Real-time data pre-processing for FPGA based **detector read out** with high-level language **HLS C++**

TWEPP 2022 Topical Workshop on Electronics for Particle Physics
Bergen, Norway, Sep 19-23, 2022

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

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

OVERVIEW

- Modern C++17 Template Library to describe an algorithm as dataflow graph
- The graph forms a deep pipeline on Hardware
- A deep pipelined graph is ideal for real-time data pre-processing. To guarantee this, we require an Initiation Interval (II) of 1
- 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

EXAMPLE SIMPLE PEAK FINDER ALGORITHM

- 10-bit ADC continuous input stream
- Detection of a gaussian noisy input pulse
- Pulse width is 10 samples
- Noise-level is 5-bit ADC values
- First stage is a triangular smooth filter
- Second stage computes the derivative
  - The result can be used to determine the pulse position
  - The zero crossing is the pule position as shown in the plot below (1st derivative)

```
117 component int11 peak_finder_adc(uint10 stream_in)
118 {
119     static HLSVar<uint14,3,-3> triangular_stream_buffer;
120     triangular_stream_buffer = stream_in;
121     static HLSVar<uint14,1,-1> smoothed_stream;
122     smoothed_stream = (triangular_stream_buffer.offset(-3) + Token<uint14>(2)*triangular_stream_buffer.offset(-2)
123             + Token<uint14>(3)*triangular_stream_buffer.offset(-1) + Token<uint14>(4)*triangular_stream_buffer.offset(0)
124             + Token<uint14>(3)*triangular_stream_buffer.offset(+1) + Token<uint14>(2)*triangular_stream_buffer.offset(+2)
125             + triangular_stream_buffer.offset(+3))/Token<uint14>(16);
126     static HLSVar<uint14> derivative;
127     derivative = ( smoothed_stream.offset(1) - smoothed_stream.offset(-1) ) / Token<uint14>(2);
128     int11 result = derivative.offset(0).value;
129     return result;
130 }
```

DISCUSSION

- The results show resource usage, initiation interval, and latency for simple components
- Comparison with VHDL implementation
  - ALMs are the limiting resources.
  - 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 default interface called hls_avalon_streaming_component

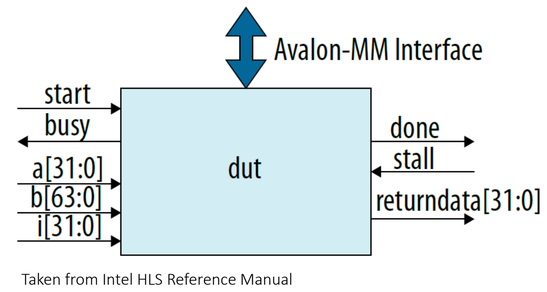
OUTLOOK and NEXT STEPS

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

RESULTS

Algorithm	Implementation	ALM	REG	MLAB	RAM	DSP	II	Latency	FMAX [MHz]
Moving average	HLS 32-bit float	73	153	4	0	3	1	14	471.7
Moving average	HLS 10-bit int	42	75	1	0	1	1	8	529.1
Moving average	VHDL 10-bit int	21	37	0	0	1	1	4	321.54
Triangular smooth	HLS 32-bit float	453	816	14	0	6	1	29	465.12
Triangular smooth	HLS 10-bit int	81	168	1	0	0	1	6	535.33
Triangular smooth	VHDL 10-bit int	54	96	0	0	0	1	4	356.76
Peak finder	HLS 32-bit float	536	1147	12	1	7	1	34	465.12
Peak finder	HLS 10-bit int	124	302	1	0	0	1	9	537.92
Peak finder	VHDL 10-bit int	62	119	0	0	0	1	6	349.04

compiled with Intel HLS Pro 20.4, Arria10.

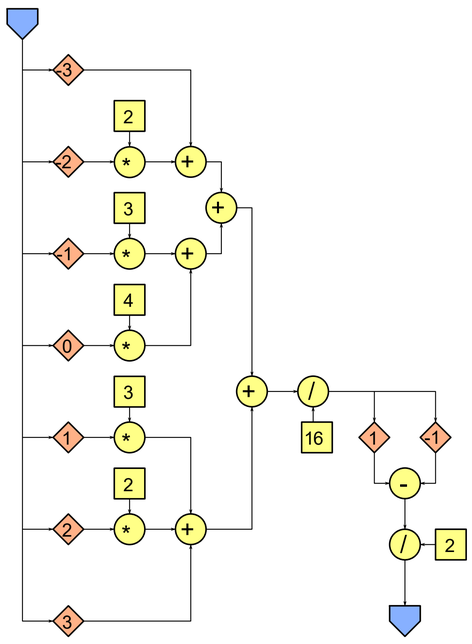


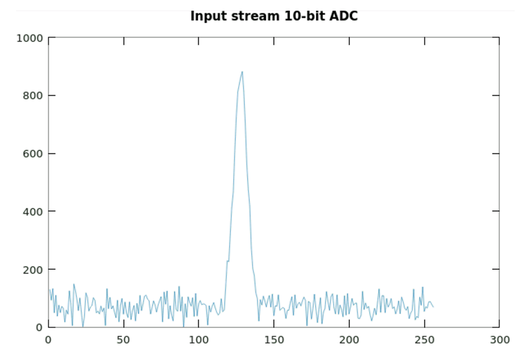
Taken from Intel HLS Reference Manual

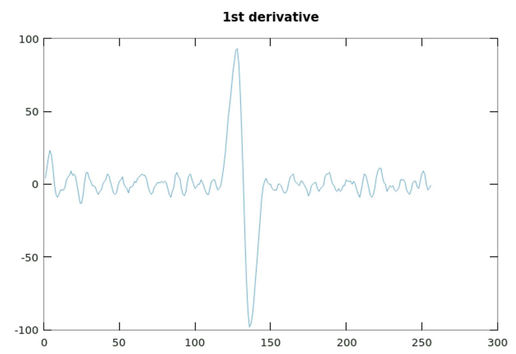
ELEMENTS of an DATAFLOW GRAPH

- Variables are static stream buffers (HLSVar). These 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.

DATAFLOW GRAPH SIMPLE PEAK FINDER







GOETHE UNIVERSITÄT FRANKFURT AM MAIN

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