Latency analysis and comparison of FPGA implementations for a CSI-2 to Ethernet gateway using the processing system and a pure implementation in processing logic

Mario Altendorfer
Deggendorf Institute of Technology
Autonomous Systems / Driver Assistance Systems
Deggendorf, Germany
mario@altendorfer.it

***Abstract*—Many modern FPGAs not only consist of a Processing Logic (PL), but also contain a Processing System (PS), whose ARM processor is able to run a Linux operating system and communicate with the Processing Logic. Because programming the PL for complex problems is very time consuming, it is often advisable to implement algorithms in the PS. This can save development time when implementing the software in the FPGA platform. However, it should not be forgotten that the pure implementation of an algorithm in hardware (PL) can be strongly parallelized and thus a significantly shorter execution time can be achieved compared to the implementation in the processing system. In this article, using the example of an implementation of a CSI-2 (Camera Serial Interface) to Ethernet gateway, we investigate which speed increases can be achieved by a pure implementation in the PL of the FPGA compared to the implementation of the gateway in the PL + PS.**

***Index Terms*—CSI-2, Ethernet, FPGA, latency**

I. Introduction

To enable autonomous driving, vehicles must be equipped with modern driver assistance systems. These systems contain sensors that generate a large amount of raw sensor data. In the development of driver assistance systems it is necessary to record sensor data in order to control and improve the function of these systems or to develop new algorithms. The recording of the data is significantly simplified if the data is transmitted via a standardized interface that is already supported by as many end devices as possible. For this purpose, a measuring device is required that can receive the protocol of the sensors and pack the data into Ethernet frames.

There are already first approaches of a CSI-2 to Ethernet converter, from which a prototype has been developed, which is able to receive Gigabit Multimedia Serial Link (GMSL-2) data and to output them via two 10 Gbps Ethernet interfaces. This prototype is called Measurement Data Interface (MDI) and contains two deserializers for the conversion from GMSL-2 to Camera Serial Interface (CSI-2) and a Xilinx Zynq 7000 FPGA for the conversion from CSI-2 to Ethernet. In the already existing gateway, the image data transmitted via CSI-2 was received by the FPGA and processed in the processing system and sent via 10 Gbps Ethernet over UDP. Within the Master Applied Research Project the programming of the FPGA is to be revised. A special focus of the new development of the FPGA hardware description is to improve the program runtime. This optimization is to be achieved by reducing the amount of buffered data in the FPGA and by outsourcing the computing operation calculated in the ARM processor to the Processing Logic.

II. Overview of FPGA implementations

a) Implementation in the processing system: The FPGA implementation of the CSI-2 to Ethernet gateway in the processing system consists first of all of a CSI-2 IP core, which can receive and process CSI-2 data so that up to four pixels per clock can be processed by the FPGA via a parallel 96 bit output. This data is forwarded to a Direct Memory Access (DMA) and stored in Random-Access Memory (RAM) via the processing system. The processing of the data now takes place exclusively in PS. For this purpose, an entire image frame is temporarily stored in RAM and then the structure of the UDP frames is generated by the PS and forwarded to the Ethernet IP core via an Advanced eXtensible Interface Stream (AXIS). The core is able to transmit Ethernet frames from this AXIS over 10 Gbps. Because an entire image frame is temporarily stored in RAM and processing takes place in PS, which carries out serial data processing, a high latency time is expected in PS implementation. Fig. 1 shows a schematic representation of the PS implementation.

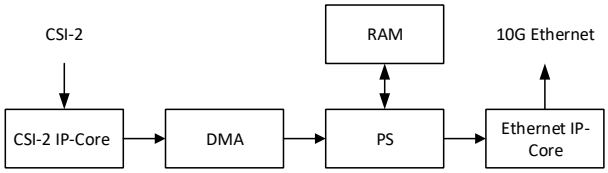

Fig. 1. Schematic representation of the PS implementation

It is a mixture of PL and PS implementation. The two IP cores are implemented in the PL, but the storage and processing of the image data takes place in PS.

b) Implementation in the processing logic: The second implementation takes place only in PL. This offers some advantages, because of this the implementation can be used with FPGAs without integrated PS, furthermore you are independent of an operating system and the latency should have a much lower jitter, because the timing of the circuit can be predicted exactly. This is not the case for a version with a conventional sequential controlled CPU.

First, the sensor data is received by the CSI-2 IP core and processed accordingly, so that it outputs a 96-bit wide data stream at a frequency of 200 MHz on each rising edge of the clock. The CSI-2 transmission is a parallel transmission, which in this application transfers the sensor data on four data lines. The CSI-2 IP core receives this data and once four pixels have been received, they are output on the next rising edge. This means that the output of the CSI-2 IP core does not always have to be continuously valid, as it operates at 200 MHz, which would allow a maximum data transfer rate of 19.2 Gbps, but CSI-2 only supports 6 Gbps over four data lines. Thus the output of the IP core clocks faster than the input. It should be noted, however, that the transmission at the output always takes place over a 96-bit wide stream that contains four pixels in a valid transmission. Thus a maximum pixel depth of up to 24 bits is supported. If this is less than 24 bits, 96 bits are transmitted in parallel, but the unused bits are marked as invalid.

Subsequently the data is temporarily stored in a FIFO. Once the data is buffered, the stream is converted from 96-bit to a 64-bit AXI stream, as this simplifies data transfer between modules and also expects a 64-bit AXIS at the end of the Ethernet IP Core processing. For a pixel depth of up to 16 bits, no special procedures are required, as only the valid data in the stream is linked to the AXIS. If the pixel depth is greater than 16 bits, more than 64 bits of valid data are transmitted per clock. This means that converting the 96-bit data stream to a 64-bit AXI stream is only possible if the data is transmitted over two clock cycles.

The transmission of an AXI stream is not only limited to the parallel transmission of payload data, additional signals are also available. These can be used to transmit further information. This allows the entire data of the stream to be declared valid or invalid. This is also possible for individual bytes of the AXI stream, so that n Bytes ($n \in 0, 1, 2, 3, 4, 5, 6, 7, 8$) of valid data can also be transmitted. A further signal with the designation tready, which is initiated by the slave of the transmission, can be used to inform the master whether it is ready to receive data. So the transmission can be monitored and also the slave can pause the transmission of the data of a master from a FIFO.

CSI-2 packs image lines into a CSI-2 long packet. This packet also includes a CSI-2 header that contains the data type, virtual channel, and the contained payload size in bytes of the image row. In addition, a timestamp is created at the time the image line is received in the CSI-2 IP core. This data is output at the beginning of the transmission of an image line via the data stream and is needed at a later time to create the frame header.

After the Conversion to AXIS module has been processed, the output AXI stream contains the CSI-2 header, a time stamp and the payload of an image line. The FIFO in packet mode ensures that at least one image line including all additional header data is buffered before proceeding with further processing of the data. In the process of pre-processing the data, the byte order can be adjusted in order to obtain a firmly defined structure of the data via Ethernet and to be able to interpret it correctly again at the receiver.

The last step is to create the frame header and create an AXIS for the Ethernet IP core. The challenge here is that it is quite common for an image line to have a larger payload than a standard Ethernet frame can contain. From this, an image line is divided into several Ethernet frames, each with a maximum payload of 1440 bytes. The size of 1440 bytes was chosen because this size is a number that has the values 1, 2, 3, 4, 5, 6 and 8 as common divisors. This makes it easier to divide an image line across multiple Ethernet frames, because regardless of the valid bytes m contained in the AXIS, it is guaranteed that n times m equals 1440 bytes at all times, so there is no need to divide data from a 64-bit stream at any time.

The generation of the AXI stream, which serves as input for the Ethernet IP core, is processed by a state machine. In the end, an Ethernet AXI Stream contains the entire Ethernet header and a maximum of 1440 bytes payload (the last frame of an image line can contain less than 1440 bytes payload). An Ethernet header containing additional information about the currently contained data is required so that the receiver can undo the division of an image line into several frames and reassemble the image from all image lines at the end. This ethernet frame is explained in chapter 3.

Fig. 2 contains the basic diagram of the implementation of the gateway in the Processing Logic.

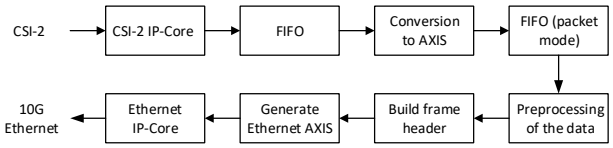

Fig. 2. Schematic representation of the PL implementation

III. Design IEEE1722 Ethernet frame format

For the transfer of the sensor data via Ethernet in the pure PL implementation a frame structure is to be designed, which has the necessary components for the transfer of the camera data and allows a simple calculation of the latency time. These include at least the standard Ethernet frame header (including VLAN tag), a time stamp, the line number and an internal line sequence number. In addition, the header of the CSI-2 packets should be transmitted via Ethernet and be part of

the designed frame header. The timestamp contained in the frame is the reception time of the CSI-2 data of the FPGA. By creating a time stamp when receiving the Ethernet frames at the measuring computer, the latency time can be calculated very easily by forming the difference.

The IEEE1722 standard describes a frame which is suitable for the transmission of audio and video data. In addition, an IEEE1722 frame offers the advantage that TSN (Time Sensitive Networking) procedures can be applied to the data traffic. Furthermore, the real-time capability of the transmission can be ensured. Due to these advantages, the designed Ethernet frame is based on the IEEE1722 standard frame. If the IEEE1722 standard frame is extended by the Ethernet header and the format-specific data, the frame structure shown in the figure results.

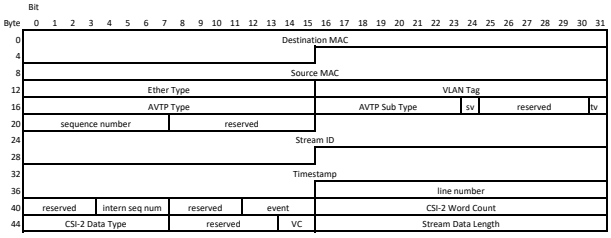

Fig. 3. Structure of the designed IEEE1722 frame

IV. Investigation of the measurement setup

Before the latency measurement can be carried out, the test setup and the available cameras must first be examined more closely. A total of four camera sensors are available for the measurement. These cameras are the OnSemi AR0820, Sony IMX490 and OmniVision OV10650 sensors. Furthermore, a 2.5 MP camera from Continental was available to measure the latency of the FPGA implementation in the processing system.

Tab. I contains an overview of the three camera sensors and their properties, which were used to measure the FPGA implementation in the Processing Logic.

TABLE I
OVERVIEW OF THE AVAILABLE CAMERA SENSORS

sensor	**resolution**	**Image line size**	**Pixel format**
AR0820	3840H x 2160V	5760 Byte	RAW12
IMX490	2880H x 1860V	5760 Byte	YUV
OV10650	1824H x 940V	3648 Byte	YUV

The selected cameras are sensors that have a GMSL-2 interface. Since two cameras differ in line size, the latency measurement can be used to determine whether this has an influence on the latency that occurs and, if so, to make a general statement about the latency time for any given line size. The general test setup used for both implementations consists of at least one camera sensor, an MDI and a Linux measurement computer. This measuring computer will record the network traffic using the application tcpdump and store the receive time stamp for each packet with an accuracy of nanoseconds.

V. Latency Measurement

The latency time measurement is particularly concerned with the measurement of the implementation in the Processing Logic, since this requires a much more accurate time synchronization and measurement. Nevertheless, we will briefly explain how the measurement works in general and which points in time are relevant for the latency measurement. If a picture line is received by the CSI-2 IP core, the hardware time of the FPGA is read out and stored into a local signal (t_{FPGA}). This time is the start of the time measurement. When the Ethernet data is received at the measurement computer, a time stamp $t_{receive}$ is also generated and the latency time ($t_{latency}$) can be calculated according to Eq. 1.

$$t_{latency} = t_{receive} - t_{FPGA} \tag{1}$$

The measurement of the latency time of the PS implementation differs from the PL implementation. Since in the PS implementation an entire image frame is buffered in the processing system, only one latency time can be created per image frame. Therefore a timestamp is created in the FPGA when the first image line of an image frame is received and additionally a timestamp when the first UDP data is received at the Linux measurement computer. The difference between the two timestamps finally results in the latency of the transmission. Since in the PL implementation an image frame is processed line by line, a latency time can be calculated for each line. For this purpose a time stamp is generated in the FPGA when an image line is recognized and additionally a time stamp is generated at the first received IEEE1722 fragment of this image line.

Fig. 4 shows the test setup for measuring the latency of the PL implementation. Here the MDI with the three Ethernet interfaces was connected to a switch that supports time synchronization. Then the switch is connected to the Linux computer with a 10 Gbps Ethernet link.

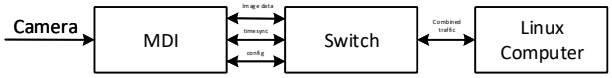

Fig. 4. Setup of the PL latency measurement

In this setup, the image data and time synchronization are each executed via a 10 Gbps Ethernet interface on the MDI. This is necessary because the current implementation does not allow a 10 Gbps Ethernet interface to send data and perform time synchronization (gPTP) simultaneously. For this reason the switch is necessary in the design. It collects the data traffic of the three Ethernet lines and forwards it to a 10 Gbps Ethernet interface to the measuring computer. This makes it possible to synchronize the hardware time of the FPGA and the hardware time of the Ethernet interface of the Linux computer to an accuracy of about 10 ns. First the latency measurement of the FPGA implementation in PS shall be considered. For this purpose the measurement result is shown in table II.

TABLE II
LATENCY MEASUREMENT IMPLEMENTATION IN PS

sensor	measurements	average	standard deviation
2.5 MP sensor	1578	39.75 ms	3.42 ms

As expected, there is a relatively high latency in the millisecond range, since an entire image frame must be buffered and the composition of the frames must be calculated serially by the ARM processor. Fig. 5 shows the graphical evaluation of the measurement result in comparison to the calculated normal distribution, which deviates from the real measurement curve due to the small number of measurements performed.

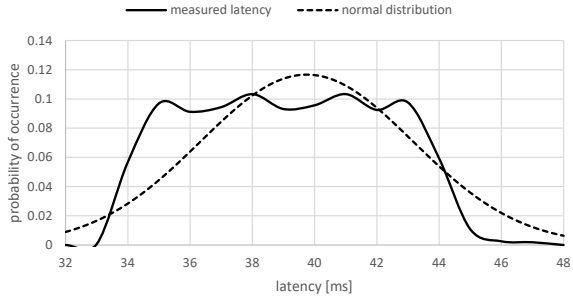

Fig. 5. statistical distribution of the measured PS latency

Now that the latency of the PS implementation is known, the measurement is performed with the three available cameras for the PL implementation. Tab. III shows the result of the latency measurement for which the mean value of the latency and the standard deviation were calculated.

TABLE III
LATENCY MEASUREMENT IMPLEMENTATION IN PL

sensor	measurements	average	standard deviation
AR0820	123858	18619 ns	21.87 ns
IMX490	123787	18245 ns	19.90 ns
OV10650	165043	14728 ns	18.51 ns

Due to the large number of measurements performed, the curve of the calculated normal distribution and the measured latency should now be very similar (see Fig. 6, 7 and 8).

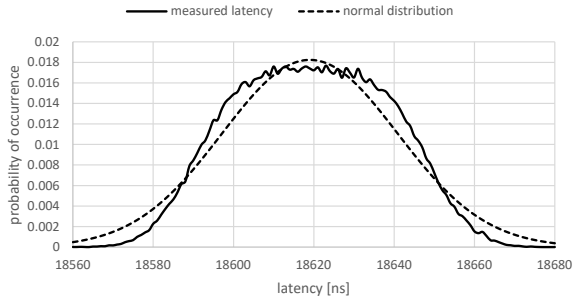

Fig. 6. statistical distribution of the AR0820 sensor PL latency

The latency for the PL implementation is significantly below the measured implementation of the PS implementation (about 1000 times faster). In comparison to the normal distribution, the maxium of the curves shows a slight plateu which indicates deviations in time synchronization.

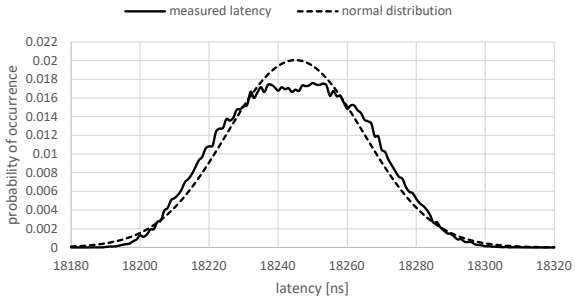

Fig. 7. statistical distribution of the IMX490 sensor PL latency

It can be seen that the latency of the two sensors AR0820 and IMX490 (see Fig. 6 and 7) show very similar latency behavior because they have an identical line size and therefore the processing in the FPGA requires an identical number of clock cycles. The sensor OV10650 has a line size of 3648 bytes and an average latency of 14728 ns. The IMX490 sensor is used as a reference for calculating a estimation of the overall latency time t as a function of the line size s [byte], since it has the same pixel format as the OV10650 sensor (see Eq. 2).

$$t_{latency} = 8653ns + s[byte] \cdot 1.6652 \frac{ns}{byte} \tag{2}$$

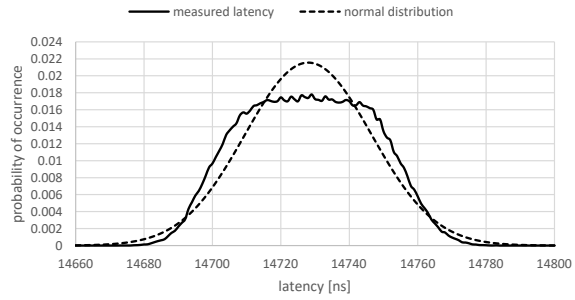

Fig. 8. statistical distribution of the OV10650 sensor PL latency

VI. CONCLUSION

By implementing an algorithm exclusively in PL, the execution time can be significantly reduced compared to an implementation in PS, but this performance gain must be weighed against the additional development effort.

REFERENCES

[1] AVTP, https://avnu.org/wp-content/uploads/2014/05/AVnu-AAA2C_Audio-Video-Transport-Protocol-AVTP_Dave-Olsen.pdf, 13.04.2020.
[2] AXI Reference Guide, https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/v13_4/ug761_axi_reference_guide.pdf, 20.01.2020.
[3] Camera Data Sheets, https://leopardimaging.com/, 13.04.2020.
[4] Crashkurs VHDL, https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.050/vorlesungen/sose09/lrob/Crashkurs_VHDL.pdf, 18.01.2020.
[5] FPGA, https://www.mikrocontroller.net/articles/FPGA, 13.04.2020.