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.

RESULTS

<table>
<thead>
<tr>
<th>Implementation (HLS)</th>
<th>ALM</th>
<th>REG</th>
<th>MLAB</th>
<th>RAM</th>
<th>DSP</th>
<th>II</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>moving_avg</td>
<td>47.5</td>
<td>95</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>moving_avg_hls</td>
<td>39.5</td>
<td>74</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>triangular_smooth_adc</td>
<td>73</td>
<td>166</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>triangular_smooth_float</td>
<td>459</td>
<td>809</td>
<td>14</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>peak_finder_adc</td>
<td>120</td>
<td>293</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation (VHDL)</th>
<th>ALM</th>
<th>REG</th>
<th>MLAB</th>
<th>RAM</th>
<th>DSP</th>
<th>II</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>moving_avg_rtl</td>
<td>21</td>
<td>37</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>peak_finder_rtl</td>
<td>65</td>
<td>129</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

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.

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

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