2018 International Conference on Field-Programmable Technology (FPT)

High-Speed Computation of CRC Codes for FPGAs

Lukáš Kekely, Jakub Cabal
CESNET, a. l. e.
Zikova 4, 160 00 Prague 6
Czech Republic
{kekely, cabal}@cesnet.cz

Jan Kořenek
IT4Innovations Centre of Excellence, FIT BUT
Božetěchova 2, 612 66 Brno
Czech Republic
korenek@fit.vutbr.cz

***Abstract*—As the throughput of networks and memory interfaces is on a constant rise, there is a need for ever-faster error-detecting codes. Cyclic redundancy checks (CRC) are a common and widely used to ensure consistency or detect accidental changes of data. We propose a novel FPGA architecture for the computation of the CRC designed for general high-speed data transfers. Its key feature is allowing a processing of multiple independent data packets (transactions) in each clock cycle, what is a necessity for achieving high overall throughput on very wide data buses. Experimental results confirm that the proposed architecture reaches an effective throughput sufficient for utilization in multi-terabit Ethernet networks (over 2 Tbps or over 3000 Mpps) on a single Xilinx UltraScale+ FPGA.**

I. Introduction

The Cyclic Redundancy Check (CRC) codes are widely deployed in digital communications and storage systems [1], [2] to detect accidental error introduced into data. The binary data are divided into transactions (packets) and each transaction is subjected to a CRC which results in a fixed-length binary check sequence. The computed check sequence value is then attached to the original data to determine its correctness.

The mathematical background of CRC and forms of its hardware representation have been extensively studied [3], [4], [5] and they are not the focus of this paper. All we need to know is that processing of multiple bits is possible for any given CRC polynomial based on XOR equations set up for each output (code) bit that together form a CRC table. Furthermore, results of multiple CRC tables can be aggregated (accumulated) together to obtain code value of longer data.

Data packets usually have variable lengths and are not aligned with data bus words. Handling of unaligned ends/starts requires additional logic and more complex architectures than simple CRC table. Furthermore, as the data buses width is growing with throughput, multiple packets per clock cycle (data bus word) need to be processed. High-speed implementation addressing this issues is the main focus of our work.

We propose a novel FPGA architecture of CRC computation for general high-speed transfers of data packets with variable lengths. The architecture can produce multiple CRC values per clock cycle in a single pipeline thus allows direct handling of multiple packets in each data word. Furthermore, it supports configurable pipelining so optimal tradeoff between frequency (throughput) and resources can be selected. When fully pipelined, the proposed architecture achieves unprecedented throughput of over 2 Tbps (over 3000 Mpps).

II. Related Work

As already mentioned, the mathematical background of CRC has been extensively studied [3], [4], [5], [6] and it is not our focus. Rather, we want to use results of these papers to construct a new architecture for practical high-speed CRC computation for variable-length data packets.

Attempts to address the challenges of variable-length data packets are made for example in [7], [8], [9], [10]. All of them propose some kind of advanced architecture based on parallelization and pipelining. When implementing Ethernet CRC-32, they report throughputs sufficient for wire-speed traffic processing of up to 100 Gbps. But scaling for higher speeds is not properly addressed in any of these works and it would bring exponential growth in FPGA area and/or significant degradation of throughput on short packets (i.e. data rate limited by packet rate).

Interesting CRC architecture [11] uses similar approaches as the works above, but it also partially addresses the packet rate limitation on short packets. The architecture can process parts of two packets in a single clock cycle (data word). The reported throughput of 40 Gbps can be thus easily scaled up to 100 or even 200 Gbps. But the structural limit of only two packet parts pre-word means that further scaling would hit the same obstacles as mentioned above.

Fastest commercially available CRC is [12]. Authors claim that it achieves line-rate processing of up to 400 Gbps. But no throughput measurements nor resource requirements are provided to back up those claims. Furthermore, further details about architecture or its parameters are also lacking.

III. Architecture Design

This section describes the CRC architecture concept. It starts with the definition of data bus format which supports multiple packets per word. This is crucial for throughput scaling above 100 Gbps. Then, basic CRC computation blocks used in our concept are introduced. Finally, the architecture itself is presented in serial and parallel version.

A. Input Bus Format

The bus format as illustrated in Fig. 1. An example of packet placement is shown at the top with the format itself at the bottom. Each word is divided into *regions*. These define the maximum number of packets per word as at most one packet can start and one can end in each region. Regions are further

978-1-7281-0214-6/18/$31.00 ©2018 IEEE
DOI 10.1109/FPT.2018.00042

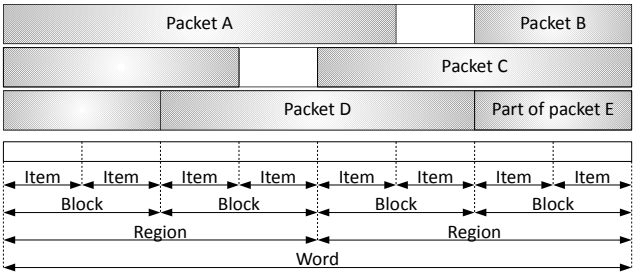

Fig. 1. Data bus format and packet placements example ($n = r = b = 2$).

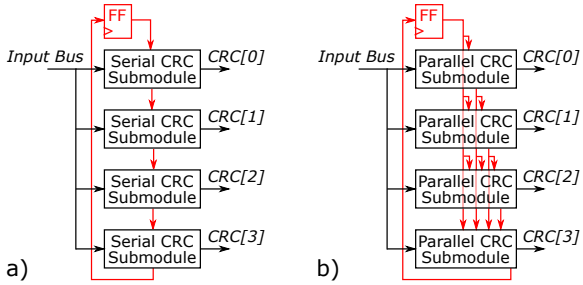

Fig. 2. CRC top-level architectures in serial and parallel version for $n = 4$.

separated into *blocks* of basic data *elements*. Each packet must start aligned to a block boundary, but can end on any data element (A and B end inside a block). Notice, that without the support of multiple packets per clock cycle, each packet would occupy a separate word (5 words required), but we are able to achieve more dense packing (only 3 words used).

To support this format, additional metadata must accompany each data word. For each region, we need to know: presence and position of a packet start (SOP and SOP_POS); presence and position of a packet end (EOP and EOP_POS). Furthermore, multiple versions of the bus with different parameters can be defined. We formally describe a specific version by four attributes: the number of regions in a word n; region size r as the number of blocks per region; block size b as the number of elements per block; element width e in bits. Using these attributes, we derive bus word width in bits like $dw = n \times r \times b \times e$. Now, we can also specify that illustration in Fig. 1 shows a bus version with $n = r = b = 2$.

B. Basic Blocks

We define 4 basic blocks: CRC table for fixed input width; accumulation logic to aggregate intermediate CRC values; input correction for packet start position; CRC finalization based on packet end position. Thanks to the division of CRC computations into blocks, the designed architecture is usable for any CRC polynomial. The change of the polynomial only requires re-generation of CRC equations inside these blocks but do not affect the top-level structure of the architecture.

As already mentioned, based on given polynomial and input width a specific **CRC table** can be generated [3]. It has a form of parallel XOR equations on input bits, one for each code (output) bit. The table basically converts the input data into an intermediary CRC value with no regard to packet borders.

CRC accumulator can be similarly generated for any polynomial. It aggregates two or more intermediary CRC values computed from separate parts of data by CRC tables. This enables handling of long packets in multiple small steps.

CRC start correction based on packet position is done by masking – i.e. the part of the word before packet start is filled with zeros. As CRC is based on XOR operations and zero is a neutral value for them ($0 \; xor \; a = a$), it is possible to show that prepending any number of leading zeros before data has no effect on the computed value [6]. Note, that also the initial value of CRC register must be shifted accordingly.

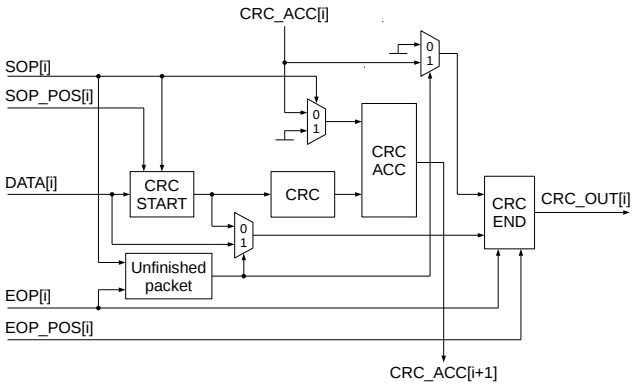

Fig. 3. The internal structure of one CRC submodule in the serial architecture.

Handling of **CRC end** is a bit more complicated. Masking cannot be directly applied, as appending zeros to data will change the CRC value. A workaround is to use a barrel-shifter to align the end of the packet with the end of the word. This way, the masking operation is converted from trailing into leading zeros and can be applied as in CRC start.

C. Top-Level Architecture

Fig. 2 shows the top-level structure of the proposed architecture. Processing of input bus word is divided between n submodules – one for each region. Serial (a) and parallel (b) architecture versions differ in the distribution of intermediate CRC values (red). In serial version, each submodule is passing its intermediate CRC result only to the next submodule and the last is passing its result to the first over a register. A potentially long critical path exists here that cannot be pipelined – from the top-level register, through CRC aggregation in all submodules, and back to the register. This is the reason for the parallel version, where each intermediate CRC is shared with each subsequent submodule. Consequently, more logic is required as each submodule performs more complicated CRC aggregation but the serial critical path is removed.

In Fig. 3 we can see the internal structure of one serial submodule. Base CRC table is used as a core computational block and corrections required for starting, ending or continuing packets is realized by separate blocks around it. They are controlled by metadata in the assigned region of the bus. The CRC start block masks input data before the packet start

and if there is no packet start present, the input data word is not altered. If a packet is continuing from the previous words, the output value of the CRC table is aggregated with an intermediate CRC value from the previous submodule. Otherwise (starting packet), the input CRC value is masked and no aggregation is performed. The output of accumulation is used as the intermediate CRC value on the input of the next submodule. Finally, CRC end block performs CRC calculation if the packet ends in the region assigned to this submodule.

In the parallel submodule, the output value of CRC accumulation is used only locally (for CRC end). So, the intermediate CRC results are not accumulated in steps through all submodules rather, each accumulation block independently aggregate values from all previous submodules (and CRC register). Other parts of the parallel implementation remain the same as in the serial version.

IV. Measured Results

We evaluate the proposed CRC architecture for high-speed Ethernet, where CRC is utilized as FCS field to ensure consistency of frames. Ethernet uses a 32 bit version of CRC with the CRC-32 division polynomial [1]. As discussed in the Related Work, published architectures can be used for Ethernet at speeds up to 200 Gbps and commercially available solutions promise throughputs of up to 400 Gbps. Their scaling towards higher speeds is limited by insufficient packet rates on the shortest packets for wider data buses. The proposed architecture addresses exactly this issue and therefore should be able to scale well even at higher throughputs.

Adjustment of the proposed architecture for Ethernet starts with the configuration of the bus format parameters from subsection III-A. Ethernet operates with bytes (octets) as the smallest data elements (element size $e = 8$). Lower (PCS/PMA) layers of Ethernet usually align frame starts with 8 B wide lanes (block size $b = 8$). Region size should correspond with the smallest allowed frames (64 B, so $r = 64/b = 8$). Smaller regions would needlessly allow more packets per word and larger regions would reduce effective bus saturation for the shortest packets. Using these attributes ($r = b = e = 8$) and expecting the shortest frames to be 64 B long, the bus format impose no more than $b - 1 = 7$ bytes of overhead per packet. Lower layers of Ethernet operate with larger overhead per packet (20 B of preamble and IFG). Therefore, the effective throughput of our bus is sufficient for wire-speed processing of Ethernet even in the worst case.

We measure results for different data bus widths ($dw = 512, 1024, 2048, 4096$) and various combinations of enabled pipeline registers in both versions of the proposed CRC architecture. In all cases, we use data bus parameters $r = b = e = 8$ (sufficient for wire-speed processing), only the value of n is changing with dw. The synthesis results form the state space of CRC implementations with different throughput, working frequency, latency, and resource usage. All values are obtained for the Xilinx UltraScale+ XCVU7P FPGA using the Xilinx Vivado 2017.3 design tool.

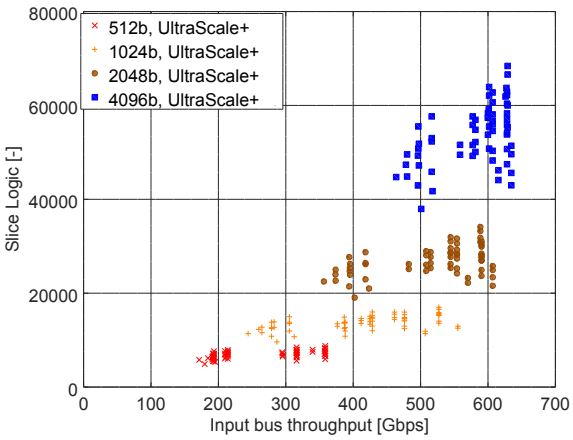

Fig. 4. Throughput and used resources of the serial version.

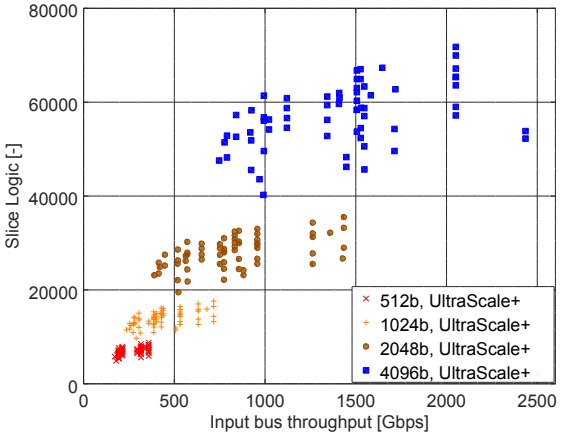

Fig. 5. Throughput and used resources of the parallel version.

Figures 4 and 5 show resource utilization and achieved throughput. Each point in the graphs represents one specific CRC implementation with a different combination of parameters (data width and pipeline registers enabling). The FPGA resources utilization linearly increases with the achieved throughput in the parallel implementation. In the best cases, we are able to reach effective throughputs of well over 2 Tbps (3000 Mpps). Unfortunately, in the serial implementation, the FPGA resources utilization increases considerably faster with throughput. This is because of notably lower frequencies due to the longer critical paths in CRC accumulation process.

Figures 6 and 7 bring the latency of different implementations into the picture. The latency depends on the number of enabled pipeline stages and achieved frequency. From the graphs, we can see that the latencies of the serial implementations are increasing notably with throughput (word width). On the other hand, the latencies of the parallel implementations remain approximately within the same bounds. This is again

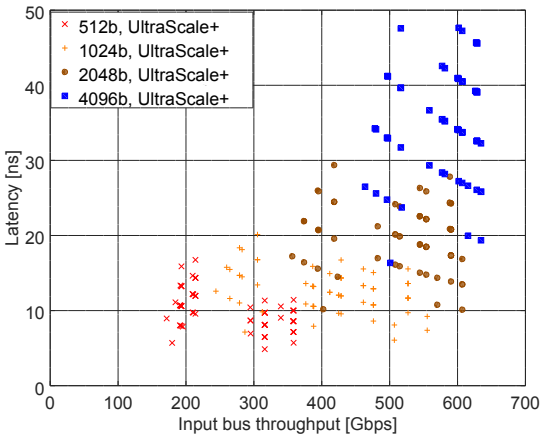

Fig. 6. Throughput and processing latency of the serial version.

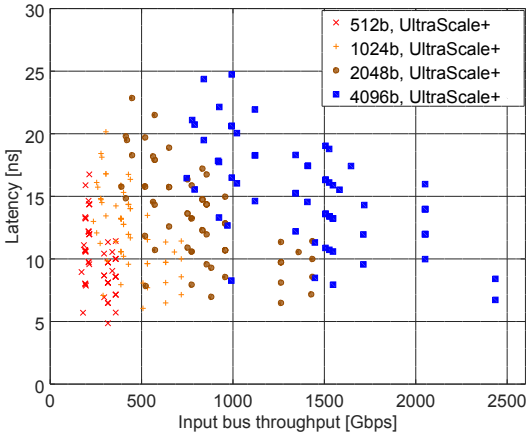

Fig. 7. Throughput and processing latency of the parallel version.

due to the higher frequency of parallel implementations.

Finally, Fig. 8 compares the best (Pareto optimal) serial and parallel implementations. We can more clearly see the difference between the serial (blue) and the parallel (red) implementations in achieved throughput (2 Tbps vs. 600 Gbps).

V. Conclusion

This paper introduces a novel FPGA architecture of general CRC computation that achieves very high processing throughputs. The proposed architecture can process multiple packets per clock cycle, can be easily adjusted for any CRC polynomial, and scale well even for very wide data buses. According to evaluation, the proposed concept enables to achieve unprecedented wire-speed throughput when computing CRC for Ethernet frames. At a cost of just a few percents of UltraScale+ FPGA resources, the achieved throughput can be over 2 Tbps (over 3000 Mpps). This is thanks to favorable frequency scaling of the designed parallel version of the

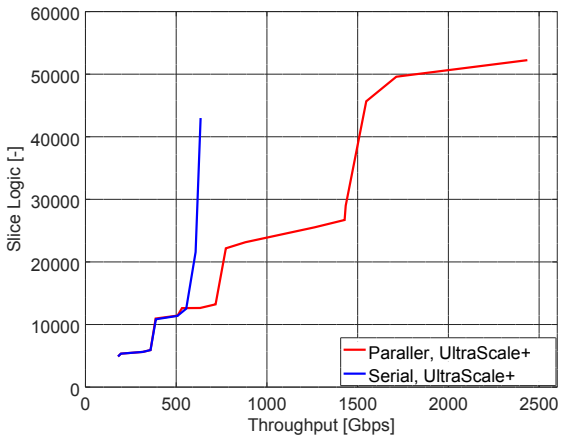

Fig. 8. Pareto optimal CRC implementations (throughput vs. resources).

proposed architecture. The proposed architecture has been verified in simulations and is currently tested on a real FPGA.

Acknowledgments

This research has been supported by the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 by the MEYS of the Czech Republic; the IT4Innovations excellence in science project IT4I XS – LQ1602; and by the Ministry of the Interior of the Czech Republic project VI20172020064.

References

[1] IEEE Computer Society, "Amendment 10: Media access control parameters, physical layers and management parameters for 200 Gb/s and 400 Gb/s operation," *IEEE Standard 802.3bs-2017*, pp. 1–372, 2017.

[2] Hybrid Memory Cube Consortium, *Hybrid Memory Cube Specification 2.1*, Altera Corporation, October 2015.

[3] M.-D. Shieh, M.-H. Sheu, C.-H. Chen, and H.-F. Lo, "A systematic approach for parallel CRC computations," *Journal of Information Science and Engineering*, vol. 17, no. 3, pp. 445–461, May 2001.

[4] T. B. Pei and C. Zukowski, "High-speed parallel CRC circuits in VLSI," *IEEE Transactions on Communications*, vol. 40, no. 4, 1992.

[5] A. Perez, "Byte-wise CRC calculations," *IEEE Micro*, vol. 3, no. 3, pp. 40–50, June 1983.

[6] C. Kennedy and A. Reyhani-Masoleh, "High-speed parallel CRC circuits," in *2008 42nd Asilomar Conference on Signals, Systems and Computers*, October 2008, pp. 1823–1829.

[7] T. Henriksson and D. Liu, "Implementation of fast CRC calculation," in *Proceedings of the Asia and South Pacific, Design Automatation Conference*, February 2003, pp. 563– 564.

[8] H. F. A. Hamed, F. Elmisery, and A. A. H. A. Elkader, "Implementation of low area and high data throughput CRC design on FPGA," *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)*, vol. 1, no. 9, 2012.

[9] M. Walma, "Pipelined cyclic redundancy check (CRC) calculation," in *16th International Conference on Computer Communications and Networks*, 2007, pp. 365–370.

[10] Bajarangbali and P. A. Anand, "Design of high speed CRC algorithm for ethernet on FPGA using reduced lookup table algorithm," in *IEEE Annual India Conference*, 2016, pp. 1–6.

[11] J. Mitra and T. Nayak, "Reconfigurable very high throughput low latency VLSI (FPGA) design architecture of CRC32," *Integration, the VLSI Journal*, vol. 56, pp. 1–14, 2017.

[12] Tamba Networks, *Datacenter Ethernet*, Tamba Networks, LLC, 2018, http://www.tambanetworks.com/products/datacenter-ethernet/.