Implementing complex **algorithms** for **detector read out** with high level language **HLS C++** on **FPGAs** as **pipelined Dataflow Graph**.

15th Pisa Meeting on Advanced Detectors
La Biodola, Isola d'Elba, May 22-28, 2022

Data Pre-Processing with High-Level-Synthesis and Dataflow Programming using HLS C++ Dataflow Template Library

Thomas Janson and Udo Kebschull
Infrastruktur und Rechnersysteme in der Informationsverarbeitung (IRI)
Goethe-Universität Frankfurt am Main

OVERVIEW

- An alternative method to develop algorithms targeting FPGAs using C++17 (Intel HLS)
- Dataflow Template Library to implement deep pipelined dataflow graphs on Hardware.
- Using C++17 compile-time features to keep hardware resources within an acceptable limit compared to VHDL implementation.

DESIGN REQUIREMENT

- Developing and testing your algorithm within a C++ framework.
- Easy use of arbitrary primitive data types (fixed-point, float, int, etc.).
- User defined data types.
- Outcome of calculation on FPGA the same as in emulation on CPU.
- Initiation interval always II=1 for maximum throughput.

RESULTS

Implementation (HLS)	ALM	REG	MLAB	RAM	DSP	II	Latency
moving_avg	47.5	95	2	1	0	1	8
moving_avg_hls	39.5	74	1	0	1	1	8
triangular_smooth_adc	73	166	1	0	0	1	6
triangular_smooth_float	459	809	14	0	6	1	29
peak_finder_adc	120	293	1	0	0	1	9
Implementation (VHDL)	**ALM**	**REG**	**MLAB**	**RAM**	**DSP**	**II**	**Latency**
moving_avg_rtl	21	37	0	0	1	1	5
peak_finder	65	129	0	0	0	1	11

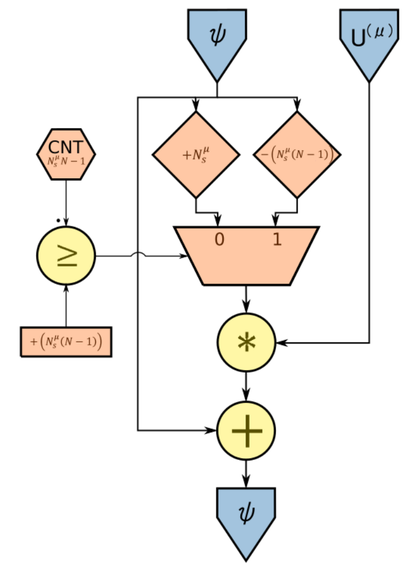
compiled with Intel HLS Pro 20.4, Arria10.

DISCUSSION

- The results show resource usage, initiation interval, and latency for simple components.
- Comparison with VHDL implementation
  - simple VHDL entities without optimization.
  - moving_average_hls vs. moving_average_rtl peak_finder_adc vs. peak_finder_rtl
  - ALMs are the limiting resources. We see that HLS needs more resources. With the two simple components, we need about two times more resources (ALMs) than VHDL counterparts.
  - Resource overhead mostly from component (interface) control logic (start, busy, done, and stall), we use the hls_avalon_streaming_component.

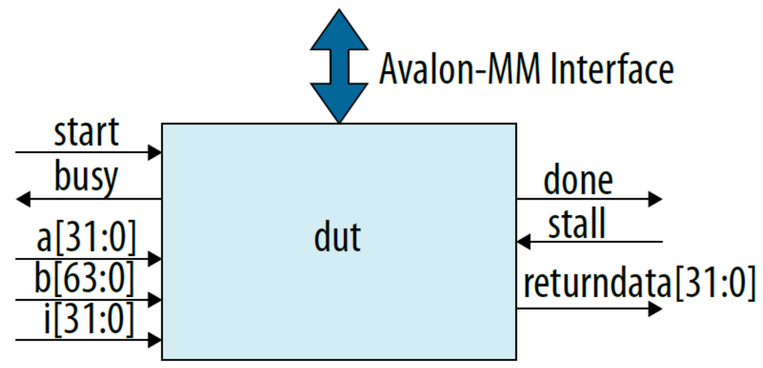
OUTLOOK and NEXT STEPS

- Tests with larger complex designs must be further provided to see how resource usage and usability scales.
- Implementation of graph balancing.
- Optimization of component interface.



ELEMENTS of an DATAFLOW GRAPH

- Variables are static stream buffers (HLSVar). This are the arcs of the dataflow graph and can hold more than one data item.
- Data items are tokens:
  - Token consists of the data value of its type and a valid bit.
- Assignment shifts Token into stream on left side of assignment only when Token on right side is valid.
- Reading from stream always from offset(0).
- Arithmetic compute nodes are circles.
- Offset Operator (diamond) picks data items out of stream buffers at given offset position.
- Each component invocation moves data items one position further through stream buffers.



Taken from Intel HLS Reference Manual

```
component Token<uint10> moving_avg(uint10 stream_in) {
    static HLSVar<uint10,1,-1> stream;
    stream = stream_in;
    const Token<uint10> three {3,true};
    Token<uint10> avg = (stream.offset(-1) + stream.offset(0)
                       + stream.offset(+1)) / three;
    return avg;
}
```

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Take a picture to
see source code for more information
https://github.com/docarat/hls_dataflow_template_library