UMinho | 2015

Ricardo Alves Teixeira Framework for Media Oriented Transport Systems

Universidade do Minho
Escola de Engenharia

Ricardo Alves Teixeira

Framework for Media Oriented Transport Systems

dezembro de 2015

Universidade do Minho
Escola de Engenharia

Ricardo Alves Teixeira

Framework for Media Oriented Transport Systems

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de Mestre em Engenharia Electrónica Industrial e Computadores

Trabalho efectuado sob a orientação do
Professor Doutor Adriano José da Conceição Tavares

dezembro de 2015

DECLARAÇÃO

Nome ____

Endereço electrónico: _______________________________ Telefone: _______________ / _______________

Número do Bilhete de Identidade: ______________________

Título dissertação:

____

____

____

Orientador(es):

____

Ano de conclusão: ___________

Ciclo de Estudos Integrados Conducentes ao Grau de Mestre em Arquitetura

____

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Guimarães, ___/___/______

Assinatura: ____

Media Oriented Transport System
MIEEIC Dissertation

Ricardo Teixeira

December, 2015

Contents

List of Figures

List of Tables

Abstract

The natural evolution of embedded systems resulted in a faster execution of tasks, increased possibility for including additional features, allied to lower power consumption and benefiting from ever-growing rates of integration as far as silicon is concerned. The automotive industry is not an exception with regards to the integration of technology for a vast arrays of applications in systems which vary from entertainment of infotainment to systems related to vehicle safety and stability such as driver assists. The existence of diverse independent systems in modern cars, combined with the necessity of centralizing the user interface, simplifying the operation of the system and minimizing the user's intervention, help to promote the comfort and reduce the likelihood of distractions taking place while driving. Modern communication oriented network standards, e.g. MOST or FlexRay, enable information compatibility when exchanged between systems communicating over different protocols. Moreover, the coexistence of packet, control and time-sensitive information are ensured within timing requirements, providing a reliable QoS (Quality of Service) and by making use of a single physical transmission mean. Synchronized multimedia data (e.g. synchronized video and audio transmission) are example of this kind of (time-sensitive) information.

This dissertation proposes a framework for design and development of network distributed applications in the field of automotive infotainment, compliant with the industry standards and using FPGA technology in order to ensure the system requirements satisfaction and promote IP Core re-utilization.

Resumo

A evolução natural dos sistemas embebidos traduziu-se numa maior rapidez na execução de tarefas, a possibilidade de incluir mais funcionalidades, aliado a menores consumos energéticos e beneficiando de crescentes e elevadas taxas de integração ao nível de silício. A indústria automóvel não é excepção no que diz respeito à integração de tecnologia para as mais variadas aplicações, com ou sem tolerância à falha, em sistemas que vão desde entretenimento ou infotainment a sistemas relacionados com a estabilidade e segurança do veículo, como é exemplo as *driver assists*. Existem de vários sistemas independentes nos modernos veículos automóveis. Estes, combinados com a necessidade de centralização ao nível de interface com o utilizador, tornam imperativa a simplicidade da operação. Para tal, requerem a minimizaccão da intervenção do utilizador, promovendo o conforto e diminuindo a probabilidade de desconcentração durante o exercício de condução. Os mais modernos standards de redes de comunicação como é exemplo o MOST ou o FlexRay, permitem a compatibilidade de informação trocada entre sistemas que comunicam através de distintos protocolos de comunicação. Para além disso, ainda garantem a coexistência de informação de controlo, informação do entretenimento e informação do tipo time-sensitive, onde os requisitos de temporização devem ser assegurados, mantendo uma qualidade de serviço fiàvel e fazendo uso de um único meio físico de transmissão. São exemplos deste tipo de informação, dados síncronos do tipo multimédia (e.g. streaming de àudio e vídeo de forma sincronizada). Pretende-se desenvolver uma framework para desenvolvimento de aplicações de rede distribuídas, do tipo infotainment e que beneficia a aplicação de tecnologias como FPGA, no offloading de computação para este dispositivo, como meio de garantir a satisfação dos requisitos, e promover a reutilização deste tipo de sistemas, mantendo o elevado desempenho na troca de dados e promovendo a portabilidade e a modularidade.

Chapter 1

Introduction

The growing number of entertainment components supplying and consuming all kinds of information in the automotive field increased the complexity of interconnecting and the control of such components. The final user (driver or passengers) had and still has the need to operate all the components through different, individual human machine interfaces. To build a system, each component must have the capability and ability to communicate with other components in such a way that the information can be used in real time. The infotainment Head Unit, sometimes also named system controller should govern all events in all components in such a way that the user is no longer confronted by the complexity of the system, but that requires considerable effort from the development point of view. Unless an integrative API is offered, enabling the design of an appropriate software structure around the logical representation of all components in the system controller. Still, several challenges in infotainment system design are faced, in order to comply with the flexibility and the scalability of the different market requirements over different levels of functionality and/or performance, provided by a unique but still efficient system architecture.

The FPGA technology closes such gap, as it allows for interfacing hardware in a more flexible and optimized way. The advantage of such a concept is a scalable approach providing a number of hardware devices, which can be re-configured by the FPGA code through the system software, as well as the fact that it promotes multiple IP Core integration, resulting in modularity and results in a briefer process of prototyping the system Besides promoting portability (due to its technological agnosticism), the true parallelism that is inherent to FPGA technology makes it possible to ensure timing and bandwidth requirements, as well as systemic-level synchronism.

In order to ally the benefits of using FPGA technology and optimized development effort, employment of CPU+FPGA platform makes it possible to combine the easier development of applications in software and ensuring the timing bandwidth requirements of the system, by means of computational offload to FPGA fabric, being the most time consuming tasks then implemented in RTL. In order to further improve the development effort, embedded operating systems offer several services and an abstraction layer for the development of applications.

With this in mind, the idea is to develop a communication protocol and the corresponding framework to support the design of media oriented distributed applications, following the OSI Reference Model, using FPGA technology as a mean of ensuring the coexistence of both control and time-sensitive data as well as meeting the performance demands and functional requirements of media oriented applications, combined with a CPU where software implemented parts of the communication protocol and framework are deployed, in order to ease the design of the whole system.

1.1 Document organization

The remainder of this document contains, firstly, the State of the Art, where the essential concepts and definitions to understand the actual work, are explained. Afterwards, comes the description of the Hardware Platforms used in this project and the related choices regarding this topic. The stack of the network protocol is examined and the design of the different parts of the communication protocol are presented. Following that, the Study Case evaluates the implemented parts of the project under test and documents the associated results. Moreover, the Conclusion chapter comprises with considerations about the obtained results and the lessons learned throughout the process of developing this dissertation. Finally, improvements that can be done to the showcased system are proposed.

Chapter 2

State of the Art

In the past, automotive manufacturers connected electronic devices in vehicles using point-to-point wiring systems.

Manufacturers began using more and more electronics in vehicles, which resulted in bulky wire harnesses that were heavy and expensive.

They then replaced dedicated wiring with in-vehicle networks, which reduced wiring cost, complexity, and weight. Several in-vehicle networks standards have been emerging, such as CAN (Controller Area Network), FlexRay and MOST (Media Oriented Systems Transport).

2.1 Controller Area Network

The CAN protocol was quickly adopted by the automotive industry, as a result of its simplicity, given that every Electronic Control Unit only requires a single CAN port to be connected to the network, instead of as many digital ports (and point-to-point connections) as the number of devices to be interconnected. Therefore, cost and weight are kept to a minimum. Moreover, every appended devices to the network are able to check the messages, as they are broadcasted within the whole network. Thus, the addition of new nodes to the CAN network has no impact on the way the network works and the existing nodes exchange information. The support for defining priorities to more important messages allows uninterrupted transmission of higher priority messages, contributing to a greater level of determinism offered by this network. Besides, not only does this protocol perform message integrity check, but also orders transmission nodes that are transmitting too many corrupted messages to suspend its communications and disconnect from the network, so that the correct functioning of the network is assured, in case a node is malfunctioning.

2.2 FlexRay

For automotive industry to continuously improve safety and performance, reduce environmental impact and enhance comfort, the speed, reliability and size of data bandwidth within car's electronic control units (ECU) is scaling proportionally. Modern control strategy and safety systems, combining multiple sensors, actuators and ECUs, begin to require synchronization and performance beyond the existing standards can provide. Coupled with growing bandwidth requirements with today's advanced vehicles there is a demand for a next-generation embedded network. The FlexRay protocol is a time-triggered protocol that provides options for deterministic data that arrives in a predictable time frame (down to the microsecond) as well as a dynamic event-driven data to handle a large variety of frames. FlexRay accomplishes this hybrid of core static frames and dynamic frames with a pre-set communication cycle that provides a pre-defined communication cycle space for static and dynamic data. In order to guarantee a greater bandwidth and prevent data collision and consequently data frame corruption, FlexRay manages multiple nodes with a TDMA (Time Division Multiple Access) scheme. Every FlexRay node is synchronized to the same clock, and each node waits for its turn to write on the bus. Because the timing is consistent with a TDMA scheme, FlexRay is able to guarantee the consistency of data delivery to nodes on the network, which provides many advantages for systems that depend on up-to-date data between nodes.

2.3 MOST - Media Oriented Systems Transport

The infotainment sector in cars has developed rapidly within the last few years. Where an AM/FM tuner was initially sufficient, many vehicles are now equipped with high-grade sound and navigation systems. This also increases the number of devices involved - radio, cell phone, CD changer, navigation system, voice operation and an additional multi-channel amplifier supplement the primary systems.

These devices must interact in concert with each other and can only be controlled via a central operating surface to ensure drivers are not unnecessarily distracted. The bandwidth of the CAN-Bus, which can only move control signals, but no audio and video signals, is no longer sufficient for these demands. MOST, Media Oriented Systems Transport, a communication system with a new, flexible architecture, used by many different manufacturers, was developed to meet these demands. It can transmit audio signals synchronously, as in telephones, and is the most widely used multimedia system in cars today. The MOST system not only comprises all layers of the Open System Interconnect (OSI) Reference Model for communication and computer network design, but also standardizes the interfaces with the applications (e.g., an AM/FM tuner), which are defined as function blocks, enabling the developer to design multi-media systems on a relatively high abstraction level.

2.3.1 CAN vs FlexRay vs MOST for multimedia oriented applications

Even though protocols such as CAN and FlexRay present high-reliability in terms of data transmission as well as determinism, other requirements must be met in order to support modern distributed infotainment applications. The Table 2.1 puts the CAN, FlexRay and MOST networks into comparison, in terms of peak bandwidth, coupled with a few examples about typical application in which each protocol is employed.

Table 2.1: In-vehicle network comparison

	CAN	**FlexRay**	**MOST150**
Maximum Bandwidth	1 Mbit/s	10 Mbit/s	150 Mbit/s
Typical Applications	Powertrain (Engine, Transmission, ABS)	High-Performance Powertrain, Safety (Drive-by-wire, active suspension, adaptive cruise control)	Distributed infotainment systems

It can be concluded that the MOST standard is the most suitable protocol for multimedia transmission not only due to the higher peak bandwidth, but also because it grants coexistence between time-sensitive (e.g., synchronized video and audio streaming) and control-information, due to the data frame structure. Besides, networks with distinct protocols, such as CAN, FlexRay or Ethernet, can be integrated by means of gateways in a MOST network, consequently providing support for networks inter-communication.

2.4 Transmission Paths

In-vehicle network signals are subject to Radio Frequency and/or Electromagnetic interferences, which are known to compromise its integrity. This section will focus on solutions that overcome such issues, such as Optical Fiber, Coaxial and Twisted-Pair cable.

2.4.1 Optical Fiber

Optical fibers are utilized more and more frequently for applications with a high data rate due to increasing EMC requirements (Electromagnetic Compatibility). Transmission lines with optical fibers do not cause any interference radiation and are insensitive to electromagnetic interference irradiation (Electromagnetic Interference, EMI). In contrast to shielded electric data lines, optical fibers are, moreover, lighter and more flexible and can thus be handled more easily. Furthermore, optical fiber provides far greater bandwidth than copper and has standardized performance up to 10Gbps, being the modulation frequency of the transceivers the only limiting factor. The extra cost of the optical transceivers and the higher costs of equipment for testing/diagnosing optical fiber based circuits is however a negative aspect, comparing to the electrical transmission paths.

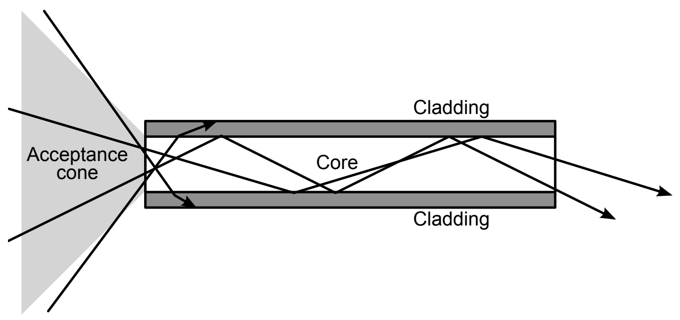


Figure 2.1: Propagation of light in the optical fiber core

2.4.2 Coaxial cable

Coaxial cable conducts electrical signal using an inner conductor (usually a solid copper, stranded copper or copper plated steel wire) surrounded by an insulating layer and all enclosed by a shield, typically one to four layers of woven metallic braid and metallic tape. The cable is protected by an outer insulating jacket. Normally, the shield is kept at ground potential and a voltage is applied to the center conductor to carry electrical signals. The advantage of coaxial design is that electric and magnetic fields are confined to the dielectric with little leakage outside the shield. Conversely, electric and magnetic fields outside the cable are largely kept from causing interference to signals inside the cable. Larger diameter cables and cables with multiple shields have less leakage. This property makes coaxial cable a good choice for carrying weak signals that cannot tolerate interference from the environment or for higher electrical signals that must not be allowed to radiate or couple into adjacent structures or circuits. Coaxial cable is used as a transmission line for radio frequency signals. Its applications include feedlines connecting radio transmitters and receivers with their antennas, computer network (Internet) connections, and distributing cable television signals.

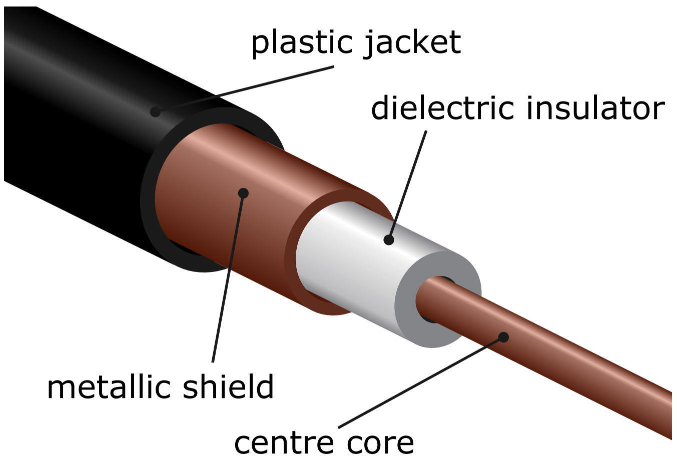


Figure 2.2: Coaxial cable cutaway

2.4.3 Twisted-Pair cable

Low-voltage differential signaling, or LVDS, also known as TIA/EIA-644, is a technical standard that specifies electrical characteristics of a differential, serial communications protocol. LVDS operates at low power and can run at very high speeds using inexpensive twisted-pair copper cables. LVDS was introduced in 1994, and has become popular in products such as LCD-TVs, automotive infotainment systems, industrial cameras and machine vision, notebook and tablet computers, and communications systems. The typical applications are high-speed video, graphics, video camera data transfers, and general purpose computer buses. LVDS is a differential signaling system, meaning that it transmits information as the difference between the voltages on a pair of wires; the two wire voltages are compared at the receiver. As long as there is tight electric- and magnetic-field coupling between the two wires, LVDS reduces the generation of electromagnetic noise. This noise reduction is due to the equal and opposite current flow in the two wires creating equal and opposite electromagnetic fields that tend to cancel each other. In addition, the tightly coupled transmission wires will reduce susceptibility to electromagnetic noise interference because the noise will equally affect each wire and appear as a common-mode noise. The LVDS receiver is unaffected by common mode noise because it senses the differential voltage, which is not affected by common mode voltage changes.

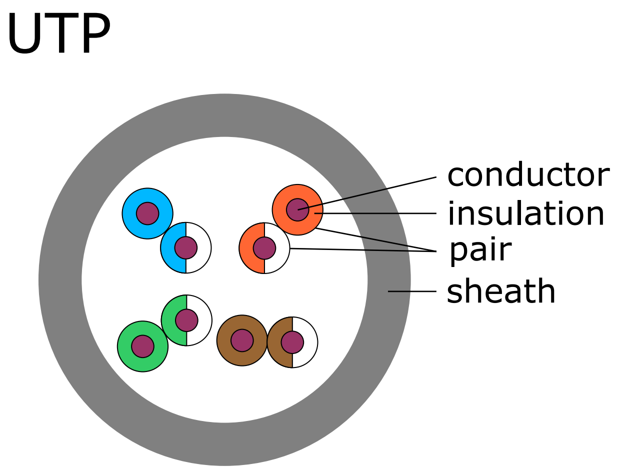


Figure 2.3: Unshielded Twisted Pair cable cutaway

2.4.4 Conclusions

Taking into account the increasing Electromagnetic Interference which in-vehicular networks are subject to and the increasing bandwidth requirements due to the growing integration of devices, specially as far as multimedia systems are concerned, optical fiber is the most suitable solution from a performance vs cost

Table 2.2: Criteria for comparison between transmission paths

Criteria	Description
Cost	Cost of the transmission path and corresponding transceivers
Reliability	Durability and reliable data transmission under environments with high EMI and RFI
Weight	Weight of the wiring loom
Ease of assembly/handling	Robustness of the cabling and capability to be mounted in tight spaces

Table 2.3: Decision table - transmission paths matchup

Criteria	Optical Fiber	Coaxial Cable	UTP Cable
Cost	4	4	5
Reliability	5	4	3
Weight	5	3	4
Ease of assembly/handling	4	4	5
Total Score	18	15	17

standpoint, not to mention the material's lighter weight, as described in the decision table, 2.2 and 2.3

However, with regards with this dissertation, as the project is developed in a laboratory where interferences are less likely to occur and due to time and cost constraints, the employment of the LVDS technique with Unshielded Twisted Pair Cables provides a reliable data transfer under these circumstances.

2.5 Node positioning and interconnection

The Physical Layer defines the structure of interconnected devices, that is, the positioning of the nodes in the network, the cabling layout and the employed type of cabling. This chapter is dedicated to the exploration of an array of network physical topologies, followed by the selection of the most suitable topology for the system that is to be developed, according to the trade-offs each method offers.

Moreover, considerations will be drawn about transmission paths which ensure fast and reliable data exchange under environments where Radio-Frequency and Electromagnetic interferences are prone to occur, as is the case with vehicles. Afterwards, comes the choice of the most convenient solution that meets the system's requirements and constraints. In this section, the main aspects of well-known network topologies are meant to be analysed, having special attention to how far scalability goes while still ensuring the timing and bandwidth requirements as well as implications of the resulting wiring loom and if it is appropriate to reliably transport time-sensitive data. The topologies to be studied are:

- Bus Network
- Star Network
- Ring Network
- Mesh Network

2.5.1 Bus Network

In a Bus Network, the devices are attached to a shared communication line, also known as backbone, which has signal terminators at its ends in order to prevent the data to be reflected back, thus causing collisions.

A host exists for network management purposes, which similarly to all the interconnected nodes, receives all the network traffic and has the same transmission priority.

The definition of a node as a Bus Master or the employment of a Media Access Control technology, such as Carrier Sense Multiple Access (CSMA) handle data collisions, in case multiple nodes simultaneously transmit data.

Unless the shared communication line fails, the normal operation of the devices is not compromised upon failure of a node in the network.

The figure 2.4 illustrates the connection of the nodes to the backbone.

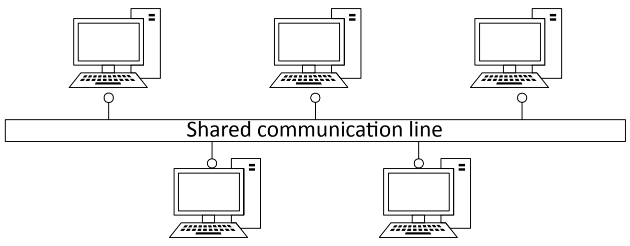


Figure 2.4: Node interconnection in a Bus topology

Even though the Bus Network offers the following advantages:

- The process of installation of a new node in the network is simple
- Requires less cable length than a Star Network Topology, thus resulting in lower costs and less weight of the wiring loom

There are also some limitations to take into consideration:

- A break in the main cable, which is a difficult problem to identify, may lead to a failure in the operation of the whole system
- The available bandwidth is reduced as more devices are attached to the network, which is not ideal for large networks
- Data collisions may be more frequent as more nodes are interconnected by means of this topology, consequently resulting in determinism not to be verified and possible failure to meet the timing requirements, which is not acceptable as the system involves the transport of time-sensitive data

2.5.2 Star Network

A Star Network is composed by a central node to which all other nodes are connected. The Central Node is responsible for rebroadcasting an incoming message to all the nodes in the network. If the Central Node is passive, it rebroadcasts the message to all the existing nodes in the network, including the echo to the node the incoming message came from. Therefore the node from which the message originated from, must have mechanisms that identify and handle the echoes. If however, the Central Node is active, it can prevent echo messages to be sent to the corresponding peripheral node.

The figure 2.5 illustrates the interconnection of the nodes by means of the Central Node.

The following advantages from this topology are outlined in the following topics:

- The normal operation of the nodes may not be affected if one leaf node fails or unless the central node functions abnormally
- The bandwidth and the delay in the transmission to any leaf node is unchanged as more nodes are attached to the network, as long as the capacity of the Central Node is not exceeded

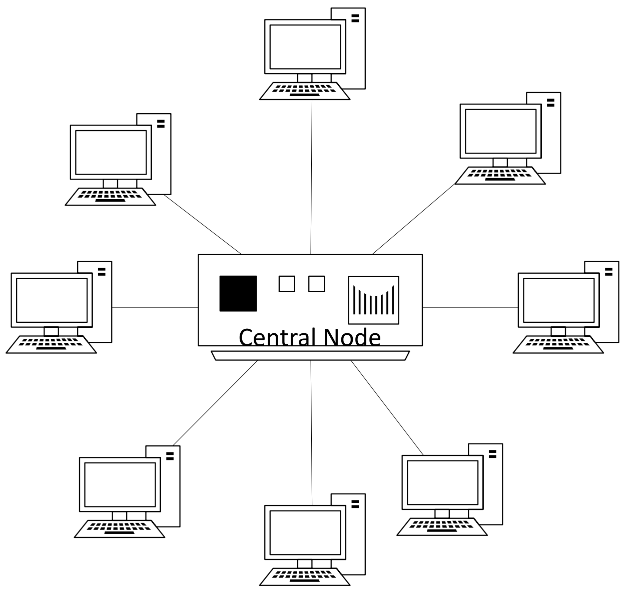


Figure 2.5: Node interconnection in a Star topology

- Centralization is beneficial as it enables easier traffic analysis and the detection of eventual malfunctions

However, there are drawbacks that should be taken into consideration:

- The number of nodes attached to the network is limited by the capacity of the central node, which disfavours scalability, not to mention the increasing costs as more capable central nodes are required
- More cable length is required, compared to Bus or Ring networks, thus resulting in greater expenses in terms of costs and weight (of the wiring loom)
- The communication relies on the central node, which is a single point of failure that can compromise the whole system

2.5.3 Ring Network

With regards to the Ring Network topology, each node is connected to two other nodes by means of a single pathway, resulting in a circular network structure through which data travels unidirectionally. All nodes have the same transmission priority, provided that the data has to pass through each one, as it travels around the ring. The figure 2.6 illustrates the connection of the nodes in a ring topology.

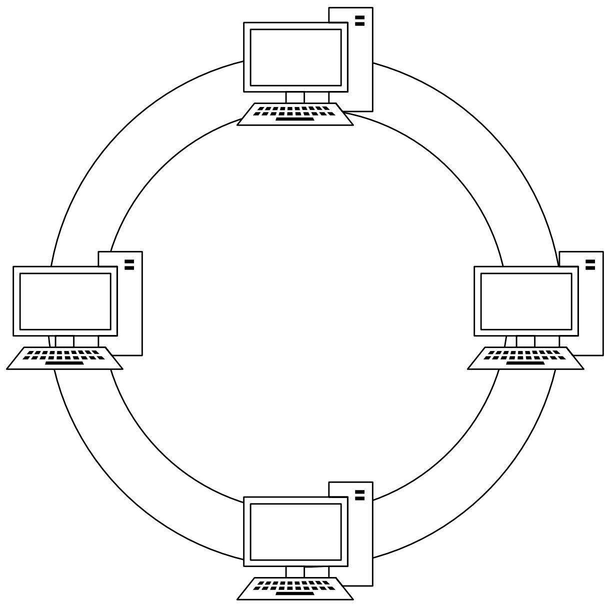


Figure 2.6: Node interconnection in a Ring topology

Advantages are explained in the following topics:

- The process of installation of a new node and reconfiguration of a new node is easy, given that it only requires moving one connection between two nodes.
- Under heavy network load, the performance degradation is fewer than on the Bus Network topology.
- Every node has access to the token and opportunity to transmit, coupled with the impossibility for data collisions to occur
- The system's integrity does not depend on solely one node (as is the case with the Star Network topology, which requires a central node)
- Faults in the system, that is, ring breaks, are easy to spot provided that a single pathway interconnects two nodes
- Adding an extra node requires only one extra cable, resulting in little extra weight added

As disadvantages:

- The procedures to start-up and shut down are more complex than with the Start network topology

- The communication delay increases with the number of nodes appended to the network
- One malfunctioning node may interrupt the communications, unless a bypass is integrated in each device, enabling data to proceed to the next nodes

2.5.4 Mesh Network

In this type of topology, a point-to-point connection exists between each pair of nodes in the system, except if it is about a Partial Mesh. If a failure occurs in any of the transmission paths, self healing mechanisms reconfigure the network, establishing alternative paths to ensure data reaches the destination. The figure 2.7 illustrates the connection of the nodes in a mesh topology. A strong point of this

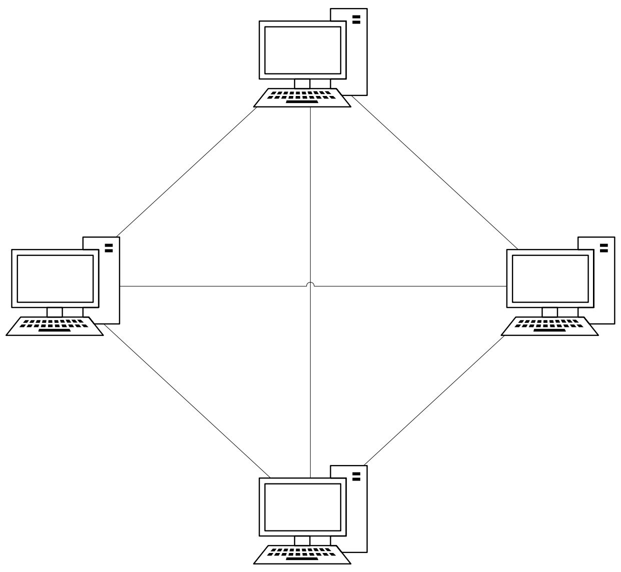


Figure 2.7: Node interconnection in a Mesh topology

network topology is the reliability, because redundant paths may exist. However, the amount of cabling required, and consequently the cost and weight are way higher compared to the aforementioned network topologies.

2.5.5 Conclusions

It is now known any of the aforementioned network topologies have at least one weak spot that can render a system unable to be fail safe. Even though the Mesh Network topology, with its self healing mechanisms, presents less risks of leading a system to operate abnormally, not only the choice of this topology would have an huge impact in terms of costs, but also on weight, as a result of the amount of cabling required increases exponentially as more devices are integrated in the network.

The Bus Network would be a great alternative, as it covers some of the disadvantages of the Mesh Network topology, however, as more devices are attached to the network, the bandwidth is affected negatively, thus limiting the dimension of a system as far as scalability is concerned. We are left with two contenders: Ring and Star topologies. Although the Star network topology provides less node-to-node delay than the Ring Network, the use of a Central Node is a limiting characteristic from a system scalability standpoint.

However, the delay each node provokes in the Ring Network topology may be drastically reduced by means of CPU offloading/Hardware Acceleration of parts of the system's application using FPGA technology, consequently allowing the timing requirements to be ensured. Besides, although the Ring network topology requires a more complex process in order to initialize the system, this is not a critical aspect, as it does not affect the post system setup phase. On top of that, the amount of cabling required is fewer, which comes as a point in favor of the Ring topology, provided that weight saving is an important aspect in the automotive industry, not to mention budget control.

To conclude, the Ring Network topology offers the best trade-off and shall be the chosen one.

The table 2.4 contains the criteria that were taken into consideration for choosing the most adequate topology for the system to be worked on.

Table 2.4: Criteria for interconnection topology comparison

Criteria	**Description**
Performance Degradation	Reduction of available bandwidth as more nodes are attached to the network
Wiring Loom	Required wiring loom to interconnect nodes
Node Complexity	Development effort to integrate the features to interact with the network
Reliability	How the system's integrity is compromised, in an event of a failure in a node or transmission path

Table 2.5: Decision table for node interconnection topology

Criteria	**Star Topology**	**Bus Topology**	**Ring Topology**	**Mesh Topology**
Performance Degradation	5	4	5	3
Wiring Loom	3	5	4	2
Node Complexity	3	4	5	3
Reliability	5	3	4	5
Score	16	16	18	13

Afterwards, the table 2.5 consists of a decision matrix, where a score is assigned to each topology, with regards to each criteria, according to what was studied in the previous subsections. The score varies between 0 and 5, meaning the punctuation of 5, the fact that the topology performs better than all other in a given aspect. If more than one topologies perform similarly, both get the same points, otherwise a score the immediately inferior value is assigned.

2.6 Electronic Design Automation tools

Digital circuit design has evolved rapidly over the last 25 years. The earliest digital circuits were designed with vacuum tubes and transistors. Integrated circuits were then invented where logic gates were placed on a single chip. The first integrated circuit (IC) chips were SSI (Small Scale Integration) chips where the gate count was very small. As technologies became sophisticated, designers were able to place circuits with hundreds of gates on a chip. These chips were called MSI (Medium Scale Integration) chips. With the advent of LSI (Large Scale Integration), designers could put thousands of gates on a single chip. At this point, design processes started getting very complicated, and designers felt the need to automate these processes. Electronic Design Automation (EDA) techniques began to evolve. Chip designers began to use circuit and logic simulation techniques to verify the functionality of building blocks of the order of about 100 transistors. The circuits were still tested on the breadboard, and the layout was done on paper or by hand on a graphic computer terminal.

With the appearance of VLSI (Very Large Scale Integration) technology, designers could design single chips with more than 100,000 transistors. As a result of the complexity of these circuits, it was not possible to verify these circuits on a breadboard. Computer- aided techniques became critical for verification and design of VLSI digital circuits. Computer programs to do automatic placement and routing of circuit layouts also became popular. The designers were now building gate-level digital circuits manually on graphic terminals. They would build small building blocks and then derive higher-level blocks from them. This process would continue until they had built the top-level block. Logic simulators came into existence to verify the functionality of these circuits before they were fabricated on chip.

As designs got larger and more complex, logic simulation assumed an important role in the design process. Designers could iron out functional bugs in the architecture before the chip was designed further.

2.7 Hardware Description Languages

For a long time, programming languages such as FORTRAN, Pascal, and C were being used to describe computer programs that were sequential in nature. Similarly, in the digital design field, designers felt the need for a standard language to describe digital circuits. Thus, Hardware Description Languages (HDLs) came into existence, allowing the designers to model the concurrency of processes found in hardware elements. Hardware description languages such as Verilog HDL and VHDL became widely used, being Verilog HDL originated in 1983 at Gateway Design Automation and VHDL developed under contract from DARPA.

Even though HDLs were popular for logic verification, designers had to manually translate the HDL-based design into a schematic circuit with interconnections between gates. The release of logic synthesis tools in the late 1980s changed the design methodology radically. Digital circuits could be described at a register transfer level (RTL) by use of an HDL. Thus, the designer had to specify how the data flows between registers and how the design processes the data. The details of gates and their interconnections to implement the circuit were automatically extracted by logic synthesis tools from the RTL description.

Thus, logic synthesis pushed the HDLs into the forefront of digital design. Designers no longer had to manually place gates to build digital circuits. They could describe complex circuits at an abstract level in terms of functionality and data flow by designing those circuits in HDLs. Logic synthesis tools would implement the specified functionality in terms of gates and gate interconnections.

HDLs also began to be used for system-level design. HDLs were used for simulation of system boards, interconnect buses, FPGAs (Field Programmable Gate Arrays), and PALs (Programmable Array Logic). A common approach is to design each IC chip, using an HDL, and then verify system functionality via simulation.

Today, Verilog HDL is an accepted IEEE standard. In 1995, the original standard IEEE 1364-1995 was approved. IEEE 1364-2001 is the latest Verilog HDL standard that made significant improvements to the original standard.

HDLs have many advantages compared to traditional schematic-based design.

- Designs can be described at a very abstract level by use of HDLs. Designers can write their RTL description without choosing a specific fabrication technology. Logic synthesis tools can automatically convert the design to any fabrication technology. If a new technology emerges, designers do not need to redesign their circuit. They simply input the RTL description to the logic synthesis tool and create a new gate-level netlist, using the new fabrication technology. The logic synthesis tool will optimize the circuit in area and timing for the new technology.

- By describing designs in HDLs, functional verification of the design can be done early in the design cycle. Since designers work at the RTL level, they can optimize and modify the RTL description until it meets the desired functionality. Most design bugs are eliminated at this point. This cuts down design cycle time significantly because the probability of hitting a functional bug at a later time in the gate-level netlist or physical layout is minimized.

- Designing with HDLs is analogous to computer programming. This also provides a concise representation of the design, compared to gate-level schematics. Gate-level schematics are almost incomprehensible for very complex designs.

With rapidly increasing complexities of digital circuits and increasingly sophisticated EDA tools, HDLs are now the dominant method for large digital designs.

The speed and complexity of digital circuits have increased rapidly. Designers have responded by designing at higher levels of abstraction. Designers have to think only in terms of functionality. EDA tools take care of the implementation details. With designer assistance, EDA tools have become sophisticated enough to achieve a close-to-optimum implementation.

Behavioral synthesis allowed engineers to design directly in terms of algorithms and the behavior of the circuit, and then use EDA tools to do the translation and optimization in each phase of the design. However, behavioral synthesis did not gain widespread acceptance. Today, RTL design continues to be very popular. Verilog HDL is also being constantly enhanced to meet the needs of new verification methodologies.

New verification languages have also gained rapid acceptance. These languages combine the parallelism and hardware constructs from HDLs with the object oriented nature of C++. These languages also provide support for automatic stimulus creation, checking, and coverage. However, these languages do not replace Verilog HDL. They simply boost the productivity of the verification process. Verilog HDL is still needed to describe the design.

For very high-speed and timing-critical circuits like microprocessors, the gate-level netlist provided by logic synthesis tools is not optimal. In such cases, designers often mix gate-level description directly into the RTL description to achieve optimum results. This practice is opposite to the high-level design paradigm, yet it is frequently used for high- speed designs because designers need to meet strict timing requirements out of circuits, and EDA tools sometimes prove to be insufficient to achieve the desired results, even though these are always evolving.

2.8 Standard integration and reuse of processor, system and peripheral cores

System-on-a-chip (SoC) and ASIC silicon densities now support system-level implementations. The buses to link processors, memory, peripherals, and special functions are necessary. On-chip multilevel bus systems have emerged to meet these needs. One is the CoreConnect on-chip bus system from IBM Microelectronics. Originally designed to support PowerPC cores for IBM ASICs, this bus system now handles other processors and can be licensed for deployment.

CoreConnect is an on-chip silicon bus bundle for ASIC or FPGA designs. It consists of a three-level system: the processor local bus (PLB), the on-chip peripheral bus (OPB), and the device control register (DCR) bus. The first bus, the PLB, connects the processor to high-performance peripherals, such as memory, DMA controllers, and fast devices. Bridged to the PLB, the OPB supports the slower-speed peripherals. The third bus, the DCR, is a separate control bus that links to all of the devices, controllers, and bridges. It provides a separate path to set and monitor the individual control registers.

2.8.1 Processor Local Bus

The PLB is the main on-chip system bus. It links the processor with on-chip memory, memory controllers, and other high-speed peripherals, including DMA controllers. It's a synchronous, multimaster, arbitrated bus. For higher throughput, it supports concurrent Reads and Writes, even for the same master. As a result, each master has a single 32-bit address bus plus separate Read and Write buses, which can be implemented as 32, 64, and 128 bits wide. The design even allows 256-bit-wide data buses.

Also, the PLB supports pipelined addressing, enabling masters to re-quest bus access while the current transaction(s) are executing. It implements four priority levels of bus access. Plus, masters can lock the bus for atomic operations, keeping out any other master until they have completed the locked transaction(s). Transaction address cycles have three phases; request, transfer, and acknowledge. Data cycles have two phases; transfer and acknowledge. The acknowledge phase can occur in the trailing portion of the same clock for the transfer phase of a single-clock data transfer.

The centerpiece of the PLB bus is the PLB macro, which includes the arbiter and bus multiplexer switch. Each bus master connects its arbitration signals, bus control signals, address bus, and Read and Write data buses to the arbiter, which functions like a giant multiplexer. Concurrently, the PLB can support a Read and a Write through its multiplexer. It only presents one master's address bus at a time, though, starting one transaction at a time. The arbiter supports up to 16 masters, with no logical restrictions on the number of slave devices.

The PLB implements an interesting variation of a split-transaction and a forward split-transaction. The traditional split-transaction separates the master request from the slave response. This is invaluable for making Reads effective, giving the device time to get its data ready to transfer. In contrast, CoreConnect's PLB permits masters to make early requests to the slave before the bus is allocated to them. This early warning enables the addressed device to set up before the bus is allocated to the master's request.

The PLB supports both fixed- and variable-length bursts. The fixed-length bursts are defined by a 3-bit field in the transaction request. Variable-length transactions are controlled by the master bus control signals.

Address pipelining also enables the bus to start a new transaction before the current transaction completes. This means that a Read or a Write can be started during a Write or a Read transaction in a master. The new transaction might be for the existing master or another master.

For larger systems with multiple CoreConnect PLB buses, IBM has provided two crossbar switches. These switch cores let designers link multiple on-chip PLBs through a central clearinghouse switch.

2.8.2 On-Chip Peripheral Bus

Designed to support slower peripherals, the OPB is implemented as a straightforward multimaster, arbitrated bus. It's a synchronous bus with a common clock, but its devices can run with slower clocks, as long as all of the clocks' rising edges are in sync with the rising edge of the main clock. This bus uses a distributed multiplexer implementation.

The OPB implements a 32-bit address bus and a separate 32-bit data bus. Transaction widths can be full-word, half-word, or byte-size. The bus supports 8-, 16-, and 32-bit wide device interfaces (aligned on the left-most byte). Data transactions can take a single cycle (for matched clocks), and burst operations are supported.

The bus masters compete for the bus via the arbiter. Each master connects directly to the arbiter via its Mn_request (bus request), Mn_busLock (bus lock), and OPB_MnGrant (bus grant) signals. A master may request to lock the bus, holding it until the bus is released.

OPB consists of two multiplexer-based buses: the OPB address bus (OPB_ABus) and the OPB data bus (OPB_Dbus). The address bus gates the selected master's address bus to the devices, and through "AND" and "OR" gating, the data bus picks up the selected master's data bus (Mn_Dbus) for a Write or the addressed slave's data bus (SIO_DBus) for a Read.

Each clocked data transfer includes a transfer-acknowledge signal from the slave device indicating completion of the data transfer. If the slave device can't complete the data transfer or accept the transfer request, it can assert an error-acknowledge signal, Sin_errAck ("Ored" to create OPB_errAck, the OPB error acknowledge). The OPB supports built-in DMA peripheral controllers with a special DMA channel. The DMA channel arbitrates for control of the PLB like a master.

2.8.3 Device Control Register Bus

The DCR bus provides an alternative path to the system for setting the individual device control registers. With it, the host CPU can set up the device-control-register sets without loading down the main PLB. This bus has a single master, the CPU interface, which can Read or Write to the individual device control registers. The bus employs a ring implementation to connect the CPU interface to the devices, which are addressed via a 10-bit address bus. A separate 32-bit data bus transfers register data. (On-chip silicon is cheap.)

This is a synchronous bus. The individual devices can run at a slower clock rate than the bus clock, but the rising edge of the device clocks must correspond to that of the faster DCR bus clock. The CPU, the master, connects to each slave device with its address bus, data bus, and control signals (dcrWrite, dcrRead). The output bus of the slave devices connects to the CPU interface via a multiplexer "OR" function. The addressed device simultaneously signals receipt of a Read transaction and the end of a Write transaction. Bursts aren't supported by this bus. Each transaction takes a minimum of three cycles (more with slower device clocks).

2.9 Interrupts

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.[1] There are two types of interrupts: hardware interrupts and software interrupts.

Hardware interrupts are used by devices to communicate that they require attention from the operating system.[2] Internally, hardware interrupts are implemented using electronic alerting signals that are sent to the processor from an external device, which is either a part of the computer itself, such as a disk controller, or an external peripheral.

2.9.1 Edge-triggered interrupts

An edge-triggered interrupt is an interrupt signalled by a level transition on the interrupt line, either a falling edge (high to low) or a rising edge (low to high). A device, wishing to signal an interrupt, drives a pulse onto the line and then releases the line to its inactive state. If the pulse is too short to be detected by polled I/O then special hardware may be required to detect the edge.

2.10 Sequential flow in FPGA

Even though FPGAs are able to perform pure parallel tasks, contrary to multi-threading in single core CPUs, where part of each tasks is executed one at a time, assuming they are assigned the same priority. FPGAs are often called upon to perform sequence and control-based actions such as implementing a simple communication protocol. For a designer, the best way to address these actions and sequences is by using a state machine. State machines are logical constructs that transition among a finite number of states. A state machine will be in only one state at a particular point in time. It will, however, move between states depending upon a number of triggers.

Theoretically, state machines are divided into two basic classes (Moore and Mealy) that differ only in how they generate the state machine outputs. In a Moore type, the state machine outputs are a function of the present state only. A classic example is a counter. In a Mealy state machine, the outputs are a function of the present state and inputs.

2.11 Audio properties

Digital audio (including digitized speech and music) has significantly lower bandwidth requirements than video. Digital audio, however, has its own unique properties that must be considered when designing multimedia network applications. To understand these properties, let's first consider how analog audio (which humans and musical instruments generate) is converted to a digital signal:

- The analog audio signal is sampled at some fixed rate, for example, at 8,000 samples per second. The value of each sample is an arbitrary real number.
- Each of the samples is then rounded to one of a finite number of values. This operation is referred to as quantization. The number of such finite values - called quantization values - is typically a power of two, for example, 256 quantization values.
- Each of the quantization values is represented by a fixed number of bits. For example, if there are 256 quantization values, then each value - and hence each audio sample - is represented by one byte. The bit representations of all the samples are then concatenated together to form the digital representation of the signal. As an example, if an analog audio signal is sampled at 8,000 samples per second and each sample is quantized and represented by 8 bits, then the resulting digital signal will have a rate of 64,000 bits per second. For playback through audio speakers, the digital signal can then be converted back - that is, decoded - to an analog signal. However, the decoded analog signal is only an approximation of the original signal, and the sound quality may be noticeably degraded (for example, high-frequency sounds may be missing in the decoded signal). By increasing the sampling rate and the number of quantization values, the decoded signal can better approximate the original analog signal. Thus (as with video), there is a trade-off between the quality of the decoded signal and the bit-rate and storage requirements of the digital signal.

The basic encoding technique that we just described is called pulse code modulation (PCM). Speech encoding often uses PCM, with a sampling rate of 8,000 samples per second and 8 bits per sample, resulting in a rate of 64 kbps. The audio compact disk (CD) also uses PCM, with a sampling rate of 44,100 samples per second with 16 bits per sample; this gives a rate of 705.6 kbps for mono and 1.411 Mbps for stereo. PCM-encoded speech and music, however, are rarely used in the

Internet. Instead, as with video, compression techniques are used to reduce the bit rates of the stream. Human speech can be compressed to less than 10 kbps and still be intelligible. A popular compression technique for near CD-quality stereo music is MPEG 1 layer 3, more commonly known as MP3. MP3 encoders can compress to many different rates; 128 kbps is the most common encoding rate and produces very little sound degradation. A related standard is Advanced Audio Coding (AAC), which has been popularized by Apple. As with video, multiple versions of a prerecorded audio stream can be created, each at a different bit rate. Although audio bit rates are generally much less than those of video, users are generally much more sensitive to audio glitches than video glitches. Consider, for example, a video conference taking place over the Internet. If, from time to time, the video signal is lost for a few seconds, the video conference can likely proceed without too much user frustration. If, however, the audio signal is frequently lost, the users may have to terminate the session.

A multimedia system in a vehicle must be able to read the different storage media and their coding formats. In addition, the different transmission formats of digital audio broadcasting and digital video broadcasting must be identified. Depending on the equipment variant, high transmission rates may be generated, which must be processed by the system. In contrast to the vast majority of other bus systems coming from different application areas, this system has been designed for Multimedia Data Transport throughout a system, as its name already indicates. In contrast to home systems, the car requires a multi source / multi sink environment, where multiple audio and video sinks, amplifiers, headphones and multiple displays are operated at the same time. There is a great variety of transmission formats in the audio field. In addition to stereo reproduction, different surround formats are relevant for the use in vehicles and are often employed in professional systems. The following is a rough overview:

2.11.1 Stereo

The uncompressed signal has a data rate of 1.4 Mbit/s at a sampling rate of 44.1 kHz and a 16-bit resolution. The most widely used compression rate is 128 kBit/s for MP3.

2.11.2 Analog Dolby Surround Pro Logic

This system consists of the channels front right, front center (center speaker for voices), front left, and rear left, rear right as one channel. The surround channel is applied to the two rear speakers and is time-delayed with regard to the front channels and band-limited to 100 to 7,000 Hz. The center speaker positions the voices to the center, which, in a car, makes localization rather independent of the listening position.

2.11.3 Analog Dolby Surround Pro Logic II

The Dolby Surround Pro Logic II enhances the Pro Logic to a 5.1 system by incorporating two separate surround channels without band limitation, and a subwoofer channel. 18.5 Audio and Video Transmission Formats 299

2.11.4 Dolby Digital, AC3

AC3 comprises five separate channels without band limitation and a separate subwoofer channel (Low Frequency Effect, LFE) limited to 120 Hz. At a sampling frequency of 48 kHz and a 16-bit resolution, transmission rates of 384 to 448 kbit/s accrue for the data which have a lossy compression ratio of 10:1. For two-channel stereo, the data rate is reduced to 192 kBit/s. Whereas MPEG-2 is the DVD standard in the video field, AC3 is the DVD standard in the audio field. MPEG-2 can only be used on a DVD as an additional audio coding format.

2.11.5 MPEG-2

MPEG-2 is a coding standard which can also be found on a DVD for audio signals, in addition to AC3. The data rate for six channels is about 400 kbit/s. MPEG-2 AAC (Advanced Audio Coding) is an enhancement which manages with half the data rate for the same quality as MPEG-2 and admits the coding of up to 48 audio channels, 16 subwoofer channels and 16 commentary channel. MPEG-2 is used as a standard in digital TV d(DVD satellite, DVD cable, DVD terrestrial). In Japan, MPEG-2 has been used as the exclusive sound format for all digital broadcasting systems since the beginning of the year 2000.

2.11.6 Digital Theatre System (DTS)

DTS is also a 5.1 system. Compared to AC3 it is distinguished by a better sound quality and increased dynamics. DTS has considerably less data compression and works with 1.536 Mbit/s for surround coding.

2.11.7 Logic 7

The 7.1 surround system developed by Lexicon controls the eight independent channels either digitally or develops a corresponding control from a stereo signal on the basis of a matrix decoding. It is thus possible to simulate the surround impression and achieve a very good ambient sound reproduction. A 5.1 source signal is enhanced to 7.1. Logic 7 is often used as professional system in vehicles.

Chapter 3

Hardware Platform

This chapter focuses on the description of the hardware platform that will support the development of the system, comprising FPGA technology, methods to accelerate the design flow and how they fit into the process of the development of this project. Moreover, considerations on the Physical and Logical Layers of the communication protocol to be developed are also outlined.

3.1 FPGA Technology

Conventional approaches to implementing communication protocol devices have disadvantages, such as, regarding their flexibility, scalability, performance, usability and/or their cost/performance ratio. Thus it makes sense to look for alternatives, such as the use of FPGA technology, which analysing the market demands, is the most natural choice.

The FPGA is structured in logic elements which, on the one hand, can be used for directly implementing specific functionality and, on the other hand, also support the loading of individual IP Cores providing pre-engineered functionality. These IP Cores can be "wired together" to provide the overall functionality of an FPGA. For an FPGA, IP Cores are available from different sources, including FPGA manufacturers and various service providers. Plus, they also can be developed individually to meet specific needs.

3.2 Soft-core Processors

One type of IP Core provided by FPGA manufacturers is the processor IP Core. It allows the execution of individual programs as in a standard processor. Modern FPGA architectures in addition include an System-on-a-Chip (SoC) hard processor system, providing the ultimate combination of hardened intellectual property (IP) for performance and power savings with the flexibility of programmable logic.

A major advantage of this approach is that, based on these individually available IP Cores, a specific set of hardware and software can be loaded into an FPGA to deliver the desired functionality. Embedded processors and hardware acceleration offer the flexibility, performance, and cost effectiveness in a development flow that is familiar to software developers. A DSP designer can combine a software design flow along with hardware acceleration. In this flow, the software developer first profiles C code and identifies the functions that are the most performance-intensive. These device features provide very high DSP capability in FPGAs compared to standalone DSP processors. Using FPGAs, DSP designers can customize their hardware for optimal implementation of their applications. All in all, the FPGA offers an additional level of flexibility unknown in other electronic components. For implementing a communication protocol oriented device, a hardware part (interface with the network) as well as a software part (protocol stack) is required.

3.2.1 The Microblaze Processor

As a full-featured, FPGA optimized 32-bit Reduced Instruction Set Computer (RISC) soft processor, Microblaze meet requirements for diverse applications such as industrial, medical, automotive, consumer, and communication infrastructure markets among others. MicroBlaze is a highly configurable and easy to use processor and can be used across FPGAs and All Programmable (AP) SoC families.

MicroBlaze is highly configurable IP core supporting various configuration options. With highly flexible and configurable core, there is the chance to implement virtually any processor use case, from a very-small-footprint state machine or micro controller to a high-performance, compute-intensive microprocessor-based system running an operating system.

3.3 Hybrid CPU+FPGA platform

The FPGA technology provides a great flexibility on the integration of various peripherals and promotes systems scalability through the reconfiguration of the involved components, as well as the reutilization of Intellectual Property cores. It also fosters portability, celerity in the prototyping process and modularity, due to its technological agnosticism. The employment of FPGA technology for computation offloading purposes, enables the fulfilment of both bandwidth and time requirements.

With this understanding of the fundamental FPGA components, you can clearly see the advantage of implementing your logic in hardware circuitry: you can realize improvements in execution speed, reliability, and flexibility. However, you face trade-offs using only an FPGA for the processing and I/O connectivity in your system. FPGAs do not have the driver ecosystem and code/IP base that microprocessor architectures and Operating Systems do. In addition, microprocessors coupled with Operating Systems provide the foundation for file structures and communication to peripherals used for many, often essential, tasks such as logging data to disk.

Embedded processors and hardware acceleration, however, offer the flexibility, performance, and cost effectiveness in a development flow that is familiar to software developers. A DSP designer can combine a software design flow along with hardware acceleration. In this flow, the software developer first profiles C code and identifies the functions that are the most performance-intensive.

As a result, a hybrid architecture, sometimes called a heterogeneous architecture, should be the most suitable approach for a optimized development of the proposed system, granting at the same time, the compliance with its specification, specially as far as performance, timing requirements and determinism are concerned.

Chapter 4

Communication Protocol Stack

The system is supposed to support a wide array of multimedia applications, which means, not only should the system be able to meet the timing requirements imposed by, for example, audio properties but also it should provide effortless integration of different combinations of modules which altogether form a multimedia system. Therefore, building a single generic multimedia device that would implement all the features that can possibly be integrated in a vehicle, would be non-sense from a scalability point of view, given the increasingly complexity that would have to be dealt with. So, the optimal approach would be to go for a distributed application system. In a distributed system, components located on networked devices communicate and coordinate their actions by exchanging messages which contain commands to be processed. The components interact with each other in order to achieve a common goal, as exemplified in the figure 4.1.

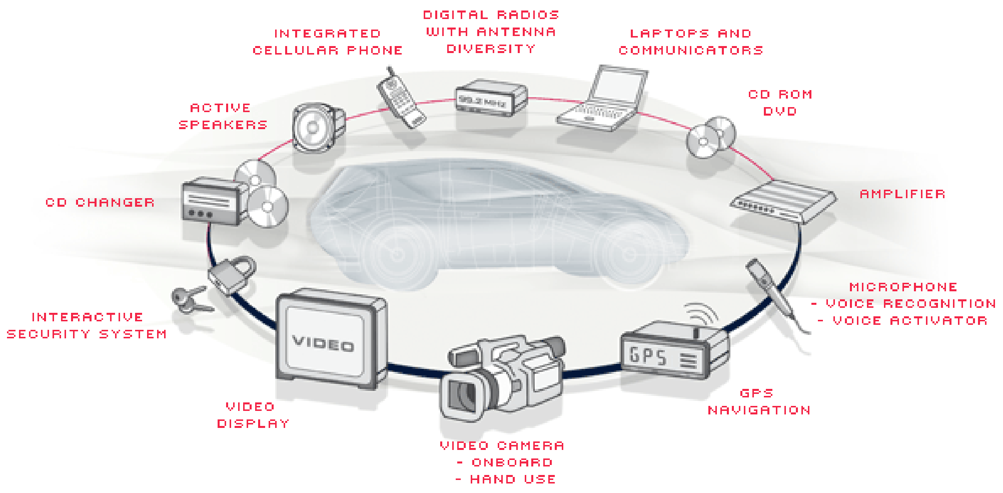
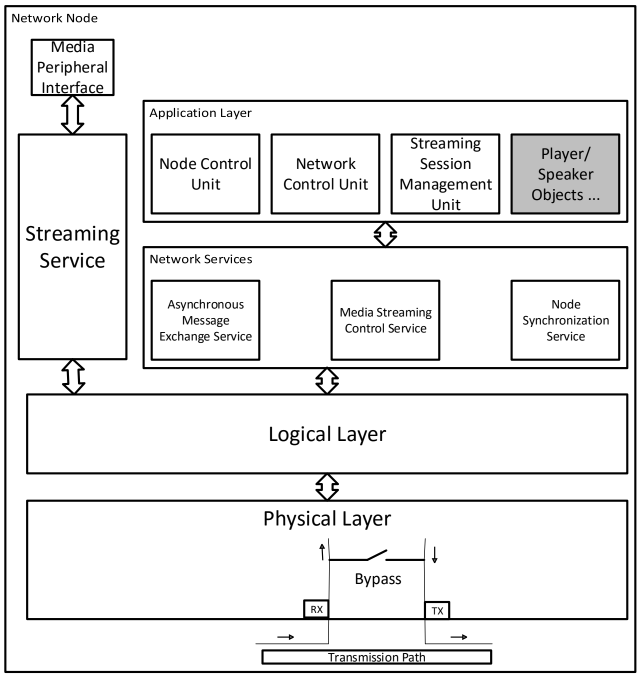

INTEGRATED CELLULAR PHONE
DIGITAL RADIOS WITH ANTENNA DIVERSITY
LAPTOPS AND COMMUNICATORS
CD ROM DVD
ACTIVE SPEAKERS
CD CHANGER
AMPLIFIER
MICROPHONE - VOICE RECOGNITION - VOICE ACTIVATOR
INTERACTIVE SECURITY SYSTEM
VIDEO
GPS
GPS NAVIGATION
VIDEO DISPLAY
VIDEO CAMERA - ONBOARD - HAND USE

Figure 4.1: Example of network oriented for media applications

Besides meeting the previously stated timing requirements, as the system consists of a network where nodes communicate with each other, it should also have mechanisms to coordinate the way communication is carried on and, as importantly, the ability to properly configuring each device before it can initiate the exchange of data with the other ones as well as dealing with failures in such a way that all the devices but the ones that have failed, continue their normal operation. A conclusion that can be drawn is that the system will result in the combination of two components: the synchronous component, for time-sensitive data, such as audio, to be consistently delivered on time to the designated destination; the asynchronous component, which is reserved for the system's administration and mode of operation control purposes. Both of these components are going to be explored, so as to build a proposal of a system that would be suitable for a wide range of multimedia applications, many of which found in the automotive industry.

Provided that implementing the whole system in RTL is very demanding from a development effort standpoint, comes the necessity to implement most of the parts of the application in software and taking advantage of the abstraction layer and services an operating system boasts. Then, for parts of the application that are required to be offloaded from the CPU, for timing requirements ensuring purposes or because FPGA technology's additional flexibility as far as hardware customization is concerned, makes the development of certain parts of the application easier than by means of software. The right balance between ease of development and meeting the system's requirements requires the spot-on establishment of the border between the parts of the application developed in RTL and in software.

4.1 Application Framework



Network Node
Media Peripheral Interface
Application Layer
Node Control Unit
Network Control Unit
Streaming Session Management Unit
Player/ Speaker Objects ...
Streaming Service
Network Services
Asynchronous Message Exchange Service
Media Streaming Control Service
Node Synchronization Service
Logical Layer
Physical Layer
Bypass
RX
TX
Transmission Path

Figure 4.2: Network communication protocol - API for the developer

As a mean of reducing the effort required by the developer, a framework can for example provide support for an application that consists of a microphone to be connected to a speaker, by means of the developed communication protocol, as stated in the figure 4.3.

4.1.1 Interface with the Developer

The following set of functions get the entire system running, being common to all media oriented devices.

As seen in the diagram 4.4, the play() function causes the signal captured by the source device to be reproduced by the sink device, whereas the stop() function interrupts such process.

More specific Objects, to control specific media components are also part of the framework, providing an abstraction layer the developer to control them. The table 4.1 shows attributes that typically are concerned with an Audio Amplifier,

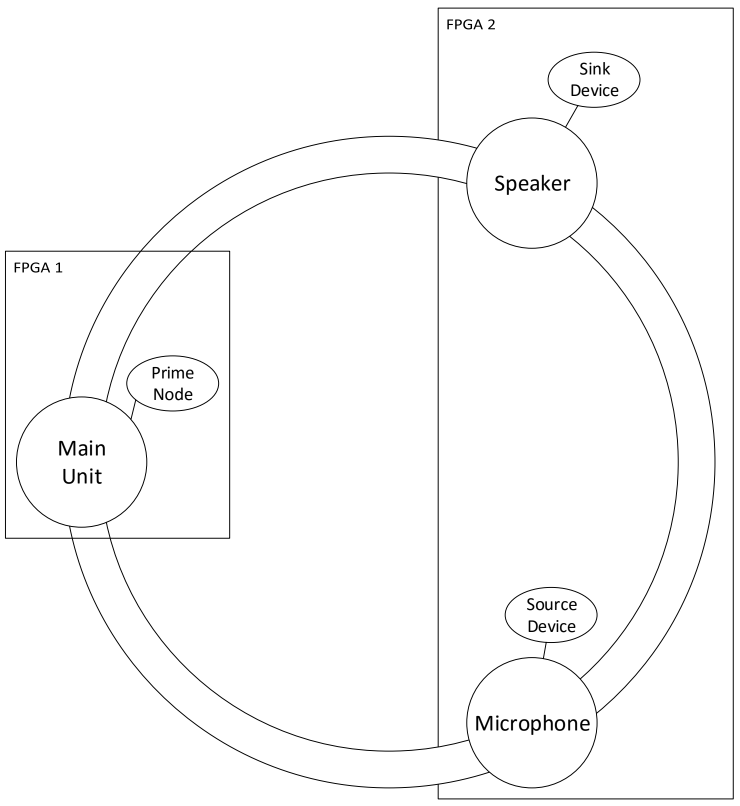


Figure 4.3: Overview of a real case scenario

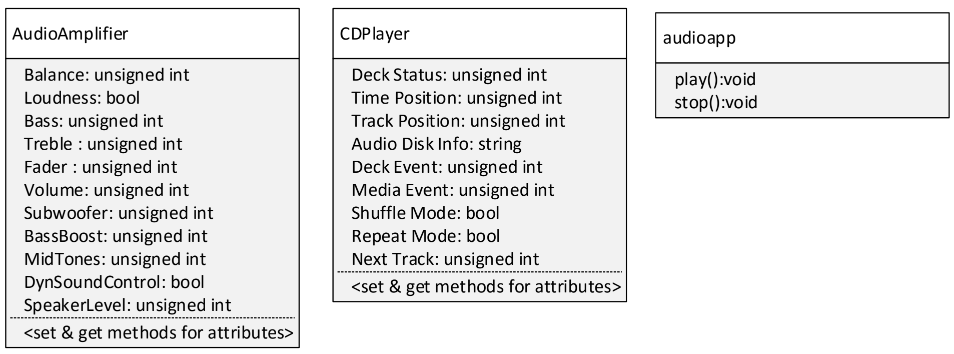


Figure 4.4: UML Class diagram concerning API for the developer

Table 4.1: List of attributes for Audio Amplifier Object

Atributes	**Type**	**Description**
Balance	Unsigned Int	Apply sound bias to left or right speaker group
Loudness	bool	Boost low sound frequencies at low volume
Bass	Unsigned Int	Adjust bass parameter
Treble	Unsigned Int	Adjust Treble Parameter
Fader	Unsigned Int	Apply sound bias to rear or front speaker group
Volume	Unsigned Int	Volume level adjustment
Subwoofer	Unsigned Int	Adjust subwoofer level
BassBoost	Unsigned Int	Bass boost for headphones
MidTones	Unsigned Int	Adjust gain for middle frequency sound
DynSoundControl	bool	Enable speed dependent volume adjustment
SpeakerLevel	Unsigned Int	Set sound level to each speaker

to which set and get methods are associated to each one, even though they have been omitted in this documentation.

Similarly, the table 4.2 contains details about the list attributes for a Disk player Object, which is also part of the API.

4.1.2 Interface with Transport Layer

As the system is powered on, it initialization is automatically processed by the Network Control Unit and all the modules it depends on.

The diagram represented in the figure 4.5 shows how addresses are assigned to each node. The Prime Node triggers the broadcast of a chain of messages, transmitted from one node to another, following the methodology explained in the previous subsection.

Having assigned the addresses to each node, messages can now be sent to a specific node, as its address is now known. One of these is for getting to know what functions each node has implemented, so that if such routines have to be remotely called, the system automatically recognizes the address it has to send the command to, therefore reducing effort required from the developer, with regards to developing a distributed application. That process is represented in the sequence diagram in

Table 4.2: List of attributes for Disk Player Object

Atributes	**Type**	**Description**
Deck Status	Unsigned Int	Status of the drive (Play/Pause/Stop/...)
Time Position	Unsigned Int	Overall time position of the disk
Track Position	Unsigned Int	Time past beginning of the track
Audio Disk Info	string	Name and length of the disk
Deck Event	Unsigned Int	Status monitoring of the drive (normal, overtemperature, ...)
Media Event	Unsigned Int	Integrity of the data or reached end of CD
Shuffle Mode	bool	Play tracks randomly
Repeat Mode	bool	Automatically play the disk again
Next Track	Unsigned Int	Index of the next track to play

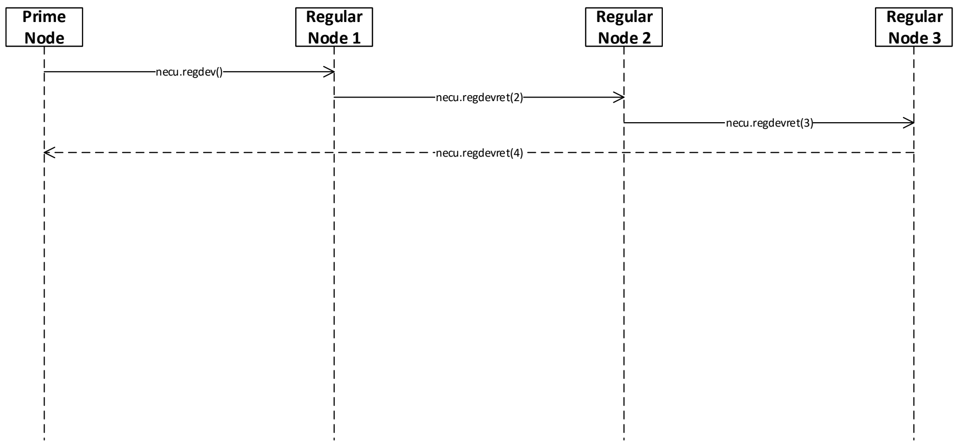


Figure 4.5: Workflow for assigning addresses to devices on the network

Table 4.3: Set of functions for interface with Transport Layer for node management

Class Members	Description
set_prime_node	Set node role as Prime Node
set_regular_node	Set node role as Regular Node
set_selfaddress	Set own address
app_loaded	Assert that the application is loaded and node is ready to enter the network

Table 4.4: Set of functions for interface with Transport Layer for network management

Class Members	Description
regdev	Register device in network, assingning an address
queryobj	Ask for implemented Objects' IDs in the nodes

the figure 4.6

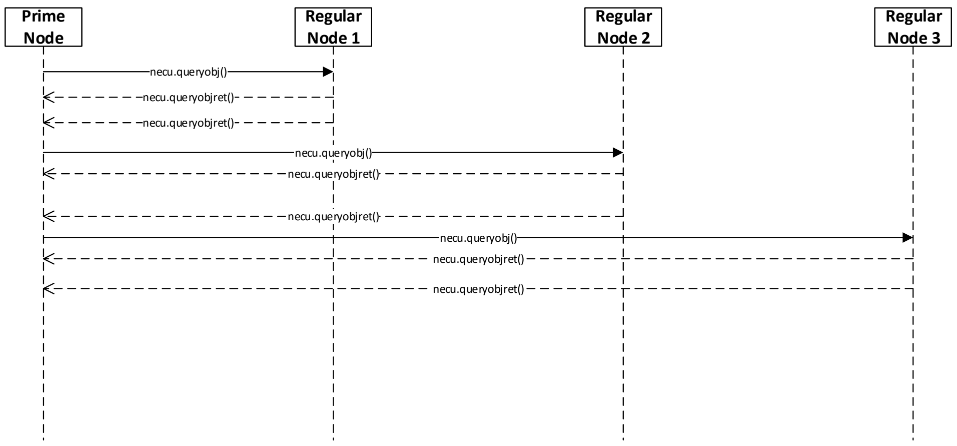


Figure 4.6: Querying implemented object in each Network Node

Following the diagrams 4.5 and 4.6, the tables 4.3 and 4.4 describe the set of functions that compose the interface between the Application Framework and Transport Layer, for node and network management purposes, respectively.

Calling the play() function opens a thread so as to proceed with the workflow that is indicated in the 4.7 diagram. The Streaming Session Management Unit has the duty of configuring the whole system, that is, all of the involved nodes, prior to actually allowing the data to be exchanged between the source and sink devices.

The next diagram, in the figure 4.7, showcases how messages should be exchanged in order for synchronous data to be transmitted between a source and sink device, which requires prior channel allocation and the establishment of a connection, after questioning how many channels are required for that connection and whether they are available. This chain of messages is automatically exchanged once the queryreqchan() method is called. In order to cancel the exchange of data between two devices, the reverse process as of when it comes to establishing a synchronous connection, comes into execution, as included in the same diagram.

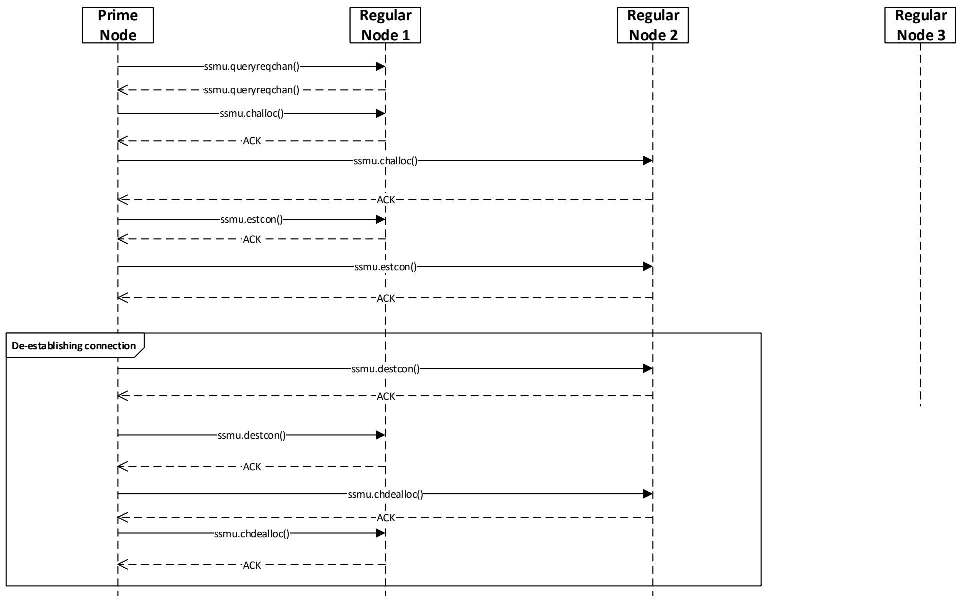


Figure 4.7: Establishing and terminating streaming connections

Following the diagram 4.7, the tables 4.5 describe the set of functions that compose the interface between the Application Framework and Transport Layer, for synchronous connections (de-)establishment and management purposes.

In order to do the reverse process, which is interrupting the data exchanged between both source and sink devices, the stop() function requires again the Streaming Session Management Unit to revert back the process that has been done with the previously stated play() function.

4.1.3 Interface with Media Oriented Components

As stated before, components like an Audio Amplifier or CD Player do not only require a datapath through which time-sensitive data is exchanged, but also require

Table 4.5: Set of functions for interface with Transport Layer for synchronous connections management

Class Members	Description
queryreqchan(int):void	Ask how many channels are required to be allocated
challoc(int):void	Allocate required channels on source/sink device
estcon(int):void	Establish connection between sink and source device
addcon():void	Add entry to Active Connections table
chdealloc(int):void	Deallocate required channels on source/sink device
destcon(int):void	De-establish connection between sink and source device
rmcon():void	Remove entry from Active Connections table

a mean of exchanging information concerning their status or configuration.

With the current system architecture, the Object which is part of the framework can easily communicate with the corresponding peripheral, bz means of the instantiation of a IP core that implements a standard communication protocol, as is the case of SPI or I2C. Such IP Core would implement a controller compliant with the communication protocol to be used, which would then communicate with the media peripherals by means of the designated transmission path. The figure 4.8 shows an example of such scenario, where a I2C Master Controller is attached to the Processor Local Bus, which enables the Object embedded in the application layer, running in the CPU, to communicate with an external media peripheral, so as to control it.

4.2 Transport Layer

The Transport Layer provides an API for the Application Framework, being composed by the following modules:

- Streaming Session Management Unit, that takes care of all the necessary procedures concerning seting up the exchange of time-sensitive data.
- Node Control Unit, that is entitled towards controlling the mode of operation

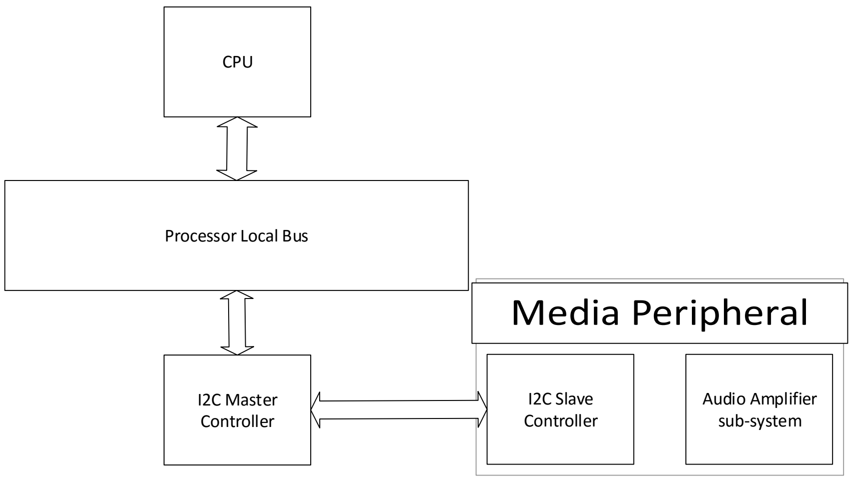


Figure 4.8: Example of interface between Object and corresponding peripheral

of the node and handling its own errors.

- Network Control Unit, that is responsible for managing the system as a whole and dealing with faults that may eventually occur

4.2.1 Streaming Session Management Unit

Before the streaming of data is possible, several steps have to be carried out in order to prepare the system to do so. Such preparation involves the configuration of the system, the sink(s) and source(s), a routine that can be automated by adding this feature in the framework this project is about, therefore minimizing the effort required by the developer as far as designing and implementing an application is concerned. The first system's preparation step in order to establish a synchronous connection is to query the node which will act as a source, with regards to how many synchronous channels are required to be allocated. After the Streaming Session Management Unit is sent back the corresponding answer, there is the need to check whether the number of required channels are free and can thus be allocated in a further stage in this process. If the aforementioned condition is valid, the Streaming Session Management Unit then proceeds to triggering the required routines to allocate the channels in the source and sink devices, by this order, being the last step the validation of the synchronous connection before data can be exchange between the sources(s) and sink(s). This stage involves the update of a table containing information about all the active synchronous connections, the

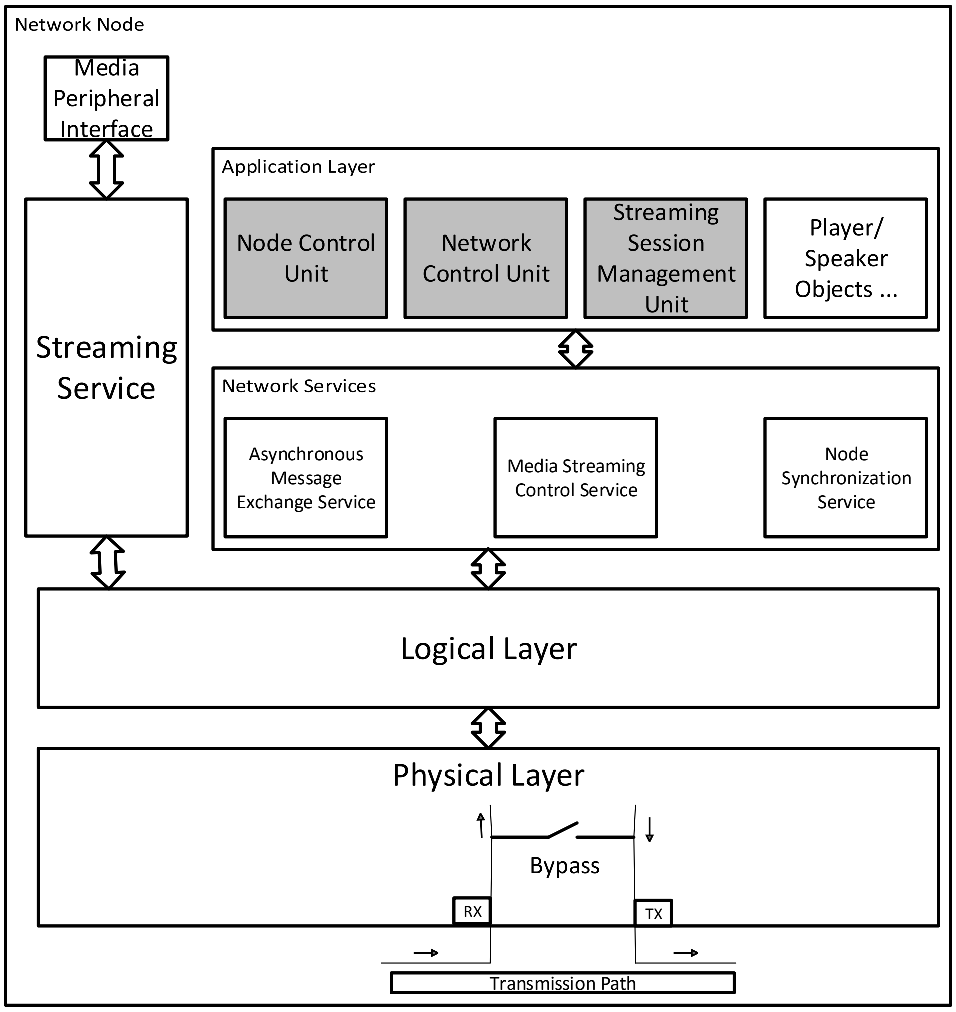

Network Node
Media Peripheral Interface
Application Layer
Node Control Unit
Network Control Unit
Streaming Session Management Unit
Player/ Speaker Objects ...
Streaming Service
Network Services
Asynchronous Message Exchange Service
Media Streaming Control Service
Node Synchronization Service
Logical Layer
Physical Layer
Bypass
RX
TX
Transmission Path

Figure 4.9: Transport Layer entities

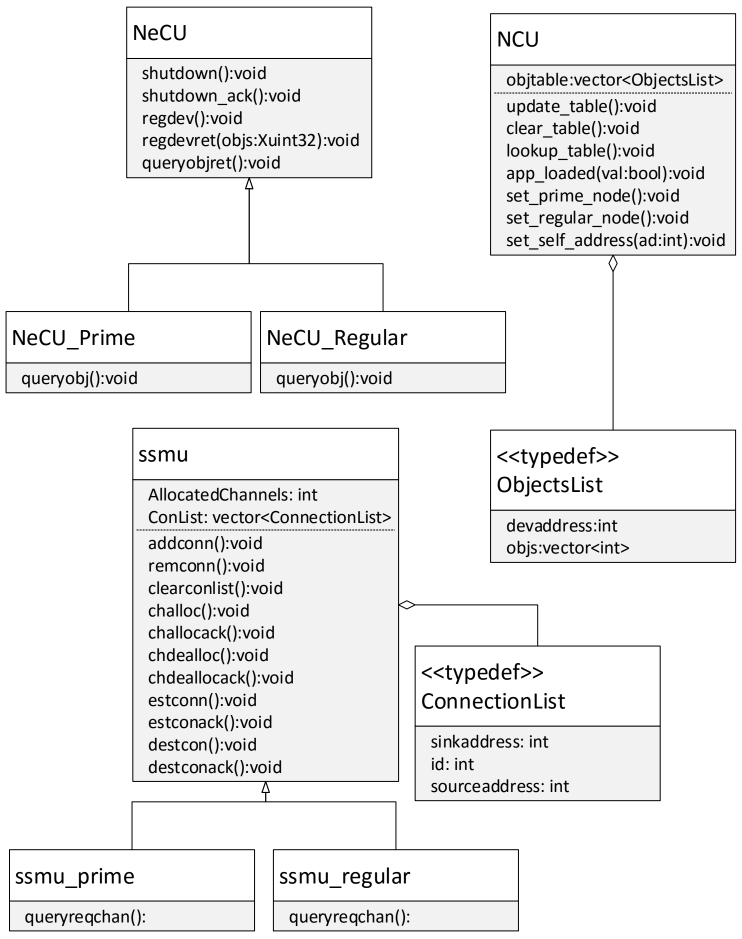


Figure 4.10: UML Class Diagram for Transport Layer

involved source and sink device(s) and which channels are allocated, so as to keep track of the bandwidth usage, information which is consulted when establishing a new connection, just as mentioned above. When it comes to closing a synchronous connection, the corresponding routines must be held in first place at the sink device(s) and only then on the source devices, so as to avoid the reproduction of random data (noise) that could be present if the source device was disconnected in first place. Finally, the table containing information about all the active synchronous connections is updated, by removing the associated connection, which translates into freeing the previously used bandwidth so that it can be available again to other synchronous connections eventually bound to be established.

The Streaming Session Management Unit is responsible for Building and removing synchronous connections.

Before introducing more details on the implementation of this part of the software application, the figure 4.11 shows the system's classes, their attributes, operations (or methods), and the relationships among objects.

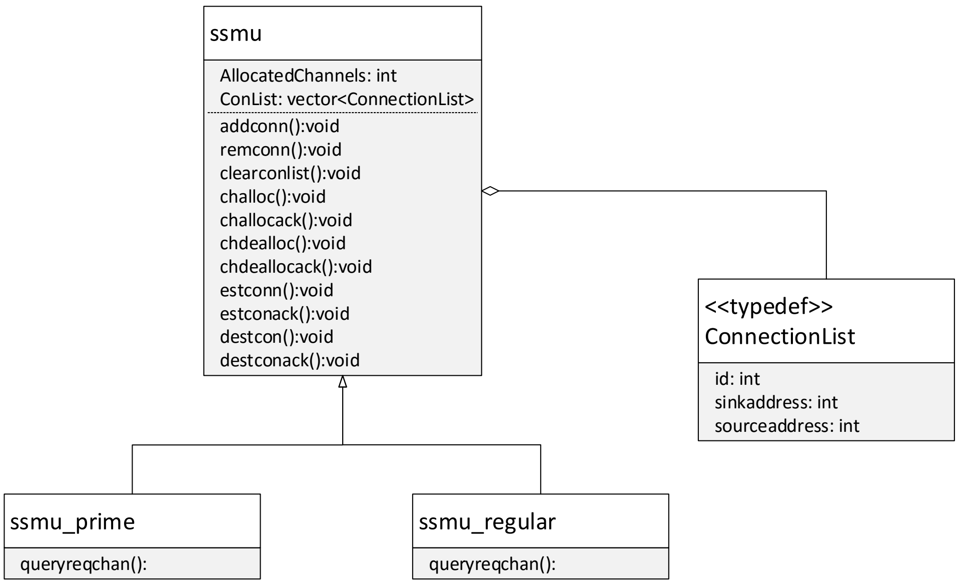


Figure 4.11: Streaming Session Management Unit Software Architecture

Besides the need to have a list of the currently active connections and how much and what channels each one are allocated to, methods to establish and de-establish synchronous connections are required. The body of the queryreqchan() function is implemented differently in the Prime Node and Regular Node, as in the Prime Node it is oriented to querying a source device how many channels are required and all the regular node needs to do is replying to such question.

4.2.2 Node Control Unit

The Node Control Unit is responsible for storing the essential data resulting from the network initialization, that will then be used when messages have to be exchanged between the nodes in runtime. The use case diagram presented in the in the figure 4.12 represents a list of action or event steps, typically defining the interactions between a role (known in the Unified Modelling Language as an actor) and a system, to achieve a goal.

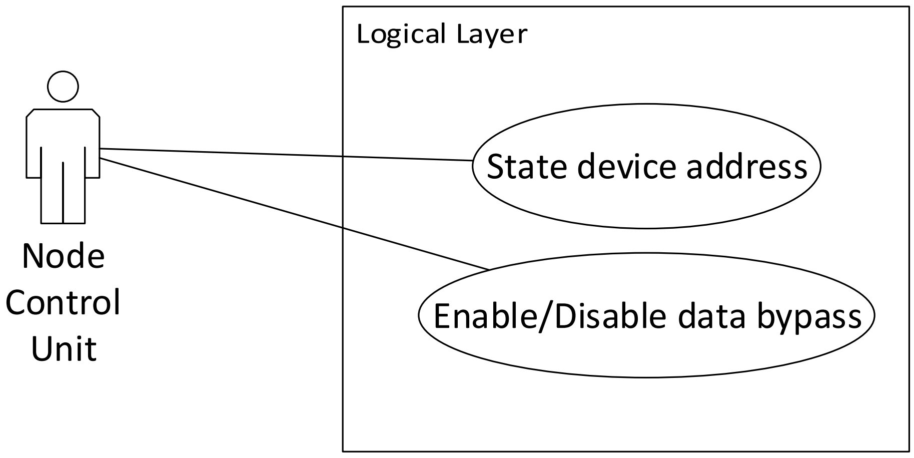


Figure 4.12: Node Control Unit controlling the node's mode of operation

Before introducing more details on the implementation of this part of the software application, the figure 4.13 shows the system's classes, their attributes, operations (or methods), and the relationships among objects. The Node Control Unit includes a table that contains data relative to the address of the registered nodes in the network and the object implemented in each one. Methods to manipulate the data in that table are made available, being them the functions update_table() clear_table() and lookup_table().

Nodes leaving or entering the network after normal system operation has been established can cause disturbances which may affect the Quality of Service. With regards to leaving a network, this may not be controllable, as nodes can crash without prior notice, due to several reasons. However, as for entering the network, there is the possibility to implement a workflow that all nodes would need to follow.

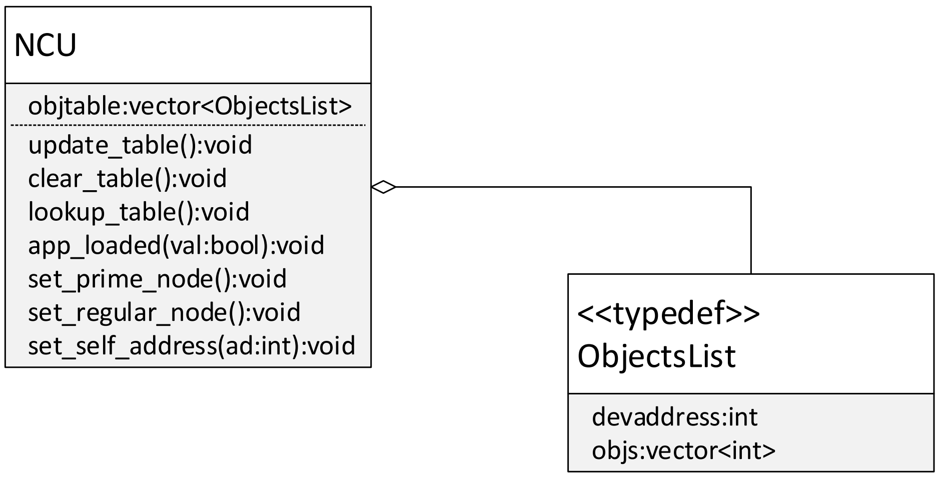


Figure 4.13: UML class diagram of the Node Control Unit Object

In an effort to minimize disturbances when the system is on duty, network nodes should not be allowed to enter the network during this phase. Instead, the act of joining the network should only take place when the system is in configuration mode, provided that there are no active synchronous connection during this stage, as can be seen in the figure 4.14. When devices are notified that the system has entered in configuration mode, their bypass mechanism can be disabled, causing datagrams to be received and parsed prior to passing them on the the next network node, rather than solely mirroring the data received in the RX port to the TX port.

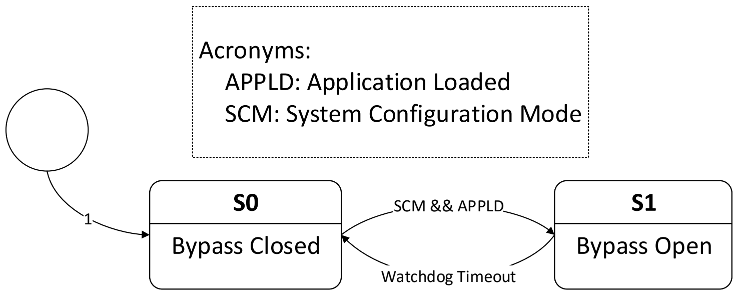


Figure 4.14: Controlling the state of the bypass mechanism

4.2.3 Network Control Unit

The Network Control Unit plays a role in the System Initialization and Fault detection and handling. The use case diagram in the figure 4.15 represents a list of action or event steps, typically defining the interactions between a role (known in the Unified Modelling Language as an actor) and a system, to achieve a goal.

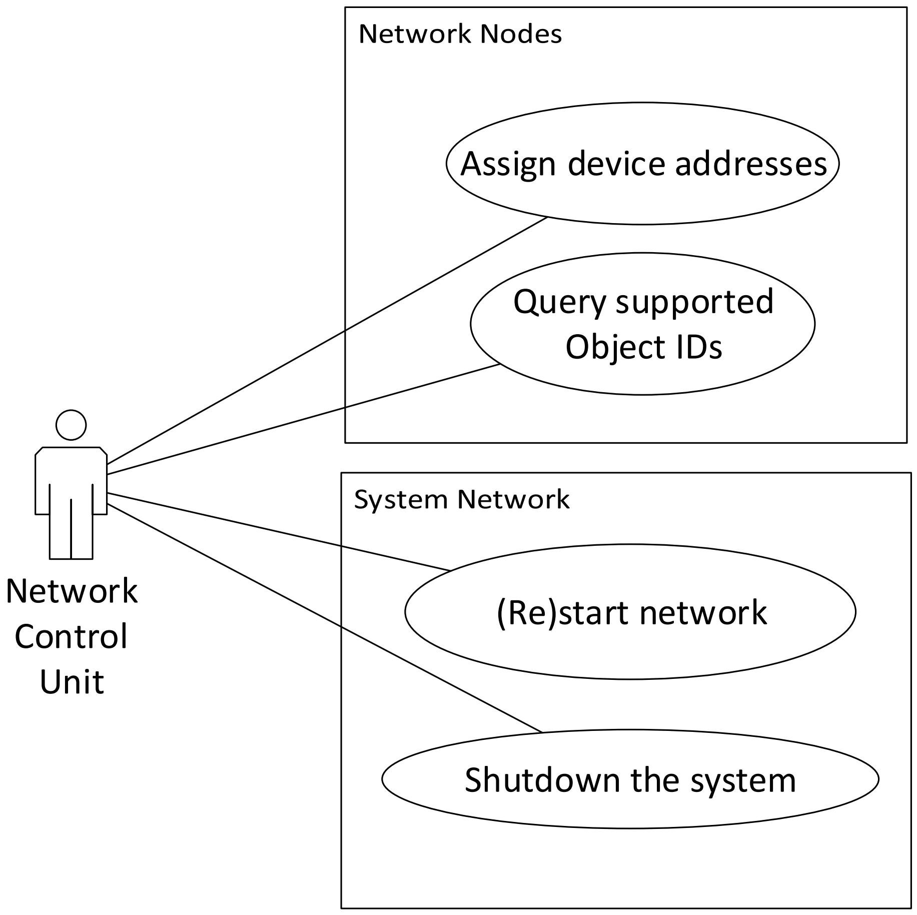


Figure 4.15: Network Control Unit effects on the Network Nodes and the System's Network

When the system is powered on, a series of steps have to be followed until it is ready to perform the designated tasks. Firstly, empty frames are emitted by the Primary Node, which after coming back from the network, validate the integrity of the transmission path as well as the transceivers embedded in each Network Interface Controller. Then, addresses are assigned by the Primary Node to all of the active Secondary Nodes, so that communication between them is possible in further stages of execution of the system.

Then, the Primary Node sends a message using a broadcast address, so that all nodes receive it. Contained in that message is the command to call a function which stores the new network address in a node's register, according to the value passed as an argument of such command, by the previous node in the network.

After that, such argument is incremented by one, so that the next node receives as an argument of the aforementioned command, the value of the new address so that it can carry out the same process of registration, explained previously. After the message has gone through all the nodes and returns to the Primary Node, the value passed as an argument which was used to register the other node in the network, can now be regarded as an indication of how many nodes are registered in the network, as each of those which got connected, have incremented the field of the message corresponding to the next free network address by one.

Finally, the Network Control Unit is responsible for gathering the required information regarding which Distributed Objects are implemented in each device and deliver it to the Node Control Unit.

The figure 4.16 shows the system's classes, their attributes, operations (or methods), and the relationships among objects. Depending on whether the Node has

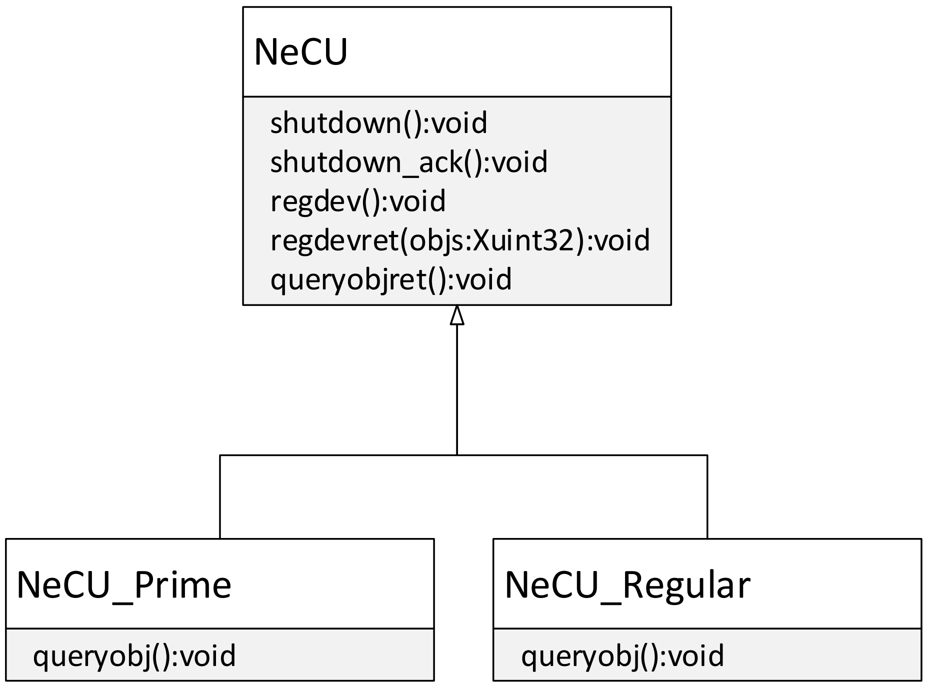


Figure 4.16: Network Control Unit Software Architecture

the role of Prime Node or a Regular Node associated, the body of the function queryobj() is different, as the Prime Node must register itself and disclose details about the objects implemented in it, before proceeding to the registration and querying the remaining nodes about the same details.

Due to several factors, nodes can suffer glitches or a major fault that prevents them to function properly. Unless potential issues are predicted and the necessary solutions are found during the process of development of the node, side effects can affect the entire network on the event of the failure of one node. To name a few examples, it can be the messages being held by a malfunctioning node, rendering the following devices to be unable to receive them, or the continuous corruption of the data to be spread through the entire network or even the successive transmission of random messages that may happen to change the configuration of any device in the network. These scenarios can easily affect the system's integrity and result in consequences way beyond a simply bad Quality of Service. Thus, when a node shows signs of abnormal operation, it must leave the network. Internally, mechanism have to be implemented in order to trigger such action, being a watchdog timer one possible solution to deal with this. If the watchdog timer is not reset within a certain period, it will assume that the system is not responding and will then reset it. Resetting the device will cause a digital bypass mechanism, implemented in RTL, to be activated, so that messages continue to travel around the network, without being read or manipulated by the device in question. The device continues in this disabled mode of operation until and only if it is ready to get back to normal operation and also solely up until the whole network enters in configuration mode, when devices are (re-)registered in the network with an unique address. When this event occurs, the bypass mechanism is disabled so that communications be resumed by the affected node(s). The network control units continuously monitors the status of the system, thus being able to detect unresponsive nodes. The lack of response by the nodes can be due to a defective transmission line, which causes the data frame not to complete its travel around the network and consequently resulting in triggering a timeout event, to signalize the anomaly. Another factor can be a problem within a node which caused it to shutdown or reset, which makes it unable to return to the network until the system is in Configuration Mode. Therefore, no acknowledgements will be sent back to the source node, if required. When that occurs, the sender node will retry the transmission of the same message until it reaches the limit of retries, which is the indicator of an unresponsive node. Having detected the fault, the system will restart, going through the same procedures as when it is powered on. This way, nodes that crashed due to a glitch may re-enter the network and nodes which are defective will be kept out of the network (because they are unable to receive any messages), thus preventing the occurrence of side-effects that can compromise the system's integrity.

Table 4.6: Framework available for Transport Layer entities

Class members	Description
gettarad():Xuint32	Get target address value
getsrcad():Xuint32	Get source address value
getfblockid():Xuint32	Get Object ID value
getinstid():Xuint32	Get Instance ID value
getfktid():Xuint32	Get Action ID value
getoptype():Xuint32	Get Action Type
getctdata():Xuint32	Get message parameters
settarad():void	Set target address value
setsrcad():voidXuint32	Set source address value
setfblockid():void	Set Object ID value
setinstid():void	Set Instance ID value
setfktid():void	Set Action ID value
setoptype():void	Set Action Type
setctdata():void	Set message parameters
pushfr():void	Queue message to send

4.2.4 Interface with Network Layer

Both Network Control Unit and Streaming Session Management Unit require the Application Message Exchange Service in order to communicate with the remaining network nodes, so that the system can be initialized and so that communication is possible with other node in order to configure them prior to establishing a synchronous connection, respectively. The table 4.6 explains which functions of the Application Message Exchange Service, situated in the underlying layer are available to the Transport Layer entitites.

The Streaming Session Management Unit also requires the Media Streaming Control Service (explained later in this document), so that the involved sources and sinks can be internally configured, based on the instructions sent by the Prime Node, in terms of channel allocation, type of operation (read/write) and enabling/disabling them. For that, the set of functions represented in the table 4.7 act as an interface between the Transport and Network level, with regards to this subject.

Table 4.7: Interface between Transport and Network Layers, for channel allocation, state and read/write operations

Atributes	Description
alloc_channels(mask:Xuint32):void	Apply channel allocation mask
en_channels(mask:Xuint32):void	Apply channel enable/disable mask
rw_channels(mask:Xuint32):void	Apply channel R/W operation mask

Table 4.8: Registers for interaction with the IP Core

Register Address	Register Index	Description
3	3:0	Address of the node itself
	4	Node acting as Prime or Regular Node
	5	Application is loaded (node can enter the network)
	6	Network is in configuration mode
	7	Flag to shutdown the system

4.2.5 Interface with IP Core

The Node Control Unit interacts with the IP Core, so as to set its address based on the initial setup of the network and to control when the device should enter the network, shutdown or so as to notify the Prime Node that the network should change its mode of operation in configuration mode, whether any node detects there has been a network node that stopped responding or problems in the transmission path disable the communication between all nodes, as summed up in the table 4.8.

4.3 Network Layer

The Network layer provides an abstraction layer for the implementation of the Transport Layer, in the following aspects:

- The exchange of messages with the Network, via its underlying layers, and handling of ACKs which is carried by the Application Message Exchange Service
- The control of the mode of operation of the time-sensitive routing datapath, done by the Media Streaming Control Service

- The periodic transmission of datagrams, carried out by the Node Synchronization Service, running in the Prime Node

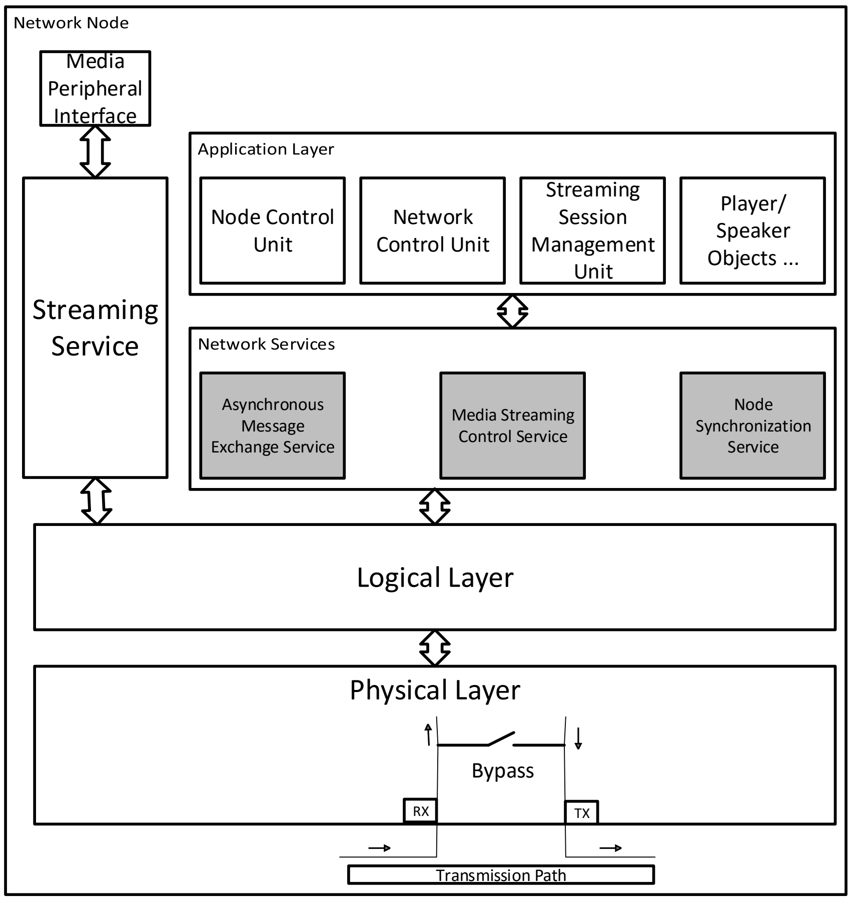

Network Node
Media Peripheral Interface
Streaming Service
Application Layer
Node Control Unit
Network Control Unit
Streaming Session Management Unit
Player/ Speaker Objects ...
Network Services
Asynchronous Message Exchange Service
Media Streaming Control Service
Node Synchronization Service
Logical Layer
Physical Layer
Bypass
RX
TX
Transmission Path

Figure 4.17: Network Layer entities

4.3.1 Application Message Exchange Service

In order to avoid unnecessary network congestion when trying to deliver a message to a busy node, which is for example, executing a command, additional mechanism are required to be implemented. The idea is that the target node captures the as soon as it is available to be processed, that is, after it has entirely been received and and validated by the integrity check algorithm. To do so, there is the need to associate the "ready to process the message" flag to an interrupt, which will cause the CPU to suspend the current code execution in order to perform the Interrupt Service Routine. Such routine stores the message, to be processed when possible. Situations in which bursts of messages are directed towards a device, should also be considered, in case processing a command contained in a message takes longer

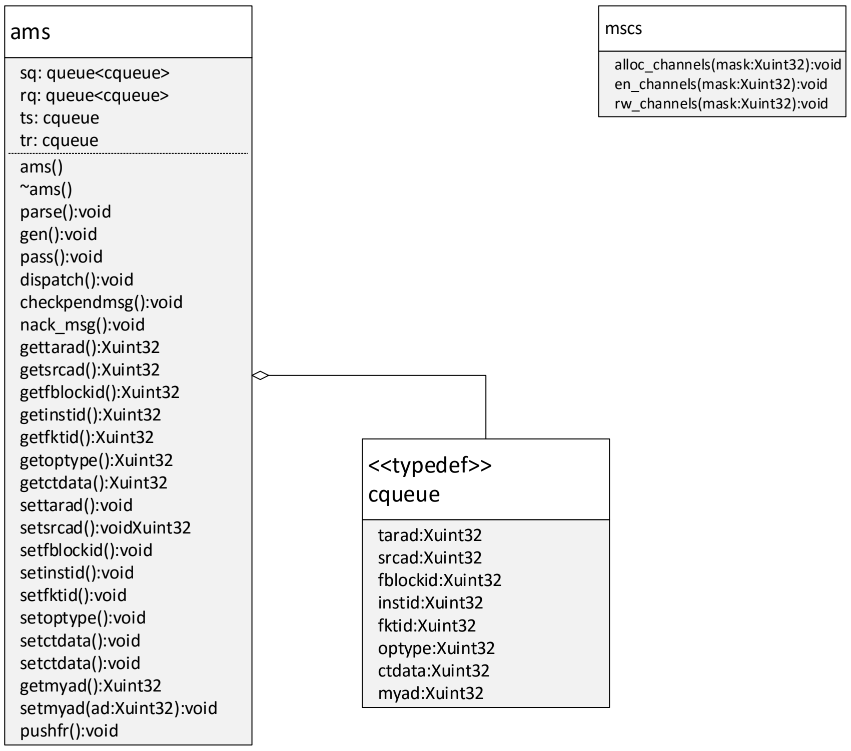


Figure 4.18: UML Class Diagram for Network Layer

than the period of arrival of new messages. Thus, it is required to implement a FIFO which acts as a storage facility for incoming messages, until the processor executes them, one at a time. Once one command is processed, it is removed from the top of the queue, so that the next command, if it exists, is also executed. When it comes to sending messages, each node only has a well-defined window of time in order to send a message, also one at a time. In an effort to ease the development of the objects that compose the Application Layer, so that the developer does not have to worry about the having to wait for the window of time to send one message after another, a FIFO for outbound message is also embedded in the Asynchronous Message Exchange Service, which acts as a temporary storage compartments for messages to be transmitted. When there is a free slot in the designated field of the data frame, it is filled by the message on the top of the queue. Contrarily to the FIFO for incoming messages, this one requires certain conditions with regards to popping a message from the front of the queue, after it has been sent out to the network. That will be explored in the next subsection. The figure 4.19 shows the storage of inbound messages in a queue, for further processing and the deployment of outbound messages to the datagram, to then be sent out to the network.

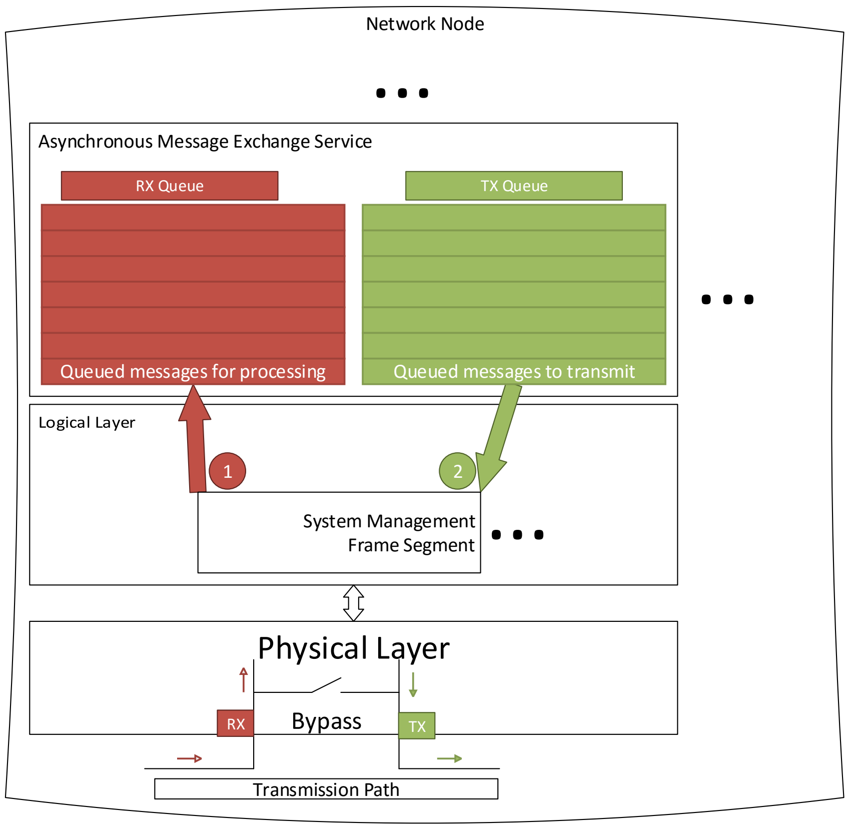


Figure 4.19: Queuing mechanism for inbound and outbound messages

The Asynchronous Message Exchange Service is responsible for dealing with Transmission Acknowledgement and also features the tasks concerning Message Queuing. The use case diagram included in the figure 4.20 represents a list of action or event steps, typically defining the interactions between a role and a system, to achieve a goal.

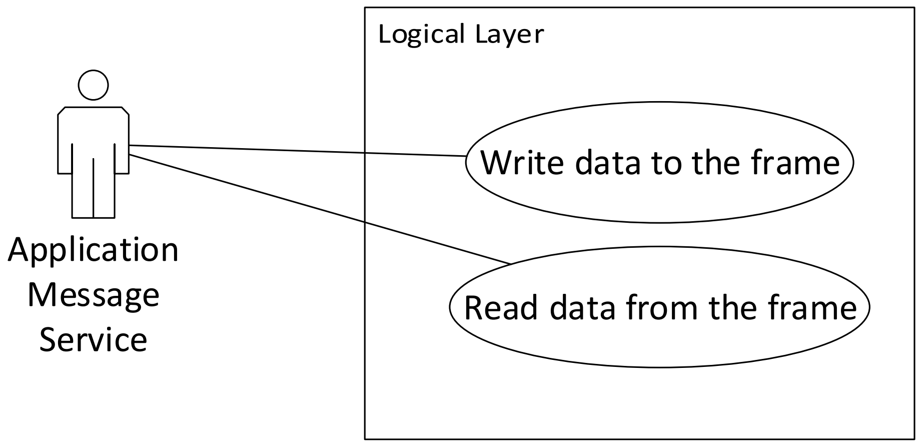


Figure 4.20: Effects of the Application Message Exchange Service in the Logical Layer

Before introducing more details on the implementation of this part of the software application, the figure 4.21 shows the system's classes, their attributes, operations (or methods), and the relationships among objects. The AMS class is comprised by the set and get methods responsible for storing the information received from the network, to be processed in a further stage, as well as the data to be sent out to the network when the opportunity arises, which can either be generated spontaneously by the application or be regarded as a response to a previously received message. The exchange of data with the network is done by the parse and gen function, which will be detailed later in this section.

The gen function verifies whether there are pending messages to be sent by the application to the network. If so, it proceeds to writing such information into the hardware implemented registers, which will then be forwarded to the network as soon as there is a free spot for them in the data frame. After that there is an indication that such frame has been sent out to the network with success or that it has been received by the addressee node, who returned an acknowledgement, that message is popped from the TX FIFO. Following that, if there is another pending message to be the hardware implemented registers are updated with the new data,

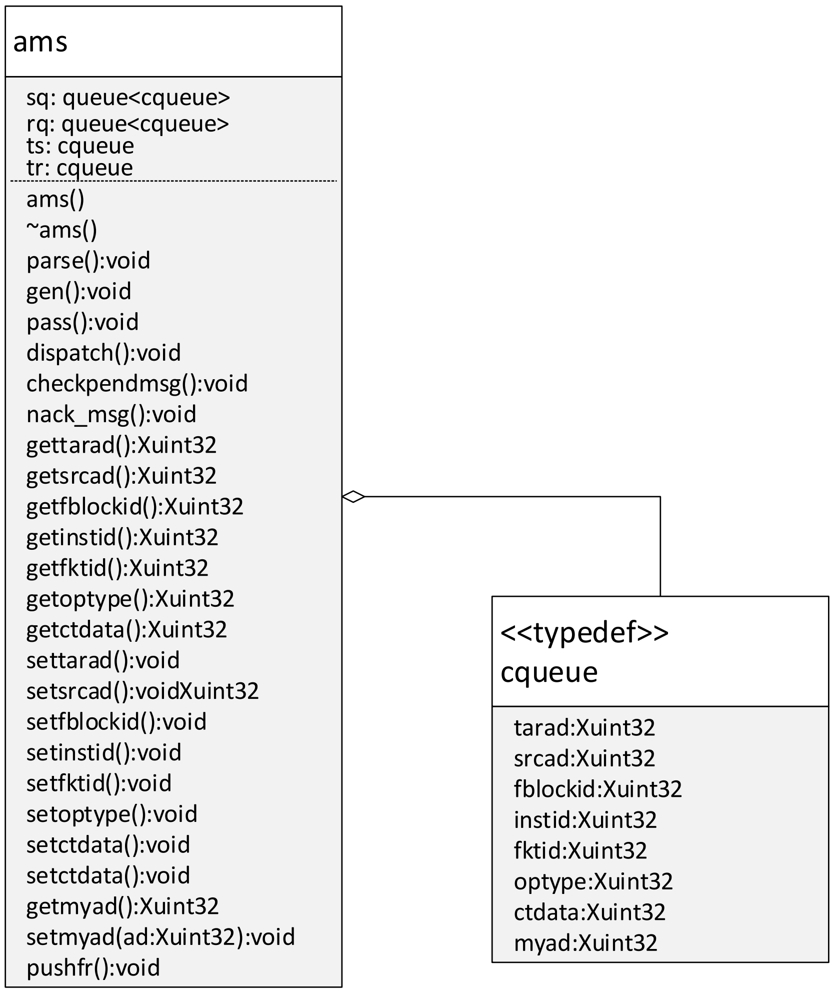
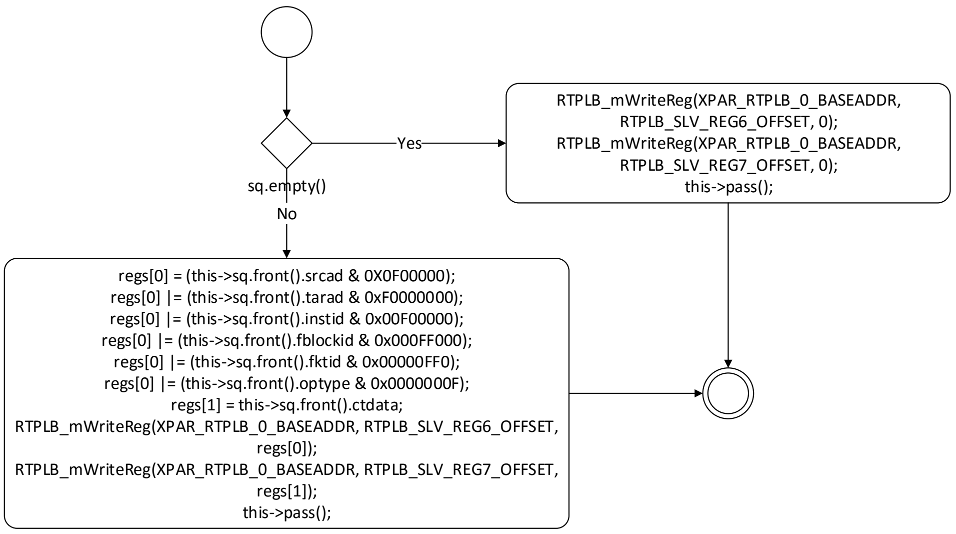


Figure 4.21: Application Message Exchange Service Software Architecture

otherwise the value of 0 to all fields is assigned. The figure 4.22 illustrates the workflow in order to write data from the CPU to the datagram by means of the PLB. All fields of the datagram are filled prior to the order to proceed with eh transmission of the data to be issued



```
RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR,
RTPLB_SLV_REG6_OFFSET, 0);
RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR,
RTPLB_SLV_REG7_OFFSET, 0);
this->pass();
```

```
regs[0] = (this->sq.front().srcad & 0X0F00000);
regs[0] |= (this->sq.front().tarad & 0xF0000000);
regs[0] |= (this->sq.front().instid & 0x00F00000);
regs[0] |= (this->sq.front().fblockid & 0x000FF000);
regs[0] |= (this->sq.front().fktid & 0x00000FF0);
regs[0] |= (this->sq.front().optype & 0x0000000F);
regs[1] = this->sq.front().ctdata;
RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR, RTPLB_SLV_REG6_OFFSET,
regs[0]);
RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR, RTPLB_SLV_REG7_OFFSET,
regs[1]);
this->pass();
```

Figure 4.22: Writing operation in datagram, via PLB registers

As per the figure 4.23, the parse function is part of an interrupt handler, executed as soon as a new message directed towards the node in question arrives. Such information is stored in the RX FIFO to then be processed by the corresponding application part. It also checks whether the received message is an acknowledgement to a previously sent message. If so, this means that message has successfully arrived at the destination, so it does no longer need to be re-sent. In that case, such message is deleted from the TX FIFO. Then, the previously stated gen function is called, so that the message exchange workflow is carried out.

Accordingly to the figure 4.24, the dispatch function checks whether there are messages (that contain commands) stored in the RX FIFO. In case this condition is valid, the member of an object specified in the message needs to be executed. In order to achieve that, a callback function associated to this particular object is called by means of a function pointer array, being the index of the array, that is, the actual pointer of the callback function to be called specified in the Object field of the received message. Having entered in that function, a the same procedure is used as for calling the specified method of the aforementioned object. Having completed the execution of that command, the RX FIFO is updated, by popping the message in the head of the queue, so that the next message awaiting to be

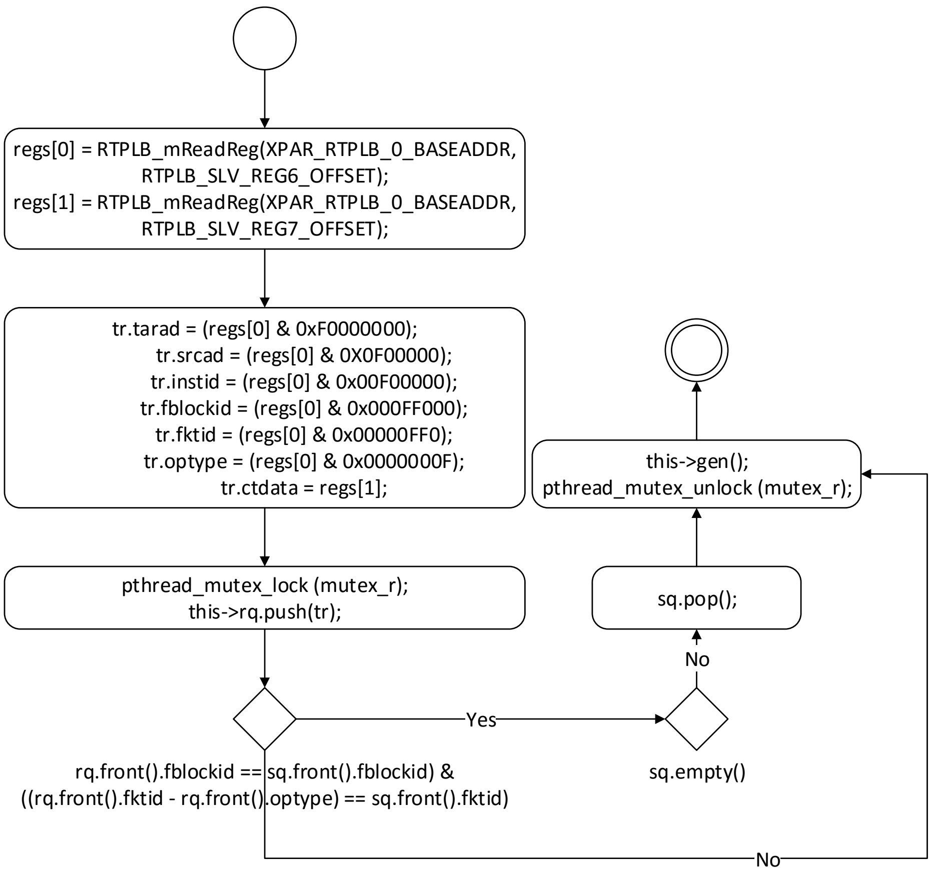
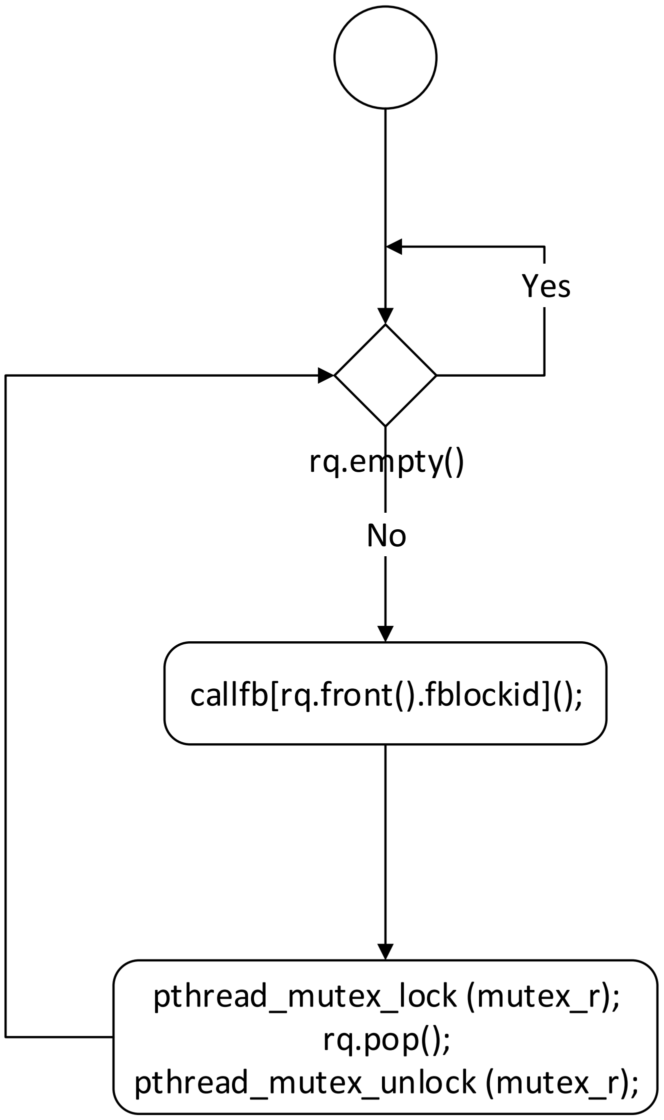

regs[0] = RTPLB_mReadReg(XPAR_RTPLB_0_BASEADDR, RTPLB_SLV_REG6_OFFSET);
regs[1] = RTPLB_mReadReg(XPAR_RTPLB_0_BASEADDR, RTPLB_SLV_REG7_OFFSET);
tr.tarad = (regs[0] & 0xF0000000);
tr.srcad = (regs[0] & 0X0F00000);
tr.instid = (regs[0] & 0x00F00000);
tr.fblockid = (regs[0] & 0x000FF000);
tr.fktid = (regs[0] & 0x00000FF0);
tr.optype = (regs[0] & 0x0000000F);
tr.ctdata = regs[1];
pthread_mutex_lock (mutex_r);
this->rq.push(tr);
rq.front().fblockid == sq.front().fblockid) &
((rq.front().fktid - rq.front().optype) == sq.front().fktid)
Yes
sq.empty()
No
sq.pop();
this->gen();
pthread_mutex_unlock (mutex_r);
No

Figure 4.23: Reading inbound data, via PLB

executed (if it exists), is also processed.



Yes
rq.empty()
No
callfb[rq.front().fblockid]();
pthread_mutex_lock (mutex_r);
rq.pop();
pthread_mutex_unlock (mutex_r);

Figure 4.24: Executing commands based on received control data

The advantage of employing the combination of callback functions and its pointers, over the usage of a switch case condition, is that for a large number of available objects and methods to be called, the time spent to call an object and then a method via the implemented solution is the same, no matter the index of the object and method to be executed. With the switch case methodology, the system would be required to successively check, one by one, the ID of the object and method to be called, until it matched the ID included in the message. This could take a variable amount of time (which would be not less than the combined callback function and function pointer methodology), depending on their ID. If it was 0, it would take only one iteration to get to the desired object and another one to call the method. However, if the ID was n, it would take 2n iterations for the desired method of an object to be called. The longer time to get commands to begin its execution would be superior which, as a consequence, could get the RX FIFO to reach its maximum capacity, in case long bursts of messages were to be sent to a node and the number of received message within a time frame would outperform the capacity for the node to process them. The switch case based solution would also induce indeterminism in this part of the system, although the execution time of this part of the system does not affect the timings at which messages are received and sent out to the network, as a result of the overall well-designed system.

As far as transmission acknowledgement is concerned, two events require the retransmission of a segment. First, a segment may be damaged in transit but nevertheless arrive at its destination. If a checksum is included with the segment, the receiving transport entity can detect the error and discard the segment. The second contingency is that a segment fails to arrive. In either case, the sending transport entity does not know that the segment transmission was unsuccessful. To cover this contingency, a positive acknowledgement scheme is used. In our case, messages may not require an acknowledgement, and thus the message is removed from the outbound message FIFO as soon as the flag that signals the frame has successfully been sent out to the network. Additionally, in case a message requires an acknowledgement, two types are available: a simple acknowledgement and one coupled with a return value, which can be regarded as a parameter containing relevant information to the execution of a chain of commands that depend on the return value of the previously executed action. In order to distinguish between the three aforementioned possibilities, a field in the frame is reserved to accommodate the following values:

- 0, for messages which require no acknowledgement to be sent back
- 1, for messages which require a simple acknowledgement to be sent back
- 2, for messages which require an acknowledgement and a return value to be sent back to the sending transport entity

If a segment does not arrive successfully, no acknowledgement will be issued and a retransmission is in order. To cope with this situation, there must be a timer associated with each segment as it is sent. If the timer expires before the segment is acknowledged, the sender must retransmit. So the addition of a timer solves that problem. Next problem: At what value should the timer be set? Two strategies suggest themselves. A fixed timer value could be used, based on an understanding of the network's typical behaviour. This suffers from an inability to respond to changing network conditions. If the value is too small, there will be many unnecessary retransmissions, wasting network capacity. If the value is too large, the protocol will be sluggish in responding to a lost segment. The timer should be set at a value a bit longer than the round trip time (send segment, receive ACK). Of course, this delay is variable even under constant network load. Worse, the statistics of the delay will vary with changing network conditions. An adaptive scheme has its own problems. Suppose that the transport entity keeps track of the time taken to acknowledge data segments and sets its retransmission timer based on the average of the observed delays.This value cannot be trusted for three reasons:

- The peer transport entity may not acknowledge a segment immediately. Recall that we gave it the privilege of cumulative acknowledgements.
- If a segment has been retransmitted, the sender cannot know whether the received acknowledgement is a response to the initial transmission or the retransmission.
- Network conditions may change suddenly.

Each of these problems is a cause for some further tweaking of the transport algorithm, but the problem admits of no complete solution. There will always be some uncertainty concerning the best value for the retransmission timer.

4.3.2 Media Streaming Control Service

The Media Streaming Control Service acts as an intermediary between the hardware implemented part and the software application, being in charge of the Data routing, according to the instructions issued by the Streaming Session Management Unit. The use case diagram found in the figure 4.25 represents a list of action or event steps, typically defining the interactions between a role and a system, to achieve a goal. This service performs enabling/disabling and allocation/deallocation of channels, that is, segments of the data frame which belong

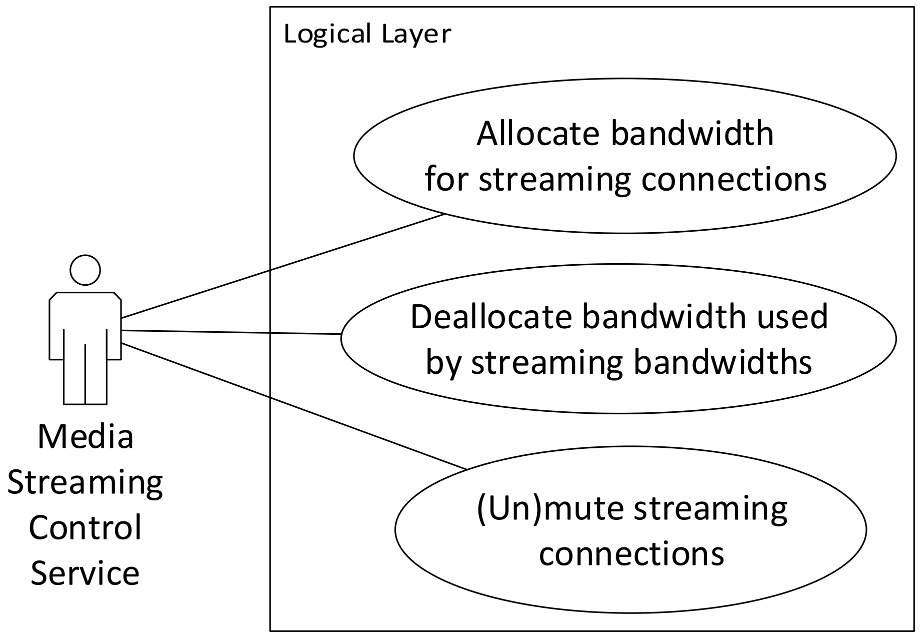

Logical Layer
Allocate bandwidth for streaming connections
Deallocate bandwidth used by streaming bandwidths
(Un)mute streaming connections
Media Streaming Control Service

Figure 4.25: Media Streaming Control Service interaction with Logical Layer

to the synchronous component of the system, by providing an abstraction layer in write operations to registers via the Processor Local Bus. The actual datapath through which synchronous data flows, has its mode of operation controlled by the values written to these registers, according to the desired effect.

Depending on the usage of channels by other sink-source sets at a particular moment, data exchange by a particular set of sink-source devices can take place using, for example, the channel 0 at one time, and the channel 1 or 2 or 3 (...) or n (where n is the maximum number of channels supported by the system) at another time. This means the node must be able to internally pick up any of the channel(s) and perform a write or read operation on the corresponding one(s) (depending whether it is a source or sink device, respectively). To do so, the datapath of the node must have a configurable routing mechanism, to enable the aforementioned operations to be carried out.

Streaming connections are established between the devices as needed and usually stay active as required. Nevertheless, there are perturbations that might lead to disturbances in the network and to unwanted audible or visible effects. The built-in intelligent muting of the node allows to automatically mute sink socket connections to prevent the application from receiving corrupted streaming data. Intelligent muting is triggered when the system enters in Configuration Mode.

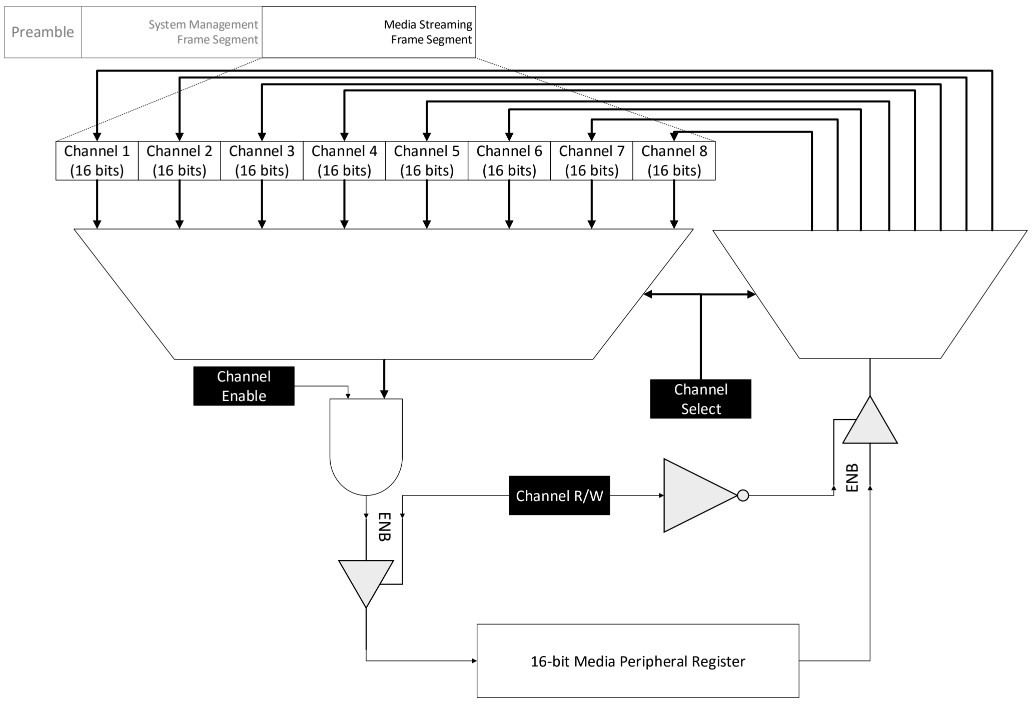


Figure 4.26: Dynamic selection of channels for read/write operations

This is due to the detection of a malfunctioning device that left the network and thus does not respond to messages directed towards it or due to a defective part in the transmission path that prevents the data frame from travelling for one end to the other of the network and back to the origin.

4.3.3 Node Synchronization Service

The Node Synchronization Service integrates the essential functionalities to ensure the Periodic time-sensitive data exchange, as required by the system's synchronous component. The use case diagram illustrated in the figure 4.27 represents a list of action or event steps, typically defining the interactions between a role and a system, to achieve a goal.

The Node Synchronization consists essentially on a timer, that runs on the Prime Node only and triggers the transmission of data to the network in a well-defined period, which in our case is imposed by the timing requirements of the synchronous component of the system. Provided that the recommended sampling frequency of an audio signal is of 48 kHz, the period at which the data frame has to start its cycle around the network is of 20.8 micro seconds. The CPU application is in charge

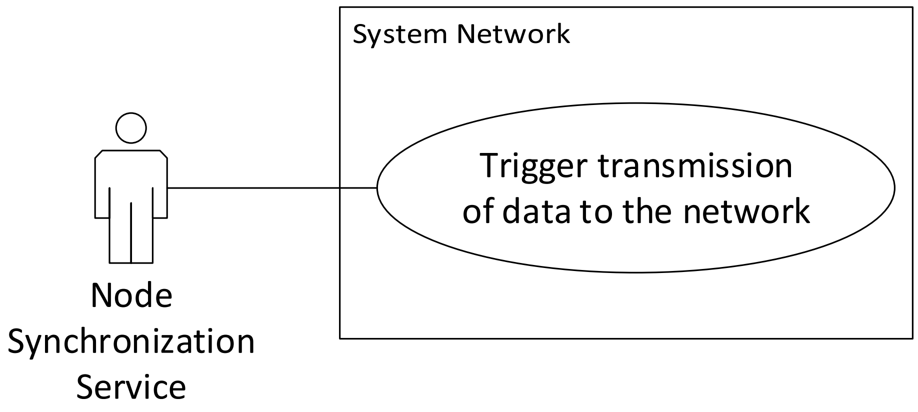

System Network
Trigger transmission of data to the network
Node Synchronization Service

Figure 4.27: Node Synchronization Service for ensuring periodic data transfer in the network

of enabling or disabling Node Synchronization Service on-the-fly, depending on whether the node takes the role of a Prime Node or a Regular Node, respectively.

As this is a point to multi-point data flow system (i.e., the streaming data has a source and any desired number of sinks), all devices share a common system clock pulse derived from the data stream. They are thus in phase with each other and can transmit all data synchronously, which makes any mechanisms for signal buffering and signal processing redundant. The system clock is generated by the embedded Node Synchronization Service, which is integrated in one of the nodes (the Primary Node). In a vehicle, it is usually located in the head unit of the infotainment system. All other nodes are synchronized onto this system clock pulse by means by means of the detection of the preamble. As soon as the sequence of bits 1010 is detected (that is, the preamble), the secondary node configures itself in order to capture the next bit of the data frame at its next clock transition, positive or negative edge, depending on whether when the preamble was detected, the node's main clock level was high or low.

When the Primary Node receives the frame again when it has travelled around the ring, it reclaims the signals by the same method and subsequently generates the next frame.

Table 4.9: My caption

Register Address	Register Index	Description
4	31:0	Network-to-application data
5	31:0	Network-to-application data
6	31:0	Application-to-network data
7	31:0	Application-to-network data
8	0	Send data to the network

4.3.4 Interface with IP Core

Prior to triggering the chain of periodically sent data frame across the network, the Node Control Unit must assert that the current node acts as a Prime Node and that the Application has started and is ready to exchange messages. Having ensured these conditions, the Node Synchronization Service periodically send a pulse via the "Send Frame" net, therefore releasing the datagram to the network, as detailed in the block diagram 4.28.

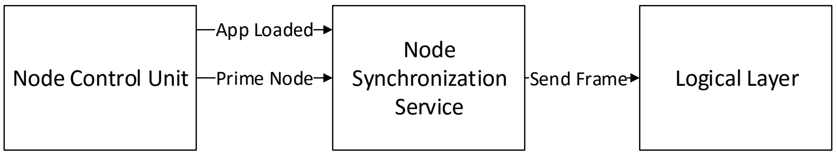


Figure 4.28: Network communication protocol - Logical Layer

The interface between the SW part of the Application Message Exchange Service and the HW implemented design, relies on the Processor Local Bus, used to read and write from and to registers, respectively. Besides the flag that is used to trigger the the sending process of the data to the network, the remaining registers comprised in the table 4.9 are data structures that hold the actual content of control messages, that is, the System Management Frame Segment.

In order to control the read/write operation on the allocated channels, the Media Streaming Control Service makes use of registers that are manipulated by the application by means of the Processor Local Bus, that are described in the table 4.10. To each bit of the each register, corresponds a channel, from 7 to 0, which matches the each of Register's Index.

Table 4.10: Registers for controlling the datapath for time-sensitive data

Register Address	Register Index	Description
0	7:0	Synchronous channel allocation
1	7:0	R/W Operation on selected channel
2	7:0	Enable/Disable Write Operation on channel

4.4 Logical Layer

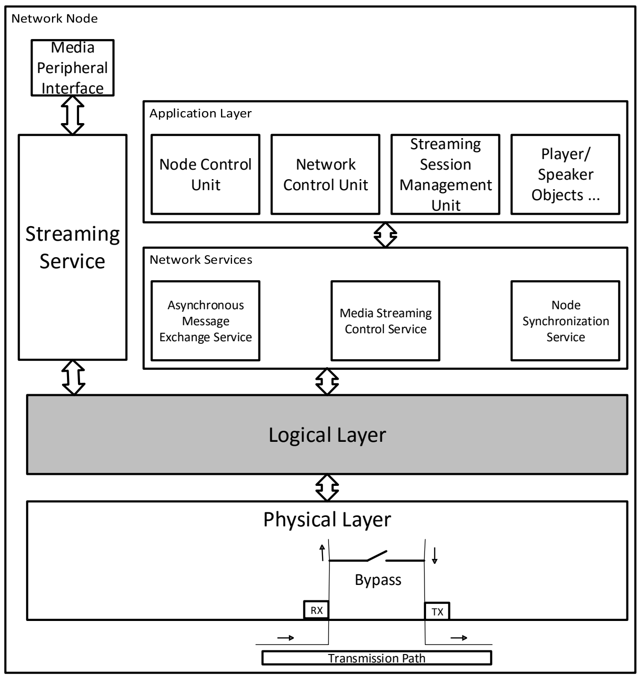

Network Node
Media Peripheral Interface
Application Layer
Node Control Unit
Network Control Unit
Streaming Session Management Unit
Player/ Speaker Objects ...
Streaming Service
Network Services
Asynchronous Message Exchange Service
Media Streaming Control Service
Node Synchronization Service
Logical Layer
Physical Layer
Bypass
RX
TX
Transmission Path

Figure 4.29: Network communication protocol - Logical Layer

The Logical Layer defines the data format this interconnected system adopted, where details about how nodes connects/disconnects from the network, as well as the methodology to synchronize it with the incoming data are exposed. After that, information concerning the node addressing scheme and error handling procedures is provided. The use case diagram included in the figure 4.30 represents a list of action or event steps, typically defining the interactions between a role (known in the Unified Modelling Language as an actor) and a system, to achieve a goal.

The following dataflow diagram, included in the figure figure 4.31, comprises the various steps involved in the process of receiving and transmitting data and how it is processed depending on whether it is asynchronous or synchronous data. It can

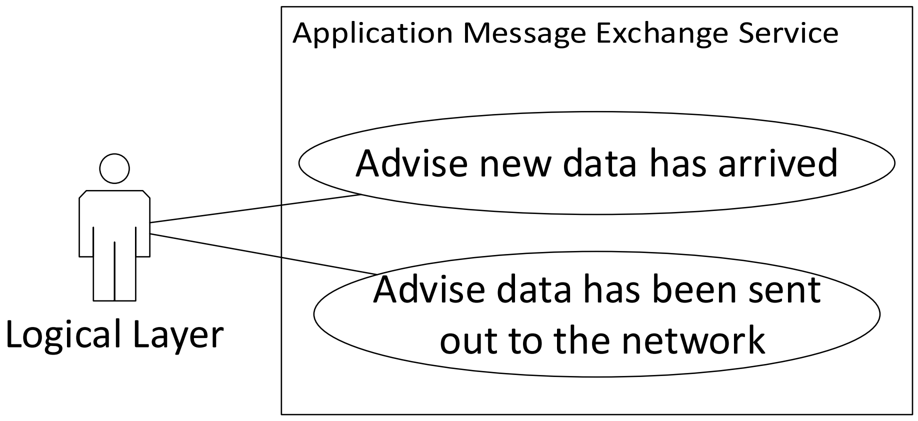

Application Message Exchange Service
Advise new data has arrived
Advise data has been sent out to the network
Logical Layer

Figure 4.30: Logical Layer interaction with Application Message Exchange Service

be noted that synchronous data does not go through all the layers off the network protocol, whereas asynchronous information does. Instead, it is sent directly to the media peripheral, as information of this nature has tighter timing requirements.

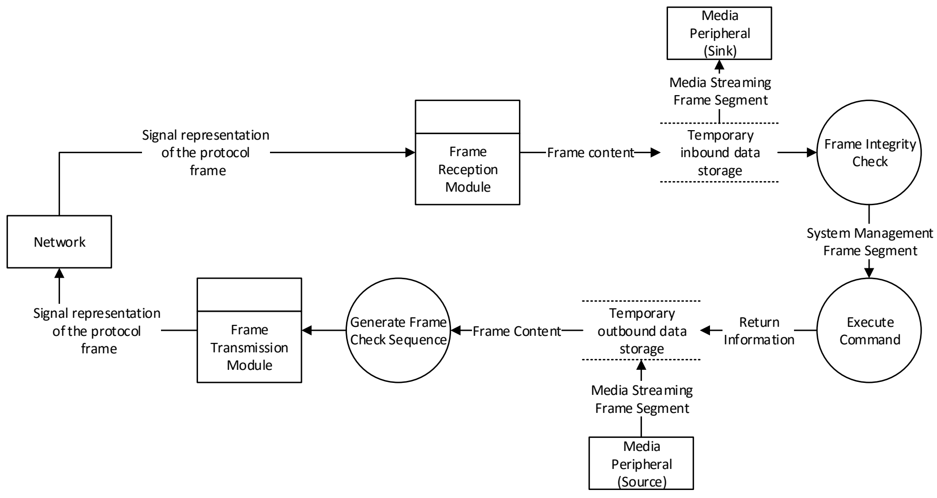
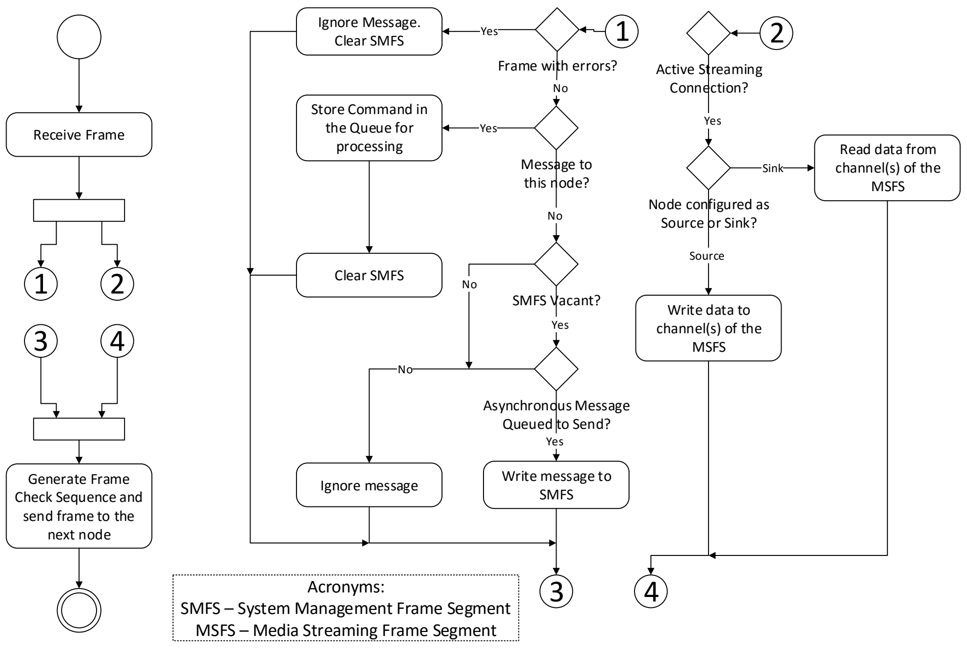

Media Peripheral (Sink)
Media Streaming Frame Segment
Signal representation of the protocol frame
Frame Reception Module
Frame content
Temporary inbound data storage
Frame Integrity Check
Network
System Management Frame Segment
Signal representation of the protocol frame
Frame Transmission Module
Generate Frame Check Sequence
Frame Content
Temporary outbound data storage
Return Information
Execute Command
Media Streaming Frame Segment
Media Peripheral (Source)

Figure 4.31: General overview of dataflow inside the network node

To complement the previously stated information, the flowchart represented in the figure 4.32 describes that asynchronous messages are sent out to the network if the corresponding frame segment is vacant, otherwise overwriting data may and would cause messages to fail to reach the destination, specially for those which do not require an acknowledgement to be sent back to the sender node, rendering the transmission of a repeated message not to be sent towards the destination.

To control the mode of operation of the node accordingly to the information dis-



Receive Frame
1
2
3
4
Generate Frame Check Sequence and send frame to the next node
Ignore Message. Clear SMFS
Yes
Frame with errors?
No
Store Command in the Queue for processing
Yes
Message to this node?
No
Clear SMFS
No
SMFS Vacant?
Yes
No
Asynchronous Message Queued to Send?
Yes
Ignore message
Write message to SMFS
Active Streaming Connection?
Yes
Sink
Read data from channel(s) of the MSFS
Node configured as Source or Sink?
Source
Write data to channel(s) of the MSFS
Acronyms:
SMFS – System Management Frame Segment
MSFS – Media Streaming Frame Segment

Figure 4.32: Data treatment by the Network Node, in detail

closed in the previous diagrams, the following state machine, graphically represented in the figure 4.33, was implemented in order to start receiving the data after the frame start delimiter (also known as preamble) is detected, followed by the validation of the received data, after ensuring it is not corrupt, by running a the CRC-32 error detection algorithm. After that, it is shown how to proceed in order to send the information back to the network, after having been processed by the CPU.

The table 4.11 describes the meaning of each state of the state machine.

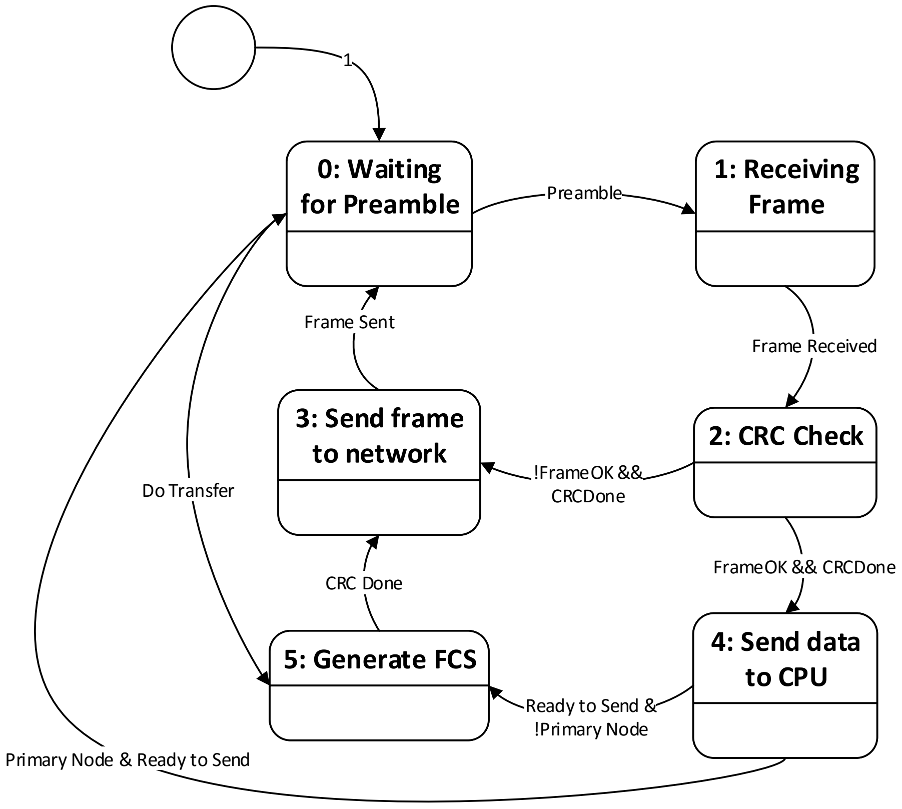


Figure 4.33: Finite State Machine for controlling data exchange and processing

Table 4.11: Finite State Machine states description

State Designation	**State Description**
0: Waiting for Preamble	Detection of the frame start delimiter
1: Receiving Frame	Stores the content of the frame, bit by bit, using a deserializer
2: CRC Check	Ensure received data is not corrupt
4: Send Data to CPU	Transmit data to CPU for processing, if it is directed to the node in question. Otherwise pass it on to the next node
5: Generate FCS	Determine Frame Check Sequence for future frame integrity validation
3: Send frame to network	Transmit data to network via a serializer

To complement the information present on the table 4.11, the table 4.12 contains information regarding the meaning the triggers that cause the machine, illustrated in the figure 4.33, to change its state.

Table 4.12: List of Finite State Machine conditions for triggering state change

Condition Designation	**Condition Description**
Preamble	The start delimiter has been detected
Do Transfer	Prime Node sends the first frame after system power-up
Send Frame	Node Synchronization Service triggers the transmission of the frame
Frame Received	All bits of the frame have been captured
FrameOK	The received data is valid (no corruption)
CRCDone	Frame integrity check is complete
/FrameOK	Received data is corrupt
Ready to Send	Data has been loaded into/from the CPU
/Prime Node	The node has the role of Regular Node
CRCDone	FCS generation is complete
Prime Node	The node has the role of Prime Node
Frame Sent	All data has been transmitted to the network

4.4.1 Signal Modulation and Node synchronism

As the interconnected devices exchange bits of the data frame one by one, with resort to a serializer. The period of the internal clock marks the interval of time at which each bit of the data frame departs from the source device. The reverse process, by means of a de-serializer, is carried on when it comes to receive each bit of the data frame.

As the system is composed by independent devices, this means phase shifts may be observed between the main clocks of each device. Provided that there is not a dedicated line to carry a clock signal which all nodes are synchronized with, another method has to be employed so that the datagrams are received, regardless of the aforementioned phase shifts.

One solution consists of sending a preamble prior to the actual information included in the data frame. The preamble consists of a set of alternating bits that signalizes the arrival of a new frame, so that the addressee device can calibrate itself in order to receive the sequence of bits flawlessly. In this case, the preamble is a set of 4 bits with the hexadecimal value of 0xAA, or 1010 in a binary base. Firstly, the node detects a rising edge transition, as the first bit, with the logic value of 1, of the frame start delimiter arrives, as can be seen in the figure 4.34.

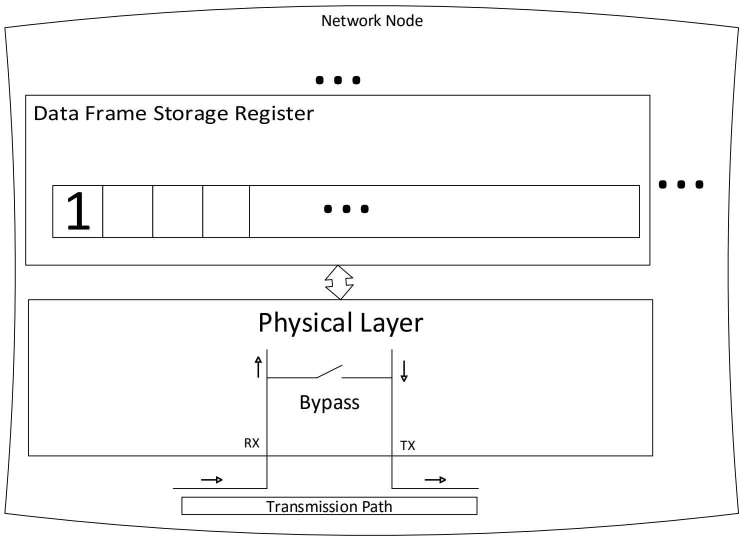


Figure 4.34: First bit of the frame start delimiter received

After that, on the next clock cycle the second bit, with the logic value of 0, presents itself at the RX Port of the node (figure 4.35).

Then, on the subsequent clock cycle, the bit with logic value of 1 arrives at the RX Port (figure 4.36).

Finally, a falling edge transition is detected at the RX Port of the node, as a result of the arrival of the 4th bit of the frame start delimiter (figure 4.36). By this time, the level of the device's internal clock is checked. If its level is HIGH, the following bits of the data frame will be captured at each negative transition of the

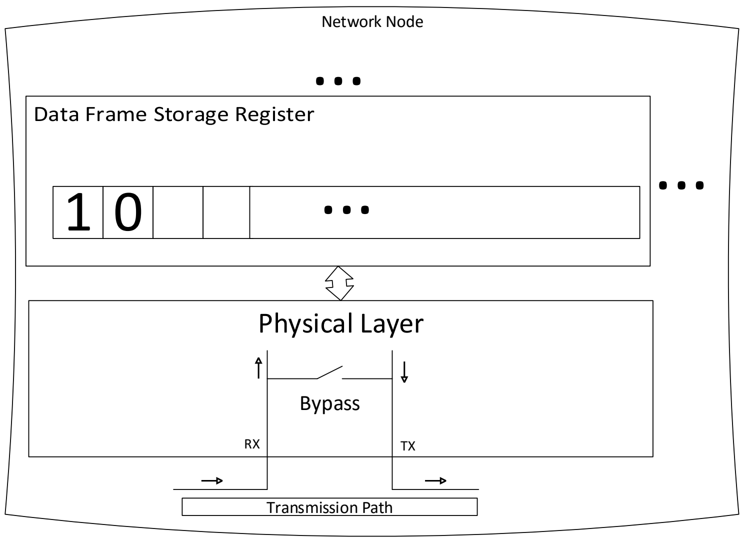


Figure 4.35: Second bit of the frame start delimiter received

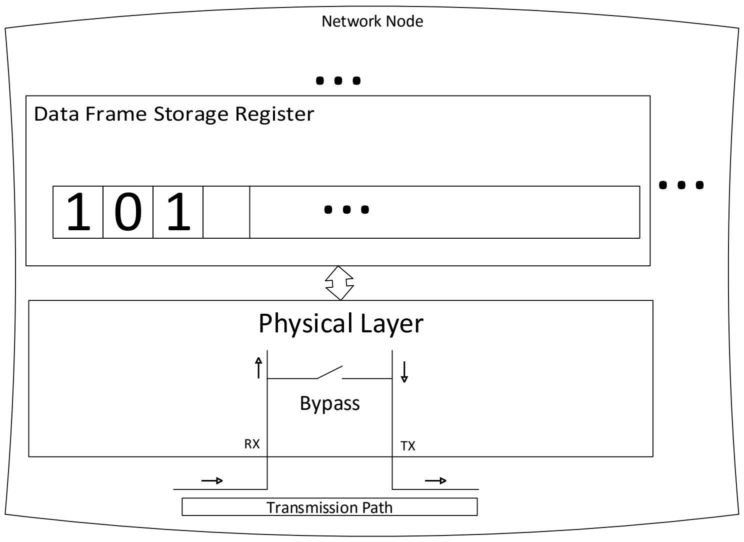


Figure 4.36: Third bit of the frame start delimiter received

device's internal clock. Otherwise, they are captured at each positive transition of the device's internal clock. This ensures that each bit is captured within a phase shift of up to 180 degrees from the exact time each bit of the data frame arrived at the RX port of the device.

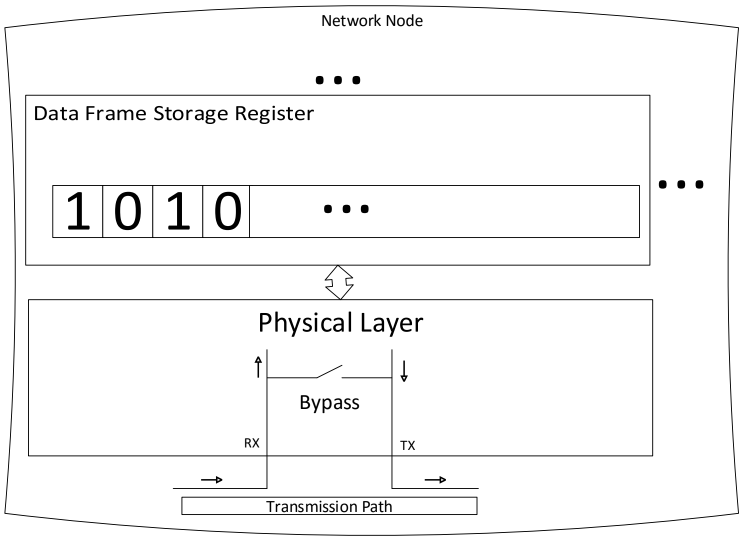


Figure 4.37: Fourth bit of the frame start delimiter received

4.4.2 Error handling

Data corruption may occur during the process of data transmission between interconnected devices, rendering such information unable to be processed. So, error-detection algorithms, such as Cyclic Redundancy Check (CRC), have been developed in order to determine if a received datagram is consistent with the one sent by the source device. Besides the datagram which contains data to be effectively processed by a node, a set of bits is included in the frame, which is known as the Frame Check Sequence. This contains the result of the error-detection algorithm executed in the source device, prior to the transmission of the frame. After the data reaches the destination, the same algorithm is executed in order to validate the integrity of the received data.

As far as the documented project is concerned, the CRC-32 variant will be used. It requires a 33-bit Frame Check Sequence to be appended to the datagram, forming the actual data frame. In terms of computation of this value for big endian datagram, it is performed as follows:

- The datagram, which can be called the quotient is appended a set of 32 Least Significant Bits with the value of zero, which will hold the value of the remainder.
- The 33 Most Significant Bits of the datagram are selected and a XOR operation with the CRC-32 Polynomial is performed if the Most Significant Bit has the value of one, followed by a shift of the datagram to the left. If its value is zero, the datagram is shifted to the left instead of a XOR operation with CRC-32 Polynomial taking place.
- This process is successively repeated until the value of the quotient is zero. The value of the remainder is included in the Frame Check Segment of the frame to be sent to another device.

After the frame is received by the addressee device, the same process is held. Upon its completion, the frame is considered as valid if the remainder equals zero.

4.4.3 Datagram structure

Packet switching is a digital networking communications method that groups all transmitted data into suitably sized blocks, called packets, that are transmitted via a medium that may be shared by multiple simultaneous communication sessions. Packet switching increases network efficiency, robustness and enables technological convergence of many applications operating on the same network.

Packets are composed of a header and payload. Information in the header is used by networking hardware to direct the packet to its destination where the payload is extracted and used by application software. Speaking of the asynchronous component, the essential information the header has to contain is the following:

- Addressee Node: node's address to which the message is directed to
- Sender Node: node's address from which the message origins from

In the payload segment, that holds the effective data is to control the system's functioning, the

- Instance ID: ID of the node's module to be accessed
- Object ID: ID of the commands container to be accessed

- Action ID: ID of the command to be executed
- Action Type: specify whether a ACK or a ACK + data has to be returned to the Sender Node
- Parameter Field: parameters required for the execution of certain commands
- Frame Check Sequence: set of bits supplied to the error detection algorithm, such as CRC-32, to determine whether the received data is valid or the frame is corrupt. This stage has to be performed before the data is parsed by the node, thus preventing incorrect data to be processed and consequently misoperation of the device and possibly of the system.

A graphical representation of on the composition of the Control Data Segment of the datagram can be seen in the figure 4.38

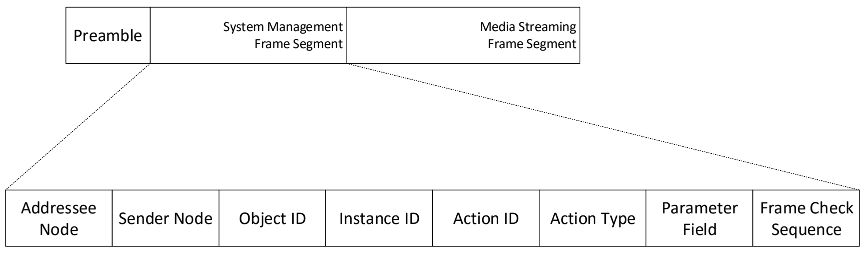


Figure 4.38: Control data elements organization in the datagram infrastructure

In order to support the simultaneous transfer of multiple audio signals, the synchronous segment can be sub-divided into a number of channels. Depending on the bandwidth an audio signal occupies, one or more channels can be reserved for that particular signal. That range of channels shall be exclusively performed a write operation by one source device and a read operation by one sink device. If another audio signal has to be simultaneously transmitted by an independent set of source-sink devices, an array of 1 to n still unused channels (depending on the signal's bandwidth) is allocated, as long as there are enough available channels.

4.5 Stream Service

The time-sensitive data follows a different path than the data meant to control/-manage the mode of operation of the system, which would add up overhead as far as transmitting data inside the node is concerned. The Stream Service acts as

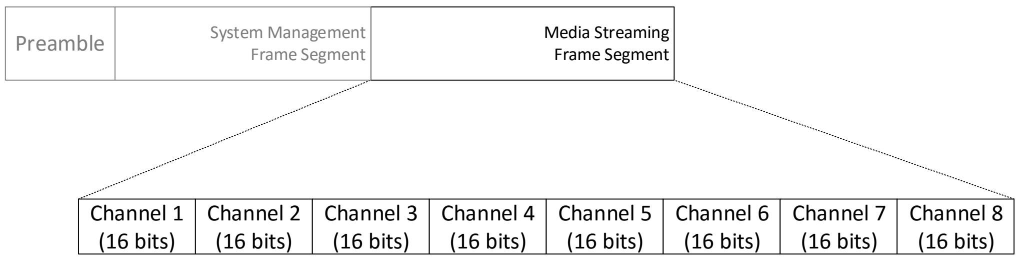


Figure 4.39: Time-sensitive data transport structure

an interface between the Logical Layer and the media peripheral, being comprised of an exclusive path through which data contained in the synchronous channels is selected, based on the channel allocation and R/W configuration (carried ou by the Streaming Session Management Unit) and then exchanged with the media peripheral that is, the external IP Core.

4.5.1 Interface with external IP Cores

With regards to the specification of the Stream Service, a bi-directional port, with a width of 16 bits, the resolution of each synchronous channel, as defined in the datagram structure, is available to be connected to a media peripheral, such as the AC97 Codec, as represented in the diagram 4.40.

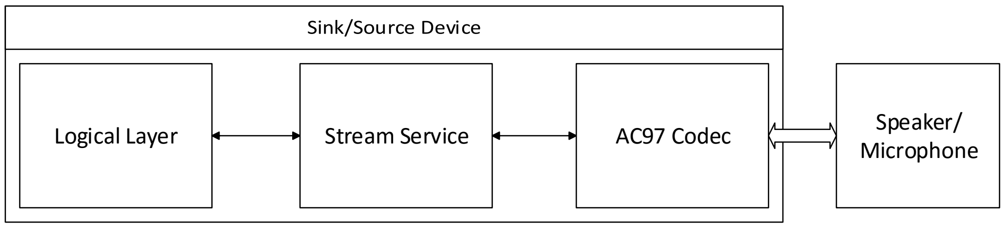


Figure 4.40: IP-to-IP Core interface, for time-sensitive data transport

Chapter 5

Case Study

The node requires additional peripherals besides the CPU in order to do its job. The Network Interface Controller consists on the custom IP that has been developed, implements the Physical and Logical layer as well as part of the services. The Node Synchronization Services, Asynchronous Message Exchange Service and the Media Streaming Control Service define the border between the hardware and software implemented parts. To enable both parts to communicate, a shared bus interconnects the CPU, the Random Access Memory as well as all of the required peripherals, such as the previously stated custom IP and the interrupt controller. The interrupt controller is intended to trigger the execution of routines immediately after the a rising edge transition on the signals rp and ft. These respectively advise that a new datagram has arrived to the node, who is the addressee of the message, so that it stores the data to be processed later on; and that the data has successfully been sent out to the network. In case such data does not require an acknowledgement to be sent back, it is deleted from the TX queue, as its delivery has been deemed complete. On the software application side, part of the aforementioned services and the units supposed to manage the system' mode of operation are executed in a Microblaze CPU. The node architecture can be seen in the figure 5.1

In order to put the unit under test, it has been instantiated on the same FPGA chip as two other independent devices. These implement solely the Physical and Logical Layer and are just responsible for forwarding the message to the next node, which results in a virtual loop back of the data transmitted by the unit under test, moments before. The entities involved in the integration tests are interconnected by means of a Ring topology, to emulate a real case scenario, just as the system

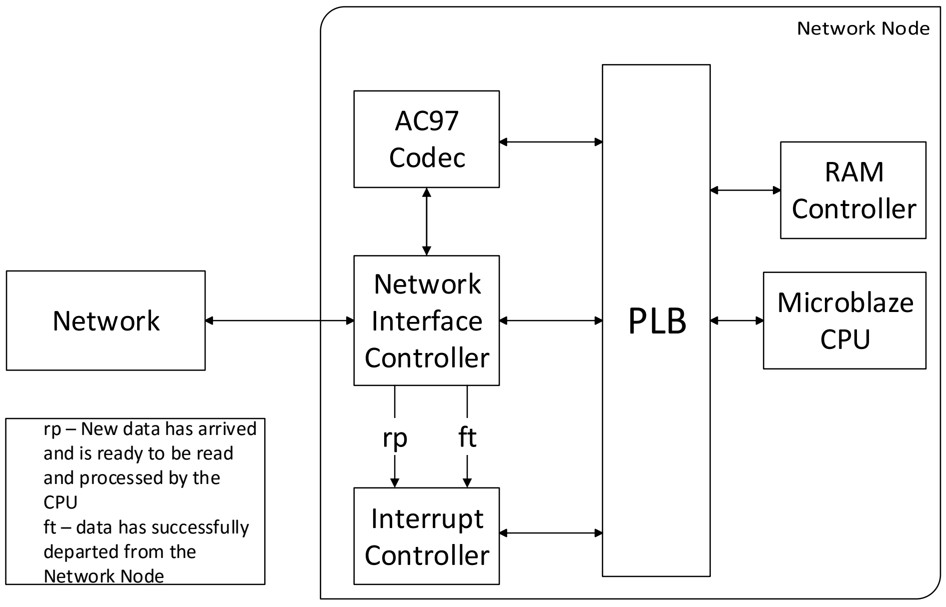

Network Node
AC97 Codec
RAM Controller
Network
Network Interface Controller
PLB
Microblaze CPU
rp – New data has arrived and is ready to be read and processed by the CPU
ft – data has successfully departed from the Network Node
rp
ft
Interrupt Controller

Figure 5.1: Implemented Network Node Internals

has been designed to.

A graphical representation of the distributed system can be seen in the figure 5.2.

The following registers, contained in the table 5.1, were assigned a value so that the desired output, described in the next section, can be achieved.

In order to provide a better view of what is written in the last two field of the table 5.1, the table 5.2 provides details regarding the datagram structure and the table 5.3, contains the values that are written in the datagram and that correspond to the last two items of the table 5.1.

So as to provide a considerable margin for further development stages of the system, it supports:

- Up to 16 nodes attached to the network, as 4 bits hold the address value of each node
- Up to 256 instances of sub-modules per node, provided that 8 bits hold the value of the Instance ID
- Up to 256 objects per instance, provided that 8 bits hold the value of the Object ID

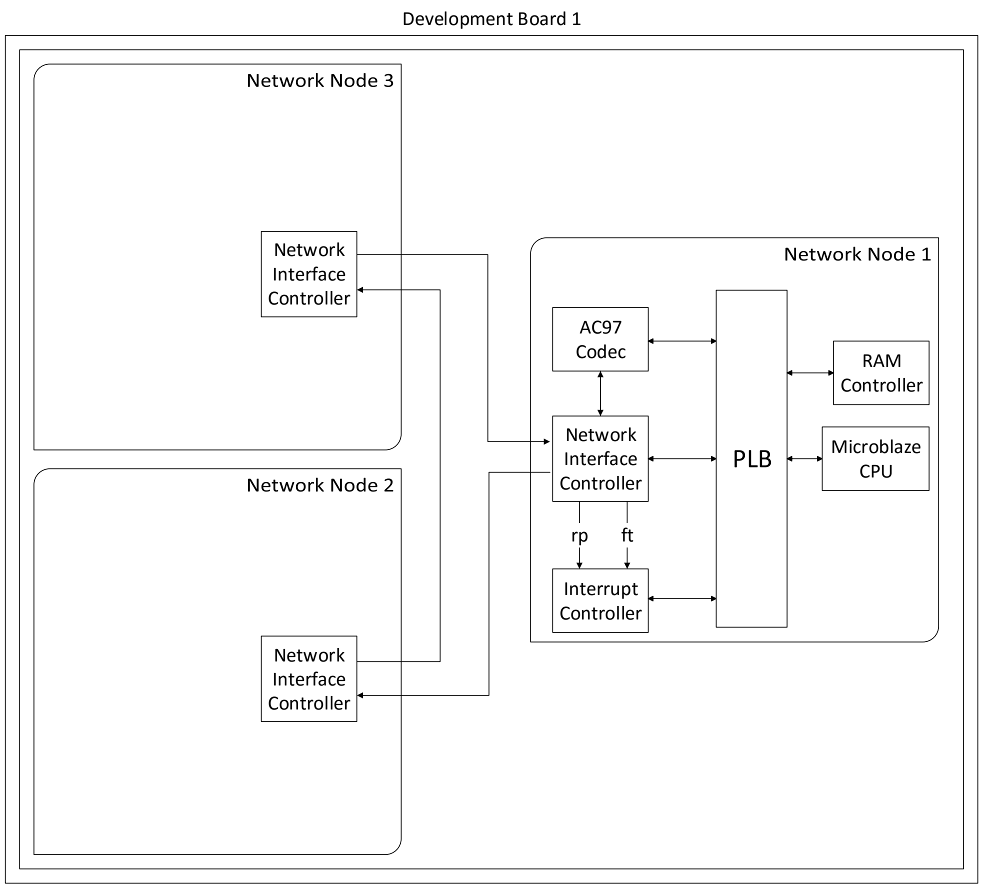


Figure 5.2: Experimental Node interconnection in a Ring topology

Table 5.1: Register values for the test runs

Register Address	Register Index	Description	Value
3	3:0	Address of the node itself	4'h1
	4	Node acting as Prime or Regular Node	1'b1
	5	Application is loaded (node can enter the network)	1'b1
	6	Network is in configuration mode	1'b0
	7	Flag to shutdown the system	1'b0
	31:0	Application-to-network data	*
7	31:0	Application-to-network data	*

Table 5.2: Structure of the datagram

Index Range	Description
255:252	Frame Start Delimiter
251:250	Not Used
249	System Shutdown Flag
248	System in Configuration Mode flag
247:179	System Management Frame Segment
178:160	Not Used
159:32	Media Streaming Frame Segment
31:0	Frame Check Sequence

Table 5.3: Structure and content of the System Management Frame Segment

Index Range	Description	Value
247:244	Addressee Node	1
243:240	Sender Node	1
239:232	Instance ID	0
231:224	Object ID	0
223:214	Action ID	0
213:212	Action Type	0
211:179	Parameter Field	0

- Up to 1024 function per Object, given that 10 bits hold the value of the Action ID

The platform includes a minimal operating system (called Xilkernel), that supports POSIX threads, thread synchronization and Inter Process Communication mechanisms, and interrupt handling. This provides an abstraction layer to deal with interrupts, due to the way the system was designed to. Moreover, to cope with increasingly complexity of the application, as more features are added, the thread support allows the application to be divided into various, simpler tasks, which makes it possible for the system to be scalable, without increasing the demand for the developer to adapt the whole application, as would be the case if the system was to be developed with a sequential based coding. The board support package then provides the necessary means (such as drivers) for the kernel to communicate with the hardware, as detailed in the diagram 5.3

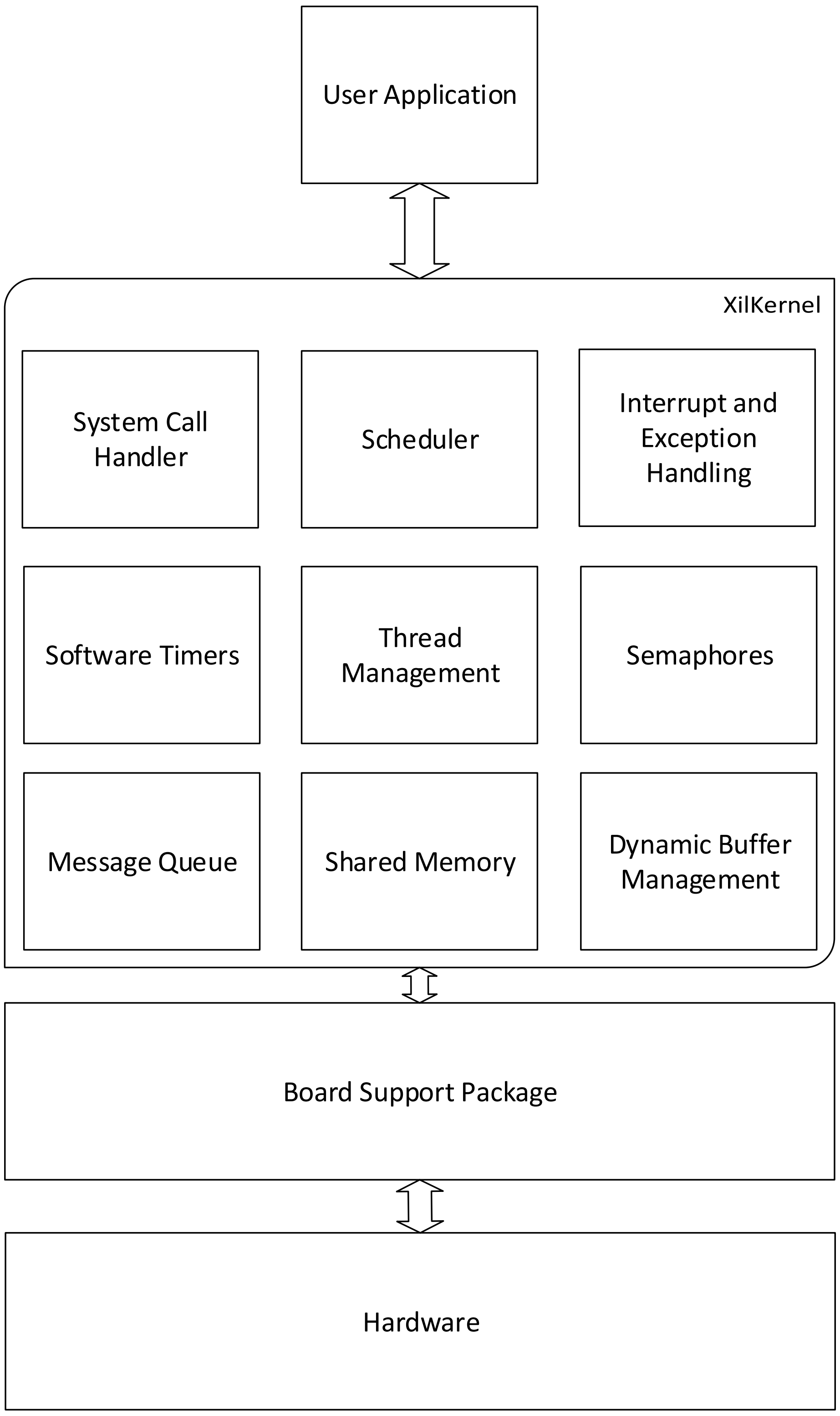


Figure 5.3: System Software Stack

5.1 Results

Firstly, the platform has to be initialized as well as the kernel. The source code excerpt 5.1 shows how such process is carried out.

Meanwhile, a static thread is also launched, which calls a callback function, called init() which is responsible for actually enabling the interrupts and initializing the interrupt handler's pointers, as well as launching a thread that is meant to process the commands contained in the datagram that arrive at the node.

Listing 5.1: Platform and kernel initilization

```
int main()
{
    init_platform();

    /* Initialize xilkernel */
    xilkernel_init();

    /* add a thread to be launched once xilkernel starts */
    xmk_add_static_thread(init, 0);

    /* start xilkernel - does not return control */
    xilkernel_start();

    /* Never reached */
    cleanup_platform();

    return 0;
}
```

The code excerpts 5.2 and 5.3, respectively show implementation details of both interrupt and thread setup.

Listing 5.2: Interrupt Setup

```
void intsetup()
{
        XIntc_RegisterHandler(XPAR_XPS_INTC_0_BASEADDR,
           XPAR_XPS_INTC_0_RTPLB_0_RP_INTR,
                                        (XInterruptHandler)
                                           InboundMsgHandler, (
                                           void*) 0);

        XIntc_RegisterHandler(XPAR_XPS_INTC_0_BASEADDR,
           XPAR_XPS_INTC_0_RTPLB_0_FT_INTR,
                                        (XInterruptHandler)
                                           OutboundMsgHandler, (
                                           void*) 0);
        XIntc_EnableIntr(XPAR_XPS_INTC_0_BASEADDR,
           XPAR_RTPLB_0_FT_MASK | XPAR_RTPLB_0_RP_MASK);

        XIntc_MasterEnable(XPAR_XPS_INTC_0_BASEADDR);

        microblaze_enable_interrupts();

}
```

Listing 5.3: Application thread setup

```
void *init(void *arg)
{
    pthread_t worker[2];
    pthread_attr_t attr;
    int ret, *result, args;
    pthread_attr_init (&attr);
    pthread_attr_setdetachstate(&attr,
       PTHREAD_CREATE_JOINABLE);

    ret  = pthread_create(&worker[0], &attr, cmd_dispatcher
       , NULL);
    if (ret!= 0) {
              xil_printf ("Xilkernel␣Demo:␣ERROR␣launching␣
                 worker␣thread" \
                         ".\r\n");
         }
   intsetup();
   ret = pthread_join(worker[0], (void**)result);
return 0;
}
```

The command dispatcher implementation is also showcased in the code excerpt 5.4. If there is an incoming message waiting to be processed, it checks whether that was an acknowledgement of a previously sent message. If so, such message does not need to be re-sent anymore, so it is popped from the TX queue. After that, the actual command contained in the message is executed, after completion of which, it is popped form the RX FIFO, so that the next message, if it exists, can be processed.

Listing 5.4: Command dispatcher implementation

```
void ams::dispatch(){
        int running = 1;
        Xuint32 conf = ((1 << 31) | (3 << 26));

        RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR,
          RTPLB_SLV_REG3_OFFSET, conf);
        while(running)
        {
                if(!rq.empty())
                {
                        pthread_mutex_lock (mutex_r);
                        if((rq.front().fblockid == sq.front
                          ().fblockid)
                                        & ((rq.front().
                                          fktid - rq.front
                                          ().optype) == sq
                                          .front().fktid))
                        {
                                if(!sq.empty())
                                {
                                sq.pop();
                                }
                        }
                        pthread_mutex_unlock (mutex_r);
                        callfb[rq.front().fblockid]();
                        pthread_mutex_lock (mutex_r);
                        rq.pop();
                        pthread_mutex_unlock (mutex_r);
                }
                else
                {
                        sleep(1);
                }

        }
}
```

Additionally, the code excerpts 5.5 and 5.6 provide details on the implementation of the Interrupt Service Routine for storing the incoming messages and the routine to generate a new message to be sent out to the network, respectively.

Listing 5.5: ISR for storing incoming messages

```
void ams::parse(){
        Xuint32 regs [NO_CTR_REG];
        regs[0] = RTPLB_mReadReg(XPAR_RTPLB_0_BASEADDR,
          RTPLB_SLV_REG6_OFFSET);
        regs[1] = RTPLB_mReadReg(XPAR_RTPLB_0_BASEADDR,
          RTPLB_SLV_REG7_OFFSET);
        tr.tarad = (regs[0] & 0xF0000000);
        tr.srcad = (regs[0] & 0X0F00000);
        tr.instid = (regs[0] & 0x00F00000);
        tr.fblockid = (regs[0] & 0x000FF000);
        tr.fktid = (regs[0] & 0x00000FF0);
        tr.optype = (regs[0] & 0x0000000F);
        tr.ctdata = regs[1];
        pthread_mutex_lock (mutex_r);
        this->rq.push(tr);
        if((rq.front().fblockid == sq.front().fblockid)
                                        & ((rq.front().
                                            fktid - rq.front
                                            ().optype) == sq
                                            .front().fktid))
        {
```

```
                        if(!sq.empty())
                        {
                        sq.pop();
                        }
            }
            this->gen();
            pthread_mutex_unlock (mutex_r);
}
```

Listing 5.6: Routine to generate new message to be transmitted

```
void ams::gen(){
            Xuint32 regs [NO_CTR_REG];
            if(!sq.empty()){
                        regs[0] = (this->sq.front().srcad & 0
                            X0F00000);
                        regs[0] |= (this->sq.front().tarad & 0
                            xF0000000);
                        regs[0] |= (this->sq.front().instid & 0
                            x00F00000);
                        regs[0] |= (this->sq.front().fblockid & 0
                            x000FF000);
                        regs[0] |= (this->sq.front().fktid & 0
                            x00000FF0);
                        regs[0] |= (this->sq.front().optype & 0
                            x0000000F);
                        regs[1] = this->sq.front().ctdata;
                        RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR,
                            RTPLB_SLV_REG6_OFFSET, regs[0]);
                        RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR,
                            RTPLB_SLV_REG7_OFFSET, regs[1]);
                        this->pass();
            }
            else{
                        RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR,
                            RTPLB_SLV_REG6_OFFSET, 0);
                        RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR,
                            RTPLB_SLV_REG7_OFFSET, 0);
```

```
                this->pass();
        }

}
```

Then, the Prime Node, aliased Network Node 1 in the figure 5.2, takes the initiative to send the first frame, as it implement has the Node Synchronization Service enabled, HDL implementation of which is exposed in the listing 5.7.

Listing 5.7: Node Synchronization Service RTL implementation

```
module tmaster(input clk, input rst, input mode, input
   enable, output dt);
        reg [12:0] count;
        assign dt = (count == 4015) ? 1 : 0;
        always @ (posedge clk or posedge rst)
        begin
                if (rst)
                begin
                        count  <= 0;
                end
                else
                begin
                        if((mode == 1) && (enable == 1))
                        begin
                                if(count <= 4015)
                                begin
                                        count <= count + 1;
                                end
                                else
                                begin
                                        count <= 0;
                                end
                        end
                end
        end
endmodule
```

In the figure 5.4, as the Prime Node goes into state no. 3, the Network Node 2 switches from state 0 to state 1, after detecting the frame start delimiter.

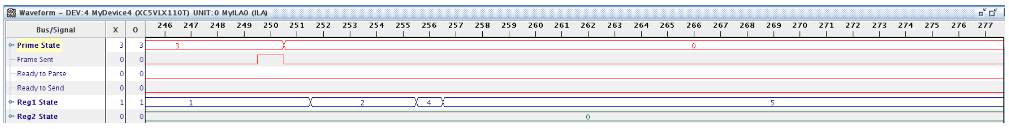


Figure 5.4: Node Synchronization Service triggering the transmission of the datagram

In order to provide a better understanding of the figure 5.4, the diagram 5.5 shows the state in which the FSM for handling the datagram is. The instants t1 to t3 indicate the sequence at which the state changes, to match the corresponding figure 5.4.

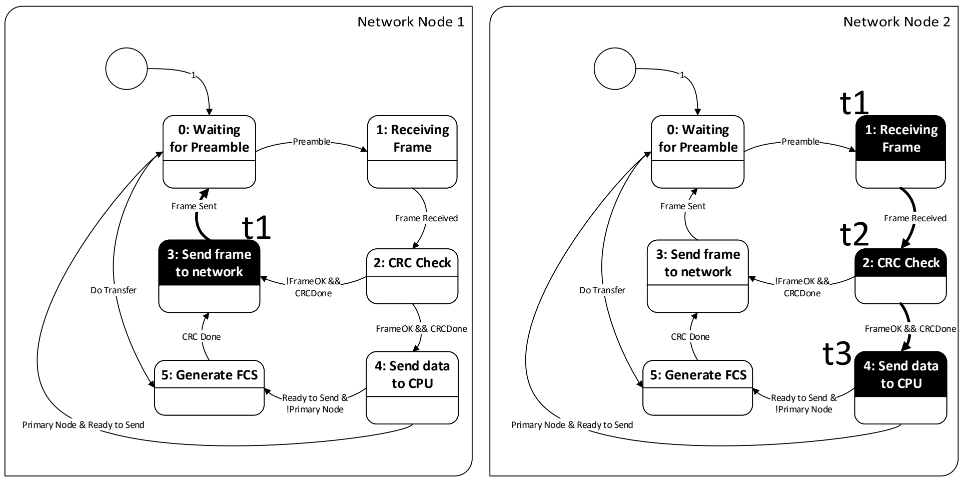


Figure 5.5: Frame reception, integrity check and parsing by the Prime Node

As shown in the listing 5.8, the HDL source code that implements the serializer, intended to sending bit-by-bit, the content of the datagram, is aligned with the de-serializer RTL implementation (presented in the listing 5.9), which has the function of capturing the content of the datagram, also bit-by-bit.

Listing 5.8: Serializer RTL implementation

```
//Send Frame
wire clrt = (fhst == `FHS0) ? 1 : 0;
reg [11:0] iterators;
wire txclken = (fhst == `FHS3) ? 1 : 0;
wire txclk = txclken & clk;
wire rscls = rst || clrt;
```

```
always @ (posedge txclk or posedge rscls)
begin
        if (rscls)
        begin
                ft <= 0;
                iterators <= `FS;
                tx <= 0;
        end
        else
        begin
                if(fhst == `FHS3)
                begin
                        tx <= tframe[iterators];
                        if (iterators == 0)
                        begin
                                ft <= 1;
                        end
                        else
                        begin
                                iterators <= iterators - 1;
                        end
                end
                else
                begin
                        ft <= 0;
                        iterators <= `FS;
                        tx <= 0;
                end
        end
end
//End Send Frame
```

Listing 5.9: Deserializer RTL implementation

```
//Frame Reception
wire clrr = (fhst == `FHS0) ? 1 : 0;
reg [11:0] iteratorr;
```

```
wire rsclr = rst || clrr;

always @ (negedge rxclk or posedge rsclr)
begin
        if (rsclr)
        begin
                rframe <= 0;
                iteratorr <= `FS-4;
                fr <= 0;
        end
        else
        begin
                if(iteratorr >= 1)
                begin
                        rframe[iteratorr] <= rx;
                        iteratorr <= iteratorr - 1;
                end
                else if(iteratorr == 0)
                begin
                        rframe[iteratorr] <= rx;
                        rframe [`FS:`FS-3] <= 4'b1010;
                        fr <= 1;
                end
        end
end
//End Frame Reception
```

As can bee seen in the figure 5.6, having captured all the bits of the frame, it then goes into state 2, to make sure data is not corrupt. If that is the case, its state is changed to the state no. 4, where the actual processing of the frame is supposed to be carried out. As the Network Node 2 (and 3) are there just in order to act as a loopback mechanism, the received data is forwarded to the next node.

In order to provide a better understanding of the figure 5.6, the diagram 5.7 shows the state in which the FSM for handling the datagram is. The instants t1 to t4 indicate the sequence at which the state changes, to match the corresponding figure 5.6.

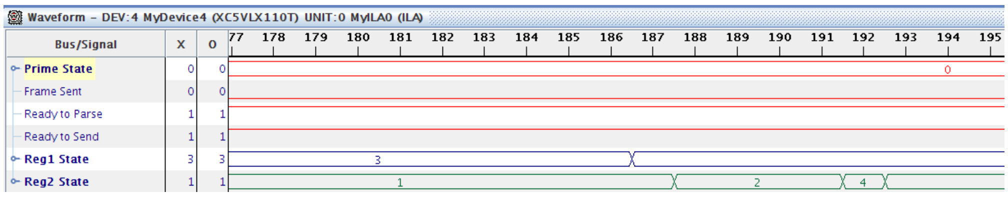


Figure 5.6: Propagation of the datagram through the network

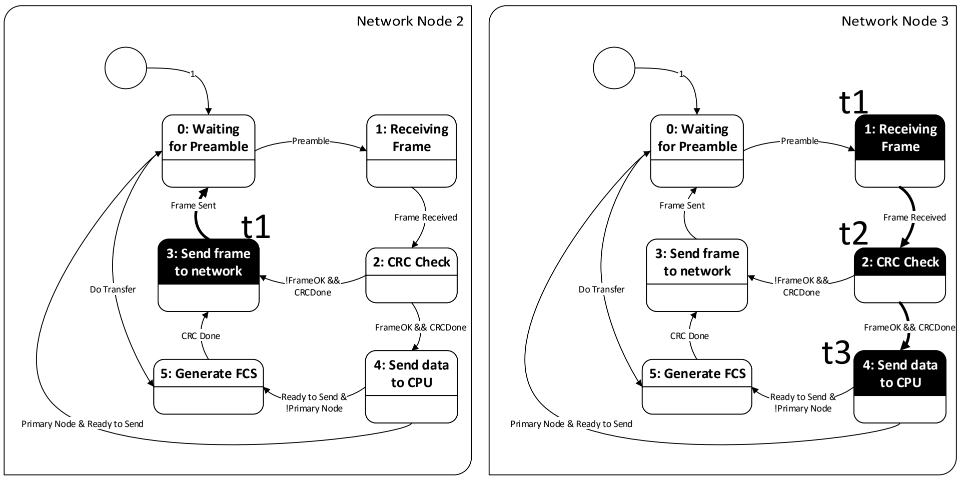


Figure 5.7: Frame reception, integrity check and parsing by the Prime Node

The network node's state becomes number 3, when data starts being sent out to the network, in this case to the Network Node 3, which behaves the same way as described in the previous paragraph.

Then, the Prime Node receives back the data frame, which by now has completed one cycle through the entire network as show in the figure 5.8.

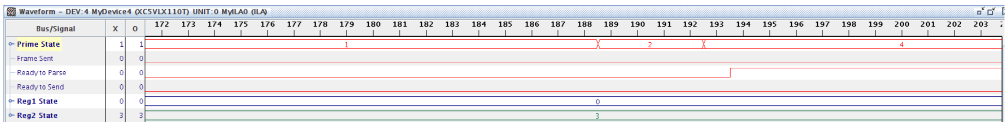


Figure 5.8: Frame reception, integrity check and parsing by the Network Node 3

In order to provide a better understanding of the figure 5.8, the diagram 5.9 shows the state in which the FSM for handling the datagram is. The instants t1 to t3 indicate the sequence at which the state changes, to match the corresponding figure 5.8.

The same applies when it comes to the Network Node 1 receiving the data directed towards it, except in state no. 4, the fact that the rp signal transits from logic

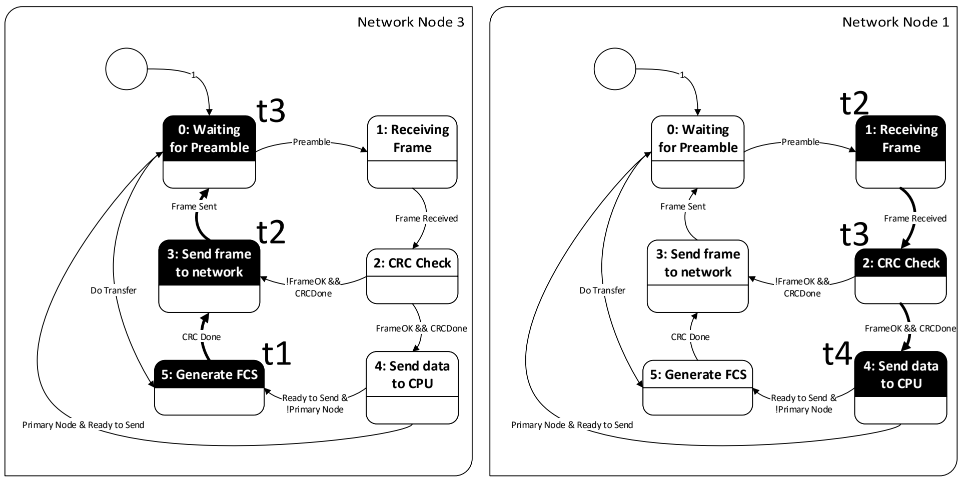


Figure 5.9: Frame reception, integrity check and parsing by the Prime Node

value 0 to 1, triggers an interrupt, causing the CPU to enter in the corresponding interrupt handler, as per the figure 5.10.

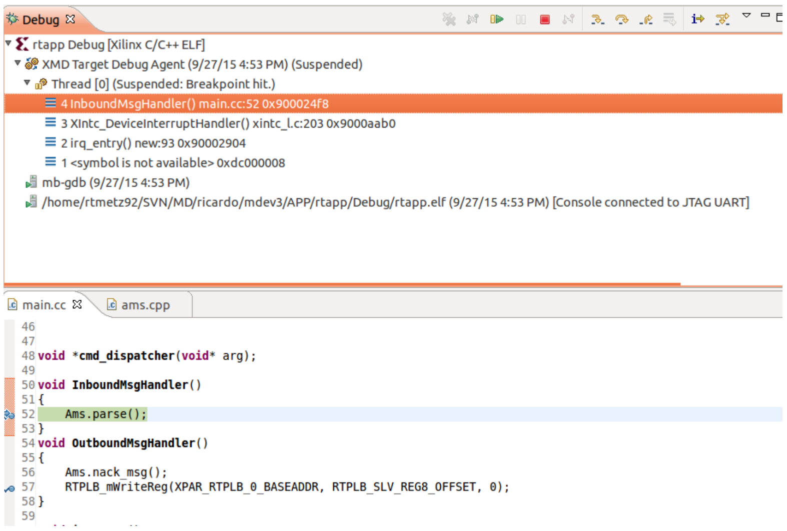
Triggering the interrupt requires that the datagram is directed towards the node in question, which is determined by the mention of its own address or the broadcast address in the "addressee" field of the data frame, otherwise the data is immediately passed on to the next node, as shown in the excerpt of code, included in the code excerpt 5.10.

Listing 5.10: Address matching verification prior to forwarding data for processing

```
`FHS4:
begin
if(rframe[247:160] != 0)
begin
        if((rframe[247:244] == address) || (rframe[247:244]
            == 4'b1111))
        begin
                //Parse rframe
                rp <= 1;
                //End Parse rframe
                if (rs == 1)
                begin
                        tframe [`FS:`FS-3] <= 4'b1010; //
                            preamble
                        tframe [`FS-4:0] <= sframe [`FS
```

```
                                    −4:0];
                                fhst <= ‘FHS5;
                    end
            end
            else
            begin
                    tframe [‘FS:0] <= rframe [‘FS:0];
                    if (master == 1)
                            begin
                                    fhst <= ‘FHS0;
                            end
                            else
                            begin
                                    fhst <= ‘FHS3;
                            end
            end
    end
    else
    begin
            rp <= 1;
            if(rs == 1)
            begin
                    tframe [‘FS:‘FS−3] <= 4’b1010; //preamble
                    tframe [‘FS−4:0] <= sframe [‘FS−4:0];
                    fhst <= ‘FHS5;
            end
    end
end
```

At this stage the frame is stored in the RX Queue.

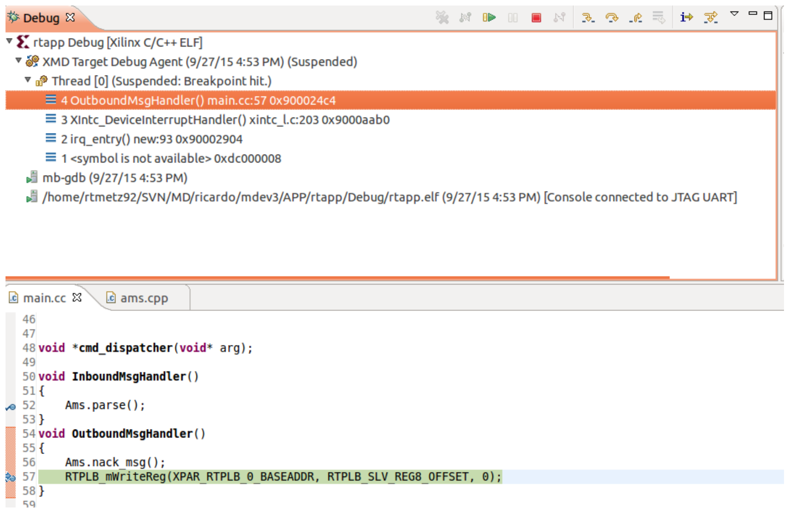


```
rtapp Debug [Xilinx C/C++ ELF]
  XMD Target Debug Agent (9/27/15 4:53 PM) (Suspended)
    Thread [0] (Suspended: Breakpoint hit.)
      4 InboundMsgHandler() main.cc:52 0x900024f8
      3 XIntc_DeviceInterruptHandler() xintc_l.c:203 0x9000aab0
      2 irq_entry() new:93 0x90002904
      1 <symbol is not available> 0xdc000008
  mb-gdb (9/27/15 4:53 PM)
  /home/rtmetz92/SVN/MD/ricardo/mdev3/APP/rtapp/Debug/rtapp.elf (9/27/15 4:53 PM) [Console connected to JTAG UART]
```

```


void *cmd_dispatcher(void* arg);

void InboundMsgHandler()
{
    Ams.parse();
}
void OutboundMsgHandler()
{
    Ams.nack_msg();
    RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR, RTPLB_SLV_REG8_OFFSET, 0);
}

```

Figure 5.10: Inside the interrupt handler for parsing inbound messages

After that, after the Node Synchronization Service triggers the transmission of a new frame to the network and that process is complete, the ft signal changes its value from 0 to 1, causing an interrupt and its handler to be executed, as seen in the figure 5.11. The same process as described previously is repeated until system power-off.



```
rtapp Debug [Xilinx C/C++ ELF]
  XMD Target Debug Agent (9/27/15 4:53 PM) (Suspended)
    Thread [0] (Suspended: Breakpoint hit.)
      4 OutboundMsgHandler() main.cc:57 0x900024c4
      3 XIntc_DeviceInterruptHandler() xintc_l.c:203 0x9000aab0
      2 irq_entry() new:93 0x90002904
      1 <symbol is not available> 0xdc000008
  mb-gdb (9/27/15 4:53 PM)
  /home/rtmetz92/SVN/MD/ricardo/mdev3/APP/rtapp/Debug/rtapp.elf (9/27/15 4:53 PM) [Console connected to JTAG UART]
```

```


void *cmd_dispatcher(void* arg);

void InboundMsgHandler()
{
    Ams.parse();
}
void OutboundMsgHandler()
{
    Ams.nack_msg();
    RTPLB_mWriteReg(XPAR_RTPLB_0_BASEADDR, RTPLB_SLV_REG8_OFFSET, 0);
}
```

Figure 5.11: Interrupt handler execution after a message has been sent out to the network

Chapter 6

Conclusion

The work involved in this project taught several lessons regarding not only about its subject, but also concerning how it has been done and the employed techniques to achieve the presented results. Consisting the work done on a implementation of several layers of a communication protocol, it is clear that FPGA technology has many aspects to be taken advantage of. First and foremost, the portability of the RTL language, which allows the same exact implementation to be deployed to different boards, as long as the design constraints are adapted appropriately to match the board's internal structure and considering that the chip has enough logic elements to perform the designated tasks. Additionally, as during the project development, many specification changes in any of the communication protocol layers might lead to minor/major changes in the other ones, the implemented hardware can easily be reconfigured, by means of changing the corresponding parts of the code. This results in a way higher flexibility when compared to microcontrollers that offers pre-implemented and static hardware modules, which can not be entirely customized to meet the applications needs, in any other way rather than by allowing the developer to control its mode of operation by manipulating values in configuration registers that might be made available for that effect. Still, it might end up being more limiting in some cases, thus demanding increasingly development effort, comparing to a FPGA technology based solution.

Following the work that has been done in this project, even though it would be feasible, the amount of required effort in order to develop all of the proposed items solely in RTL would be very high. The solution to tackle this is to find a balance between making the most of the aforementioned FPGA capabilities and also the greater simplicity brought by development in software. Thus, the ability to bring up an hybrid platform, which incorporates hardware and software implemented parts and the necessary means for both parts to communicate between each other in a totally customized way, as the developer desires, leads to a briefer time-to-market while still leaving open the possibility to make the most of the previously stated FPGA technology advantages, not to mention the ability to execute processes in a pure parallel way and as the datapath and its control unit can be fully customized to perform an specific task, it allows for a more optimized circuit than those of general purpose CPUs, the execution time of routines can be drastically reduced. This is also why many times it is convenient to make use of custom RTL circuits for CPU offloading purposes, in order to meet strict application time requirements, which in case of this project, it is critical to ensure they are respected, as well as the existence of determinism.

Now, speaking of the communication protocol itself, the complexity that is involved makes up for a strong reason to split the problem into various smaller parts and then carry out with finding the solution to each of these, one at a time. So, it is important to divide the protocol into various layers, each of which implements a small subset of routines that altogether contribute towards the system to perform what is was designed to. As is the case of this project, it is also relevant to outline the need to take into consideration, right from the beginning of its development, that the coexistence of synchronous and asynchronous data has to be ensured. They have to be combined in such a way that all the necessary support to control the system is provided and the time-sensitive data is transported in an efficient way, so that its imposed time requirements are met. This lead to two distinct sets of layers of the communication protocol, one for the asynchronous control data to be treated which is more extensive and longer lasting process; and the other for synchronous data to be processed, which must be as optimized as possible for performance, so that such data does not go through as many delays until it reaches its destination. This affects all the protocol's layers overlying and including the Logical Layer, which must be tailored to obey to that distinguished treatment of both kinds of data. The Physical Layer is also not exempt of being appropriately set, provided that it could otherwise be a possible bottleneck that would affect both the system's performance and scalability metrics, mainly because multimedia application demand ever increasingly bandwidth and timing requirements.

Finally, this project contributed significantly to get in-depth knowledge on how communication protocols work and how they are actually implemented, in practice. The acquired knowledge applies not only to the specific kind of communication protocol that has been developed but also to it acts as a strong base to understand how other communication protocols work, provided that as is the case of the developed project, most of the communication protocols are based in a Reference Model, such as the OSI Model, which might grant some similarities in at least part of each existing communication protocol, nowadays.

6.1 Future Work

Having developed the foundations of this system, there are always improvements to be done and also further work in order to be able to make the system fully functional and ready to be applied in a real life scenario.

For that effect, the first step would be to complete the implementation of the application layer, provided that it has only reached the design phase. Considering that the idea would be to reproduce an audio signal, the integration of the AC97 Codec (which is a COTS) with the already implemented parts would be required. Then, the deployment of the implemented project to various boards and its interconnection, with for example, a coaxial cable would already make this system eligible for use in a real life environment. However, note that the boards to have the application deployed as well as its accessories must be compliant the kind of environments they would be subject to. For example, if the idea would be to apply it to a vehicular infotainment system, the components should be Automotive Grade.

In order to further enrich the system, in terms of features, it would be convenient to get an Embedded Linux Distribution to be properly configured and deployed to the boards, having as pros the fact that the services the Linux Operation System offer, would enable the easier development of multimedia applications, by reusing existing open source modules rather than developing applications from the ground up.

Finally, in order to reduce costs, specially regarding the boards to be used, cheaper alternatives can be found, where there is the possibility to easily deploy the HW implemented parts, given the technological agnosticism of the FPGA technology, being the only aspect that would require additional effort, the configuration of the communication mean between the FPGA and the CPU, if the replacement board does not support the same one as the boards to be replaced. In addition, the HW design constraints are very likely to the required to be changed, provided that pin assignment of the required components may differ from board to board, even if based in the same FPGA family. However, the effort required for this task is negligible, when compared to the aforementioned ones.

Bibliography

[1] "Meshdynamics : Highest performance Voice, Video and Data Outdoors."

[2] J. Abbate, *Inventing the Internet.* MIT Press, 2000.

[3] G. P. Agrawal, *Fiber-optic communication systems.* New York: John Wiley & Sons, 2002.

[4] ——, *Fiber-Optic Communication Systems.* Hoboken, NJ, USA: John Wiley & Sons, Inc., oct 2010. [Online]. Available: http://doi.wiley.com/10.1002/9780470918524

[5] Z. Al Mosheky, P. J. Melling, and M. A. Thomson, "In situ real-time monitoring of a fermentation reaction using a fiber-optic FT-IR probe," *Spectroscopy*, vol. 16, no. 6, 2001.

[6] G. Alarcia and S. Herrera, *C.T.N.E.'s PACKET SWITCHING NETWORK. ITS APPLICATIONS*, 1974. [Online]. Available: http://rogerdmoore.ca/PS/CTNEA/CTA.html

[7] M. S. Alfiad, "111 Gb/s POLMUX-RZ-DQPSK Transmission over 1140 km of SSMF with 10.7 Gb/s NRZ-OOK Neighbours," *Proceedings ECOC 2008*, 2008.

[8] V. Alwayn, *Splicing.* Cisco Systems, 2004. [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=170740{&}seqNum=9{&}rl=1

[9] G. R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison-Wesley, 2000.

[10] P. C. Archibald, "<title>Scattering from Infrared Missile Domes</title>," *Optical Engineering*, vol. 17, no. 6, p. 176647, dec 1978. [Online]. Available: http://opticalengineering.spiedigitallibrary.org/article.aspx?doi=10.1117/12.7972298

[11] S. Arora and B. Barak, *Computational Complexity - A Modern Approach.* Cambridge, 2009.

[12] P. Ashwood-Smith, *Shortest Path Bridging IEEE 802.1aq Overview.* Huawei, 2011.

[13] ——, *Shortest Path Bridging IEEE 802.1aq Overview.* Huawei, 2011.

[14] ——, “Shortest Path Bridging IEEE 802.1aq Overview,” 2011.

[15] R. M. Atkins, P. G. Simpkins, and A. D. Yablon, “Track of a fiber fuse: a Rayleigh instability in optical waveguides,” *Optics Letters*, vol. 28, no. 12, p. 974, jun 2003. [Online]. Available: http://ol.osa.org/abstract.cfm?id=72607

[16] H. Attiya and Welch, *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* Wiley-Interscience, 2004.

[17] A. Bache and Y. Matras, *Fundamental Choices in the Development of RCP, the Experimental Packet-Switching Data Transmission Service of the French PTT*, 1976. [Online]. Available: http://rogerdmoore.ca/PS/RCPBAC/RB.html

[18] A. Bache, B. Long, H. Layec, L. Guillou, and Y. Matras, *RCP, the Experimental Packet-Switched Data Transmission Service of the French PTT: History, Connections, Control*, 1976. [Online]. Available: http://rogerdmoore.ca/PS/RCPHCC/RH.html

[19] P. Baran, *RAND Paper P-2626*, 1962. [Online]. Available: http://www.rand.org/pubs/papers/P2626/

[20] G. Bartenev, “The structure and strength of glass fibers,” *Journal of Non-Crystalline Solids*, vol. 1, no. 1, pp. 69–90, dec 1968. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0022309368900070

[21] R. J. Bates, *Optical Switching and Networking Handbook.* New York: McGraw-Hill, 2001.

[22] A. G. Bell, “On the Production and Reproduction of Sound by Light,” *American Journal of Science, Third Series*, vol. XX, no. 118, pp. 305 – 324, 1880.

[23] A. D. Birrell, R. Levin, M. D. Schroeder, and R. M. Needham, “Grapevine: an exercise in distributed computing,” *Communications of*

the ACM, vol. 25, no. 4, pp. 260–274, apr 1982. [Online]. Available: http://portal.acm.org/citation.cfm?doid=358468.358487

[24] D. Boggs, J. Shoch, E. Taft, and R. Metcalfe, "Pup: An Internetwork Architecture," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 612–624, apr 1980. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1094684

[25] T. Boutell and G. Randers-Pehrson, *PNG (Portable Network Graphics) Specification, Version 1.2*. Libpng.org, 1998. [Online]. Available: http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html

[26] N. Bozinovic, Y. Yue, Y. Ren, M. Tur, P. Kristensen, H. Huang, A. E. Willner, and S. Ramachandran, "Terabit-scale orbital angular momentum mode division multiplexing in fibers." *Science (New York, N.Y.)*, vol. 340, no. 6140, pp. 1545–8, jun 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/23812709

[27] K. Brayer, "Evaluation of 32 Degree Polynomials in Error Detection on the SATIN IV Autovon Error Patterns," 1975. [Online]. Available: http://www.dtic.mil/srch/doc?collection=t3{&}id=ADA014825

[28] K. Brayer and J. L. Hammond, *Evaluation of error detection polynomial performance on the AUTOVON channel*. New York: Institute of Electrical and Electronics Engineers, 1975.

[29] R. D. Bright and M. A. Smith, *EXPERIMENTAL PACKET SWITCHING PROJECT OF THE UK POST OFFICE*. Sussex, United Kingdom: Noordhoff International Publishing, 1973. [Online]. Available: http://rogerdmoore.ca/PS/EPSSB.html

[30] M. J. V. D. Burgt, *Coaxial Cables and Applications*. Belden.

[31] D. Burnett and H. Sethi, "Packet switching at philips research laboratories," *Computer Networks (1976)*, vol. 1, no. 6, pp. 341–348, nov 1977. [Online]. Available: http://rogerdmoore.ca/PS/NPLPh/PhilipsA.html

[32] M. Cagalj, I. Aad, S. Ganeriwal, and J.-P. Hubaux, *On selfish behavior in CSMA/CA networks*, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1498536

[33] N. Cam-Winget, R. Housley, D. Wagner, and J. Walker, “Security flaws in 802.11 data link protocols,” *Communications of the ACM*, vol. 46, no. 5, p. 35, may 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=769800.769823

[34] M. K. Carson, *Alexander Graham Bell: Giving Voice To The World.* New York: Sterling Publishing, 2007. [Online]. Available: http://books.google.com/books?id=a46ivzJ1yboC

[35] G. Castagnoli, S. Brauer, and M. Herrmann, “Optimization of cyclic redundancy-check codes with 24 and 32 parity bits,” *IEEE Transactions on Communications*, vol. 41, no. 6, pp. 883–892, jun 1993. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=231911

[36] T. Chakravarty, *Performance of Cyclic Redundancy Codes for Embedded Networks.* Pittsburgh: Carnegie Mellon University, 2001.

[37] M. Chandy, *Parallel Program Design.*

[38] M. Chiang and M. Yang, “Towards Network X-ities From a Topological Point of View: Evolvability and Scalability,” *Proc. 42nd Allerton Conference*, 2004.

[39] ——, “Towards Network X-ities From a Topological Point of View: Evolvability and Scalability,” *Proc. 42nd Allerton Conference*, 2004.

[40] G. Chretien, W. Konig, and J. Rech, *The SITA Network.* Sussex, United Kingdom: Noordhoff International Publishing, 1973. [Online]. Available: http://rogerdmoore.ca/PS/SITAB.html

[41] R. Cole and U. Vishkin, “Deterministic coin tossing with applications to optimal parallel list ranking,” *Information and Control*, vol. 70, no. 1, pp. 32–53, jul 1986. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0019995886800237

[42] D. Comer, “The computer science research network CSNET: a history and status report,” *Communications of the ACM*, vol. 26, no. 10, pp. 747–753, oct 1983. [Online]. Available: http://portal.acm.org/citation.cfm?doid=358413.358423

[43] A. committee PRQC, “mesh topology.” [Online]. Available: http://www.atis.org/glossary/definition.aspx?id=3516

[44] G. Cook, *Catalogue of parametrised CRC algorithms*, 2012. [Online]. Available: http://reveng.sourceforge.net/crc-catalogue/all.htm

[45] J. Corbet, *Receive packet steering.* LWN.net, 2009. [Online]. Available: https://lwn.net/Articles/362339/

[46] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* MIT Press, 1990.

[47] G. Coulouris, *Distributed Systems: Concepts and Design (5th Edition).* Addison-Wesley, 2011.

[48] G. Coulouris, G. Blair, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design (5th Edition).* Boston: Addison-Wesley, 2011.

[49] L. Cuenca, *A PUBLIC PACKET SWITCHING DATA COMMUNICATIONS NETWORK: EIGHT YEARS OF OPERATING EXPERIENCE.* IEEE, 1980. [Online]. Available: http://rogerdmoore.ca/PS/CTNEC1.html

[50] A. B. D. Allan, E. D. Fedyk, E. P. Ashwood-Smith, and P. Unbehagen, *IS-IS Extensions Supporting IEEE 802.1aq.* IETF, 2012. [Online]. Available: http://www.ietf.org/mail-archive/web/ietf-announce/current/msg10166.html

[51] D. W. Davies, *Oral History 189: D. W. Davies interviewed by Martin Campbell-Kelly at the National Physical Laboratory.* Charles Babbage Institute University of Minnesota, Minneapolis, 1986. [Online]. Available: http://conservancy.umn.edu/handle/107241

[52] T. S. Design, "Xilinx EDK Tutorial," pp. 1–12, 2013.

[53] R. Després, *RCP, THE EXPERIMENTAL PACKET-SWITCHED DATA TRANSMISSION SERVICE OF THE FRENCH PTT*, 1974. [Online]. Available: http://rogerdmoore.ca/PS/RCPDEP/RD.html

[54] S. Diego, "Bridging MOST to IEEE Standards," no. July, 2012.

[55] S. Dolev, *Self-Stabilization.* MIT Press, 2000.

[56] J. Duffy, *Largest Illinois healthcare system uproots Cisco to build $40M private cloud.* PC Advisor, 2012. [Online]. Available: http://www.pcadvisor.co.uk/news/internet/3357242/largest-illinois-healthcare-system-uproots-cisco-build-40m-private-cloud/

[57] H. J. R. Dutton, "Understanding Optical Communications," *International Technical Support Organisation*, vol. 1, pp. 1–638, 1998.

[58] J. Edge, *Receive flow steering.* LWN.net, 2010. [Online]. Available: https://lwn.net/Articles/382428/

[59] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems.* Addison-Wesley, 2000.

[60] W. C. Elmore and M. A. Heald, *Physics of Waves*, 1969.

[61] S. Ely and D. Wright, *L.F. Radio-Data: specification of BBC experimental transmissions 1982.* Research Department, Engineering Division, The British Broadcasting Corporation, 1982.

[62] G. C. Ewing, *Reverse-Engineering a CRC Algorithm.* Christchurch: University of Canterbury, 2010. [Online]. Available: http://www.cosc.canterbury.ac.nz/greg.ewing/essays/CRC-Reverse-Engineering.html

[63] J. Faber, *Java Distributed Computing.* O'Reilly, 1998. [Online]. Available: http://docstore.mik.ua/orelly/java-ent/dist/index.htm

[64] K. Features, "Why Use a," pp. 1–52, 2011.

[65] W. Feldenkirchen, *Werner von Siemens - Inventor and International Entrepreneur*, 1994.

[66] M. Feldman, "National LambdaRail Opens for Business," *HPCwire*, 2008. [Online]. Available: http://www.hpcwire.com/hpcwire/2008-10-28/national{_}lambdarail{_}opens{_}for{_}business.html

[67] J. G. Fletcher, *Principles of Design in the Octopus Computer network*, 1975. [Online]. Available: http://portal.acm.org/citation.cfm?id=810357

[68] K. Francis, "Fiber .... Have You Seen The Light ?" 2012.

[69] R. Freeman, C. I. Constru, T. Espec, and O. Fpgas, "Fundamentos da tecnologia FPGA Visão geral," pp. 1–3, 2013.

[70] T. L. Friedman, *The World is Flat.* Picador, 2007.

[71] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum-Weight Spanning Trees," *ACM Transactions on Programming*

Languages and Systems, vol. 5, no. 1, pp. 66–77, jan 1983. [Online]. Available: http://portal.acm.org/citation.cfm?doid=357195.357200

[72] W. Gambling, "The rise and rise of optical fibers," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 6, no. 6, pp. 1084–1093, nov 2000. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=902157

[73] B. M. Gammel, *Matpack documentation: Crypto - Codes.* Matpack.de, 2005. [Online]. Available: http://www.matpack.de/index.html{#}DOWNLOAD

[74] V. K. Garg, *Elements of Distributed Computing.* Wiley-IEEE Press, 2002.

[75] P. Geremia, "Cyclic redundancy check computation: an implementation using the TMS320C54x," no. SPRA530, 1999.

[76] R. Ghaemi, A. Milani Fard, M. Sadeghizadeh, and H. Tabatabaee, "Evolutionary Query Optimization for Heterogeneous Distributed Database Systems," *World Academy of Science, Engineering and Technology*, no. 19, pp. 43 – 49, 2008.

[77] S. Ghosh, *Distributed Systems - An Algorithmic Approach.* Chapman & Hall/CRC, 2007.

[78] G. S. Glaesemann, "Advancements in Mechanical Strength and Reliability of Optical Fibers," *Proc. SPIE*, vol. CR73, 1999. [Online]. Available: http://www.corning.com/WorkArea/downloadasset.aspx?id=7783

[79] A. P. Glover, "Application Note : Embedded Hardware Systems Using and Creating Interrupt-Based," vol. 778, pp. 1–37, 2005.

[80] B. Godfrey, *A primer on distributed computing*, 2002. [Online]. Available: http://www.bacchae.co.uk/docs/dist.html

[81] A. Govind, *Nonlinear Fiber Optics, Fifth Edition.*

[82] J. Gowar, *Optical communication systems.* Hempstead, UK: Prentice-Hall, 1993.

[83] D. Groth and T. Skandier, *Network+ Study Guide, Fourth Edition'.* Sybex, Inc., 2005.

[84] ——, *Network+ Study Guide, Fourth Edition'.* Sybex, Inc., 2005.

[85] A.-R. Haarala, *Libraries as key players at the local level.* [Online]. Available: http://edoc.hu-berlin.de/conferences/eunis2001/e/Haarala/HTML/haarala-ch2.html

[86] P. Halvorsen, T. Plagemann, and V. Goebel, "Improving the I/O performance of intermediate multimedia storage nodes," *{ACM}/{S}pringer Multimedia Systems*, vol. 9, no. 1, pp. 56–67, 2003.

[87] H. Hamilton, *Distributed Algorithms.* [Online]. Available: http://www2.cs.uregina.ca/{~}hamilton/courses/330/notes/distributed/distributed.html

[88] J. Hammond, Joseph L., J. E. Brown, and S.-S. Liu, "Development of a Transmission Error Model and an Error Control Model," *Unknown*, vol. 76, 1975.

[89] J. Hecht, *City of Light, The Story of Fiber Optics.* New York: Oxford University Press, 1999. [Online]. Available: http://books.google.com/?id=4oMu7RbGpqUC{&}pg=PA114

[90] ——, *Understanding Fiber Optics.* Prentice Hall, 2002.

[91] T. Herbert and W. de Bruijn, *Documentation/networking/scaling.txt.* kernel.org, 2014. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/scaling.txt

[92] M. P. Herlihy and N. N. Shavit, *The Art of Multiprocessor Programming.* Morgan Kaufmann, 2008.

[93] B. Hitz, "Origin of 'fiber fuse' is revealed," *Photonics Spectra*, 2003. [Online]. Available: http://www.photonics.com/Article.aspx?AID=16745

[94] H. H. HOPKINS and N. S. KAPANY, "A Flexible Fibrescope, using Static Scanning," *Nature*, vol. 173, no. 4392, pp. 39–41, jan 1954. [Online]. Available: http://www.nature.com/doifinder/10.1038/173039b0

[95] J. D. Jackson, *Classical Electrodynamics.* New York: John Wiley & Sons, Inc., 1962.

[96] J. Jang, C. Kim, and Y. Yu, "A Study on the MOST150/Ethernet Gateway of In-Vehicle Network," *Ijcsns*, vol. 10, no. 9, pp. 62–65, 2010. [Online]. Available: http://paper.ijcsns.org/07{_}book/201009/20100910.pdf

[97] D. T. Jones, “An Improved 64-bit Cyclic Redundancy Check for Protein Sequences.”

[98] U. G. June, “Virtex-5 FPGA Packaging and Pinout Specification,” vol. 195, pp. 1–416, 2012.

[99] M. Kabir, “Network Architecture of a Modern Automotive Infotainment System,” *Advances in Automobile Engineering*, vol. 01, no. 03, pp. 1–5, 2012. [Online]. Available: http://www.omicsgroup.org/journals/2167-7670/2167-7670-1-101.digital/2167-7670-1-101.html

[100] M. Karabulut, E. Melnik, R. Stefan, G. Marasinghe, C. Ray, C. Kurkjian, and D. Day, “Mechanical and structural properties of phosphate glasses,” *Journal of Non-Crystalline Solids*, vol. 288, no. 1-3, pp. 8–17, aug 2001. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0022309301006159

[101] I. Keidar, “ACM SIGACT news distributed computing column 32,” *ACM SIGACT News*, vol. 39, no. 4, p. 53, nov 2008. [Online]. Available: http://webee.technion.ac.il/{~}idish/sigactNews/{#}column32

[102] P. T. Kirstein, *A SURVEY OF PRESENT AND PLANNED GENERAL PURPOSE EUROPEAN DATA AND COMPUTER NETWORKS.* Sussex, United Kingdom: Noordhoff International Publishing, 1973. [Online]. Available: http://rogerdmoore.ca/PS/Kirs1973/Ki.html{#}GEISCO

[103] G. M. Kizer, *Microwave communication.* Iowa State University Press, 1990. [Online]. Available: http://books.google.co.uk/books?id=T2fI766k2R0C{&}pg=PA312

[104] G. Knott and N. Waites, *BTEC Nationals for IT Practitioners.* Brancepeth Computer Publications, 2002.

[105] P. Koopman, “32-bit cyclic redundancy codes for Internet applications,” in *Proceedings International Conference on Dependable Systems and Networks.* IEEE Comput. Soc, 2002, pp. 459–468. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1028931

[106] P. Koopman and T. Chakravarty, “Cyclic redundancy code (CRC) polynomial selection for embedded networks,” in *International Conference on Dependable Systems and Networks, 2004.* IEEE, 2004, pp. 145–154.

[Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1311885

[107] P. Koopman, *Blog: Checksum and CRC Central.*

[108] P. Koopman, K. Driscoll, and B. Hall, *Cyclic Redundancy Code and Checksum Algorithms to Ensure Critical Data Integrity.* Federal Aviation Administration, 2015.

[109] E. Korach, S. Kutten, and S. Moran, "A modular technique for the design of efficient distributed leader finding algorithms," *ACM Transactions on Programming Languages and Systems*, vol. 12, no. 1, pp. 84–101, jan 1990. [Online]. Available: http://portal.acm.org/citation.cfm?doid=77606.77610

[110] G. Kostovski, P. R. Stoddart, and A. Mitchell, "The Optical Fiber Tip: An Inherently Light-Coupled Microscopic Platform for Micro- and Nanotechnologies," *Advanced Materials*, vol. 26, no. 23, pp. 3798–3820, jun 2014. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/24599822

[111] M. B. Kounavis, *A Systematic Approach to Building High Performance, Software-based, CRC generators.* Intel, 2005.

[112] D. Kouznetsov and J. Moloney, "Highly efficient, high-gain, short-length, and power-scalable incoherent diode slab-pumped fiber amplifier/laser," *IEEE Journal of Quantum Electronics*, vol. 39, no. 11, pp. 1452–1461, nov 2003. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1242365

[113] C. R. Kurkjian, P. G. Simpkins, and D. Inniss, "Strength, Degradation, and Coating of Silica Lightguides," *Journal of the American Ceramic Society*, vol. 76, no. 5, pp. 1106–1112, may 1993. [Online]. Available: http://doi.wiley.com/10.1111/j.1151-2916.1993.tb03727.x

[114] C. R. Kurkjian, O. S. Gebizlioglu, and I. Camlibel, "<title>Strength variations in silica fibers</title>," in *Proceedings of SPIE*, M. J. Matthewson, Ed., vol. 3848, dec 1999, pp. 77–86. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=909071

[115] C. Kurkjian, "Mechanical stability of oxide glasses," *Journal of Non-Crystalline Solids*, vol. 102, no. 1-3, pp. 71–81, jun 1988. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0022309388901147

[116] ——, “Mechanical properties of phosphate glasses,” *Journal of Non-Crystalline Solids*, vol. 263-264, pp. 207–212, mar 2000. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0022309399006377

[117] C. Kurkjian, J. Krause, and M. Matthewson, “Strength and fatigue of silica optical fibers,” *Journal of Lightwave Technology*, vol. 7, no. 9, pp. 1360–1370, 1989. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=50715

[118] J. F. Kurose and K. W. Ross, *COMPUTER NETWORKING A Top-Down Approach.*

[119] L. Lavandera, *ARCHITECTURE, PROTOCOLS AND PERFORMANCE OF RETD.* IEEE, 1980. [Online]. Available: http://rogerdmoore.ca/PS/RETDB.html

[120] B. Lee, “Review of the present status of optical fiber sensors,” *Optical Fiber Technology*, vol. 9, no. 2, pp. 57–79, apr 2003. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1068520002005278

[121] M. M. Lee, C. V. Cryan, J. M. Roth, and T. G. Ulmer, “The Fiber Fuse Phenomenon in Polarization-Maintaining Fibers at 1.55 μm,” *Conference on Lasers and Electro-Optics/Quantum Electronics and Laser Science Conference and Photonic Applications Systems Technologies*, 2006.

[122] Y.-J. Lee, Y.-H. Park, S.-W. Sok, H.-Y. Kim, and C.-H. Lee, “Fast-path I/O architecture for high performance streaming server,” *Middleware 2011*, vol. 50, no. 2, pp. 99–120, 2008. [Online]. Available: http://www.springerlink.com/index/10.1007/s11227-008-0254-5

[123] P. Lind and M. Alm, “A Database-Centric Virtual Chemistry System,” *Journal of Chemical Information and Modeling*, vol. 46, no. 3, pp. 1034–1039, may 2006. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/16711722

[124] N. Linial, “Locality in Distributed Graph Algorithms,” *SIAM Journal on Computing*, vol. 21, no. 1, pp. 193–201, feb 1992. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/0221015

[125] T. Lorenz, “Advanced Gateways in Automotive Applications,” *Elektrotechnik und Informatik der Technische Universität Berlin*, 2008.

[126] N. A. Lynch, *Distributed Algorithms.* Morgan Kaufmann, 1996.

[127] N. Mandayam, G. Editor, S. Wicker, J. Walrand, T. Basar, J. Huang, and D. Palomar, "Game Theory in Communication Systems [Guest Editorial]," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 7, pp. 1042–1046, sep 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4604730

[128] C. C. Martel, J. M. Cunningham, and M. S. Grushcow, *THE BNR NETWORK: A CANADIAN EXPERIENCE WITH PACKET SWITCHING TECHNOLOGY.* [Online]. Available: http://rogerdmoore.ca/PS/BNR/BNRnet.html

[129] P. Melling and M. Thomson, "Reaction monitoring in small reactors and tight spaces," *American Laboratory News*, 2002.

[130] P. J. Melling and M. Thomson, *Fiber-optic probes for mid-infrared spectrometry.* Wiley, 2002.

[131] S. Mendicino, *Octopus: The Lawrence Radiation Laboratory Network*, 1970. [Online]. Available: http://www.rogerdmoore.ca/PS/OCTOA/OCTO.html

[132] S. F. Mendicino, "1970 OCTOPUS: THE LAWRENCE RADIATION LABORATORY NETWORK," *COMPUTER NETWORKS*, pp. 95 – 100, 1972. [Online]. Available: http://rogerdmoore.ca/PS/OCTOA/OCTO.html

[133] T. Microblaze and S. Kit, "Module 3 : Adding Custom IP to an Embedded System to an Embedded System."

[134] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 217–252, aug 1997. [Online]. Available: http://portal.acm.org/citation.cfm?id=263335

[135] P. J. Nahin, *Oliver Heaviside: The Life, Work, and Times of an Electrical Genius of the Victorian Age*, 2002.

[136] M. Naor and L. Stockmeyer, "What Can be Computed Locally?" *SIAM Journal on Computing*, vol. 24, no. 6, pp. 1259–1277, dec 1995. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/S0097539793254571

[137] S.-M. F. Nee, L. F. Johnson, M. B. Moran, J. M. Pentony, S. M. Daigneault, D. C. Tran, K. W. Billman, and S. Siahatgar, "<title>Optical and surface properties of oxyfluoride glass</title>," in *Proceedings of SPIE*, A. J. Marker III and E. G. Arthurs, Eds., vol. 4102, oct 2000, pp. 122–133. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=916530

[138] L. Network and B. Communication, "Controller Area Network (CAN) Overview," pp. 2–6, 2014.

[139] J.-i. Nishizawa and Suto, *Terahertz wave generation and light amplification using Raman effect.* New Delhi, India: Narosa Publishing House, 2004. [Online]. Available: http://books.google.com/?id=2NTpSnfhResC{&}pg=PA27

[140] T. Olzak, *Protect your network against fiber hacks.* CNET, 2007. [Online]. Available: http://blogs.techrepublic.com.com/security/?p=222

[141] F. Optics and S. Solutions, "White Paper Connecting it all together - Fiber Optics in Security & Surveillance."

[142] M. Oriented and S. Transport, *MOST ToGo Architecture and Implementation User ' s Guide Supporting MOST Networks*, 2014, no. May.

[143] H. W. Ott, *Noise Reduction Techniques in Electronic Systems*, 1976.

[144] C. H. Papadimitriou, *Computational Complexity.* Addison-Wesley, 1994.

[145] W. Paper, "Fiber or Coaxial : Which one is best for your business ?" 2016.

[146] R. Paschotta, *Fibers.* RP Photonics. [Online]. Available: https://www.rp-photonics.com/fibers.html

[147] ——, *Brillouin Scattering.* RP Photonics. [Online]. Available: https://www.rp-photonics.com/brillouin{_}scattering.html

[148] ——, *Tutorial on Passive Fiber optics.* RP Photonics. [Online]. Available: https://www.rp-photonics.com/passive{_}fiber{_}optics.html

[149] D. Pearson and D. Wilkin, *Some Design Aspects of a public packet switching network*, 1974. [Online]. Available: http://rogerdmoore.ca/PS/EPSSFer/EF.html

[150] D. L. Pehrson, *AN ENGINEERING VIEW OF THE LRL OCTOPUS COMPUTER NETWORK*, 1970. [Online]. Available: http://www.computer-history.info/Page4.dir/pages/Octopus.dir/index.html

[151] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000. [Online]. Available: http://www.ec-securehost.com/SIAM/DT05.html

[152] J. Pelkey, *Entrepreneurial Capitalism and Innovation: A History of Computer Communications 1968-1988.* [Online]. Available: http://www.historyofcomputercommunications.info/Book/6/6.3-CYCLADESNetworkLouisPouzin1-72.html

[153] A. Perez, "Byte-Wise CRC Calculations," *IEEE Micro*, vol. 3, no. 3, pp. 40–50, jun 1983. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4070930

[154] I. Peter, *Ian Peter's History of the Internet*, 2004. [Online]. Available: http://www.nethistory.info/HistoryoftheInternet/

[155] W. Peterson and D. Brown, "Cyclic Codes for Error Detection," *Proceedings of the IRE*, vol. 49, no. 1, pp. 228–235, jan 1961. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4066263

[156] T. Plagemann, V. Goebel, P. Halvorsen, and O. Anshus, "Operating system support for multimedia systems," *Computer Communications*, vol. 23, no. 3, pp. 267–289, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366499001802

[157] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Section 22.4 Cyclic Redundancy and Other Checksums.* New York: Cambridge University Press, 2007. [Online]. Available: http://apps.nrbook.com/empanel/index.html{#}pg=1168

[158] B. A. Proctor, I. Whitney, and J. W. Johnson, "The Strength of Fused Silica," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 297, no. 1451, pp. 534–557, mar 1967. [Online]. Available: http://rspa.royalsocietypublishing.org/cgi/doi/10.1098/rspa.1967.0085

[159] T. Ramabadran and S. Gaitonde, "A tutorial on CRC computations," *IEEE Micro*, vol. 8, no. 4, pp. 62–75, aug 1988. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7773

[160] P. N. Recommended and N. Designs, "Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel," *Intellectual Property*, vol. 529, pp. 1–12, 2004.

[161] A. Rehmann and J. D. Mestre, "Air Ground Data Link VHF Airline Communications and Reporting System (ACARS) Preliminary Test Report," 1995.

[162] A. Richardson, *WCDMA Handbook*. Cambridge, UK: Cambridge University Press, 2005. [Online]. Available: http://books.google.co.uk/books?id=yN5lve5L4vwC{&}lpg=PA223{&}dq={&}pg=PA223{#}v=onepage{&}q{&}f=false

[163] T. Ritter, "The Great CRC Mystery," *Dr. Dobb's Journal*, vol. 11, no. 2, pp. 26 – 34, 1986. [Online]. Available: http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM

[164] J. Robert Lee Lounsbury, "OPTIMUM ANTENNA CONFIGURATION FOR MAXIMIZING ACCESS POINT RANGE OF AN IEEE 802.11 WIRELESS MESH NETWORK IN SUPPORT OF MULTIMISSION OPERATIONS RELATIVE TO HASTILY FORMED SCALABLE DEPLOYMENTS."

[165] ——, "Optimum Antenna Configuration for Maximizing Access Point Range of an IEEE 802.11 Wireless Mesh Network in Support of Multimission Operations Relative to Hastily Formed Scalable Deployments," 2007.

[166] ——, "Optimum Antenna Configuration for Maximizing Access Point Range of an IEEE 802.11 Wireless Mesh Network in Support of Multimission Operations Relative to Hastily Formed Scalable Deployments," 2007.

[167] L. G. Roberts and B. D. Wessler, "Computer network development to achieve resource sharing," in *Proceedings of the May 5-7, 1970, spring joint computer conference on - AFIPS '70 (Spring)*. New York, New York, USA: ACM Press, 1970, p. 543. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1476936.1477020

[168] A. Rubini, G. Kroah-Hartman, and J. Corbet, *Linux Device Drivers, Third Edition, Chapter 10. Interrupt Handling*. O'Reilly Media, 2005.

[169] P. Russell, "Photonic crystal fibers." *Science (New York, N.Y.)*, vol. 299, no. 5605, pp. 358–62, jan 2003. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/12532007

[170] G. Sasaki, K. Sivarajan, and R. Ramaswami, *Optical Networks: A Practical Perspective.* Morgan Kaufmann, 2009.

[171] R. A. Scantlebury and P. Wilkinson, *The National Physical Laboratory Data Communications Network*, 1974. [Online]. Available: http://www.rogerdmoore.ca/PS/NPLPh/NPL1974A.html

[172] S. Schmechtig and J. Kjelsbak, "FlexRay - The Hardware View FlexRay - The Hardware View," no. February, 2006.

[173] M. Schwartz, R. Boorstyn, and R. Pickholtz, "Terminal-oriented computer-communication networks," *Proceedings of the IEEE*, vol. 60, no. 11, pp. 1408–1423, 1972. [Online]. Available: http://rogerdmoore.ca/PS/TONET/TON.html{#}GEISCO

[174] J. M. Senior and M. Y. Jamro, *Optical fiber communications: principles and practice.* Pearson Education, 2009. [Online]. Available: http://cds.cern.ch/record/1493478

[175] K. Seo, "Evaluation of high-power endurance in optical fiber links," *Furukawa Review*, no. 24, pp. 17 – 22, 2003.

[176] Z. Shi, C. Beard, and K. Mitchell, *Competition, cooperation, and optimization in Multi-Hop CSMA networks*, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2069084

[177] A. Skontorp, "<title>Nonlinear mechanical properties of silica-based optical fibers</title>," in *Proceedings of SPIE*, P. F. Gobin and C. M. Friend, Eds., vol. 4073, aug 2000, pp. 278–289. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=910705

[178] L. Skuja, M. Hirano, H. Hosono, and K. Kajihara, "Defects in oxide glasses," *physica status solidi (c)*, vol. 2, no. 1, pp. 15–24, jan 2005. [Online]. Available: http://doi.wiley.com/10.1002/pssc.200460102

[179] R. G. Smith, "Optical power handling capacity of low loss optical fibers as determined by stimulated Raman and brillouin scattering." *Applied optics*, vol. 11, no. 11, pp. 2489–94, nov 1972. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/20119362

[180] P. Specification, C. Specifics, R. Used, and D. T. Requirements, "LogiCORE IP Fast Simplex," pp. 1–9, 2011.

[181] W. Stallings, *Data and computer communications*, 1997. [Online]. Available: http://tinnakorn.cs.rmu.ac.th/Courses/DataCommNetwork/Book/DataAndComputerCommunications8E.pdf

[182] B. Stewart, *Paul Baran Invents Packet Switching*, 2000. [Online]. Available: http://www.livinginternet.com/i/ii{_}rand.htm

[183] M. Stigge, W. Müller, H. Plötz, and J.-P. Redlich, "Reversing CRC - Theory and Practice," 2006.

[184] T. Strang, "Multimedia Protocols."

[185] R. Sundstrom and G. Schultz, *1980 SNA'S First Six Years: 1974-1980*, 1980. [Online]. Available: http://rogerdmoore.ca/PS/SNA6Y/SNA6.html

[186] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks, Fifth Edition*, 2011.

[187] S. Taylor and J. Metzler, *Vint Cerf on why TCP/IP was so long in coming*, 2008. [Online]. Available: http://www.networkworld.com/newsletters/frame/2008/0128wan1.html

[188] T. Teachout, *The New-Media Crisis of 1949*. Wall Street Journal. [Online]. Available: http://www.wsj.com/news/articles/SB10001424052970204683204574357332713730174

[189] G. Tel, *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[190] P. Thaler, "16-bit CRC polynomial selection," 2003.

[191] K. Tomaru, S. Yamaguchi, and T. Kato, *A Private Packet Network and Its Application in A Worldwide Integrated Communication Network*, 1980. [Online]. Available: http://rogerdmoore.ca/PS/HIPA/HIA.html

[192] D. Tran, G. Sigel, and B. Bendow, "Heavy metal fluoride glasses and fibers: A review," *Journal of Lightwave Technology*, vol. 2, no. 5, pp. 566–586, 1984. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1073661

[193] L. R. W. TYMES, *TYMNET - A terminal oriented communication network*. [Online]. Available: http://rogerdmoore.ca/PS/TYMNET/TY.html

[194] ——, “Routing and Flow Control in TYMNET,” *IEEE TRANSACTIONS ON COMMUNICATIONS*, vol. COM-29, no. 4, pp. 392 – 98, 1981. [Online]. Available: http://www.rogerdmoore.ca/PS/TYMFlow/TF.html

[195] J. Tyndall, *Total Reflexion*, 1870. [Online]. Available: http://www.archive.org/details/notesofcourseofn00tyndrich

[196] ——, *Six Lectures on Light*, 1873. [Online]. Available: http://www.archive.org/details/sixlecturesonlig00tynduoft

[197] D. Type and P. Date, “FlexRay Automotive Communication Bus Overview,” *Exchange Organizational Behavior Teaching Journal*, pp. 1–7, 2009.

[198] J. WELLS, “Hair light guide,” *Nature*, vol. 338, no. 6210, pp. 23–23, mar 1989. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/2918918

[199] E. A. e. Williams, *National Association of Broadcasters Engineering Handbook (Tenth Edition).* Taylor & Francis, 2007.

[200] R. N. Williams, *A Painless Guide to CRC Error Detection Algorithms V3.0*, 1996. [Online]. Available: http://www.repairfaq.org/filipg/LINK/F{_}crc{_}v3.html

[201] Xilinx, “Custom Peripheral Design Guide,” 2008.

[202] ——, “L507 Evaluation Evaluation Platform Platform User Guide,” vol. 347, pp. 1–60, 2009.

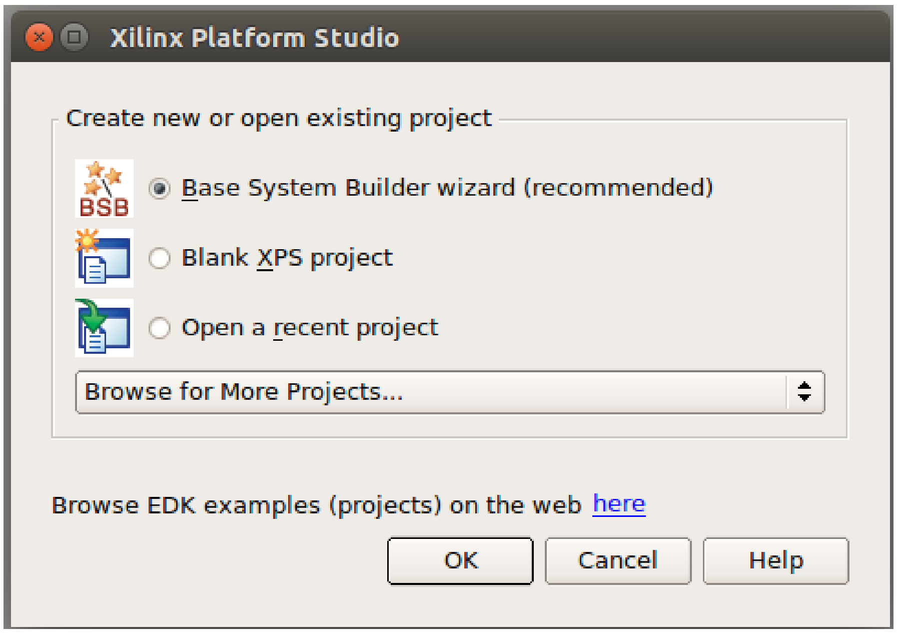
[203] I. Xilinx, “Xilinx UG190 Virtex-5 FPGA User Guide,” *6.375*, vol. 190, pp. 1–385, 2012. [Online]. Available: http://www.xilinx.com/support/documentation/user{_}guides/ug190.pdf

[204] Xilinx Inc., “Xilkernel,” *Xilinx int., Application Note*, pp. 1–56, 2006. [Online]. Available: http://www.xilinx.com/ise/embedded/edk91i{_}docs/xilkernel{_}v3{_}00{_}a.pdf

Appendix A

Building Hybrid CPU+FPGA System via Xilinx EDK

This guide is meant for the Xilinx ISE 13.2 Design Suite, being Xilinx XUPV5-LX110T the board used, explaining how to setup the EDA to build a Hybrid CPU+FPGA system, based on the PLB, to enable communication between the CPU and the RTL implemented IP Cores. Having started the Xilinx Platform Studio, proceed to call the Base System Builder wizard.


Xilinx Platform Studio

Create new or open existing project

(●) Base System Builder wizard (recommended)
( ) Blank XPS project
( ) Open a recent project

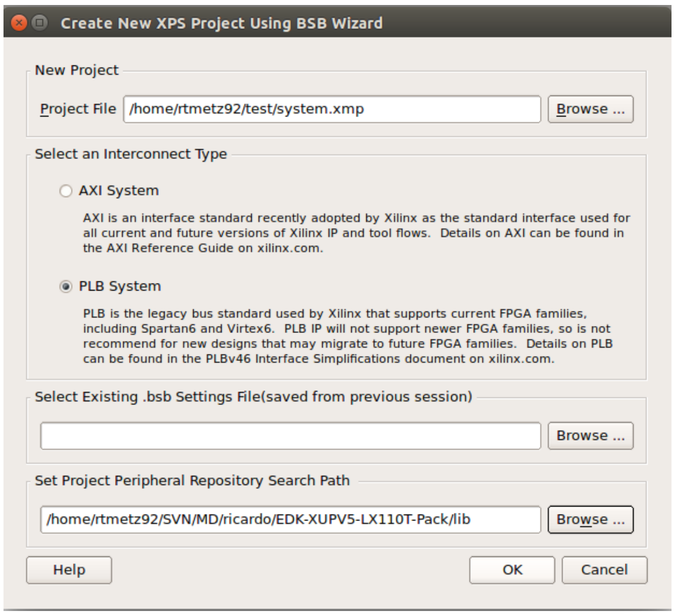
Browse for More Projects...

Browse EDK examples (projects) on the web here

Figure A.1: Building Hybrid CPU+FPGA System via Xilinx EDK with the help of the setup wizard

After that, select the PLB System Interconnect type and make sure to include the

EDK-XUPV5-LX110T-Pack specific configuration file, so that the tool can recognize the board and generate the System accordingly to the board's specifications.


Create New XPS Project Using BSB Wizard

New Project

Project File: /home/rtmetz92/test/system.xmp

Select an Interconnect Type

○ AXI System

AXI is an interface standard recently adopted by Xilinx as the standard interface used for all current and future versions of Xilinx IP and tool flows. Details on AXI can be found in the AXI Reference Guide on xilinx.com.

● PLB System

PLB is the legacy bus standard used by Xilinx that supports current FPGA families, including Spartan6 and Virtex6. PLB IP will not support newer FPGA families, so is not recommend for new designs that may migrate to future FPGA families. Details on PLB can be found in the PLBv46 Interface Simplifications document on xilinx.com.

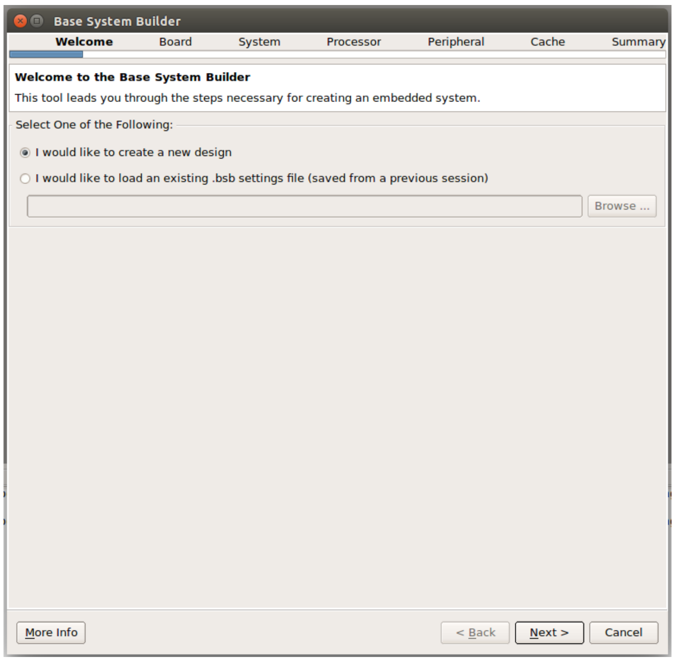
Select Existing .bsb Settings File(saved from previous session)

Set Project Peripheral Repository Search Path

/home/rtmetz92/SVN/MD/ricardo/EDK-XUPV5-LX110T-Pack/lib

Figure A.2: Building Hybrid CPU+FPGA System via Xilinx EDK - Interconnect Bus solution selection and board specification importation

Then, the choice to create a new design is done.


Base System Builder

Welcome to the Base System Builder

This tool leads you through the steps necessary for creating an embedded system.

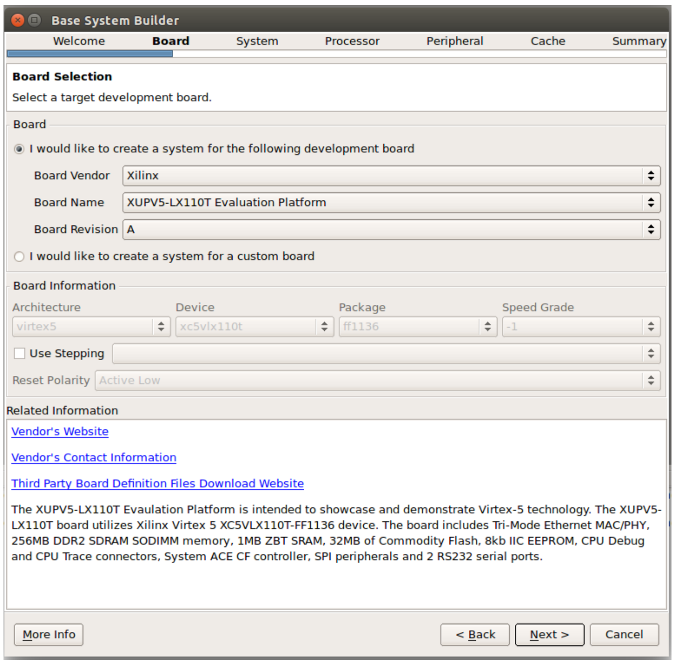
Select One of the Following:

● I would like to create a new design

○ I would like to load an existing .bsb settings file (saved from a previous session)

Figure A.3: Building Hybrid CPU+FPGA System via Xilinx EDK - Creation of a new design from scratch

The proper board has to be selected.



Board Selection

Select a target development board.

Board

(●) I would like to create a system for the following development board

Board Vendor: Xilinx

Board Name: XUPV5-LX110T Evaluation Platform

Board Revision: A

( ) I would like to create a system for a custom board

Board Information

Architecture	Device	Package	Speed Grade
virtex5	xc5vlx110t	ff1136	-1

[ ] Use Stepping

Reset Polarity: Active Low

Related Information

Vendor's Website

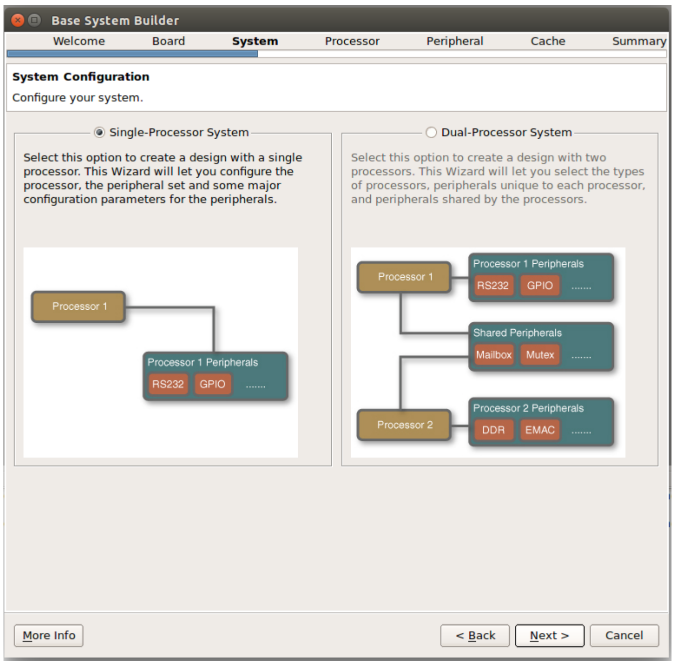
Vendor's Contact Information

Third Party Board Definition Files Download Website

The XUPV5-LX110T Evaulation Platform is intended to showcase and demonstrate Virtex-5 technology. The XUPV5-LX110T board utilizes Xilinx Virtex 5 XC5VLX110T-FF1136 device. The board includes Tri-Mode Ethernet MAC/PHY, 256MB DDR2 SDRAM SODIMM memory, 1MB ZBT SRAM, 32MB of Commodity Flash, 8kb IIC EEPROM, CPU Debug and CPU Trace connectors, System ACE CF controller, SPI peripherals and 2 RS232 serial ports.

Figure A.4: Building Hybrid CPU+FPGA System via Xilinx EDK - Board selection

Furthermore, in this case, a single processor is going to be part of the solution to be built.



System Configuration

Configure your system.

(●) Single-Processor System

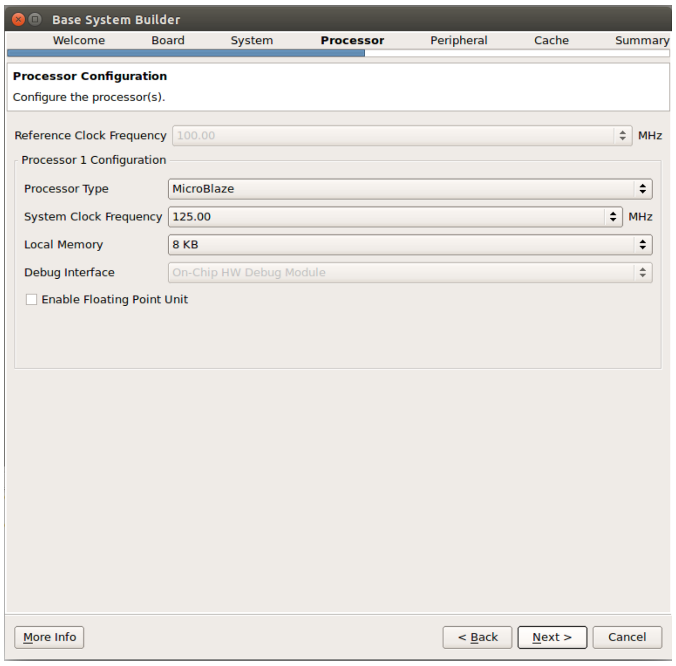
Select this option to create a design with a single processor. This Wizard will let you configure the processor, the peripheral set and some major configuration parameters for the peripherals.

( ) Dual-Processor System

Select this option to create a design with two processors. This Wizard will let you select the types of processors, peripherals unique to each processor, and peripherals shared by the processors.

Figure A.5: Building Hybrid CPU+FPGA System via Xilinx EDK based on single or multiple processors

The following CPU configuration is to be selected.



Processor Configuration
Configure the processor(s).

Reference Clock Frequency 100.00 MHz

Processor 1 Configuration

Processor Type	MicroBlaze
System Clock Frequency	125.00 MHz
Local Memory	8 KB
Debug Interface	On-Chip HW Debug Module

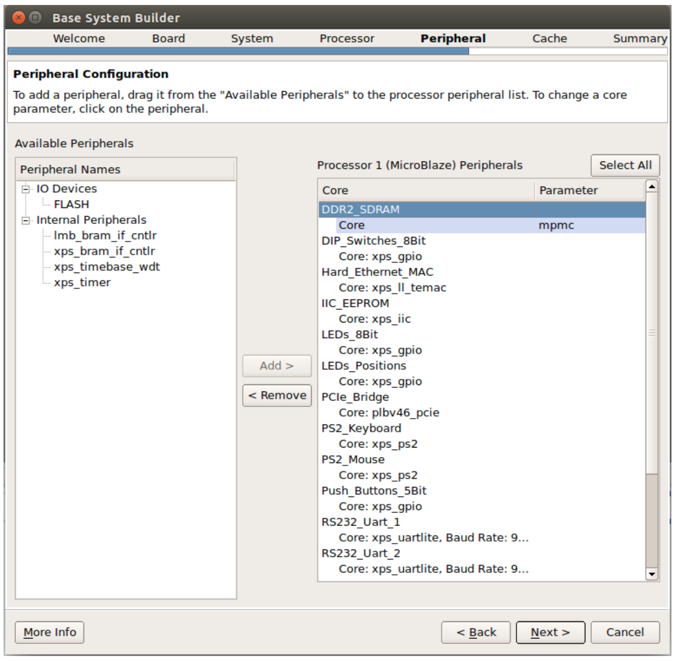
☐ Enable Floating Point Unit

Figure A.6: Building Hybrid CPU+FPGA System via Xilinx EDK - Processor configuration

Then, extra peripherals can be added to the solution and those who are not relevant for the application can be removed, so as to save the usage of FPGA's logic elements.

In this case, it is not relevant to enable the microprocessors cache, therefore, both options of the menu in the figure A.8 are left disabled.

After the last step, the EDA tool is configured to generate an Hybrid CPU + FPGA for this board.



Base System Builder

Welcome | Board | System | Processor | **Peripheral** | Cache | Summary

Peripheral Configuration

To add a peripheral, drag it from the "Available Peripherals" to the processor peripheral list. To change a core parameter, click on the peripheral.

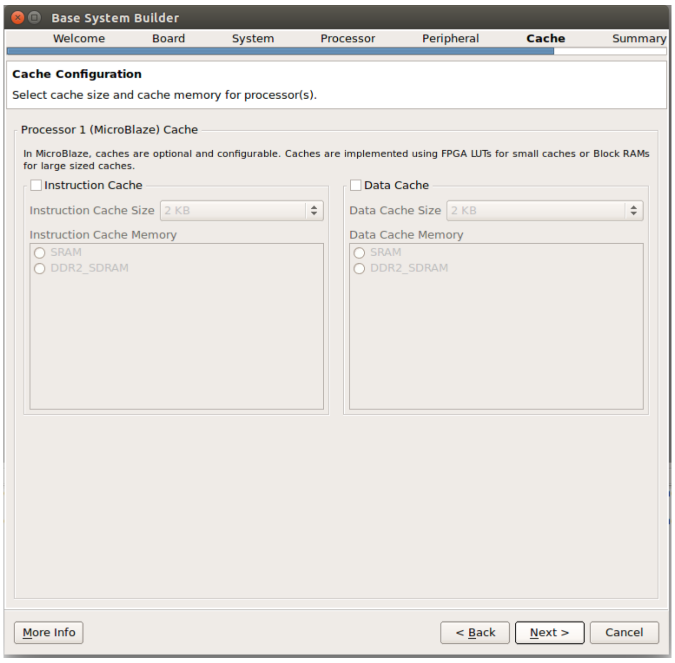
Available Peripherals

Peripheral Names
- IO Devices
  - FLASH
- Internal Peripherals
  - lmb_bram_if_cntlr
  - xps_bram_if_cntlr
  - xps_timebase_wdt
  - xps_timer

Processor 1 (MicroBlaze) Peripherals

Core	Parameter
DDR2_SDRAM	
Core	mpmc
DIP_Switches_8Bit	
Core: xps_gpio	
Hard_Ethernet_MAC	
Core: xps_ll_temac	
IIC_EEPROM	
Core: xps_iic	
LEDs_8Bit	
Core: xps_gpio	
LEDs_Positions	
Core: xps_gpio	
PCIe_Bridge	
Core: plbv46_pcie	
PS2_Keyboard	
Core: xps_ps2	
PS2_Mouse	
Core: xps_ps2	
Push_Buttons_5Bit	
Core: xps_gpio	
RS232_Uart_1	
Core: xps_uartlite, Baud Rate: 9...	
RS232_Uart_2	
Core: xps_uartlite, Baud Rate: 9...	

Figure A.7: Building Hybrid CPU+FPGA System via Xilinx EDK - Selecting peripherals for the application



Base System Builder

Welcome | Board | System | Processor | Peripheral | **Cache** | Summary

Cache Configuration

Select cache size and cache memory for processor(s).

Processor 1 (MicroBlaze) Cache

In MicroBlaze, caches are optional and configurable. Caches are implemented using FPGA LUTs for small caches or Block RAMs for large sized caches.

☐ Instruction Cache

Instruction Cache Size: 2 KB

Instruction Cache Memory: ○ SRAM ○ DDR2_SDRAM

☐ Data Cache

Data Cache Size: 2 KB

Data Cache Memory: ○ SRAM ○ DDR2_SDRAM

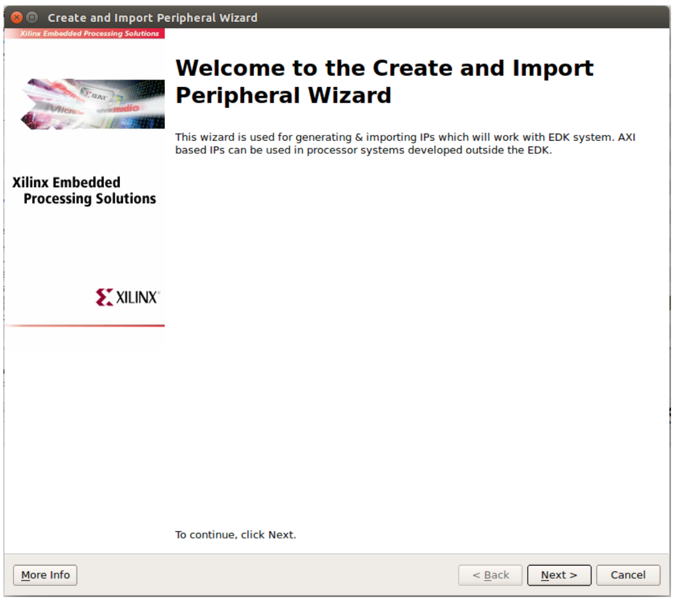
Figure A.8: Building Hybrid CPU+FPGA System via Xilinx EDK - Processor cache features

Appendix B

Integration of Custom IP in the Hybrid CPU+FPGA System

B.1 Creating peripheral template for custom IP integration

The Xilinx Platform Studio also helps the integration of custom IPs in the generated design. The wizard represented in the figure B.1 must be performed as per the instructions included in this chapter, in order to get the desired effect.


Create and Import Peripheral Wizard
Xilinx Embedded Processing Solutions
XILINX
More Info
< Back
Next >
Cancel

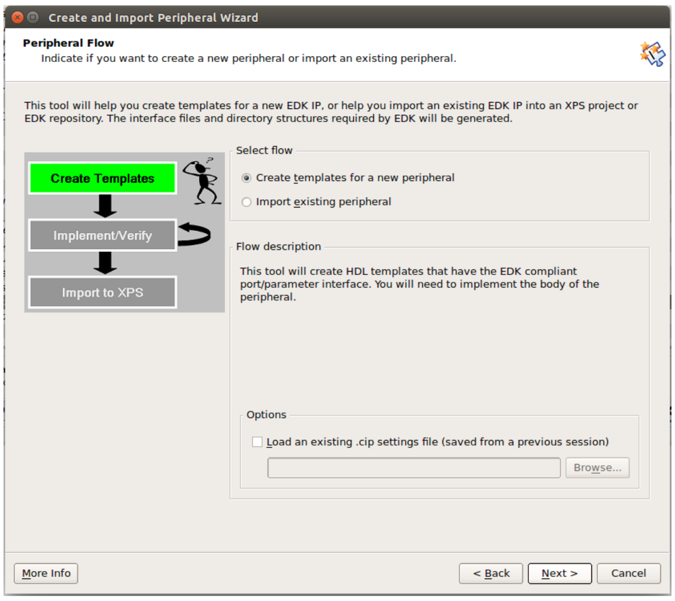
Welcome to the Create and Import Peripheral Wizard

This wizard is used for generating & importing IPs which will work with EDK system. AXI based IPs can be used in processor systems developed outside the EDK.

To continue, click Next.

Figure B.1: Creating peripheral template for custom IP integration - Initial menu

Firstly, the aim is to create a template compliant with the system design that has been generated in the previous chapter, that will be tailored in order to accommodate the custom IP to be integrated in the design.


Create and Import Peripheral Wizard

Peripheral Flow
Indicate if you want to create a new peripheral or import an existing peripheral.

This tool will help you create templates for a new EDK IP, or help you import an existing EDK IP into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

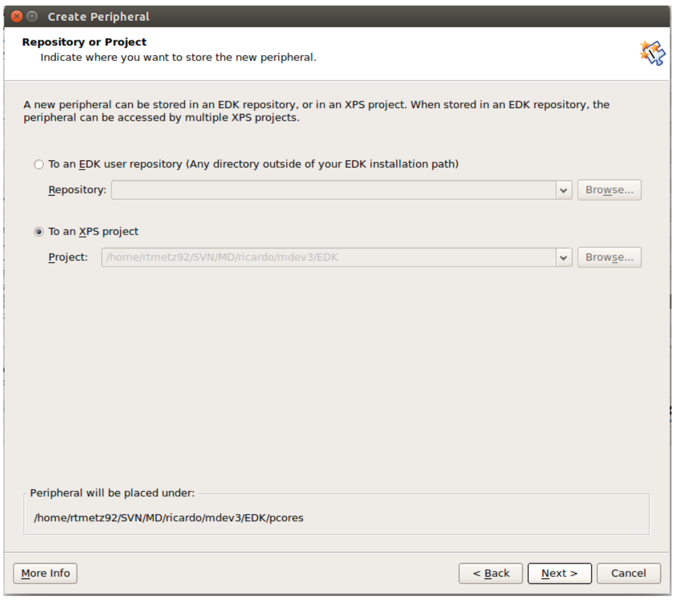
Select flow
[✓ Create templates for a new peripheral]
Import existing peripheral

Flow description
This tool will create HDL templates that have the EDK compliant port/parameter interface. You will need to implement the body of the peripheral.

Options
Load an existing .cip settings file (saved from a previous session)

Figure B.2: Creating peripheral template for custom IP integration

In this case, the choice is to add the peripheral to a local XPS project.


Create Peripheral

Repository or Project
Indicate where you want to store the new peripheral.

A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK repository, the peripheral can be accessed by multiple XPS projects.

To an EDK user repository (Any directory outside of your EDK installation path)
Repository:

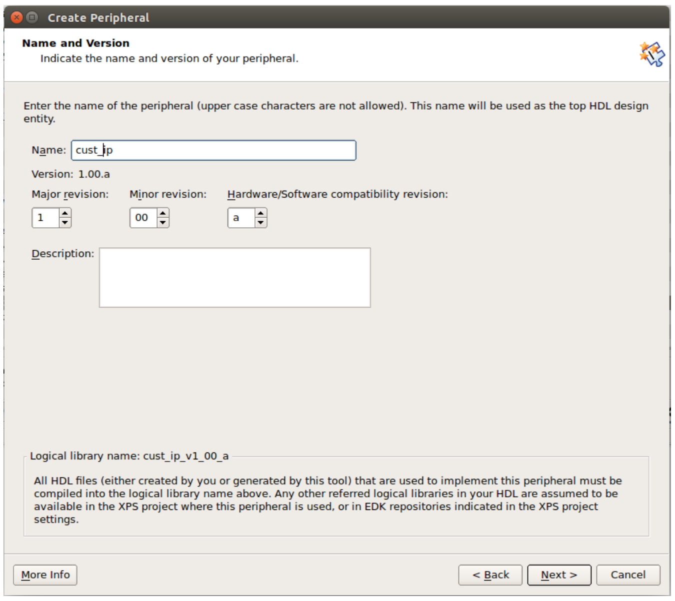
[✓ To an XPS project]
Project: /home/rtmetz92/SVN/MD/ricardo/mdev3/EDK

Peripheral will be placed under:
/home/rtmetz92/SVN/MD/ricardo/mdev3/EDK/pcores

Figure B.3: Creating peripheral template for custom IP integration locally

After that, a name is assigned to the IP to be integrated.

And the Interconnect Interface is selected, which in this case will be the PLB.


Create Peripheral

Name and Version

Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.

Name: cust_ip

Version: 1.00.a

Major revision: 1

Minor revision: 00

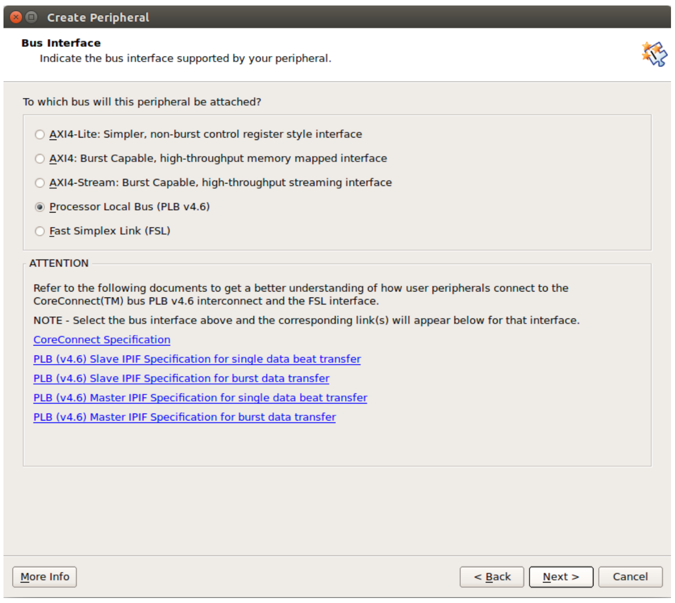
Hardware/Software compatibility revision: a

Description:

Logical library name: cust_ip_v1_00_a

All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library name above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.

Figure B.4: Creating peripheral template for custom IP integration - Assigning a name to the peripheral


Create Peripheral

Bus Interface

Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

- ○ AXI4-Lite: Simpler, non-burst control register style interface
- ○ AXI4: Burst Capable, high-throughput memory mapped interface
- ○ AXI4-Stream: Burst Capable, high-throughput streaming interface
- ● Processor Local Bus (PLB v4.6)
- ○ Fast Simplex Link (FSL)

ATTENTION

Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) bus PLB v4.6 interconnect and the FSL interface.

NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface.

CoreConnect Specification

PLB (v4.6) Slave IPIF Specification for single data beat transfer

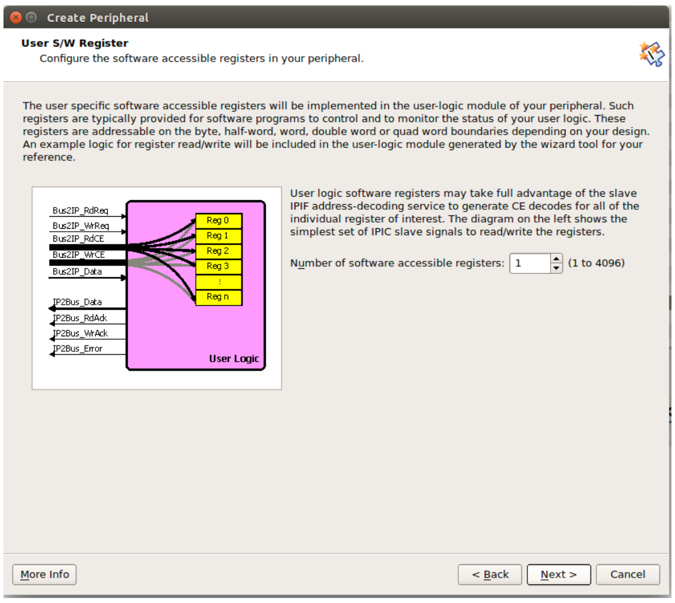
PLB (v4.6) Slave IPIF Specification for burst data transfer

PLB (v4.6) Master IPIF Specification for single data beat transfer

PLB (v4.6) Master IPIF Specification for burst data transfer

Figure B.5: Creating peripheral template for custom IP integration - Interconnect Bus selection

The custom IP and the software application, run in the CPU, communicate with each other by means of registers. A number of them can be selected according to the user's needs.



Create Peripheral

User S/W Register
Configure the software accessible registers in your peripheral.

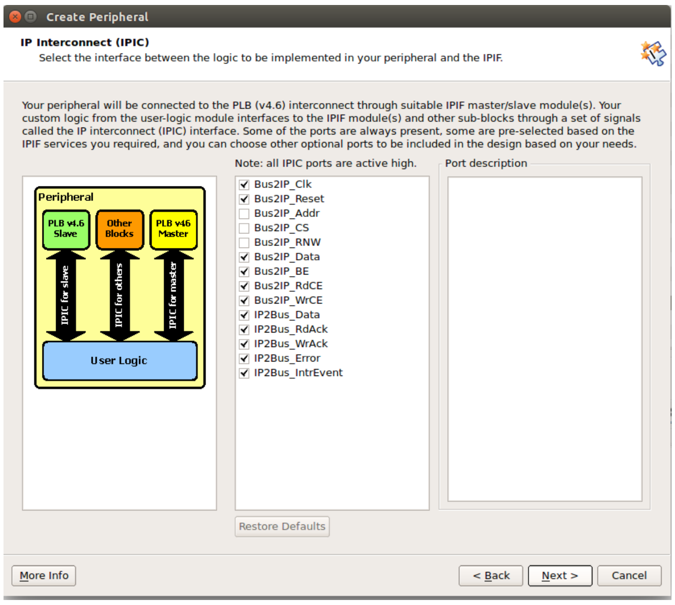
The user specific software accessible registers will be implemented in the user-logic module of your peripheral. Such registers are typically provided for software programs to control and to monitor the status of your user logic. These registers are addressable on the byte, half-word, word, double word or quad word boundaries depending on your design. An example logic for register read/write will be included in the user-logic module generated by the wizard tool for your reference.

User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIC slave signals to read/write the registers.

Number of software accessible registers: 1 (1 to 4096)

Figure B.6: Creating peripheral template for custom IP integration - allocating registers for communication between CPU and custom IP core

In this case, the pre-selected PLB ports are enough for this application.



Create Peripheral

IP Interconnect (IPIC)
Select the interface between the logic to be implemented in your peripheral and the IPIF.

Your peripheral will be connected to the PLB (v4.6) interconnect through suitable IPIF master/slave module(s). Your custom logic from the user-logic module interfaces to the IPIF module(s) and other sub-blocks through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present, some are pre-selected based on the IPIF services you required, and you can choose other optional ports to be included in the design based on your needs.

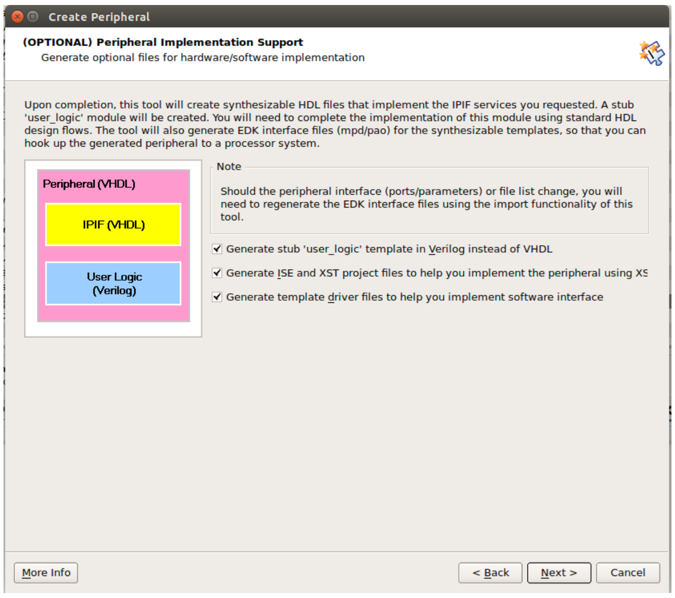
Note: all IPIC ports are active high.

- ✓ Bus2IP_Clk
- ✓ Bus2IP_Reset
- ☐ Bus2IP_Addr
- ☐ Bus2IP_CS
- ☐ Bus2IP_RNW
- ✓ Bus2IP_Data
- ✓ Bus2IP_BE
- ✓ Bus2IP_RdCE
- ✓ Bus2IP_WrCE
- ✓ IP2Bus_Data
- ✓ IP2Bus_RdAck
- ✓ IP2Bus_WrAck
- ✓ IP2Bus_Error
- ✓ IP2Bus_IntrEvent

Port description

Figure B.7: Creating peripheral template for custom IP integration - PLB ports

Provided that the custom IP was developed in Verilog and the automatic generation of drivers to access the PLB registers is desired, the configuration shown in the figure B.8 is selected.


Create Peripheral

(OPTIONAL) Peripheral Implementation Support

Generate optional files for hardware/software implementation

Upon completion, this tool will create synthesizable HDL files that implement the IPIF services you requested. A stub 'user_logic' module will be created. You will need to complete the implementation of this module using standard HDL design flows. The tool will also generate EDK interface files (mpd/pao) for the synthesizable templates, so that you can hook up the generated peripheral to a processor system.

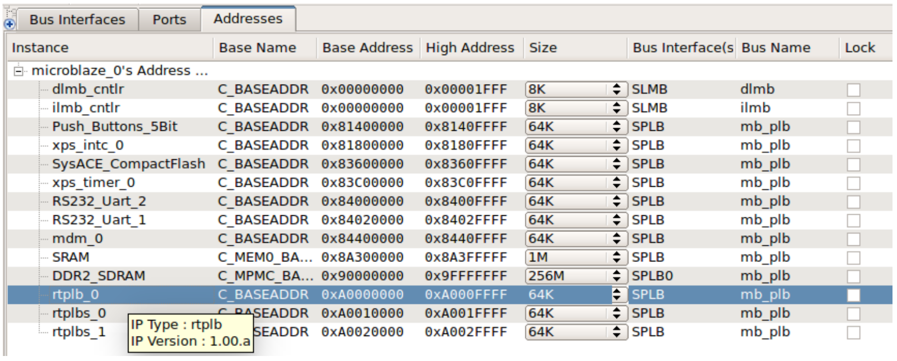
Note

Should the peripheral interface (ports/parameters) or file list change, you will need to regenerate the EDK interface files using the import functionality of this tool.

- [✓] Generate stub 'user_logic' template in Verilog instead of VHDL
- [✓] Generate ISE and XST project files to help you implement the peripheral using XS
- [✓] Generate template driver files to help you implement software interface

Figure B.8: Creating peripheral template for custom IP integration - automatic generation of templates for software drivers and HDL source code

Finally, the peripheral template is created and shall be instantiated in the RTL design, by means of assigning the range of memory addresses the register are going to be allocated in.



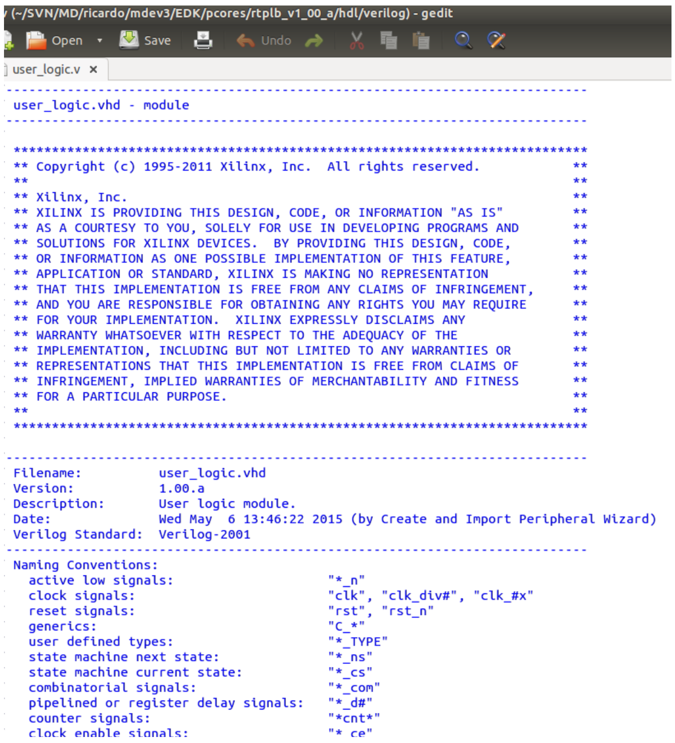
Instance	Base Name	Base Address	High Address	Size	Bus Interface(s	Bus Name	Lock
microblaze_0's Address ...							
dlmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	dlmb	☐
ilmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	ilmb	☐
Push_Buttons_5Bit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	mb_plb	☐
xps_intc_0	C_BASEADDR	0x81800000	0x8180FFFF	64K	SPLB	mb_plb	☐
SysACE_CompactFlash	C_BASEADDR	0x83600000	0x8360FFFF	64K	SPLB	mb_plb	☐
xps_timer_0	C_BASEADDR	0x83C00000	0x83C0FFFF	64K	SPLB	mb_plb	☐
RS232_Uart_2	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb	☐
RS232_Uart_1	C_BASEADDR	0x84020000	0x8402FFFF	64K	SPLB	mb_plb	☐
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	☐
SRAM	C_MEM0_BA...	0x8A300000	0x8A3FFFFF	1M	SPLB	mb_plb	☐
DDR2_SDRAM	C_MPMC_BA...	0x90000000	0x9FFFFFFF	256M	SPLB0	mb_plb	☐
rtplb_0	C_BASEADDR	0xA0000000	0xA000FFFF	64K	SPLB	mb_plb	☐
rtplbs_0	C_BASEADDR	0xA0010000	0xA001FFFF	64K	SPLB	mb_plb	☐
rtplbs_1	SEADDR	0xA0020000	0xA002FFFF	64K	SPLB	mb_plb	☐

IP Type : rtplb
IP Version : 1.00.a

Figure B.9: Creating peripheral template for custom IP integration - mapping newly created peripheral

B.2 Tailoring peripheral template to accommodate custom IP

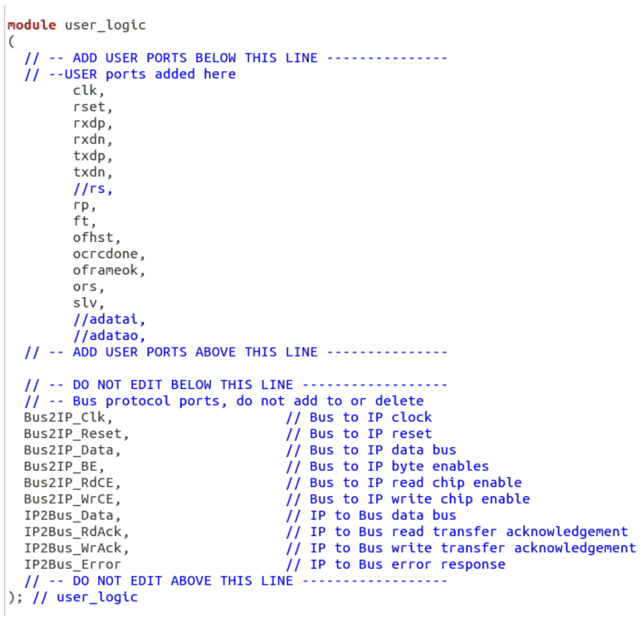
Part of the previous steps, the file user_logic.v has been generated in the directory under which the project was saved.


(~/SVN/MD/ricardo/mdev3/EDK/pcores/rtplb_v1_00_a/hdl/verilog) - gedit
Open
Save
Undo
user_logic.v

```
//----------------------------------------------------------------------------
// user_logic.vhd - module
//----------------------------------------------------------------------------
//
// ***************************************************************************
// ** Copyright (c) 1995-2011 Xilinx, Inc.  All rights reserved.            **
// **                                                                       **
// ** Xilinx, Inc.                                                          **
// ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
// ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
// ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
// ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
// ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
// ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
// ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
// ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
// ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
// ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
// ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
// ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
// ** FOR A PARTICULAR PURPOSE.                                             **
// **                                                                       **
// ***************************************************************************
//
//----------------------------------------------------------------------------
// Filename:          user_logic.vhd
// Version:           1.00.a
// Description:       User logic module.
// Date:              Wed May  6 13:46:22 2015 (by Create and Import Peripheral Wizard)
// Verilog Standard:  Verilog-2001
//----------------------------------------------------------------------------
// Naming Conventions:
//   active low signals:                    "*_n"
//   clock signals:                         "clk", "clk_div#", "clk_#x"
//   reset signals:                         "rst", "rst_n"
//   generics:                              "C_*"
//   user defined types:                    "*_TYPE"
//   state machine next state:              "*_ns"
//   state machine current state:           "*_cs"
//   combinatorial signals:                 "*_com"
//   pipelined or register delay signals:   "*_d#"
//   counter signals:                       "*cnt*"
//   clock enable signals:                  "*_ce"
```

Figure B.10: File to which custom IP's source code is inserted

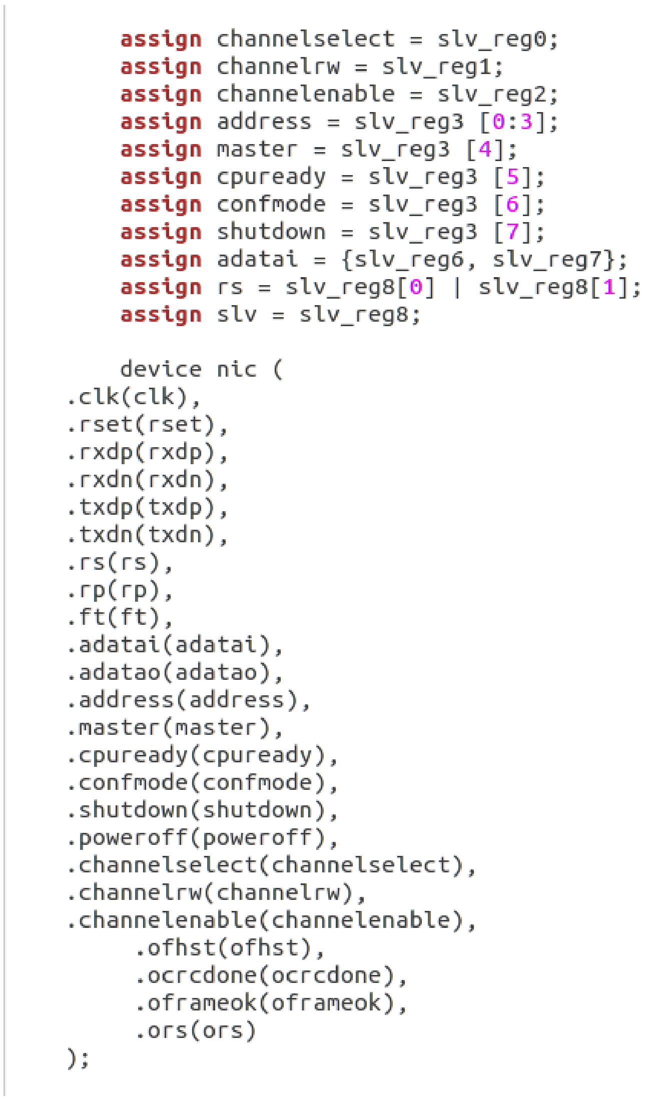
Shall the custom IP contain input and output ports that need to interact with the exterior of the PLB infrastructure, such ports can be declared in the header of the top module called user_logic.



```
module user_logic
(
  // -- ADD USER PORTS BELOW THIS LINE ---------------
  // --USER ports added here
        clk,
        rset,
        rxdp,
        rxdn,
        txdp,
        txdn,
        //rs,
        rp,
        ft,
        ofhst,
        ocrcdone,
        oframeok,
        ors,
        slv,
        //adatai,
        //adatao,
  // -- ADD USER PORTS ABOVE THIS LINE ---------------

  // -- DO NOT EDIT BELOW THIS LINE ------------------
  // -- Bus protocol ports, do not add to or delete
  Bus2IP_Clk,                     // Bus to IP clock
  Bus2IP_Reset,                   // Bus to IP reset
  Bus2IP_Data,                    // Bus to IP data bus
  Bus2IP_BE,                      // Bus to IP byte enables
  Bus2IP_RdCE,                    // Bus to IP read chip enable
  Bus2IP_WrCE,                    // Bus to IP write chip enable
  IP2Bus_Data,                    // IP to Bus data bus
  IP2Bus_RdAck,                   // IP to Bus read transfer acknowledgement
  IP2Bus_WrAck,                   // IP to Bus write transfer acknowledgement
  IP2Bus_Error                    // IP to Bus error response
  // -- DO NOT EDIT ABOVE THIS LINE ------------------
); // user_logic
```

Figure B.11: Custom IP's I/O ports to communicate with the rest of the system design

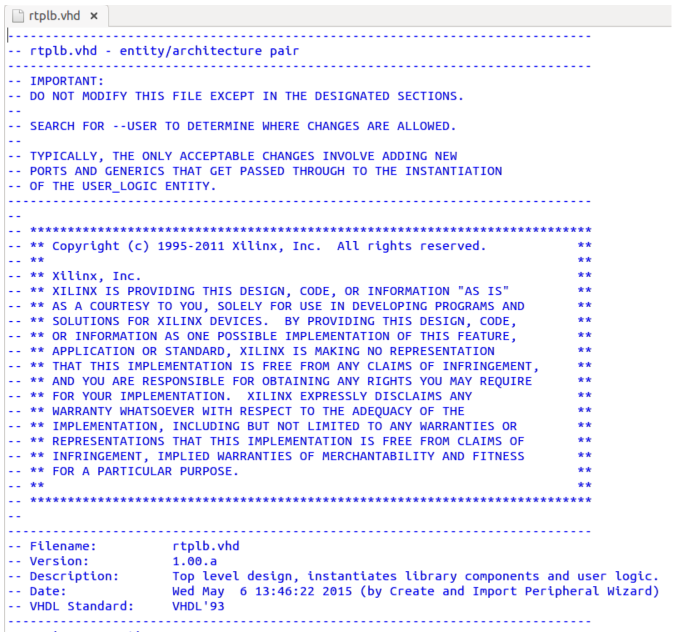
Inside that module, the custom IP module can be instantiated - in this case the module called device, of which its instance is aliased as nic. Note that in the same source file, the RTL code of the actual custom IP must be present.



```
    assign channelselect = slv_reg0;
    assign channelrw = slv_reg1;
    assign channelenable = slv_reg2;
    assign address = slv_reg3 [0:3];
    assign master = slv_reg3 [4];
    assign cpuready = slv_reg3 [5];
    assign confmode = slv_reg3 [6];
    assign shutdown = slv_reg3 [7];
    assign adatai = {slv_reg6, slv_reg7};
    assign rs = slv_reg8[0] | slv_reg8[1];
    assign slv = slv_reg8;

    device nic (
.clk(clk),
.rset(rset),
.rxdp(rxdp),
.rxdn(rxdn),
.txdp(txdp),
.txdn(txdn),
.rs(rs),
.rp(rp),
.ft(ft),
.adatai(adatai),
.adatao(adatao),
.address(address),
.master(master),
.cpuready(cpuready),
.confmode(confmode),
.shutdown(shutdown),
.poweroff(poweroff),
.channelselect(channelselect),
.channelrw(channelrw),
.channelenable(channelenable),
     .ofhst(ofhst),
     .ocrcdone(ocrcdone),
     .oframeok(oframeok),
     .ors(ors)
);
```

Figure B.12: Instantiation of the custom IP's module

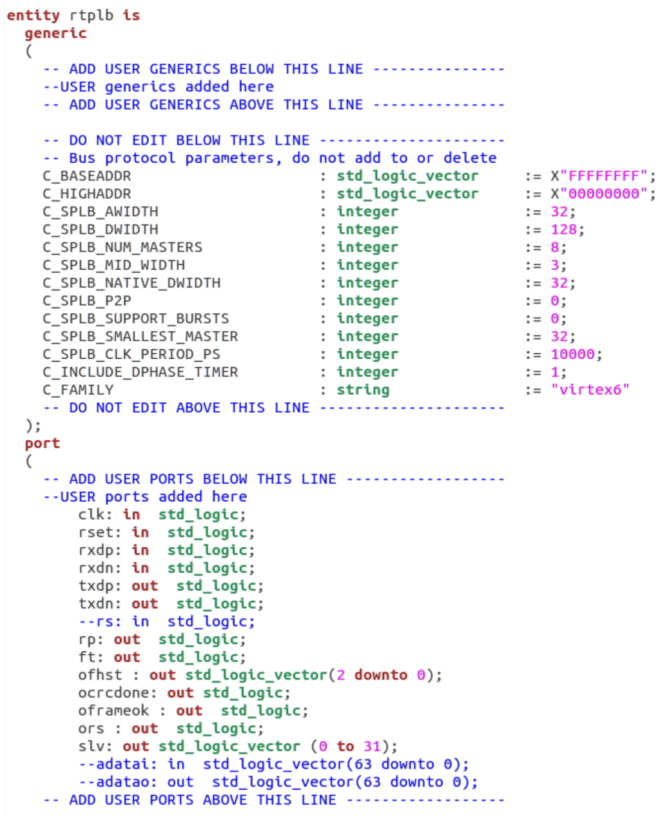
In a separate file, also generated by the EDA tool, the same procedures have to be done in case a custom IP contain input and output ports that need to interact with the exterior of the PLB infrastructure, as indicated in the figures B.13, B.14 and B.15.



rtplb.vhd ×

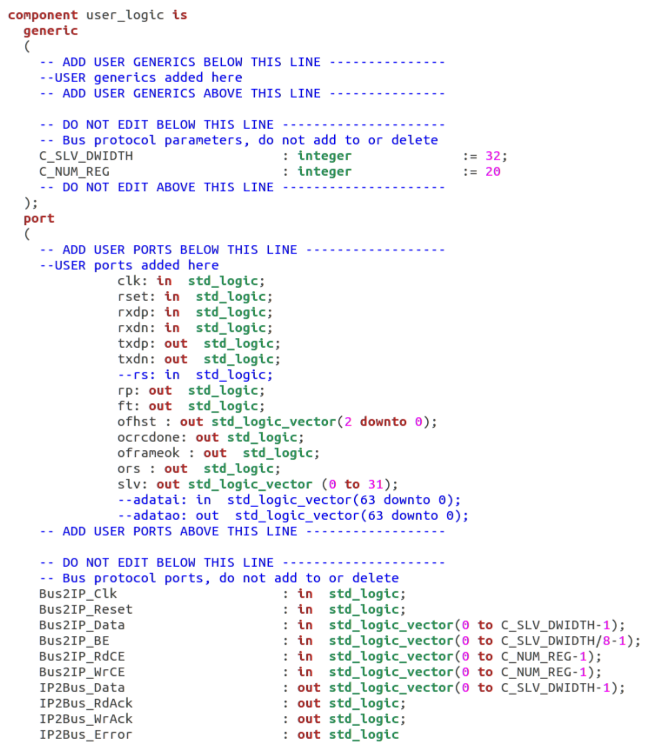
```
------------------------------------------------------------------------------
-- rtplb.vhd - entity/architecture pair
------------------------------------------------------------------------------
-- IMPORTANT:
-- DO NOT MODIFY THIS FILE EXCEPT IN THE DESIGNATED SECTIONS.
--
-- SEARCH FOR --USER TO DETERMINE WHERE CHANGES ARE ALLOWED.
--
-- TYPICALLY, THE ONLY ACCEPTABLE CHANGES INVOLVE ADDING NEW
-- PORTS AND GENERICS THAT GET PASSED THROUGH TO THE INSTANTIATION
-- OF THE USER_LOGIC ENTITY.
------------------------------------------------------------------------------
--
-- ***************************************************************************
-- ** Copyright (c) 1995-2011 Xilinx, Inc.  All rights reserved.            **
-- **                                                                       **
-- ** Xilinx, Inc.                                                          **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
-- ** FOR A PARTICULAR PURPOSE.                                             **
-- **                                                                       **
-- ***************************************************************************
--
------------------------------------------------------------------------------
-- Filename:          rtplb.vhd
-- Version:           1.00.a
-- Description:       Top level design, instantiates library components and user logic.
-- Date:              Wed May  6 13:46:22 2015 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
```

Figure B.13: Peripheral's top level design file



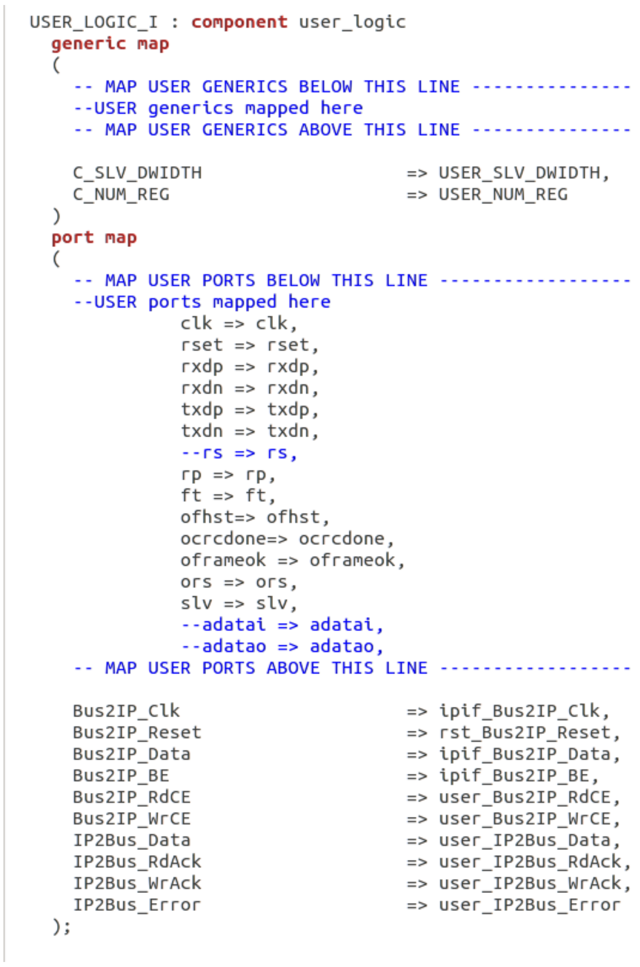
```
entity rtplb is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_BASEADDR                     : std_logic_vector     := X"FFFFFFFF";
    C_HIGHADDR                     : std_logic_vector     := X"00000000";
    C_SPLB_AWIDTH                  : integer              := 32;
    C_SPLB_DWIDTH                  : integer              := 128;
    C_SPLB_NUM_MASTERS             : integer              := 8;
    C_SPLB_MID_WIDTH               : integer              := 3;
    C_SPLB_NATIVE_DWIDTH           : integer              := 32;
    C_SPLB_P2P                     : integer              := 0;
    C_SPLB_SUPPORT_BURSTS          : integer              := 0;
    C_SPLB_SMALLEST_MASTER         : integer              := 32;
    C_SPLB_CLK_PERIOD_PS           : integer              := 10000;
    C_INCLUDE_DPHASE_TIMER         : integer              := 1;
    C_FAMILY                       : string               := "virtex6"
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
        clk: in  std_logic;
        rset: in  std_logic;
        rxdp: in  std_logic;
        rxdn: in  std_logic;
        txdp: out  std_logic;
        txdn: out  std_logic;
        --rs: in  std_logic;
        rp: out  std_logic;
        ft: out  std_logic;
        ofhst : out std_logic_vector(2 downto 0);
        ocrcdone: out std_logic;
        oframeok : out  std_logic;
        ors : out  std_logic;
        slv: out std_logic_vector (0 to 31);
        --adatai: in  std_logic_vector(63 downto 0);
        --adatao: out  std_logic_vector(63 downto 0);
    -- ADD USER PORTS ABOVE THIS LINE ------------------
```

Figure B.14: Allowing custom IP's I/O ports to interact with the exterior of the PLB infrastructure



```vhdl
component user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                   : integer              := 32;
    C_NUM_REG                      : integer              := 20
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
              clk: in  std_logic;
              rset: in  std_logic;
              rxdp: in  std_logic;
              rxdn: in  std_logic;
              txdp: out  std_logic;
              txdn: out  std_logic;
              --rs: in  std_logic;
              rp: out  std_logic;
              ft: out  std_logic;
              ofhst : out std_logic_vector(2 downto 0);
              ocrcdone: out std_logic;
              oframeok : out  std_logic;
              ors : out  std_logic;
              slv: out std_logic_vector (0 to 31);
              --adatai: in  std_logic_vector(63 downto 0);
              --adatao: out  std_logic_vector(63 downto 0);
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                     : in  std_logic;
    Bus2IP_Reset                   : in  std_logic;
    Bus2IP_Data                    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic
```

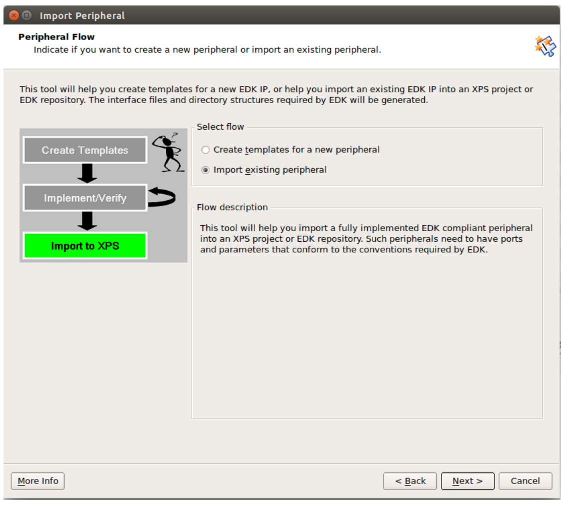
Figure B.15: Allowing custom IP's I/O ports to interact with the exterior of the PLB infrastructure



```
USER_LOGIC_I : component user_logic
  generic map
  (
    -- MAP USER GENERICS BELOW THIS LINE ---------------
    --USER generics mapped here
    -- MAP USER GENERICS ABOVE THIS LINE ---------------

    C_SLV_DWIDTH                   => USER_SLV_DWIDTH,
    C_NUM_REG                      => USER_NUM_REG
  )
  port map
  (
    -- MAP USER PORTS BELOW THIS LINE ------------------
    --USER ports mapped here
              clk => clk,
              rset => rset,
              rxdp => rxdp,
              rxdn => rxdn,
              txdp => txdp,
              txdn => txdn,
              --rs => rs,
              rp => rp,
              ft => ft,
              ofhst=> ofhst,
              ocrcdone=> ocrcdone,
              oframeok => oframeok,
              ors => ors,
              slv => slv,
              --adatai => adatai,
              --adatao => adatao,
    -- MAP USER PORTS ABOVE THIS LINE ------------------

    Bus2IP_Clk                     => ipif_Bus2IP_Clk,
    Bus2IP_Reset                   => rst_Bus2IP_Reset,
    Bus2IP_Data                    => ipif_Bus2IP_Data,
    Bus2IP_BE                      => ipif_Bus2IP_BE,
    Bus2IP_RdCE                    => user_Bus2IP_RdCE,
    Bus2IP_WrCE                    => user_Bus2IP_WrCE,
    IP2Bus_Data                    => user_IP2Bus_Data,
    IP2Bus_RdAck                   => user_IP2Bus_RdAck,
    IP2Bus_WrAck                   => user_IP2Bus_WrAck,
    IP2Bus_Error                   => user_IP2Bus_Error
  );
```

Figure B.16: Allowing custom IP's I/O ports to interact with the exterior of the PLB infrastructure

Having edited the source files, the Xilinx Platform Studio tool's information needs to be refreshed, as the peripheral template has been changed as a result of the tailoring process that has been carried out.



Import Peripheral

Peripheral Flow
Indicate if you want to create a new peripheral or import an existing peripheral.

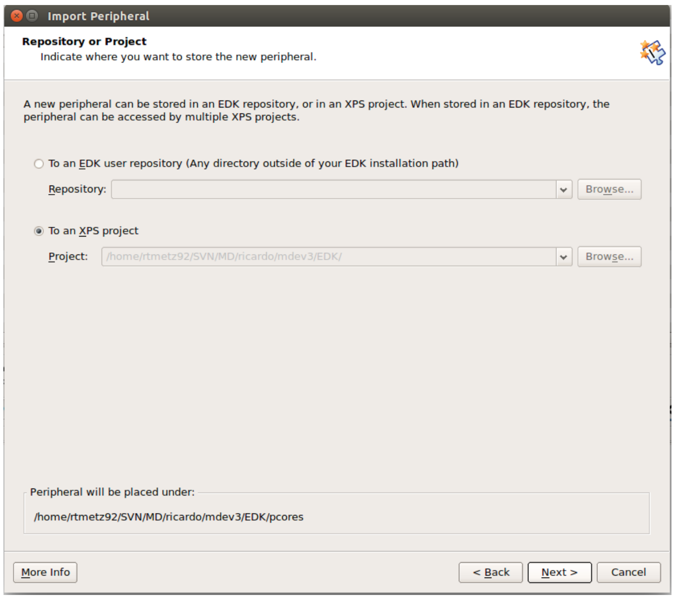
This tool will help you create templates for a new EDK IP, or help you import an existing EDK IP into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

Select flow
- ( ) Create templates for a new peripheral
- (●) Import existing peripheral

Flow description
This tool will help you import a fully implemented EDK compliant peripheral into an XPS project or EDK repository. Such peripherals need to have ports and parameters that conform to the conventions required by EDK.

Figure B.17: Updating peripheral specification

Similarly to the process of creating a peripheral template, the importation of the updated peripheral is meant to be stored locally.



Import Peripheral

Repository or Project
Indicate where you want to store the new peripheral.

A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK repository, the peripheral can be accessed by multiple XPS projects.

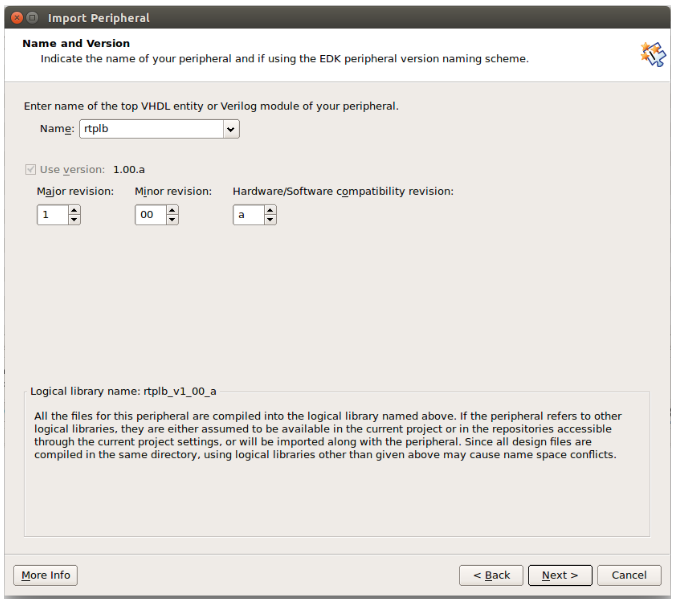
( ) To an EDK user repository (Any directory outside of your EDK installation path)
Repository:

(●) To an XPS project
Project: /home/rtmetz92/SVN/MD/ricardo/mdev3/EDK/

Peripheral will be placed under:
/home/rtmetz92/SVN/MD/ricardo/mdev3/EDK/pcores

Figure B.18: Updating peripheral specification in a previously created local peripheral

Besides that, the name of the peripheral must match the one that has been created in preceding steps.


Import Peripheral

Name and Version
Indicate the name of your peripheral and if using the EDK peripheral version naming scheme.

Enter name of the top VHDL entity or Verilog module of your peripheral.
Name: rtplb

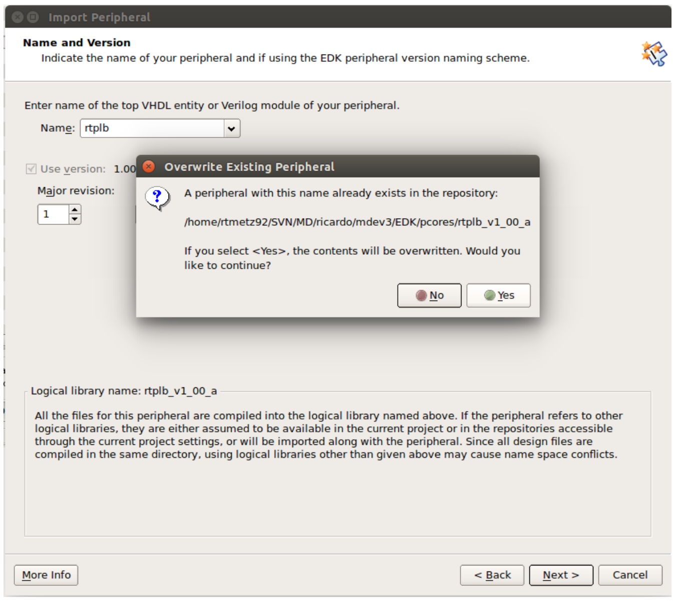
Use version: 1.00.a
Major revision: 1
Minor revision: 00
Hardware/Software compatibility revision: a

Logical library name: rtplb_v1_00_a

All the files for this peripheral are compiled into the logical library named above. If the peripheral refers to other logical libraries, they are either assumed to be available in the current project or in the repositories accessible through the current project settings, or will be imported along with the peripheral. Since all design files are compiled in the same directory, using logical libraries other than given above may cause name space conflicts.

Figure B.19: Inserting the same name of the existing peripheral to be updated

Provided that the name of the peripheral matches the one of the previously created model, one should confirm to overwrite the changes done in the meantime.


Import Peripheral

Name and Version
Indicate the name of your peripheral and if using the EDK peripheral version naming scheme.

Enter name of the top VHDL entity or Verilog module of your peripheral.
Name: rtplb

Use version: 1.00
Major revision: 1

Overwrite Existing Peripheral

A peripheral with this name already exists in the repository:

/home/rtmetz92/SVN/MD/ricardo/mdev3/EDK/pcores/rtplb_v1_00_a

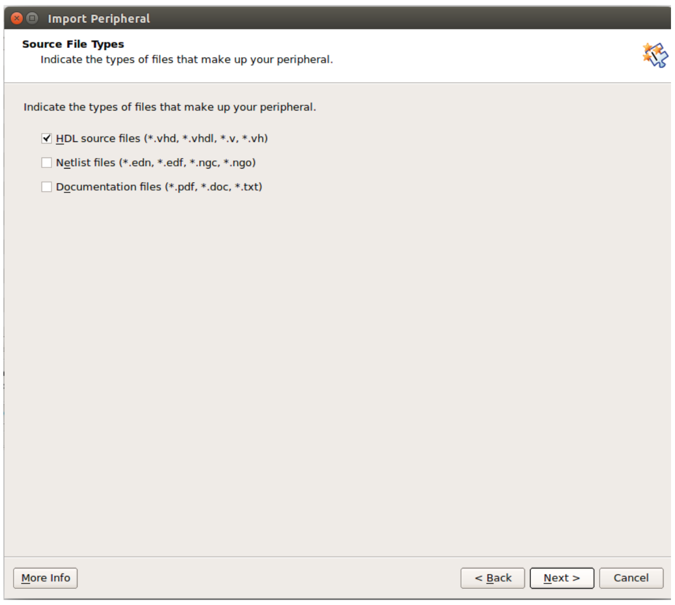
If you select <Yes>, the contents will be overwritten. Would you like to continue?

Logical library name: rtplb_v1_00_a

All the files for this peripheral are compiled into the logical library named above. If the peripheral refers to other logical libraries, they are either assumed to be available in the current project or in the repositories accessible through the current project settings, or will be imported along with the peripheral. Since all design files are compiled in the same directory, using logical libraries other than given above may cause name space conflicts.

Figure B.20: Confirmation to overwrite the performed changes

In this case, only the source files need to be imported.



Source File Types

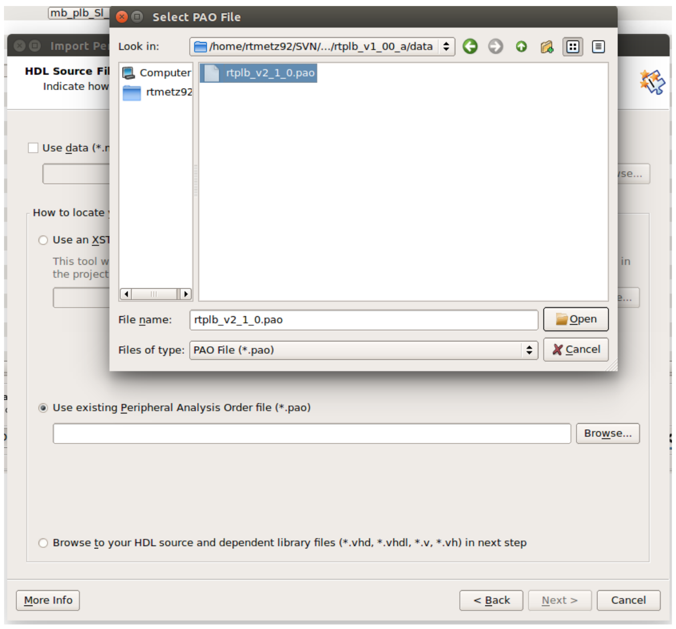
Indicate the types of files that make up your peripheral.

Indicate the types of files that make up your peripheral.

- [✓] HDL source files (*.vhd, *.vhdl, *.v, *.vh)
- [ ] Netlist files (*.edn, *.edf, *.ngc, *.ngo)
- [ ] Documentation files (*.pdf, *.doc, *.txt)

Figure B.21: Importing HDL source files

A file with the PAO extension, that also has been generated by the EDA tool and that corresponds to the current peripheral, need to be located and selected, so that the wizard can continue the process of updating the specification of the peripheral that holds the custom IP.



Select PAO File

Look in: /home/rtmetz92/SVN/.../rtplb_v1_00_a/data

Computer
rtmetz92
rtplb_v2_1_0.pao

File name: rtplb_v2_1_0.pao
Files of type: PAO File (*.pao)

Open
Cancel

Import Pe

HDL Source Fi
Indicate how

Use data (*.r

How to locate

Use an XST
This tool w
the project

Use existing Peripheral Analysis Order file (*.pao)

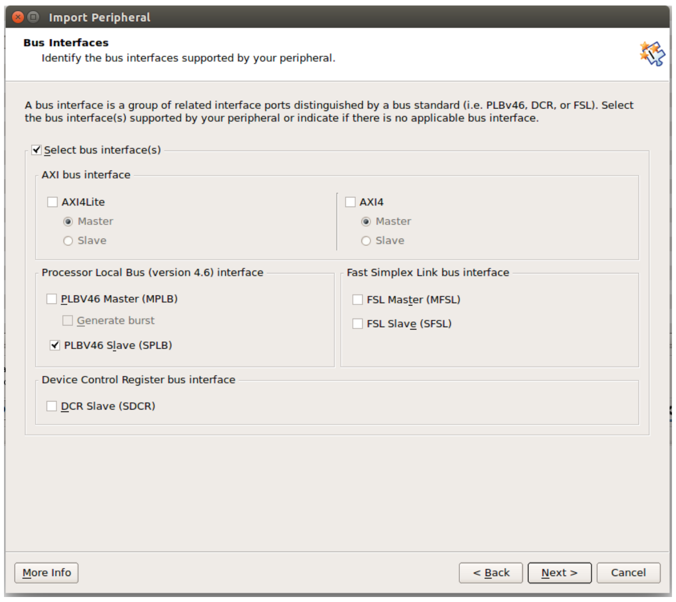
Browse...

Browse to your HDL source and dependent library files (*.vhd, *.vhdl, *.v, *.vh) in next step

More Info
< Back
Next >
Cancel

Figure B.22: Selecting Peripheral Order Analysis file

After that, the peripheral acts as a PLB slave, so the choice needs to be done accordingly.



Import Peripheral

Bus Interfaces
Identify the bus interfaces supported by your peripheral.

A bus interface is a group of related interface ports distinguished by a bus standard (i.e. PLBv46, DCR, or FSL). Select the bus interface(s) supported by your peripheral or indicate if there is no applicable bus interface.

[✓] Select bus interface(s)

AXI bus interface
- [ ] AXI4Lite — [● Master] ○ Slave
- [ ] AXI4 — [● Master] ○ Slave

Processor Local Bus (version 4.6) interface
- [ ] PLBV46 Master (MPLB)
  - [ ] Generate burst
- [✓] PLBV46 Slave (SPLB)

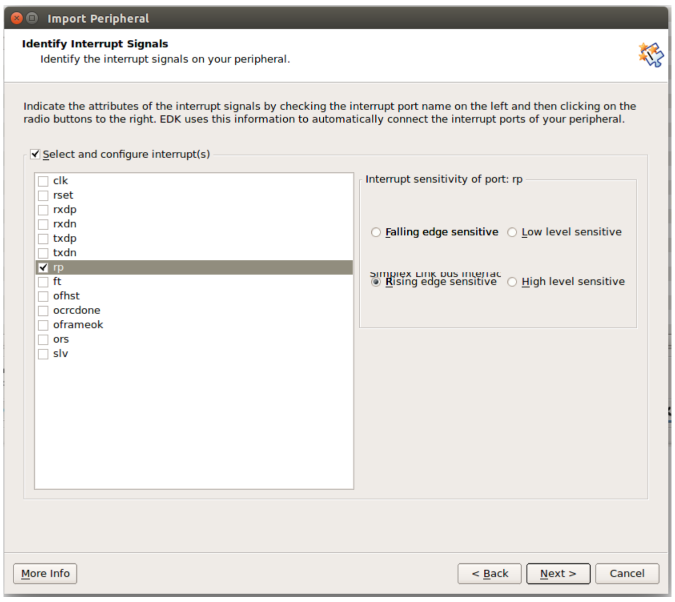
Fast Simplex Link bus interface
- [ ] FSL Master (MFSL)
- [ ] FSL Slave (SFSL)

Device Control Register bus interface
- [ ] DCR Slave (SDCR)

More Info
< Back
Next >
Cancel

Figure B.23: Setting peripheral as PLB slave

The list of I/O ports that interact with the outside part of the PLB infrastructure are listed in the wizard, as per the figure B.24. In case output ports of that peripheral are linked with an interrupt controller, its interrupt sensitivity can be selected. In this case, the signal's rising edge sensitivity is selected.


Import Peripheral

Identify Interrupt Signals
Identify the interrupt signals on your peripheral.

Indicate the attributes of the interrupt signals by checking the interrupt port name on the left and then clicking on the radio buttons to the right. EDK uses this information to automatically connect the interrupt ports of your peripheral.

[✓] Select and configure interrupt(s)

- [ ] clk
- [ ] rset
- [ ] rxdp
- [ ] rxdn
- [ ] txdp
- [ ] txdn
- [✓] rp
- [ ] ft
- [ ] ofhst
- [ ] ocrcdone
- [ ] oframeok
- [ ] ors
- [ ] slv

Interrupt sensitivity of port: rp

Falling edge sensitive   Low level sensitive
[✓ Rising edge sensitive]   High level sensitive

Figure B.24: Setting up interrupt sensitivity of custom IP ports

Finally, upon the wizard's steps completion, the custom IP has been imported into the overall system design. Having connected the custom IP's I/O ports accordingly to project's specification, requesting the Xilinx Platform Studio tool to generate the logic of the Hybrid CPU + FPGA platform, prior to developing the software application to be executed on the CPU side, the whole platform configuration is ready to be deployed to the designated board.

Appendix C

Implementation of FSM for controlling datagram reception, transmission and loading to application

```
//Frame Handling FSM
reg [2:0] fhst;
reg pd;
reg crcdone;
reg frameok;
reg [31:0] fcs;
reg fr;
reg dt;
reg td;
wire [`FS:0] sframe;
reg [`FS:0] rframe;

assign sframe [249] = shutdown;
assign sframe [248] = confmode;
assign sframe [247:160] = {adatai ,0};

assign poweroff = rframe [249];
assign adatao = rframe [247:168];
```

```
always @ (posedge clk or posedge rst)
 begin
 if(rst)
        begin
                dt <= 0;
        end
        else
        begin
                if (dtr == 1)
                begin
                        dt <= 1;
                end
                else if (td)
                begin
                        dt <= 0;
                end
        end
 end
always @ (posedge clk or posedge rst)
 begin
        if(rst)
        begin
                fhst <= 0;
                rp <= 0;
                td <= 0;
        end
        else
begin
case (fhst)
        `FHS0:
        begin
                if (pd)
                begin
                        fhst <= `FHS1;
                        rp <= 0;
```

```
          end
          else if(dt)
          begin
                    tframe [`FS:`FS-3] <= 4'b1010; //
                       preamble
                    tframe [`FS-4:0] <= sframe [`FS
                       -4:0];
                    fhst <= `FHS3;
                    td <= 1;
          end
end

`FHS1:
begin
          if (fr == 1)
          begin
                    fhst <= `FHS2;
          end
end

`FHS2:
begin
          if((frameok == 1) && (crcdone == 1))
          begin
                    fhst <= `FHS4;
          end
          else if((frameok == 0) && (crcdone == 1))
          begin
                    tframe [`FS:160] <= 0;
                    fhst <= `FHS3;
          end
end

`FHS3:
begin
          if (ft == 1)
          begin
```

```
                    fhst <= `FHS0;
                end
            end

`FHS4:
begin
if(rframe[247:160] != 0)
begin
            if((rframe[247:244] == address) || (rframe[247:244]
                == 4'b1111))
            begin
                //Parse rframe
                rp <= 1;
                //End Parse rframe
                if (rs == 1)
                begin
                tframe [`FS:`FS-3] <= 4'b1010; //preamble
                tframe [`FS-4:0] <= sframe [`FS-4:0];
                fhst <= `FHS5;
                end
            end
            else
            begin
                tframe [`FS:0] <= rframe [`FS:0];
                if (master == 1)
                begin
                    fhst <= `FHS0;
                end
                else
                begin
                    fhst <= `FHS3;
                end
            end
end
else
begin
            rp <= 1;
```

```
          if(rs == 1)
          begin
                    tframe [`FS:`FS-3] <= 4'b1010; //preamble
                    tframe [`FS-4:0] <= sframe [`FS-4:0];
                    fhst <= `FHS5;
          end
end
end

          `FHS5:
          begin
          td <= 0;
          if(crcdone == 1)
          begin
                    if(master == 1)
                    begin
                              tframe [31:0] <= fcs [31:0];
                              fhst <= `FHS0;
                    end
                    else
                    begin
                              tframe [31:0] <= fcs [31:0];
                              fhst <= `FHS3;
                    end
          end
          end
endcase
end
end
//End     Frame Handling FSM
```