|       |                                          |
|-------|------------------------------------------|
| APCI  | Application Protocol Control Information |
| CRC   | Cyclic Redundancy Check                  |
| ESD   | End of Stream Delimiter                  |
| ET    | Ethertype                                |
| FCS   | Frame Check Sequence                     |
| GID   | Groupe ID                                |
| ID    | IDentifier                               |
| IFG   | Inter-frame gap                          |
| IP    | Internet Protocol                        |
| MAC   | Medium Access Control                    |
| MII   | Media Independent Interface              |
| NPCI  | Network Protocol Control Information     |
| PAD   |                                          |
| PCF   | Packet Control Field                     |
| PHY   | PHYsical layer entity sublayer           |
| SAoE  | Simple Automation over Ethernet          |
| SAoIP | Simple Automation over IP                |
| SD    | SAoE Device                              |
| SFD   | Start-of-Frame Delimiter                 |
| SimAP | Simple Automation Protocol               |
| SNB   | SAoE/SAoIP Network Bridge                |
| SPDA  | Shared Physical Destination Address      |
| SPSA  | Shared Physical Source Address           |
| SSD   | Start-of-Stream Delimiter                |
| TPCI  | Transport Protocol Control Information   |

## 13.2 Files and folders for the project

The folder where all information about this project must be store:

I:\Institut\Projets\MAC

The folder to find information about the board with PHY (Intel chip: LXT972A):

P:\PCB\Produit\Logiciel\FPGA-EBS-ETH

### 13.3 RAM structure

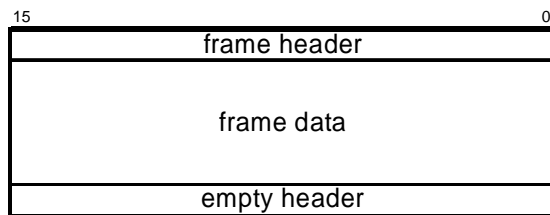
To read data in RAM, there are specific headers.

A header indicates when he is valid, when there's an error and where are the beginning and the end.

There are 3 headers:

- § empty header
- § frame header
- § error header

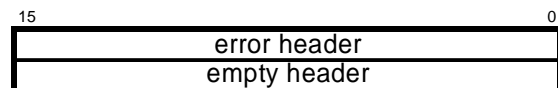
#### 13.3.1 Structure



If it's a frame valid:

1. frame header
2. frame data
3. empty header

By adding the length from frame header to address of frame header, one obtains address of empty header.



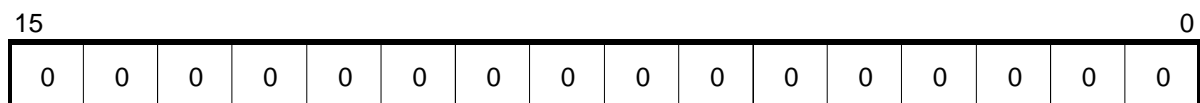
If an error is occurred:

1. error header
2. empty header

To have address of empty header, it is necessary to add 1 to address of error header.

### 13.3.2 Header

#### 13.3.2.1 Empty header



### 13.3.2.2 Error header

|    |            |    |   |              |
|----|------------|----|---|--------------|
| 15 | 14         | 12 | 9 | 0            |
| HV | error name | 0  | 0 | error number |

HV: Header Valid

Error name:

§ "000" : nothing

§ "001" : CRC error

§ "010" : Rx error

§ "100" : buffer is full

### 13.3.2.3 Frame header

|    |   |        |
|----|---|--------|
| 15 | 9 | 0      |
| HV | 0 | length |

HV: Header Valid

## 13.4 Abreviations tirée du 802.3

10P label to indicate “pertains to 10PASS-TS port-type”  
10P/2B label to indicate “pertains to 10PASS-TS and 2BASE-TL port-types”  
2B label to indicate “pertains to 2BASE-TL port-type”  
2-PAM two level pulse amplitude modulation  
8802-3 ISO/IEC 8802-3 (IEEE Std 802.3)  
8802-5 ISO/IEC 8802-5 (IEEE Std 802.5)  
AIS Alarm Indication Signal  
ANSI American National Standards Institute  
ASIC application-specific integrated circuit  
ASN.1 abstract syntax notation one as defined in ISO/IEC 8824: 1990  
AUI attachment unit interface  
BER bit error ratio  
BERT Bit Error Ratio Tester  
BIP Bit Interleaved Parity  
BPSK binary phase shift keying  
BR bit rate  
BT bit time  
CAT3 Category 3 balanced cable  
CAT4 Category 4 balanced cable  
CAT5 Category 5 balanced cable  
CD0 clocked data zero  
CD1 clocked data one  
CDR clock and data recovery  
CID Consecutive Identical Digit  
CJPAT continuous jitter test pattern  
CMIP common management information protocol as defined in ISO/IEC 9596-1: 1991  
CMIS common management information service as defined in ISO/IEC 9595: 1991  
CMOS complimentary metal oxide semiconductor  
CO central office  
CPE customer premises equipment  
CPR coupled power ratio  
CRC cyclic redundancy check  
CRPAT continuous random test pattern  
CRV code rule violation  
CS0 control signal zero  
CS1 control signal one  
CVH clocked violation high  
CVL clocked violation low  
CW continuous wave  
DA destination address  
DCD duty cycle distortion

DDJ data dependent jitter  
DFE decision feedback equalizer  
DJ deterministic jitter  
DMT discrete multi-tone  
DSL digital subscriber line  
DTE data terminal equipment  
EFM Ethernet in the first mile  
EIA Electronic Industries Association  
ELFEXT equal-level far-end crosstalk  
EMI Electromagnetic Interference  
EPD End\_of\_Packet delimiter  
ERDI Enhanced Remote Defect Indication  
FC-PH Fibre Channel—Physical and Signaling Interface  
FDDI fibre distributed data interface  
FEC forward error correction  
FEXT far-end crosstalk  
FIFO first in, first out  
FLP fast link pulse  
FOIRL fiber optic inter-repeater link  
FOMAU fiber optic medium attachment unit  
FOMDI fiber optic medium dependent interface  
FOPMA fiber optic physical medium attachment  
FOTP fiber optic test procedure  
FSW frame synchronization word  
GMII Gigabit Media Independent Interface  
HH header hub  
IB indicator bits  
IEC International Electrotechnical Commission  
IH intermediate hub  
IPG inter-packet gap  
IRL inter-repeater link  
ISI penalty intersymbol interference penalty  
ISO International Organization for Standardization  
LACP Link Aggregation Control Protocol  
LACPDU Link Aggregation Control Protocol Data Unit  
LAG Link Aggregation Group  
LAG ID Link Aggregation Group Identifier  
LAN local area network  
LCD Loss Of Code-Group Delineation  
LLC logical link control  
LLID logical link identifier  
LOF Loss Of Framing  
LOP Loss Of Pointer  
LOS Loss Of Signal  
LSDV link segment delay value  
LT line termination

---

LVDS Low-Voltage Differential Signals  
MAN Metropolitan Area Network  
MAU medium attachment unit  
MC message code  
MDELFEXT multiple-disturber equal-level far-end crosstalk  
MDFEXT multiple-disturber far-end crosstalk  
MDI medium dependent interface  
MDIO management data input/output  
MDNEXT multiple-disturber near-end crosstalk  
MIB management information base  
MMD MDIO Manageable Device  
MMF multimode fiber  
MP message page  
MPCP multipoint control protocol  
MPS Maintain Power Signature  
Mux Multiplexer  
NEXT Near-end Crosstalk  
NLP normal link pulse  
NPA next page algorithm  
NRZI non return to zero and invert on ones  
NT network termination  
NTT Need To Transmit  
OAM operations, administration, and maintenance  
OAMPDU operations, administration, and maintenance protocol  
ODN optical distribution network  
OFL overfilled launch  
OFSTP optical fiber system test procedure  
OH overhead  
OIF Optical Internetworking Forum  
OLT optical line terminal  
OMA Optical Modulation Amplitude  
ONU optical network unit  
ORLT optical return loss tolerance  
P2MP point to multipoint  
P2P point to point  
P2PE point-to-point emulation  
PAF PME aggregation function  
PAM pulse amplitude modulation  
PCB printed circuit board  
PCS physical coding sublayer  
PD Powered Device  
PDU Protocol Data Unit  
PDV path delay value  
PI Power Interface  
PICS protocol implementation conformance statement  
PIPO parallel in parallel out

PISO parallel in serial out  
pk-pk peak-to-peak  
PLL phase locked loop  
PLM Path Label Mismatch  
PLS physical signaling sublayer  
PMA physical medium attachment  
PMD physical medium dependent  
PME physical medium entity  
PMI physical medium independent  
PMS-TC physical media specific - transmission convergence  
ppd peak-to-peak differential  
PRBS pseudo random bit sequence  
PSD power spectral density  
PSE Power Sourcing Equipment  
PVV path variability value  
RD running disparity  
REI Remote Error Indication  
RFI radio frequency interference  
RIN relative intensity noise  
RJ random jitter  
ROFL radial overfilled launch  
RS reconciliation sublayer  
SA source address  
SDH Synchronous Digital Hierarchy  
SDV segment delay value  
SEF Severely Errored Frame  
SELV Safety Extra Low Voltage  
SERDES serializer and deserializer circuit  
SES Severely Errored Second  
SHDSL single-pair high-speed digital subscriber line  
SIPO serial in parallel out  
SMF single-mode fiber  
SMSR side mode suppression ratio  
SNR signal-to-noise ratio  
SONET Synchronous Optical Network  
SPD Start\_of\_Packet delimiter  
SPE Synchronous Payload Envelope  
SR symbol rate  
ST symbol time  
STA station management entity  
STP shielded twisted pair (copper)  
STS Synchronous Transport Signal  
SVV segment variability value  
TBI Ten-Bit Interface  
TC transmission convergence  
TCM trellis coded modulation



TDR time domain reflectometer  
TIA Telecommunications Industry Association  
TLV Type/Length/Value  
TPS-TC transport protocol specific transmission convergence sublayer  
TSS Test Signal Structure  
UCT unconditional transition  
UI unit interval  
UP unformatted page  
UPBO upstream power backoff  
UTP unshielded twisted pair  
VC Virtual Container  
VDSL very high speed digital subscriber line  
VLAN Virtual Bridged Local Area Network (see IEEE P802.1Q)  
VTU VDSL transceiver unit  
VTU-O VTU at the central office end  
VTU-R VTU at the remote end  
WAN Wide Area Network  
WCMB worst-case modal bandwidth  
WIS WAN Interface Sublayer  
WWDM wide wavelength division multiplexing  
XAUI 10 Gigabit Attachment Unit Interface  
xDSL generic term covering the family of all DSL technologies  
XGMII 10 Gigabit Media Independent Interface  
XGXS XGMII Extender Sublayer  
XS Extender Sublayer  
XSBI 10 Gigabit Sixteen-Bit Interface

## **Appendix 8: Time and frequency synchronization**



# Acknowledgements

The research described in this thesis has been accomplished with the support from many people – mentors, colleagues - whom we would like to fully acknowledge. Your support, ideas, comments and energy were essential in allowing them to successfully complete this thesis.

Patrick ARLOS, supervisor of the thesis. We would like to thank you for the time and energy you invested. You guided them through our first steps in electrical research, your advices and ideas were useful to complete this project. Your drive is contagious. We could not have wished for a better supervisor.

Oliver A. GUBLER, former graduate student. Your contribution helps them to solve specific issues we had in VHDL programming. Thank you for your time and interest for our work.

Jörgen ANDERSSON, universitetadjunkt. Your explanation guided them to understand VHDL language. Thanks also always for being available when we needed your help.

Karoliina HÄKKINEN, International coordinator exchange student. You warmly welcomed them in Blekinge Institute of Technology and always been friendly and kind with them.



## *Table of contents*

---

|              |                                          |           |
|--------------|------------------------------------------|-----------|
| <b>1.</b>    | <b>Introduction</b>                      | <b>5</b>  |
| <b>1.1</b>   | <b>Global Positioning System</b>         | <b>6</b>  |
| <b>1.2</b>   | <b>Field of Programmable Gate Arrays</b> | <b>8</b>  |
| <b>1.3</b>   | <b>Other components</b>                  | <b>10</b> |
| <b>1.3.1</b> | <b>RS-422</b>                            | <b>10</b> |
| <b>1.3.2</b> | <b>RJ45</b>                              | <b>11</b> |
| <b>1.3.3</b> | <b>Edge connector</b>                    | <b>11</b> |
| <b>1.3.4</b> | <b>DB-25 connector</b>                   | <b>12</b> |
| <b>1.4</b>   | <b>VHDL</b>                              | <b>13</b> |
| <b>1.5</b>   | <b>UART</b>                              | <b>14</b> |
| <br>         |                                          |           |
| <b>2</b>     | <b>Design</b>                            | <b>16</b> |
| <b>2.1</b>   | <b>Functional description</b>            | <b>17</b> |
| <b>2.2</b>   | <b>Hardware design</b>                   | <b>19</b> |
| <b>2.3</b>   | <b>Software design</b>                   | <b>22</b> |
| <b>2.3.1</b> | <b>Master</b>                            | <b>22</b> |
| <b>2.3.2</b> | <b>Slave</b>                             | <b>27</b> |
| <b>2.3.3</b> | <b>Client</b>                            | <b>29</b> |
| <b>3</b>     | <b>Implementation</b>                    | <b>33</b> |
| <b>3.1</b>   | <b>Hardware implementation</b>           | <b>33</b> |
| <b>3.2</b>   | <b>Software implementation</b>           | <b>35</b> |
| <br>         |                                          |           |
| <b>4</b>     | <b>Results obtained</b>                  | <b>36</b> |
| <br>         |                                          |           |
|              | <b>Conclusion</b>                        | <b>38</b> |

---



## *Table of contents*

---

|                                     |                                   |      |
|-------------------------------------|-----------------------------------|------|
| <b><i>Appendix A</i></b>            | <b><i>Abbreviations</i></b>       | A 2  |
| <b><i>Appendix B</i></b>            | <b><i>Schematics</i></b>          | A 4  |
| Master                              |                                   | A 4  |
| Slave                               |                                   | A 6  |
| Client                              |                                   | A 8  |
| <b><i>Appendix C</i></b>            | <b><i>VHDL Programs</i></b>       | A 9  |
| Master program                      |                                   | A 9  |
| Slave program                       |                                   | A 14 |
| Client program                      |                                   | A 21 |
| <b><i>Appendix D</i></b>            | <b><i>FPGAs table address</i></b> | A 26 |
| Master's FPGA table address         |                                   | A 26 |
| Slave's FPGA table address (4 RJ45) |                                   | A 27 |
| Slave's FPGA table address (6 RJ45) |                                   | A 27 |
| Client's FPGA table address         |                                   | A 28 |
| <b><i>Appendix E</i></b>            | <b><i>References</i></b>          | A 28 |



## *1. Introduction*

Clock synchronization is a well-known problem in distributed systems. All algorithms are more or less sensitive to the drift of the network's internal clock. There are different types of synchronizations: it can be absolute, relative, global (through the entire network) or local, depending on its needs. Certain solutions just use information provided by a close environment. In our project, we will use a GPS solution.

Work stations, and even office automation are more and more networked. The synchronization problem on devices like these is important. Indeed, when you create a file on a station, it is physically created on a server at the server's date and not the station's one. Thus, if both are not synchronized, we can have files whose date of modification is more recent or older than the server's.

To solve this problem, we have to synchronize every single device on a universal time basic: UTC (coordinated universal time), and receive data via a GPS.

Many applications require synchronization between the elements of a network. For instance account to account flow of money between banks must be synchronized to insure security. There is also synchronized data exchange between several manufactures of the same brand located on different places which could be useful to save money.

In the following part you will find all the explanation on the devise we used to build our solution and a presentation of the language used to program.



## 1.1 Global Positioning System (GPS)

To measure time, it is necessary to have a scale of time, an origin on this scale, and an interval of time which will be used as unit. Any series of reproducible events can be used as scale of time. Astronomy provides us several of them: rotation of the earth on itself or around the sun.

Once the scale defined, we need to check specific properties allowing us to scientific work:

- *Universality*, this scale of time must be used and understood by a lot of people.
- *Reliability*, it is necessary that this scale cannot be stopped, and to create holes in the measurement of time.
- *Precision*, the reading must be as precise as possible.
- *Uniformity*, the measurement value shouldn't depend on the time or the place it has been done.

If we take as oscillator the rotation of the earth on itself, we can see that this rotation is not uniform over a few days. That's the reason why we have to find another solution, more reliable.

Since 1955 with the appearance of the atomic clocks we noticed a drift of the seconds defined thanks to natural clocks. Today, we use artificial oscillators, more precise than quartz clocks crystal controlled and the atomic clocks. Those are used more as standard of frequencies or timekeeping solutions that direct time measures.

There are two interesting ways to achieve this synchronization. We are able to design the solution through the internet. On the basis that we cannot synchronize computers with a great accuracy, utilization of a more precise clock via the internet is required. The drift is too important when using high-speed networks.



On the other hand, a Global Positioning System (GPS) approach is more suitable in terms of price and accuracy.

Each satellite is equipped of an atomic clock with a precision of 1 picosecond ( $1 \cdot 10^{-12}$  s). Then we can transport the precision of the satellite towards the receiver by synchronization signal: the satellite and the receiver generate signals having the same form. The receiver compares its signal with that received from the satellite and appreciates the shift between the two coded signals. This operation makes it possible to be sure to measure the time in the same way to the ends and, being sure of the signal course duration's measurement and, finally, the measurement of the distance.

The atomic clocks on the satellites are set to "GPS time", which is the number of seconds since 00:00:00 UTC, January 6, 1980. In 2006, GPS time is 14 seconds ahead of UTC, because it does not follow leap seconds. Receivers thus apply a clock-correction offset (which is periodically transmitted along with the other data) in order to display UTC correctly, and optionally adjust for a local time zone.

In order to get the time needed later in the device we had to use The Acutime™ 2000 GPS smart antenna designed by Trimble Company.

This antenna is designed for long-term reliability and features Trimble's latest GPS technology. It generates a pulse-per-second (PPS) output synchronized to UTC within 50 nanoseconds, outputting a timing packet for each pulse.

It is the ideal solution for precise timing and synchronization of data networks and provides a cost-effective and independent timing source for any application.





## 1.2 Field of Programmable Gate Arrays

A field-programmable gate array or **FPGA** is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates (such as AND, OR, XOR, NOT) or more complex combinatorial functions such as decoders or simple math functions. In most FPGAs, these programmable logic components (or logic blocks) also include memory elements, which may be simple flip-flops or more complete blocks of memories.

A hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer (hence the term "field-programmable") so that the FPGA can perform whatever logical function is needed.

FPGAs are generally slower than their application-specific integrated circuit (ASIC). However, they have several advantages such as a shorter time to market, ability to re-program in the field to fix bugs, and lower non-recurring engineering costs.

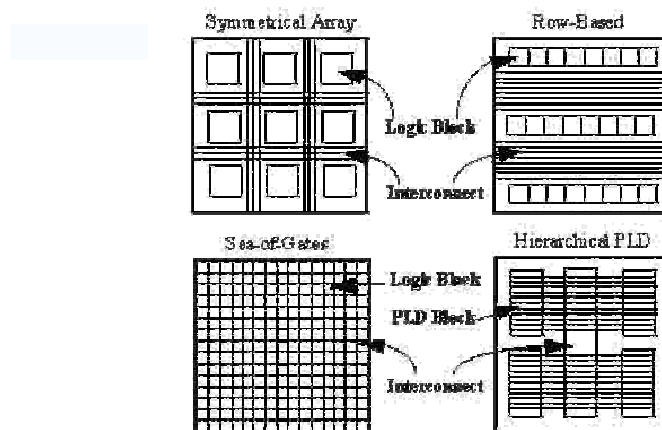
To define the behavior of the FPGA you need a hardware description language (HDL) or a schematic design. Common HDLs are VHDL and Verilog. In our case, we worked with VHDL.

Applications of FPGAs include DSP, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, and a growing range of other areas. FPGAs originally began as competitors to CPLDs and competed in a similar space, that of glue logic for PCBs.

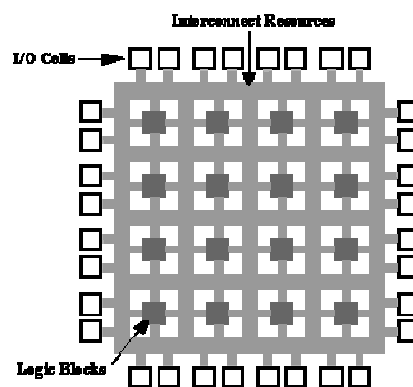
As their size, capabilities and speed increased they began to take over larger and larger functions to the state where they are

now marketed as competitors for full systems on chips. They now find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture.

There are 4 principal categories of architecture (see figure 1). FPGAs internal structure is represented on figure 2.



*Fig.1: FPGAs different architectures*



*Fig. 2: FPGAs internal structure*

## 1.3 Other Components

### 1.3.1) RS-422

RS-422, is a serial data communication protocol which specifies 4 wire, full-duplex, differential line, multi-drop communications. It provides for balanced data transmission with unidirectional/non-reversible, terminated or non-terminated transmission lines. RS-422 does not allow multiple drivers but only multiple receivers. You can see on the figure 3 below the RS-422 receiver and transmitter used on the solution

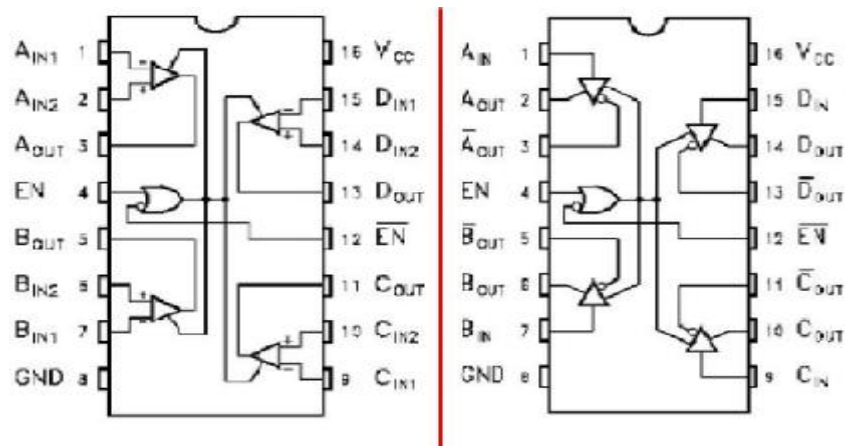


Fig. 3: RS-422 Receiver and RS-422 Transmitter

Several key advantages offered by this standard include the differential receiver defined in RS-423, a differential driver and data rates as high as 10 Mega baud.

The mechanical connections for this interface are mostly specified by DB-25 connector, however devices exist which have 4 screw-posts to implement transmit and receive pair only. The maximum cable length is 1200 m. Maximum data rates are 10 Mbaud/s at 12 m or 100 kbit/s at 1200 m. RS-422 cannot implement a truly multi-point communications network, although only one driver can be connected to up to ten receivers. A common use of EIA-422 is for RS-232 extenders.



### *1.3.2) RJ45*

RJ45 (Registered Jack 45) is a physical interface often used for terminating twisted pair type cables. It has eight "pins" per connector.

### *1.3.3) Edge connector*

An edge connector is the portion of a printed circuit boards consisting of traces leading to the edge of the board that are intended to plug into an edge connector socket as you can see on figure 4 below.

An edge connector socket, often popularly referenced simply as a "slot", is any type of female electrical connector for use with printed circuit boards having matching edge connectors. They consist of a plastic "box" open on one side, with pins on one or both side(s) of the longer edges, sprung to push into the middle of the open center.

The edge connector is a money-saving device because it only requires a single female connector, and they also tend to be fairly robust. For a time they were used in the vast majority of connectors found in computers, but modern computers have demanded many more pins than can easily be accommodated on the edge of a reasonable size board, and today more traditional male/female connectors are more common.



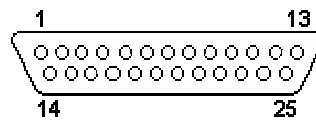
*Fig. 4: Edge connector*



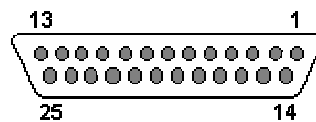
### 1.3.4) DB-25 Connector

The DB-25 connector (named for its "B"-size "D"-shaped shell and 25 pins) is practically ubiquitous in the electronics industry. The DB-25 connector is used for a variety of purposes. Two common applications are the parallel printer interface on the IBM PC and the RS-232 connections.

Figure 5 is a good set of figures for DB-25 male and female connectors, as viewed from the pin side.



DB-25 Male



DB-25 Female

*Fig 5: DB-25 Connectors*



## 1.4 VHDL

VHDL or VHSIC Hardware Description Language is commonly used as a design-entry language for field-programmable gate arrays (FPGA) and application-specific integrated circuits in electronic design automation of digital circuits.

### **Hardware description language**

In electronics, a hardware description language or HDL is any language from a class of computer languages for formal description of electronic circuits. It can describe the circuit's operation, its design, and tests to verify its operation by means of simulation.

An HDL is a standard text-based expression of the temporal behavior and circuit structure of an electronic system. In contrast to a software programming language, an HDL's syntax and semantics include explicit notations for expressing time and concurrency which is the primary attributes of hardware. Languages whose only characteristic is to express circuit connectivity between hierarchies of blocks are properly classified as net list languages.

HDLs are used to write executable specifications of some piece of hardware. A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time, provides the hardware designer with the ability to model a piece of hardware before it is created physically. It is this executability that gives the illusion of HDLs being a programming language. Simulators capable of supporting discrete event (digital), and continuous time (analog) modeling exist and HDL's targeted for each are available.



## 1.5 UART

A UART or Universal Asynchronous Receiver-Transmitter is a piece of computer hardware that translates between parallel bits of data and serial bits. A UART is usually an integrated circuit used for serial communications over a computer or peripheral device serial port. UARTs are now built into some microcontrollers

### **Basics**

Bits have to be moved from one place to another using wires or some other medium. Over many miles, the expense of the wires becomes large. To reduce the expense of long communication links carrying several bits in parallel, data bits are sent sequentially, one after another, using a UART to convert the transmitted bits between sequential and parallel form at each end of the link. Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms.

By convention, teletype-style UARTs send a "start" bit, five to eight data bits, least-significant-bit first, an optional "parity" bit, and then a "stop" bit. The start bit is the opposite polarity of the data-line's normal state. The stop-bit is the data-line's normal state, and provides a space before the next character can start.

A stretched "stop" bit also helps resynchronization. The parity bit can either make the number of bits odd, or even, or it can be omitted. Odd parity is more reliable because it assures that there will always be a data transition, and this permits many UARTs to resynchronize.

Speeds for UARTs are in bits per second (bit/s or bps), although often incorrectly called the baud rate. Standard mechanical teletype rates are 45.5, 110, and 150 bit/s. Computers have used from 110 to 230,400 bit/s. Standard speeds are 110, 300, 1200, 2400, 4800, 9600, 19,200, 28,800, 38,400, 57,600, and 115,200 bit/s. Concerning our project we will use 9600 bps speed.

The UART usually does not directly generate or receive the voltage levels that are put onto the wires interconnecting different equipment. An interface standard is used, which defines voltage levels and other characteristics of the interconnection. Examples of interface standards are EIA, RS 232, RS 422 and RS 485. Depending on the limits of the communication channel to which



the UART is ultimately connected, communication may be "full duplex" (both send and receive at the same time) or "half duplex" (devices take turns transmitting and receiving). Beside traditional wires, the UART is used for communication over other serial channels such as an optical fiber, infrared, wireless Bluetooth in its Serial Port Profile (SPP) and the DC-LIN for power line communication

Today, UART is commonly used with RS232 for embedded systems communications. It is useful to communicate between microcontrollers and also with PCs. Many chips provide UART functionality in silicon, and low cost chips exist to convert uart to RS232 signals





## 2. Design

The most important point in our solution was the clock's accuracy and reliability. The solution had to synchronize the different units towards a single input source: GPS receptions. All devices had to use the same time format, that's the reason why we chose to use UTC. Indeed, this format is easy to manage and compatible with different hardware configurations.

In order to make easier future upgrades, our solution had to be scalable. That's the reason why we managed to design boards as small as possible. Each unit, also called tier, also had to run without any other source than the *Master's*. An adapted solution was also required to face to different setups.

Each unit should run on a single power source except the master which uses two sources. Each unit should be able to detect erroneous timing signals in order to prevent spreading of incorrect information.

The GPS acquisition was done by the *Trimble Acutime 2000 kit* [4], based on a multi-channel receiver which needs to be outside at the beginning to see if the GPS was working and to configure the Port A. The accuracy of the system depends on the satellites' accessibility: we can't of course receive signals from every satellite. In the best case the system accuracy should be around 50 ns.

For each tier, the utilization of an FPGA is required as you can see on figure 6 below. We used three *Spartan 3 FPGA kits* [5], [6] operating at a frequency of 50 MHz based on *Epson SG-8002JF crystal oscillator*. Unfortunately and because of climatic conditions (temperature, humidity, etc...), these devices do not operate at exactly the same frequency. So we always had to check and measure their exact frequency. By using the GPS reference of 1 pulse per second, we just had to increment a counter at each FPGA clock's pulsation. After 1 PPS we noted the number of pulsation which gives us the exact frequency ( $f = 1/T$ ). By default we should have  $50 \times 10^6$  oscillations.

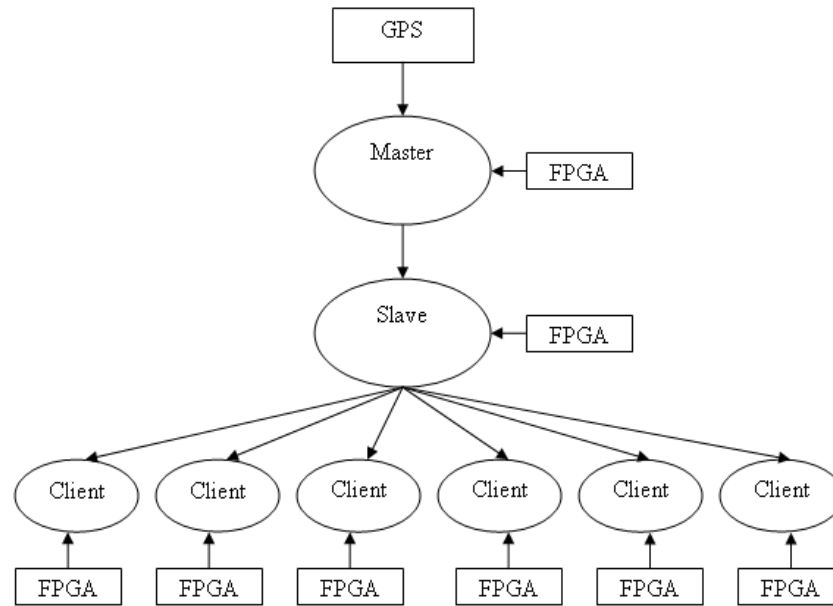


Fig.6: System's overview

## 2.1 Functional description

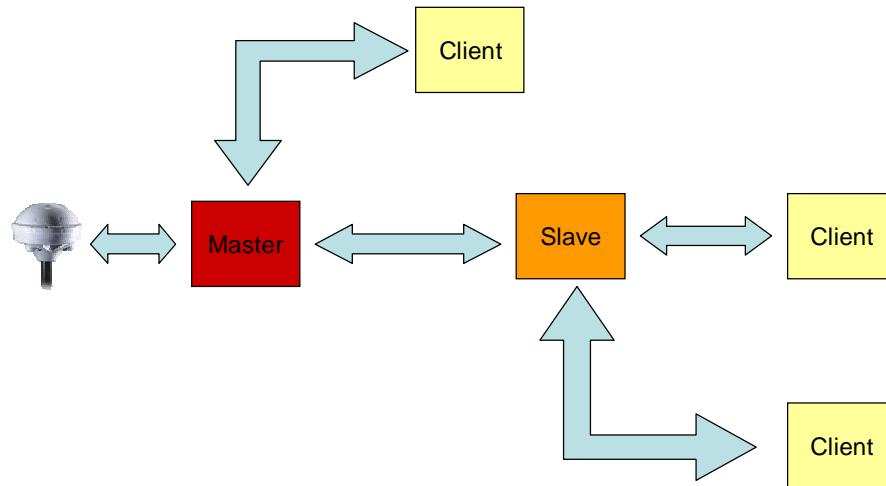
The solution suggested by Henrik Forsberg in his thesis consists of a hierarchical approach centralized by the first tier where the time source is. The tiers are named:

- 1<sup>st</sup> tier: *Master*. Receiving the PPS, which are the highest accurate reference signal used and the UTC from GPS.
- 2<sup>nd</sup> tier: *Slaves*, dispatching the signals from the master to different clients.
- 3<sup>rd</sup> tier: *Clients* (or measurement points). In our final design, we can have up to 6 *Clients* per *Slave*.

We receive signals from the GPS station on the first tier (Master) and then transmit it to the Slaves which can be servers of different networks, in the end leading us to the third and last tier, what we call Clients: workstations for instance. We can process without slave if the distance is shorter than 1200m or if only four



clients are requested. In our application the client receive directly the data from the master as shown on figure 6.1.



*Fig.6.1: Functional description*

As you can see on this figure, the Master communicates with the GPS Antenna. The Master sends data to configure the antenna (with Trimble Acutime in our case) and to enable the communication and receives all the data from it. With those data the Master obtains a Time and PPS. It will transmit that information to all the boards linked to it. For the Time we use two UARTs. The first to decode the data send by the antenna, then we choose the appropriate data we need, and finally the second UART allow to send those data to Clients or Slaves

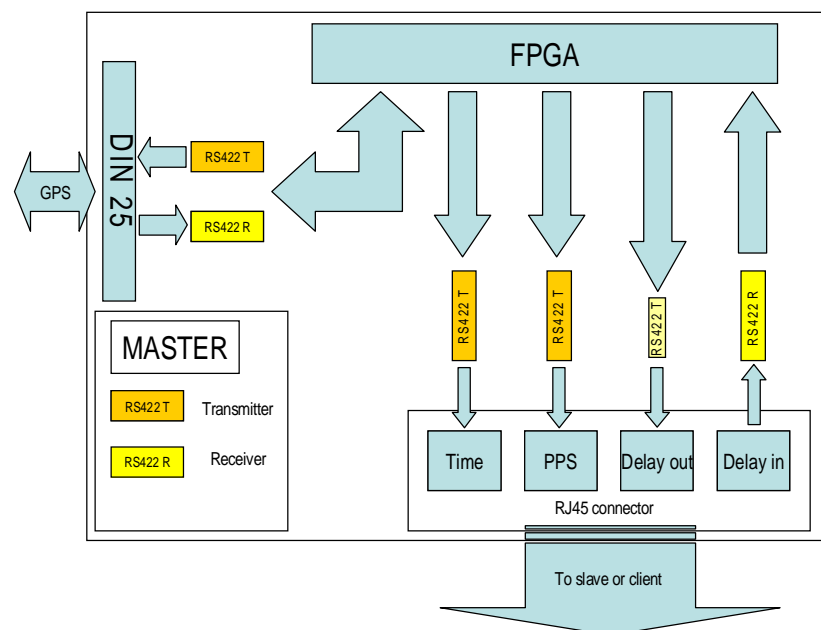
The Master receives Delay request from Clients or Slaves. When it receives this request (a bit high), it can resend directly this bit or wait few seconds before to resend it if there is a delay due to the wire between the Antenna and it (this delay is a constant that we can define in our code).

When the Client receives the answer from the Master, it can calculate its delay (due to the wire) dividing the time between sending and receiving by two. It also receives Time and PPS from the Master. With an appropriate UART we decode the Time sends by the Master and adding the Delay we obtain the correct time. We can display this time on the FPGA or use it by a different way.

The Slave allows to connect more Clients and to increase the distance between Clients and Master. It receives Time and PPS that it transmits directly to the Clients linked to it. It also calculates the Delay between Master and it. When it receives the delay request from one its Clients, the Slave wait during the delay due to the wire between Master and it before resending the bit.

## 2.2 Hardware design

We designed three boards; the schematics are presented in appendix (from page A4 to A8) In Table 1 we list the equipment needed to build the boards. In Figure 7, 8 and 9 we show an overview of the boards.



*Fig. 7: Master board*

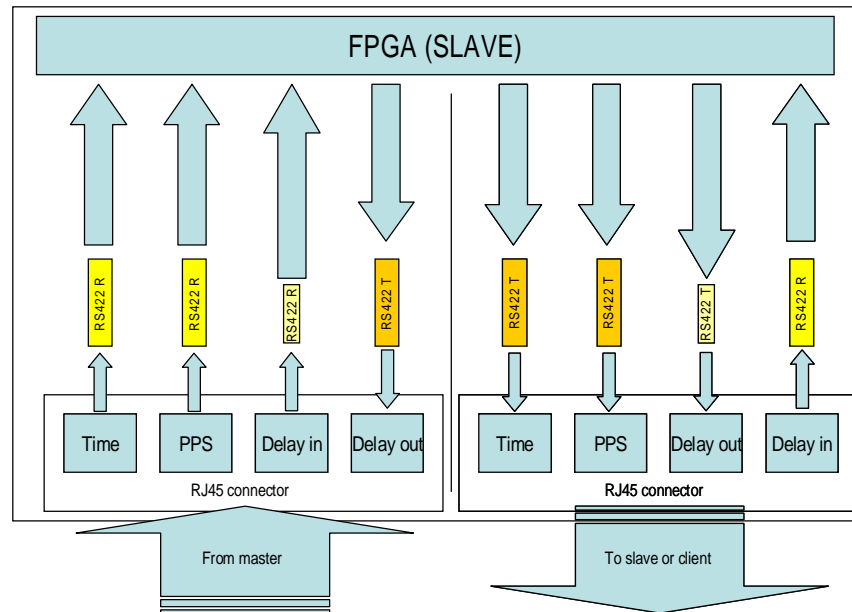


Fig. 8: Slave board

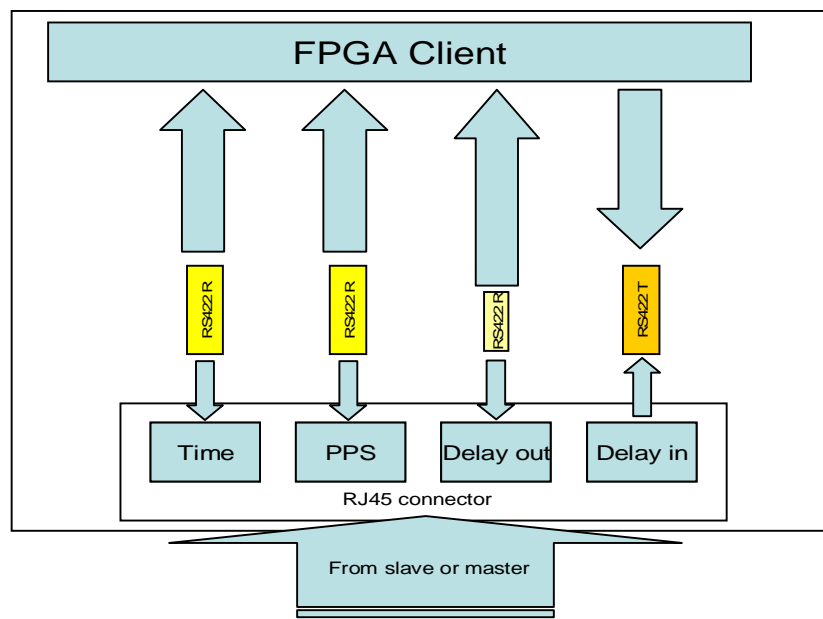


Fig. 9: Client board

In order to reach the objectives we needed different software and tools. Find a list of below:

- Proteus Software
- PCB Express (freeware: <http://www.expresspcb.com> )



- Oscilloscope
- Cutting pliers
- Soldering station
- Soldering pump
- Multimeter
- Electric cables of different colors

Moreover we needed the components we described before. Thanks to documentations we founded [2],[3]we made a list of all the items -the number needed are specified in the table below. Plastic boards are also requested to solder components on it. You need a 100mm\*160mm for Master and Slave and a smaller 80 mm\*60mm for the Client one.

|                          | Elfa Reference  | Master | Slave | Client | Total |
|--------------------------|-----------------|--------|-------|--------|-------|
| DIN 25                   | FCC17B25PE-A50  | 2      |       |        | 2     |
| RJ45                     | FRJA-408        | 4      | 1     | 1      | 6     |
| RJ45 dual                | FRJA-408-02     |        | 3     |        | 3     |
| Socket 16 pins           | 0-0390261-4     | 5      | 5     | 1      | 11    |
| Socket 8 pins            | 0-0390261-2     | 4      | 10    | 1      | 15    |
| Connector 40 pins        | C3510-40SPGB00R | 1      | 1     |        | 2     |
| Power supply GPS         |                 | 1      |       |        | 1     |
| LED                      | 204-10SURD      | 2      | 1     | 2      | 5     |
| Resistor 100ohms         | RD18JN100TJ2    | 1      | 1     | 1      | 3     |
| RS422 quad transmitter   | DS26LS31CN      | 3      | 3     |        | 6     |
| RS422 quad receiver      | DS26LS32ACN     | 2      | 2     | 1      | 5     |
| RS422 single transmitter | SN75176A        | 4      | 7     | 1      | 12    |
| RS422 single receiver    | SN7517A         |        | 3     |        | 3     |

**Table 1: Equipment used in board construction**



## 2.3 Software design

Objectives of the software part:

- All the clients have to be synchronized on time.
- The slave have to calculate is own delay with the master.
- The client have to calculate is own delay with the slave.
- The GPS send the PPS. Master and slave have to transmit it. Master, slave and client must post it.
- Calculate continuously the number of seconds since the creation of the UTC Time (6<sup>th</sup> January 1980) (32 bits)
- Calculate fraction of second (64 bits).

In order to build the VHDL programs for each tier, we needed to understand exactly how master, slave and client worked. Thus we have built module and schematics for each tier. Module helped them to understand the global operation, ways in and ways out for the PPS, time and delay when the schematics where useful to build the program itself.

### 2.3.1 MASTER

#### *Master's objective*

The *Master* receives and treats the data of time and frequency. It is the starting point of our solution and the only source of time that must synchronize all the other devices thereafter. It receives the PPS, which is the highest accurate reference signal used and the UTC from GPS.

#### *Master's module*

You can see this module on figure 10. Each circle represents a module of our program. Arrows are the inputs and outputs required by every module to work. Concerning the *Master*, we have got 5 different modules: Clock, PPS handler, GPS configuration & control, Delay and Time handler.

The module Clock is a 32 bits counter which counts each FPGA Pulse from the CLK and gives the counter value. This one represents the oscillation of the FPGA Clock.

The PPS Handler needs the PPS GPS (the Pulse Per Second coming from the GPS Antenna) and the Clock counter (described



above) to calculate the Frequency of the FPGA Clock. It also transmits the PPS with PPS OUT.

The DELAY module allows the calculation of the delay time due to the wire between the master and the board linked to (Slave or Client). So it needs a signal DELAY BIT IN (the number indicate on which RJ45 the signal come from) coming from the Slave or Client.

When the Master receives it, it sends another signal DELAY BIT OUT to the Slave or Client (calculating the time between the bit sent and the one received, it could know its delay). We also have a constant called OWN DELAY which can indicate the theoretical delay between the Master board and the GPS Antenna.

Then we have the TIME HANDLER which transforms the GPS TIME (coming from the GPS Antenna) in a TIME OUT transmitted to the linked board.

Finally we have the GPS Configuration & Control which permit the communication GPS – Computer thanks to the FPGA.



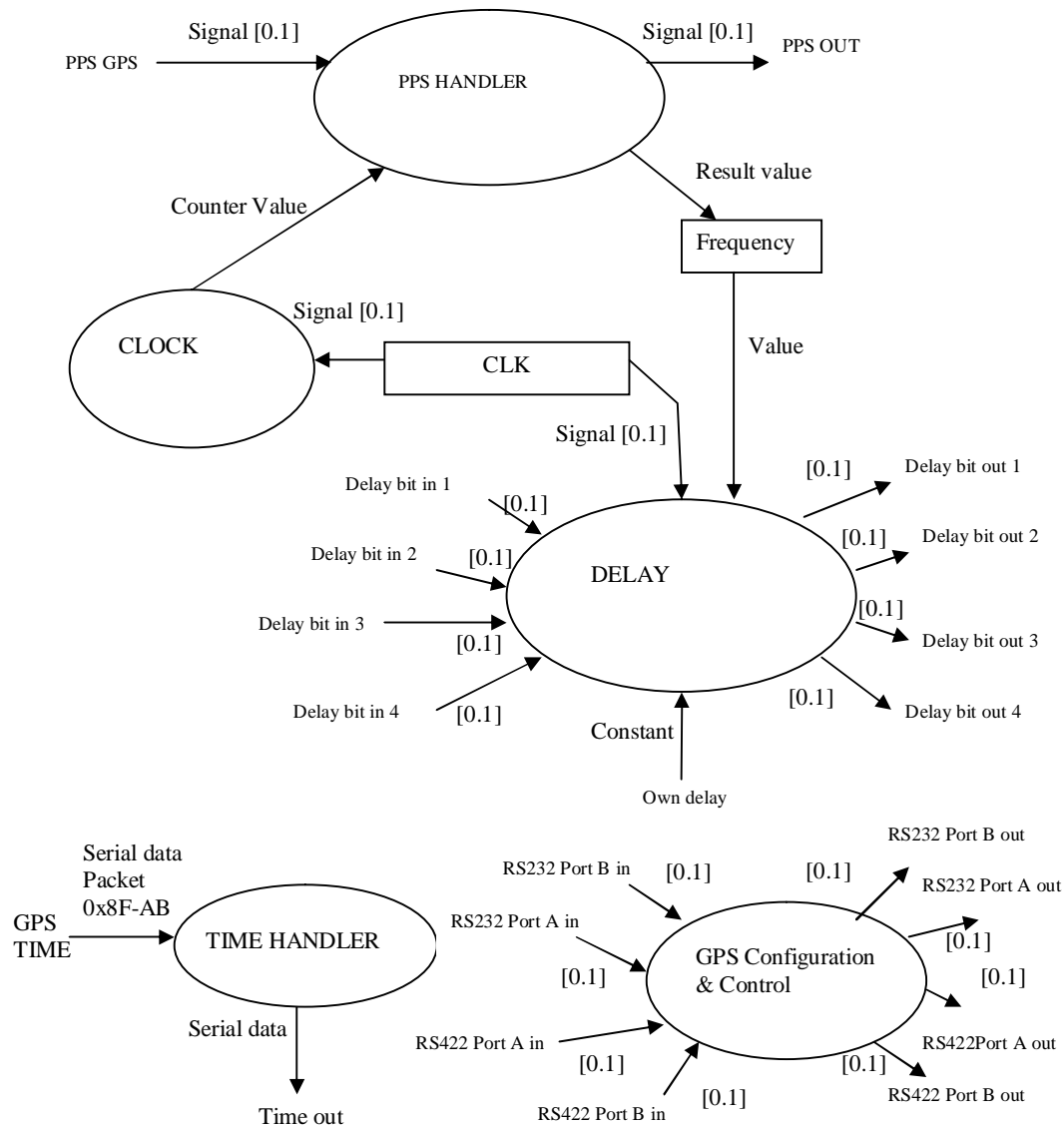


Fig 10: Master's module

### Master's schematics

To understand each module, we need to take a look inside and describe any single operation, what we called the schematics.

On the Master's Clock, see figure 11, you can see how works the 32 bits counter. We have the Frequency (which is the oscillation of the FPGA Clock) and we count each pulse. Then we transmit this 32 bits value to the others modules.



## CLOCK

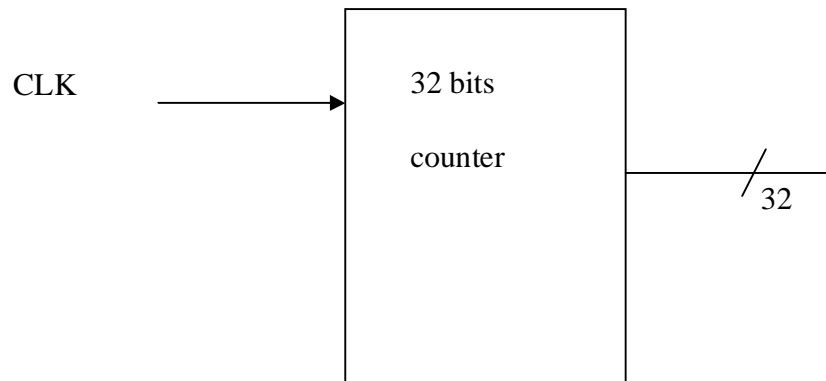


Fig.11: Master's clock

For the PPS Handler, on figure 12, we need the Clock (32 bits value) and the PPS. With the software (piece of code you can see into the Master Program in the Appendix) we put the value of Clock into a LATCH for the first PPS goes high. For the second PPS we put the value into another latch. Then we do a subtraction with the two Latch to know the number of pulse during one second (it is the out called Frequency). We also relayed the PPS with PPS OUT.

## PPS Handler

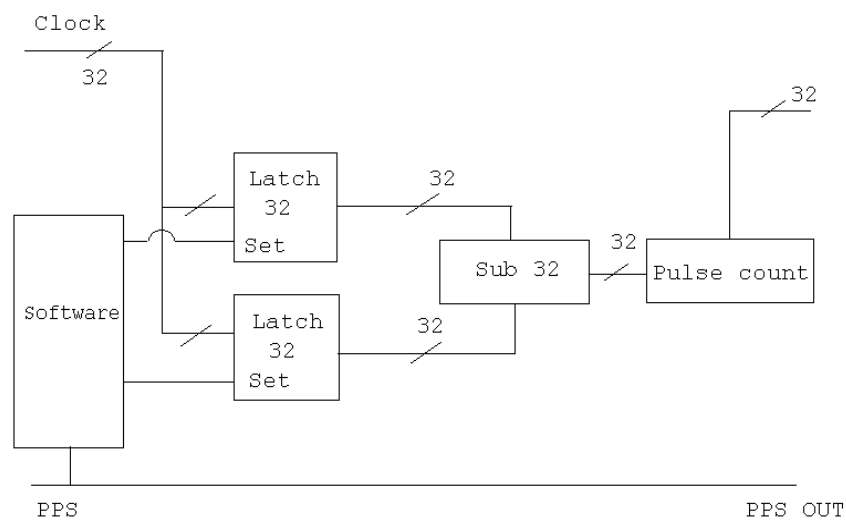


Fig.12: Master's PPS handler



The Time handler permits to transform the Time received by the GPS Antenna into a time easier to transmit. For that we use two UART as shown on figure 13. The Time coming from the GPS is a serial signal, so we need a UART Receiver to have a parallel message. Then we can transform the data and just keep the interesting part. When we got it, we transform the parallel message into a serial signal with the UART Transmitter and we have the TIME OUT.

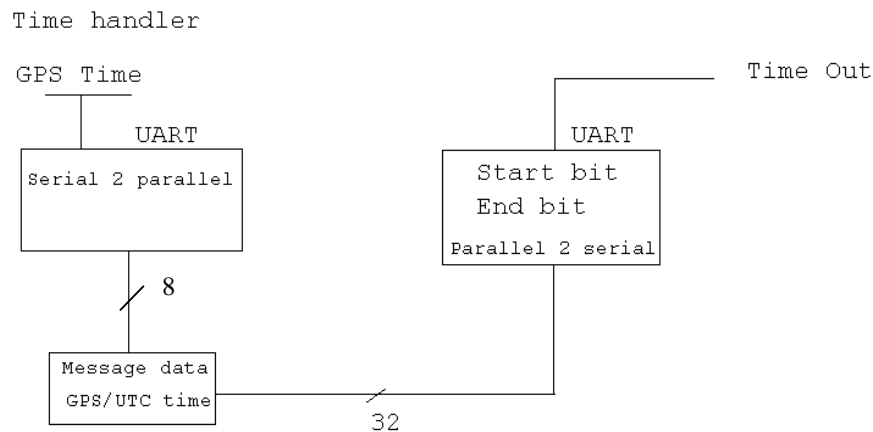


Fig 13: Master's Time handler

The Delay module is very easy to understand. When we receive a Bit coming from the board linked to the Master (called Delay PPS in) we wait during OWN DELAY (its value is 0 in theory) and resend it, see figure 14.

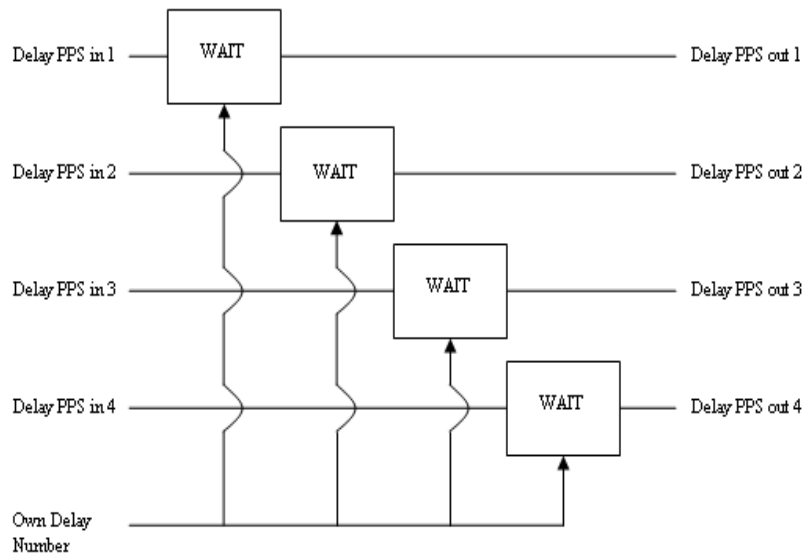


Fig 14: Master's Delay



### **2.3.2 SLAVE**

#### *Slave's objective*

The *Slave* is connected to the *Master*. We could have many slaves, but chose to focus on only one. Its accuracy must not depend on the distance from the *Master*. Finally it has to keep the time and frequency synchronization as precise as possible, and transmit it to the *Clients*. It acts as a relay between the *Master* and *Clients* and is very useful when working on long distances (more than 2 km) where RS422 would lose the signal.

#### *Slave's module*

You can see this module on figure 15. It is the same kind of figure than the Master's module. We have just 4 modules in this case. CLOCK is exactly the same than the master.

PPS HANDLER does the same operation but doesn't return the Frequency (in our Slave Program in the Appendix we calculate the Frequency so we can use it later, or just display it on a screen)

TIME HANDLER just relay the TIME from the Master to the Client.

Finally the DELAY is the only part really different from the Master. Delay bit in is the signal received from the client and Delay bit out is the signal sent to it (the number indicate which RJ45 we are using). Delay bit out M is the signal that the Slave sends to the Master and of course Delay bit in M is the signal received from it.

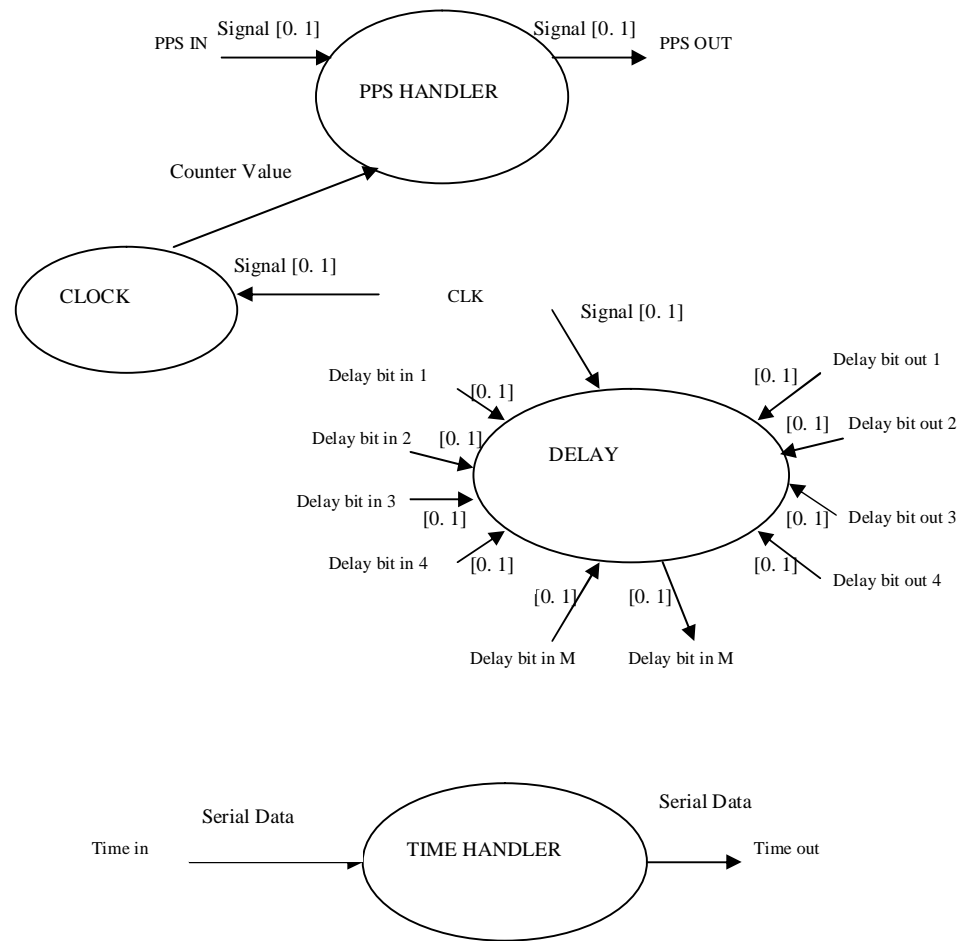


Fig. 15: Slave's module

### Slave's schematics

Slave's Clock and the Slave's PPS Handler is exactly the same than the Master's Clock and the Master's PPS Handler.

Slave's Time Handler just relay the time received from the master and sends it to the client, see figure 16.

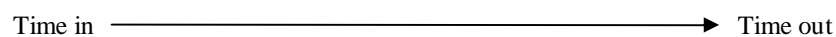


Fig. 16: Slave's Time Handler

The Delay works like on the Master but the Wait statement is not define by a constant (like OWN DELAY), see figure 17. Actually the Slave calculates his "OWN DELAY". For that every X second (we can define the number in second into the Slave program) the



Slave send a bit (Delay Bit Out M) with the BIT GENERATOR. When it send the bit we start the COUNTER 32 bits and we put its value into a Latch (Latch 32 bits) when we receive the bit from the Master (Delay Bit in M). Then we DIVIDE BY 2 to have to delay in the wire in one way. Here we have the delay needed to implement the WAIT. Like in the Master when we receive a bit from the Client (Delay BIT in) we wait during this delay before sending it back (Delay BIT out).

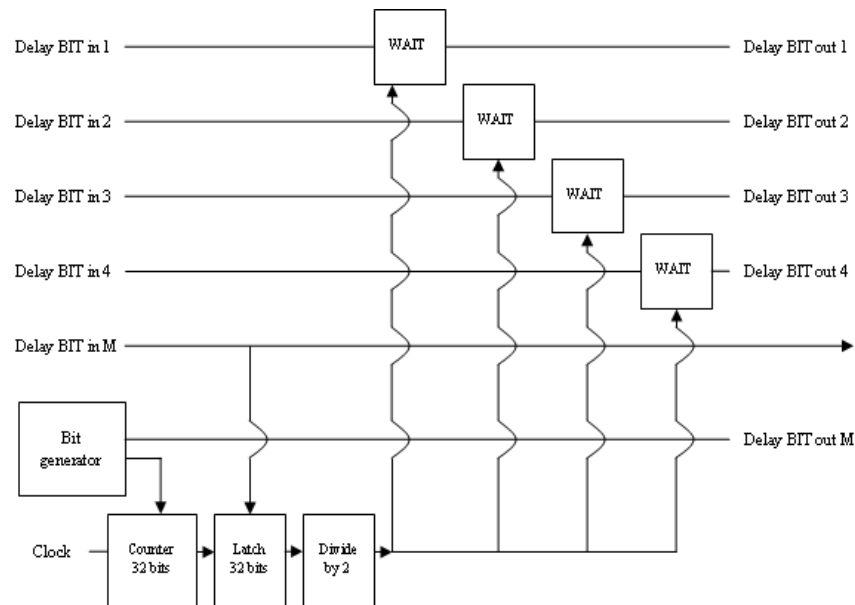


Fig: 17. Slave's Delay

### 2.3.3 CLIENT

#### *Client's objective*

*Clients* are our measurement points. They are connected to the *Slave* or the *Master* and located at the last tier of our hierarchy. Their purpose is obviously to give us information on the overall system performance by measuring time and frequency. We tested our final design with only 2 clients. Of course we could have a better idea of the solution's performance by multiplying *Clients*, but we would also need more FPGAs, which are quite expensive. However, we can plug up to 6 *Clients* on the *Slave*, increasing by this way the solution's flexibility.



### Client's module

You can see this module on figure 18.

CLOCK and PPS HANDLER are exactly the same than those in the Master.

The DELAY module is quiet similar than the Slave one. We send a bit (Delay bit out) and when we receive the bit from the Slave or the Master (delay bit in) with the CLK and the Frequency (calculate into the PPS Handler) we can calculate the DELAY ESTIMATION.

In the TIME HANDLER we receive the time coming from the Master and we add the DELAY ESTIMATION to obtain the SYSTEM TIME.

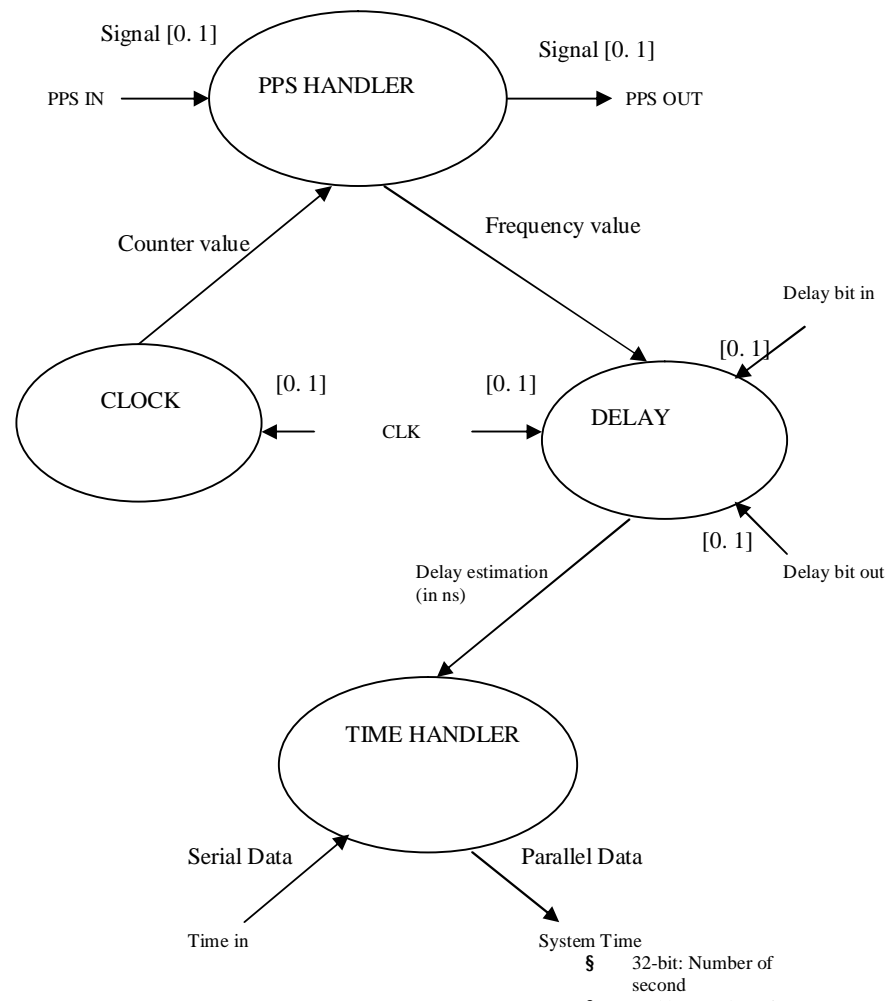


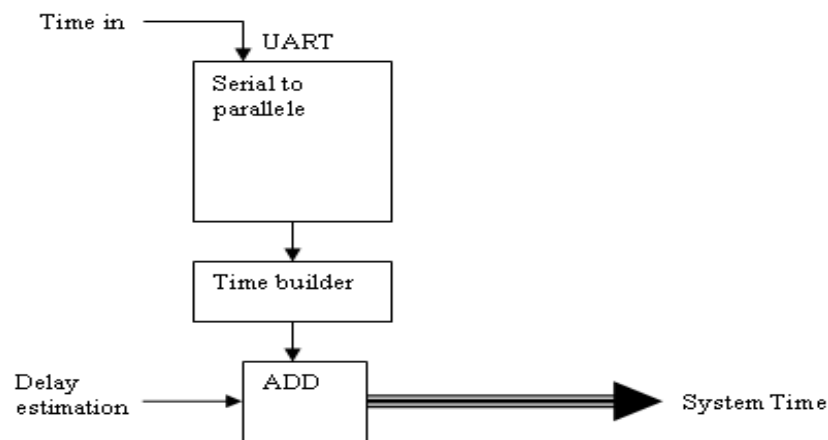
Fig. 18: Client's module



### *Client's schematics*

Client's Clock and the Client's PPS Handler is exactly the same than the Master's Clock and the Master's PPS Handler.

We receive the Time (Time in) into the TIME HANDLER, as you can see on figure 19. Then we use a UART receiver to transform the serial into a parallel message and we use a Time Builder to get the format we want. Finally we add Delay estimation to have the correct time.



*Fig. 19: Client's time handler*

The DELAY, figure 20 below, is almost the same than the Slave one. We generate a bit and start (or reset) a Counter. When we receive the bit from the Master or the Slave we put the counter value into a Latch. We divide by two to have the delay in one way. At this moment we have the delay in number of pulse. We divide by the frequency to obtain a Time that we send to the Time Handler (Delay Estimation).



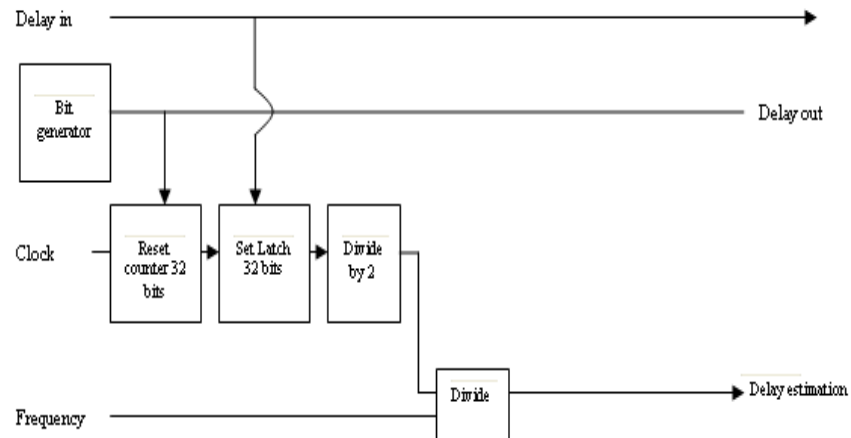


Fig. 20: Client's Delay

## 3. Implementation

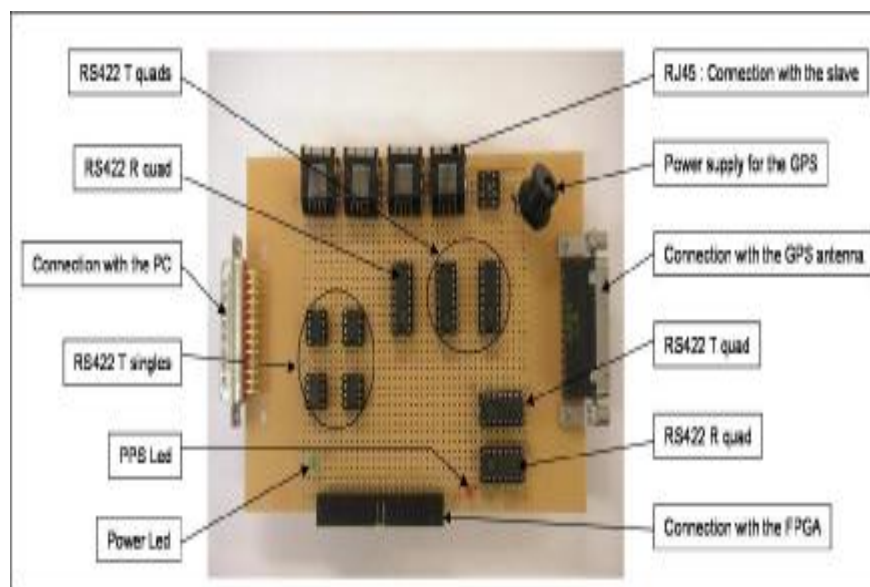
### 3.1 Hardware implementation

The first job was to understand the way the RS 422 worked both in transmitter and receiver. Thus we made test on IDL-600 Analog Lab device with RS 422 on test board linked with oscilloscope to see the evolution of the signal on the different pins.

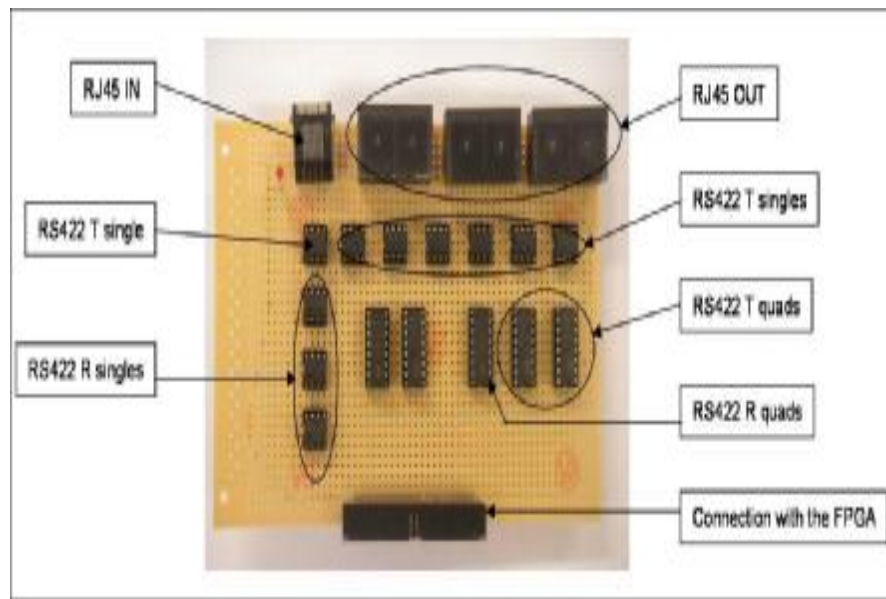
Once we had understood and validated the exact operation of that device we had to determine the number needed for each board before building the schematics.

Then we have started drawing the schematics through Proteus software first. After a lot of modifications due to some improvements, a first set of boards was drawn.

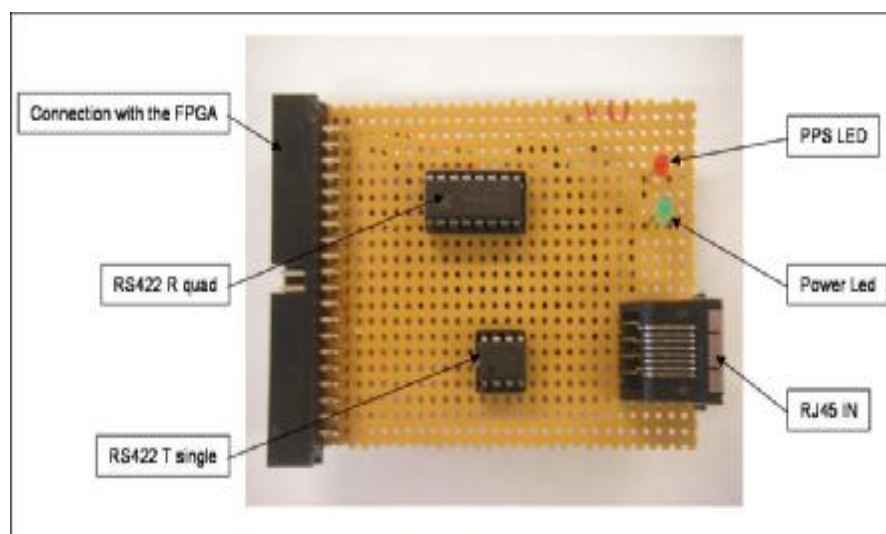
According to the schematics we have built the three first tiers and tested and validated it with the different FPGAs (Find the list of test in the software part – See chapter 3.4 page 31) Find the tiers in figure 21, 22 and 23 below.



*Fig. 218: Master's board*



*Fig. 22: Slave's board*



*Fig. 23: Client's board*

We continuously improve the boards thus we succeed in reducing the number of entrance data in the FPGA which helped them to facilitate the programming.

Finally we have drawn the final boards through PCB software. It allowed the building of printed circuit boards instead of cable boards. (Find the schematics in appendix B from page A 4 to A 8).



## 3.2 Software implementation

The main aim was to build a VHDL program for each master, slave and client. In order to reach this goal, we made several programs, helped with books [7], [8], to improve our programming skills first and to help to build the final one for each tiers.

Here is the list of all the programs we made before building the final one.

- Xilinx tutorials to know how to use ISE 7.1

  - In depth tutorial
  - Quick start tutorial

- Connector polarity test

- FPGA Communication

  - LED utilization

  - Switch utilization

  - Expansion connector utilization for FPGA's external communication

  - FPGA's clock utilization

  - Time reception test

  - PPS reception test

  - FPGA's serial port communication test (We used HyperTerminal software to observe the received signals)

  - Bit generation test

  - Frequency calculation test

  - UART test

Once all those programs built, we started to program the final code for each master, slave and client. Thanks to the module and schematics drew before and the programming skills developed with the different previous programs we succeeded in programming the codes for each tiers. Find the three codes in appendix C (Comments are added in the different programs to explain the different parts)



## ***4. Results obtained***

We built one Master board, two Slaves - one featuring 4 RJ45 and the other 6 RJ45 - and also two Clients and all of them are working properly.

We also drawn all the schematics corresponding to those boards in cable configuration as well as in printed circuit configuration in case of a future industrialization

We succeeded in calculate delays for each tier. To sum up we send a bit and start a counter simultaneously. When the upper level tier (Master for slave and slave for client) receive this bit, it sends it back to the transmitter tier. We stop the counter when we receive the bit back. The delay is the time needed to realize this operation.

We also succeeded in transmitting and posting the PPS for each tier.

We are able to send the Port A on the three different tiers but we didn't treat the data received because UART doesn't work. We can also receive the complete "Time" message send by the GPS.

### **Future work**

#### Hardware part

If we the decision to use mechanical UART is taken, the main work to do at this step of the project will be to look forward how to introduce UARTs on the boards in understanding properly how it works. This document should be really useful for that. (<http://www.elfa.se/pdf/73/737/07375322.pdf>)

#### Software part

Create a UART receiver to receive the Port A on the Master and treat the data of the 0x8F-AB (<http://www.diamondpoint.co.uk/manuals/gps/trimble/Acutime2000.pdf>, see page 181) to select only the useful data and send those ones to the client thanks to a UART transmitter.



The client, helped with a third UART, will treat and post the message received. It will receive 96 bits of data (32 bits for the number of second and 64 bits for the fraction of second)

For instance for the number of second, the master select the 8F-AB packet and keep only the useful data such Time of week (TOW), Week number, UTC Offset and UTC Time and send the selected data to the client which use the formulate below to calculate the number of second.

*Number of second = UTC Time + GPS Week number\*number of second in a week + Time of Week + UTC Offset*



## Conclusion

One interest of this project was to improve our skills in working in a team in an international environment. Indeed each member worked on a different part, thus forced us to work hand by hand and communicate every single day to progress in the same way.

We also improved our knowledge in project management after building a project plan at the beginning, we tried to follow it as close as it was possible and correct it when it was necessary.

We learned a lot in electrical engineering after carried out a project of five months in this specific department and more specifically on schematics building and VHDL programming.

Unfortunately we didn't succeed in finishing the whole project, even if we did a lot of work. We needed a bit more time to conclude the unit. We hope this will be done by our successors thanks to our work and this project will be useful for a possible industrialization to answer issues of synchronization for many applications.