Report Number: WUCS-01-10

2001-01-01

Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware

Florian Braun, John Lockwood, and Marcel Waldvogel

The ongoing increases of line speed in the Internet backbone combined with the need for increased functionality of network devices presents a major challenge. These demands call for the use of reprogrammable hardware to provide the required flexible, high-speed functionality, at all network layers. The Field Programmable Port Extender (FPX) provides such an environment for development of networking components in reprogrammable hardware. We present a framework to streamline and simplify networking applications that process ATM cells, AAL5 frames, Internet Protocol (IP) packets and UDP datagrams directly in hardware.

... Read complete abstract on page 2.

Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware

Florian Braun, John Lockwood, and Marcel Waldvogel

Complete Abstract:

The ongoing increases of line speed in the Internet backbone combined with the need for increased functionality of network devices presents a major challenge. These demands call for the use of reprogrammable hardware to provide the required flexible, high-speed functionality, at all network layers. The Field Programmable Port Extender (FPX) provides such an environment for development of networking components in reprogrammable hardware. We present a framework to streamline and simplify networking applications that process ATM cells, AAL5 frames, Internet Protocol (IP) packets and UDP datagrams directly in hardware.

Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware

Florian Braun, John Lockwood and Marcel Waldvogel

WUCS-01-10

June 2001

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware

Florian Braun John Lockwood Marcel Waldvogel
Technical Report WU-CS-01-10
Applied Research Laboratory*
Washington University in St. Louis

June 18, 2001

Abstract

The ongoing increases of line speed in the Internet backbone combined with the need for increased functionality of network devices presents a major challenge. These demands call for the use of reprogrammable hardware to provide the required flexible, high-speed functionality, at all network layers. The Field Programmable Port Extender (FPX) provides such an environment for development of networking components in reprogrammable hardware. We present a framework to streamline and simplify networking applications that process ATM cells, AAL5 frames, Internet Protocol (IP) packets and UDP datagrams directly in hardware.

1 Introduction

In recent years, field programmable logic has become sufficiently capable to implement complex networking applications directly in hardware. The Field Programmable Port Extender has been implemented as a flexible platform for the processing of network data in hardware at multiple layers of the protocol stack. Layers are important for networks because they allow applications to be implemented at a level where the insignificant details are hidden. At the lowest layer, networks need to modify the raw data that passes between interfaces. At higher levels, the applications process variable length frames or packages as in the Internet Protocol. At the user-level, applications may transmit or receive messages in User Datagram Protocol messages. An important application for the network layer is routing and forwarding packets to other network nodes.

*This research was supported in part by NSF ANI-0096052 and Xilinx Corp.

2 Background

In the Applied Research Lab at Washington University in St. Louis, a rich set of hardware components and software for research in the field of ATM and active networking has been developed. The modules described in this document are primarily targeted to this kit, though the design is written in portable VHDL and could be used in any FPGA-based system.

2.1 Switch Fabric

The central component of this research environment is the Washington University Gigabit Switch (WUGS, [1]). It is a fully featured 8-port ATM switch, which is capable of handling up to 20 Gbps of network traffic. Each port is connected through a line card to the switch. The WUGS provides space to insert extension cards between the line cards and the switch itself.

2.2 Field Programmable Port Extender

The Field Programmable Port Extender (FPX, [2, 3]) provides reprogrammable logic for user applications. It uses the same interface as the switch to the line-card, so it can be inserted between these two cards, as illustrated in figure 1.

The FPX contains two FPGAs: the Network Interface Device (NID) and the Reprogrammable Application Device (RAD). The NID interconnects the WUGS, the line card and the RAD via a small switch. It also provides the logic to dynamically reprogram the RAD. The RAD can be programmed to hold user-defined modules. Hardware based processing of networking data is made possible that way. The RAD is also connected to two SRAM and two SDRAM components. The memory modules can be used to cache cell data or hold large tables. Figure 2 illustrates the major components on an FPX board.

2.3 FPX Modules

User applications are implemented on the RAD as modules. Modules are hardware components with a well-defined interface which communicate with the RAD and other infrastructure components. The basic data interface is a 32-bit wide, Utopia-like interface. The data bus carries ATM header information, as well as the payload of the cells. The other signals in the module interface are used for congestion control and to connect to memory controllers to access the off-chip memory. The complete module interface is documented in [4].

Usually, two application modules are present on the RAD. Typically, one handles data from the line card to the switch (ingress) and the other handles data from the switch to the line card (egress). As with the Transmutable Telecom System [5], modules can be replaced by reprogramming the FPGA in the system at any time. In the case of the FPX, this functionality occurs via partial reprogramming of the RAD FPGA. A reconfiguration component performs a handshaking protocol with the modules to prevent loss of data.

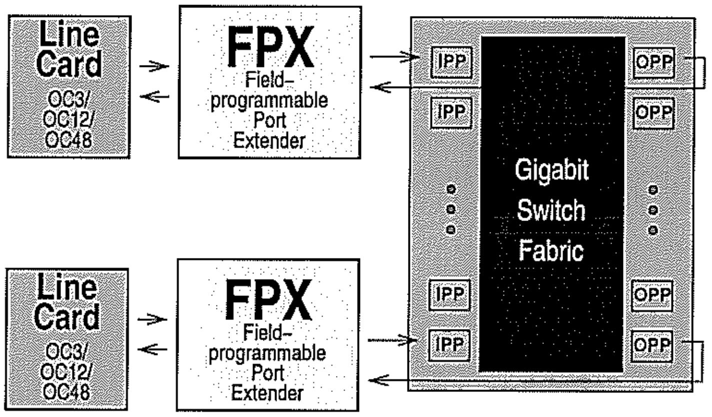


Figure 1: WUGS configuration using the Field Programmable Port Extender

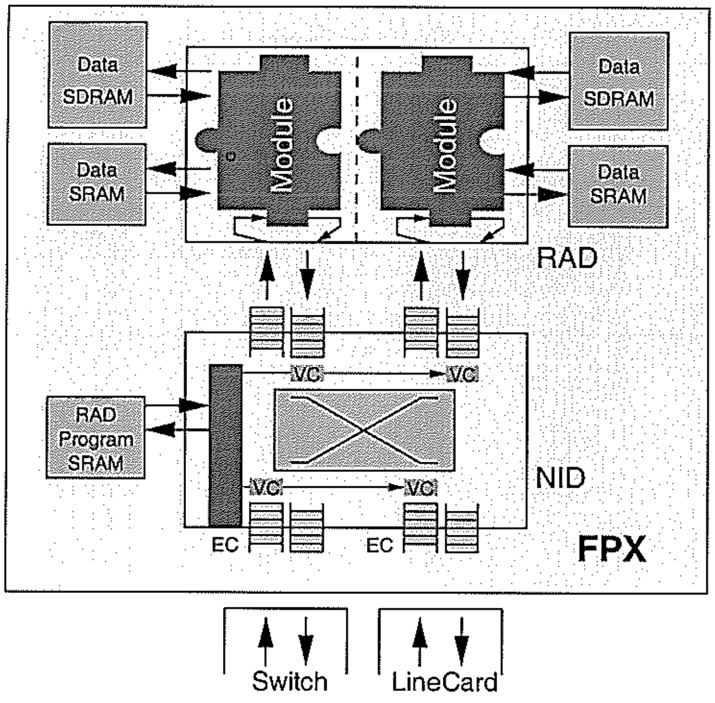


Figure 2: Components on an FPX board

3 Network Wrapper Concept

Network protocols are organized in layers. On the ATM data link layer, data is sent in fixed size cells. To provide variable length data exchange, a family of ATM Adaption Layers exists. Section 3.2.1 gives an introduction to the ATM Adaption Layer 5 (AAL5), which is widely used to transport IP data over ATM networks. The Network layer uses IP packets to support routing through multiple, physically separated networks.

Components have been developed for the FPX that allow applications to handle data on different levels of abstraction. A similar implementation exists for IP over Ethernet and the corresponding network layers [6]. On the cell level, a Start of Cell (SOC) signal is given to an application module. For AAL5 frame based applications, Start-of-Frame (SOF) and End-of-Frame (EOF) signals indicate the beginning or the end of an AAL5 frame, respectively. An additional data-enable signal indicates whether valid payload data is being sent.

Translation steps are necessary between layers. A classical approach would be to create components for each protocol translation, for instance from cell level to AAL5 frame level. There would need to be a component for the reverse step as well, in our example from the frame level back to the cell level, i.e., segmentation. In a new approach, we combine these two translation units into one component, which has four interfaces as a consequence: two to support the lower level protocol and two to provide a higher level interface, respectively. Also the two components are connected to each other. This is useful to exchange additional information or to bypass the application. Latter is done in the cell processor (section 3.1).

When an application module is embedded into the new translation unit, it gets a shape like the letter U (figure 3). Regarding the data stream, the application only connects to the translating component, which wraps up the application itself. Therefore we will refer to the surrounding components as *wrappers*.

To support higher levels of abstraction, the wrappers can be nested. Since each of them has a well defined interface for an outer and an inner protocol level, they fit together within each other, as shown in figure 3. As a result, we get a very modular design method to support applications for different protocols and levels of abstraction.

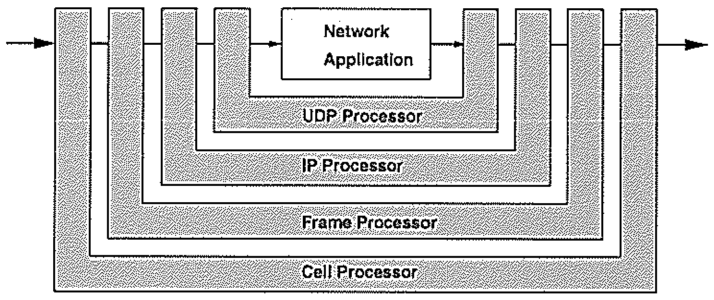


Figure 3: Wrapper concept

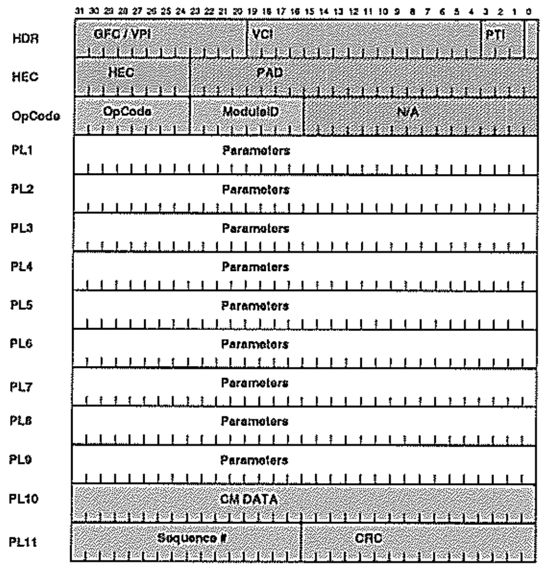


Figure 4: Control cell format

Associating each wrapper with a specific protocol, we get a layer model comparable to the well known OSI/ISO networking reference model. This modularity gives application developers more freedom in their designs. They can choose the level of abstraction they need for their specific application, while not needing to deal with the handling of complicated protocol issues, like frame boundaries or checksums.

3.1 Cell based processing

At the lowest level of abstraction, data is sent in fixed length cells. Applications or wrappers working on that protocol level typically process the ATM header and filter cells by their virtual channel.

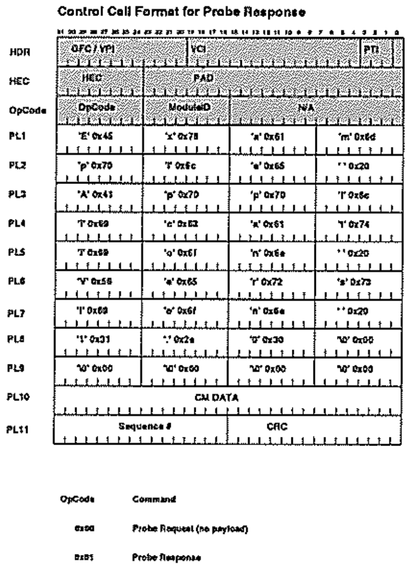
3.1.1 Control cells

FPX Modules communicate with software via control cells. These are ATM cells with a well-defined structure and are used to perform remote configuration.

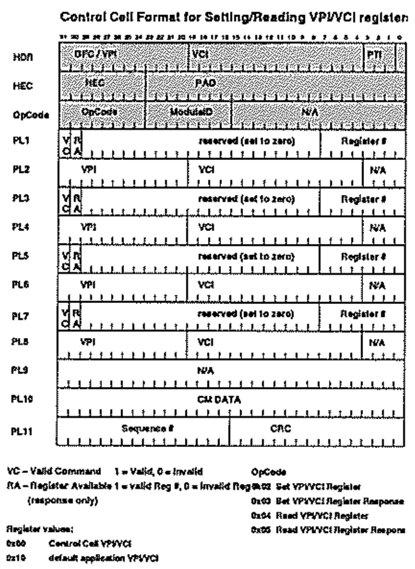
Control cells contain an opcode field, multiple parameter fields, a sequence number, and a 16-bit CRC, to ensure data integrity (figure 4). They are sent on specific virtual channels, defaulting to VCI 34 for the NID and VCI 35 for the RAD.

Control cells to the NID are used to setup the routes between the line card, the switch and user applications on the RAD. They are also used to upload new application modules to the RAD.

Control cells to the RAD contain an additional field, the module ID, to address the



(a) Probe Response


Control Cell Format for Setting/Reading VPI/VCI registers
HDR
HEC
OpCode
PL1
PL2
PL3
PL4
PL5
PL6
PL7
PL8
PL9
PL10
PL11
GFC / VPI
VCI
PTI
HEC
PAD
OpCode
ModuleID
N/A
VC
RA
reserved (set to zero)
Register #
VPI
VCI
N/A
CM DATA
Sequence #
CRC
VC – Valid Command 1 = Valid, 0 = Invalid
RA – Register Available 1 = valid Reg #, 0 = invalid Reg #
(response only)
Register values:
0x00 Control Cell VPI/VCI
0x10 default application VPI/VCI
OpCode
0x02 Set VPI/VCI Register
0x03 Set VPI/VCI Register Response
0x04 Read VPI/VCI Register
0x05 Read VPI/VCI Register Respons

(b) Set/Read VPI/VCI Registers

Figure 5: Standard Control Cell Opcodes

application module. Some standard opcodes are understood by all FPX modules, like writing to VPI/VCI registers, so that applications can operate on any virtual channel. Therefore opcodes in the range 0x00 to 0x0F are allocated for common use. Opcodes from 0x10 to 0xFF can be used for user applications. So far the following opcodes have been defined for common usage:

1. Opcode 0x00 is used as a "Probe Request". Applications should response with the "Probe Response" (0x01) and an identification string. This mechanism is used by the control software to query the configuration on the FPX.The control cell structure can be seen in figure 5(a).

2. Opcodes 0x02 and 0x04 are used to setup and query VPI/VCI registers. Applications can be configured to operate on any virtual channel by writing to one of these registers. Registers in the range from 0x00 to 0x0F are again allocated for common usage, while register numbers from 0x10 to 0xFF can be used by user applications. Register 0x00 defines the virtual channel on which control cells are sent, while 0x10 should be used as the application's default channel. The control cell structure can be seen in figure 5(b).

The control cell processor (CCP) is a standard FPX module with the hardcoded module ID zero. It is connected to all four off-chip memories and understands how to

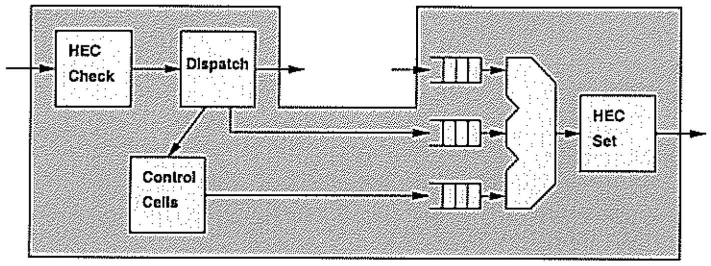


Figure 6: Cell processor

process specific control cells which are read from and written to them. It is useful for applications, which need to setup lookup tables in memory. With the CCP, no special code has to be written to modify these tables.

3.1.2 FPX Cell Processor

The wrapper on the lowest level is the cell processor (figure 6). It performs every necessary step on the cell level that is common to all FPX modules. First of all, incoming ATM cells are checked against their Header Error Control (HEC) field, which is part of the 5 octet header. An 8 bit CRC is used to prevent errored cells from being misrouted. If the check fails, the cell is dropped.

Accepted cells are queried about their virtual channel information in the next step. The cell processor distinguishes between three different flows:

1. The cell is on the data VC for this module. In this case, the cell will be forwarded to the inner interface of the wrapper and thus to the application.
2. The cell is on the control cell VC and is tagged with the correct module ID. Control cells are processed by the cell processor itself. We will cover this mechanism later in this section.
3. None of the above, i.e. this cell is not destined for this module. These cells are bypassed and take a shortcut to the output of the cell processor.

The cell processor provides three FIFOs to buffer cells from either of the three paths. A multiplexer combines them and forwards the cells to their last stop. Just before they leave the cell processor, a new HEC is computed.

The control cell handling inside the cell processor is designed to be very flexible, thus making it easy for application developers to extend its functionality to fit the needs of their modules. Since user applications will typically support more control cell opcodes than the standard codes, extendibility was an important goal in the design of the cell processor.

A control cell processing framework takes care of CRC checking and setting, buffering of common data structures and providing common information. A master state machine waits for control cells destined for this module and will then store opcode, user

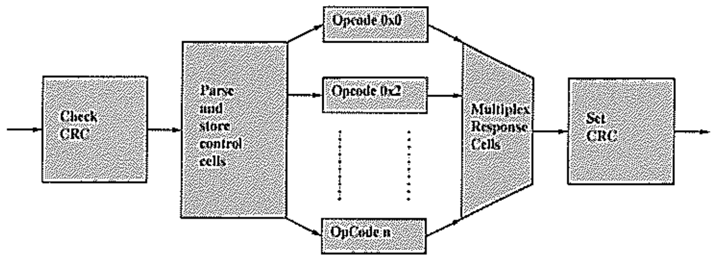


Figure 7: Control Cell Handling

data, CM field and sequence number, while at the same time the CRC is being checked. Every opcode has its own state machine. So adding a new command does not interfere with existing ones. Every state machine polls the signal *state*, if a control cell with a valid CRC ($crc\_ok =' 1'$) has been read ($state = CRC$) and gets active on its opcode. For any incoming control cell (request), a response cell should be sent, if the command has been processed successfully. Because there is a state machine for every opcode, which generates its own response cell, a multiplexer takes care of forwarding the correct one to the output port. Every process for a control cell opcodes sends its response cells out on *data_XX* and indicates the start on *soc_XX*, where *XX* is the opcode. The process *ccselection* checks all *soc_XX* signals and forwards new control cells. They get a valid CRC before they are forwarded to the cell multiplexer. The currently selected opcode is presented in *opcode_sel*. Processes should check this signal to make sure their cell has been transmitted. A diagram for the cell processing framework can be seen in figure 7.

The process *sm00* is responsible for detecting Probe Requests (0x00), while the process *data00_out* generates the Probe Response cell. The default string is "Generic Cell Processor 1.0". To change the Response string this part needs to be updated.

The processes *sm02* and *sm04* are responsible for setting (0x02) and reading (0x04) VPI/VCI registers. The response cell is generated by *data04_out* in both cases. For opcode 0x02 the values are written to the registers just before they are read again by sm04 for the response. Since the register values for the acknowledgment are always read from the actual register, this is a good check to see if the write operation was successful. To support additional registers these processes need to be changed.

3.2 Frame based processing

To handle data with arbitrary length over ATM networks, data is organized in frames, which are sent as multiple cells. Several adaption layers have been specified, which differ in the property of being connection-oriented or connectionless, in the ability to multiplex several protocols over one virtual channel and to reorder cells during transmission.

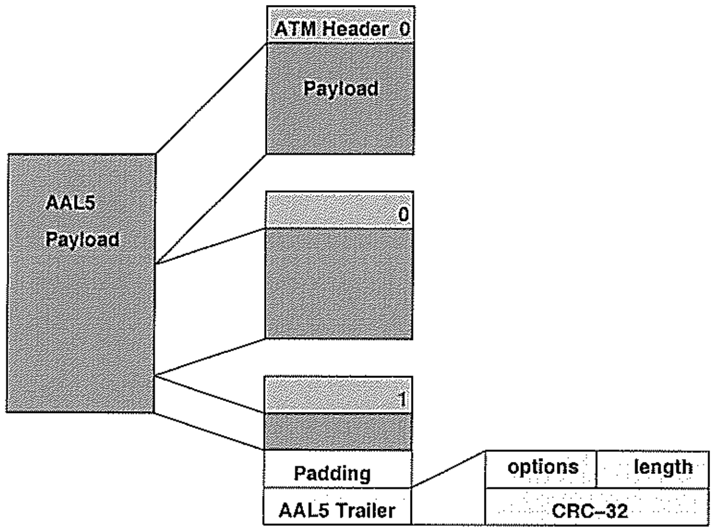


Figure 8: AAL5 frame segmentation

3.2.1 ATM Adaption Layer 5

In this section we will briefly describe the ATM Adaption Layer 5 (AAL5, [7, 8]), which is widely used for IP networks and is also one of the simpler protocols.

In AAL5 datagrams or frames of arbitrary length are put into protocol data units (PDU). In the simplest case (Flow Type 0), the frame starts at the beginning of the PDU, but it is also possible to prepend an additional header to distinguish between several protocols. We will only discuss the simple case in this document. A PDU's length is always a multiple of 48 octets, because a PDU is sent as a multiple of ATM cells. One bit in the ATM header, the user bit of the PTI field, is used to indicate whether a cell is the last one of a PDU. The last 8 octets of the PDU are used by a trailer, which contains information about the actual length of the frame and a 32 bit CRC to ensure data integrity. Any gap between the frame and the trailer is filled with padding. Since PDUs are multiples of 48 octets, the trailer always ends at a cell boundary and can therefore be located. The segmentation of frames with AAL5 is illustrated in figure 8.

3.2.2 FPX Frame Processor

The frame processor is a wrapper module for the FPX to handle AAL5 frame data. Its interface is designed to give application modules a more abstract view of the data. The frame processor replaces the Start-of-Cell signal with three signals (figure 14, namely Start-of-Frame (SOF), End-of-Frame (EOF) and Data-Enable (DataEn).

As the name indicates, SOF indicates the transmission of a new frame. Note that the Header-Error-Control (HEC) is not available with this wrapper. It is assumed that only valid ATM cells are passed to this wrapper and that valid HECs are generated from outgoing cells.

The Data-Enable signal indicates valid payload data. It can be seen as an enable signal for the data processing application. It is completely independent from the cell structure. Applications can therefore resize frames or append data very easily. Also generating new frames is now more convenient. Note that the Data-Enable signal is not asserted when padding is sent, since it is not considered a part of the frame. The End-of-Frame signal is asserted with the last valid payload word being sent. Applications thus have enough time to start appending data to a frame, if necessary.

After the EOF signal, two more words are sent, while DataEnable is still hi. These 8 octets represent the AAL5 trailer, but the interpretation is different. The first word contains the length and the option field. While the option field is simply copied to the new frame, the length field is not taken literally. The actual length of the frame can be determined from the three signals SOF, EOF and DataEn. Since the FPX uses 32 bits data width internally, it is only accurate to 4 octets, though. Thus the lower two bits of the length field are used to determine the valid octets within one 32 bit word. These two bits must be set by the application if the frame length changes by a number of octets not divisible by 4. The high-order bits of this field are changed by the frame processor according to the actual length of the frame. The second word is used to handle data integrity, i.e., the CRC-32. The frame processor handles the CRC-32 of AAL5 frames for the application. This happens right after cells enter the wrapper and just before they leave it again. The application sees an all-zero word if the frame is correct, otherwise the CRC field is replaced with a non-zero value. It is essential that applications copy and forward the two additional words.

The frame processor maintains a simple state machine to keep track of frame boundaries and to generate the Start-of-Frame and End-of-Frame. Recall that only one bit is used to mark the last cell of a frame. At first sight it seems to be simple to generate Data-Enable – assert the signal during the last 12 words of each cell, i.e., 48 payload octets. But in that case padding will also be recognized as valid data. Instead the frame processor buffers the last cell of a frame and waits for the length field before it forwards that cell. In fact, only the last two words of every cell but the last one have to be deferred for the special case, that the last word is padding only. So DataEn and EOF can be determined appropriately.

On the output side of the processor, data is accumulated in a buffer, until either the size of one cell has been reached or the total size of the frame has been determined. The cells are then sent out. Since the ATM header is only given once together with the SOF signal, it is copied and prepended to all generated cells, while the user bit of the PTI field is set appropriately to indicate the last cell.

3.2.3 CRC issues

AAL5 frames are secured with a CRC-32 sent with the last cell. The CRC protects the payload data from transmission errors and also detects dropped cells.

Recall that there is no sequence number for cells in AAL5. Usually the data integrity is checked before anything is passed to applications. In case the check gives a negative result, the frame is being dropped. This approach requires buffering of the whole frame, before the CRC can be checked. That results in a long delay before data can be forwarded to the application. On the other hand outbound cells can be forwarded immediately, since the (new) CRC is simply appended to the frame.

The FrameProcessor replaces the CRC field with an indication, as to whether the packet is erroneous. On outbound data the frame processor computes a new CRC for the (possibly) new payload. Under normal circumstances, outgoing frames will have a correct CRC. On erroneous frames the indication will flip some bits of the computed CRC and thus invalidate the frame. The receiving node can still detect errors and ignore the frame. With our approach we can speed up frame processing and save buffer resources in normal cases, while transmission errors can still be detected.

3.3 IP Packet Processing

The Internet Protocol is a very popular communication means across several networks. It uses packets on the lowest levels. Sub-protocols, like UDP or TCP are used to send datagrams or establish reliable connections.

3.3.1 The FPX IP Processor

The IP processor was developed to support IP based applications. It inherits the signalling interface from the frame processor and adds a Start-of-Payload (SOP) signal, to indicate the payload after the IP header, which can be of variable length. This wrapper serves three primary functions:

1. Checking the IP header integrity, i.e., the correctness of the header checksum. Corrupt packets are dropped.

2. Decrementing the Time To Live (TTL) field. As of RFC 1812 [9] all IP processing entities are required to decrement this field. Once this field reaches zero, the packet should not be forwarded any more. This is to prevent packets from looping around in networks due to mis-configured routers.

3. Recompute the length and the header checksum on outgoing IP packets.

An IP header usually has the length of 20 bytes, or 5 words.[1] The whole header has to have passed before any decision about its integrity can be made. The IP Processor computes and then compares the header checksum. On a failure, the IP-packet is dropped by not propagating any signal to the application. If the Time-To-Live field of an incoming packet is already zero, the packet is also dropped and an ICMP packet is sent instead. Otherwise the TTL field is decremented by one. On outgoing IP packets the length field in the header and the header checksum are set accordingly. Therefore a whole packet has to be buffered, before it can be sent out.

To save and share resources with other wrappers, the IP wrapper understands an updating protocol. The IP processor can apply arbitrary changes to the packet payload for other wrappers. Updating commands are optional and are inserted between the last payload word (EOF signal asserted hi) and the AAL5 trailer. An unused bit (15) in the AAL5 length field is used to indicate update words or the start of the trailer. The length

[1]This applies to the vast majority of IP packets that do not contain any IP options.

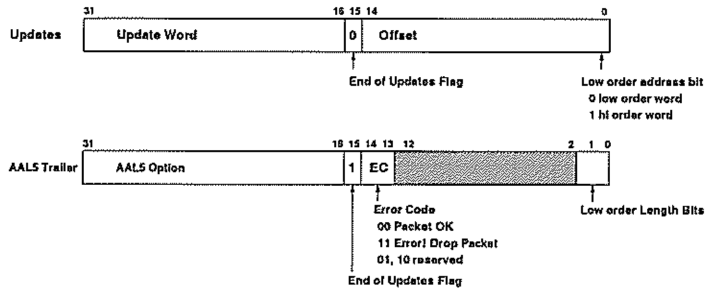


Figure 9: IP update protocol

field is also used to hold an error code, so that packets can be dropped before they are sent out. Update words contain a 16 bit update field and a 15 bit update offset address. The 16 bit word at the offset address in the buffer is replaced by the update field. The format of these field is illustrated in figure 9.

3.3.2 The FPX UDP Processor

The UDP processor is a wrapper to support user datagrams over IP. This wrapper takes care of the UDP checksum and the length field in the header for outgoing datagrams. Incoming datagrams are also checked for the checksum, but the result is only available at the end, when the whole packet has passed through the wrapper. The UDP processor uses almost the same signals as the IP processor, only replacing the SOP signal with the Start-of-Datagram (SOD) signal. Applications can simply process datagrams or even generate new ones without being concerned about correct header values.

To determine the correct checksum for outgoing datagrams, the whole packet should be buffered. Since the IP processor already buffers a full IP packet, this seems to be a waste of resources. Therefore the UDP processor informs the IP processor about necessary updates in the packet and leaves the buffering to that wrapper. This way only resources for one buffer have to be allocated. Recall that the lower level wrappers can process data on the fly and therefore only need small buffers.

4 Implementation Results

Our framework is designed for the FPX. The system clock on the FPX is 100 MHz and the FPGA used is a Xilinx Virtex E 1000-7. The following table gives the size (in lookup tables) and the maximum speed of our components on the FPX hardware.

Wrapper/Module	LUTs	Speed (MHz)
Cell Processor	760	118
Frame Processor	312	116
IP Processor	680	109
UDP Processor	368	114

5 Conclusions

We have presented a framework for IP packet processing applications in hardware. Although our current implementation was directed for use in the Field Programmable Port Extender, the framework is very general and can easily be adapted to other platforms. We introduce the concept of U-shaped wrappers, where each handles a particular protocol level. Unlike the traditional pre-/postprocessor separation, the U shape allows to put a component logically together, increasing the flexibility and reducing the number of cross-dependencies. The common interface between layers also lowers the learning curve.

The framework is useful for application developers, who are designing in the area of IP networking and ATM. Applications themselves don't have to take about network protocol issues. The complete IP processing framework only utilizes 8% of the RAD FPGA, leaving enough space for complex networking applications.

A Interface Description

The interface of the wrappers is derived from the FPX module interface [4]. The common part consists of the signals *CLK*, *Reset_l*, *Enable_l* and *Ready_l*. The datapath interface is split into an inner and an outer interface. The outer interface uses an analog naming convention as for FPX modules, i.e. incoming signals are named *xxx_MOD_IN* (e.g. *SOC_MOD_IN*), while outgoing signals use *xxx_OUT_MOD* (e.g. *SOC_OUT_MOD*). The inner interface connects the wrapper to the application or a higher level wrapper. The naming convention here is *xxx_OUT_APPL* (e.g. *SOC_-OUT_APPL*) for signals to the application, and *xxx_APPL_IN* (e.g. *SOC_APPL_IN*) for signals coming back into the wrapper. None of the wrappers connects to neither the SRAM nor the SDRAM controller. These interfaces can be used by the application.

The interfaces for all wrappers are shown in Figures 10–13. The signals are described below, while a timing diagram can be seen in figure 14.

CLK

This is the clock of the FPX (i.e. 100 MHz). All signals are synchronous to this clock.

Reset_l

This is the reset of the FPX. It is a synchronous, active low reset which is asserted low for one clock cycle.

Enable_l

This signal is used to enable the wrapper. See the RAD module interface documentation [4] for further details.

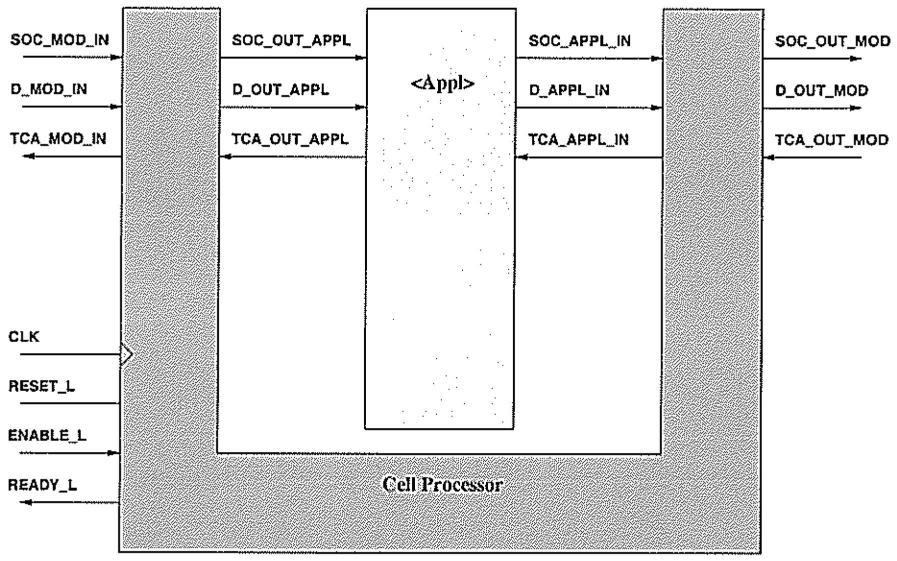


Figure 10: Cell Processor Interface

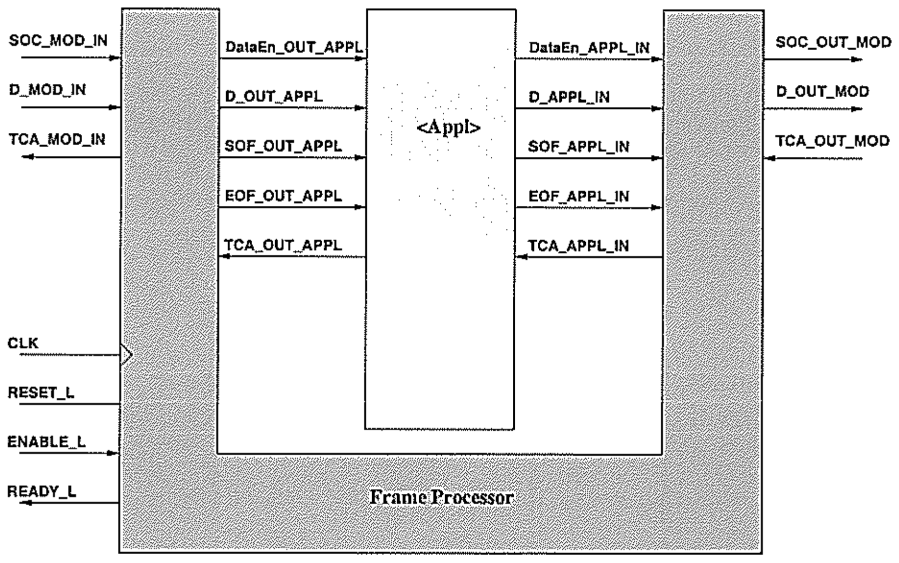


Figure 11: Frame Processor Interface

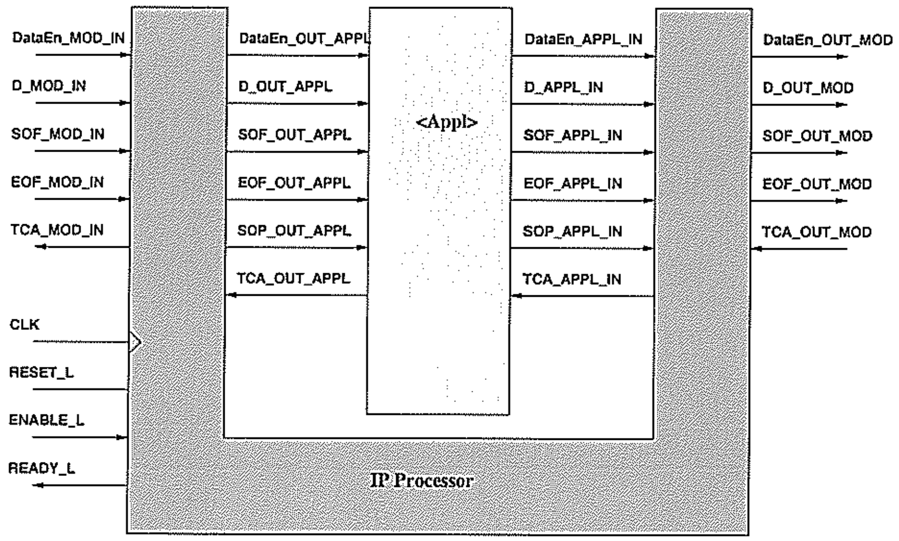


Figure 12: IP Processor Interface

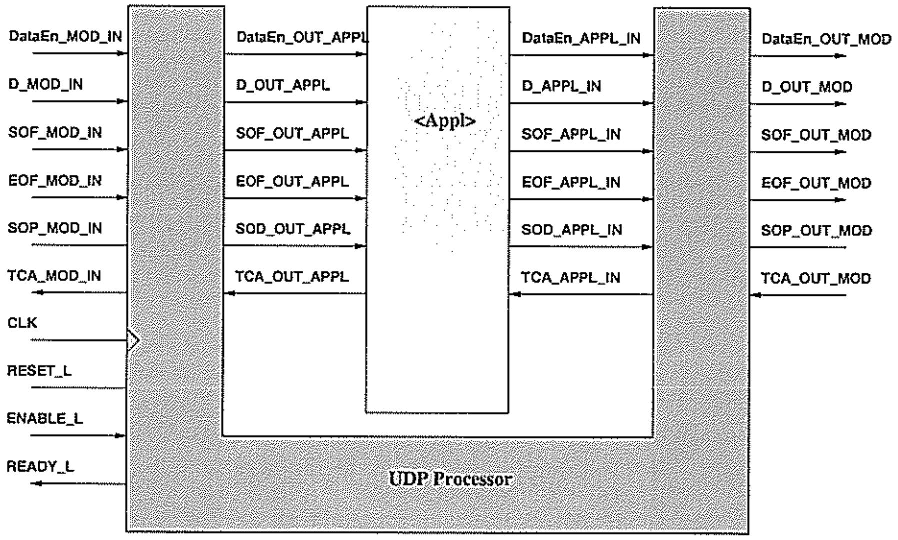


Figure 13: UDP Processor Interface

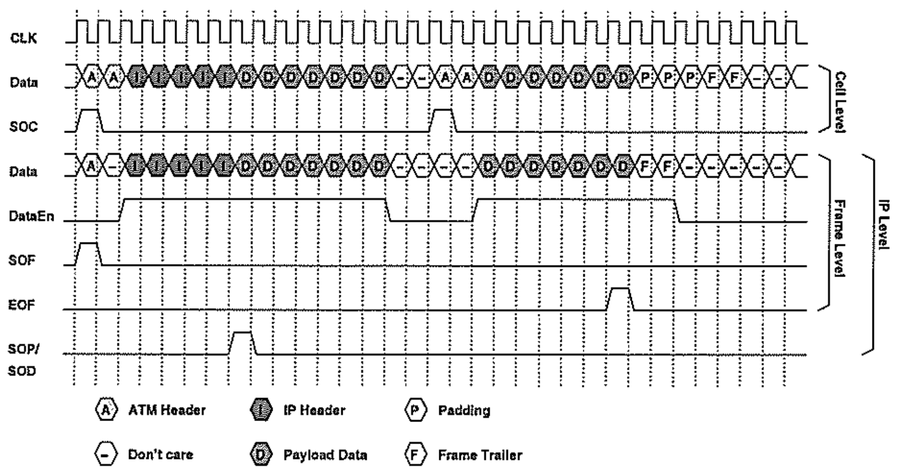


Figure 14: Timing diagram for different signalling protocols

Ready_I

This signal performs the handshake in response to the Enable_I signal. After Enable_I is deasserted hi and after all buffers are flushed this signal is asserted low. See the RAD module interface documentation [4] for further details.

D_MOD_IN, -OUT_APPL, -APPL_IN, -OUT_MOD

This is a 32 bit databus which feeds the module with data. The data format is depending on the wrapper.

- The Cell Processor and the outer interface of the Frame Processor use a cell based data format. The first word of a cell is available when the SOC_xxx signal is asserted hi. A cell is exactly 14 words long (2 words header, 12 words payload). On each clock cycle one word is transmitted.
- The inner interface of the Frame Processor and all higher level wrappers are cell independent, i.e. valid data is only transmitted on this bus when the DataEn_xxx signal is asserted hi. The first word of the ATM header is available when the SOF_xxx signal is asserted hi (the DataEn_xxx signal is low then). After the EOF_xxx signal additional data is sent, which is described in the wrapper sections.

SOC_MOD_IN, -OUT_APPL, -APPL_IN, -OUT_MOD

The *Start of Cell* signal is asserted hi if a new cell is transmitted through the D_xxx bus. The first word of the cell is available on the data bus on the same clock cycle as

SOC_xxx is asserted. Otherwise this signal is set to lo. The ATM header is followed by the ATM HEC and 12 payload words.

SOF_MOD_IN, -OUT_APPL, -APPL_IN, -OUT_MOD

The *Start of Frame* signal is asserted hi when a new frame is transmitted on the D_xxx bus. The ATM header of the first cell is available on the bus when this signal is asserted hi. Following this peak the DataEn_xxx signal indicates valid frame payload.

EOF_MOD_IN, -OUT_APPL, -APPL_IN, -OUT_MOD

The *End of Frame* signal is asserted hi when the last payload word of a frame is sent. Following this signal usually two more words are sent on the bus: the option-/length-field and the CRC-field. For correct contents the CRC field is zero. The UDP Processor sends more update words to the IP Processor as described in section 3.3.

DataEn_MOD_IN, -OUT_APPL, -APPL_IN, -OUT_MOD

The *Data Enable* signal indicates if data on D_xxx is valid payload of a frame or an IP packet. While payload is sent this signal is hi, otherwise it is lo. This signal is also hi for the frame trailer.

SOP_MOD_IN, -OUT_APPL, -APPL_IN, -OUT_MOD

The *Start of Payload* signal is asserted hi when the first IP payload is transmitted on D_xxx. If the frame does not contain a valid IP packet this signal is not asserted at all.

SOD_MOD_IN, -OUT_APPL, -APPL_IN, -OUT_MOD

The *Start of Datagram* signal is asserted hi when the first UDP header word is transmitted on D_xxx. If the frame is not an IP packet or the IP packet is not a UDP datagram this signal is not asserted.

TCA_MOD_IN, -OUT_APPL, -APPL_IN, -OUT_MOD

This signal performs the handshake back to the input for cell transfer. When this signal is asserted high, the input is free to send cells. Holding this signal low prevents the input from sending cells.

References

[1] Tom Chaney, J. Andrew Fingerhut, Margaret Flucke, and Jonathan S. Turner. Design of a gigabit ATM switch. Technical Report WU-CS-96-07, Washington University in St. Louis, 1996.

[2] John W. Lockwood, Jon S. Turner, and David E. Taylor. Field programmable port extender (FPX) for distributed routing and queuing. In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, pages 137–144, Monterey, CA, USA, February 2000.

[3] John W. Lockwood, Naji Naufel, Jon S. Turner, and David E. Taylor. Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX). In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, pages 87–93, Monterey, CA, USA, February 2001.

[4] David E. Taylor, John W. Lockwood, and Naji Naufel. Rad module infrastructure of the field programmable port extender (fpx). `http://www.arl.wustl.-edu/arl/projects/fpx/references/`, January 2001.

[5] Toshiaki Miyazaki, Kazuhiro Shirakawa, Masaru Katayama, Takahiro Murooka, and Atsushi Takahara. A transmutable telecom system. In *Proceedings of Field-Programmable Logic and Applications*, pages 366–375, Tallinn, Estonia, August 1998.

[6] Hamish Fallside and Michael J. S. Smith. Internet connected FPL. In *Proceedings of Field-Programmable Logic and Applications*, pages 48–57, Villach, Austria, August 2000.

[7] Juha Heinanen. Multiprotocol encapsulation over ATM adaptation layer 5. Internet RFC 1483, July 1993.

[8] Peter Newman et al. Transmission of flow labelled IPv4 on ATM data links. Internet RFC 1954, May 1996.

[9] Fred Baker. Requirements for IP version 4 routers. Internet RFC 1812, June 1995.