Contribution ID: **168** Type: **Poster**

Modern C++17 Data Pre-Processing HLS Dataflow Template Library

Tuesday 20 September 2022 16:40 (20 minutes)

Developing and implementing algorithms for detector read-out using FPGAs is traditionally done by using a hardware description language like VHDL, Verilog, or System Verilog. In the proposed approach here, we discuss an alternative way using higher level languages like the Intel HLS Compiler. Intel HLS supports C++17 standard and is ideal to apply methods from Modern C++ to implement complex algorithms more easily. In this work we have developed a dataflow template library. This enables a shorter development time for increasingly complex algorithm requirements, which is also important for next generation experiments in the future.

Summary (500 words)

Data pre-processing with FPGAs is common practice for detector read-out in high energy physics. For the ALICE experiment in run three, we have the common read-out FPGA card based on an Intel Arria 10 FPGA called Common Readout Unit (CRU), which is mostly responsible to manage time frame building correctly and to reduce the huge amount of read-out data, e.g., with zero suppression and other algorithms. Traditionally, in this framework, the read-out firmware is developed using a Hardware Description Language (HDL). In the ALICE run three, we have a versatile common firmware design mostly designed in VHDL. The design is common for all detectors and is responsible for interface handling. Detector specific algorithms can be implemented as part of the firmware design as so-called user logic, which is mostly an entity written in VHDL, but could also be an IP written in HLS or other tools. That enables the option to implement an algorithm also in a high-level language like Intel HLS. For this reason, we have developed an Intel HLS Dataflow Template Library. The Intel HLS compiler is a modern C++ compiler which support many features up to the standard C++17. Thereby, programming at compile time is the most important feature, for which it turns out that it enables resource-saving implementations. However, the implementation of an algorithm written in C++ targeting an FPGA has its own peculiarities which come from the fact that the target hardware is completely different compared to a CPU. Intel HLS generates from a function labelled as component an IP, which can be then instantiated into an FPGA design, e.g., as part of the user-logic for the CRU. The corresponding RTL module has interfaces to allow the overall system to interact with the HLS component. For example, there are start and done signals. A function call issues the start signal for one clock cycle, and some latency, means some clock cycle later, the done signal indicates that the function returns with a valid result on its return output port. In addition, a component function is pipelined by default, which means you can call the function again one clock cycle later before the previous call returns. This behaviour is used to build a dataflow template library. With this library, we describe our algorithm as a deep pipelined dataflow graph. In this picture, a variable becomes a static data stream buffer representing an arc of the dataflow graph, the nodes are arithmetic operators, and other operators to orchestrate the streaming data through the pipelines. The results show that all designs always get an initiation result of one and the maximum throughput. Although the latency depends on how the graph is implemented and keeps in response to the programmer. The resource consumption keeps acceptable low due the fact that we use compile-time programming as shown. With this methodology an algorithm can easily implemented and tested especially for the growing design requirements for next generation experiments.

Primary authors: JANSON, Thomas (Goethe University Frankfurt (DE)); KEBSCHULL, Udo Wolfgang (Goethe University Frankfurt (DE))

Presenter: JANSON, Thomas (Goethe University Frankfurt (DE))

Session Classification: Tuesday posters session

Track Classification: Programmable Logic, Design Tools and Methods