

# Design Throughput and Latency Optimization in Catapult HLS

Stuart Swan

Platform Architect

Mentor, A Siemens Business

4 December 2019 (revised for Catapult 2022.1)

This short tutorial presents a series of simple design examples using Catapult HLS to demonstrate how the use of Catapult synthesis directives affect throughput and latency of the generated design in RTL. After going through this tutorial you should have a good idea how Catapult transforms your designs when it performs operations like loop pipelining and unrolling, and you should understand how to model your design to achieve your specific throughput and latency targets.

## Description of Design

The design input is very simple so that it is easy to understand how Catapult transforms it. It is also easy to view the transformations in the Catapult GUI.

There are eight designs with identical functionality and differing throughput/latency characteristics. The designs are selected via the DESIGN\_1, DESIGN\_2, etc. compiler macros.

The basic design takes in a packet that has a single coefficient and a data array with 10 elements. All items are 32-bit integers:

```
36 struct packet : public Connections::message {
37     static const int data_len = 10;
38
39     ac_int<32,false> coeff;
40     ac_int<32,false> data[data_len];
```

The design multiplies each element in the array by the incoming coefficient and outputs the new packet:

```
59 #pragma hls_design top
60 class dut : public sc_module
61 {
62 public:
...
66     Connections::Out<packet> CCS_INIT_S1(out1);
67     Connections::In <packet> CCS_INIT_S1(in1);
...
77     void main() {
78         out1.Reset();
79         in1.Reset();
80         wait(); // WAIT
81
82 #ifdef DESIGN_1
83     while (1) {
84         packet p = in1.Pop();
85         for (int i=0; i < packet::data_len; i++) { p.data[i] *= p.coeff; }
86         out1.Push(p);
87     }
```

The design uses a 1 ns clock.

The testbench produces and consumes packets as quickly as possible. A given characteristic of the input packets is that 75% of the time the coefficients are either 0 or 1.

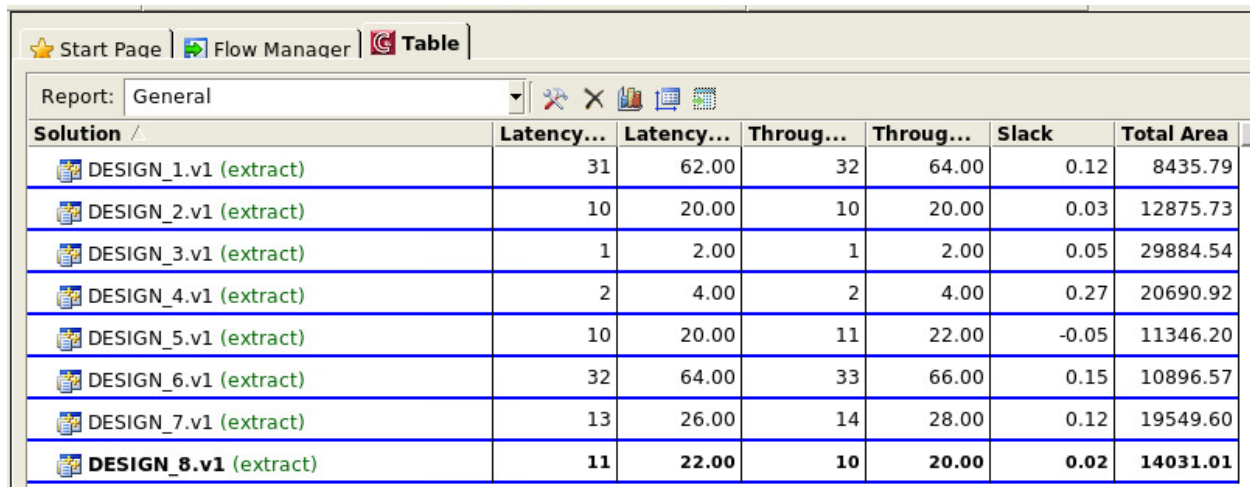
The design source code and script is in the HLSLIBS.org Matchlib kit at

[https://github.com/hlslibs/matchlib\\_toolkit/tree/main/examples/57\\_thruput\\_latency\\_control](https://github.com/hlslibs/matchlib_toolkit/tree/main/examples/57_thruput_latency_control)

If you have a Catapult version 2022.1 or new installation you can run all of the design scenarios in this example by doing:

```
dofile
$env(MGC_HOME)/shared/examples/matchlib/toolkit/examples/57_thruput_latency_control/go_hls.tcl
```

This will result in eight solutions being generated. The table results show the throughput and area for each:



Report: General						
Solution /	Latency...	Latency...	Throug...	Throug...	Slack	Total Area
DESIGN_1.v1 (extract)	31	62.00	32	64.00	0.12	8435.79
DESIGN_2.v1 (extract)	10	20.00	10	20.00	0.03	12875.73
DESIGN_3.v1 (extract)	1	2.00	1	2.00	0.05	29884.54
DESIGN_4.v1 (extract)	2	4.00	2	4.00	0.27	20690.92
DESIGN_5.v1 (extract)	10	20.00	11	22.00	-0.05	11346.20
DESIGN_6.v1 (extract)	32	64.00	33	66.00	0.15	10896.57
DESIGN_7.v1 (extract)	13	26.00	14	28.00	0.12	19549.60
<b>DESIGN_8.v1 (extract)</b>	<b>11</b>	<b>22.00</b>	<b>10</b>	<b>20.00</b>	<b>0.02</b>	<b>14031.01</b>

Let's now look at each specific Catapult synthesis run using this design example.

## DESIGN\_1

Summary: Basic design, no Catapult synthesis directives

Throughput (Catapult)	32 ns
Area (Catapult)	8435
Simulation Throughput	32 ns

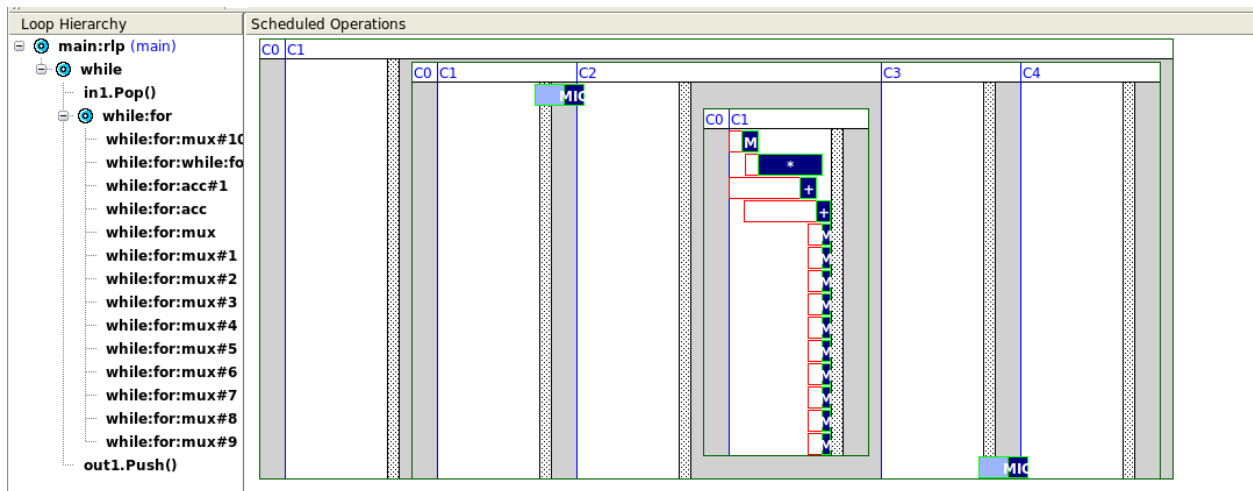
The area numbers are reported in the Catapult GUI.

(The most accurate throughput number is printed when the RTL simulation finishes:

```
# 1606 ns sc_main/top Throughput is: 32 ns
# (vsim-6574) SystemC simulation stopped by user.
# Simulation PASSED
# End time: 13:43:49 on Jan 24,2022, Elapsed time: 0:00:00
```

Since this design uses no Catapult synthesis directives, Catapult by default minimizes the area at the expense of latency. Thus, this design is the smallest and slowest of all of these examples.

In the Catapult Schedule view we can see the rolled loop and the single multiplier instance:



## DESIGN\_2

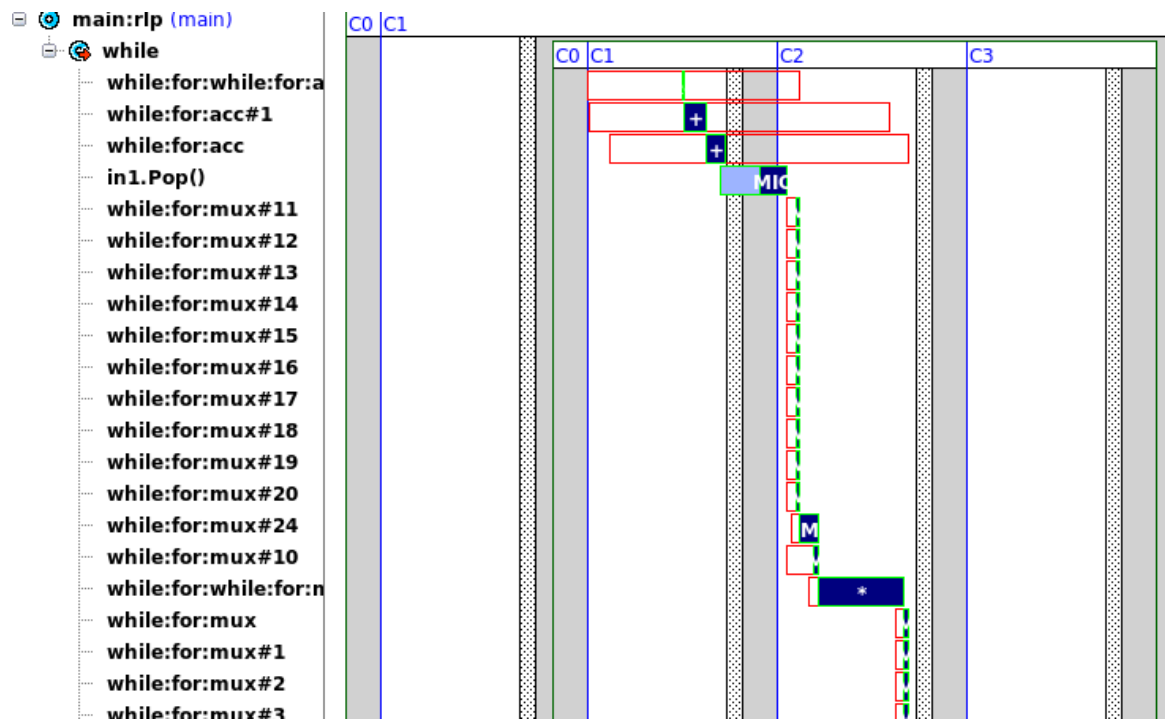
Summary: Same as previous design, but add pipeline directive

```
89 #elif DESIGN_2
90 #pragma hls_pipeline_init_interval 1
91 #pragma pipeline_stall_mode flush
92 while (1) {
93     packet p = in1.Pop();
94     for (int i=0; i < packet::data_len; i++) { p.data[i] *= p.coeff; }
95     out1.Push(p);
96 }
```

Throughput (Catapult)	10 ns
Area (Catapult)	12875
Simulation Throughput	10.020 ns

Area goes up and throughput improves somewhat due to the pipelining, but the inner loop is still rolled and stalls the outer pipelined loop.

In the Catapult Schedule view we can see the rolled loop and the single multiplier instance:



DESIGN\_3

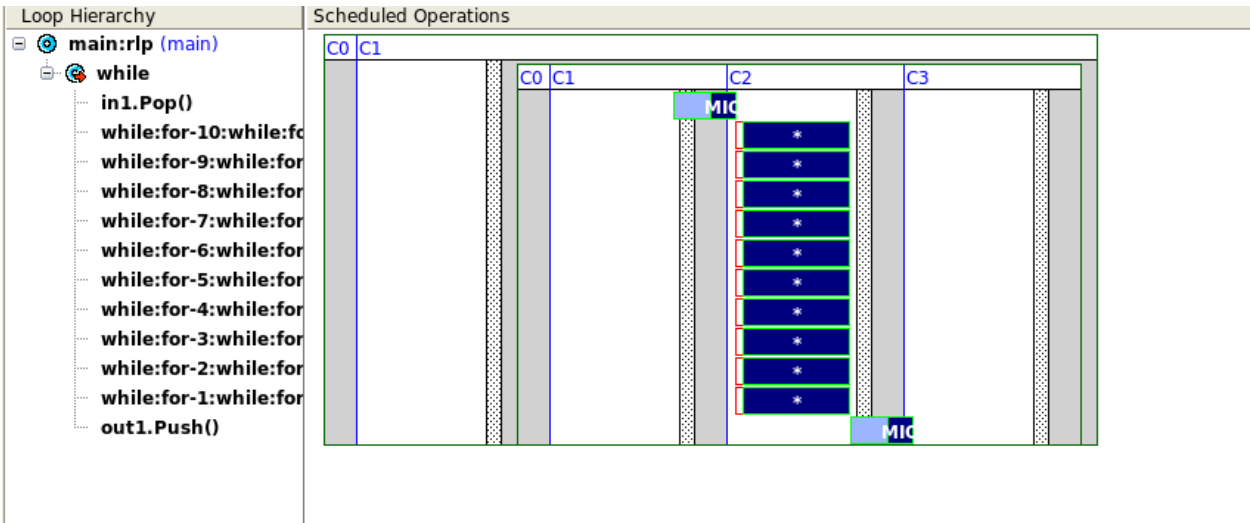
Summary: Same as previous design, but unroll the inner loop with the multiply operation

```
98  #elif DESIGN_3
99      #pragma hls_pipeline_init_interval 1
100      #pragma pipeline_stall_mode flush
101      while (1) {
102          packet p = in1.Pop();
103          #pragma unroll yes
104          for (int i=0; i < packet::data_len; i++) { p.data[i] *= p.coeff; }
105          out1.Push(p);
106      }
```

Throughput (Catapult)	1 ns
Area (Catapult)	29884
Simulation Throughput	1.020 ns

This is the largest area design of all the examples, with the best (optimal) throughput. This design represents the typical case for datapath dominated blocks where high throughput is the primary design criteria.

In the Catapult Schedule view we can see the unrolled loop and the 10 multiplier instances:



DESIGN\_4

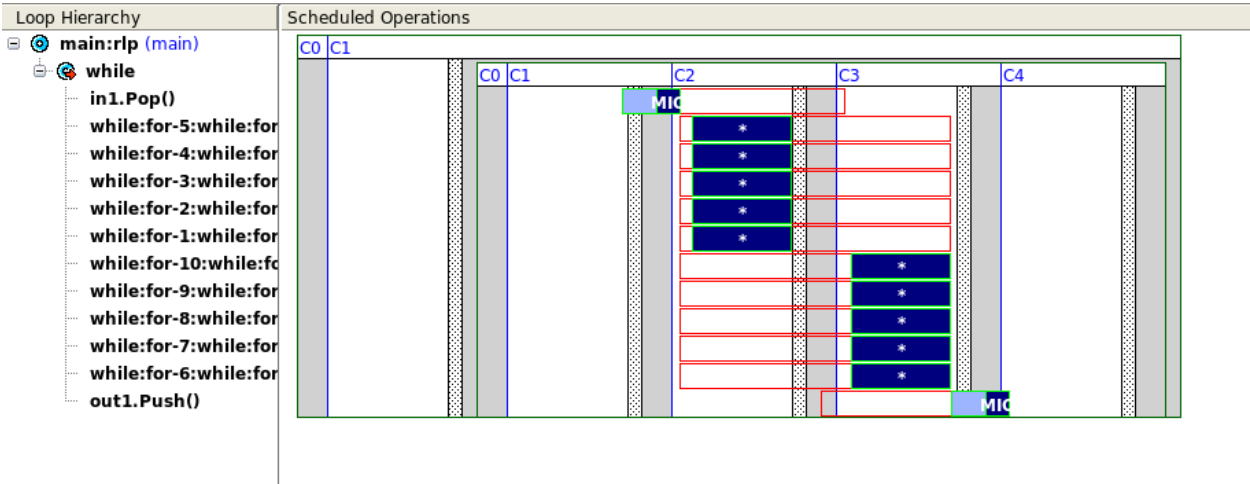
Summary: Same as previous design, but pipeline with Il=2

```
108 #elif DESIGN_4
109     #pragma hls_pipeline_init_interval 2
110     #pragma pipeline_stall_mode flush
111     while (1) {
112         packet p = in1.Pop();
113         #pragma unroll yes
114         for (int i=0; i < packet::data_len; i++) { p.data[i] *= p.coeff; }
115         out1.Push(p);
116     }
```

Throughput (Catapult)	2 ns
Area (Catapult)	20690
Simulation Throughput	2.020 ns

This design has half the throughput compared to the previous design, and more than half the area.

In the Catapult Schedule view we can see the 5 multiplier instances being shared in separate csteps within the pipelined outer loop:



## DESIGN\_5

Summary: Code optimization for "0" and "1" coefficients

```
118 #elif DESIGN_5
119     while (1) {
120         packet p = in1.Pop();
121
122         if (p.coeff == 0) {
123             #pragma unroll yes
124             for (int i=0; i < packet::data_len; i++) { p.data[i] = 0; }
125         } else if (p.coeff != 1) {
126             #pragma unroll yes
127             for (int i=0; i < packet::data_len; i++) { p.data[i] *= p.coeff; }
128         }
129         out1.Push(p);
130     }
```

Throughput (Catapult)	11 ns
Area (Catapult)	11346
Simulation Throughput	11 ns

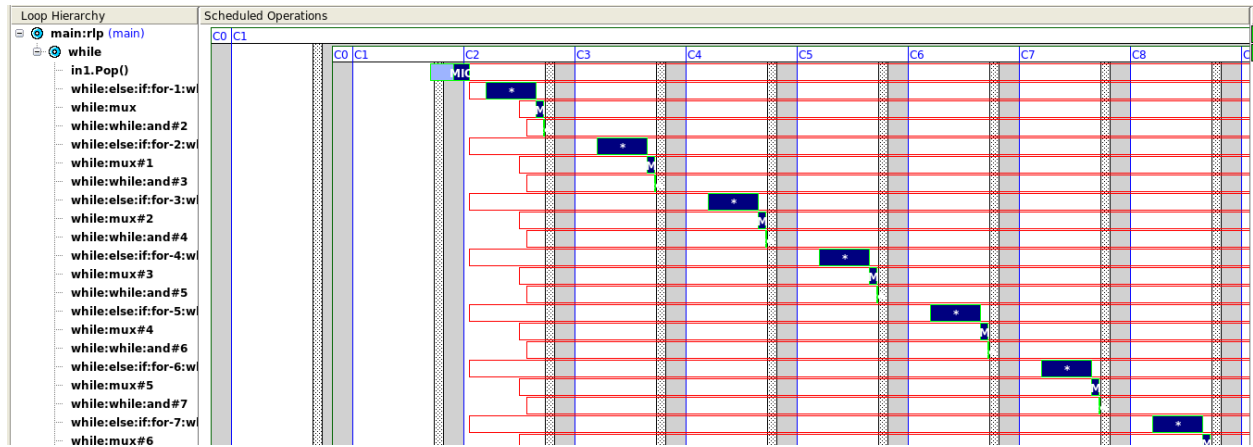
This design is an attempt to perform some latency optimizations based on the knowledge that the coefficients "0" and "1" occur frequently in input packets (they occur 75% of the time). If the coefficients are either 0 or 1, we can avoid doing the time-consuming multiplication operations and instead use hard coded logic to set the output data values.

Note that the two inner loops are unrolled, and the outer loop performing Push and Pop is not pipelined.

Even though the inner loops are unrolled, the design implementation has similar characteristics (including area and throughput) to DESIGN\_1, since Catapult by default minimizes area at the expense of latency. Catapult only generates 1 multiplier instance in this design, and shares that multiplier across 10 clock cycles to process a single packet.

The optimization for "0" and "1" coefficients is not reducing the latency for those packets because Catapult by default makes each branch of the if/else statement take matching latencies (the longest of any of the branches, in this case 10 cycles). In the next design we fix this.

If you inspect the schedule view in Catapult for this design you will see that the branches of the if/else have been merged (or "zipped") into the same csteps that include the multiply operations. **This is a key point: by default Catapult merges the branches of if/else statements into a merged set of csteps in the schedule. The latency of all of the branches will be equal to the latency of the longest branch.**



The actual throughput for this design and subsequent designs needs to be determined in RTL simulation since the latency/throughput now potentially depends on the input stimulus to the design.



DESIGN\_6

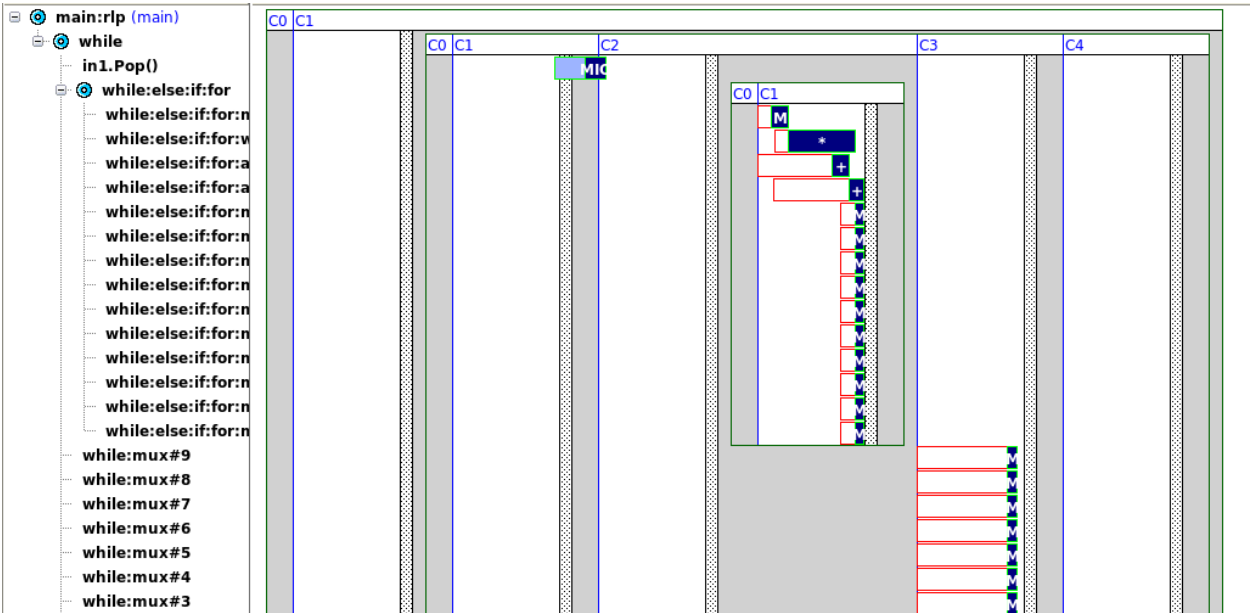
Summary: Same as previous design but leave inner multiply loop rolled.

```
132 #elif DESIGN_6
133
134     while (1) {
135         packet p = in1.Pop();
136         if (p.coeff == 0) {
137             #pragma unroll yes
138             for (int i=0; i < packet::data_len; i++) { p.data[i] = 0; }
139         } else if (p.coeff != 1) {
140             for (int i=0; i < packet::data_len; i++) { p.data[i] *= p.coeff; }
141         }
142         out1.Push(p);
143     }
```

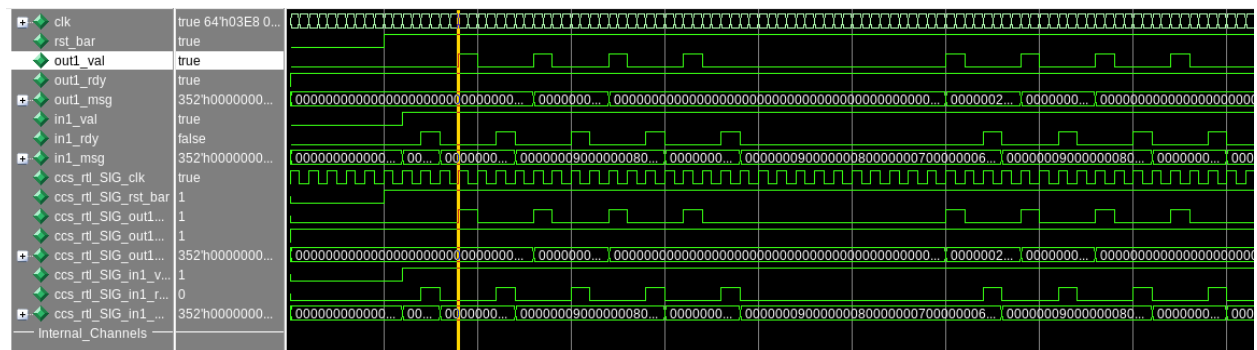
Throughput (Catapult)	33 ns
Area (Catapult)	10896
Simulation Throughput	10.020 ns

The rolled loop within one of the if/else branches causes Catapult not to merge (or zip) the branches together. Instead, Catapult generates separate csteps for each of the branches. This enables the optimization for the "0" and "1" coefficients to have the desired effect on latency.

If you inspect the schedule view in Catapult for this design you will see that the two branches of the if/else have been preserved as rolled loops, and their csteps have not been merged together.



If you inspect the RTL waveforms you will see that packets with 0 and 1 coefficients are produced quickly, with only a few clock cycles of latency, while other packets take around 10 cycles to produce:



## DESIGN\_7

Summary: Same as DESIGN\_5 but add LATENCY\_CONTROL directives

```
145 #elif DESIGN_7
146     while (1) {
147         packet p = in1.Pop();
148
149         if (p.coeff == 0) {
150             LATENCY_CONTROL_BEGIN()
151             #pragma unroll yes
152             for (int i=0; i < packet::data_len; i++) { p.data[i] = 0; }
153             LATENCY_CONTROL_END()
154         } else if (p.coeff != 1) {
155             LATENCY_CONTROL_BEGIN()
156             #pragma unroll yes
157             for (int i=0; i < packet::data_len; i++) { p.data[i] *= p.coeff; }
158             LATENCY_CONTROL_END()
159         }
160         out1.Push(p);
161     }
```

The LATENCY\_CONTROL... macros are defined as:

```
26 #define LATENCY_CONTROL_BEGIN() \
27     _Pragma("hls_preserve_loop yes") \
28     while(true) {
29
30
31 #define LATENCY_CONTROL_END() \
32     break; \
33 }
```

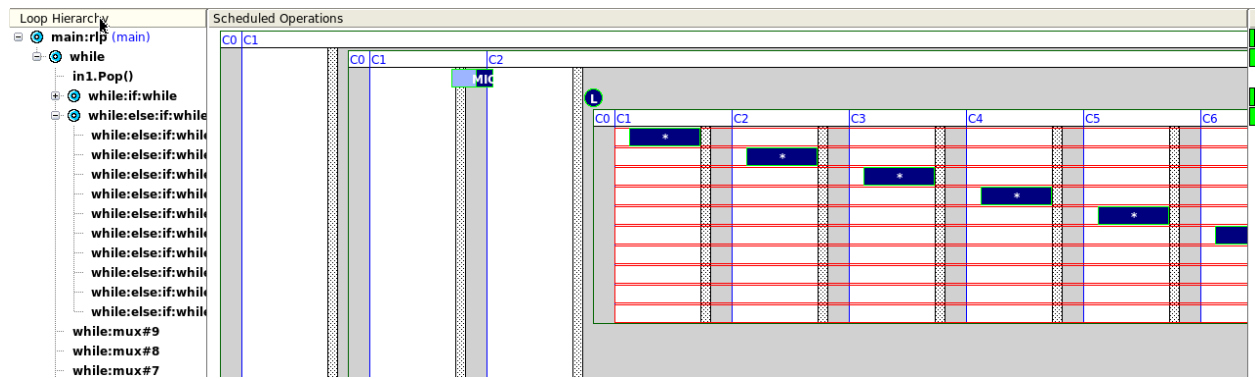
Throughput (Catapult)	14 ns
Area (Catapult)	19549
Simulation Throughput	5.4 ns

Throughput/area is similar to the previous design. You may be wondering what the LATENCY\_CONTROL directives are useful for. The LATENCY\_CONTROL directives are useful if you had a design similar to the previous design but which did not contain a rolled loop within the if/else branch, and you wanted Catapult to keep the latencies of the if/else branches separated.

Note: The LATENCY\_CONTROL directives tell Catapult to finish each branch of the if/else as soon as possible, and this now makes the "0" and "1" coefficient optimization work.

(Keep in mind that since the two inner loops are unrolled they "disappear" in Catapult).

If you inspect the schedule view in Catapult for this design you will see that the 2 branches of the if/else have been preserved as rolled loops (which are embedded within the LATENCY\_CONTROL directives), and their csteps have not been merged together.



## DESIGN\_8

Summary: Pipeline outer Pop/Push loop, keep multiply loop rolled

```

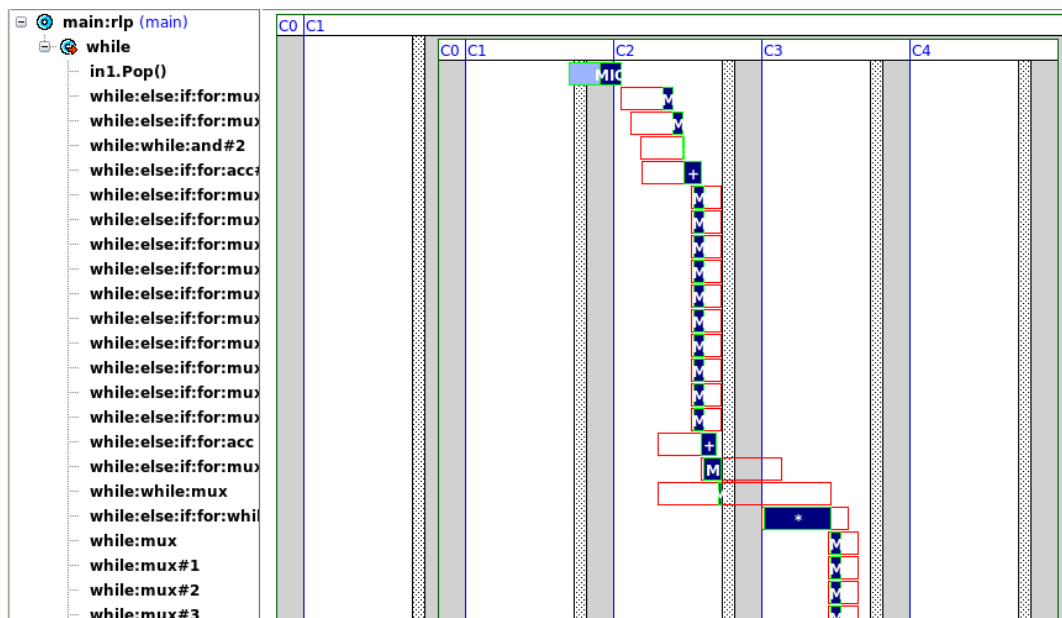
163 #elif DESIGN_8
164     #pragma hls_pipeline_init_interval 1
165     #pragma pipeline_stall_mode flush
166     while (1) {
167         packet p = in1.Pop();
168
169         if (p.coeff == 0) {
170             #pragma unroll yes
171             for (int i=0; i < packet::data_len; i++) { p.data[i] = 0; }
172         } else if (p.coeff != 1) {
173             for (int i=0; i < packet::data_len; i++) { p.data[i] *= p.coeff; }
174         }
175         out1.Push(p);
176     }

```

Throughput (Catapult)	10 ns
Area (Catapult)	14031
Simulation Throughput	3.2 ns

In this design the optimization for latency for "0" and "1" coefficients works because the loop with the multiplier is still rolled, and in addition we are now pipelining the outer loop that performs the Push and Pop operations. The rolled inner loop effectively stalls outer pipelined loop when it executes. This results in a design with a good balance of throughput and latency versus area for our particular requirements.

The schedule view shows the pipelined loop and the inner rolled loop:



The RTL waveforms show that packets with 0 and 1 coefficients are emitted quickly, without clock cycle gaps between them (in comparison to DESIGN\_6). This is because the outer loop is pipelined.

