Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

Break out Box for Transmission of Synchronous Video and CAN Data Streams over Gigabit Ethernet

Examensarbete utfört i Elektroniksystem
vid Tekniska högskolan i Linköping
av

Erik Irestål

LiTH-ISY-EX--09/4248--SE

Linköping 2009

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

Break out Box for Transmission of Synchronous Video and CAN Data Streams over Gigabit Ethernet

Examensarbete utfört i Elektroniksystem
vid Tekniska högskolan i Linköping
av

Erik Irestål

LiTH-ISY-EX--09/4248--SE

Handledare: **Lars Asplund**
ENEA Services Linköping AB

Examinator: **Kent Palmkvist**
ISY, Linköpings universitet

Linköping, 20 May, 2009

Avdelning, Institution
Division, Department

Division of Electronics Systems
Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Datum
Date

2009-05-20

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
__

ISRN
LiTH-ISY-EX--09/4248--SE

Serietitel och serienummer **ISSN**
Title of series, numbering __

URL för elektronisk version
http://www.es.isy.liu.se
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-ZZZZ

Titel
Title: Break out Box for Transmission of Synchronous Video and CAN Data Streams over Gigabit Ethernet

Författare
Author: Erik Ireståal

Sammanfattning
Abstract

Active safety systems for automobiles in the form of camera systems have evolved rapidly the last ten years, Autoliv Electronics in Linköping develops multiple such systems. In their development process there is a need for a Break out Box (BoB) to record and playback video and CAN data as if the camera system was used in an actual automobile. The aim of this thesis has been to develop a BoB for these camera systems. The work has been divided into three phases; identification of requirements, design of the BoB and implementation of a prototype.

The project has addressed four known issues with the currently used BoB; bandwidth, modularity, synchronization and usability. The result is a new BoB which is based on an FPGA connecting to a PC over Gigabit Ethernet. The design is an extendible platform for multiple channels of video, CAN data, other serial data and future extensions. A prototype proves the design concept by successfully recording video for the Autoliv NightVision system onto a PC.

Nyckelord
Keywords: NightVision, StereoVision, Break out Box, digital video recording, digital video playback, LVDS, CAN, FPGA

Abstract

Active safety systems for automobiles in the form of camera systems have evolved rapidly the last ten years, Autoliv Electronics in Linköping develops multiple such systems. In their development process there is a need for a Break out Box (BoB) to record and playback video and CAN data as if the camera system was used in an actual automobile. The aim of this thesis has been to develop a BoB for these camera systems. The work has been divided into three phases; identification of requirements, design of the BoB and implementation of a prototype.

The project has addressed four known issues with the currently used BoB; bandwidth, modularity, synchronization and usability. The result is a new BoB which is based on an FPGA connecting to a PC over Gigabit Ethernet. The design is an extendible platform for multiple channels of video, CAN data, other serial data and future extensions. A prototype proves the design concept by successfully recording video for the Autoliv NightVision system onto a PC.

Acknowledgments

Many people have contributed to this master thesis. Especially I would like to thank my supervisor Lars Asplund at ENEA Services Linköping AB. Thanks also to the people at Autoliv Electronics Linköping, where I performed most of my thesis work, and my examiner Kent Palmkvist.

Lastly I would like to thank my family, friends and my Lord and Saviour Jesus Christ.

Till Mamma och Pappa.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives the background to the problem which this master thesis aims to present a solution to. The methods used in the work are presented and discussed as well as the limitations.

1.1 Background

Active safety systems for automobiles in the form of camera systems have evolved rapidly the last ten years. Autoliv Electronics manufactures a NightVision system based on an infrared (IR) camera, but are currently also developing other safety systems based on cameras for visible light.

1.1.1 NightVision

Autoliv Electronics released the first commercial NightVision system in 2005 and since then it has been an option with the BMW 5-, 6- and 7-series automobiles. During the fall of 2008 the second generation of the NightVision system was released and now includes pedestrian detection as an added feature.

The NightVision system has three major parts; the IR camera, the electronic control unit (ECU) and the video display, see figure 1.1. The video from the IR camera, which is placed in the front spoiler of the car, is transmitted to the ECU over a low-voltage differential signaling (LVDS) link, a serial link. In the implementation of the NightVision system the LVDS link uses a single differential pair. For the ECU to communicate with the IR camera, for initialization and exposure control, there is a private Controller Area Network (CAN) link, also a serial link, between the two. Furthermore the ECU also receives data from the CAN bus of the automobile. The CAN bus of an automobile is used by the different systems of the automobile to exchange data. The information that the ECU receives on the CAN bus is for example the outside temperature, speed and the yaw rate of the car, that is the rate of which the car is currently rotating. The ECU has one LVDS video input port and two CAN ports. The video received in

the ECU is processed to enhance the image quality, pedestrians are detected and warnings are posted if necessary. The output port of the ECU is an analog video signal that is connected to a display in the dashboard of the automobile, see figure 1.2.

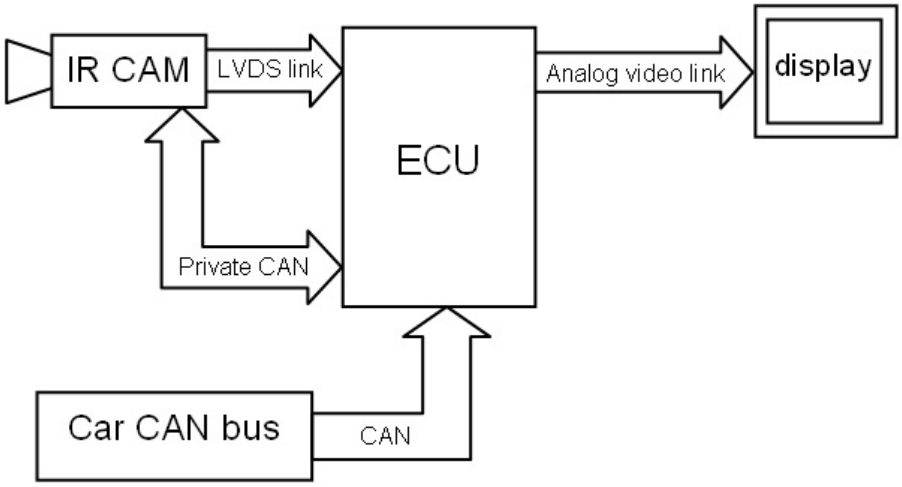


Figure 1.1. An overview of the main parts of the NightVision system; the IR camera, the ECU and the video display.

Figure 1.2. The interior of a car with a NightVision system, notice the display in the center of the dashboard displaying the humans in bright white.

1.1.2 Future camera systems

The future camera systems that are currently under development by Autoliv Electronics generally have the same parts as the NightVision system, see figure 1.1, but they differ from the NightVision system in that they are based on cameras for visible light. One of the projects is the StereoVision system, it is based on two cameras which are processed together by an ECU. Furthermore these systems have higher video frame resolutions than the NightVision system as well as a higher video frame rate. To handled this increased bandwidth from the camera a different LVDS link is used between the cameras and the ECU. The control link between the cameras and the ECU is different as well, instead of a CAN link the systems use either the serial Inter-Integrated Circuit (I^2C) link or the serial RS232 link.

1.1.3 Development and testing issues

In the development and testing process of the NightVision system and the other future systems there is a need for the possibility to repeatedly test the system as if it was used in an actual automobile. By pre-recording video and the corresponding data on the CAN bus in an automobile, it is possible replay the video and the CAN data in the development laboratory or the testing facility. For this to be possible there has to be a mechanism for recording and playback of video and automobile CAN data and currently there is a Break out Box (BoB) that makes this possible. This BoB is a hardware box with an LVDS and CAN interface on one side and a FireWire (IEEE1394) interface on the other side, see figure 1.3. With the BoB it is possible to record a sequence in an automobile to a Personal Computer (PC), see figure 1.4, and then play back the sequence to the ECU of the system as if the sequence happened in real time, see figure 1.5.

The currently used BoB was designed to work with the original NightVision system, therefore it can only handle the LVDS link used by the NightVision system. That implies that the future camera systems need either to be based around the same LVDS link, with its bandwidth limitation, or another BoB will be needed. Furthermore there have been issues with the synchronization of the video from the IR camera and CAN data from the automobile CAN bus. The ECU needs the data streams to be synchronized as they are in a real system setup, but when using the BoB to record and later to playback the data the synchronization is lost to such a degree that the determinism of the NightVision system can not be guaranteed. Lastly the usability of the BoB currently used has been an issue, the total lack of diagnostics and status of the BoB reduces the user experience significantly. Trial and error, and finally restarting the BoB ends up being the solution to erroneous behavior.

Figure 1.3. The current BoB, which also is know as the "Wicer box".

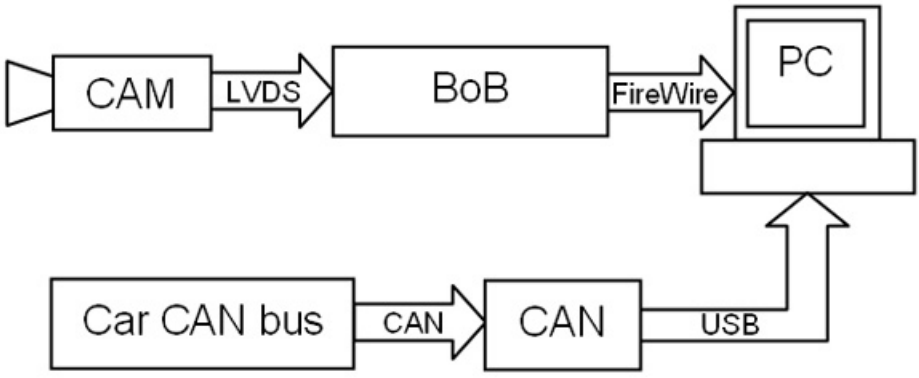


Figure 1.4. Block diagram of the NightVision recording scenario, which takes place in an automobile.

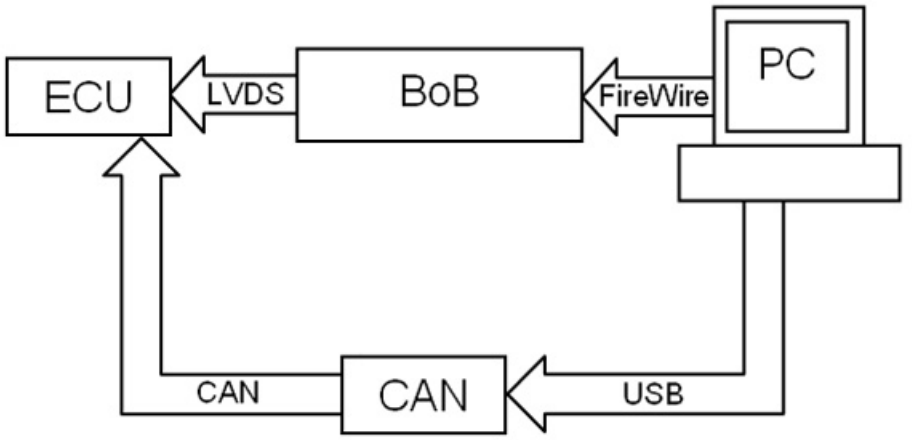


Figure 1.5. Block diagram of the NightVision playback scenario, which takes place in the development lab or the test bench post production.

1.2 Problem statement

The issues mentioned lead to the aim of this thesis, to develop a new BoB that addresses the bandwidth problem, the LVDS link problem, the synchronization problem and the problem with the lack of usability. In chapter 2, Requirement specification, the individual specifications for the bandwidth, the LVDS link, the synchronization and the usability will be specified.

1.3 Method

The development of the new BoB has essentially been divided into three distinct phases; *identification of requirements*, *design* of the BoB and *implementation of a prototype* of the BoB on prototype hardware. During the whole development process there has been a special emphasis on using modern and automated development methods. As ENEA Services Linköping AB are consultants with the emphasis on software development, inspiration in the development process of the BoB has been drawn from the software domain, to be applied in the hardware development.

1.3.1 Requirement specification methodology

In the requirement specification phase, two templates for documenting requirements, [1] and [2], where used. Every stakeholder of the BoB within Autoliv was consulted and a formal requirement specification was determined. This requirement specification has been attached in chapter B of the appendix.

1.3.2 Design methodology

In the design phase a top-down methodology was used together with a software architecture methodology which was adapted to fit in the hardware domain. The software architecture methodology is called "The '4+1' View Model of Software Architecture" [3]. In this model Kruchten uses four different system views; the logical view, the process view, the development view and the physical view, together with the use scenarios, to describe the architecture of a software system. In the hardware domain I have translated the four views into the functional view, the process view, the implementation view and the physical view. The use scenarios are the identified in the requirement specification.

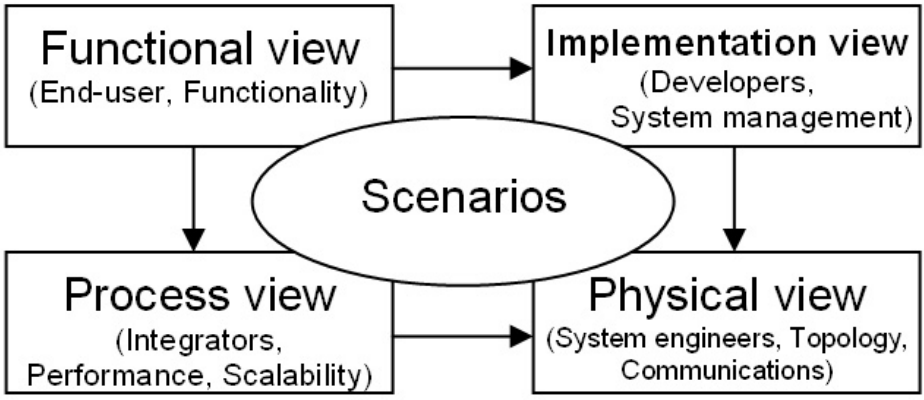


Figure 1.6. 4+1 view model adapted to the hardware domain.

1.3.3 Implementation methodology

The BoB is modeled in VHDL, a hardware description language. Test benches for model verification are written in VHDL. A test driven methodology, as described in [4], has been used to develop the VHDL model of the BoB. Using this methodology a test bench for the model to be built is always built first and it is checked until the model passes the test. This way testing is not neglected but performed for every module of the final VHDL model and it implies that once the model is finished it has already been tested.

Apart from using the test driven methodology a number of modern FPGA development methods has been used in the implementation phase of the BoB. ENEA Services Linköping AB has an interest in developing standards for FPGA development. Part of the thesis work has therefore included evaluating a number of methods and tools for FPGA development.

- The open source software TortoiseSVN has been used for *version control*.

- The open source software DoxyGen has been used for *automated documentation* of the VHDL code.
- To determine the *coverage* of a VHDL test bench ModelSim Code Coverage has been used.
- To ensure a *unified VHDL coding style* automated checks has been run with HDL Designer.
- The open source scripting language Make has been used for *automated verification, synthesis and implementation* of the VHDL model.

1.4 Limitations

The development of a new BoB for NightVision and future camera systems is a task more time consuming than the time frame of a master thesis. Therefore the master thesis has been focused on identifying and documenting all requirements of the new BoB, designing a platform for the BoB which can be extended and adapted when future needs may arise and finally to implement a prototype of the BoB. The prototype is to be regarded as a proof of concept of the chosen design. The development of the actual hardware for the BoB has not been part of the thesis, neither has the adaption of the BoB to fit the future camera systems been part of the thesis. Further more the master thesis has not included any work on the software application to be used in the PC to record and playback video from and to the BoB. The prototype has been built to work for recording of video for the existing second generation NightVision system.

1.5 Disposition

The organization of the report follows the main phases of the master thesis. Chapter 1 gives an introduction to the problem at hand. Chapter 2-4 correspond to the three main phases; identifying the requirements of the BoB, designing the BoB and finally implementing a prototype of the BoB. Chapter 5 concludes the report and discusses the results of the master thesis.

In the appendix there is a list of all the abbreviations, appendix A. The original requirement specification and the original design specification has been attached, appendix B and C. This way the interested reader gets a detailed understanding of the requirement specification and the design of the BoB.

Chapter 2

Requirement specification

In this chapter the requirement specification of the new BoB is established. For the original requirement specification, see appendix B.

2.1 Stakeholders

In order to establish all the requirements of the new BoB every stakeholder of the BoB was consulted. The BoB will mainly be used by three different project groups; the NightVision group, the StereoVision group and one additional camera system group working with cameras for visible light. Within these projects the BoB will be used for a multitude of different development and testing purposes.

- The BoB will be used for testing during *development of algorithms* and verification of the implementation of the same.
- The BoB will be used for *system test and final verification* of the current NightVision system, assembled at the Autoliv Electronics production facility in Motala, as well as the future camera systems.
- The BoB will be used in *product validation*, a process in which currently the NightVision system is subjected to tough environments; heat, electromagnetic radiation etc, and is tested over a prolonged time, up to thousands of hours.
- The BoB will be available to the *customers* of the NightVision system and the other future camera systems, for them to perform their own tests.
- Autoliv Electronics Vision, in Goleta, CA, USA, assembles the NightVision cameras and they may want to use the BoB for *camera development* and testing purposes.
- Lastly the BoB will also be used in the *FNIR project* (Fusing Far and Near InfraRed imaging for pedestrian injury mitigation) in which Autoliv Electronics AB is a partner [5].

2.2 Use cases

To ensure that every aspect of the requirements is covered, all possible use cases for the BoB are identified. These use cases correspond to the scenarios in the "4+1 view model" used in the design phase.

2.2.1 Recording data

When gathering test video driving around in an automobile, the BoB will enable video from the automotive camera and CAN data from the automobile CAN bus to be recorded onto a PC, see figure 2.1.

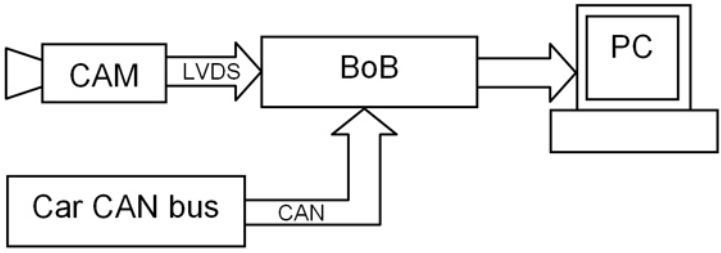


Figure 2.1. Use case recording with the BoB.

2.2.2 Data playback

When in the test lab, the BoB will enable playback of recorded video and CAN data from PC to an ECU, see figure 2.2.

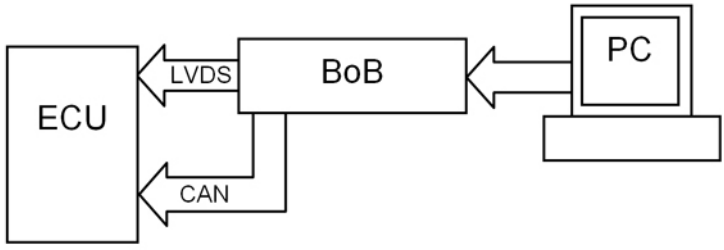


Figure 2.2. Use case playback of video and CAN data from the PC with the BoB.

2.2.3 Recording tapped data

When gathering test video driving around in an automobile, the BoB will enable tapping and recording of data, video and CAN data, onto a PC, while the camera system functions as normal, see figure 2.3.

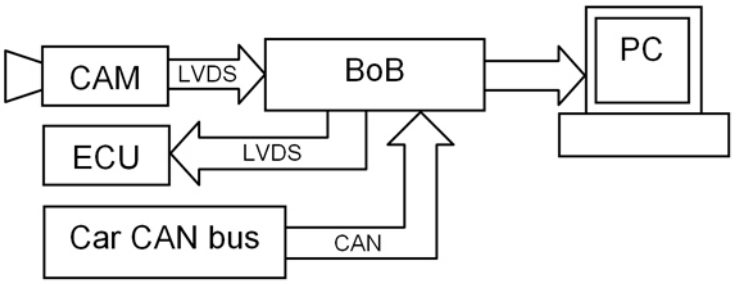


Figure 2.3. Use case tapping data for recording with the BoB.

2.2.4 Recording processed data

When gathering test data in an automobile, the BoB will enable recording of processed video from the ECU onto a PC, see figure 2.4.

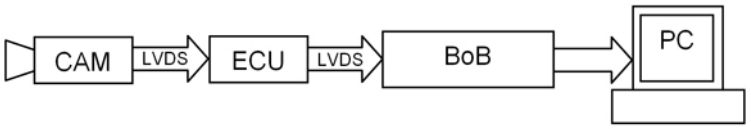


Figure 2.4. Use case recording processed data with the BoB.

2.2.5 Recording of both unprocessed and processed data

Using two BoBs the unprocessed and the processed data can be compared and recorded simultaneously to a PC, see figure 2.5.

2.2.6 Testing of system algorithms

For high testability of the camera systems, the BoB needs to support playback of CAN data together with video over the LVDS link transparently from a PC to the ECU. The setup is the same as for normal playback.

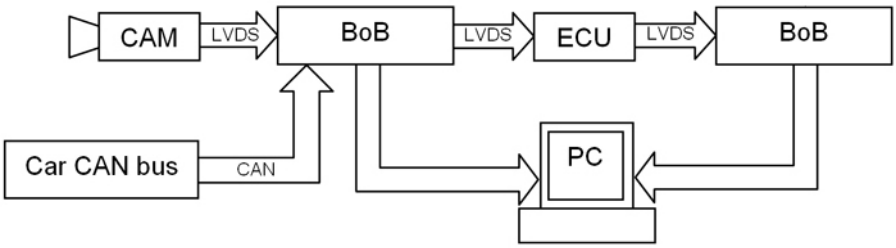


Figure 2.5. Use case recording both unprocessed and processed data with two BoBs.

2.3 Bandwidth

The bandwidth requirement of the BoB is determined by the resolution, the frame rate and the pixel depth of the video from the camera. For all the different camera systems the video is uncompressed to allow for image enhancement and object identification, e.g. pedestrian detection, therefore even modest resolutions require high bandwidth. Serial links for CAN, I^2C or RS232 data has a maximum bandwidth of one Mbit/s, but usually far lower, and therefore can be neglected in comparison to video. The specification for the NightVision system is listed in table 2.1.

NightVision video specification	
Horizontal video frame resolution	324 pixels
Vertical video frame resolution	256 pixels
Pixel depth	14 bits/pixel
Frame rate	30 frames/second
Effective bandwidth	*4.35 MB/s*

Table 2.1. NightVison system video specification.

It is the StereoVision system and the future system that require higher bandwidths. In a future generation of StereoVision the bandwidth requirement will be almost eight times the bandwidth of the current NightVision system, neither does the current LVDS link nor the FireWire 400 Mbit/s link used in the current BoB support this system.

The bandwidth requirements of the different systems are listed in table 2.2. As one can see the effective bandwidth of the NightVision system, table 2.1, differs from the actual bandwidth of the system, table 2.2. This is because the full bandwidth of the LVDS link between the IR camera and the ECU in the NightVision system always carries data, it is video data, which requires 4.35 MB/s, control data and empty data to fill up the bandwidth. To conclude, the BoB is required to support a bandwidth of up to 618 Mbit/s.

System	Bandwidth (Mbit/s)	Bandwidth (MB/s)
NightVision	80 Mbit/s	10 MB/s
StereoVision	256 Mbit/s	32 MB/s
Future system	270 Mbit/s	33.75 MB/s
Future StereoVision	618 Mbit/s	77.25 MB/s

Table 2.2. Bandwidth requirements of the different camera systems.

2.4 Modularity

The current BoB is a static system that only supports one LVDS link and no serial links. The new BoB is required to have greater modularity, different LVDS links are required to be supported, the future system currently being developed will use a different LVDS link with a greater bandwidth than the LVDS link used in the NightVision system. Not only different LVDS links are required to be supported but also different serial links, CAN, I^2C and RS232 in different combinations. The BoB is required to be reconfigurable to function with all camera systems Autoliv Electronics is developing. Requiring the BoB to have a CAN interface makes the external USB connected CAN host, which currently is used (see figure 1.4 and figure 1.5), redundant and saves hardware as the new BoB will replace two hardware boxes, the current BoB and the USB connected CAN host.

2.5 Synchronization

The biggest issue with the currently used BoB is the synchronization between video and automobile CAN data. Video is recorded via the BoB and CAN data is recorded with an USB connected CAN interface, with this solution synchronization between the data streams is lost. Test engineers at Autoliv Electronics estimate that the time shift between the data streams is on the order of a few video frames. The underlying mechanism for the lost synchronization is memory buffers in the PC and the fact that the operating system (OS) used on the PC is Windows XP, which is not a real time operating system. The incoming memory buffer for the video data on the FireWire link and incoming memory buffer for the CAN data on the USB link are different depending on different device drivers. Furthermore the buffers might not even be static over time but may depend on the current work load of the PC. These assumptions have not been proved but seem to be the most plausible, explaining the behavior using the current BoB.

The lost synchronization between video and CAN data makes the camera system behave nondeterministically in certain test scenarios. Autoliv Electronics have had its customers, who also do testing on the camera system, report supposed bugs of the camera system, which the test engineers at Autoliv have been unable to reproduce. The reason for this is assumed to be the fact that in the laboratory the recorded situation will not be reproduced by the, in this case, NightVision system

in the same way as it took place in the live situation in the automobile. To solve the issue of synchronization one needs to establish what the required synchronization between the video and the CAN data needs to be.

Examining the NightVision system one quickly realizes that its main loop is based on one video frame, see figure 2.6. During every frame, which is $1/30 \approx 33$ ms long, the loop polls for CAN data twice hence every ∼17 ms. The CAN data on the automobile CAN bus is transmitted with industry standard time intervals. The most frequent parameter used by the camera system is updated every 20 ms. That means that it at the most takes 33 ms between two different values of the most time critical CAN data parameter. However perfect synchronization is needed between the video and the CAN data because the main loop wraps around every 33 ms and the CAN data is updated every 20 ms. Since the rate of the two processes is different they will always be out of phase and therefore even the slightest time shift between the video and the CAN data will cause an altered behavior of the ECU in playback compared to recording of the data. However the finer the resolution of the synchronization is, the fewer video frames are affected and the nondeterministic behavior is decreased. The highest achievable synchronization resolution would be to use a counter updated with the main clock of the BoB to time stamp the video and the CAN data. However the nondeterministic behavior may also partially be contributed to the architecture of the NightVision system, which has multiple clock domains which are not synchronized at startup of the system.

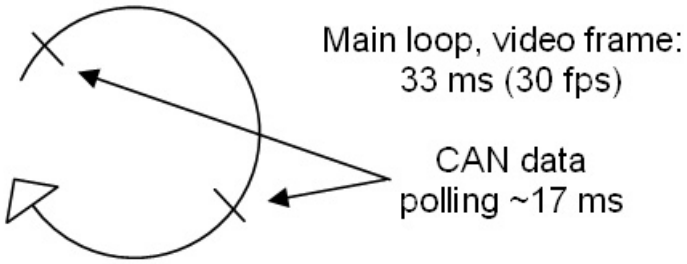


Figure 2.6. Main loop of the NightVision system with CAN polling indicated.

2.6 Usability

The usability of the BoB refers to the ease of use of the BoB and the possibility to automate its operation. The current BoB only has one LED indicating whether it has power or not, there is no other possibility to diagnose the status of the BoB. Furthermore the hardware of the current BoB, which is a small FPGA, needs to be reprogrammed to switch its operation from recording to playback or vice versa. At Autoliv Electronics this has meant that certain BoBs are only used for recording and others are only used for playback since Autoliv engineers can not reconfigure

the BoBs themselves. The new BoB is required to be setup in software and to support both record and playback. It is also required to have additional status information about which mode it is operating in and other information useful for diagnostics.

Externally the new BoB is required to have both a power LED and a LED indicating the status of the link between the BoB and the PC, the current FireWire link lacks such LED. The physical connector of the link between the BoB and the PC is also required to have some sort of mechanism hindering the connector to unplug accidentally, something that has been fairly common with the FireWire connector of the current BoB.

Lastly the usability also concerns the interface to the PC. The current BoB only supports using Windows XP as the operating system because of OS specific device drivers for the Firewire link of the current BoB. In addition to that a FireWire extension card is needed in every PC with which the BoB is used, since FireWire often is not a motherboard standard of most desktop PCs. The new BoB is therefore required to use an off-the-shelf communication link between it and the PC to support multiple OSes without special drivers and extension cards.

Chapter 3

System design

This chapter presents the design of the new BoB as well as the design development process, together with design decisions.

3.1 Design proposal

Autoliv Electronics, with their knowledge of the issues with the current BoB, proposed the new BoB to be an FPGA based design with a 1 Gbit/s Ethernet link between the BoB and the PC. This proposal was thoroughly examined, using the 4+1 design view model [3], with the result presented below.

3.2 4+1 design views

The four different architectural views, presented by the "4+1 View Model" [3], which I have modified into *the functional view*, *the process view*, *the implementation view* and *the physical view*, see figure 1.6, complement each other and make up the hardware description of the new BoB, see figure 3.1. This hardware description can be implemented in any hardware description language, in our case in VHDL.

The **functional view** identifies the main functions of the system and decomposes them into sub-functions which together make up a functional flow, pictured as a functional graph. In the downward vertical direction an edge in the functional graph asks the question "How?", referring to how the function in the node above is going to be performed. In the upward vertical direction an edge in the functional graph asks the question "Why?", referring to why the function in the node below needs to be performed. The directed nodes, which mainly are horizontal, shows the data flow of the whole function. The use cases or use scenarios of the system, in our case identified in the requirement specification, is the starting point of the functional view.

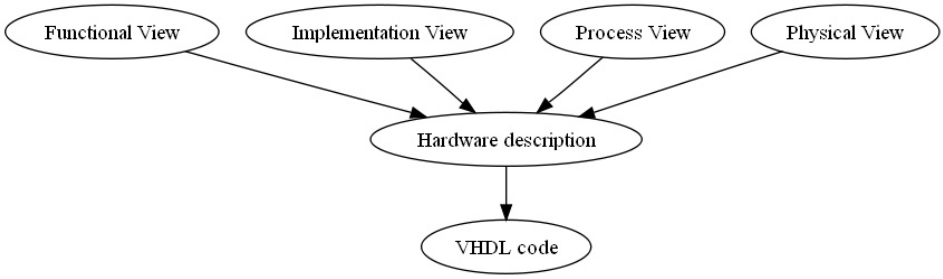


Figure 3.1. The four complementing views of the architecture.

The **process view** describes the behavior of the system is terms of data flow, memory allocation, configuration registers, timing analysis and bandwidth analysis. The process view also addresses the real time performance of the system.

The **implementation view** describes how the hardware modules are organized in hardware abstraction layers, a concept borrowed from the software domain. The implementation view addresses the modularity of a system as well as its maintainability, as long as the interface between modules is not changed it is possible to exchange or modify a module. Furthermore the implementation view also addresses portability of the system and its different modules across different hardware components. The implementation view together with the process view is the basis for the choice of hardware for the design; microcontroller, DSP or FGPA etc. Finding a balance between the performance of the hardware and the cost of the hardware is part of this process.

The **physical view** is a description of how the modules in the implementation view are mapped to the actual hardware and interconnected with each other. Whereas the implementation view is hardware platform independent the physical view is not.

3.2.1 Functional view

The new BoB together with a PC is required to perform four main tasks or functions. These are

- Record data, see figure 3.2
- Playback data, see figure 3.3
- Configure the BoB, see figure 3.4
- Get Diagnostics of the BoB, see figure 3.5

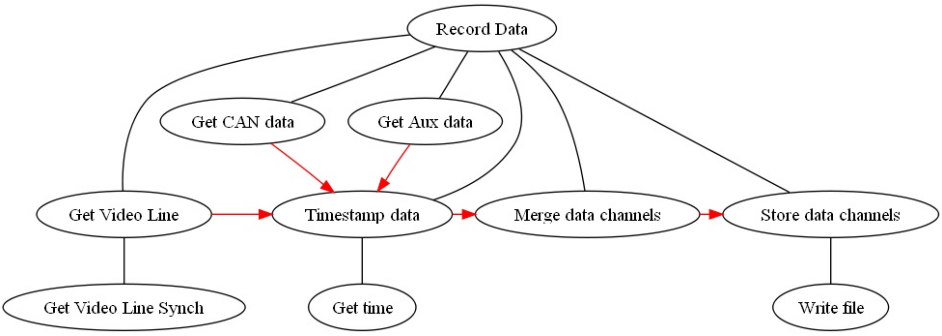


Figure 3.2. Functional graph of the function "Record Data".

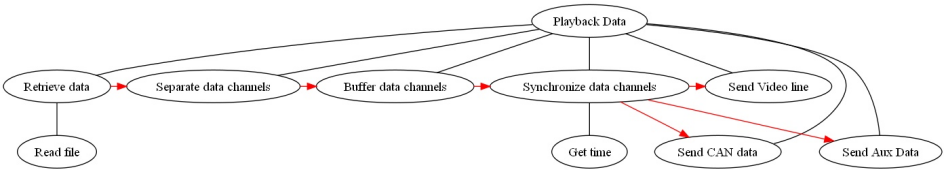


Figure 3.3. Functional graph of the function "Playback Data".

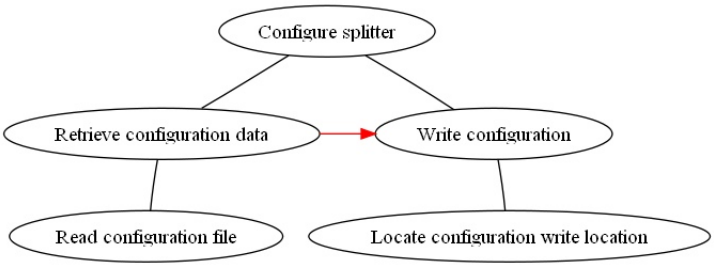


Figure 3.4. Functional graph of the function "Configure".

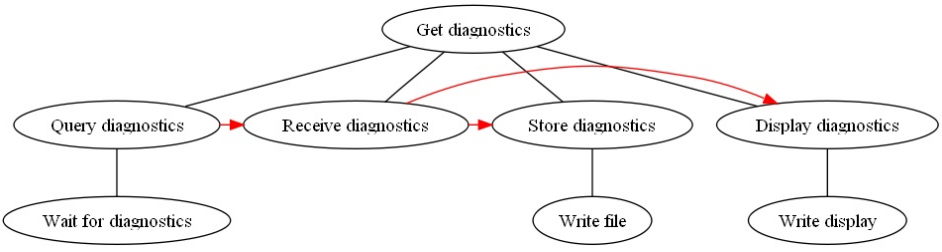


Figure 3.5. Functional graph of the function "Get diagnostics".

3.2.2 Process view

The process view and the implementation view does not, as can be seen in figure 1.6, follow after each other but are concurrent steps in the design process, therefore to fully understand the process view one has to understand the implementation view, as they describe the same system from different viewpoints.

PC communication link One of the main issues with the current BoB is its limited bandwidth, which is limited both by the LVDS link and by the FireWire link of 400 Mbit/s. The new BoB is required to have a bandwidth of at least 618 Mbit/s. Candidate technologies for the communication link between the BoB and the PC with high enough bandwidth are listed in table 3.1.

Technology	**Theoretical bandwidth**
FireWire 800 (IEEE 1394b-2002) [6]	800 Mbit/s
Gigabit Ethernet [7]	1 Gbit/s
SATA [8]	1.5 Gbit/s
USB 3 [9]	5 Gbit/s

Table 3.1. Candidate technologies for the PC communication link.

Of the four candidate technologies, USB 3 can immediately be disregarded because it is a draft standard at the moment and hardware components can be bought earliest in 2010. Examining FireWire 800 one realizes that it has a strong heritage from FireWire 400, but an increased bandwidth. Therefore the PC will still need an extension card with device drivers and since the cabling is simular to that of FireWire 400, it still may disconnect accidentally. With all the issues from the current BoB in mind regarding FireWire, this technology was disregarded. Gigabit Ethernet and SATA were developed with different goals in mind, Gigabit Ethernet for networking and SATA mainly for connection with hard drives. A possibility would be to equip the BoB with a hard drive connected over a SATA link, this would alter the need of connecting the BoB to a PC at all times. However such a solution would not in an easy way support real time viewing of the video being recorded, something that surfaced as a preferable property of the BoB. Gigabit Ethernet on the other hand is de facto standard on motherboards, and is supported by almost every OS there is. The concern was whether a sustained bandwidth of 618 Mbit/s could be obtained using Gigabit Ethernet and an non real time OS, such as Windows XP. In his article "Driven to Distraction" [10], Wilson examines a number of different Gigabit Ethernet drivers and Internet Protocol (IP) stacks for Windows XP. The article concludes that a high sustained bandwidth with Gigabit Ethernet and Windows XP on a modern PC of today, 2009, is not a problem. Using special drivers the CPU load was decreased with 10-15%. Therefore Gigabit Ethernet was chosen for the communication link between the BoB and the PC. There are a number of advantages with Gigabit Ethernet over SATA; it supports IP, which will enable the BoB to host web services and

have a web interface, its cabling has a mechanisms to prevent it from accidentally disconnecting and it is a cheap technology due to its large volumes.

Time stamping The different data streams recorded by the BoB need to be time stamped in order to be played back synchronously. The current BoB is merging, hence time stamping, the video and the CAN data in the PC and therefore the synchronization is lost due to issues previously discussed. The alternative is to time stamp the data streams as soon as they are retrieved in the BoB. By letting the BoB be responsible for the synchronization a communication protocol supporting real time data is needed in the BoB. The choice of Gigabit Ethernet infers that IP [11], will be used between the BoB and the PC. On top of IP there is a transport layer and on top of that there is an application layer, in the TCP/IP stack. For a transport protocol the choice is either User Datagram Protocol (UDP) [12], or Transmission Control Protocol (TCP) [13]. Because of the complexity of TCP and the simple needs of the BoB, UDP was chosen. For an application protocol Real-time Transport Protocol (RTP) [14], was chosen, mainly because it is the only reasonable choice for real-time applications and because it suits the needs of the BoB perfectly. The RTP protocol supports a sequence number, a time stamp, a unique identifier and a length indicator for every RTP packet, see figure 3.6. The sequence number will help identify missing packets, the time stamp will enable synchronization, the identifier will separate the different kinds of data; video, CAN, etc and the length indicator will enable correct routing of data packets in the BoB.

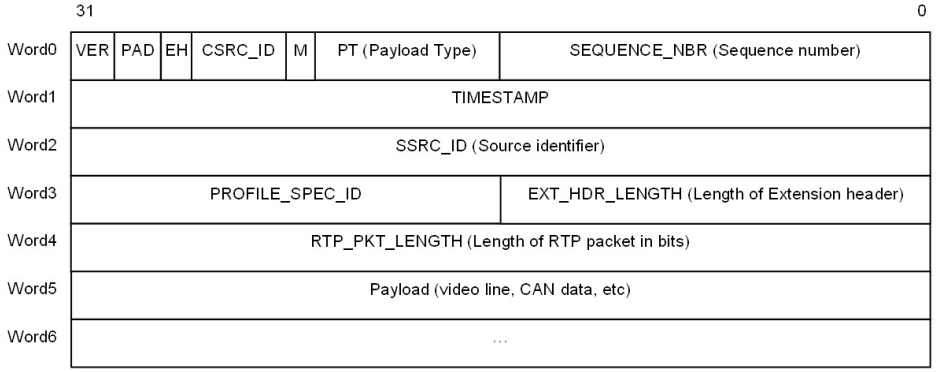


Figure 3.6. RTP protocol header; five 32 bit words long.

Real time performance For the BoB not to be dependent on a PC with real-time performance, it needs to have a buffer for data itself. The on chip memory of an FPGA is limited and it will be needed for the actual system design, therefore an off chip memory solution is needed. There are multiple available memory types, for the BoB a 32 bits wide 16 MB SDR SDRAM was chosen because of its

cost effectiveness. Suppose half the memory is used as a video buffer, that would correspond to 0.8 seconds of NightVision video, plenty for use as a buffer to ensure real time performance.

Timing Internally the data in the BoB is designed to propagate with a width of 32 bits. The reason therefore is the data width of the memory and the fact that the clock frequency of the memory is the same as the main clock domain of the BoB. To determine a minimum clock frequency of the BoB one has to take into consideration the bandwidth of the different camera systems, as well as the number of memory accesses for the data. The bandwidths were determined in section 2.3 and all data will first be written to the buffer memory once and then read from the same. The memory accesses will not be possible to perform back to back, there will be overheads in terms of the operation of the memory access protocol and the fact that the memory needs to be refreshed. In the minimum clock frequency calculation the overhead is assume to be 75%, table 3.2 shows the minimum clock frequency for the different systems. A clock frequency of 70 MHz will be enough for all systems, if the over head assumption is correct.

System	Bandwidth	Bit width	Overhead	Clock freq.
NightVision	80 Mbit/s	32 bits	75%	8.75 MHz
StereoVision	256 Mbit/s	32 bits	75%	28.0 MHz
Future system	270 Mbit/s	32 bits	75%	29.5 MHz
Future StereoVision	618 Mbit/s	32 bits	75%	69.6 MHz

Table 3.2. Minimum clock frequency calculation for the BoB for the different camera systems.

Data flow The data through the BoB flows either from the FPGA drivers to the Ethernet controller for recording, or the opposite way for playback. Figure 3.7 shows the data flow.

3.2.3 Implementation view

The modules of the BoB can be classified into to three different classes, or layers. There are *the Routing layer*, *the Networking layer* and *the Packetizing layer*.

Routing Layer The top layer of the hardware abstraction layers is the routing layer which operates on data packetized in RTP packets. The routing layer routes packets from one channel to another, without knowing the current operation of the BoB; record or playback. The modules that make up the routing layer is the *channel router* and the *synchronizer*.

Networking Layer The middle layer is the networking layer, its task is to handle the IP and the UDP protocols as well as separating UDP from TCP packets. Whether for example IP version 4 (IPv4) or IP version 6 (IPv6) [15] is usedx<

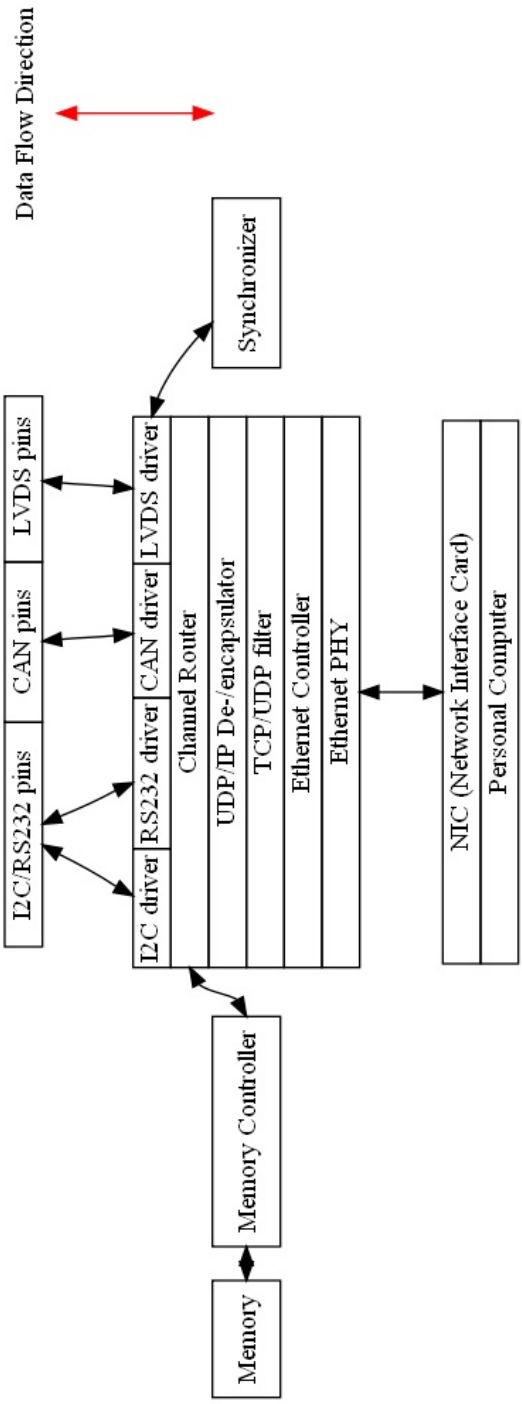


Figure 3.7. Data flow in the BoB.

as the IP protocol is known only by the networking layer and is transparent for the other layers. The *UDP IP De-/encapsulator* and the *TCP/UDP filter* are the modules that are part of the networking layer.

Packetizing layer The lowest layer is the packetizing layer, modules in this layer handles data from off chip components and packetizes it in RTP packets, except for the Ethernet Controller. As the figure shows the packetizing layer interacts both with the Routing layer and the Networking layer, contrary to praxis where every layer interface only to one other layer above itself. The modules in the Packetizing layer have multiple clock domains which the data crosses. The modules in this layer are the *Ethernet Controller*, the *LVDS driver*, the *CAN driver*, the *I^2C driver* and the *RS232 driver*.

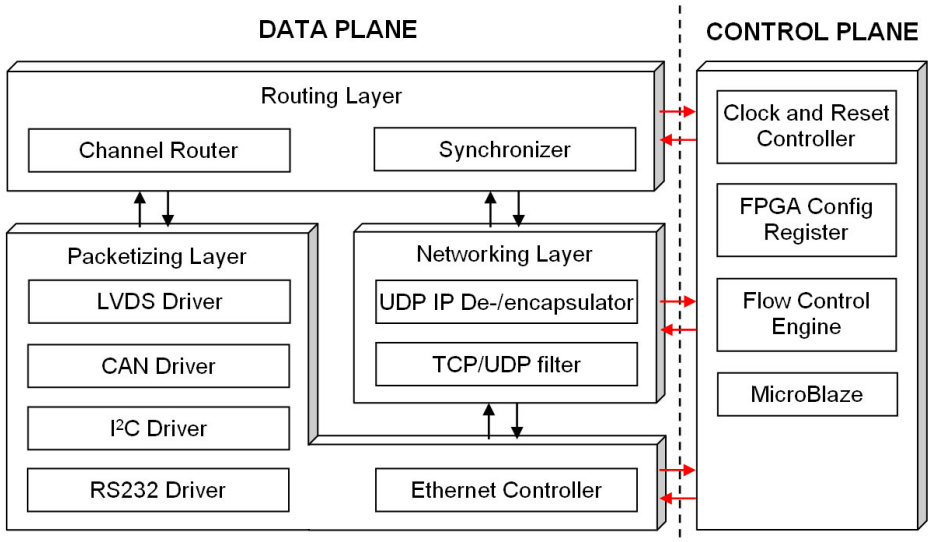


Figure 3.8. Implementation view

Data and control plane The modules of the BoB can, apart from being classified into hardware abstraction layers, also be classified into either a data plane or a control plane. Every module that is part of the hardware abstraction layers is a module that is part of the data flow, therefore the hardware abstraction layers make up the data plane. The remaining modules will be part of a control plane, modules handling the control of the BoB. The modules in the control plane is the *Clock and reset controller*, the *FPGA Config Register* and the *Flow Control Engine*. These modules interact with modules of every layer in the data plane, as shown in figure 3.8. The *MicroBlaze* is also part of the control plane.

Hardware Once the data and the control plane of the implementation view of the BoB have been established it is possible to consider the choice of hardware on which to implement the BoB. Considering only the clock frequency calculations in section 3.2.2 both an FPGA solution as well as a processor based software solution seem possible, available processors operate at hundreds of MHz. However the modules in the packetizing layer in the data plane interface with technology specific integrated circuits (ICs). These are ICs for LVDS serialization and deserialization and for CAN, Ethernet and serial communication. As the ICs for LVDS signaling have manufacturer specific communication protocols, glue logic would be needed to connect the LVDS IC to a processor bus. For this reason, together with the fact that a prototype would not easily be built for such a processor based system, the BoB will be implemented on an FPGA. For a high usability, even though the design is implemented on an FPGA, a soft processor core will be designed into the BoB.

As the FPGA used will be from Xilinx, as it is the FPGA supplier of Autoliv Electronics, the soft processor core will be their MicroBlaze processor [16]. The Microblaze has support for the lightweight TCP/IP stack, lwIP, which is an open-source implementation of the TCP/IP protocol stack originally by Adam Dunkels [17]. The Microblaze will run a web interface for status and control of the BoB and it will communicate with the PC over TCP/IP. In figure 3.8 the MicroBlaze is part of the control plane.

3.2.4 Physical view

The physical view, pictured in figure 3.9, shows how the different modules of the BoB are interconnected. All of the modules will be implemented on the FPGA, anything that is off the FPGA has a block arrow connected to it. The grey solid boxes shows the different clock domains in the FPGA. There is a main clock domain with the clock *clk_main*. Every IC connected to the FPGA, such as the LVDS deserializer, the LVDS serializer, the CAN IC and the other serial ICs operate with their own clock. Therefore there are multiple clock domains which the data needs to cross.

The module MPMC is a Multi Port Memory Controller, which is an intellectual property from Xilinx. This module provides an easy to use interface to the SDRAM for the channel router and at the same time it allows the MicroBlaze to connect to the SDRAM. The SDRAM access is time multiplexed between the channel router and the MicroBlaze. In figure 3.9, the Microblaze is dotted, indicating that it is not a necessity for the design to function. The MicroBlaze is connected to the FPGA config register, which it can read and write for status and control purposes. The FPGA config register can also be read and written through the channel router.

Operation The BoB operates in either record mode or in play back mode. In record mode, the LVDS driver receives video data from the LVDS deserializer. The data is time stamped and one full video line is buffered and packetized in an RTP

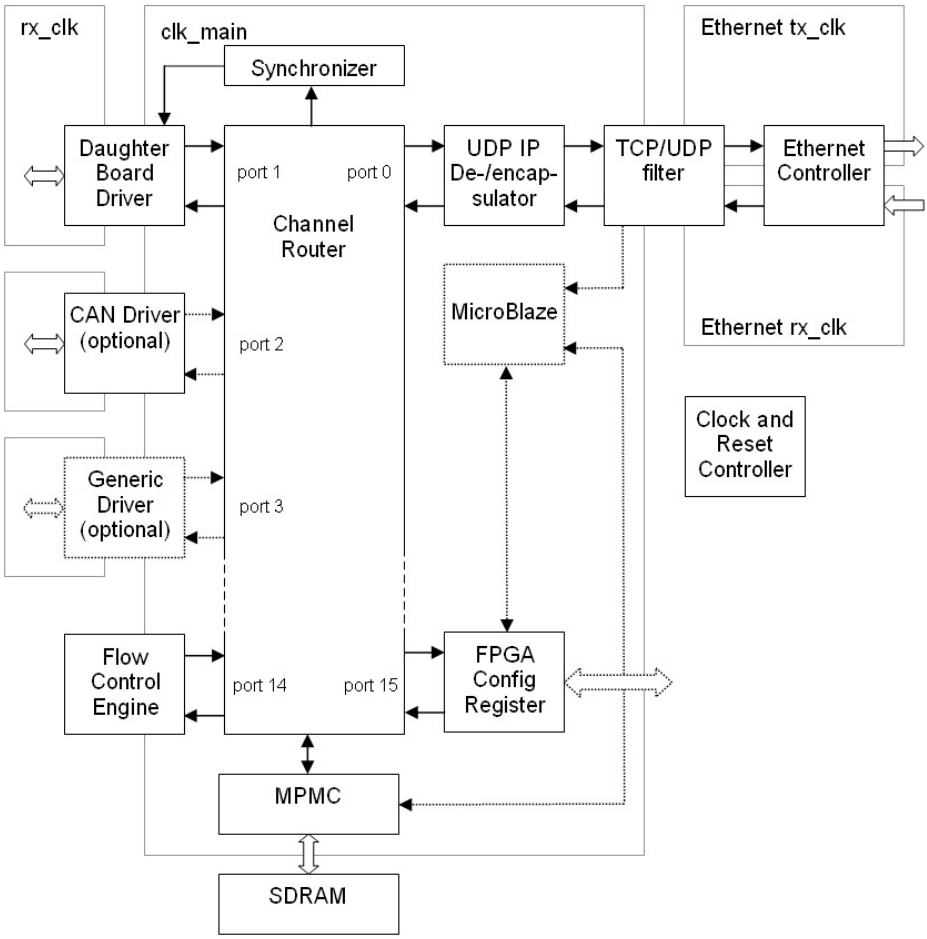


Figure 3.9. Block diagram of the physical view of the BoB. The block arrows indicate signals off the FPGA.

packet, before it is transmitted to the channel router. The channel router serves the transmission requests from the channels in a round robin fashion. Once the incoming data port of the channel router is available it receives the RTP packet from the LVDS driver. The UDP IP De-/encapsulator is constantly requesting packets from the channel router. Once the channel router has buffered one RTP packet of video or CAN data for the UDP IP De-/encapsulator it starts transmitting that packet to the same. The UDP IP De-/encapsulator adds an UDP and an IP header with correct packet length indicators, checksums, ports and addresses. Once that is done the data packet with the video line, which now is an IP packet, is transmitted to the TCP/UDP filter. The TCP/UDP filter merges data from the UDP IP De-/encapsulator and the MicroBlaze and adds an Ethernet header to the IP packet before it transmits it to the Ethernet Controller. The Ethernet Controller, which is an intellectual property from Xilinx, interfaces to the Ethernet PHY off the FPGA. The video line is then received in the PC as an Ethernet frame.

For playback the operation is reversed as the Ethernet Controller indicates that it receives Ethernet packets. The TCP/UDP filter filters the UDP packets for the UDP IP De-/encapsulator and the TCP packets for the MicroBlaze. The UDP IP De-/encapsulator checks the IP address and the UDP port of the packet as well as the IP checksum, before the IP and UDP headers are stripped. If the packet was correct the remaining RTP packet is requested to be transmitted to the channel router. Once the channel router accepts the RTP packet it is placed in the buffer memory of the channel for which the data of the RTP packet is meant. In playback mode the LVDS driver is requesting data and once it receives data it checks its time stamp and transmits the data to the LVDS serializer at the correct time. The synchronizer is used both for time stamping of incoming packets and for synchronization of outgoing packets. For the first outgoing packet the channel router sets the synchronizer accordingly, for the synchronization to function properly and such that the data buffer in the channel router has been filled enough.

Flow control The Gigabit Ethernet link between the BoB and the PC is capable of transferring data at up to 1 Gbit/s. None of the camera systems have a bandwidth that high and therefore the data flow need to be controlled when the BoB is in playback, otherwise the buffer memory of the BoB would overflow. The flow control engine requests data for the buffer memories and it uses buffer memory fill levels to determine when to request data and when stop requesting data. The software on the PC serves the data requests of the BoB and data congestion is avoided.

3.3 Module design

Every module of the design will not be discussed in this section, in appendix C the original design specification is attached. It includes module descriptions of every module of the BoB.

3.3.1 Channel Router

The channel router is at the heart of the BoB and routes data 32 bits wide between sixteen channels. It assumes RTP packets and uses the length in the RTP header and the SSRC identifier of the RTP header to route the packets to the correct destination. RTP packets from channels 1-15 are automatically routed to channel 0, the UDP IP De-/encapsulator, this is typical for operation in recording mode. RTP packets from channel 0, hence the PC, are routed to the channel corresponding to the SSRC identifier of the RTP header, this is typical for operation in playback mode.

The channel router has two data ports, one for incoming data and one for outgoing data, see figure 3.10. Every module connected to the channel router shares these data ports but have independent data request and acknowledgement signals for transmission and reception, a total of four signals for every channel. The channel router has two arbitration units, one for transmission of data to the channel router and one for reception of data from the channel router. A module can request both to transmit and to receive from the channel router at the same time, but each arbitration unit will only queue one request per module. This way no module will dominate the channel router. Internally the channel router has two FIFOs, First In First Out buffer memories, that can hold at least two RTP packets of maximum size. These allow the interface protocol between the channel router and the module to transfer one whole RTP packet without pausing during the transmission.

The protocol of the MPMC, which interfaces to the SDRAM, allows for burst writes and burst reads to and from the SDRAM. The channel router always writes and reads the SDRAM in burst mode to minimize the overhead of the SDRAM. The processes which write to and read from the SDRAM look up the write and read address of the current channel in the SDRAM address/fill control register file.

Interface protocol Figure 3.11 shows the interface protocol of the channel router. The channel router only answers to transmission or reception requests and is therefore transparent to the mode of operation. As soon as there is a request for either data transmission or reception the arbitration unit of the channel router queues that request and the module is not allowed to cancel its request. Once the channel router acknowledges a certain module's request that module must be ready to either transmit or to receive data the next clock cycle. Once a transmission to the channel router is finished the module sets its request signal low and no more data is transferred. For a reception from the channel router, the channel router sets the acknowledgement signal low for the currently receiving channel once the data is finished, this is the scenario in figure 3.11. The interfaces between all other modules in the design use the same interface protocol as the channel router.

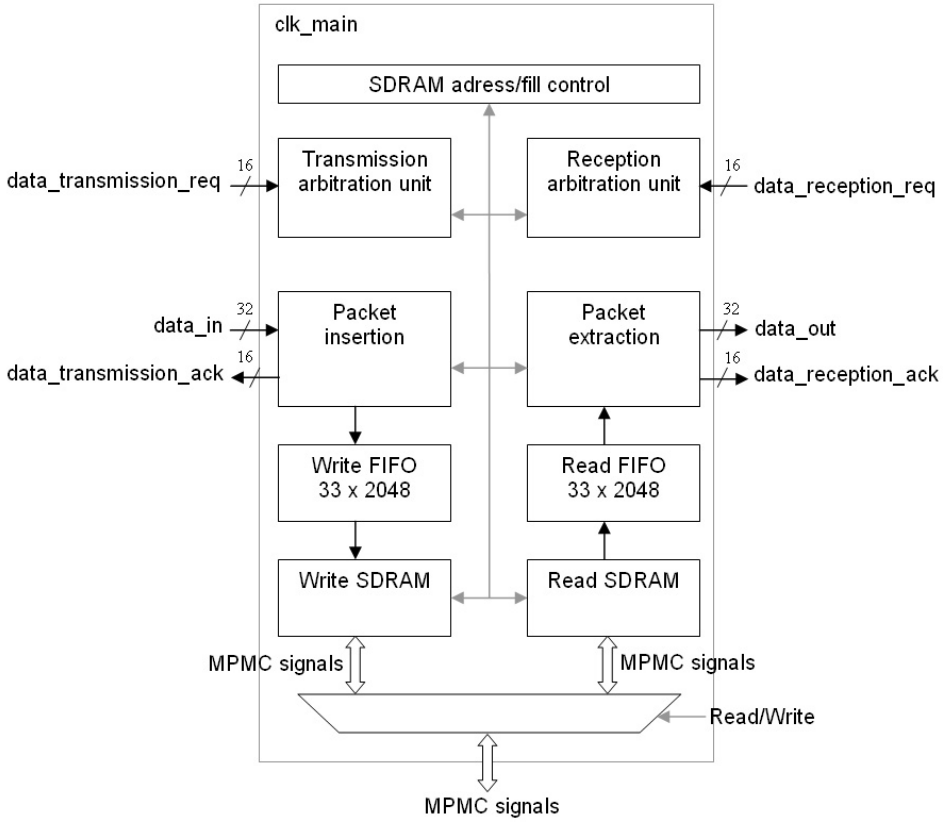


Figure 3.10. Block diagram of the channel router.

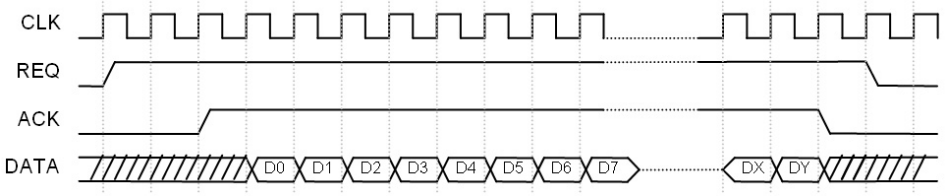


Figure 3.11. Timing diagram for the channel router interface.

3.3.2 LVDS driver

The LVDS driver interfaces to both an LVDS deserializer, for video recording, and an LVDS serializer, for video playback. Since the different camera systems use different LVDS links, the LVDS driver module is different for every system. The data protocol used over the LVDS link between the camera and the ECU of a system is also different for every system. Figure 3.12 shows the design of the LVDS driver for NightVision with its video protocol [18]. The design in the figure is only for recording and relaying of video data.

For NightVision the deserialized data received in the LVDS driver is 24 bits wide. The NightVision video protocol only specifies that the 20 least significant bits (LSB) are used. The four most significant bits (MSB) of the 20 bits used are control bits, where one bit is a parity bit of the other 19 bits, another is the video line synchronization bit and the two other bits are reserved for future use. The remaining 16 LSB are the image word, hence every pixel could have a pixel depth of 16 bits. Currently only 14 of these 16 bits are used for every pixel, nevertheless all 16 bits need to be transmitted. The LVDS driver has an asynchronous FIFO, in which data is written every clock cycle of the LVDS deserializer. As soon as there is a line in the FIFO it is read out in the main clock domain where the parity bit is checked and the video is synchronized on video frame and video line basis. Every new video line, whether it is a video, control or blank line, is time stamped with the 32 bit counter from the module synchronizer. To every line an RTP header is added and while waiting for a full video line to be transmitted to the channel router the RTP packet is stored in a FIFO.

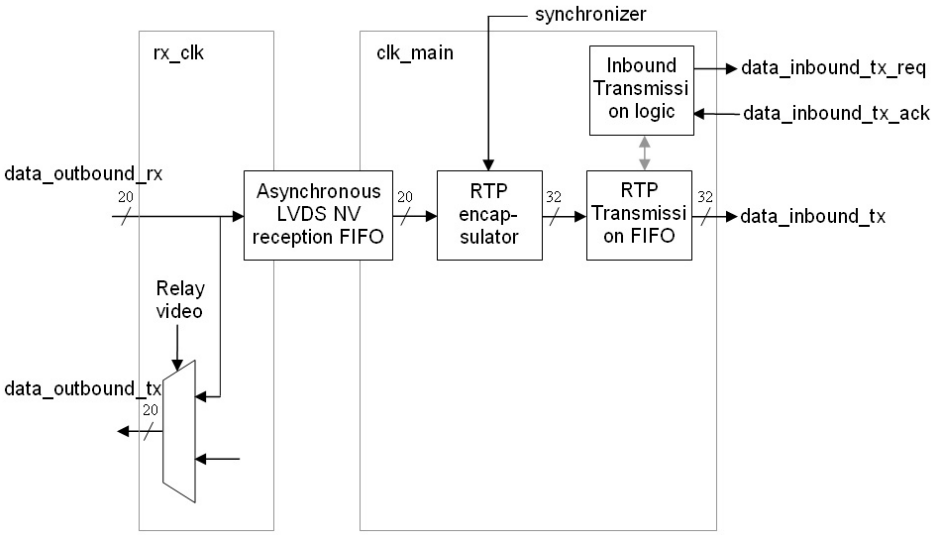


Figure 3.12. Block diagram of the LVDS driver for NightVision.

Chapter 4

Implementation

This chapter presents the implementation of the BoB as a VHDL model synthesized and built for a prototype hardware.

4.1 Model and simulation

The design of the BoB has been modeled in VHDL. Every module of the design except for the MPMC, which is a memory controller, and the Ethernet Controller, which both are intellectual properties of Xilinx, has been custom made. The size of the whole design is approximately 6000 lines and test benches for module tests and system level tests are also approximately 6000 lines. Every module has passed unit tests and the whole design has been tested in system tests, according to the test driven development methodology [4].

4.1.1 Overhead simulation

Apart from testing the design for correctness the VHDL model of the BoB has been simulated to estimate the overhead occurring when writing and reading to and from the SDRAM with the MPMC from the channel router. The SDRAM is the bottleneck in the design since all data needs to be both written and read from the SDRAM, in comparison to all other modules, where the data flow is through those module. To estimate the overhead three different situations were simulated.

Situation 1 In this situation a very short recording scenario is simulated, the UDP IP De-/encapsulator, an LVDS driver and a CAN driver was connected to the channel router. The two drivers sent two very short RTP packets each to the UDP IP encapsulator, through the channel. As table 4.1 shows the overhead in situation 1 is 169%, which is more than twice the estimated overhead in paragraph Timing of subsection 3.2.2.

Situation 2 In this situation a short recording scenario is simulated, the UDP IP De-/encapsulator and an LVDS driver was connected to the channel router.

Situation 1	
Packet 1 (Length in 32 bits words)	51
Packet 2 (Length in 32 bits words)	18
Packet 3 (Length in 32 bits words)	81
Packet 4 (Length in 32 bits words)	18
Total length (in 32 bits words)	168
Data in both directions (in 32 bits words)	336
Simulation clock	20 ns/clock cycle
Ideal time without overhead	*6720 ns*
Simulation time	*18110 ns*
Overhead	**169%**

Table 4.1. Overhead simulation of situation 1.

The LVDS driver sent ten RTP packets of almost maximum length, which is the packet length close to that of the future camera system. As table 4.2 shows the overhead in situation 2 is 55%, which is below the estimated overhead of 75% in paragraph Timing of subsection 3.2.2.

Situation 2	
Packets (Length in 32 bits words)	379
Number of packets	10
Total length (in 32 bits words)	3790
Data in both directions (in 32 bits words)	7580
Simulation clock	20 ns/clock cycle
Ideal time without overhead	*151600 ns*
Simulation time	*234770 ns*
Overhead	**55%**

Table 4.2. Overhead simulation of situation 2.

Situation 3 In this last situation a recording scenario of approximately one video frame is simulated, the UDP IP De-/encapsulator and an LVDS driver was connected to the channel router. The LVDS driver sent 500 RTP packets of almost maximum length, which is the packet length close to that of the future camera system. As table 4.3 shows the overhead in situation 3 is 49%, which is below the estimated overhead of 75% in paragraph Timing of subsection 3.2.2.

Conclusion To conclude the overhead simulation, it is clear that the channel router will perform at an overhead less than 75%, which was the assumption in the minimum clock frequency calculation in paragraph Timing in subsection 3.2.2. The fact that situation 1 gives an overhead of 169% is merely due to the design of the channel router which essentially works as a pipeline which needs to be filled and

Situation 3	
Packets (Length in 32 bits words)	379
Number of packets	500
Total length (in 32 bits words)	189500
Data in both directions (in 32 bits words)	379000
Simulation clock	20 ns/clock cycle
Ideal time without overhead	*7580000 ns*
Simulation time	*11321670 ns*
Overhead	**49%**

Table 4.3. Overhead simulation of situation 3.

emptied. When more packets are sent through the channel router the time to fill and to empty the pipeline becomes increasingly small compared to the total time. The overhead seems to level out at approximately 50% as it only decreases from 55% to 49% increasing the number of packets from ten to 500. With a simulated overhead of ∼50%, the design will perform at the required bandwidths as long as it can be built to run at ∼70 MHz.

4.2 Implementation on prototype hardware

To prove the design of the BoB it was implemented in hardware. For the implementation of the VHDL model of the BoB on the prototype hardware the FPGA development software design suite from Xilinx has been used, it includes ISE Foundation™ Software, Platform Studio and the EDK and ChipScope™.

4.2.1 Prototype hardware

The prototype hardware needed for the BoB had to include an FPGA, an SDRAM memory, a Gigabit Ethernet interface and general purpose input/output pins to connect an LVDS deserializer to the FPGA. The prototype hardware used is a development board, the ExtremeDSP Spartan-3A DSP Development Board [19], with a Xilinx FPGA. The FPGA is a Spartan-3A DSP 3400A, which is one of the bigger FPGAs in the Xilinx Spartan low-cost FPGA line. The FPGA has more than enough system gates needed by the design of the BoB, but approximately the number of user input/output pins that the BoB use. Therefore the FPGA chosen in the actual BoB hardware is a Spartan-3A DSP 1800A FPGA, which has less number of system gates but approximately the same number of pins.

Figure 4.1 shows the prototype hardware setup with an LVDS deserializer connected to the FPGA of the development board via the general purpose input/output (GPIO) list. The prototype supports recording of NightVision video, where the IR camera is connected to the LVDS connector. Because of the limited number of GPIO pins only a single LVDS deserializer can be connected to the de-

velopment board, there are no pins to connect a CAN IC or other devices. As the figure of the prototype hardware shows the development board is equipped with DDR2 SDRAM memory, which is on the backside of the printed circuit board (PCB). This memory is different from the Single Data Rate (SDR) SDRAM that the actual BoB hardware will be equipped with. The memory controller, the MPMC, was chosen partly because of this, since the back end of the MPMC easily can be changed to interface either to a DDR2 SDRAM memory or an SDR SDRAM memory.

4.2.2 Implementation result

The VHDL model of the BoB used in the implementation of the prototype did not include the soft processor, the MicroBlaze, for running a web interface for the BoB. The MicroBlaze system was evaluated separately on the FPGA development board, using a modified template from Xilinx [20]. In the scoop of the master thesis the integration of the MicroBlaze into the VHDL model was too timeconsuming, and was therefore left out.

Synthesis The synthesis of the VHDL model of the BoB, with Xilinx ISE Foundation^{TM} Software, yields the shorted summary presented in table 4.4. Important to notice is the maximum clock frequency, which is 80.97 MHz. This means that the design meets the requirement of the minimum clock frequency calculation in paragraph Timing of subsection 3.2.2. Using the simulated overhead of 50% and a clock frequency of 80 MHz the implementation of the BoB should theoretically be able to handle a bandwidth of ∼850 Mbit/s, well above the required 618 Mbit/s.

Build design summary report	
Finite State Machines (FSMs)	25
Adders/Subtractors	129
Counters	5
Accumulators	21
Registers (Flip-flops)	3248
Comparators	66
Multiplexers	102
Maximum clock frequency	*80.97 MHz*

Table 4.4. Implementation synthesis design summary.

Build Building the synthesized design for the prototype hardware yields the shorted summary presented in table 4.5. The logic utilization of the FPGA is low, and the IOB utilization is modest. One has to remember that in the final design there will be both an LVDS deserializer and an LVDS serializer, as well as multiple

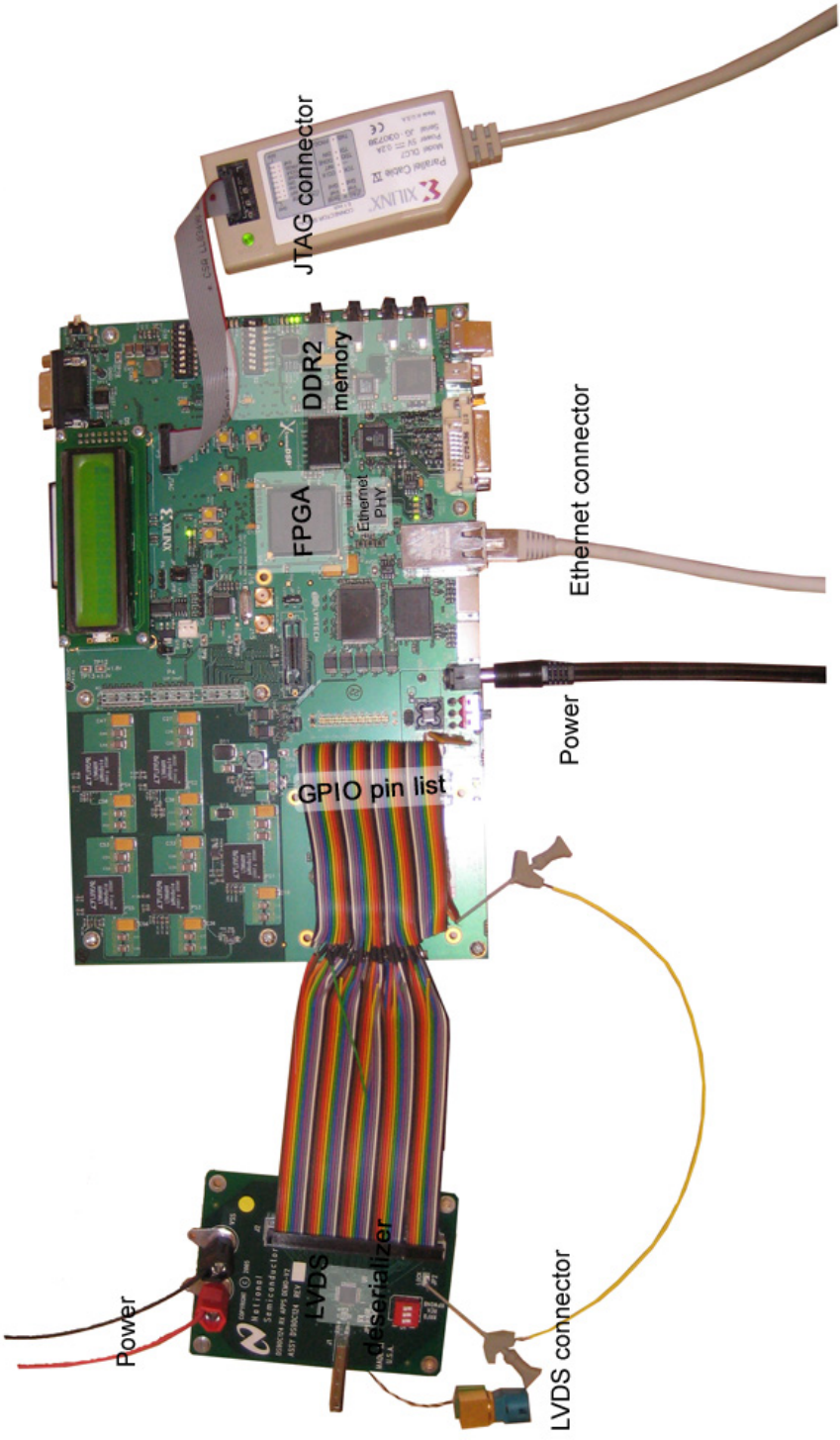


Figure 4.1. FPGA development board, with labels for the components used in the BoB design.

serial drivers together with a soft processor. Therefore the FPGA will be better utilized using the final design.

Synthesis design summary report		
Logic Utilization		
Number of Slice Flip Flops	6,167 out of 47,744	12%
Number of 4 input LUTs	9,020 out of 47,744	18%
Logic Distribution		
Number of occupied Slices	6966 out of 23872	29%
Input/Output Buffers (IOBs)		
Number of External IOBs	126 out of 469	26%
Number of External Input IOBs	37	
Number of External Output IOBs	53	
Number of External Bidir IOBs	36	
Other		
Number of BUFGMUXs	5 out of 24	20%
Number of DCMs	2 out of 8	25%
Number of RAMB16BWERs	37 out of 126	29%
Total equivalent gate count for design	2575473	

Table 4.5. Implementation build design summary.

Chapter 5

Result

This chapter presents the results of the master thesis, as well as future work that need to be done for the BoB to be a finalized product.

5.1 BoB prototype

The prototype of the BoB, which was developed for the NightVision system, has successfully been tested in a recording scenario. Figure 5.2 shows one video frame of a sequence recorded using the prototype. The scene is typical for the NightVision system, showing the rear end of a car, where white indicate warmth and black indicate cold. As development of software for the BoB was not part of the master thesis the NightVision video frame has been extracted with a Matlab script working on IP packets from the BoB, stored on the PC with an IP packet sniffer. In the final development and testing system, in which the BoB will be part, a commercial software framework for recording and playback of data of all kinds will be used. Since the software environment for the BoB has not been developed yet, playback of video and data through the BoB is not possible and therefore the prototype of the BoB only supports recording of data. However configuration and status of the BoB, setting and reading the FPGA config register, has successfully been tested and proves that playback will work once there is a software to play back video from the PC.

5.2 Future work

To transform the prototype of the BoB into a fully functional product there are a number of tasks to complete. The VHDL model needs slight modifications to fit the actual hardware on which the BoB will be implemented. There is also work to complete the VHDL model with drivers for the LVDS links and video protocols of the other camera systems as well as drivers for the serial links; CAN, I^2C and RS232. The MicroBlaze, which will provide a web interface for the BoB needs to be implemented, but most importantly the software environment which will

interface with the BoB needs to be developed.

As a last remark the new BoB has proven to fulfill at least three of the four main requirements; bandwidth, modularity and usability. The last, synchronization, has not yet been tested due to the lack of a software environment and play back support in the prototype of the BoB. Synchronization should not be a problem either as the BoB has been designed to be a flexible platform to extend for future needs.

Figure 5.1. Extraction of a NightVision video frame recorded with the BoB implemented on the FPGA development board.

Bibliography

[1] IHS (1994) *System/Subsystem Specification (SSS) (DI-IPSC-81433)*, Englewood: IHS Inc. (www.ihs.com)

[2] IHS (1994) *Software Requirements Specification (SRS) (DI-IPSC-81433)*, Englewood: IHS Inc. (www.ihs.com)

[3] Philippe Kruchten (1995) *Architectural Blueprints – The "4+1" View Model of Software Architecture*, IEEE Software Volume 12, Number 6 (November 1995), p 42-50

[4] Hellberg Tomas (2008) *Testdriven utveckling*, Linköping: ENEA Services Linköping AB

[5] *FNIR – Fusing Far and Near InfraRed imaging for pedestrian injury mitigation*, www.fnir.nu, last viewed February 25, 2009.

[6] Henehan B, Johas-Teener M, Scholles M, Thompson D (2006) *1394 Standards and Specifications Summary* Southlake: 1394 Trade Association

[7] LAN/MAN Standards Committee of the IEEE Computer Society (2005) *Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, IEEE Std 802.3™-2005* New York: IEEE

[8] Cavin Rob (2002) *Serial ATA : A Comparison with Ultra ATA Technology* Beverly Hills: Computer Technology Review, November 2002

[9] *Universal Serial Bus 3.0 Specification Revision 1.0* Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, NEC Corporation, ST-NXP Wireless and Texas Instruments, November 12, 2008

[10] Wilson Andrew (2008) *Driven to Distraction*, Nashua: Vision Design Systems, February 2008, p 47-50

[11] Postel Jon (1981) *RFC 791 – Internet Protocol Darpa Internet Program Protocol Specification*, Arlington: Defense Advanced Research Projects Agency

[12] Postel Jon (1980) *RFC 768 – User Datagram Protocol*, Marina del Rey: Information Sciences Institute, University of Southern California

[13] Postel Jon (1981) *RFC 793 – Transmission Control Protocol Darpa Internet Program Protocol Specification*, Arlington: Defense Advanced Research Projects Agency

[14] Casner S., Frederick R., Jacobson V. & Schulzrinne H. (2003) *RFC 3550 – RTP: A Transport Protocol for Real-Time Applications*, Washington DC: The Internet Society

[15] Deering S. & Hinden R. (1998) *RFC 2460 – Internet Protocol, Version 6 (IPv6) Specification*, Washington DC: The Internet Society

[16] Xilinx (2007) *MicroBlaze Processor Reference Guide*, San Jose: Xilinx Inc. (www.xilinx.com)

[17] Dunkels Adam (2001) *Design and Implementation of the lwIP TCP/IP Stack*, Kista: Swedish Institute of Computer Science

[18] Sundin Mats (2006) *NV2 LVDS video link specification*, Linköping: Autoliv Electronics AB

[19] Lyrtech (2007) *ExtremeDSP Spartan-3A DSP Development Board – Technical Reference Guide*, Quebec City: Lyrtech Inc. (www.lyrtech.com)

[20] Velusamy Siva (2008) *LightWeight (lwIP) Application Examples (XAPP1026)*, San Jose: Xilinx Inc. (www.xilinx.com)

Appendix A

Abbreviations

BoB	Break out Box
CAN	Controller Area Network
DDR2	Double-Data Rate 2
ECU	Electronic Control Unit
FIFO	First In, First Out
FNIR	Far and Near InfraRed
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GPIO	General Purpose Input/Output
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IOB	Input/Output Buffers
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IR	Infrared
LSB	Least Significant Bit
LVDS	Low Voltage Differential Signaling
MPMC	Multi Port Memory Controller
MSB	Most Significant Bit
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
RAM	Random Access Memory
RTP	Real-time Transport Protocol
SATA	Serial AT Attachment
SDR	Single Data Rate
SDRAM	Synchronous Dynamic RAM
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Appendix B

Requirement Specification

Starting on the next page the original requirement specification document is attached. The specification documents every requirement of the BoB and is therefore more accurate than chapter 2, Requirement specification, which aims at presenting an overview of the requirements focusing on the major points. In the original requirement specification the new BoB is referred to as *the Splitter* and the old BoB is referred to as *the Wicer Box*, by which name it is known at Autoliv Electronics.

Requirement specification for the next generation “Wicer box”

Revision 4, 2008-10-13

Author: Erik Irestål

Contents

1 Scope

1.1 Identification

The system that this document describes is the second generation of the currently used “Wicer box”, which is a development box within the NightVision and the StereoVision project groups at Autoliv Electronics AB, Linköping.

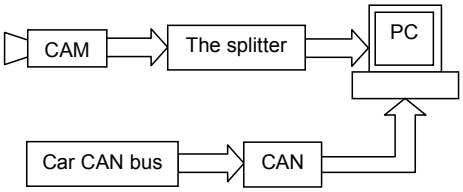
1.2 System overview

The system is a hardware box, which will be known as “the splitter”, and a software API and application run on a desktop computer. The purpose of this system is multifold; it can be used in a variety of cases, depending on the need.

1.2.1 System use cases

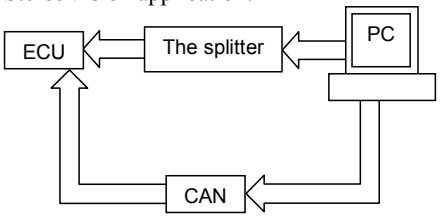
1.2.1.1 Case I: Recording data

When gathering test video driving around in a car, the splitter will enable video from the automotive camera to be recorded onto a desktop/laptop computer.



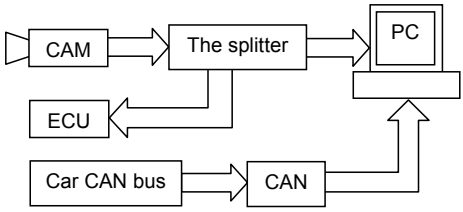
1.2.1.2 Case II: Data playback

When in the test lab, the splitter will enable playback of recorded video from the desktop computer to an ECU, the processing unit of e.g. the NightVision or the StereoVision application.



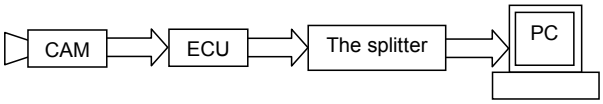
1.2.1.3 Case III: Recording tapped data

When gathering test video driving around in a car, the splitter will enable tapping and recording of data onto a desktop/laptop computer, while the NightVision or the StereoVision system functions as normal.



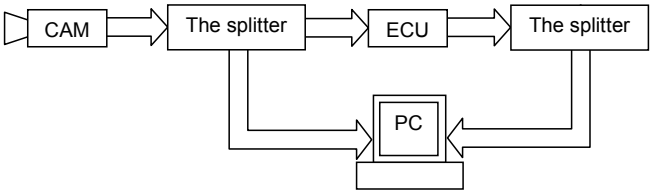
1.2.1.4 Case IV: Recording processed data

When gathering test video in a car, the splitter will enable recording of processed video from the ECU onto a desktop computer



1.2.1.5 Case V: Recording both unprocessed and processed data

Using two splitters the unprocessed and the processed video can be compared and recorded simultaneously onto only one desktop computer.



1.2.1.6 Case VI: Testing of NightVision and StereoVision algorithms

For high testability of the NightVision and StereoVision systems the splitter needs to support playback of CAN data together with video transparently from a desktop computer to the ECU. The setup is the same as for case II.

Apart from handling video as described in cases I-VI, the splitter will have the functionality to be able to control a camera using either the I^2C or the RS232 standards. The splitter will have no hardware configuration switches, instead it will be controlled and configured via the software application on the desktop computer.

1.2.2 System users

The splitter will be used for a multitude of development and testing purposes within the NightVision, the StereoVision and future camera system project groups at Autoliv Electronics AB.

- It will be used for testing during development of algorithms and verification of the implementation of the same.
- The splitter will be used for system test and final verification of the current NightVision system, assembled at the Autoliv Electronics production facility in Motala, and the StereoVision system as well as future camera systems.

- The splitter will be used in product validation, a process in which currently the NightVision system is subjected to tough environments; heat, electromagnetic radiation etc, and is tested over a prolonged time, up to thousands of hours.
- The splitter will be available to the customers of the NightVision system and the StereoVision system as well as to customers of future camera systems, for them to perform their own tests.

AEV, Autoliv Electronics Vision, in Goleta, CA, USA, assembles the NightVision cameras and they may want to use the splitter for development and testing purposes.

The splitter will also be used in the FNIR project (Fusing Far and Near InfraRed imaging for pedestrian injury mitigation) in which Autoliv Electronics AB is a partner.

1.2.3 System developer

As his master thesis project, Erik Irestål, employed by ENEA Linköping and directed by Lars Asplund, will develop the hardware box, the splitter.

1.3 Document overview

The purpose of this document is to summarize all the requirements of the next, hence the second, generation of the “Wicer box”. This document is intended to state all the requirements of every group within Autoliv Electronics AB that has an interest in the splitter. To identify the requirements, every requirement is numbered with a requirement identification number (RIN), which is a three digit number preceded by the letter R, an example would be R123. A requirement specification list is attached at the end of the document.

2 Requirements

2.1 Required modes

The splitter will be required to be able to operate in the following modes:

2.1.1 Idle (R001)

In this mode the I/O ports of the splitter and the camera connected to the splitter can be configured and controlled. The splitter is waiting either to start to record or to start to playback data in this mode.

2.1.2 Recording data (R002)

In this mode the splitter records data from one of its I/O ports to the desktop computer.

2.1.3 Playback of data (R003)

In this mode the splitter plays data from the desktop computer to one of its I/O ports.

In all of the modes, diagnostics of the splitter can be retrieved.

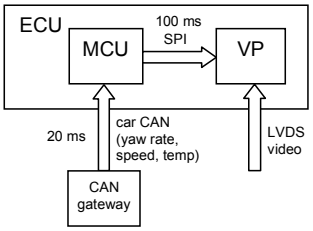
2.2 System capability requirements

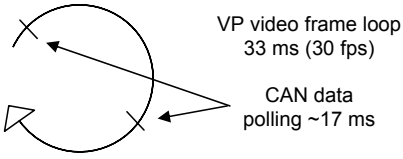
2.2.1 Video recording

The splitter is required to be able to record video from an LVDS receiver connected to its LVDS I/O ports **(R004)**. The format and protocol of the video may differ between different LVDS standards and cards. The splitter is required to be able to record video from the LVDS receiver based on National DS90C124 (http://www.national.com/pf/DS/DS90C124.html), which currently is used in the NightVision and StereoVision projects at Autoliv Electronics AB **(R005)**. The FNIR project uses an infrared camera, which has a low-rate LVDS transmitter implemented in an FPGA, transmitting over three differential pairs. The splitter needs to support and deserialize the data on those differential pairs and other custom LVDS links with up to 5 differential pairs **(R007)**. The splitter is required to be future compatible in hardware, in the sense that it should be able to accommodate other LVDS receivers for video recording than the receiver mentioned above, with only changes to its onboard firmware configuration **(R008)**.

2.2.2 Relaying video while recording

The splitter is required to be able to output video to an LVDS transmitter at the same time as video is recorded from an LVDS receiver onto the desktop computer **(R009)**. The maximum latency between the input video from the LVDS receiver and the output video to the LVDS transmitter is limited by the fact that the ECU needs to process the video from the camera together with concurrent CAN data from the car; yaw rate, speed and temperature. The figure below shows an overview of the ECU, the microcontroller, MCU, is connected to the car’s CAN bus. The car’s yaw rate, the speed of the car’s rotation, is the most critical parameter and the MCU receives it every 20 ms. The SPI link between the MCU and the Vision Processor, VP, has only got bandwidth enough to send the yaw rate to the VP once every 100 ms. With video at 30 frames per second the main loop of the VP, which is based on a video frame, executes for a maximum of 33 ms. It polls the yaw rate value twice during that loop, therefore the maximum duration between two different yaw rate values is $100 + 33/2 \approx 117$ ms. The latency between the input and the output of the video in the splitter is required to be complaint with a future increase of the bandwidth of the SPI link so that the yaw rate can be updated every 20 ms in the VP. Therefore the latency is required to be able to cope with a maximum duration between two different yaw rate values of $20 + 33/2 \approx 37$ ms **(R010)**.





2.2.3 Video playback to a single LVDS transmitter

The splitter is required to be able to play video from the desktop computer to a single LVDS transmitter connected to an LVDS I/O port **(R011)**. The format and protocol of the video may differ between different LVDS standards and cards. The splitter is required to be able to playback video to the LVDS transmitter based on National DS90C241 (http://www.national.com/pf/DS/DS90C241.html), which currently is used in the NightVision and StereoVision projects at Autoliv Electronics AB **(R012)**. In the same way that the splitter needs to support and deserialize data on custom LVDS links **(R007)**, it also is required to serialize data onto custom LVDS links with up to 5 differential pairs **(R014)**. The splitter is required to be future compatible in hardware, in the sense that it should be able to accommodate other LVDS transmitters for video recording than the transmitter mentioned above, with only changes to its onboard software configuration **(R015)**.

2.2.4 Video and data playback to a single LVDS transmitter (R016)

The splitter is required to be able not only to send video to an LVDS transmitter but also video and data combined that is being played back from the desktop computer.

2.2.5 Real-time video and CAN data processing support (R053)

Using the splitter in a car, the whole system, the splitter and the software on the computer, is required to support real-time processing of the video and the CAN data and not only recording of the same. This is used during algorithm and ECU development when powerful computers emulate the ECU of a normal NightVision, StereoVision or future system.

2.2.6 Splitter configuration from the desktop computer (R017)

The “Wicer box” can’t be configured at all without sending it back to the manufacturer. Configuration, reset and restart of the splitter are required to be performed only from the software on the desktop computer. The splitter will have no hardware configuration switches at all.

2.2.7 Splitter diagnostics from the desktop computer (R018)

To get any diagnostics from the current “Wicer box” isn’t possible. Therefore when an error appears in the test lab it can be difficult to track the cause of the error. With this background the splitter is required to have diagnostics indicating the current mode of the splitter, the status of the communication link between the splitter and the desktop computer and the status of the currently connected I/O cards. All these parameters will be monitored in software on the desktop computer.

2.2.8 Splitter status LEDs

With the background of requirement R018 the splitter is required to have LEDs, indicating its power status **(R019)** and the status of the connection to the desktop computer **(R020)**, to simplify the use of the splitter.

2.2.9 Capability of controlling camera

The different automotive cameras used in the different projects, in which the splitter will be used for development, are controlled differently. The splitter is required to support control of the cameras over both I^2C **(R021)** and over RS232 **(R022)**. The current StereoVision camera uses the CAN protocol to control the camera, therefore the software on the desktop computer, together with an, to the desktop computer connected, external CAN interface, is also required to be able to control the camera **(R023)**. The control interface is required to be in software on the desktop computer **(R024)**. Also for development purposes the control software is important, because it is easier and faster to develop algorithms for exposure control for a desktop computer than for the ECU.

2.2.10 CAN bridge (R025)

In a normal setup and in many test situations, especially the way that the NightVision system is tested currently, the camera is controlled by the ECU rather than by the desktop computer. Therefore the splitter needs to have a CAN bridge, to relay the private CAN communication from the ECU to the camera. This requirement is a result of the fact that the physical interface and the cables for LVDS and CAN data between the ECU and the camera are combined.

2.2.11 I^2C bridge (R026)

Future camera systems will use the I^2C protocol to control the camera. The splitter is required to have an I^2C bridge to relay the I^2C link between the ECU and the camera. As for requirement R025 this setup will be the normal setup for future systems, but will also be used in certain tests.

2.2.12 Emulate camera in software (R027)

Not only is camera control capabilities over I^2C **(R021)**, RS232 **(R022)** and CAN **(R023)**, important for testability of the system, in order for an ECU connected to the splitter to work properly without a camera connected to it, the desktop computer is required to emulate the camera and answer to messages sent to the splitter from the ECU. The ECU expects for example acknowledgements from the camera after having sent exposure parameters to the same. In playback mode **(R003)** only a splitter might be connected to the ECU instead of a camera and therefore the splitter is required to send "dummy" acknowledgements to the ECU. The ECU also requests temperature readings from the camera; therefore the splitter is required to send a "dummy" temperature to the ECU when asked for. This behavior is true for the ECU of all systems, but will likely only be used by the StereoVision and future projects. Nonetheless the emulation of cameras is required to be supported over I^2C, RS232, and CAN.

2.2.13 Software API (R028)

For future extensions of the desktop computer software and for customers of the NightVision, StereoVision and future systems to be able to build their own specific test applications, the splitter is required to be supplied with a software API. The software application used on the computer for video recording, camera control and camera emulation is preferably also built using the same API.

2.2.14 Video format support

The splitter is required to support the currently used raw data video format **(R029)**. This way the existing video material for the NightVision project will be supported also with the new splitter. The splitter is also required to support a possible future migration to a commercial software framework for synchronized video and data record and playback **(R030)**.

2.2.15 Video and CAN data record synchronization (R031)

The system as a whole, the hardware splitter and the software on the desktop computer, is required to synchronize the video recorded from the camera, via the splitter, and the CAN data recorded, via an external CAN interface connected to the desktop computer, from the car’s CAN bus.

2.2.16 Video time stamping (R032)

The video recorded from the LVDS receiver is required to be time stamped in order for the synchronization of requirement R031 to be accurate enough.

2.2.17 Data loss indication for the communication link between the splitter and the desktop computer (R033)

The current development box, the “Wicer box”, doesn’t indicate if video data is lost on the communication link between the box and the desktop computer. Since recorded video becomes distorted when video data is lost the system as a whole, the hardware splitter and the software on the desktop computer, is required to be able to indicate when the communication link between the splitter and the desktop computer loses video data packets. This could potentially happen when the bandwidth of the communication link is exceeded. The indication could either be an LED on the splitter or a message in the software, or both.

2.2.18 Operating system compatibility

The system as a whole, the hardware splitter and the software on the desktop computer, is required to be compatible with both Microsoft Windows XP **(R034)** and Linux **(R035)**.

2.2.19 Voltage requirement (R036)

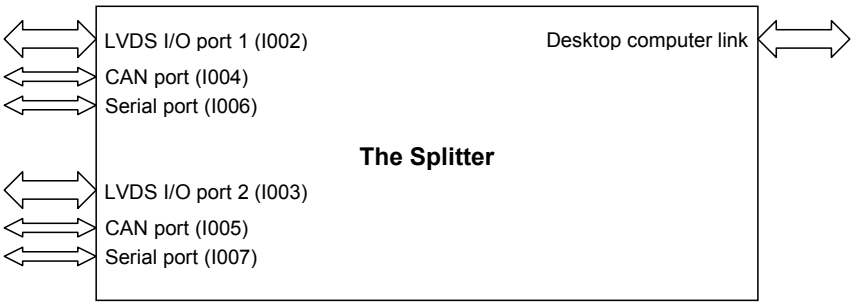
The splitter is required to be able to run at 12V, the voltage of the electrical system of a car.

2.3 System external interface requirements

2.3.1 Interface identification

The external interfaces of the splitter are:

- A communication link between the splitter and the desktop computer
- The two LVDS I/O ports
- The two CAN ports
- The two serial (I^2C/RS232) ports



The interfaces will be identified with interface identification numbers (IIN), a three digit number preceded by the letter I, for example I456.

2.3.2 Communication link between the splitter and the desktop computer (I001)

The communication link between the splitter and the desktop computer is required not to be the currently, by the “Wicer box” and the desktop computer, used FireWire link, an IEEE 1394 link **(R037)**. A multitude of problems have been experienced with the current FireWire link:

- Separate FireWire I/O cards are needed in the desktop computers.
- Now and again software hot fixes are needed to be installed under Windows XP in order for the FireWire communication link to work.
- At times Windows XP loses the connection to the “Wicer box”, even though the box is physically connected.
- The physical FireWire connector doesn’t have a mechanism hindering the connector to be pulled out of the FireWire jack accidentally. This has been a problem using the “Wicer box” in tougher environments, such as in a car when driving around capturing test video.

The communication link between the splitter and the desktop computer is required to be a standard link, in order to avoid the problems described above **(R038).** The bandwidth of the link is crucial, the FireWire link of the “Wicer box” has bandwidth 400 MBit/s and that is not enough to accommodate the bandwidth required by the StereoVision project. The bandwidth of the link is required to be enough to accommodate the higher bandwidth of the StereoVision project **(R039)**. A standard which has evolved and has multiple operation modes at different bandwidths is preferable, since the NightVision project could do with only 100 MBit/s. Such a standard is likely cheaper at 100 MBit/s than at a higher bandwidth, adding the extra bandwidth only when needed **(R040)**. An already standardized communication protocol stack both for Windows XP **(R041)** and for Linux **(R042)** is required. Lastly a communication link is required that has physical connectors that can’t be unplugged accidentally **(R043)**.

2.3.3 The two LVDS I/O ports (I002, I003)

The splitter is required to have an input for an LVDS receiver and an output for an LVDS transmitter card. The splitter needs to support the LVDS cards listed in the table below and have future compatibility to support other cards as well.

LVDS I/O card	*Receiver/Transmitter*	*Usage*	*Requirement*
National DS90C124	Receiver	Used in NightVision 2 and current StereoVision	R005
National DS90C241	Transmitter		R012

Table 2.3.3.1

Each of the two LVDS I/O ports is required to be a GPIO, general purpose input/output, with 32 pins **(R044)**. Requirement R007 and R014 will be realized within the 32 pin GPIO.

The splitter is also required to support future LVDS receivers and transmitters that have a backchannel for either I^2C or RS232 **(R045)**.

2.3.4 The two CAN ports (I004, I005)

The two CAN ports will be standard CAN ports that will relay the CAN communication between the ECU and the camera **(R025)**. For CAN communication between the camera and the desktop computer an external CAN interface connected to the desktop computer will be used **(R023)**.

2.3.5 The serial ports (I006, I007)

Either the I^2C or the RS232 protocol will run on the serial port. The serial ports will be used either to relay the communication between the ECU and the camera **(R026)**. A serial port can also be configured to control either a camera **(R021, R022)** or to connect to an ECU when emulating the camera in software **(R027)**. Since there are only two serial ports both R026 and R021 or R027 can’t be fulfilled at the same time.

2.4 System environment requirements

The environment in which the splitter will be used will vary greatly. The splitter will be mounted in cars for the purpose of recording test video, it will also be used in the product validation process where harsh environments are simulated. The splitter will also be used in production running for long periods of time day in and day out.

2.4.1 Sustainability against high or low temperatures

The splitter will be used both in high and low temperatures, for example in heat tests during product validation or in potentially cold cars. In both environments the splitter is required to be able to operate correctly **(R046)**.

2.4.2 Sustainability against moisture

The splitter is required to operate correctly in moist environments, in which it may be used **(R047)**.

2.4.3 Sustainability against electromagnetic radiation

The splitter is required to operate correctly in environments in which the electromagnetic radiation is higher than normal, such as prototype cars or test labs **(R048)**.

2.4.4 Sustainability against supply voltage variations

The splitter is required to operate correctly in environments with variations in supply voltage, which is a common phenomenon in prototype cars **(R049)**.

2.5 Computer resource requirements

The desktop computer, to which the splitter will be connected and on which the software application will run, is required to be powerful enough to handle the real-time video recording and playback **(R050)**.

2.6 System development requirements

The system development is required to be well documented, using a documentation style that is easy to understand and maintain **(R051)**. The quality and the readability of the hardware description code and the software programming code are required to be such that a third person easily can get an overview and understand the design of the splitter and extend or redo the design with ease **(R052)**.

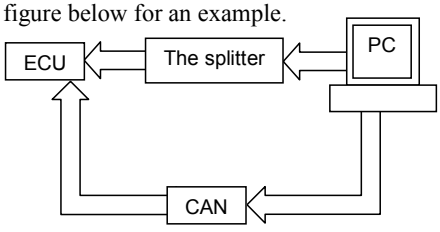
3 Notes

3.1 Future extensions

There are a multitude of extensions that the splitter could support. Some of those might not be possible to add at a later stage in the life of the splitter, while others will.

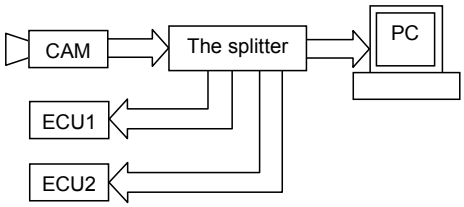
3.1.1 CAN communication capability

The capability to record or playback CAN data via the splitter or to control a camera over CAN via the splitter is a neat feature, but costly to implement. The splitter will be extendable to have such CAN capability but in an early stage, the synchronization that is required between recorded video and CAN data **(R031)** can be implemented in software on the desktop computer. Hence at an early stage the system will keep the current solution where an external CAN interface is connected to the desktop computer, over which CAN messages can be sent to and received from the ECU and the camera, see figure below for an example.



3.1.2 Capability of three or more LVDS I/O cards

A possible extension to the splitter would be to add the capability to connect more than only one LVDS receiver card and one LVDS transmitter card. There could be cases where one would want to compare different imaging algorithms on different ECUs. Then the need to connect one LVDS receiver and two LVDS transmitters would exist.



3.1.3 Internal flash memory for test video

If a flash memory is added to the splitter, it could be used as a standalone test equipment. Video from a camera could be recorded onto the flash memory and be played back directly from the flash memory without the need of a desktop computer.

3.1.4 Testability of camera over CAN

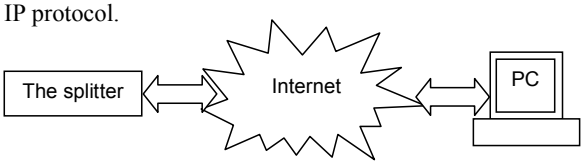
From the perspective of camera manufacturers, it would be interesting to make the splitter support testing and further diagnostics of cameras over CAN, I^2C or RS232.

3.1.5 Internet diagnostics

For the purpose of diagnostics it would be useful with the possibility to diagnose a splitter remotely over the Internet. This way Autoliv Electronics AB in Linköping could remotely access diagnostics of its customers using the splitter in their test environments.

3.1.6 Internet browser interface

An Internet browser interface for the splitter is another useful feature, which could be used to configure and diagnose the splitter over the Internet. If the splitter could be connected to an Internet router this feature could be implemented without the need for a desktop computer to always be connected to the splitter for it to be configured and diagnosed. To support this possible extension an additional requirement on the communication link between the splitter and the desktop computer is that it supports the IP protocol.



RIN	Requirement	Project			
R001	One of the splitter's operation modes is IDLE.	G			
R002	One of the splitter's operation modes is RECORDING DATA.	G			
R003	One of the splitter's operation modes is PLAYBACK OF DATA.	G			
R004	Record video from an LVDS receiver connected to the splitter's LVDS I/O ports.	G			
R005	Support LVDS receiver card National DS90C124.			NV	SV
R007	Support and deserialize custom LVDS links using up to five differential pairs.		FNIR		
R008	Support future LVDS receiver cards with only minor reconfiguration of the splitter.	G			
R009	Relay incoming video from an LVDS receiver to an LVDS transmitters while the desktop computer records the video.	G			
R010	The latency between the incoming and outgoing video of requirement R009 is required to be small enough to cope with a maximum duration between two different yaw rate values in the VP of 37 ms.	G			
R011	Playback video from the desktop computer to a single LVDS transmitter connected to a LVDS I/O port.	G			
R012	Support LVDS transmitter card National DS90C241.			NV	SV
R014	Support and serialize custom LVDS links using up to five differential pairs.		FNIR		
R015	Support future LVDS transmitter cards with only minor reconfiguration of the splitter.	G			
R016	Playback video and data to an LVDS transmitter.	G			
R017	Splitter configuration from the desktop computer.	G			
R018	Splitter diagnostics from the desktop computer.	G			
R019	LED indicating whether or not the splitter has power.	G			
R020	LED indicating the status of the connection between the splitter and the desktop computer.	G			
R021	Support in the splitter for control of camera parameters over I2C.	G			
R022	Support in the splitter for control of camera parameters over RS232.				SV
R023	Control of camera parameters over CAN using an external CAN interface connected to the desktop computer.	G			
R024	Camera control interface in software on the desktop computer.	G			
R025	CAN bridge relaying the private CAN communication between the ECU and the camera.			NV	SV
R026	I2C bridge relaying the I2C communication between the ECU and the camera.	G			
R027	Camera emulation in software over I2C, RS232 and CAN links.				SV
R028	Software organized as API + application.	G			
R029	Raw video format support in the splitter.	G			
R030	Support for commercial software framework for synchronized video and data record and playback, in case of a future migration.				SV
R031	Video and CAN data record synchronization.	G			
R032	Video time stamping to enable video and CAN data record synchronization.	G			
R033	Data loss indication for the communication link between the splitter and the desktop computer.	G			
R034	Compatibility with Windows XP.			NV	SV
R035	Compatibility with Linux.		FNIR		
R036	Splitter power supply voltage of 12 V.	G			
R037	The communication link between the splitter and the desktop computer can not be a FireWire link.	G			
R038	The communication link between the splitter and the desktop computer must be a standard link, to avoid problems experienced with the "Wicer box".	G			
R039	The bandwidth of the communication link is required to be high enough for the StereoVision project.				SV
R040	A communication link which can operate at different bandwidth at different cost.	G			

R041	A standardized communication protocol for Windows XP.			NV	SV
R042	A standardized communication protocol for Linux.		FNIR		
R043	A communication link with physical connectors that can't be unplugged accidentally.	G			
R044	Each LVDS I/O port is a GPIO with 32 pins.	G			
R045	Support for future LVDS receivers and transmitters that have a backchannel for I2C or R232.				SV
R046	Operation in high and low temperatures.	G			
R047	Operation in moist environments.	G			
R048	Operation in environments with electromagnetic radiation higher than normal.	G			
R049	Operation with variations in supply voltage.	G			
R050	The desktop computer, to which the splitter will be connected, needs to be powerful enough to handle real-time video recording and playback.	G			
R051	The system development is required to be well documented, using a documentation style that is easy to understand and maintain .	G			
R052	The quality and the readability of the code are required to be such that the code is overviewed, understood and extended/redone with ease.	G			
R053	Real-time video and CAN data processing support.	G			
IIN	**Interface**	**Project**			
I001	Communication link to the desktop computer	G			
I002	LVDS I/O card interface	G			
I003	LVDS I/O card interface	G			
I004	CAN port			NV	SV
I005	CAN port			NV	SV
I006	Serial port				SV
I007	Serial port				SV

Abbreviations

FNIR Fusing Far and Near InfraRed imaging for pedestrian injury mitigation
G Generic
IIN Interface Identification number
NV NightVision
RIN Requirement identification number
SV StereoVision

Additional Requirements/Comments:

Appendix C

Design specification

Starting on the next page the original design specification document is attached. The specification documents the design both on the system level and on the module level. There are block diagrams for every module of the BoB, which in this document is referred to as *the Splitter*. Compared to chapter 3, System design, the module descriptions of this document are more detailed.

Splitter Design Specification

Issued by Erik Irestål

Rev. 3, 2009-02-18

Table of Contents

1 Introduction

1.1 Abbreviations

BoB	Break out Box
CAN	Controller Area Network
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
IP	Internet Protocol
LVDS	Low Voltage Differential Signaling
MPMC	Multi Port Memory Controller
RAM	Random Access Memory
RTP	Real-time Transport Protocol
SDRAM	Synchronous Dynamic RAM
TBD	To Be Determined
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

2 Detailed Design Specification

2.1 Top level system design

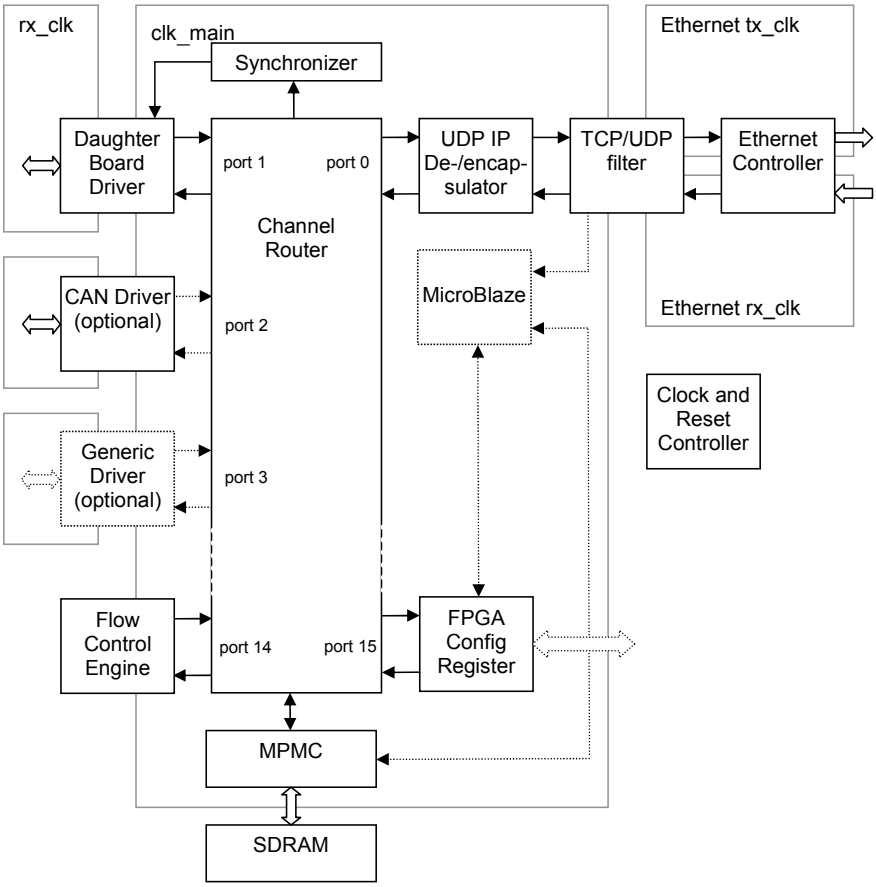


Figure2.1-1 Block level diagram of the top level of the Splitter, the block arrows represent off chip signals.

2.2 Channel router module

The data links connected to the channel router; the Ethernet link, the LVDS video link, the CAN bus etc, are defined as channels in the BoB. The channel router handles the data from these channels and routes it to the correct destination channel. The data is assumed to be packetized in RTP packets with an RTP header and payload.

The external interface of the channel router is presented below

Generic Parameters		
Name	**Type**	**Comment**
data_channel_width	Positive	Bit width of data
number_of_channels	Positive	Number of channels of channel router
synchronizer_width	Positive	Bit width of synchronizer

Ports			
Name	**Type**	**Dir**	**Comment**
reset_main_domain	std_logic	in	Synchronous reset main clock domain
clk_main	std_logic	in	Clock in main clock domain
data_in	std_logic_vector (data_channel_width - 1:0)	in	Data input for Channel router
data_transmission_req	std_logic_vector (number_of_channels - 1:0)	in	Data transmission request for each channel
data_transmission_ack	std_logic_vector (number_of_channels - 1:0)	out	Data acknowledgement for transmissions
data_out	std_logic_vector (data_channel_width - 1:0)	out	Data output for Channel router
data_reception_req	std_logic_vector (number_of_channels - 1:0)	in	Data reception request for each channel
data_reception_ack	std_logic_vector (number_of_channels - 1:0)	out	Data acknowledgement for receptions

Name	**Type**	**Dir**	**Comment**
mpmc_0_PIM1_InitDone_pin	std_logic	in	Initialization of MPMC port is done
mpmc_0_PIM1_RdFIFO_Latency_pin	std_logic_vector (1:0)	in	Latency from Pop signal of read FIFO to valid data on data bus
mpmc_0_PIM1_RdFIFO_Flush_pin	std_logic	out	Flush read FIFO
mpmc_0_PIM1_RdFIFO_Empty_pin	std_logic	in	Read FIFO is empty
mpmc_0_PIM1_WrFIFO_Flush_pin	std_logic	out	Flush write FIFO
mpmc_0_PIM1_WrFIFO_AlmostFull_pin	std_logic	in	Write FIFO almost full
mpmc_0_PIM1_WrFIFO_Empty_pin	std_logic	in	Write FIFO is empty
mpmc_0_PIM1_RdFIFO_RdWdAddr_pin	std_logic_vector (3:0)	in	Indicates the word of a cacheline transfer to which mpmc_0_PIM1_RdFIFO_Data_pin corresponds.
mpmc_0_PIM1_RdFIFO_Pop_pin	std_logic	out	Read FIFO pop signal
mpmc_0_PIM1_RdFIFO_Data_pin	std_logic_vector (31:0)	in	Read FIFO data bus
mpmc_0_PIM1_WrFIFO_Push_pin	std_logic	out	Write FIFO push signal
mpmc_0_PIM1_WrFIFO_BE_pin	std_logic_vector (3:0)	out	Write FIFO Byte enable
mpmc_0_PIM1_WrFIFO_Data_pin	std_logic_vector (31:0)	out	Write FIFO data bus
mpmc_0_PIM1_RdModWr_pin	std_logic	out	Read Modify Write for writes
mpmc_0_PIM1_Size_pin	std_logic_vector (3:0)	out	Size, indicating the type of read or write from SDRAM
mpmc_0_PIM1_RNW_pin	std_logic	out	Read or Write,
mpmc_0_PIM1_AddrAck_pin	std_logic	in	Address acknowledgement
mpmc_0_PIM1_AddrReq_pin	std_logic	out	Address request
mpmc_0_PIM1_Addr_pin	std_logic_vector (31:0)	out	Address bus of MPMC port

Figure 2.2-1 External ports of the channel router

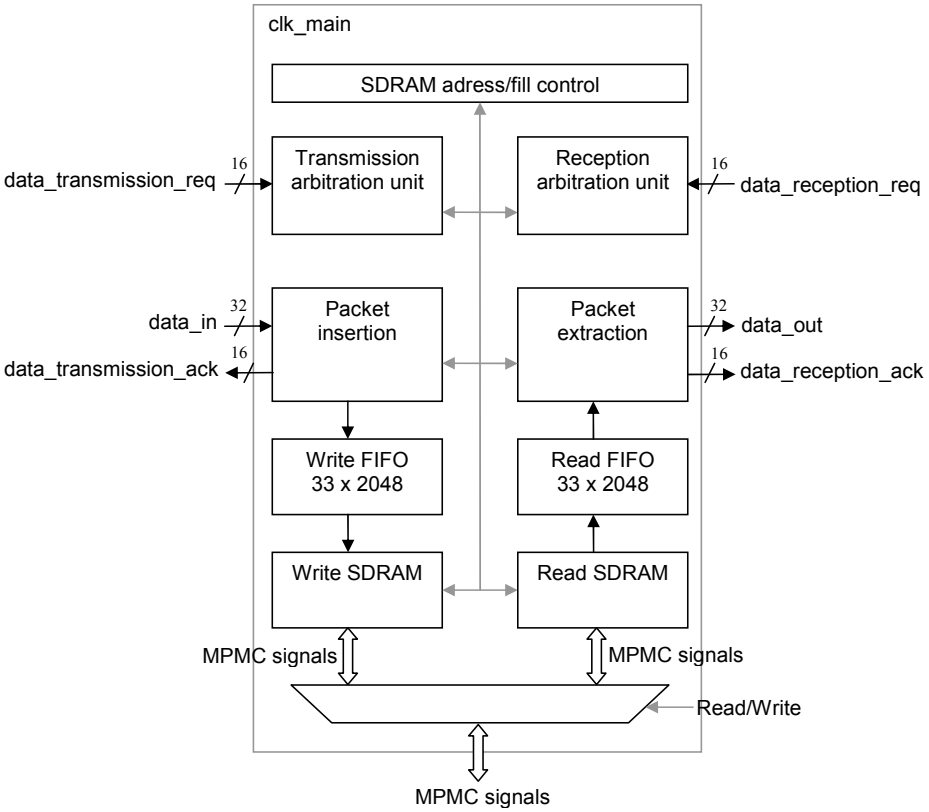


Figure 2.2-2 Block level diagram of the channel router, the grey arrows represent internal control signals

The channel router supports up to 16 channels, the routing scheme assumes that channel 0 is the Ethernet link from the BoB to a desktop computer. Data on channels 1 to 16 is routed to channel 0 and data on channel 0 is routed to the appropriate destination channel determined from the SSRC identifier in the RTP packet header. The routing is handled by the Packet insertion module which writes the packets to the Write FIFO together with a control line before every packet containing the destination channel and the SSRC identifier of the packet. The most significant bit in the FIFO line is a line valid bit. Between packets in the FIFO there has to be two invalid lines. The Packet insertion module accepts new packets as long as the Write FIFO has enough space for at least one packet and as long as the circular buffer in the SDRAM for all channels has free space for at least one packet.

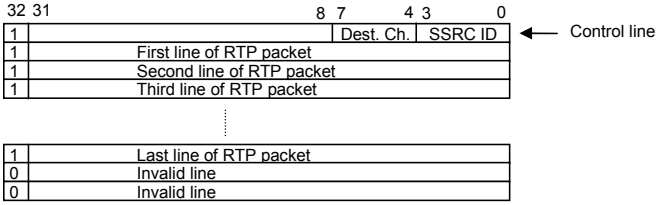


Figure 2.2-3 Data structure of the Write and Read FIFO.

Counters keep track of the number of RTP packets in the Write FIFO and the Read FIFO. As soon as there is at least one packet in the Write FIFO the Write SDRAM module requests access to the MPMC. Once access is granted the RTP packet is written to the SDRAM in the circular buffer of the

destination channel. The Read SDRAM module waits for a request from the reception arbitration unit for one of the 16 channels. Once a channel requests an RTP packet the Read SDRAM module checks if there is a packet in the SDRAM for that channel, if so access to the MPMC is requested. Once access is granted the packet is written to the Read FIFO. Once the Read FIFO holds at least one packet the Packet extraction module acknowledges the correct channel and reads out the RTP packet for that channel. For the SDRAM not to overflow the Read SDRAM module has priority over the Write SDRAM module in the contention for the MPMC. The Write SDRAM module and the Read SDRAM module always writes and reads to/from the MPMC using 32 word burst mode, this minimizes the overhead accessing the memory. Behavioural simulation of the Channel Router shows that transmission and reception of normal sized RTP packets through the Channel Router has a 50% overhead, this is due to channel arbitration overhead, MPMC contention and the memory access scheme used by the MPMC etc.

The arbitration units give priority to the lower numbered channels in the case of two or more channels requesting to either transmit or receive packets to/from the channel router in the same clock cycle. Each channel can only have one outstanding transmission request and one outstanding reception request.

The data flow of the channel router is externally request based, that is to say that the channel router is passive and acknowledges requests both to receive and to transmit RTP packets. Hence it is the task of the drivers connected to the channel router to request to transmit RTP packets to and to request to receive RTP packets from the channel router. Implementing the channel router like this makes it transparent to whether the BoB is working in recording or in playback mode.

2.3 Clock and reset controller module

The Clock and reset module handles all the clocks and reset for the splitter, some clocks related to the Ethernet Controller are generated within that module.

Ports			
Name	**Type**	**Dir**	**Comment**
clk_main_external	std_logic	in	External main clock
clk_main	std_logic	out	Internal main clock
clk_main90	std_logic	out	Internal main clock phase shifted 90 degrees
clk_gtx_external	std_logic	in	External 125 MHz clock for Ethernet transmission at 1 Gbps
clk_gtx	std_logic	out	Internal 125 MHz clock for Ethernet transmission at 1 Gbps
clk_mii_tx_external	std_logic	in	External 125 MHz clock for Ethernet transmission at 1 Gbps
clk_mii_tx	std_logic	out	Internal 125 MHz clock for Ethernet transmission at 1 Gbps
phyrxclk_external	std_logic	in	External 125 MHz clock for Ethernet transmission at 1 Gbps
phyrxclk	std_logic	out	Internal 125 MHz clock for Ethernet transmission at 1 Gbps
lvds_rx_clk	std_logic	in	External LVDS reception clock
reset_external_n	std_logic	in	External asynchronous reset
reset_main_domain	std_logic	out	Synchronous reset for main clock domain
reset_phy_n	std_logic	out	Reset for phy (active low)
reset_ethernet_controller	std_logic	out	Synchronous reset for Ethernet clock domain
reset_lvds_rx_domain	std_logic	out	Synchronous reset for LVDS FPGA driver clock domain

reset_can_domain	std_logic	out	Synchronous reset for CAN FPGA driver clock domain
reset_i2c_domain	std_logic	out	Synchronous reset for I2C FPGA driver clock domain
reset_rs232_domain	std_logic	out	Synchronous reset for RS232 FPGA driver clock domain

Figure 2.3-1 External ports of the Clock and reset controller

2.4 FPGA Config Register

The FPGA config register is connected to port 15 of the channel router. It holds the configuration file of the BoB to which a configuration is written by sending an RTP packet to the FPGA config register. To read out the configuration and status of the splitter there is a special bit assigned to make the FPGA config register transmit the configuration file in an RTP packet.

The configuration file is 32 bits wide and 10 lines depth. Presently only a small number of bits are reserved, fig 2.4-3. The lines of the configuration file will be assigned to the modules of the BoB for module configuration and status.

Generic Parameters			
Name	**Type**	**Comment**	
data_channel_width	positive	Bit width of data	
Ports			
Name	**Type**	**Dir**	**Comment**
reset_main_domain	std_logic	in	
clk_main	std_logic	in	Clock in main clock domain
data_inbound_tx	std_logic_vector (data_channel_width - 1:0)	out	Data port to transmit to the channel router
data_inbound_tx_req	std_logic	out	Request to transmit data to the channel router
data_inbound_tx_ack	std_logic	in	Acknowledgement for transmission of data to the channel router
data_inbound_rx	std_logic_vector (data_channel_width - 1:0)	in	Data port to receive from the channel router
data_inbound_rx_req	std_logic	out	Request to receive data from channel router
data_inbound_rx_ack	std_logic	in	Acknowledgement for reception of data from the channel router
gpio_led_1	std_logic	out	GPIP led 1
gpio_led_2	std_logic	out	GPIP led 2
gpio_led_3	std_logic	out	GPIP led 3
gpio_led_4	std_logic	out	GPIP led 4
gpio_led_5	std_logic	out	GPIP led 5
gpio_led_6	std_logic	out	GPIP led 6
gpio_led_7	std_logic	out	GPIP led 7
gpio_led_8	std_logic	out	GPIP led 8
reset_out	std_logic	out	Reset the splitter from the configuration

Figure 2.4-1 External ports of the FPGA config register

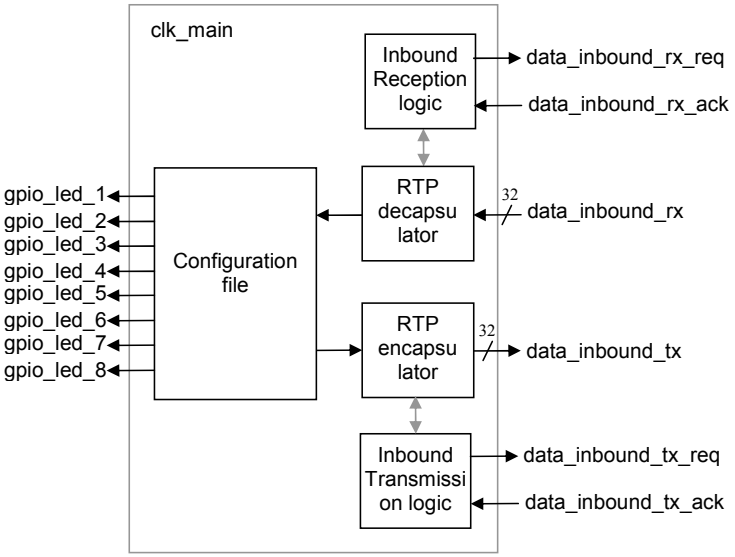


Figure 2.4-2 Block level diagram of the FPGA config register, the grey arrows represent internal control signals

Line	*Bit*	*Function*
0	0	Request for configuration file
1	24	GPIO led 1
1	25	GPIO led 2
1	26	GPIO led 3
1	27	GPIO led 4
1	28	GPIO led 5
1	29	GPIO led 6
1	30	GPIO led 7
1	31	GPIO led 8

Figure 2.4-3 Assigned bits of the configuration file

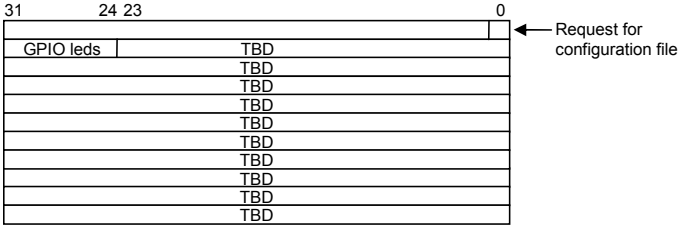


Figure 2.4-4 Data structure of the configuration file.

2.5 Ethernet controller module

The Ethernet controller is a wrapper around the IP Tri-Mode Ethernet Media Access Controller provided by Xilinx, through Core Generator. The wrapper instantiates the Ethernet Controller Configuration which configures the IP through its configuration vector, the Clock Generator which generates the clocks needed by the Ethernet IP and the GMII interface sub module creates a correct implementation of the GMII interface to the Ethernet PHY.

Ports			
Name	**Type**	**Dir**	**Comment**
reset	std_logic	in	Asynchronous reset
clk_main	std_logic	in	Clock in main clock domain
rx_clk	std_logic	out	Reception clock for the EMAC reception client
rx_data	std_logic_vector(7:0)	out	Frame data received is supplied on this port.
rx_data_valid	std_logic	out	Valid data on rx_data port
rx_good_frame	std_logic	out	Asserted at end of frame reception to indicate that the frame should be processed by the MAC client.
rx_bad_frame	std_logic	out	Asserted at end of frame reception to indicate that the frame should be discarded by the MAC client.
tx_clk	std_logic	out	Transmission clock for the EMAC transmission client
tx_data	std_logic_vector(7:0)	in	Frame data to be transmitted
tx_data_valid	std_logic	in	Valid data on tx_data port
tx_ack	std_logic	out	Handshaking signal. Asserted when the current data on tx_data port has been accepted.
tx_underrun	std_logic	in	Asserted by client to force MAC core to corrupt the current frame.
gtx_clk	std_logic	in	Input of 125 Mhz clock for transmission at 1 Gbps
gmii_txd	std_logic_vector(7:0)	out	Data for transmission to PHY
gmii_tx_en	std_logic	out	Enable signal for transmission to PHY
gmii_tx_er	std_logic	out	Error control signal to PHY
gmii_tx_clk	std_logic	out	Output of transmission clock when operating at 1 Gbps
gmii_rxd	std_logic_vector(7:0)	in	Data reception from PHY
gmii_rx_dv	std_logic	in	Data Valid control signal from PHY
gmii_rx_er	std_logic	in	Error control signal from PHY
gmii_rx_clk	std_logic	in	Input of reception clock from the PHY, 125 Mhz for 1 Gbps, 25 Mhz for 100 Mbps or 2.5 Mhz for 10 Mbps
gmii_col	std_logic	in	Control signal from PHY
gmii_crs	std_logic	in	Control signal from PHY
mii_tx_clk	std_logic	in	Input of transmission clock from the PHY for transmission at 10 Mbps or 100 Mbps

Figure 2.5-1 External ports of the Ethernet Controller

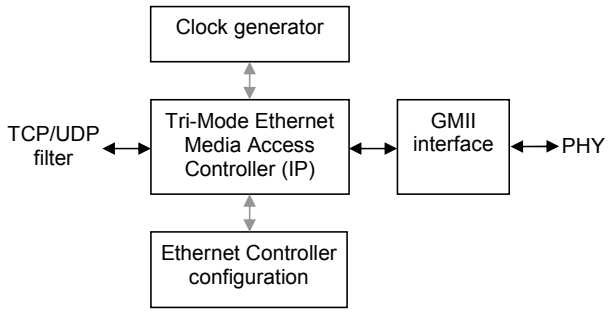


Figure 2.5-2 Block level diagram of the Ethernet controller, the grey arrows represent internal control signals and clocks

2.6 LVDS Driver NightVision module

The task of the LVDS Driver NightVision is to receive deserialized data, to time stamp and to packetize that data into RTP packets. A future revision of the module will also have the capability to perform the reverse task, transmitting data to an LVDS serializer. The LVDS driver NightVision is built to interface with LVDS deserializer National DS90C124, used by the NightVision project.

As soon as reset is released and the lock signal from the deserializer is high the deserialized data is written to the Asynchronous LVDS NV reception FIFO. The RTP encapsulator reads the Asynchronous LVDS NV reception FIFO as soon it holds at least one line of data. It time stamps the data with the synchronizer signal and performs parity checks on the data. As of now the parity check increments an error counter and isn't used for anything but Chipscope debugging. The RTP encapsulator synchronizes with a video frame of the NightVision video and starts encapsulating data into RTP packets as soon as it gets frame synchronization. Into every RTP packet one line of video is packed. All types of video lines; blank, control and image lines are encapsulated and written to the RTP Transmission FIFO. As soon as there is at least one full RTP packet in the RTP Transmission FIFO, the Inbound Transmission logic requests the channel router to send an RTP packet.

Because of the clock frequency difference between rx_clk (4 MHz) and clk_main (62.5 MHz), it is assumed that the Asynchronous LVDS NV reception FIFO never overflows.

Generic Parameters			
Name	**Type**	**Comment**	
data_channel_width	positive	Bit width of data	
synchronizer_width	positive	Bit width of synchronizer	
Ports			
Name	**Type**	**Dir**	**Comment**
reset_main_domain	std_logic	in	Synchronous reset main clock domain
reset_lvds_domain	std_logic	in	Synchronous reset lvds clock domain
clk_main	std_logic	in	Buffered clock in main clock domain
synchronizer	std_logic_vector (synchronizer_width - 1:0)	in	Synchronizer for timestamping
tx_clk	std_logic	out	Transmission clock for the LVDS serializer
data_outbound_tx	std_logic_vector (19:0)	out	Transmission data for the LVDS serializer
rx_clk	std_logic	in	Reception clock from the LVDS deserializer
data_outbound_rx_locked	std_logic	in	Reception data and clock from LVDS deserializer is locked
data_outbound_rx	std_logic_vector (19:0)	in	Reception data from the LVDS deserializer
data_inbound_tx	std_logic_vector (data_channel_width - 1:0)	out	Data port to transmit to the channel router
data_inbound_tx_req	std_logic	out	Request to transmit data to the channel router
data_inbound_tx_ack	std_logic	In	Acknowledgement for transmission of data to the channel router

data_inbound_rx	std_logic_vector (data_channel_width - 1:0)	in	Data port to receive from the channel router
data_inbound_rx_req	std_logic	out	Request to receive data from channel router
data_inbound_rx_ack	std_logic	in	Acknowledgement for reception of data from the channel router

Figure 2.6-1 External ports of the LVDS Driver NightVision

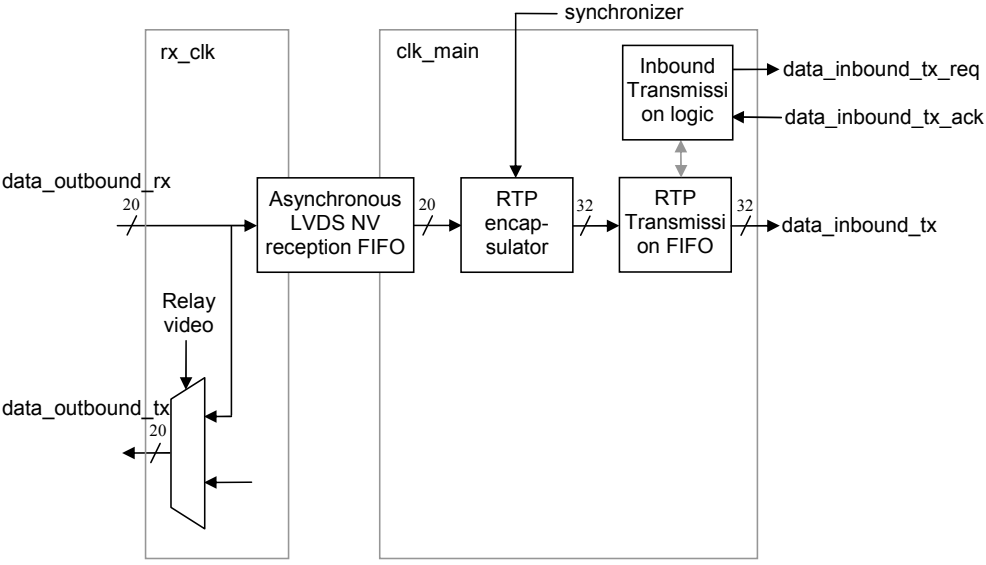


Figure 2.6-2 Block level diagram of the LVDS Driver NightVision, the grey arrows represent internal control signals

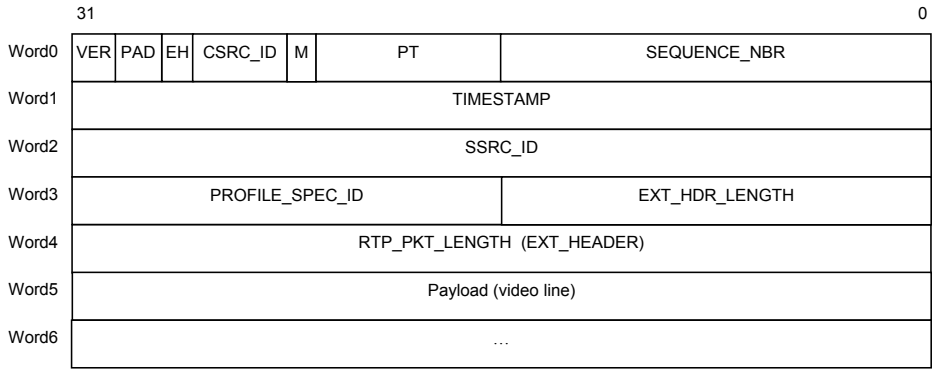


Figure 2.6-3 RTP Packet Format

The fields used are:

Name	Bits	Description
VER	2	RTP Version. Always “10”.
PAD	1	Padding. Always “0”.

EH	1	Extended Header Flag. Always “1” (use Extended Header)
CSRC_ID	4	TBD… Always “0000”.
M	1	M-Flag. Always “0”.
PT	7	Payload Type. Use “0000001” for first line in a frame and “0000000” for all other lines.
SEQUENCE_NBR	16	16-bit RTP packet Sequence Number.
TIMESTAMP	32	32-bit Time Stamp. Uses the upper 32-bits of the Synchronizer input, thus resolution is (1/clk_main)*2^10 = 13.65 uS.
SSRC_ID	32	32-bit Source identifier. Use 1 for NightVision data.
PROFILE_SPEC_ID	16	TBD… Always “0000000000000000”.
EXT_HDR_LENGTH		Extended Header Length. Always “000000000000001” (1 ext header word is used).
RTP_PKT_LENGTH	32	The total length of the RTP Packet (in number of bits). Fixed for NightVision to "00001500".

Figure 2.6-4 Explanation of the RTP protocol

2.7 Synchronizer module

The synchronizer module is essentially a counter which is used for time stamping of the incoming data; video, CAN, etc. The counter can be set to a desired value using the set_synchronizer port. If set_synchronizer has value other than 0x00000000 the counter is set to that value minus a predefined head start value. The external interface of the Synchronizer is presented below.

Generic Parameters			
Name	**Type**	**Comment**	
synchronizer_width	positive		
Ports			
Name	**Type**	**Dir**	**Comment**
reset_main_domain	std_logic	in	Synchronous reset main clock domain
clk_main	std_logic	in	Buffered clock in main clock domain
synchronizer	std_logic_vector (synchronizer_width - 1:0)	out	Synchronizer value
set_synchronizer	std_logic_vector (synchronizer_width - 1:0)	in	Set synchronizer value, asserted by the channel router

Figure 2.7-1 External ports of the Synchronizer

2.8 TCP/UDP filter

The TCP/UDP filter is connected between the UDP/IP encapsulator and the Ethernet controller. Its task is to filter the IP packets received from the Ethernet controller to the either the UDP/IP encapsulator (UDP packets) or to a Microblaze interface (TCP packets) and to act as an arbiter between UDP IP encapsulator and the Microblaze for transmitting to the Ethernet controller. At the current revision there is no Microblaze connected to the TCP/UDP filter. Therefore the arbitration mechanism isn’t implemented yet. The TCP/UDP packet filtering is implemented and functions for UDP packets.

IP packets received from the UDP IP encapsulator are written to the asynchronous UDP Inbound Packet FIFO. Once the FIFO isn’t empty the Inbound Transmission Logic requests to transmit an Ethernet packet. The frequency of clk_main is 62.5 MHz and that of tx_clk is 125 MHz (at 1 Gbps), but since the write port of the UDP Inbound Packet FIFO is four times as wide as the read port, it is assumed that as soon as there is something in the FIFO, data will be written to the FIFO faster than it is read from the FIFO.

Ethernet frames received from the Ethernet controller are written into the synchronous Small Outbound FIFO while the transport protocol is determined, TCP or UDP. Once the transport protocol is determined the Ethernet frame is read from the Small Outbound FIFO, the Ethernet header is removed and the IP packet is routed to its correct destination. The Frame Check Sequence (FCS) of an

Ethernet frame is located at the end of a frame, therefore the validity of an Ethernet frame is written to the valid FIFO, once the whole IP packet has been written to the UDP Outbound FIFO. For IP packets using TCP as the transport protocol this functionality has yet to be implemented.
Once there is at least one packet in the UDP Outbound FIFO the validity of the packet is checked. Valid packets, hence packets whose Ethernet FCS was correct, are transmitted to the UDP IP encapsulator, after the transmission request was acknowledged. Invalid packets are read out from the UDP Outbound FIFO and are discarded.

Generic Parameters			
Name	**Type**	**Comment**	
data_channel_width	positive	Width of the Channel router data bus	
Ports			
Name	**Type**	**Dir**	**Comment**
reset_main_domain	std_logic	in	Synchronous reset main clock domain
reset_ethernet_domain	std_logic	in	Synchronous reset Ethernet clock domain
clk_main	std_logic	in	Buffered clock in main clock domain
rx_clk	std_logic	in	Buffered receive Ethernet clock
tx_clk	std_logic	in	Buffered transmit Ethernet clock
clientemactxd	std_logic_vector (7:0)	out	Frame data to be transmitted
clientemactxdvld	std_logic	out	Control signal for clientemactxd port
emacclienttxack	std_logic	in	Handshaking signal. Asserted when the current data on clientemactxd has been accepted.
clientemactxunderrun	std_logic	out	Asserted by client to force MAC core to corrupt the current frame.
emacclientrxd	std_logic_vector (7:0)	in	Frame data received is supplied on this port.
emacclientrxdvld	std_logic	in	Control signal for the emacclientrxd port.
emacclientrxgoodframe	std_logic	in	Asserted at end of frame reception to indicate that the frame should be processed by the MAC client.
emacclientrxbadframe	std_logic	in	Asserted at end of frame reception to indicate that the frame should be discarded by the MAC client.
data_outbound_tx	std_logic_vector (data_channel_width - 1:0)	out	Data port to transmit to the IP/UDP encapsulator
data_outbound_tx_req	std_logic	out	Request to transmit data to the IP/UDP encapsulator
data_outbound_tx_ack	std_logic	in	Acknowledgement for transmission to the IP/UDP encapsulator
data_outbound_rx	std_logic_vector (data_channel_width - 1:0)	in	Data port to receive from the IP/UDP encapsulator
data_outbound_rx_req	std_logic	in	Request to receive data from the IP/UDP encapsulator
data_outbound_rx_ack	std_logic	out	Acknowledgement for reception from the IP/UDP encapsulator

Figure 2.8-1 External ports of the TCP/UDP filter

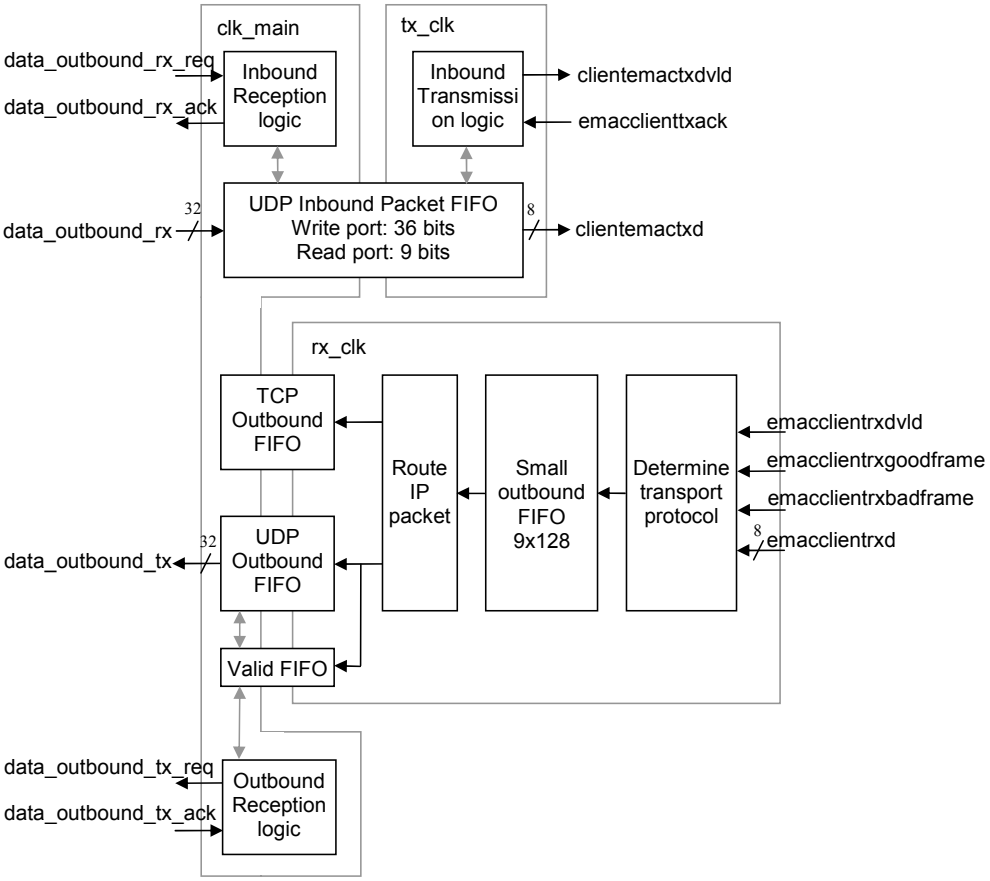


Figure 2.8-2 Block level diagram of the TCP/UDP filter, the grey arrows represent internal control signals

2.9 UDP IP encapsulator module

The UDP IP encapsulator is connected between the channel router and the TCP/UDP filter. The UDP IP encapsulator encapsulates RTP packets or decapsulates IP packets. The inbound data flow is encapsulated first with an UDP header and then with an IP header. The length of the RTP packets is used to calculate the length fields of the UDP and IP headers. For the IP header a checksum is calculated. IPv4 is the only IP version currently supported for encapsulation of the RTP packets. The inbound reception logic is constantly requesting RTP packets from the channel router. The inbound transmission logic is requesting to transmit as soon as there is at least one IP packet in the inbound packet FIFO.

For the outbound dataflow the incoming IP packets are decapsulated of their IP and UDP headers to get RTP packets. Both IP versions IPv4 and IPv6 are supported and the checksum of IPv4 headers is checked. If the IP destination address or the UDP destination port of a packet is incorrect or if the checksum of the IPv4 header is corrupt, the packet is discarded. The UDP checksum is never checked, it's assumed correct. The outbound reception logic acknowledges requests to receive packets if the outbound packet FIFO has space for one RTP packet of maximum size. The outbound transmission logic requests to transmit to the channel router when there is at least one RTP packet in the outbound packet FIFO. The external interface of the UDP IP encapsulator is presented below.

Generic Parameters			
Name	**Type**	**Comment**	
data_channel_width	positive	Width of the channel router data bus	
Ports			
Name	**Type**	**Dir**	**Comment**
reset_main_domain	std_logic	in	Reset in main clock domain
clk_main	std_logic	in	Clock in main clock domain
data_inbound_rx	std_logic_vector (31:0)	in	Data port to receive data from the UDP/TCP filter
data_inbound_rx_req	std_logic	in	Request for data port to receive data from the UDP/TCP filter
data_inbound_rx_ack	std_logic	out	Acknowledgement for data port to receive data from the UDP/TCP filter
data_inbound_tx	std_logic_vector (31:0)	out	Data port to transmit data to the UDP/TCP filter
data_inbound_tx_req	std_logic	out	Request for data port to transmit data to the UDP/TCP filter
data_inbound_tx_ack	std_logic	in	Acknowledgement for data port to transmit data to the UDP/TCP filter
data_outbound_tx	std_logic_vector (data_channel_width - 1:0)	out	Data port to transmit to the channel router
data_outbound_tx_req	std_logic	out	Request to transmit data to the channel router
data_outbound_tx_ack	std_logic	in	Acknowledgement for transmission of data to the channel router
data_outbound_rx	std_logic_vector (data_channel_width - 1:0)	in	Data port to receive from the channel router
data_outbound_rx_req	std_logic	out	Request to receive data from channel router
data_outbound_rx_ack	std_logic	in	Acknowledgement for reception of data from the channel router

Figure 2.9-1 External ports of the UDP IP encapsulator

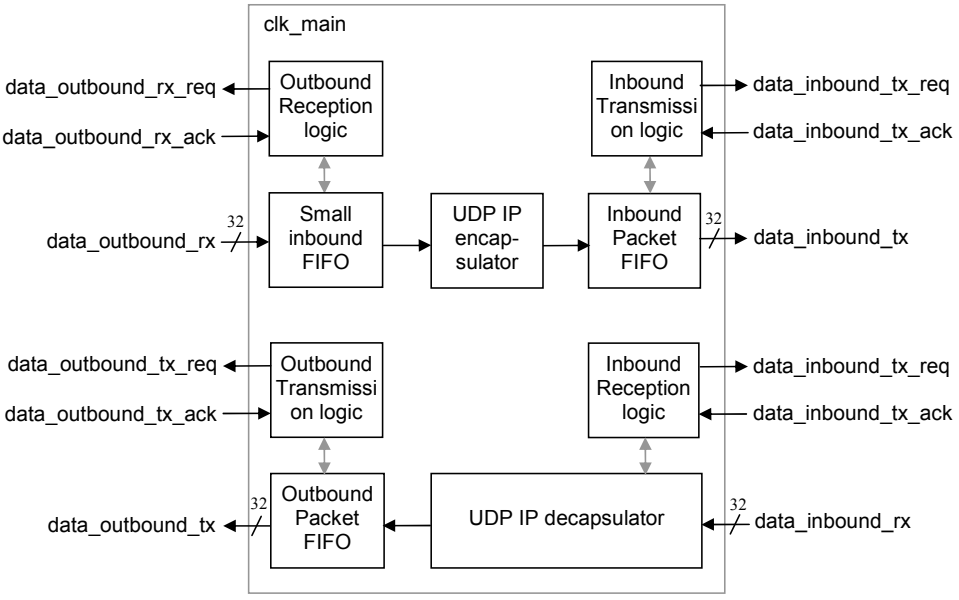


Figure 2.9-2 Block level diagram of the UDP IP encapsulator, the grey arrows represent internal control signals