Design Throughput and Latency Optimization in Catapult HLS

Stuart Swan

Platform Architect

Mentor, A Siemens Business

4 December 2019

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

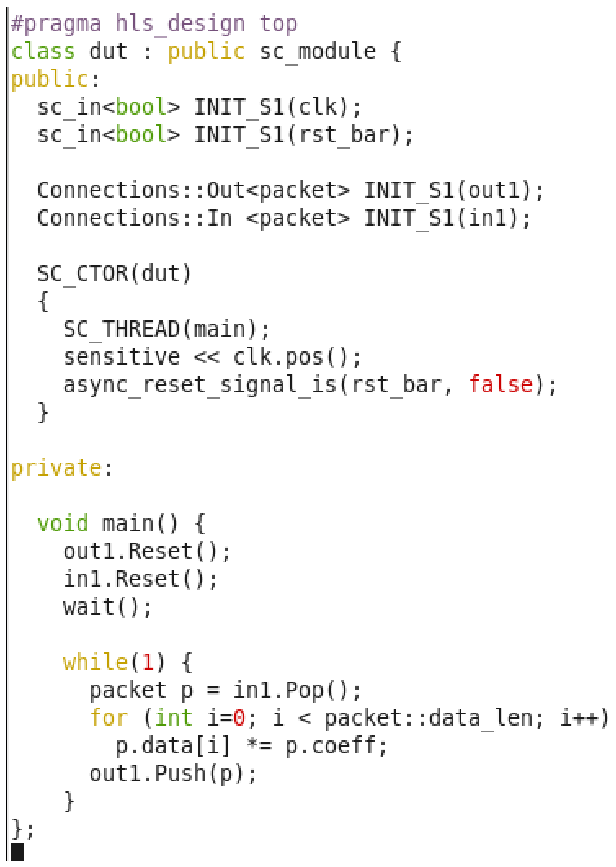
The design input is very simple so that it is easy to understand how Catapult transforms it. It is also easy to view the transformations in the Catapult GUI.

There are eight designs with identical functionality and differing throughput/latency characteristics. The designs are selected via the DESIGN_1, DESIGN_2, etc., #define directives.

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.

```
struct packet
{
  static const int data_len = 10;

  NVUINTW(32) coeff;
  NVUINTW(32) data[data_len];
};
```

The design multiplies each element in the array by the incoming coefficient and outputs the new packet. The design uses a 1 ns clock.



```
#pragma hls_design top
class dut : public sc_module {
public:
  sc_in<bool> INIT_S1(clk);
  sc_in<bool> INIT_S1(rst_bar);

  Connections::Out<packet> INIT_S1(out1);
  Connections::In <packet> INIT_S1(in1);

  SC_CTOR(dut)
  {
    SC_THREAD(main);
    sensitive << clk.pos();
    async_reset_signal_is(rst_bar, false);
  }

private:

  void main() {
    out1.Reset();
    in1.Reset();
    wait();

    while(1) {
      packet p = in1.Pop();
      for (int i=0; i < packet::data_len; i++)
        p.data[i] *= p.coeff;
      out1.Push(p);
    }
};
```

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 scripts are located in the Mentor Matchlib kit at kit/mentor/examples/57_thruput_latency_control .

Let’s now look at a series of specific Catapult synthesis runs using this design example.

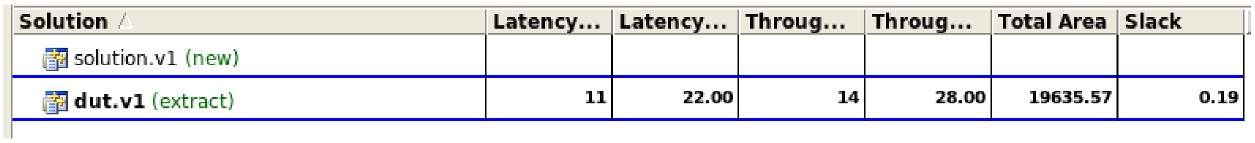
DESIGN_1

Summary: Basic design, no Catapult synthesis directives

Throughput: 14 ns

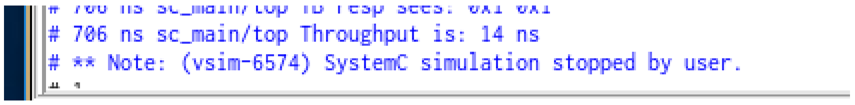
Area: 19635

The area numbers are reported in the Catapult GUI:



Solution	Latency...	Latency...	Throug...	Throug...	Total Area	Slack
solution.v1 (new)						
dut.v1 (extract)	11	22.00	14	28.00	19635.57	0.19

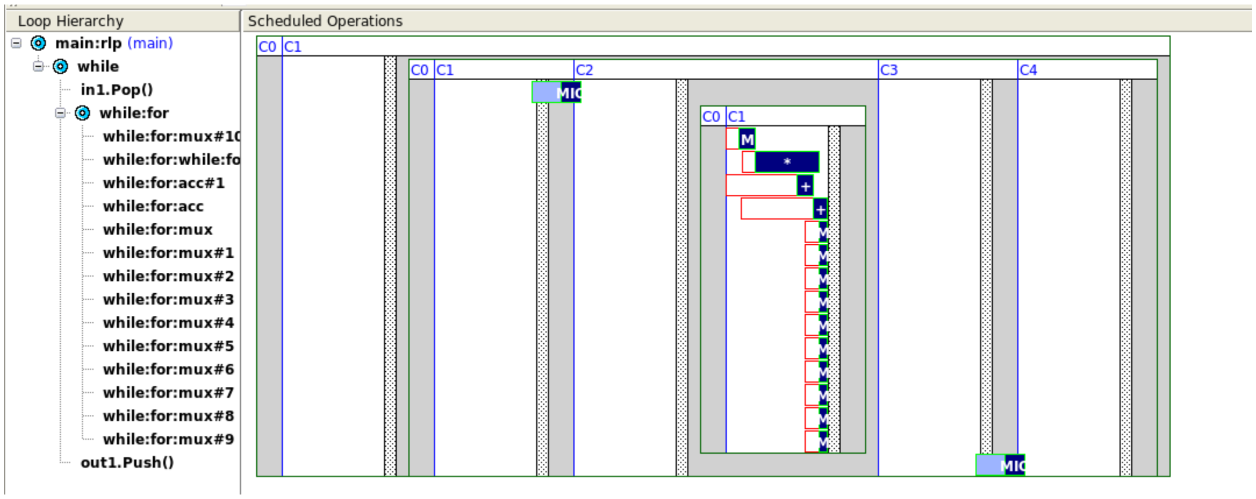
The most accurate throughput number is printed when the RTL simulation finishes:



```
# 706 ns sc_main/top TB resp sees: 0x1 0x1
# 706 ns sc_main/top Throughput is: 14 ns
# ** Note: (vsim-6574) SystemC simulation stopped by user.
```

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.

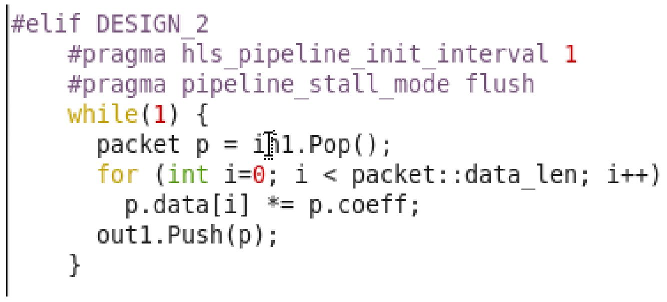
In the Catapult Schedule view we can see the rolled loop and the single multiplier instance:



```
Loop Hierarchy
main:rlp (main)
  while
    in1.Pop()
    while:for
      while:for:mux#10
      while:for:while:fo
      while:for:acc#1
      while:for:acc
      while:for:mux
      while:for:mux#1
      while:for:mux#2
      while:for:mux#3
      while:for:mux#4
      while:for:mux#5
      while:for:mux#6
      while:for:mux#7
      while:for:mux#8
      while:for:mux#9
    out1.Push()
```

DESIGN_2

Summary: Same as previous design, but add pipeline directive



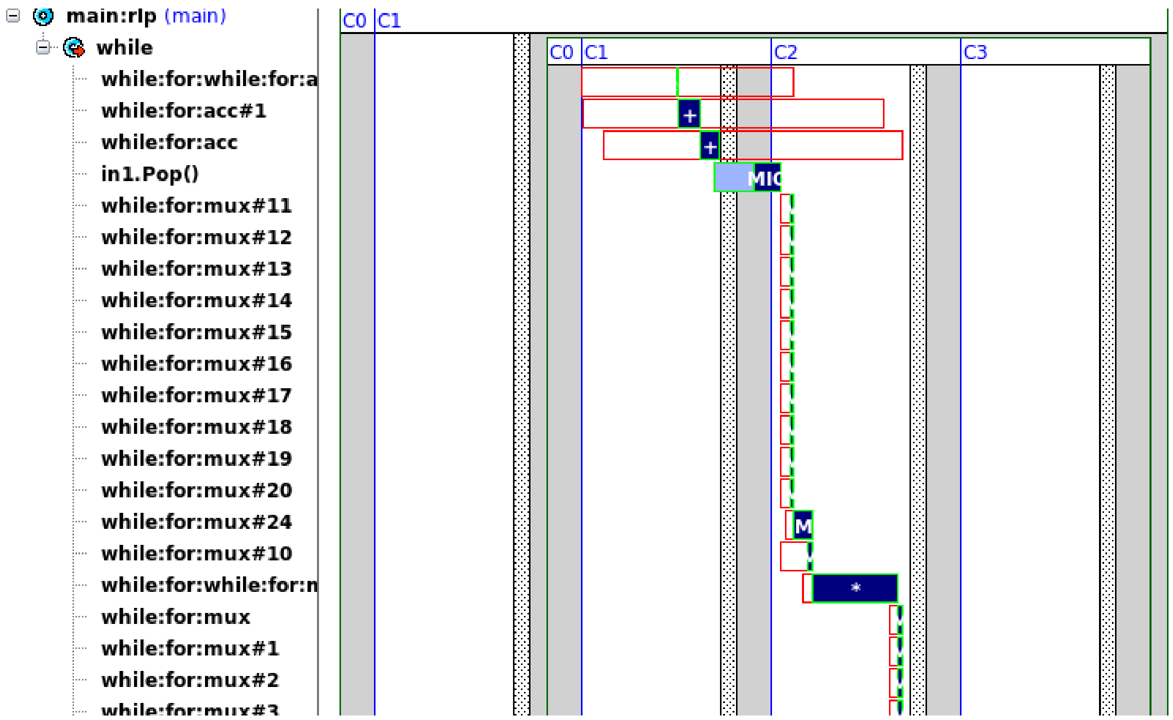
```
#elif DESIGN_2
    #pragma hls_pipeline_init_interval 1
    #pragma pipeline_stall_mode flush
    while(1) {
      packet p = in1.Pop();
      for (int i=0; i < packet::data_len; i++)
        p.data[i] *= p.coeff;
      out1.Push(p);
    }
```

Throughput: 10 ns

Area: 28192

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

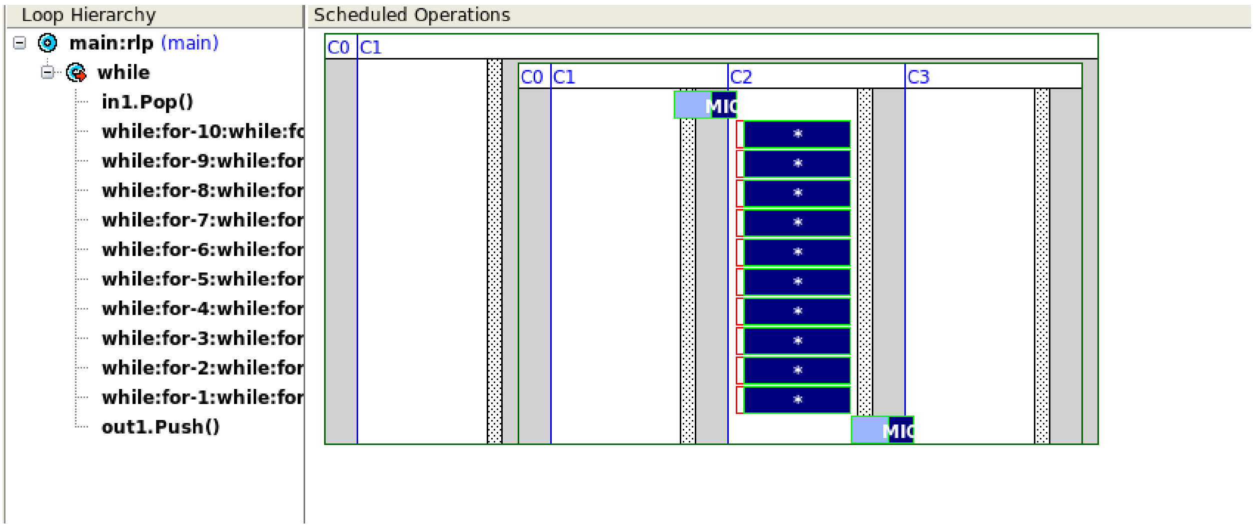
```
#elif DESIGN_3
    #pragma hls_pipeline_init_interval 1
    #pragma pipeline_stall_mode flush
    while(1) {
      packet p = in1.Pop();
      #pragma unroll yes
      for (int i=0; i < packet::data_len; i++)
        p.data[i] *= p.coeff;
      out1.Push(p);
    }
```

Throughput: 1 ns

Area: 65388

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 II=2

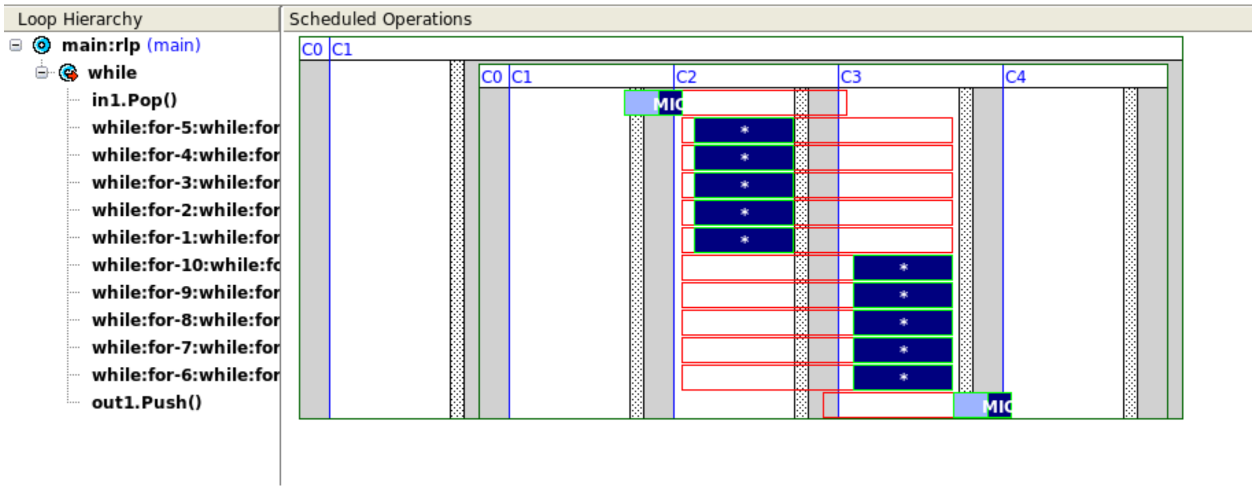
```
#elif DESIGN_4
    #pragma hls_pipeline_init_interval 2
    #pragma pipeline_stall_mode flush
    while(1) {
      packet p = in1.Pop();
      #pragma unroll yes
      for (int i=0; i < packet::data_len; i++)
        p.data[i] *= p.coeff;
      out1.Push(p);
    }
```

Throughput: 2 ns

Area: 45525

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

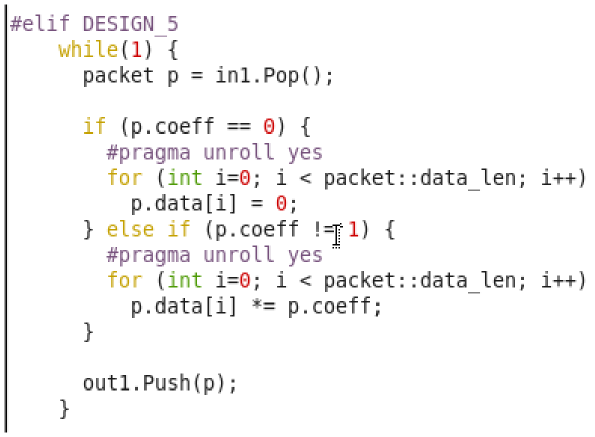
In the Catapult Schedule view we can see the 5 multiplier instances being shared in separate csteps within the pipelined outer loop:



```
Loop Hierarchy
main:rlp (main)
  while
    in1.Pop()
    while:for-5:while:for
    while:for-4:while:for
    while:for-3:while:for
    while:for-2:while:for
    while:for-1:while:for
    while:for-10:while:fo
    while:for-9:while:for
    while:for-8:while:for
    while:for-7:while:for
    while:for-6:while:for
    out1.Push()
```

DESIGN_5

Summary: Code optimization for “0” and “1” coefficients



```
#elif DESIGN_5
    while(1) {
      packet p = in1.Pop();

      if (p.coeff == 0) {
        #pragma unroll yes
        for (int i=0; i < packet::data_len; i++)
          p.data[i] = 0;
      } else if (p.coeff != 1) {
        #pragma unroll yes
        for (int i=0; i < packet::data_len; i++)
          p.data[i] *= p.coeff;
      }

      out1.Push(p);
    }
```

Throughput: 12 ns

Area: 23282

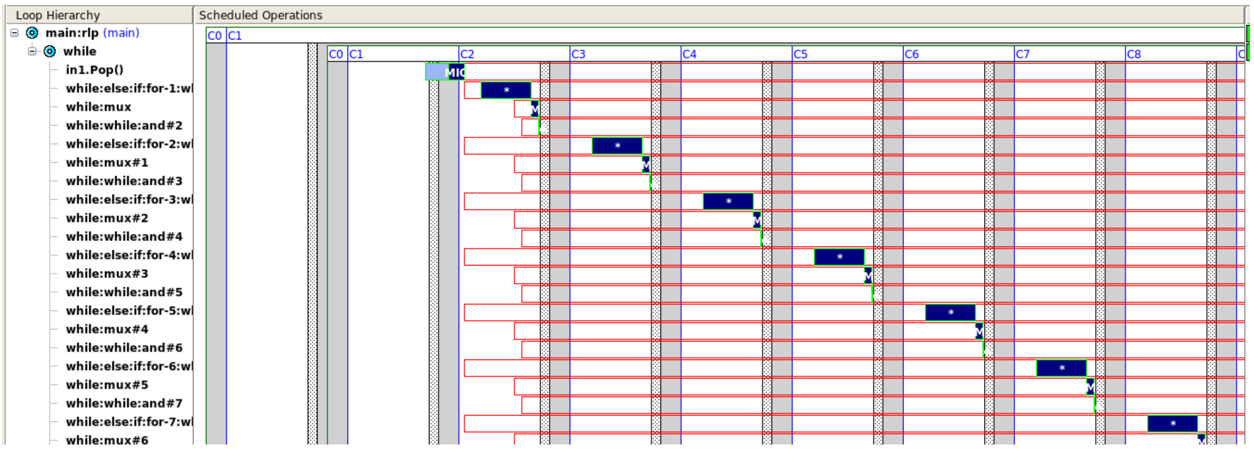
This design is an attempt to perform some latency optimizations based on the knowledge that the coefficients "0" and "1" occur frequently in input packets (they occur 75% of the time). If the coefficients are either 0 or 1, we can avoid doing the time-consuming multiplication operations and instead use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

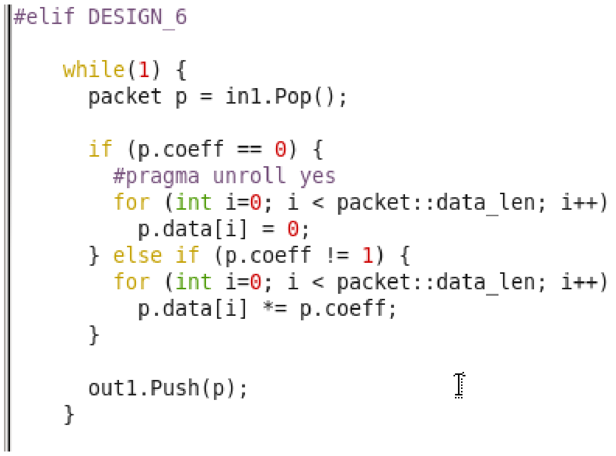
If you inspect the schedule view in Catapult for this design you will see that the branches of the if/else have been merged (or "zipped") into the same csteps that include the multiply operations. This is a key point: by default Catapult merges the branches of if/else statements into a merged set of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



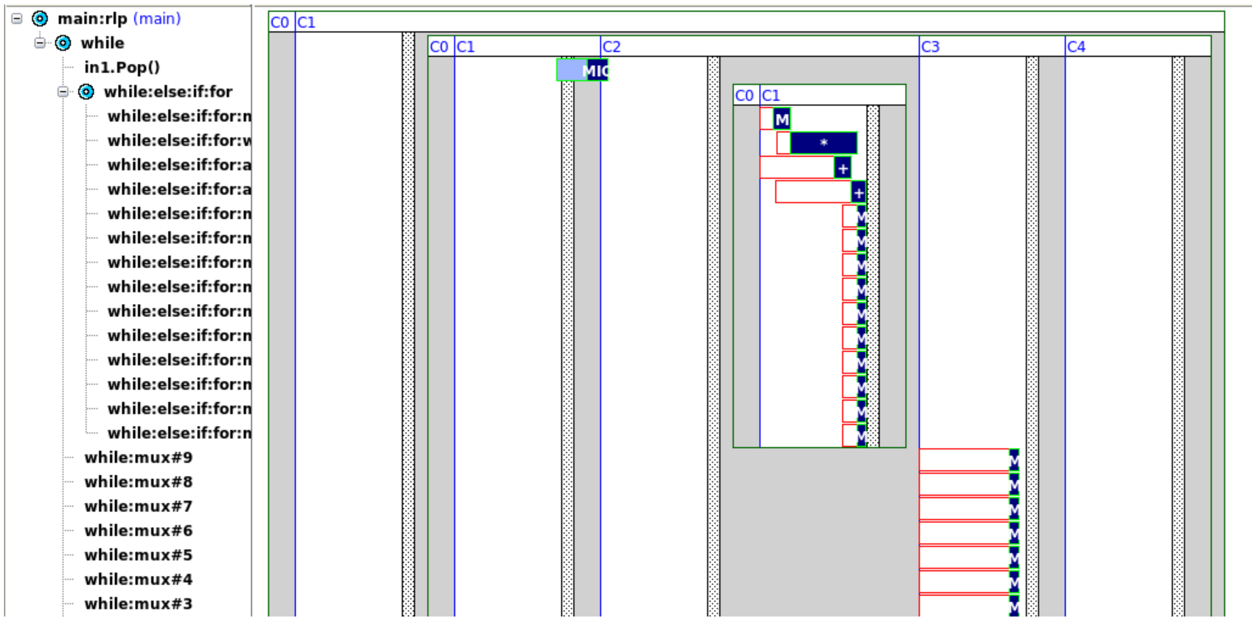
```
#elif DESIGN_6

    while(1) {
      packet p = in1.Pop();

      if (p.coeff == 0) {
        #pragma unroll yes
        for (int i=0; i < packet::data_len; i++)
          p.data[i] = 0;
      } else if (p.coeff != 1) {
        for (int i=0; i < packet::data_len; i++)
          p.data[i] *= p.coeff;
      }

      out1.Push(p);
    }
```

Throughput: 6.4 ns

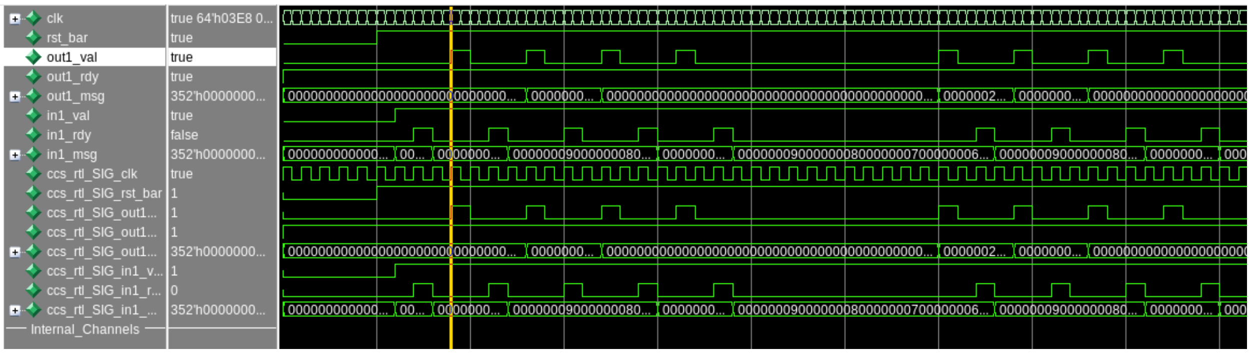
Area: 27657

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.



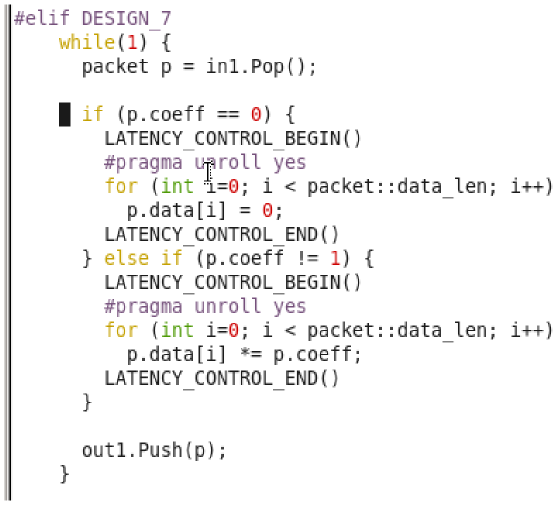
If you inspect the RTL waveforms you will see that packets with 0 and 1 coefficients are produced quickly, with only a few clock cycles of latency, while other packets take around 10 cycles to produce:



clk	true 64'h03E8 0...
rst_bar	true
out1_val	true
out1_rdy	true
out1_msg	352'h0000000...
in1_val	true
in1_rdy	false
in1_msg	352'h0000000...
ccs_rtl_SIG_clk	true
ccs_rtl_SIG_rst_bar	1
ccs_rtl_SIG_out1...	1
ccs_rtl_SIG_out1...	1
ccs_rtl_SIG_out1...	352'h0000000...
ccs_rtl_SIG_in1_v...	1
ccs_rtl_SIG_in1_r...	0
ccs_rtl_SIG_in1_...	352'h0000000...
Internal_Channels	

DESIGN_7

Summary: Same as DESIGN_5 but add LATENCY_CONTROL directives



```
#elif DESIGN_7
    while(1) {
      packet p = in1.Pop();

      if (p.coeff == 0) {
        LATENCY_CONTROL_BEGIN()
        #pragma unroll yes
        for (int i=0; i < packet::data_len; i++)
          p.data[i] = 0;
        LATENCY_CONTROL_END()
      } else if (p.coeff != 1) {
        LATENCY_CONTROL_BEGIN()
        #pragma unroll yes
        for (int i=0; i < packet::data_len; i++)
          p.data[i] *= p.coeff;
        LATENCY_CONTROL_END()
      }

      out1.Push(p);
    }
```

Throughput: 6.4 ns

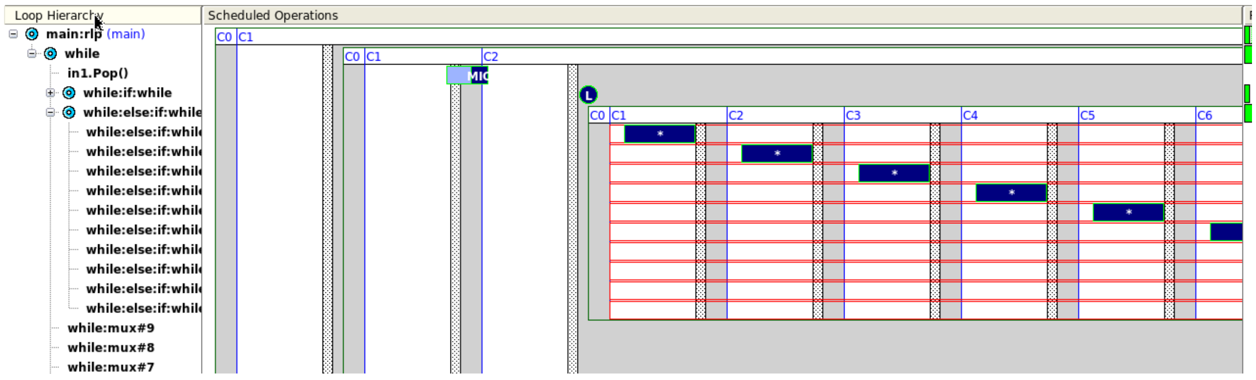
Area: 25638

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.

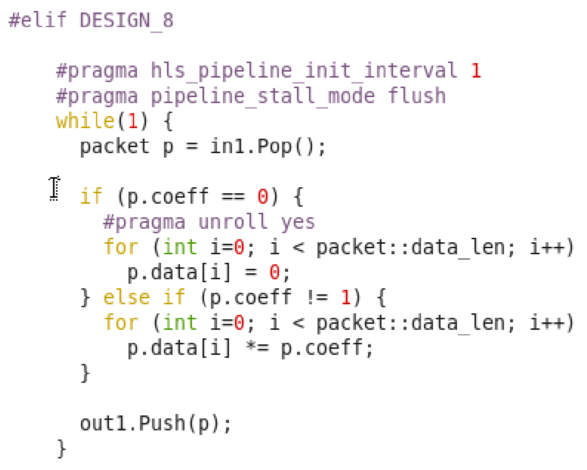
(Keep in mind that since the two inner loops are unrolled they "disappear" in Catapult).

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



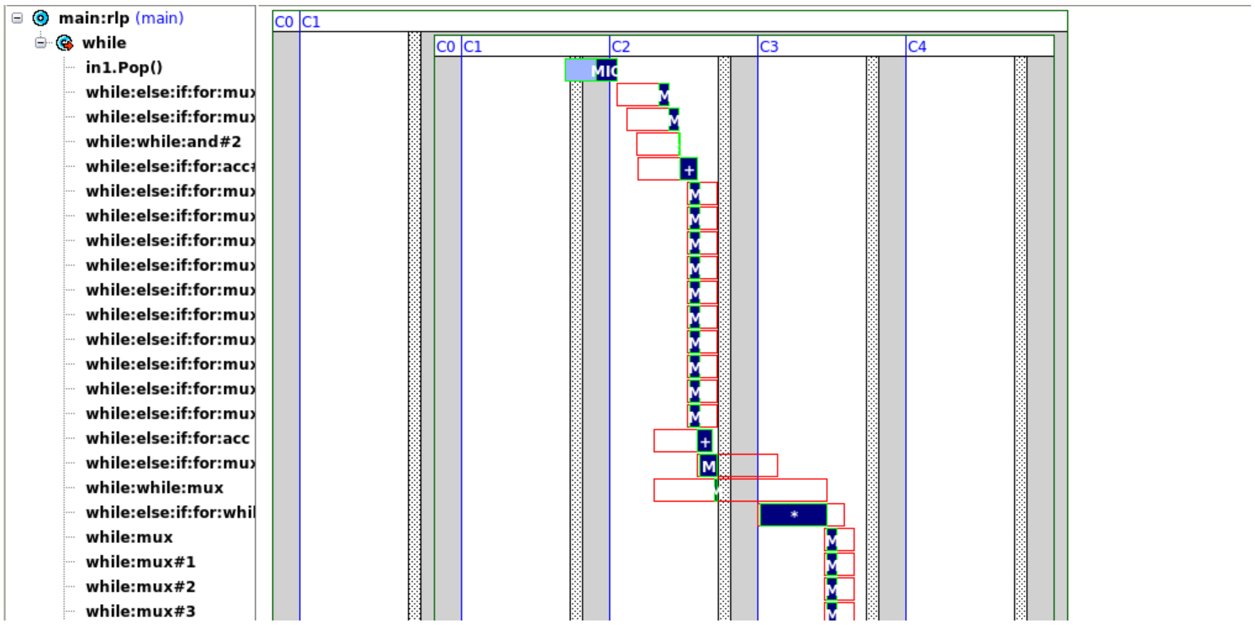
```
#elif DESIGN_8

    #pragma hls_pipeline_init_interval 1
    #pragma pipeline_stall_mode flush
    while(1) {
      packet p = in1.Pop();

      if (p.coeff == 0) {
        #pragma unroll yes
        for (int i=0; i < packet::data_len; i++)
          p.data[i] = 0;
      } else if (p.coeff != 1) {
        for (int i=0; i < packet::data_len; i++)
          p.data[i] *= p.coeff;
      }

      out1.Push(p);
    }
```

Throughput: 3.2 ns

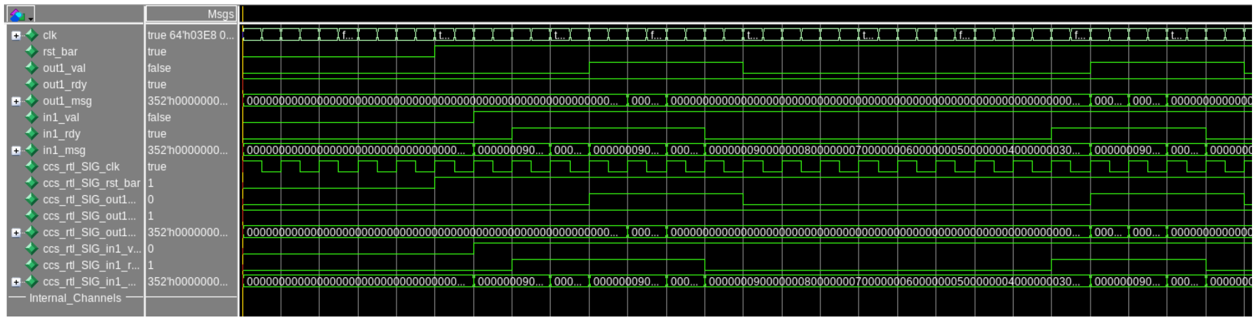
Area: 30364

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.


Msgs
clk true 64'h03E8 0...
rst_bar true
out1_val false
out1_rdy true
out1_msg 352'h0000000...
in1_val false
in1_rdy true
in1_msg 352'h0000000...
ccs_rtl_SIG_clk true
ccs_rtl_SIG_rst_bar 1
ccs_rtl_SIG_out1... 0
ccs_rtl_SIG_out1... 1
ccs_rtl_SIG_out1... 352'h0000000...
ccs_rtl_SIG_in1_v... 0
ccs_rtl_SIG_in1_r... 1
ccs_rtl_SIG_in1_... 352'h0000000...
Internal_Channels