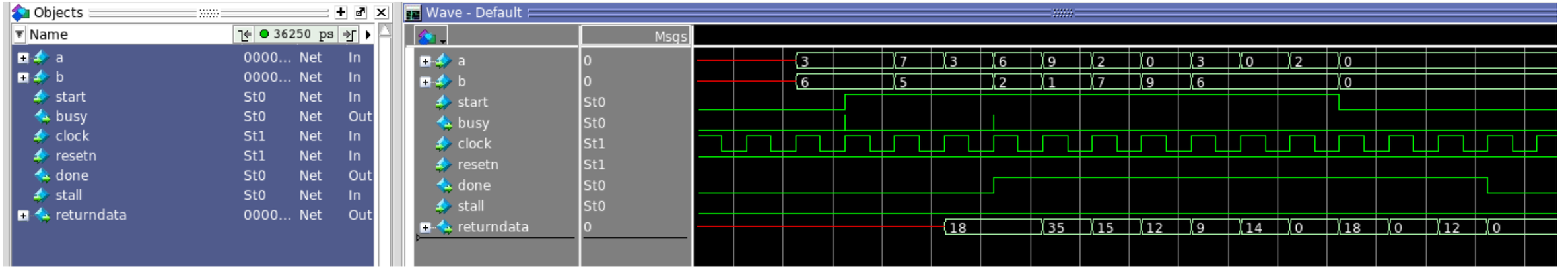
Modern C++17 Data Pre-Processing HLS Dataflow Template Library

Thomas Janson and Udo Kebschull

Component Interface




```
#include "HLS/hls.h"
#include "HLS/stdio.h"
#include "assert.h"

#define ELEMENTS 10

component int mymult(
                    int a,
                    int b)

{
        return a*b;
}
```

```
int main() {
        srand(0);
        int a[ELEMENTS];
        int b[ELEMENTS];
        int result[ELEMENTS];

        for (int i=0; i<ELEMENTS; ++i) {
                a[i]=rand()%10;
                b[i]=rand()%10;
                ihc_hls_enqueue(&(result[i]),&mymult,a[i],b[i]);
        }

        ihc_hls_component_run_all(mymult);

        for (int i=0; i<ELEMENTS; ++i) {
                printf("%d*%d=%d\n", a[i], b[i], result[i]);
                assert (result[i]==a[i]*b[i]);
        }
        return 0;
}
```

Result: Moving Average – Resource Consumption HLS vs. VHDL

```
begin

        oready <= '1';
        ovalid <= '1';

        data_stream: process(clock)
        begin
                if rising_edge(clock) then
                        if resetn = '0' then
                                summand01 <= (others => '0');
                                summand02 <= (others => '0');
                                summand03 <= (others => '0');
                        elsif( (ivalid = '1') and (iready = '1') ) then
                                summand01 <= u_ufixed(arg1);
                                summand02 <= summand01;
                                summand03 <= summand02;
                        end if;
                end if;
        end process data_stream;

        sum <= summand01 + summand02 + summand03;

        sum_reg: process(clock)
        begin
                if rising_edge(clock) then
                        sumreg <= sum;
                end if;
        end process sum_reg;

        res <= sumreg * denominator;

        out_reg: process(clock)
        begin
                if rising_edge(clock) then
                        oreg <= std_logic_vector(res(9 downto 0));
                end if;
        end process out_reg;

        result <= oreg;

end architecture rtl;
```

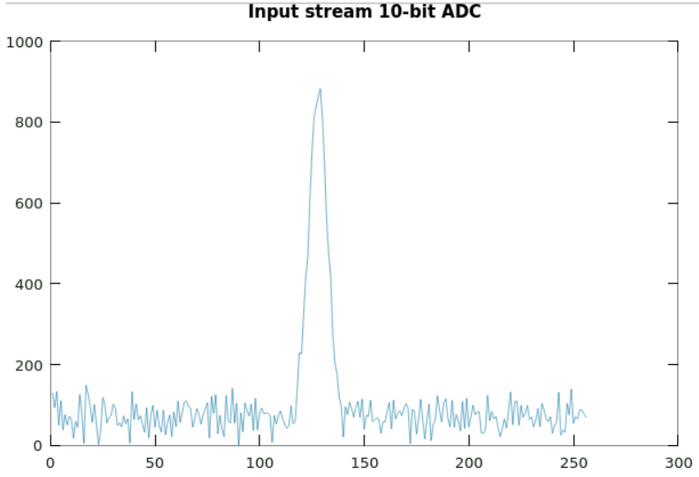
```
13 component Token<uint10> moving_avg(uint10 stream_in) {
14     static HLSVar<uint10,1,-1> stream;
15     stream = stream_in;
16     const Token<uint10> three {3,true};
17     Token<uint10> avg = (stream.offset(-1) + stream.offset(0) + stream.offset(+1)) / three;
18     return avg;
19 }
```

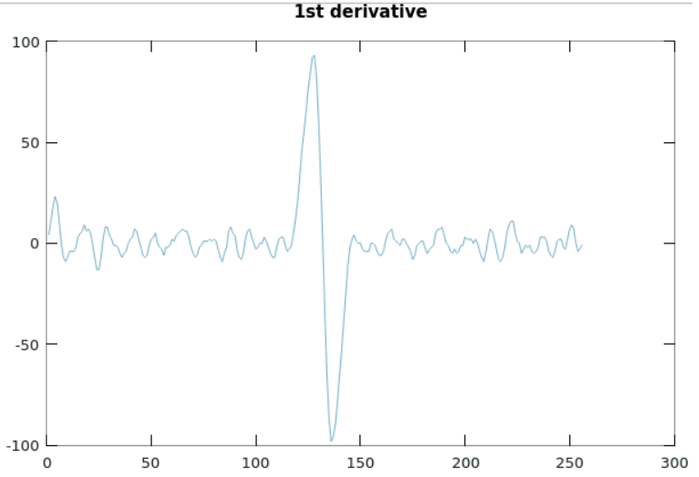
```
21 component uint10 moving_avg_hls (uint10 stream_in) {
22     static HLSVar<uint10, 1,-1> stream;
23     constexpr ac_fixed<33,10,false> rezipthree {1.0/3.0};
24     stream = stream_in;
25     ac_fixed<33,10,false> res = (stream.offset(-1) + stream.offset(0) + stream.offset(1)).value * rezipthree;
26     uint10 round_off_result = res.slc<10>(23);
27     uint10 round_up_result = round_off_result + 1;
28     uint10 result = res[22] ? round_up_result : round_off_result;
29     return result;
30 }
```

Impl	ALM	ALUT	REG	MLAB	RAM	DSP	FMax
moving_avg	48.5	-	94	2	1	0	645.16 MHz
moving_avg_hls	42	-	75	1	0	1	529.10 MHz
moving_avg VHDL	21		37	0	0	1	321.54 MHz

Result: Simple Peak Finder

```
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 }
```





- Gaussian noisy input pulse
  - width 10 samples
- 10-bit ADC input data
- Noise-Leve 5-bit ADC values

- Triangular smooth filter
- Central-Difference Method

Impl	peak_finder_adc
ALM	124
ALUT	-
REG	302
MLAB	1
RAM	0
DSP	0
FMax	537.92
II	1
Latency	9

Result Overview

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

- All components can be easily implemented with Initiation Interval II=1.
- The component is pipeline by default, and Modern C++ features helps to reduce resources.
- The Compiler optimize to II=1 and lowest possible Latency but depends how you write the program.