Etherblade.net Ver.1

(universal ethernet encapsulator hardware core)

Vladimir Efimov

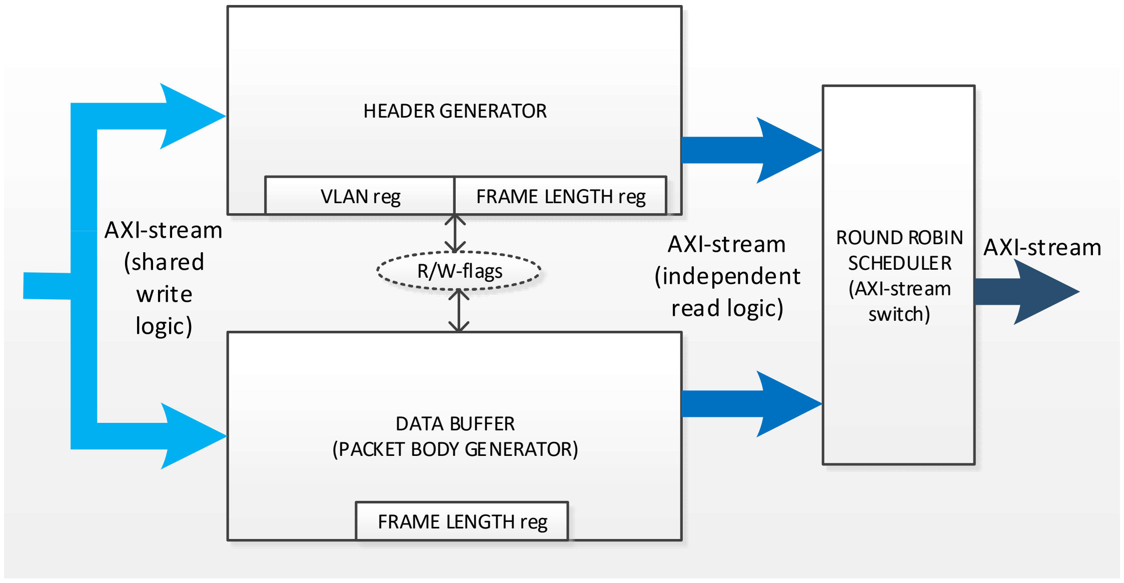
https://www.linkedin.com/in/vladimir-efimov

(Jun. 2018)

1. The high-level design of the "Etherblade.net – Ver.1":

The high level design of the encapsulator is depicted below and consists of the three main blocks: "HEADER GENERATOR", "DATA BUFFER" and "ROUND ROBIN SCHDEULER".

ENCAPSULATOR HIGH LEVEL DESIGN


HEADER GENERATOR
VLAN reg
FRAME LENGTH reg
AXI-stream (shared write logic)
R/W-flags
AXI-stream (independent read logic)
ROUND ROBIN SCHEDULER (AXI-stream switch)
AXI-stream
DATA BUFFER (PACKET BODY GENERATOR)
FRAME LENGTH reg

Both "HEADER GENERATOR" and "DATA BUFFER" blocks are equipped with a set of internal registers. The shared write logic writes the complete incoming Ethernet frame to the buffer memory in the "DATA BUFFER". At the same time, some "interesting" bytes of that frame such as "VLAN ID" are being written in parallel to the corresponding registers of "HEADER GENERATOR".

Separate sets of read logic of "HEADER GENERATOR" and "DATA BUFFER" blocks generate two independent AXI stream outputs assembled into the final stream of encapsulated datagrams by the "ROUND ROBIN SCHEDULER".

Let's look deeper into each of these blocks.

1.1 The **"ROUND-ROBIN SCHEDULER"** block assembles a header and a body data in order to produce final "encapsulated" datagrams. Upon reset of the system the "ROUND-ROBIN SCHEDULER" starts with the HEADER followed by the BODY and so on.

1.2 The **"DATA BUFFER"** block is a simple data storage block similar to the one implemented in "Ethernet hardware encapsulator - Ver.1" which is also known as "EHA_v1" project.

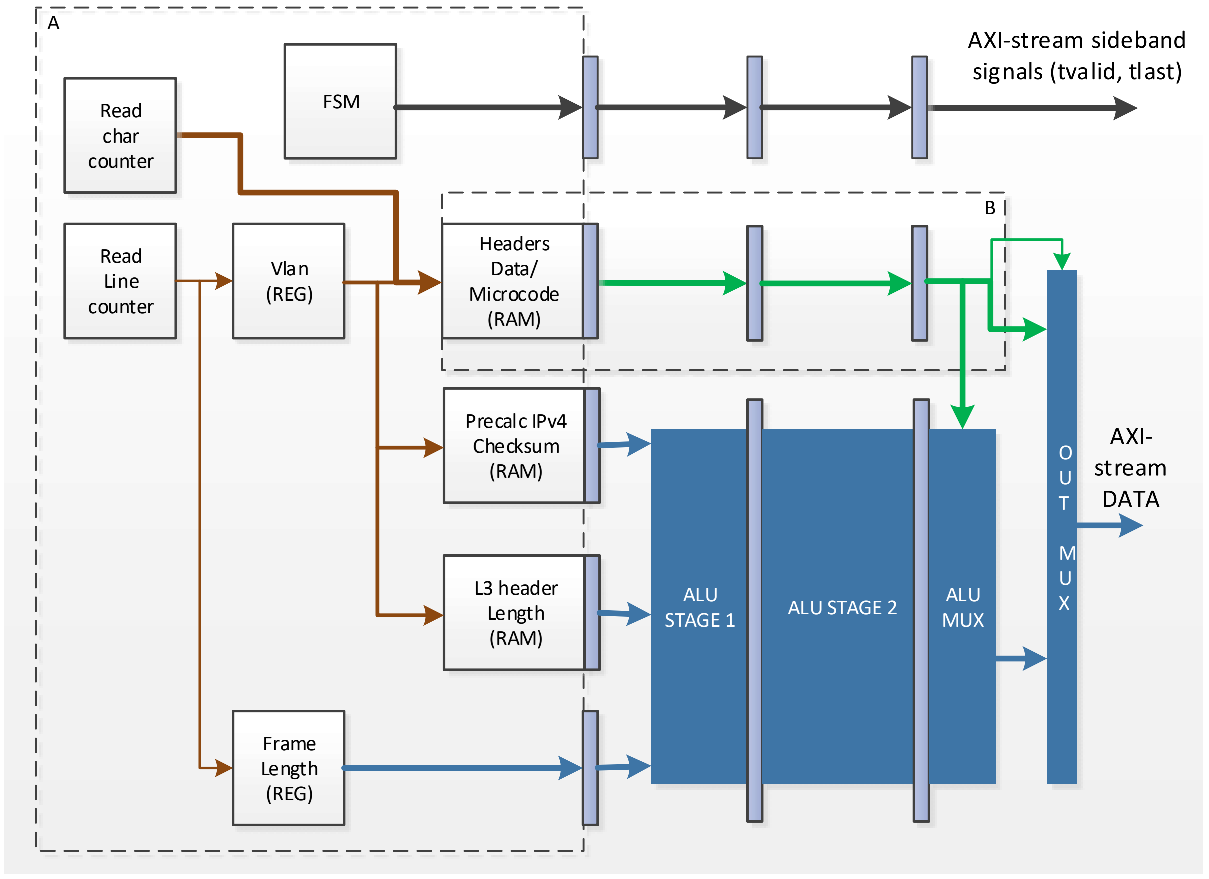
1.3 The **"R/W-flags"** is a shared read/write flags generation module. It is generating "read_allow" and "write_allow" signals for both "HEADER GENERATOR" and "DATA BUFFER" modules (see remark [1] for more details at the end of this chapter).

1.4 The **"HEADER GENERATOR"** block internal structure is shown below.

The role of the "HEADER GENERATOR" is to produce headers for the datagrams that have been stored in the "DATA_BUFFER". The "HEADER_GENERATOR" contains supplementary data such as "Vlan" number and "Frame length" value in its registers but not the datagram itself. There are three

RAM memory blocks programmed externally by a user with a data which is necessary for generating these headers.

“HEADER GENERATOR” PIPELINE ORGANISATION



The set of elements encircled in the “Box_A” provide the initial data that is being processed further in 3-stages pipeline (see remark [2] for more information about elements in “Box_A”). This 3-staged pipeline was introduced for the arithmetical logic unit – ALU (depicted in blue colour) to make it meet the timing constraints of the system. The ALU contains set of adders/invertors and its role to produce some values on its output that may be required in a header – these values could be “packet/payload length”, “IPv4 checksum” etc.

But most of the values in generated headers are static. The microcode format instructs when the symbol in the header shall be a statically defined byte or a value produced by the ALU shall take place in a header.

And this is depicted in the ”Box_B” where we can see how the format of the microcode symbol stored in the “Headers/Data microcode RAM” drives the set of hardware multiplexers to make the required value appear at the end of the pipeline at the output of the “HEADER GENERATOR” block (see remark [3] for information about the format of the microcode).

Remarks:

[1]. **R/W flags:** Writing into both "HEADER GENERATOR" and "DATA BUFFER" is allowed only when: ("HEADER_GENERATOR" is not FULL) and ("DATA_BUFFER" is not FULL);

Reading is independent and asynchronous which means that reading from "HEADER_GENERATOR" is happening when ("HEADER_GENERATOR" is not EMPTY) and reading from "DATA_BUFFER" is happening when ("DATA_BUFFER" is not EMPTY).

[2]. **"Box_A" elements:** Everything within "Box_A" is controlled by the FSM. The FSM and the pipeline is driven by the "Drive" condition produced from the signal on the output AXI stream interface. This is "Drive = ˜(tvalid & ˜tready)" – see "EHA_v1" project for general pipeline organisation.

[3]. **Microcode format and "Box_B" hardware:** The microcode is stored in the "data/microcode" RAM and each microcode symbol is 9bits wide. The 8 lower bits can either be a data byte (static value) that goes into a generated header as is or it can be an address of a "dynamic value" that shall be "picked up" from the ALU output. The most significant bit of the microcode symbol distinguishes the purpose of the remaining 8bits. This is clearly indicated in the "Box_B" where the bus is split in such a way that some of the wires become input signals for the set of output multiplexers while others are multiplexers drivers. The hardware in the "Box_B" demonstrates the match between the software (the format of the microcode) and the hardware (the wiring controlling hardware multiplexers).

2. Development stages of the project.

EBLADE_V1_1.0a – refactored store-and-forward buffer

This is refactored store-and-forward buffer developed in "EHA_v1" project.

As a result of the refactoring all "write logic" and "flags generator logic" became separate blocks as they will be shared among "HEADER GENERATOR" and "DATA BUFFER" modules.

Otherwise the functionality remains the same. If you send a datagram to the input, the same datagram will be presented at the output.

EBLADE_V1_2.0a – two store-and-forward buffers + round robin scheduler

This is further modification of the project (1.0a) where two store-and-forward buffers are placed in parallel with a shared input write logic, flags generators and newly developed "ROUND ROBIN SCHEDULER" module at the output.

When a datagram arrives via the input interface it gets written in parallel into both "DATA BUFFER" blocks and these two "DATA BUFFER" blocks produce two equal datagrams at their outputs which "ROUND_ROBIN_SCHEDULER" stitches together into a single double sized datagram presented at the output interface of the core.

EBLADE_V1_3.0a – header generator

In this project the “HEADER_GENERATOR” was developed and placed into the frame build in the project (1.0a) instead of the “DATA_BUFFER” block.

During the reception of the frame at the input interface all the necessary information is being written into the registers of the “HEADER_GENERATOR”. At the output of the module only the header for this frame is being generated.

EBLADE_V1_4.0a – final assembly

This project is based on (2.0a) where the first “DATA BUFFER” is replaced with the “HEADER_GENERATOR” developed in (3.0a).

EBLADE_V1_4.0a-vivado – wrapper for vivado

In this project an external wrapper has been developed for (4.0a). This wrapper makes the interface with Xilinx BRAM controllers so the core can be used in the vivado IP integrator where it can be embedded in a microprocessor based system-on-chip design.

For more information, please refer to project’s RTL code and diagrams on

https://etherblade.net