CHALMERS

Implementation of NFS functionality

in a 10G Ethernet based embedded field recorder in the domain of professional video storage

Master of Science Thesis in Integrated Electronic Systems Design program

HOANG NGUYEN

Department of Computer Science and Engineering
Division of Computer Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, November 2014

Implementation of NFS functionality
in a 10G Ethernet based embedded field recorder in the domain of professional video storage

HOANG NGUYEN

Examiner: LARS SVENSSON

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, November 2014

Abstract

The objective of this thesis project is to integrate a hardware-based network file system (NFS) functionality module into an existing FPGA-based network file interface (NFI) system - a project currently under development at Technicolor Hannover. In order to support the NFS functionality, a data transfer mechanism between the NFI system on the 10Gigabit Ethernet interface board (10GE board) and the user interface board is also implemented. For these purposes, several hardware modules, software modifications, and relevant data transfer mechanism are introduced. The particular challenges to solve are the compatibility of the new modules with the existing system, and the limitation of the interface pins between the boards.

In the course of this thesis work, the project's system architecture is presented and analyzed. A new communication architecture for data transfer between the boards has been proposed and implemented with the support of the currently available DCR bus and I2C bus interface. Furthermore, NFS functionality module, together with the support modules (timeout detector, application wrapper) are also implemented within the NFI system. Last but not least, the software on the FPGA embedded processor (the PowerPC) was modified in order to support the operations of NFS functionality and data transfer mechanism. Several interrupt signals with their handling procedures were added to ensure proper functionality of the new enhanced system. All those modules and mechanism were conceptually designed and implemented using Xilinx ISE development tools and VHDL, and verified with Mentor ModelSim and Xilinx ChipScope. The underlying FPGA technology is Xilinx Virtex 5.

Keywords**:** *NFS functionality, data transfer mechanism, timeout detector, application wrapper, software enhancement.*

Acknowledgement

This report constitutes my Master of Science thesis at Chalmers University of Technology. The work has been done within the 2020 3D Media project of Technicolor company.

I would like to give special thanks to my supervisors at Technicolor, Thomas Brüne and especially Stefan Abeling, for their great supports to my work. Without them, probably this thesis would not be done.

I also acknowledge the guidance of my examiner at Chalmers, Lars Svensson, for my thesis management and documentation issues. Furthermore, I would like to give thanks to the following persons in Technicolor:

- Michael Drexler for software support.
- Klaus Gaedke – Laboratory Manager Image Processing, Carolin Schmaehl and Heike Diekmann – HR staffs, for non-technical issues.

Finally, I would like to acknowledge to Technicolor company for the nice working environment during my thesis work.

Table of Contents

Table of Figures

1 Introduction

Nowadays, the demand of media entertainment is increasing rapidly. Higher video quality requires more powerful cinematography systems.

Current film infrastructure – so called digital cinematography (DC) - is based on HD-SDI [1] device connections, allowing the maximum data transmission between film devices of 1,5 Gbps [2].

The foreseen problem is the bandwidth requirements of data transmission: for 3D productions with full-HD or higher in resolution, we need more than 3 Gbps. Furthermore, the current infrastructure has no advanced control for devices (e.g. it is impossible to control/configure the device remotely), nor full duplex picture streaming nor versatile meta-data treatment.

These shortcomings motivate the derivation of the new connection architecture: the high-speed Ethernet interface. The benefit of this usage is the cost and time saving, since the Ethernet interface is widely used in IT domain. Furthermore, with peak speeds up to 10 Gbps, even 100 Gbps on the horizon, together with the versatile protocol layers, the above connection speed disadvantages of current HD-SDI interface can be overcome.

The DC's next generation uses the advantages of existing Ethernet network instead of proprietary connections. Besides, it combines the control stream and data stream in the same physical connection. The proposed physical connection for the new DC is the 10 Gigabit Ethernet (10GE) interface, as in Figure 1.1.

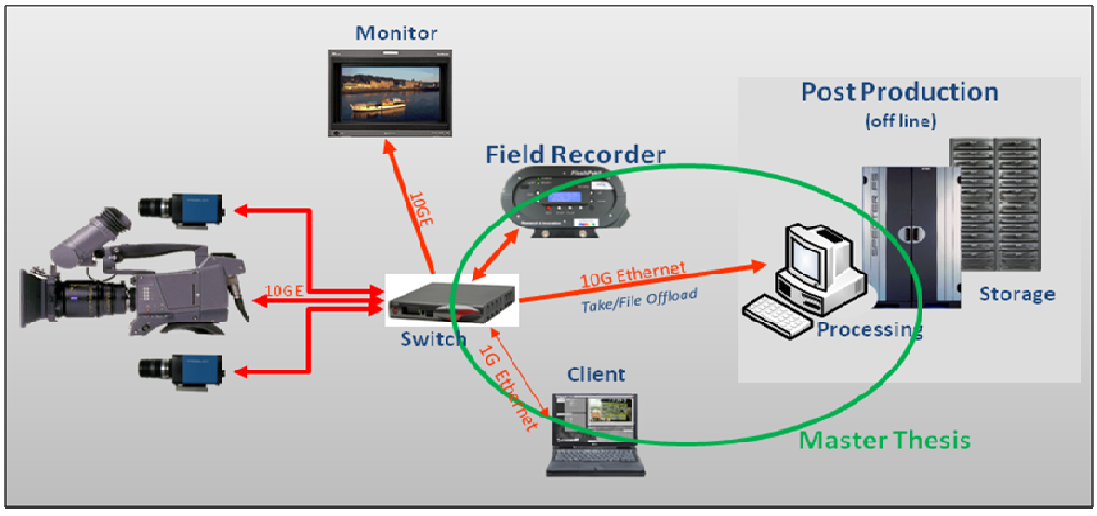

Figure 1.1 Digital cinematography based on 10 Gigabit Ethernet connection

Media data captured by a main camera and several satellite cameras will be transferred to the field recorder for intermediate storage (via record operation). This data can be reviewed on a

monitor (via playback operation), or can be downloaded to another storage device for further processing. These operations may be carried out either from the field recorder (record, playback) or from a client in the same network (record, playback, download).

In this architecture, all device interfaces and functional behaviors are based on Ethernet, with support of media data streaming as well as full device control in both directions at the same time. All devices are connected through an Ethernet switch.

2 Overview of the NFI-10GE system

In this thesis project, the work involves the field recorder, named "FlashPakII". Its function is to store "pure" video data, which comes directly from a trifocal camera, with multi-stream supported. It also manages data for user interface: playback to monitor, copy to post production, delete available data, so on. The field recorder is composed from three main components: the XGE board (for interface control and data streaming), the memory board (for data storage) and the user-interface board (for direct control by users). Figure 2.1 shows the opened field recorder.

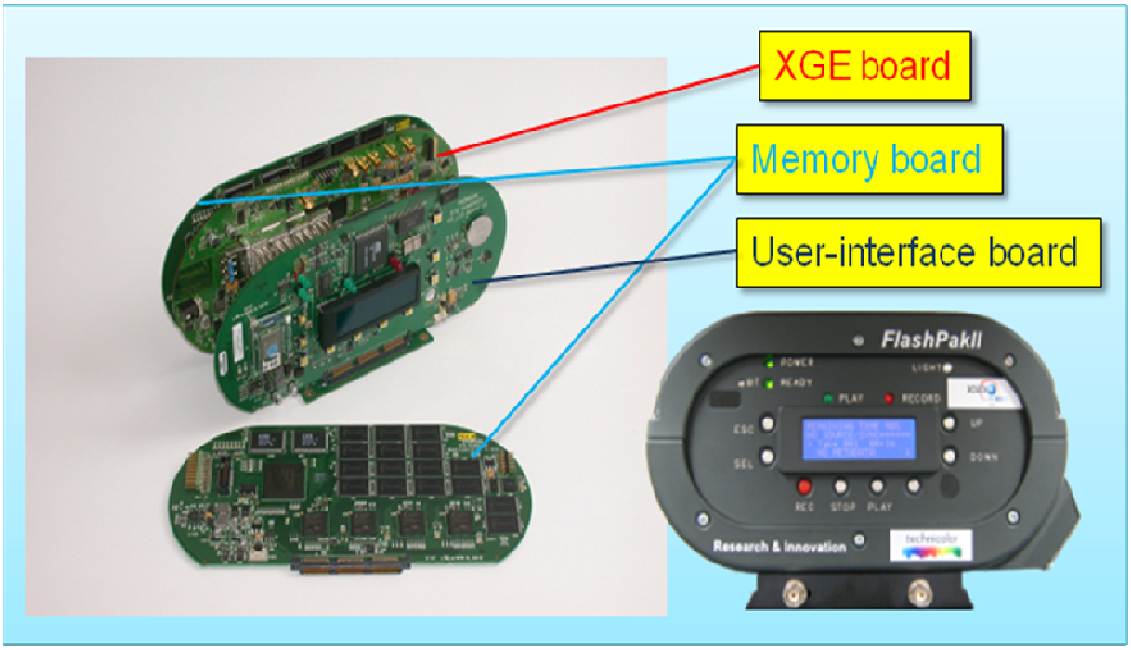

Figure 2.1 Structure of the Field Recorder

In the context of this thesis work, we will investigate in detail the XGE board, which is responsible for data transmission between the cameras (data source) and all other devices. The system in XGE board is contained in a FPGA chip, the Virtex-5 from Xilinx Inc, as seen in Figure 2.2. The task of this project is to enhance the functionality of this system.

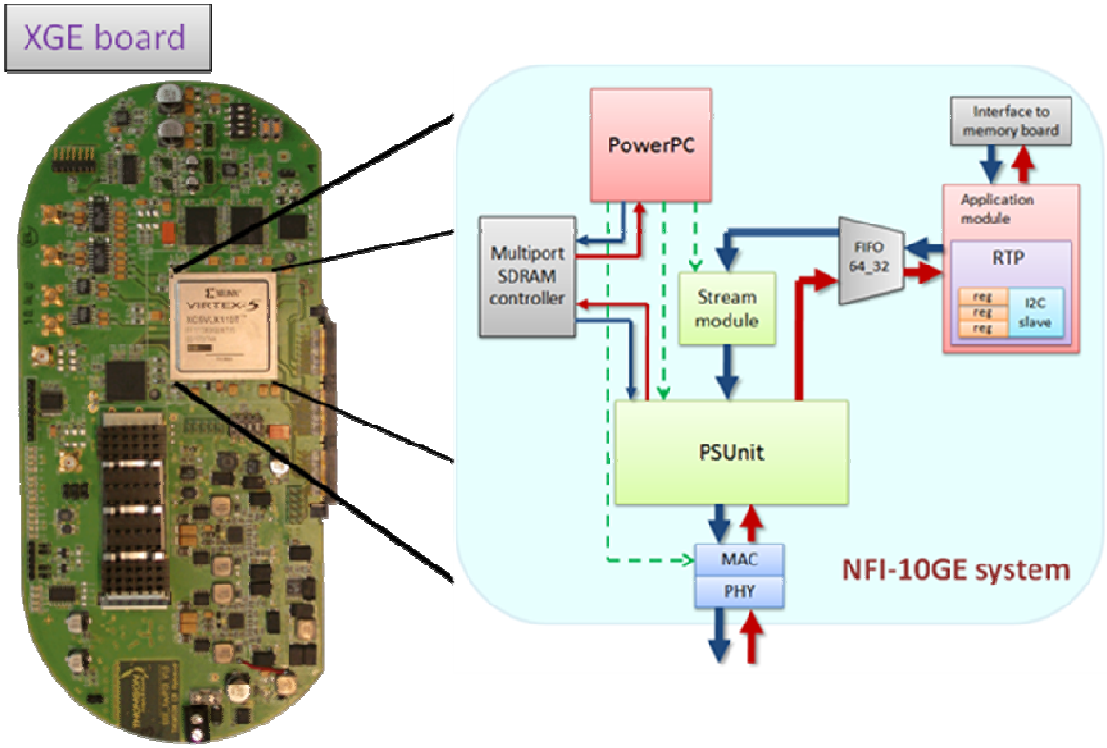

Figure 2.2 Architecture of XGE board

2.1 Architecture of the NFI-10GE system

This is the embedded soft- hardware system which resides on the XGE board. As indicated by its name, the Network File Interface – 10 Gigabit Ethernet based (NFI-10GE) processes all Ethernet and IP related protocol layers and responds to any requests, whether with video data or not, depending on the type of request. For the ease of processing management, data is routed through the real-time path (high throughput) or the non-real-time path (low throughput). Non-real-time data will be processed by the PowerPC microprocessor (software based) with a data rate up to 1 Gbps and mainly for control purpose, while real-time data will go through hardware modules up to 10 Gbps, and be able to transport AV-data due to the complete support of hardware UDP and IP protocol implementation.

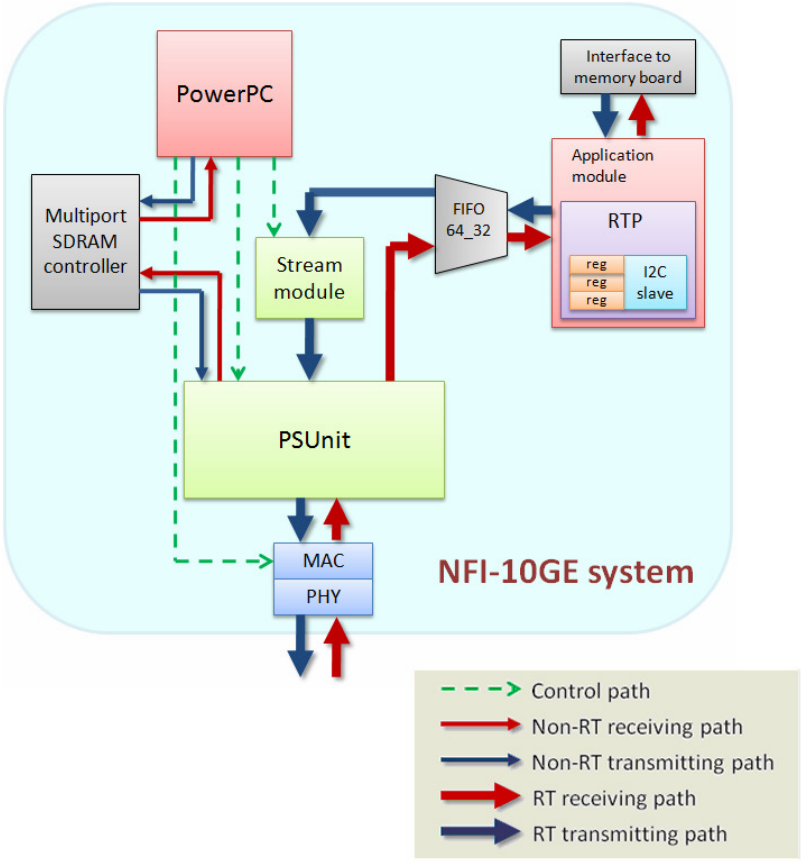

Figure 2.3 Module structure and data flow of NFI-10GE system on XGE board

In Figure 2.3, the structure and main data flow of the NFI-10GE system is sketched out. There are 6 main modules:

- *The PHY module*: implementing the physical layer of the OSI model [3], this module is in charge of receiving/transmitting data from/to XFP module to/from higher layers.
- *The MAC module*: implementing the data-link layer, this module is responsible for the Ethernet protocol. Ethernet header of packets is processed here.
- *Parser and Scheduler unit (PSUnit)*: in this unit the real-time-critical AV-data is separated from the non real-time-critical control data. For TX behavior, data from different sources (RT and non-RT) are joined into one unified stream to be transmitted. Additionally, UDP/IP header is de-packetized here. PSUnit is in charge of network layer and transport layer in the system.
- *Stream module*: this module does the UDP packetizing for transmitting data from the application unit, and then passes the packets to the PSUnit for further processing. Stream module implements the transport layer in the OSI model.
- *PowerPC*: this module acts as an embedded processor within the FPGA chip. In the NFI-10GE system, it processes all non-RT packets (e.g. ping) as determined by the

PSUnit. Furthermore, it takes the role as the system controller by configuring all other units via the device-control-register (DCR) bus [4].

- *Application module*: this module processes the incoming packets from the PSUnit. In this module, high throughput data operations are executed (RTP playback, RTP recording). Application module implements the application layer in the OSI model.

There are also some additional modules for intermediate purposes, such as a multiport SDRAM controller (for interfacing the on-board SDRAM to the PSUnit and the PowerPC), the memory board interface (for interfacing between memory controller board and the application module), and the FIFO64_32 module (for converting 64-bit data bus line to 32-bit data bus line and vice versa).

2.2 Hardware modules of the NFI-10GE system

Now we will have a deeper look on the hardware modules above.

2.2.1 PHY and MAC

Here the XAUI IP core is implemented, supporting four separated RX-TX lanes times 3.125 Gbps bandwidth connection to XFP transceiver.

Implementing the data-link layer, the MAC core contains a 10 Gigabit Ethernet MAC IP core compatible with IEEE 802.3ae-2002 (this IP core is available from Xilinx). It supports jumbo frames with data payload up to 9000 bytes. The peripheral of this IP core consists of a TX path with checksum calculation, a TX FIFO of 18KB, and a RX path. The FIFO is optimized for jumbo frames. The configuration of this module is done by the PowerPC via DCR bus interface.

2.2.2 Parser and Scheduler Unit (PSUnit)

This unit manages the data flow among the application module, the PowerPC module, the stream module, and the MAC module. In RX mode, it receives data packets from MAC with Ethernet header removed, and decides by parsing whether to use the non-real-time path to PowerPC (for software processing) or the real-time path to application module for data packet processing. In TX mode, this unit multiplexes data packets from PowerPC and stream module, and passes them to the MAC unit. The IP header of packets is formed/stripped in the PSUnit. The PSUnit control and status registers are accessible via DCR bus interface.

The possible data transfer directions through the PSUnit are:

- *The PowerPC is about to send non-real-time data (TCP/UDP packed):* The PowerPC builds a packet or a packet chain into a TX queue (frame buffer) and then signals the PSUnit to process the frame. The PSUnit fetches the frame from the TX queue and buffers it in the 9KB FIFO. Finally the scheduled MUX sends the frame with 1 Gbps to the 10GE MAC unit.

- *The PowerPC processes non-real-time data (TCP/UDP packed):* The received frames from client are parsed by the header-parser of the PSUnit to decide whether they are non-real-time or real-time frames. Non-real-time frames are written into the 108KB FIFO because it could be that the SDRAM controller has to wait some cycles to get access to the SDRAM.

- *The stream unit is about to send real-time UDP packets:* The stream unit writes the real-time frame into a FIFO of 18KB that is able to store two jumbo frames. The schedule MUX will send the frame with 10 Gbps to the 10GE MAC unit.

- *The application unit processes real-time UDP packets:* Frames that are parsed for real-time distribution to the application are immediately transmitted without caching mechanism to guarantee the data rate of about 10 Gbps.

2.2.3 Stream module

This module provides the high bandwidth stream up to 10 Gbps throughput towards the PSUnit. Stream module also has a DCR interface for configuration and initialization by the PowerPC. Its implementation supports a simplified IP layer (without options and without segmentation & reassembling and reordering capabilities) and an UDP layer. Data throughput of stream module has to be 10 Gbps.

2.2.4 Application module

Consisting of an RTP sub-module, the application module analyzes the payload data from the PSUnit and processes it in case it is an RTP packet. Before this project work, there were two paths of data through application module depending on its operation mode:

- Recording mode: RTP packet stream comes from the cameras after being removed IP/UDP header by PSUnit, will be stripped out the RTP header and transmitted to the memory board for storage.

- Playback mode: raw video data from memory board will be packetized in RTP format and transmitted out to a displayed device for viewing.

The user changes the operation mode via the user interface board by directly pressing the control buttons on the field recorder (PLAYBACK or RECORD). The user interface board communicates with the application module (also with the whole XGE board) by the I2C bus line [5].

2.2.5 PowerPC

This is an embedded processor within the FPGA chip, which supports the processing of non real-time packets. All types of non-real-time packets coming from other devices in the network will be routed to this processor. The PowerPC configures all hardware units via its DCR bus.

2.3 Software architecture of the NFI-10GE system

The software using in PowerPC is decided to be a light operational system since the tasks processed by PowerPC are few and very primitive; moreover, the processing speed is also a big concern as the data throughput of non-real-time path is much lower than real-time path.

2.3.1 NFS functionality

Network File System (NFS) [6] is a network protocol allowing a client computer to access files contained in a server (here: the NFI-10GE system) over a network in the same way as accessing local files. Several tasks need to be carried out to perform this file system access activity:

Mount to server: the first procedure is to establish the connection between the client and the server, thus making the file system on the server available to the client. This is done using the *mount* protocol. This protocol gives the client access to the server by providing to the client information about the server (e.g. the port number to which the client should send NFS packets) as well as information about the files residing on the server (e.g. the file handles which represent unique files) to provide the ability to access files in later procedures.

Get directory information: with the connection enabled, more detailed information about the files (i.e. file attributes) and about the directories (e.g. directory hierarchy) can be read. In the current version of the system software (before this thesis work), for simplicity, no hierarchy is used, i.e. all files are contained in the root directory. Via this procedure, the file size, file creation date, file accessibility (e.g. readable, writeable), etc… will be provided to the client. The procedure may be invoked by running the *ls* command from the client.

Download file contents to client: finally, the file contents can be downloaded to the client by running the *cp* command, the same way as a file-copy operation on local storage of a computer. In this operation, the copied file will be packetized into 4KB packets, and transferred to the client continuously.

2.3.2 File system structure

The file system contains information of all available files, and will be used for file management activities. In this file server (the NFI-10GE system), the file system is stored in SDRAM [7], and is managed by the software using in PowerPC. Information in the file system includes:

- *File name*: the name which will be displayed on screen.
- *File handle*: a unique code for each file.
- *File length*: the number if bytes occupied by a file
- *Block number*: the number of 512-byte blocks occupied by a file

Before this project work, the file system was simulated with two files in NFI-10GE system. They were called simulated files because only the file system was generated, but not the real file contents. Hence, the software-task of this project is to modify the available software to maintain a real file system when new real data is recorded into the server via a recording operation.

3 Environments for system development

This chapter will give the readers a fundamental background on the environments for the thesis work, from the chip on which the system is developed to the supporting tools for system implementation and verification.

3.1 FPGA

3.1.1 Overview

Simulation and prototyping have been very important parts of the electronic industry as they ensure the correctness of the product functionality before it is fabricated. These verification steps save much time and cost in a system development process. Among the most effective prototyping tools are FPGA chips, which combine advantages of programmability (like software/micro-processor) and fast signal processing (like ASICs).

A Field Programmable Gate Array (FPGA) is an integrated circuit which can be electrically configured to implement different functionalities. To this end, the FPGA chip contains many identical logic blocks, each can be configured to implement an independent logic function. All these blocks can be interconnected by programmable interconnect network, so that the user may generate different functions by setting a dedicated function to each logic block and connecting them by configuring the interconnect. As a result, a desired circuit has been created.

3.1.2 FPGA architecture

Now we will take a deeper look into the architecture of a typical FPGA chip. As mentioned above, FPGA consists of three main parts: the configurable logic blocks (CLB), the input/output blocks (IOB), and the programmable interconnection network, as sketched out in Figure 3.1.

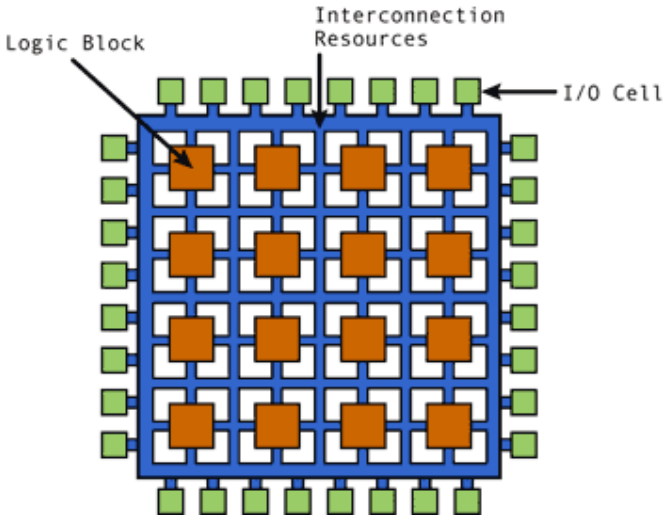

Figure 3.1 FPGA architecture [21]

3.1.2.1 Configurable logic block

This is the main logic resource of the FPGA performing user-specified logic functions. The quantity and features of CLBs in a FPGA chip vary from device to device, but every CLB consists of a configurable look-up-table (LUT) with typically four or six inputs, some circuitry for selection (e.g. multiplexer), and flip-flops. The LUT can be configured to handle combinational logic, shift registers, or RAM; the flip-flops are for clocked storage elements; while multiplexers are for routing signals within the block and to and from external resources. More detail in CLB structure is provided via the CLB of the Virtex-5 family FPGA chip [8] as used in our system.

The basic Virtex-5 logic element consists of one six-input LUT, one configurable flip-flop, and three multiplexers [9]. A fast dedicated carry logic is added for special logic and arithmetic function performance. In some elements, the LUTs can be configured as a RAM, so called distributed RAM, or as a shift register. These elements are grouped together forming a slice, providing logic, arithmetic and storage functions, as shown in Figure 3.2.

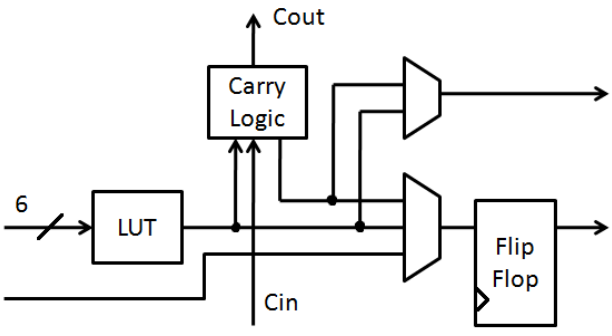

Figure 3.2 Virtex-5 logic element

One CLB element is composed of a pair of slices, as illustrated in Figure 3.3. Depending on the function of the LUT in a slice, it is divided into memory slice (sliceM) and logic slice

(sliceL). These two slices have no direct connections to each other, and each slice is organized as a column, with independent carry chain. Each CLB is connected to local and global routing resources via a switch matrix. All CLBs in an FPGA chip are identical, and are arranged in columns and rows [9]. Figure 3.4 demonstrates the organization in columns and rows of the CLBs and slices in the Virtex-5 chip.

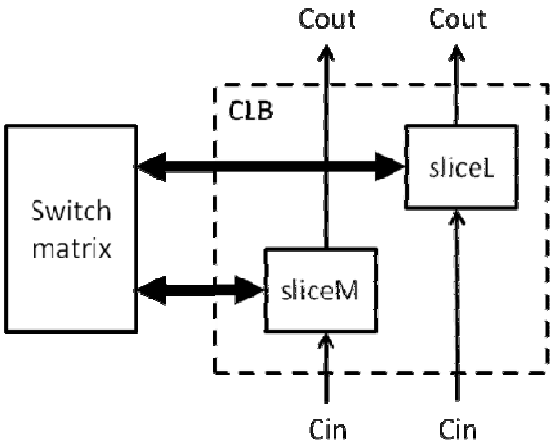

Figure 3.3 Virtex-5 CLB architecture

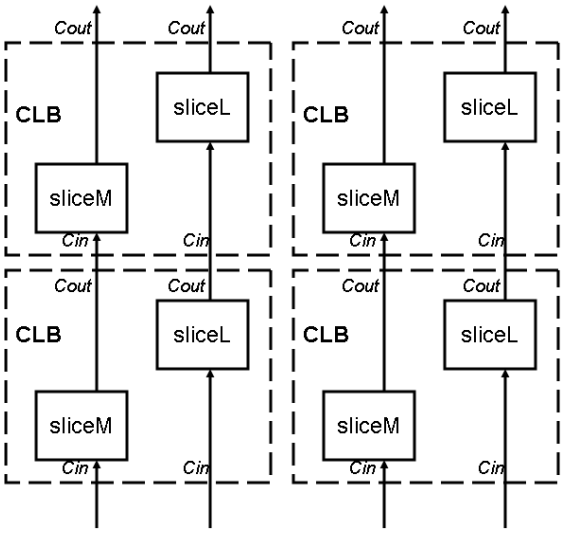

Figure 3.4 Column and row organization of CLBs and slices

3.1.2.2 Input/output block

A configurable input/output block (IOB) is used to bring signals into a FPGA chip from outside circuits, and also send out signals from internal chip circuits. A basic IOB contains a tri-state output buffer, an input buffer, and two flip-flops. One flip-flop is used to clock the output signal in order to shorten the clock-to-output delay of the output signal. Another one is for registering the input signal in order to reduce the hold time requirement of the device. Figure 3.5 sketches out a basic structure of an IOB.

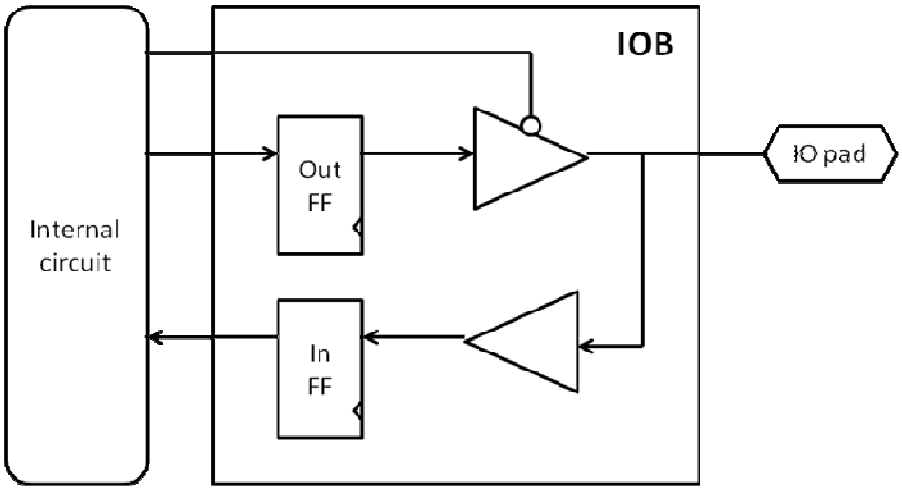

Figure 3.5 Basic structure of an IOB

Special IOBs with high-drive clock buffers (known as clock drivers) are distributed around the FPGA chip. These buffers connect to clock input pads, and drive clock signals from the outside onto fast, low-skew global clock lines within the FPGA.

3.1.2.3 Programmable interconnection

All component logic resources after being configured will be connected together to create a complete circuit via programmable interconnection. This step is an important part of a FPGA design as it decides the speed of the whole system. There are two types of interconnection: direct and general purpose interconnections, as illustrates in Figure 3.6.

Direct connection is a connection between neighboring logic blocks, with short distance wires. This provides a fast speed since no switching matrix is used.

General purpose connection is a routing type using switching matrix when there is no direct connection available. It is slower than the direct method, but it can be used for all distances.

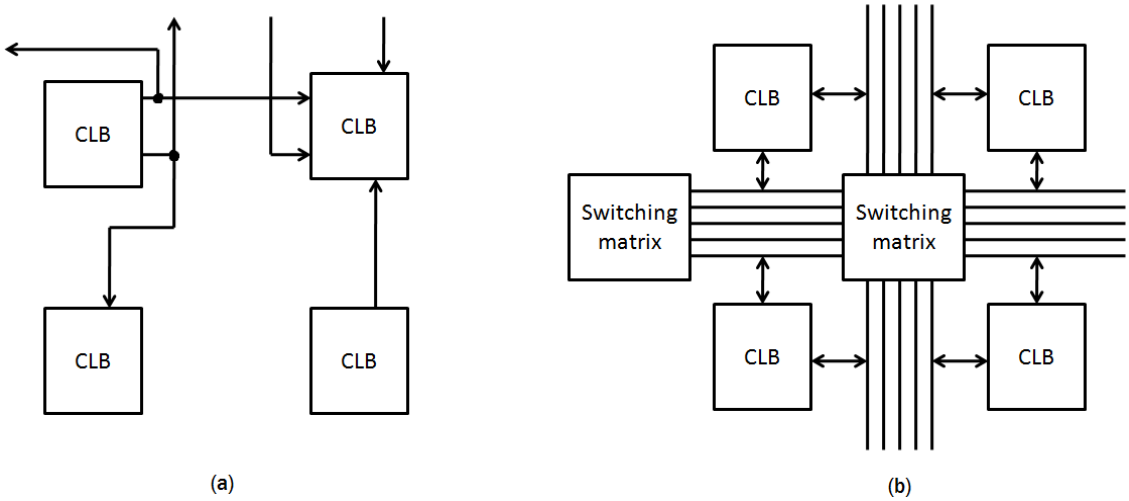

Figure 3.6 (a) direct interconnection and (b) general purpose interconnection

Different synthesis tools have different algorithms of wire routing, and the successful one is the one which has the best circuit performance in terms of speed, power consumption, and occupied area.

3.2 VHDL

Nowadays, circuits are very complex. An abstract view is needed to handle this complexity. It is common practice to use a hardware-description language (HDL) to describe a digital circuit. A HDL is a type of computer language used for modeling a (digital) electronic system. Starting from an architecture design of a system, such modeling is the first step in order to synthesize and verify the system with the help of computer-aided design (CAD) tools.

There are several HDL, such as Verilog, VHDL, SystemVerilog, SystemC, etc. Among them, Verilog and VHDL are in common use. In this system development, VHDL is used.

VHDL (stands for VHSIC HDL: very high speed integrated circuit HDL) was standardized by IEEE (Institute of Electrical and Electronics Engineers) in the early 1980s. Through some revisions, the current most widely used version is the 1987 version which has been adopted by IEEE as a standard (IEEE Std. 1076-1987), sometimes referred as VHDL'87. However, there is a newer version VHDL'93 which is in the process of replacing VHDL'87 [10].

VHDL is a powerful language for hardware design as it can satisfy the demands of a hardware design process:

- The structure of the design can be described in VHDL with components, sub-components, and how they are connected.
- The behavior of the design can be described in VHDL as such behavior is familiar with programming language format.
- There is a possibility of verification as the design is possible to simulate before manufacturing.

For more detail of how to become familiar with VHDL coding, it is recommended to have a look at [11] and [12].

3.3 Software tools

3.3.1 For system development

3.3.1.1 Xilinx ISE

Xilinx ISE is a digital hardware design environment developed by Xilinx to be used for designing, implementing, simulating and downloading digital systems into a FPGA or CPLD target device [13]. The version of ISE used in this thesis work is ISE v10.1.

3.3.1.2 EDK

Xilinx EDK (embedded development kit) is a tool for embedded processor (both hard and soft core) development within a FPGA system [14]. With the support of this tool, it is possible to add an embedded processor beside the hardware logic part, to make the system more powerful and flexible with additional software solutions. There are two main environments inside EDK: the XPS (Xilinx platform studio) for hardware portion of the embedded processor design, and the SDK (software development kit) for the software portion of the processor with C/C++ supported. The version of EDK used in this thesis work is EDK v10.1.

3.3.1.3 Overview of the design flow

A typical design flow of a digital hardware system by ISE includes six main steps as shown in Figure 3.7.

- Starting from design creation, a project for the proposed system will be generated. With the help of ISE project navigator tool, all hardware modules (source files) with structured interconnections are built in VHDL.

- In the second step, the design will be synthesized. The synthesis engine with the support of Xilinx Synthesis Technology [15] compiles the design to transform VHDL source codes into an architecture-specific design netlist. In this step, the register transfer level (RTL) schematic (level of generic symbols such as adders, multipliers, counters, logic gates...) and the technology schematic (e.g. LUTs, carry logic, IOBs or other technology-specific components level) of the design are generated.

- In the third step, constraint parameters can be added to the design, via the ISE constraint editor tool. Here the system can be constrained in timing, occupied area, IO pins and other parameters.

- Next step is the design implementation, where the design will be turned into physical resources of a target device (a FPGA chip or a PROM chip). This step is divided into three processes: translate process (for merging the netlist from the synthesis step with the constraint parameters from the third step), map process (for turning the previous netlist into the real device resources), and place and route process (for putting those elements on the chip surface and connecting them to form a complete system).

- In the fifth step, the result of the design implementation is analyzed. Device resource utilization, timing performance, power utilization and performance against constraints can be analyzed in this step.

- The last step in the design flow is device configuration and programming where the configuration files will be generate and downloaded onto the target device.

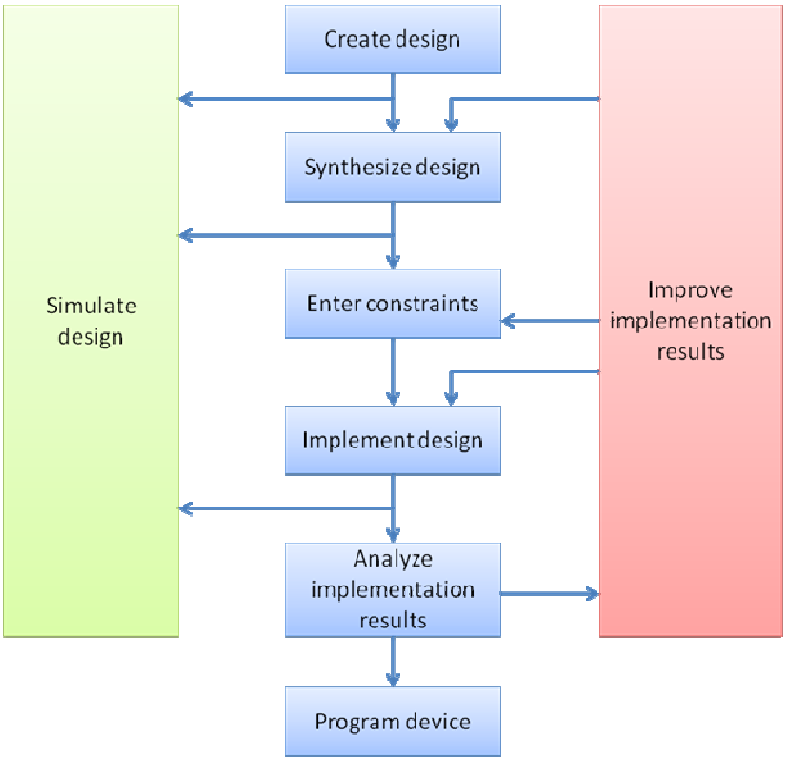

Figure 3.7 Design flow of the ISE

In between of some steps of this design flow, it is possible to verify the design with the help of simulation tools. In this thesis work, the two tools used for system verification are ModelSim and Xilinx's ChipScope. In the next section, more detail will be provided for these tools.

3.3.2 For system verification

3.3.2.1 ModelSim

ModelSim is a tool developed by Mentor Graphics Inc which is used to simulate and verify the functionality of a digital design. Actually, ISE tools also support functional simulation, however with ease of use as well as powerful performance, ModelSim is widely used as an effective verification tool for digital hardware design.

In order to simulate a design, ModelSim goes through five basic steps [16]:

- *Collecting files and mapping libraries*: all source files and libraries of the design are collected preparing to be simulated.
- *Compiling the design*: these source codes will be compiled with one of the three supported language compilers: Verilog, VHDL, or SystemC.
- *Loading the design for simulation*: after being compiled, all modules will be put into a hierarchy and linked together by connecting their ports.

- *Simulating the design*: after loading successfully, the design will be simulated from the time zero. The simulation duration is set by the user.

- *Debugging the design*: after running the simulation, it is possible to debug the design using many supported functionalities such as waveform analysis, tracing signal.

3.3.2.2 ChipScope

ChipScope [17] is a set of tools developed by Xilinx which allows a designer to probe the internal signals of a design inside an FPGA chip, as one can do with a logic analyzer. In order to use the ChipScope in an existing design project, first the Core Generator tool is used to generate the ChipScope cores, which perform the trigger and waveform capturing functionality on the FPGA. Afterwards, these cores need to be instantiated in the VHDL code, and those cores need to be connected to the signals to be monitored. The complete design is then recompiled.

There are two main types of Chipscope cores:

- *ICON core*: stands for integrated controller core; it provides a communication path between the JTAG boundary scan [18] port of the target FPGA and ILA cores.

- *ILA core*: stands for integrated logic analyzer core, it is customizable and can be used to monitor any internal signal of the design.

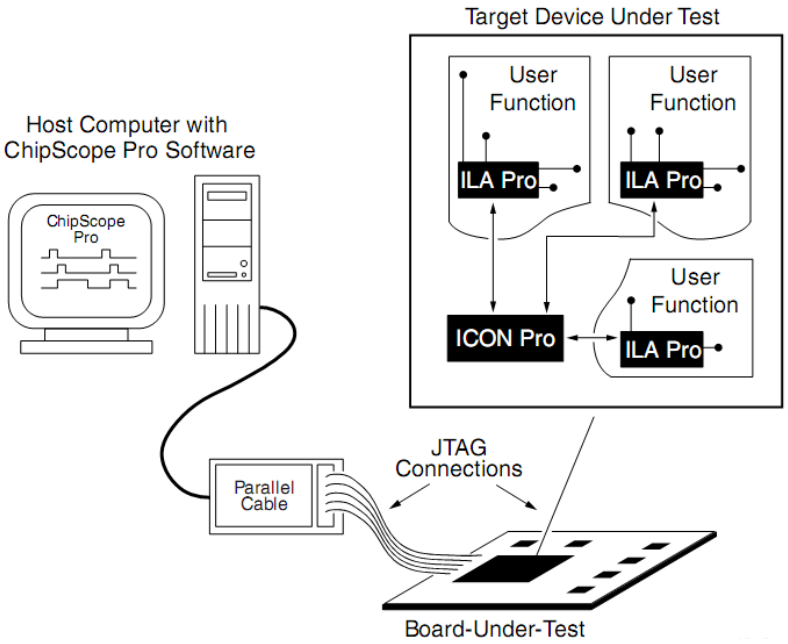

Figure 3.8 ChipScope tool operation [17]

Figure 3.8 demonstrates an operation of the ChipScope tool for capturing and analyzing signals of a FPGA system. ILA cores are generated and probed to signals of the design which are needed to be analyzed. Through ICON core, these probed signals will be transferred (via JTAG connection) to the ChipScope software on the computer for displaying and analyzing.

4 NFI-10GE system: current status and proposed architecture enhancements

4.1 The desired system functionality

In the cinematography system, the NFI-10GE system of the field recorder was designed to be:

- An intermediate media data storage for the cameras at a maximum resolution and frame rate of 4K (4096×2160) format with 30-bit color depth in 4:4:4 chroma subsampling rate [19] at 30 frames per second. Uncompressed video with multi-stream supported producing by camera devices will be stored in real-time speed to the internal solid-state memory board of the field recorder. The data bandwidth of the whole system is targeting 9 Gbps to support single stream recording at highest quality or multi-stream with different 2K format or below, with maximum of eight streams supported. This media data can also be reviewed in real-time using display devices (monitors).

- A media server for data downloading from client devices (computer based). One option is a faster than real-time download of a single stream. As a file server, basic functions for file management are required, such as provide file table information, delete file(s), and erase all files.

4.2 The current functionality

According to the design above, a field recorder has been prototyped, and known by the name "FlashPak". In the time of writing this thesis, the second version of FlashPak – the FlashPakII - is being used, which has been sketched out in previous chapters. Now is the time to look back at the module structure of NFI-10GE to see if it can support the design functionality.

4.2.1 PHY and MAC

They are the completed IP cores from Xilinx: XAUI v7.3 and 10-Gigabit Ethernet MAC v8.4. Full functionality is supported within these licensed cores.

4.2.2 PSUnit

This module is developed by Technicolor. At the time of writing this thesis, development has been completed. Only RTP streaming record (multi-record supported) and playback are directed to the application module (hardware path); all other type packets goes to the PowerPC module (software path) to be processed by the embedded software.

4.2.3 Stream module

This module is developed by Technicolor. It had also been completed when the work of this thesis started. Interfacing with the real-time data path, the stream module works at a clock of 156 MHz with 64-bit data bus, and then the maximum throughput bandwidth can reach 10 Gbps, enough to meet the requirements on the system. The UDP header of the TX packet is formed in the stream module, with the header information provided by the PowerPC via DCR bus interface.

4.2.4 Application module

This module is developed by Technicolor. Only RTP module is completed at the time of starting the thesis work. This RTP module is in charge of two main functions: playback and recording.

- For playback scenario: the playback request from user interface board (triggered by PLAY button) was replied with the real-time RTP packets stream containing video data in the memory controller board. In fact, the memory controller board receives the playback command from the user interface board first, prepares to put video data to its FIFO, and then informs the NFI-10GE system via I2C bus connection. The PowerPC of NFI-10GE will trigger the hardware preparing to form and transmit playback video data taking from the FIFO of memory board. The playback process will be stopped only if there is no more data in the FIFO (end of file) or by user intervention (press STOP button).

- For recording scenario: has the same procedure as the playback, but the data path of the recording scenario is reversed: a real-time TX path from cameras to RTP module writing into FIFO. This operation is stopped only if the STOP button on user interface is triggered.

Application unit has not been completed due to a lack of NFS packet processing for non-real-time data. Accordingly, one of the main tasks of this thesis work is to develop the NFS module functionality.

4.2.5 PowerPC

This is the embedded hard-processor within the Virtex-5 FPGA chip. The software environment of PowerPC is developed by Technicolor. At the time of the thesis work started, it was done for non-real-time packet processing, and the NFS file management (e.g. *mount, access, readdirplus, unmount*…). However, the NFS functionality is emulated completely in software, with the simulated files generated by the software itself. Hence, another task of this thesis work is to develop the software in order to be compatible with the NFS hardware-based which will also be developed.

4.3 Proposal for the system enhancements

4.3.1 Hardware

4.3.1.1 NFS module

In order to complement the NFS real-time path processing into the NFI-10GE system, the NFS functionality module needed to be implemented in parallel with the RTP module. This new module will process the NFS read request packets coming from a client device, and respond with an appropriate NFS packet (e.g. non-real-time video data in case of valid request, or error acknowledgment in case of invalid request). A connection with PowerPC is necessary as the PowerPC is the only module which is able to request video data from memory board.

The operation of the NFS module:

- NFS read request packet which is sent from a client after passing lower level network layer processing (e.g. MAC unit to remove Ethernet header, PSUnit to remove IP header) will be passed to NFS module for analyzing request type.
- Information of request will be extracted from NFS header. Parameters such as XID (transaction ID – a random number to distinguish different packets), requested size will be stored in specific registers for later reply packet formation.
- In case of valid read request, the read-reply module will be triggered. It is a finite-state-machine (FSM) based module. Read-reply module is in idle state, waiting for the trigger signals to switch to reply states. In these states, all necessary information of the NFS reply packet is formed; some parameters are taken from previous request information (e.g. XID, size). Video data is also requested in this phase, so that the video data will "jump" exactly right next to the last header word. As a result, the NFS read reply is generated, and be forwarded to TX path for transmission. Figure 4.1 shows a typical valid NFS read-reply packet format generated by NFS functionality module.

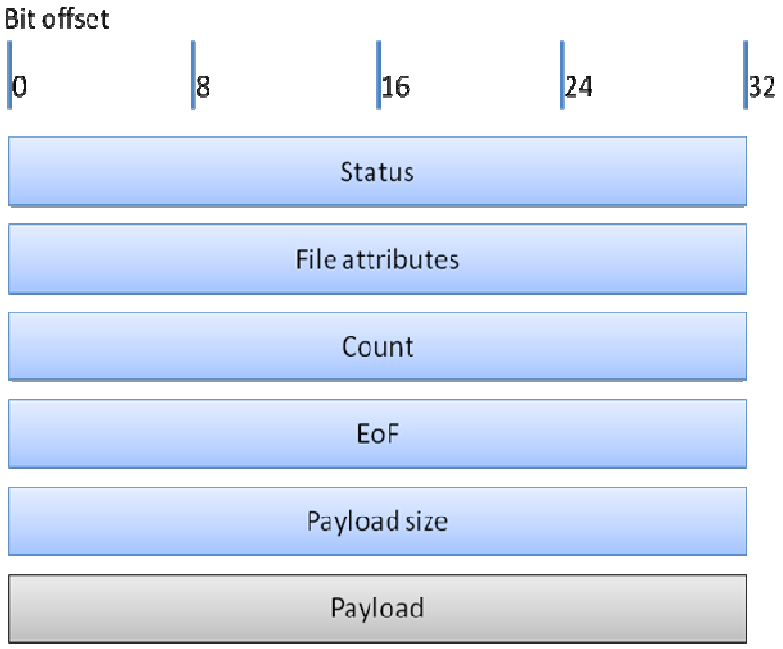

Figure 4.1 NFS read-reply packet format

- In case of invalid read request, the error-reply module will be triggered. Having the same structure as the read-reply module, error-reply module will generate a NFS error reply packet using FSM, which will also be sent to TX path for transmission.

4.3.1.2 Data transfer mechanism between NFI-10GE and user interface board

In the current FlashPakII architecture, the NFI-10GE interface board and user interface board communicate via an I2C bus (NFI-10GE is the slave device). There is an interrupt connection between them, but it has not been used yet. With the growth in the complexity of communication between these two boards when new functionality is added (e.g. NFS file table transfer, communicating commands…), a new communication mechanism for this interface is essential. Here the author proposed a data transfer mechanism using the available resources of this interface, to ensure the reliable of data for NFS functionality between the two boards, as seen in Figure 4.2.

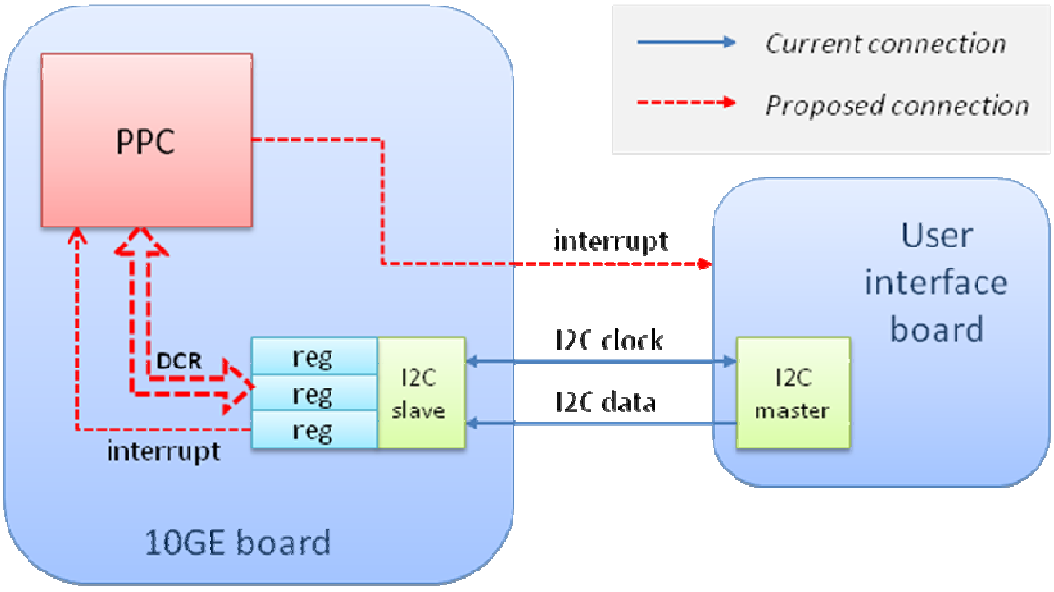

Figure 4.2 Proposed data transfer mechanism between 10GE and user interface board

With this transmission mechanism, the communication between the two boards would become more reliable and flexible. With the possibility of adding user-defined commands, further system upgrade can be done easily and quickly by upgrading the software of the two boards, without touching their hardware architectures.

4.3.1.3 Application wrapper

As the current RTP module and the proposed NFS module have the same data interface signals to other modules (e.g. the 32-bit data bus, the data-direction control buses), it is necessary to have a multiplexer in order to select the appropriate path, avoiding the conflict of data flow which may lead to malfunction. Figure 4.3 refers to the structure of the application wrapper with multiplexer implemented.

The common register space is another problem to be solved by this application wrapper module. It is necessary to have common registers which can be accessed from both user interface board and 10GE board in order to establish a connection between them serving for data transfer mechanism mentioned above. Since there are two different bus interfaces for RTP functionality (I2C bus connected with user interface board) and NFS functionality (DCR bus connected with PowerPC), the common registers which can be accessed both from PowerPC and user interface board are essential for the mutual communication (e.g. file table transfer, command transfer). With the wrapper module, multi-port access registers are available for NFS functionality supported. Figure 4.3 illustrates the proposed common registers (c.reg) implemented with both DCR and I2C controlled.

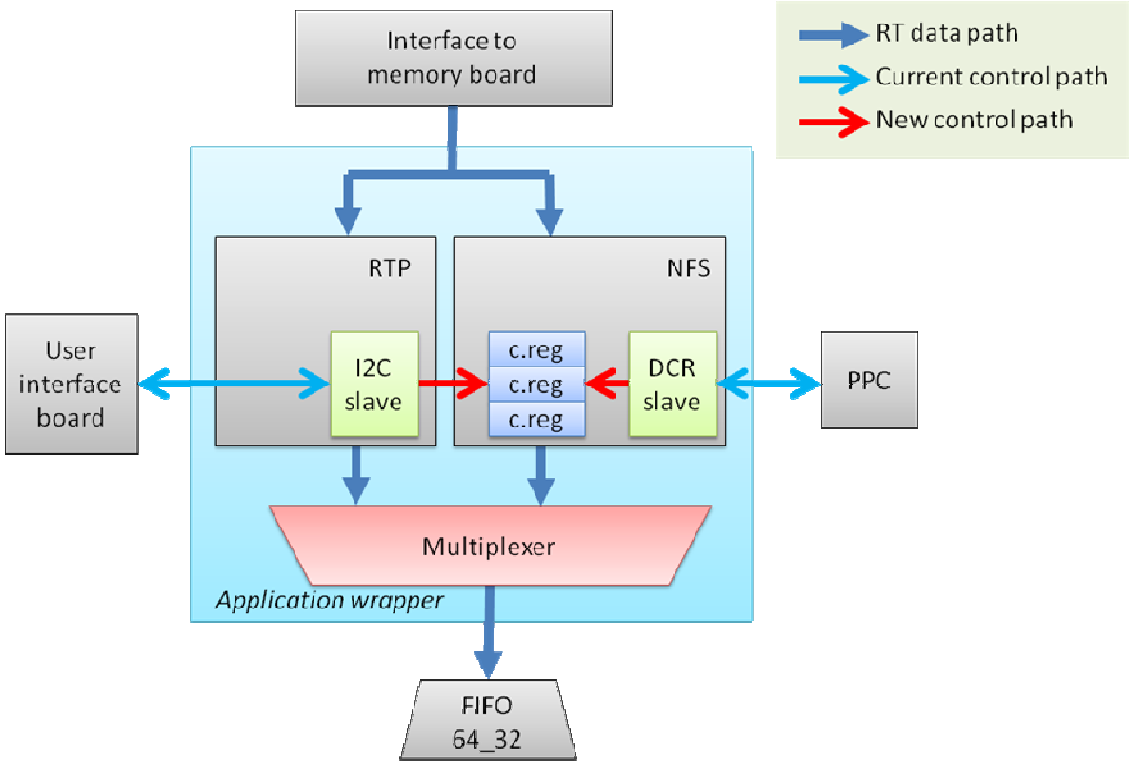

Figure 4.3 Application wrapper structure with common registers supported

4.3.1.4 Timeout detector

Due to the original RTP record mechanism of the system, there is no byte-exact information of every take stored in the memory controller board, but only the timing length. Hence, when the NFS read request command is performed (which need exact take size), the take size value given to a client can be inaccurate compared with its actual size. There will be two possibilities:

- The client reads the take with a requested size larger than the actual size. In this case, when the whole take in the memory board has been read out, the NFS functionality module still waits for take data since request(s) from the client still comes. Consequently, the system will stall.

- The client reads the take with a request size smaller than the actual size. In this case, when all requests has been replied, the memory board still stores remained data of the reading take (since not all are requested). As a result, the memory will not be flushed for the new take read request processing; the data of the previous take may be mixed with the data of the next take reading leading to incorrect take data in one file.

In order to handle these unwanted cases, the additional timeout_detector has been implemented within NFI-10GE system. It detects the long-time response of the NFS module counting from the time of finishing the reply activity: if it is longer than a certain period, it will be considered as the last read request from client (of the requesting file); no more read request available. An output signal will be triggered. Receiving this triggering, the NFS

module will be reset to be ready for other requests. At the same time, the PowerPC is also interrupted to stop the data transfer operation of the memory board (by sending command to the user interface board). As a result, the system switches back to the idle state, waiting for new requests from client.

4.3.2 Software

The current software has performed the basic NFS functionality: *mount* (establish the connection between the FlashPakII and the client), *ls* (request for the list of files), *cp* (copy the selected file data to client). However, the files are still simulated, rather than real files with real information on the memory board. In order to save time and also to inherit the result of the previous work, all software based NFS functionality will be utilized; only the simulated file information will be replaced by the real one getting from the user interface board. Hence, the software enhancement in this thesis work focuses on the transmission of the file information from the user interface board to the 10GE board, so that the PowerPC can use it for replying to a client, replacing the simulated information.

4.4 Description of the implementations

4.4.1 NFS-sub module

In fact, the NFS functionality module was developed in the previous FlashPak version, and is re-used in the FlashPakII system. Only some minor modifications are needed for the compatibility with FlashPakII. Basically, its architecture is completed, as seen in Figure 4.4. NFS functionality module is composed of three main parts: RPC_NFS, reply, and configuration modules.

- RPC_NFS module: this module plays a role as a NFS header analyzer. Receiving an input NFS packet from PSUnit, RPC_NFS module analyzes its packet header, extracts information from it and stores them into specific registers (*NFS_reg*). This information will be used for later reply packet. Furthermore, the NFS header of a packet is also checked for validity. In case an invalidity is found, an error reply signal will be triggered controlling the reply module for an error reply answer.

- Reply module: this module acts as a NFS reply packet generator. In the current system, two types of reply are possible: read reply (answer a valid NFS read request) and error reply (answer an invalid NFS read request). For read reply operation, a NFS reply header will be formed with reply information storing in *NFS_reg* registers (e.g. XID, payload size), continuing by video data from memory board (via memory interface module). In case of error reply activation, a standard error reply packet will be generated and transmitted to the client.

- Configuration module: the PowerPC configures the whole NFS functionality module (e.g. the operation modes) by accessing the configuration registers in this module via DCR bus.

With the availability of this module in the cinematography system, the hardware part of NFS functionality has been implemented, enabling the raw media data transmission from the memory board of the field recorder to a client device for post video processing.

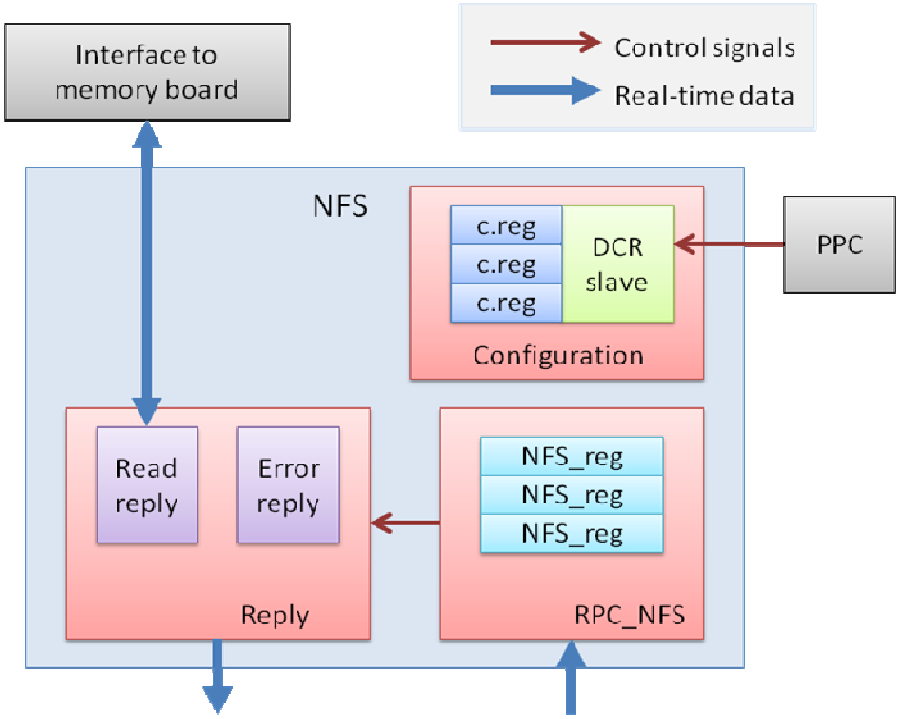

Figure 4.4 NFS functionality module architecture

4.4.2 Data transfer mechanism between 10GE and user interface board

The physical links between the 10GE board and user interface board have been built, but not fully used. Only I2C connection was used, with one-way-connection. To establish the two-way-connection between them, some additional registers are generated:

- *Reg_NFS_command*: For sending commands to the user interface board, the PowerPC will write a specific value (user-defined) into this register. The LSB bit of *reg_NFS_command* is directly connected to the external interrupt pin of the user interface board.
- *Reg_take_order*: stores the take order in the memory. The first and last take is also indicated here:
    - Bit 5-0: indicates the order of current take (min=0, 1, 2, … 127=max)
    - Bit 6: indicates the first take (1) or not (0)
    - Bit 7: indicates the last take (1) or not (0)
- *Reg_take_nr:* stores the number of the current take.

- *Reg_stream_nr:* stores the stream number of the current take.
- *Reg_take_size:* (five eight-bit registers) stores the size in byte of the current take.
- *Reg_NFS_intr:* connected directly to the PowerPC as an interrupt signal. This register is written by the user interface board when it finishes filling the file-table information and wants to signal the PowerPC.

These registers also need to be accessible from both PowerPC and user interface board. This is done by implementing the common registers within the application wrapper. The addresses of these registers is shown in Table 1.

Register name	Address	Remark
NFS_take_nr	0x6d	For NFS read reply
NFS_stream_nr	0x6e	For NFS read reply
Reg_NFS_command	0x6f	
Reg_take_order	0x70	
Reg_take_nr	0x71	For FT request
Reg_stream_nr	0x72	For FT request
Reg_take_size1	0x73	Highest significant byte
Reg_take_size2	0x74	
Reg_take_size3	0x75	
Reg_take_size4	0x76	
Reg_take_size5	0x77	Lowest significant byte
Reg_intr_FT	0x78	

Table 1: Registers' addresses

Furthermore, new interrupt signals need to be established to the PowerPC: *intr_NFS, intr_FT,* and *intr_timeout.*

- *Intr_NFS:* this is an interrupt from the NFS module, triggering when a first read reply packet is generated, asking for video data of a requested take. When the PowerPC gets the *intr_NFS*, it will ask the memory board for data by writing a specific command to the *reg_NFS_command* register.

- *Intr_FT:* this is an interrupt from the user interface board via *reg_intr_FT* register, signaling the PowerPC about the availability of a file-table information in file-table registers.

- *Intr_timeout:* this is an interrupt from the timeout detector, signaling the PowerPC when an NFS timeout is detected. Receiving this signal, the PowerPC knows that the read reply operation is finished, and the memory board needs to stop transfering data. As a result, PowerPC will write a specific command to the *reg_NFS_command* register.

Figure 4.5 illustrates the architecture and operational procedures of this mechanism.

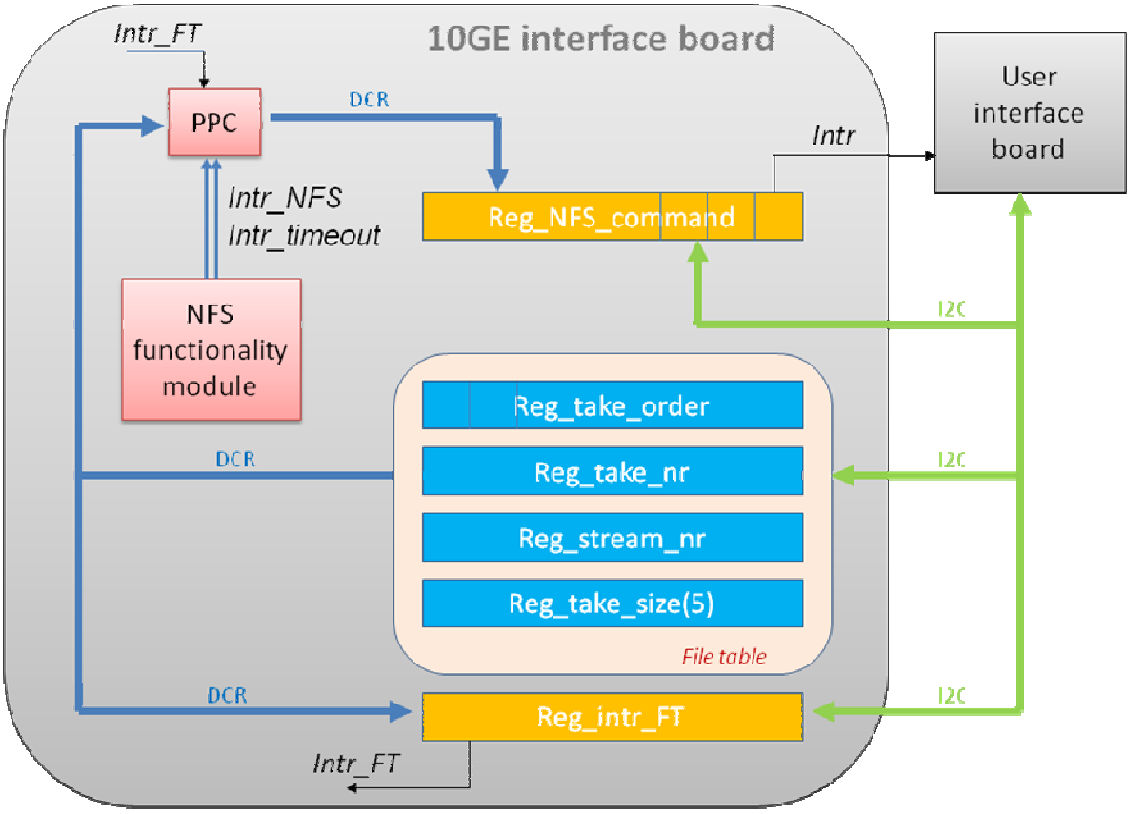

Figure 4.5 Data transfer mechanism between 10GE and user interface board

Here is the procedure to transfer the file-table from the user interface board to the 10GE interface board:

- First, the PowerPC of NFI-10GE system receives the '*ls*' command from a client, requesting a list of the files on the memory board.

- The PowerPC then asks the memory controller board for the file table by writing to a register named *reg_NFS_command*, a self-defined command.

- The memory controller board will be interrupted by that writing activity, read out the command from that register, and clear it.

- The memory controller board writes the file-table information into specific registers. Then it notifies the PowerPC by writing to *reg_intr_FT*, a register which triggers the interrupt to the PowerPC.

- The PowerPC gets the interrupt, reads the file-table information, and replies to the client.

- After that, the PowerPC will ask for the next file-table (if available). This procedure will repeat until the last file-table is reached.

With this mechanism, file-table information is transferred one by one with fixed size, no dynamic memory needed, so system resources are saved. With the availability of the self-defined command register, it will be easy to update/modify/add communication procedures between these two boards without changing the hardware architecture.

4.4.3 Application wrapper

The multiplexer is included for selecting one of the two output paths: RTP or NFS data. When the RTP path is chosen, the data transmitted onto the network will be from the RTP module. Conversely, the NFS data will be transferred when the NFS path is selected. The signals responsible for the selection are *new_NFS_read_request*, *play_enable,* and *record_enable*:

- *New_NFS_read_request* is the signal generated by PSUnit, signaling the availability of a new NFS read request packet from a client. In case this signal is triggered, the multiplexer will switch to NFS path.

- *Play_enable* and *record_enable* are the signals coming from RTP functionality module when a request for playback or recording appears. These cases happen when an user presses a PLAY or RECORD button on the user interface board, requesting a playback (display video data on a monitor device) or record (store in real-time the video data from camera(s)). If one of these signals triggers, the multiplexer will switch to the RTP path.

For the common register space implementation, both the I2C slave and the DCR slave modules are utilized. The new registers have been created inside NFS module, within the current configuration registers. The difference is that beside the current DCR bus interface, the new registers have an additional I2C bus interface taking from the I2C slave of RTP module. As a result, common registers with two bus interfaces (DCR and I2C) have been generated, responsible for the data transfer mechanism between the 10GE board and the user interface board.

4.4.4 Timeout detector

For the operation of this module, two input signal types are used, as illustrated in Figure 4.6:

- *New_NFS_read_request*: this signal comes from the PSUnit, indicating the availability of a read request packet of a client requesting for video data of a certain take. PSUnit

has a function to detect such type of request and indicate it by the *new_NFS_read_request* signal.

- Reply-activity indicator: this is a group of signals coming from the memory controller board (*data_ready* and *data_valid*: indicating the status of the memory's FIFO) and the NFS functionality module (*data_request*: used for requesting data from memory controller board). Reply-activity indicator is active when one of its signals activated. Otherwise it is not considered as a read reply activity.

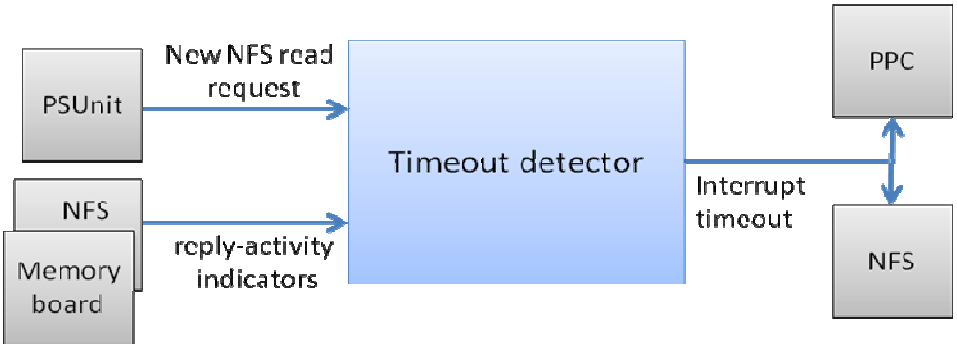

Figure 4.6 Timeout detector module

Figure 4.7 demonstrates the timing diagram of the timeout detector operation. Like other timeout detections, its operation is based on an internal counter. An additional signal *counter_lock* is used in order to ensure that the interrupt is triggered only one times when timeout is detected. In the beginning, the counter is reset to zero. *Counter_lock* is active, and the no detection operates. When new NFS read request packet comes, the *counter_lock* is inactive, but the module is still in the idle state since reply-activity is happening. When finish replying, the timeout counter starts (detect state). There are two possibilities:

- The new NFS read request packet comes, but the counter did not reach a certain timeout value. In this case, the state is back to idle, and the counter is reset to zero.

- The counter reaches the timeout value. In this case, it is considered as no more NFS request packet will come; the last read request of a take has reached. An output *interrupt_timeout* signal is triggered, sending to the PowerPC to stop the memory controller board which is still in sending data state. Its memory's FIFO is also flushed to be ready for a new take read out process. On the other hand, the interrupt signal is also sent to the NFS functionality module to stop its replying state, coming back to the idle state and waiting for a new read reply operation. For the internal operation, the *counter_lock* is activated, and the timeout detector will be locked until a new read activity (for new take, of course) comes.

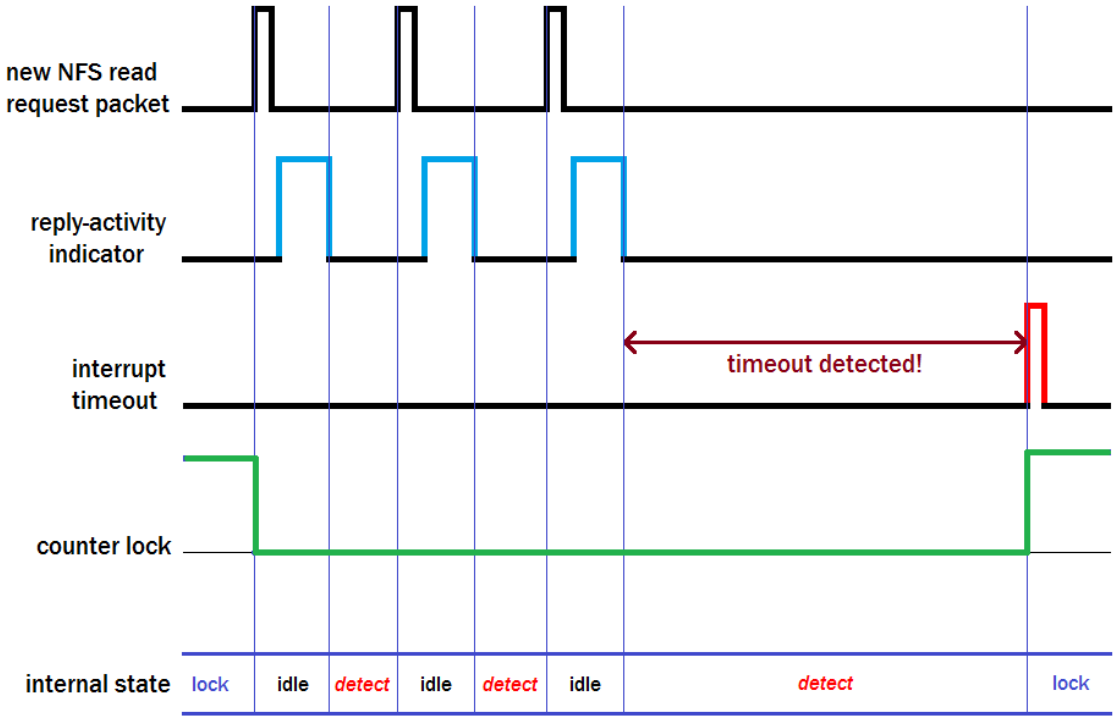

Figure 4.7 Timeout detector timing diagram

4.4.5 Software enhancement

As described in the data transfer mechanism implementation, the PowerPC software communicates with the user interface board by writing commands onto the *reg_NFS_command* register and reading information from file-table registers when receiving an interrupt *intr_FT*.

Previously, the PowerPC software generates two simulated files with their simulated parameters (e.g. file name, file size, file attributes) once when booting up; the directory contents will be fixed during the whole working time. When processing and replying the NFS requests of the client, this file information will be used.

Now, the procedures to generate simulated files are kept, but the number of files generated and their information will be taken from the user interface board where the real file information is stored. In this case, the data transfer mechanism will be applied to get the file information. Whenever the client requests for the file information, the PowerPC software re-generates the files again. This ensures that the file information sending to the client is always up-to-date. Implementing in this way, all the previous work of software can be utilized, and then time is saved.

5 Verification

5.1 Verification with ModelSim

5.1.1 Unit test

For functional verification, three hardware modules have been checked with ModelSim [20]: NFS functionality module, application wrapper, and timeout detector. The data transfer mechanism has not been tested with ModelSim due to the involvement of the processor (PowerPC) which cannot be simulated on this version of ModelSim.

5.1.1.1 NFS functionality module

In order to verify this module's operation, simulated NFS read request packets with IP and UDP header removed have been applied to the input port. Other control signals are also simulated (e.g. data valid, data ready from memory controller board).

Two types of read request are used: a valid one and an invalid one. Both have correct results at the output: for valid read request packets, the output is correct read reply packets. Conversely, for invalid read request packets, error packets were sent out. And then, it is concluded that the NFS module is correct in functional behavior.

5.1.1.2 Application wrapper

Two main functions of the application wrapper have been implemented: the multiplexer and the common register space. Only the multiplexer is verified with ModelSim.

The three selecting signals for the multiplexer (*new_NFS_read_request, play_enable* and *record_enable*) are simulated, switching values randomly. In all cases, the result signals (*RTP_path* and *NFS_path*) are correct. And then, it is concluded that the multiplexer is correct in functional behavior.

5.1.1.3 Timeout detector

For testing the functionality of this module, its input signals have been simulated similar to reality: a *new_NFS_read_request* comes followed by the *reply activity indicator*, repeating for several times and then inactive. The timeout counter has detected successfully the long inactive period, and raising the *interrupt_timeout* correctly. The *lock* signal is also activated, and then active again when *new_NFS_read_request* signal is triggered. From this result, it is concluded that the timeout detector is correct in functional behavior.

5.1.2 System test

In this verification phase, the whole hardware system (not including the PowerPC) has been simulated in ModelSim. It is checked for NFS read request, RTP playback and RTP record operations.

- For NFS read request: the full read request packets have been used, serving as the input of the PSUnit. All configuration parameters for all needed modules have been set by setting the default values on their config registers since PowerPC software was not involved. As a result, the output packets from PSUnit are in a correct format with correct data payload as expected.

- For RTP playback: simulated data from memory controller has been generated using an additional counter. The signal indicating the playback mode from the user interface board also be triggered manually in order to set the whole 10GE system into playback mode. As a result, the packets at the output of PSUnit are in a correct format with correct data payload as expected.

- For RTP record: simulated data packet from cameras are generated and put into the input of PSUnit. Via hardware modules, the IP, UDP, RTP header of these packets have been stripped out. Coming out from the output of application module, all raw (simulated) video data have been verified correctly intuitively.

For these positive results, it is concluded that the 10GE system is correct in functional behavior.

5.2 Verification with ChipScope

During the real operation of the FlashPakII, the ChipScope cell is added for unit and system verification. When a bug (unwanted result) is found, all concerning modules will be probed to find the bug. Then the design will be examined and modified. This debug procedure is repeated until the system is bug-free. All control and data signals are added into ChipScope to examined state by state, even clock cycle by clock cycle.

5.3 Performance benchmarking

At the time of writing this section, the NFS functionality hardware module has been implemented successfully on the FlashPakII. From a Linux client, the connection to the FlashPakII can be established with '*mount*' procedure. Performing '*ls*' or '*ll*' commands, the list of all available files together with their sizes and attributes will appear on the screen. Files can be downloaded to the client by the copy command '*cp*' normally.

The average download operation is at a speed of approximately 25 MBytes per second. In more detail:

- The response time of the FlashPakII for each read request is 16 micro-seconds. It is calculated from the time of receiving a read request to the time that the reply packet for that request transmitted completely.
- The gap between two consecutive read request packets is 164 micro-seconds.
- These values are taken from ChipScope, with additional clock counter in order to determine exactly the number of clock cycle gap.
- The number of packets transmitted in one second will be $1/164.10^{-6} = 6097$ packets. Each packet is 4096 bytes, resulting in data transmission speed of approximately 25 Mbytes per second.

6 Outlook

The NFS performance speed is not really impressive. Each video file can be up to tens of gigabyte. With the current transfer speed, it may consume tens of minute to download. There are several ways to improve it:

- By increasing the NFS packet format. The current NFS payload size is 4KB. If the payload of 8KB is used (jumbo frame), the number of read request packet will be reduced to a half. As mentioned in the performance benchmark section, the response time for each read request is 16 micro-seconds, while the time between two consecutive read request packets is 164 micro-seconds. It means the main time consuming phase is the reply packet processing and request sending of the client. If the number of these process reduces by a half, the transfer speed can be improved much: new transmission time will be (16×2 + 164÷2) = 114 micro-seconds (the number of request reduces a half, while the time response of each request is double as the file size is now two times bigger), comparing with the old transmission time of (16 + 164) = 180 micro-seconds, i.e. 58% faster. Some modifications in the NFS functionality module of the FlashPakII are needed in this case.

- By continuous reply mechanism. Currently, the FlashPakII and the Linux client processes request packet one by one: when a request releases, it needs to be processed and reply successfully before the next request is generated. This is also the mechanism of NFS protocol. If all requests can be generated and sent continuously without needing acknowledgements of the previous ones; similarly, the FlashPakII is able to store and process all receiving request packets, then the huge wasting time (up to 90% of the total transmission time) of waiting for the client to process each reply packet will be eliminated. This solution is more speed gained than the first one, but the modification in hardware is more complicated. Nevertheless, this will not be the standard NFS mechanism.

Here are some proposed solutions for the performance improvement of the system. Due to the limitation time of the thesis work, they could not be done. However, for the future system upgrading, they can be considered as the effective solutions.

7 Conclusion

The objectives of this thesis have been successfully fulfilled. The NFS functionality module has been implemented into the NFI system with the support for function from the timeout detector. Besides, the data transfer mechanism between the NFS system and the user interface board has also been implemented, making the availability of the interconnection between the two boards.

The NFS module has been implemented and combined with the available RTP module. These two modules are wrapped by the application wrapper module which multiplexes the data paths of the two modules into one unified path for the compatibility with the current interfaces of other relevant modules such as the PSUnit, the memory board interface module. The wrapper module has run properly by correctly switching between the usage of NFS module (in case of NFS replying the request from a Linux client) and RTP module (in case of playback or recording mode requesting by button pressed on the FlashPakII). The timeout detector also supports the NFS functionality by triggering an interrupt correctly when detecting the end of a read request sequence, stopping the whole system, turning it to the idle state waiting for a new NFS read request sequence from client. As a result, the video data from the memory board has been downloaded successfully to a requested client via NFS functionality.

Next, the data transfer mechanism has also been implemented successfully. Via this way of communication, the file table as well as other type of commands (e.g. NFS stop, NFS copy) are delivered correctly between NFI system and the user interface board. With the support of the common register space implemented in application wrapper module, it is now possible for two boards to send and receive data/command by accessing these registers using DCR (for NFI system) and I2C (for user interface board). By utilizing the advantage of the common register space, the challenge of the interface pin limitation has been overcome without using more pins. Two-way communication is reached with the flexibility in updating – only by adding more user-defined command format.

The software program on PowerPC runs properly with the new updated functions and procedures handling the new data transfer mechanism and the new NFS functionality. All file information contained in the memory board has been delivered exactly to a requested client via Linux *'ls'* command. The read request from the client also be recognized correctly (with correct take number and stream number) by the PowerPC. Consequently, the correct video data is sent out via NFS module.

All new implemented modules have been verified successfully with ModelSim and ChipScope. The NFS functionality has been verified correctly with correct video data received by the client.

References

[1] Charles A. Poynton, *Digital Video and HDTV*, Morgan Kaufmann. ISBN 1-55860-792-7, 2003

[2] Thomas Brune, Kai Dorau and Holger Kropp, *10GEth-IO-Module system specification revision 1.5*, Deutsche Thomson Brandt GmbH Hannover, September 2007

[3] Zimmermann, Hubert, *OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection,* IEEE Transactions on Communications (Volume 28, Issue 4), pp. 425–432, April 1980

[4] *Device Control Register Bus 3.5 Architecture Specifications*, IBM Inc., Jan 2006

[5] Dominique Paret, *The I2C Bus : From Theory to Practice*, ISBN 978-0-471-96268-7, 1997

[6] Russel Sandberg, *The Sun Network Filesystem: Design, Implementation and Experience,* Sun Microsystems, Inc.

[7] Bougerol, A. ; Miller, F. ; Buard, N., *SDRAM Architecture & Single Event Effects Revealed with Laser*, On-Line Testing Symposium, 2008. IOLTS '08. 14th IEEE International , pp. 283-288, 2008

[8] *Virtex-5 FPGA User Guide*, UG190 (v4.2), Xilinx Inc., San Jose, CA, May 2008

[9] Bradley F. Dutton and Charley E. Stroud, *Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs,* Proc. IEEE Southeastern Symp. on System Theory, pp. 230-234, 2009

[10] Peter J. Ashenden and Jim Lewis, *The Designer's Guide to VHDL,* Third Edition, Systems on Silicon, ISBN 0-1208-8785-1, 2008

[11] Peter J. Ashenden, *The VHDL cookbook*, First edition, July 1990

[12] Perry, Douglas L., *VHDL programming by example,* Fourth Edition, ISBN 0-07-140070-2

[13] *ISE In-Depth Tutorial version 10.1*, Xilinx Inc., Copyright 1995 – 2007

[14] *EDK concepts, tools, and techniques,* UG863, Xilinx Inc., April 13, 2011

[15] *XST User Guide*, UG627 (v11.3), Xilinx Inc., San Jose, CA, Sep 2009

[16] *ModelSim SE user manual version 6.0b*, Mentor Graphics Corporation, Published 15 Nov 2004

[17] *ChipScope Pro Software and Cores User Guide*, UG029 (v13.4), Xilinx Inc., San Jose, CA, Jan 2012

[18] *IEEE Std 1149.1 (JTAG) Testability*, Texas Instruments Inc., 1997

[19] Kerr, Douglas A., *Chrominance Subsampling in Digital Images,* Jan 2012

[20] *ModelSim® Tutorial*, Mentor Graphics Corporation, ©1991 – 2008

[21] Bob Zeidman, *The death of the structured ASIC*, Chip Design Magazine, April 2006