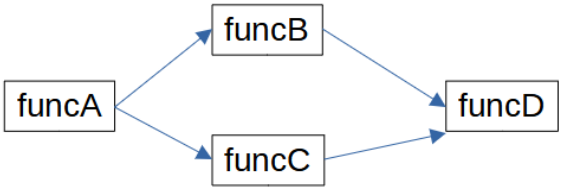
Lab_A: Dataflow Report

Introduction to Overall System

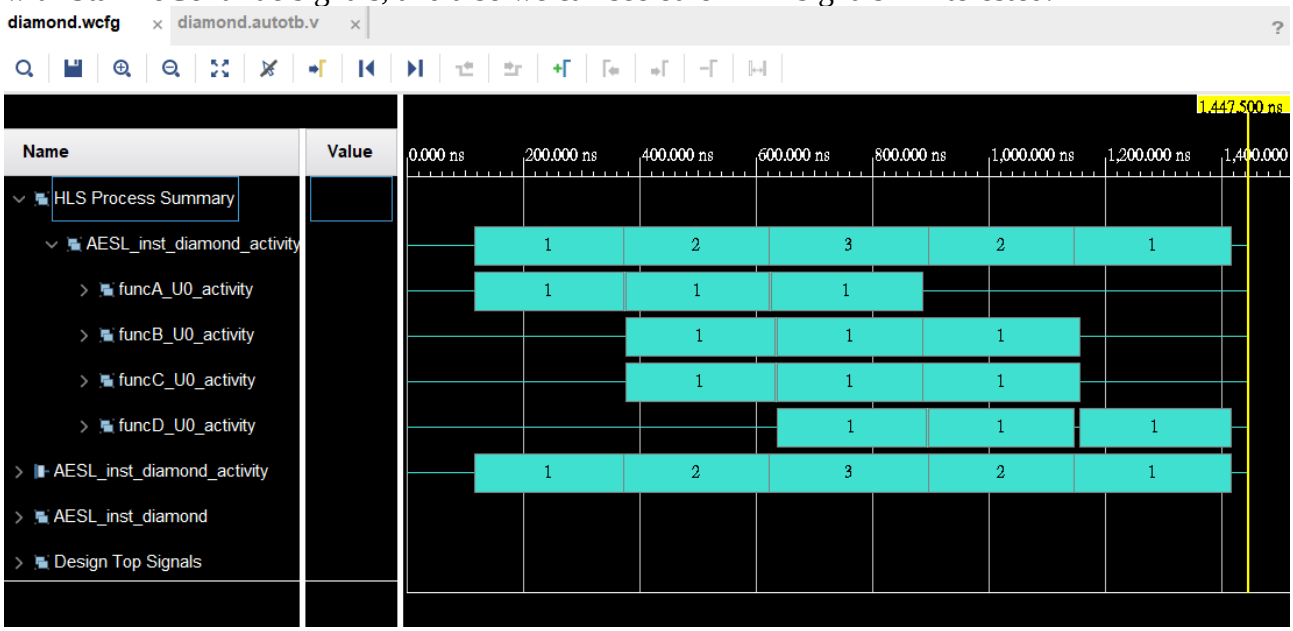
```
void diamond(data_t vecIn[N], data_t vecOut[N])
{
  data_t c1[N], c2[N], c3[N], c4[N];
#pragma HLS dataflow
  funcA(vecIn, c1, c2);
  funcB(c1, c3);
  funcC(c2, c4);
  funcD(c3, c4, vecOut);
}
```



The code above is the top function of this project. It is easy to see that the workflow of functions can be described by the figure next to the code, which is the same as the result of Dataflow Viewer. The TestBench calls the top function three times. In order to verify the initiation interval (II), we also perform another test which calls the top function twenty times, and the result agrees with the original TestBench.
Each subfunction uses pipeline and unroll with factor two, and since all of them have similar tasks and operating time, the overall system has little stalling time.

Observed and Learned

Vitis HLS can use Dataflow optimization to transform sequentially implementing C++ codes into parallel operations. If each subfunction has approximately the same operation time, they can operate compactly and with smaller II.
The Dataflow Viewer can show the dataflow in task level, and the stalling time, II and latency of each subfunction. The analysis of stalling time can help us to find which process we should accelerate or slow down to make the pipeline in task level better.
The Waveform Viewer or Timeline Viewer can visually show us the time each process consume and the result of dataflow optimization. We can see funcB and funcC run almost simultaneously, and there is only little gap between cycles. Waveform Viewer can show the stalling time (back pressure) with StallNoContinue signals, and also we can see other RTL signals if interested.

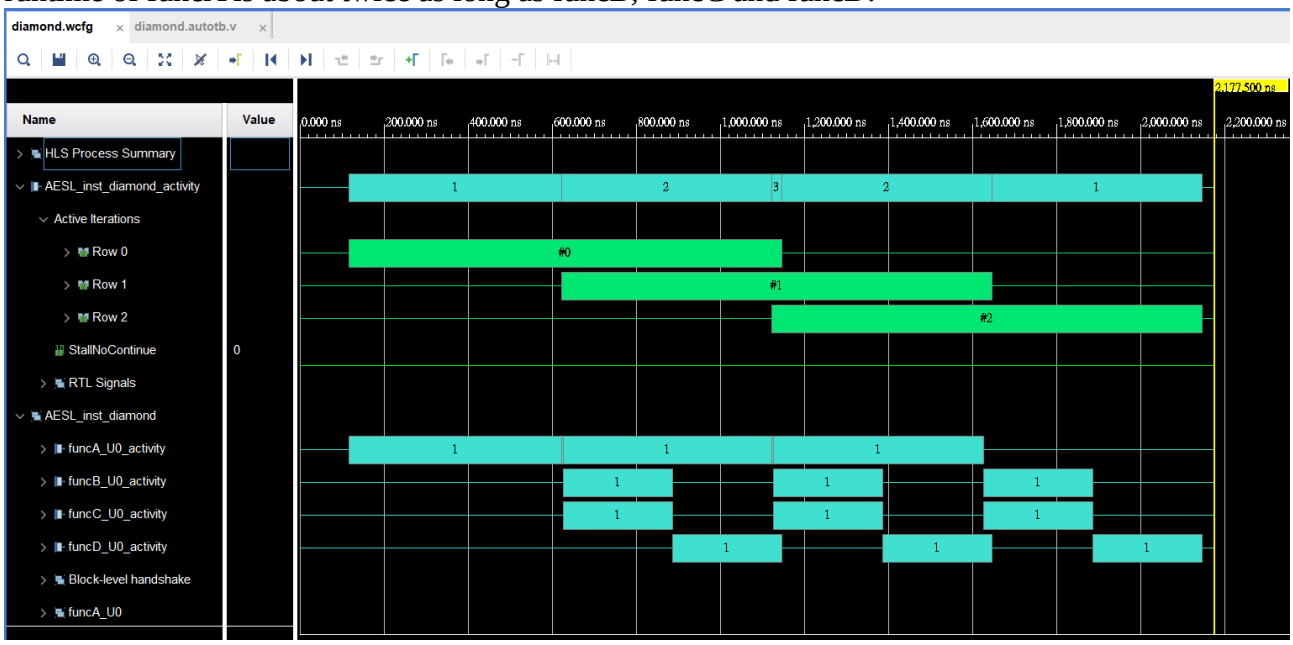


Back pressure implies the next process is still processing and not ready for new data. Forward pressure means the new process is stopped from starting a new execution by handshaking protocols. They can be read from the Dataflow Viewer after Cosimulation, in the form of Cosim Stall No Continue and Cosim Stall No Start respectively.

Also, we can see that nonparallel subfunctions (only funcB and funcC are parallel) have non-overlapping execution time. This is due to the use of PIPO. For example, funcB and funcC only start when the buffer from funcA is clear.
The default Dataflow used by Vitis is only composed of PIPO.

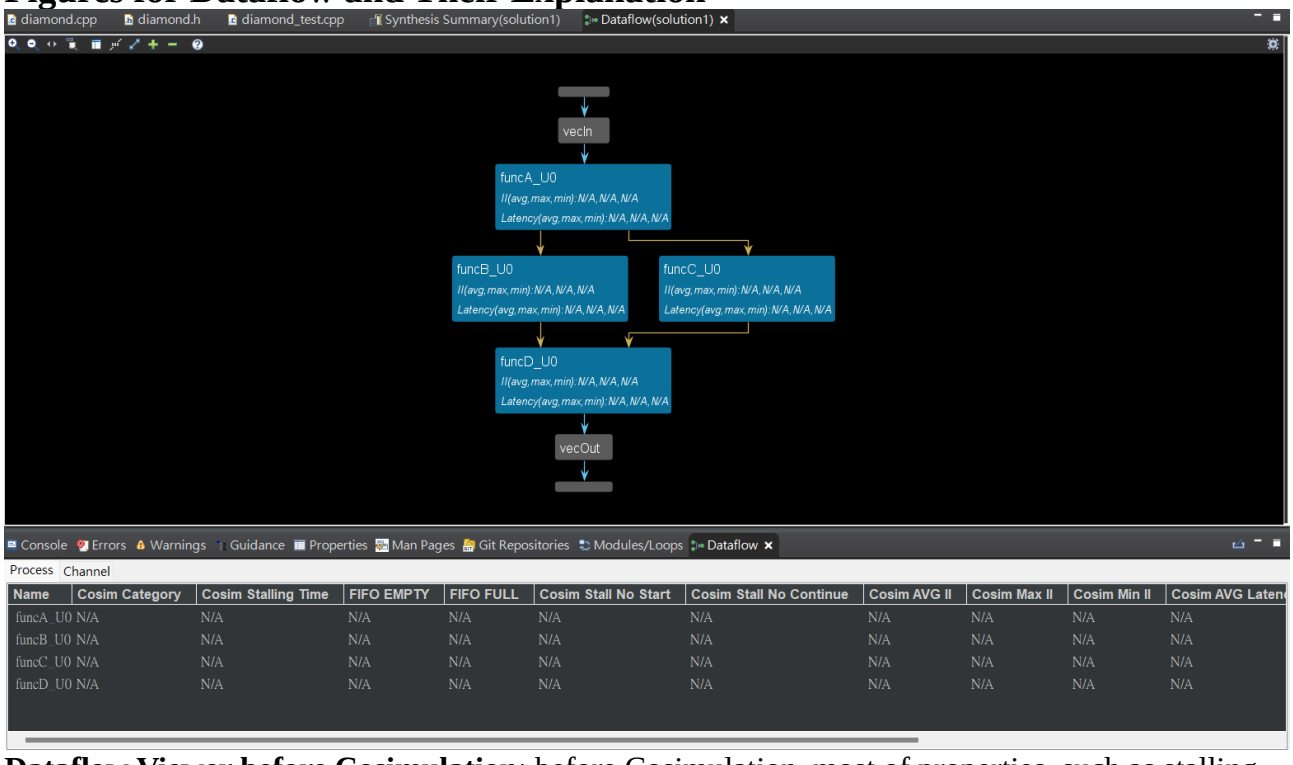
Some Additional Experiments

The figure below shows the experiment which sets the unroll factor of funcA to 1. We can see the runtime of funcA is about twice as long as funcB, funcC and funcD.

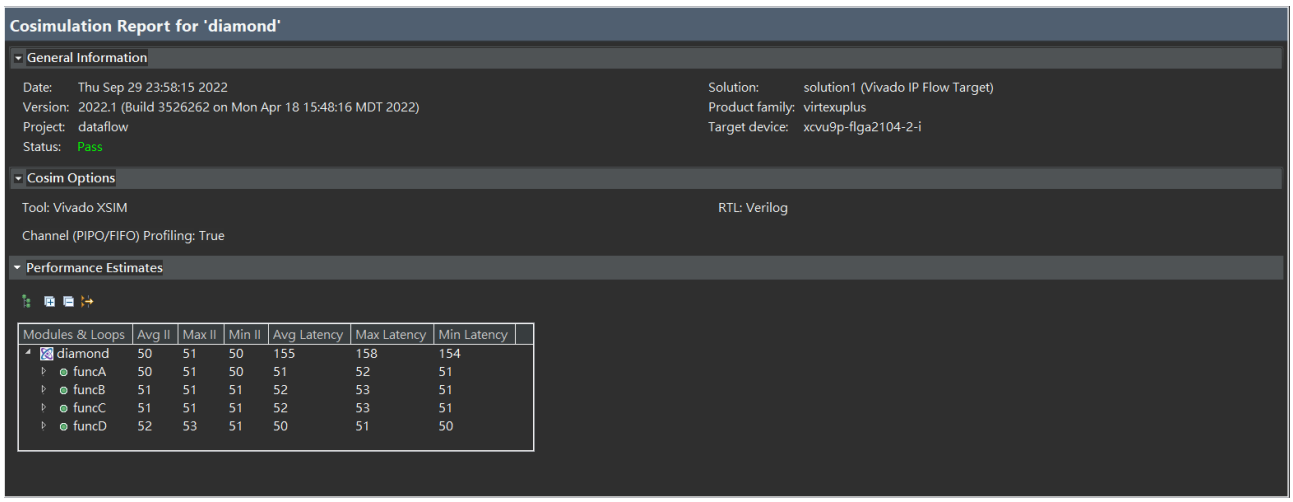


In Dataflow Viewer, we can see that the back pressure is erased, but the forward pressure is larger in funcB, funcC and funcD as the Cosim Stall No Start goes up to 24.02%.

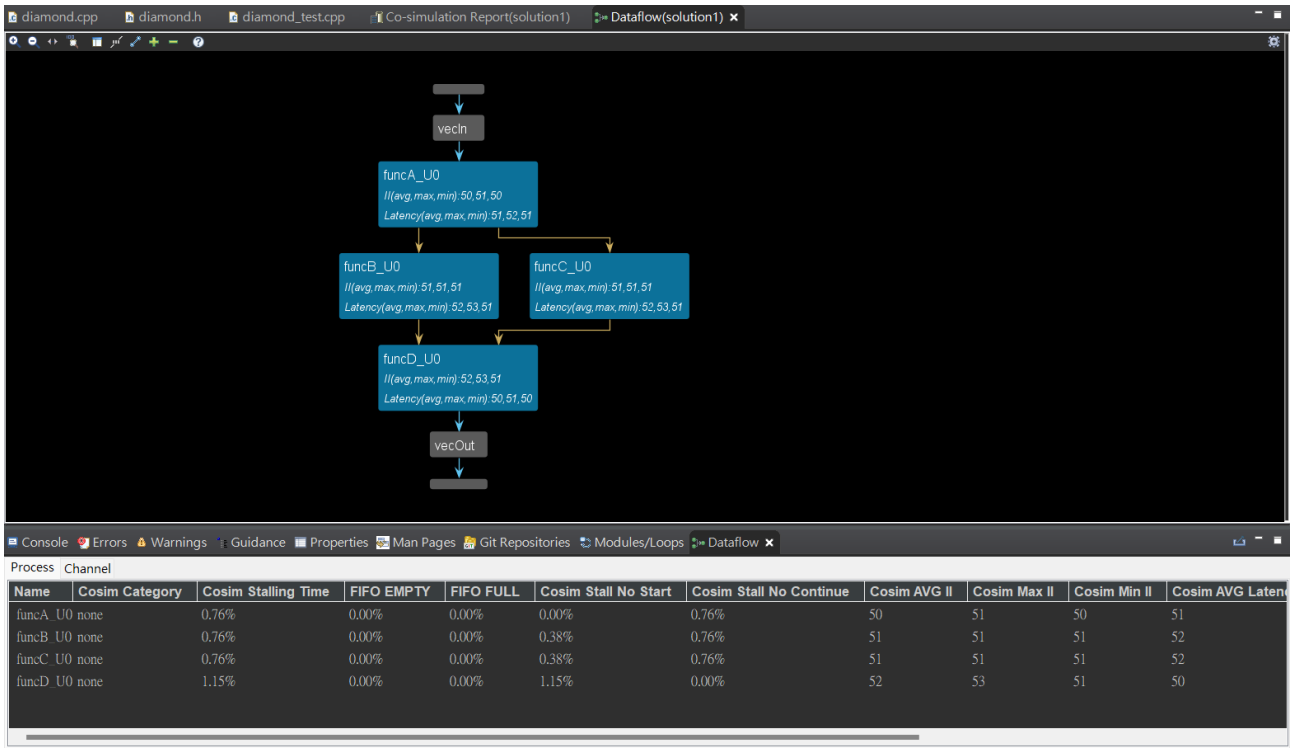
Figures for Dataflow and Their Explanation



Dataflow Viewer before Cosimulation: before Cosimulation, most of properties, such as stalling time, II, latency, are not calculated and not shown in Dataflow Viewer.



Cosimulation Result



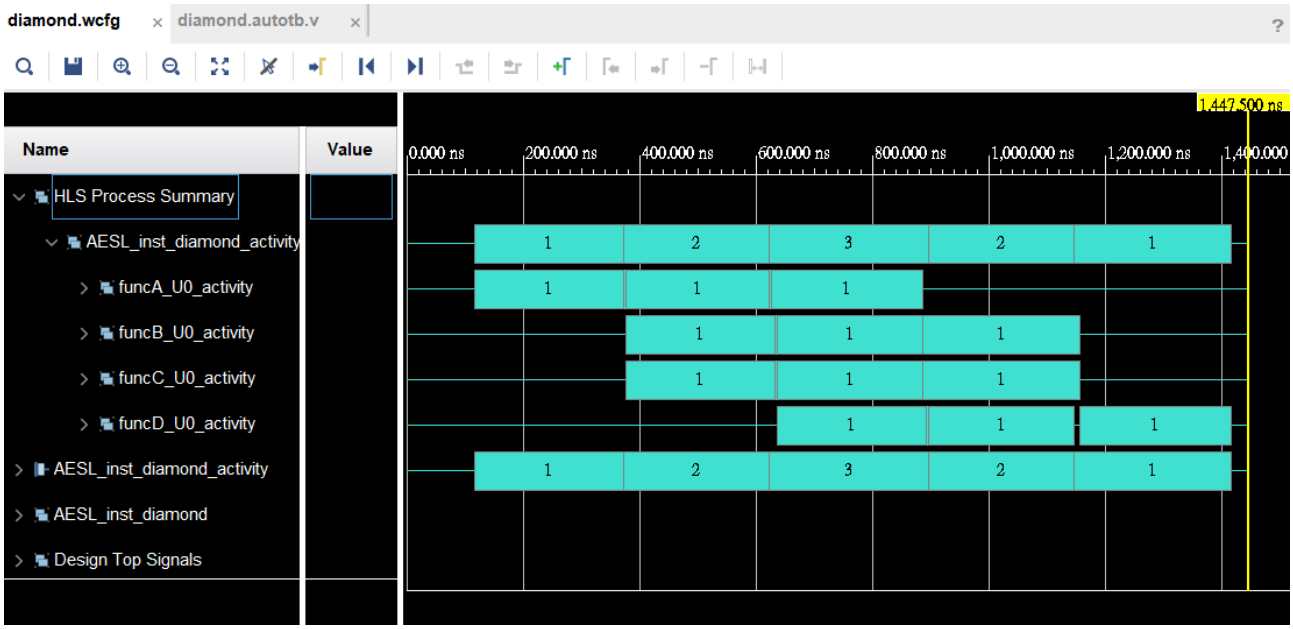
Dataflow Viewer after Cosimulation

Name	Cosim Category	Cosim Stalling Time	FIFO EMPTY	FIFO FULL	Cosim Stall No Start	Cosim Stall No Continue	Cosim AVG II	Cosim Max II	Cosim Min II
funcD_U0	none	1.15%	0.00%	0.00%	1.15%	0.00%	52	53	51
funcC_U0	none	0.76%	0.00%	0.00%	0.38%	0.76%	51	51	51
funcB_U0	none	0.76%	0.00%	0.00%	0.38%	0.76%	51	51	51
funcA_U0	none	0.76%	0.00%	0.00%	0.00%	0.76%	50	51	50

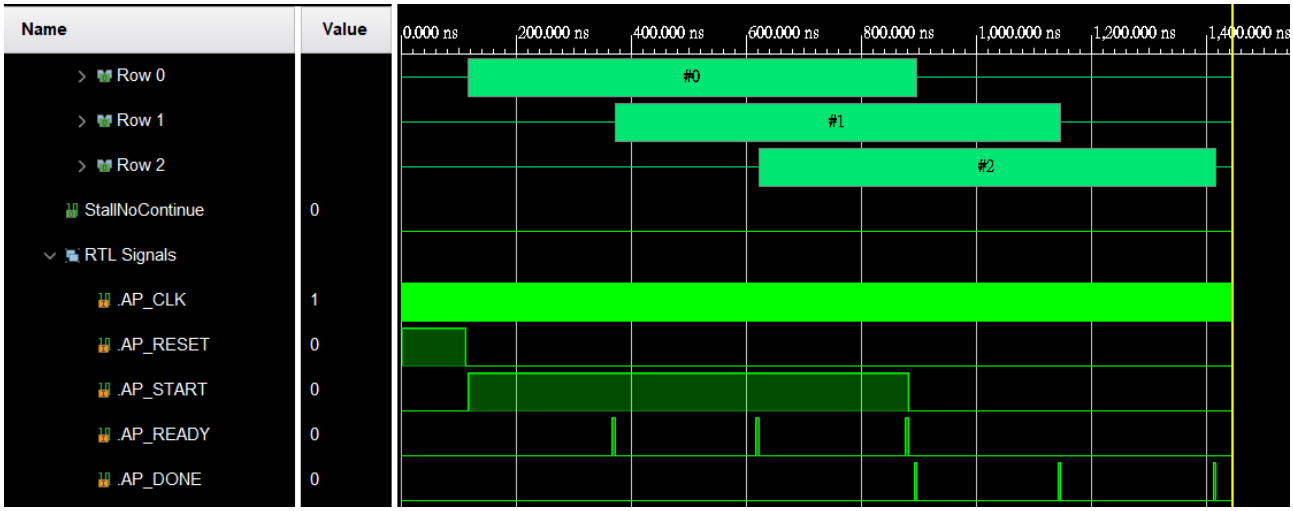
Process Table in Dataflow Viewer: after Cosimulation, we can see stalling time (which represents forward and back pressure), II and latency of each sunfunction in Process Table.

Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
c1	N/A	N/A	N/A	N/A	0	PIPO	PIPO	8	funcA_U0	funcB_U0	N/A
c2	N/A	N/A	N/A	N/A	0	PIPO	PIPO	6	funcA_U0	funcC_U0	N/A
c3	N/A	N/A	N/A	N/A	0	PIPO	PIPO	8	funcB_U0	funcD_U0	N/A
c4	N/A	N/A	N/A	N/A	0	PIPO	PIPO	7	funcC_U0	funcD_U0	N/A

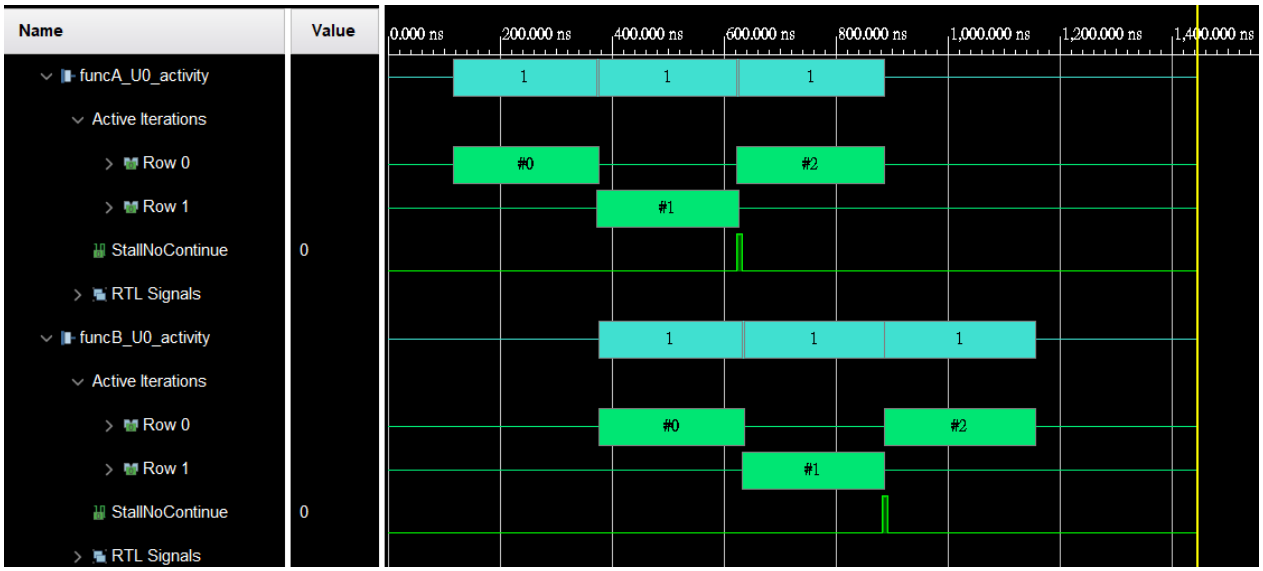
Channel Table in Dataflow Viewer: there is little information in Channel Table since we use PIPO which has less arguments than FIFO.



Function Activity Waveforms: the AESL_inst_diamond_activity signal represents how many Iterations overlap in a time period. By Dataflow Viewer there are three stage in the top function (since funcB and funcC run parallel), so there are at most three iterations run at the same time.

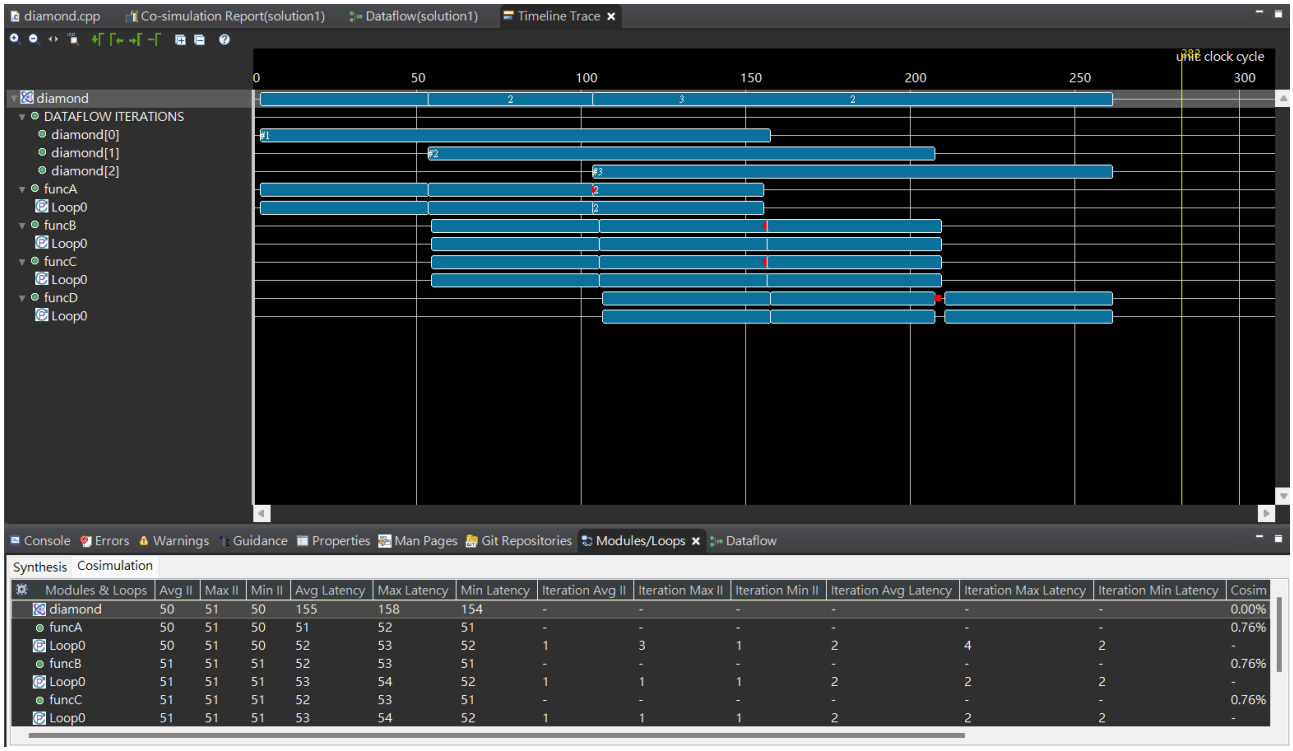


Start and End Time of Each Iteration and RTL Signals

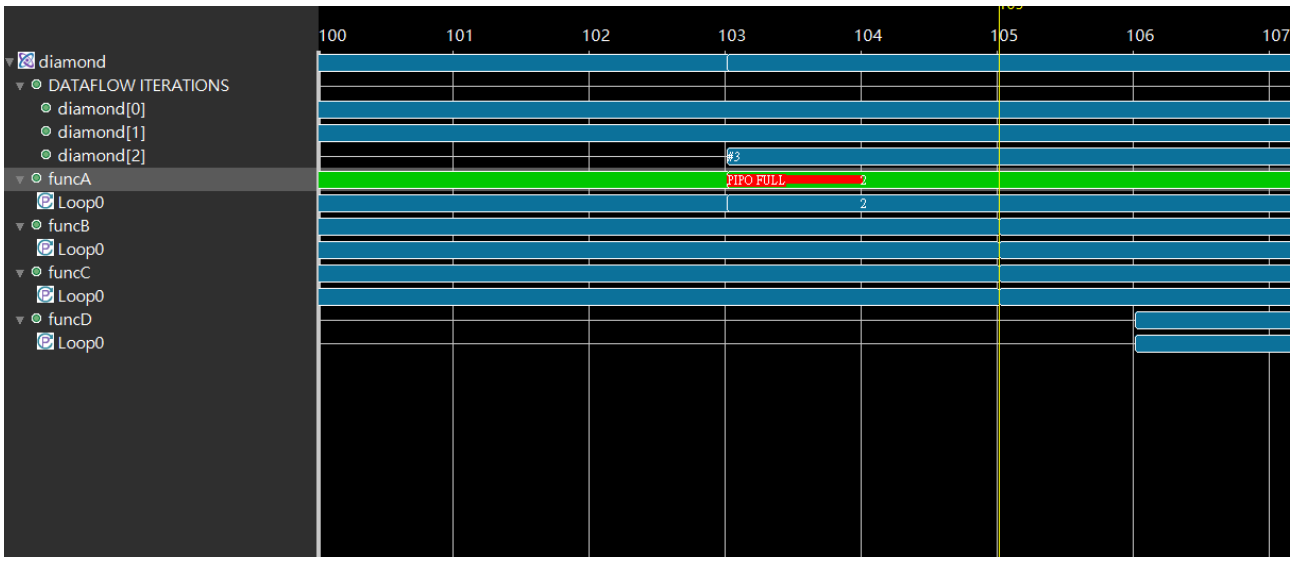


funcA and funcB Activities and Back Pressure Signals

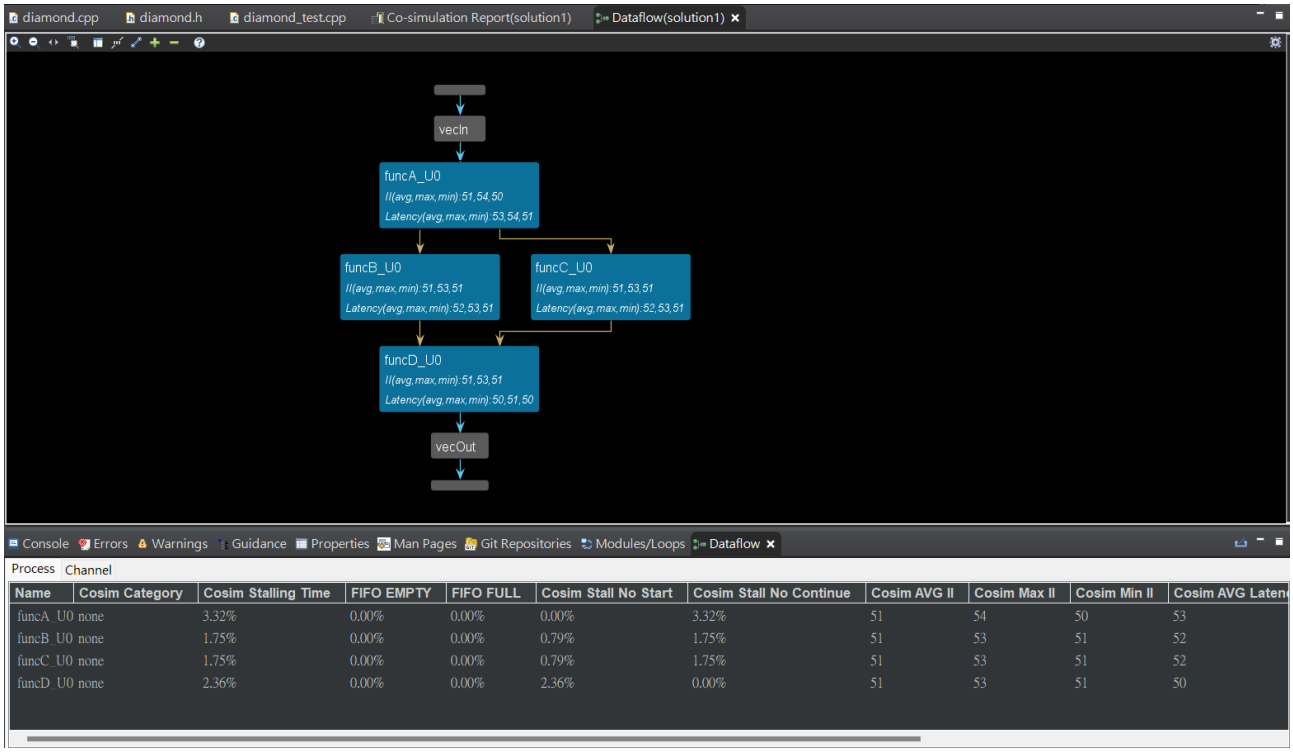
funcC and funcD Activities and Back Pressure Signals: StallNoContinue signals represent back pressure of processes, so they may appear in funcA, funcB and funcC but not funcD since it is the last process. It is not good to have StallNoContinue signals in a long period, but we must notice that maybe we do not have StallNoContinue signal but have lots of forward pressure. (For example, if funcA takes more time, then funcA, funcB and funcC will have no back pressure.)



Timeline Trace Viewer: it is similar to Waveform Viewer. They present the timeline in figures, but Timeline Trace Viewer does not need to open Vivado and it can be compared with Dataflow Viewer more easily. It also provides the timeline of each iteration and subfunction.



PIPO States in Timeline Trace Viewer: Timeline Trace Viewer uses PIPO states to indicate back pressure and forward pressure.



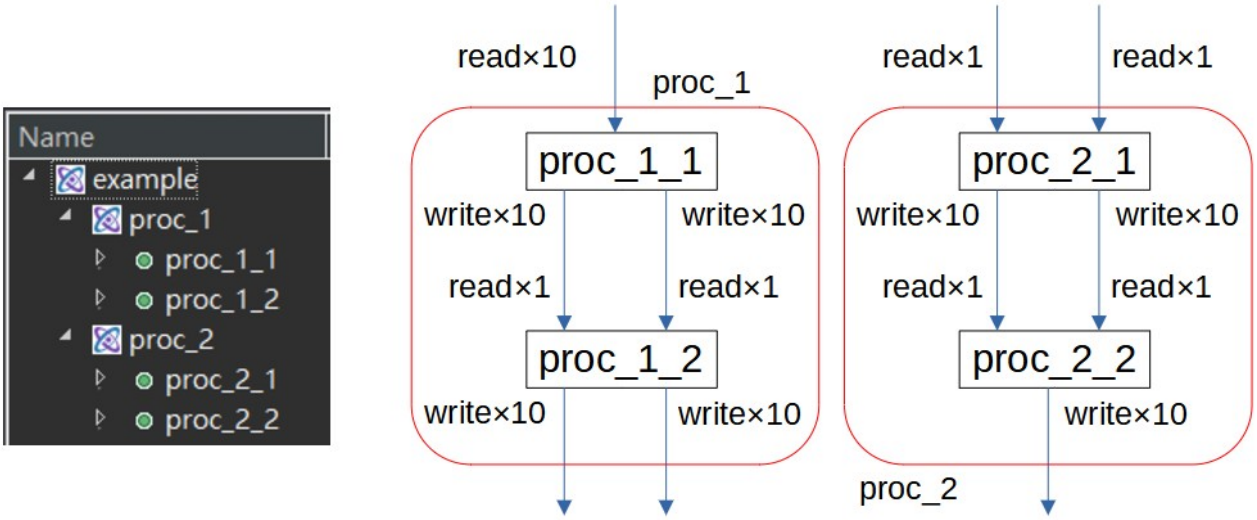
Dataflow Viewer after Cosimulation (Call Function 20 Times): since the original TestBench calls the top function only three times, we modify it to call the top function twenty times to check if II is the same as the original test. The result shows that it is similar to the test with three calls.

GitHub: https://github.com/b07901181/AAHLS-Lab-A-5

Lab_A: Deadlock Result

Introduction to Overall System

The system has function hierarchy shown in the figure below. And by inspiring the code, we can find that if there are two channels between subprocesses, the producer will write to the first channel for ten times and then to the second channel for ten times, while the consumer reads from the channels alternatively (that is, first channel, second channel, first channel, … and so on.)



This kind of data transfer causes deadlock if the depths of FIFOs are not enough. Also, depths of FIFOs and PIPOs might be insufficient for optimal performance. In this case we need depth equal to ten for the first channel and depth equal to one for the second channel. (Default depth is two.)
PIPOs are created when there are scalars or arrays between processes, and FIFOs are created using hls::streams or hls::stream_of_blocks variables.

```
hls::stream<int> data_channel1;
hls::stream<int> data_channel2;
```

Observed and Learned

We can use Dataflow Viewer to detect deadlock, and it also provides enough guidance to size FIFOs and prevent deadlock.
There are three ways to adjust FIFO depth: Manual FIFO Sizing, Global FIFO Sizing and Automated FIFO Sizing.
In Manual FIFO Sizing we can adjust each FIFO by ourselves with the data Cosim Max Depth in Dataflow Viewer, which gives us more freedom. We also need to notice that Cosim Max Depth gives the depth of FIFO that is able to run through the task, but it does not ensure the best performance.
In Global FIFO Sizing we adjust the setting of solution to provide enough depth for Cosimulation. It sets all depth in Cosimulation to a large enough value to ensure Cosimulation passes.
In Automated FIFO Sizing, Vitis Cosimulation automatically finds the right FIFO sizes by iterative tests. It increases depth of FIFOs that block writers in each iteration until the performance no longer improves, and sometimes this algorithm may not converge. Also, it may get a result with larger FIFO sizes, which uses more resource.

Some Additional Experiments

The tutorial says that the second channels only need depth one to run, but depth two can reach optimal performance. I try both cases to verify this result.

In the following tables from Dataflow Viewer we find that with second FIFO depth equal to two, the Cosim Stalling Time significantly decreases for every process, and also this design prevents all FIFO Full Status. Cosim II and latency also reduce a lot.

Process Channel

Name	Cosim Category	Cosim Stalling Time	FIFO EMPTY	FIFO FULL	Cosim Stall No Start	Cosim Stall No Continue	Cosim AVG II	Cosim Max II	Cosim Min II	Cosim AVG Latency	Cosim Max Latency	Cosim Min Latency
proc_1_1_U0	write_block	26.89%	0.00%	26.89%	0.00%	0.00%	41	46	34	38	42	33
proc_1_2_U0	read_block	40.09%	23.11%	16.98%	0.00%	0.00%	47	56	43	45	55	42
proc_1_U0	write_block	19.81%	0.00%	16.98%	2.83%	0.00%	40	43	34	62	65	56
proc_2_1_U0	write_block	50.94%	33.96%	16.98%	0.00%	0.00%	55	79	43	51	78	42
proc_2_2_U0	read_block	77.36%	77.36%	0.00%	0.00%	0.00%	55	79	43	51	78	42
proc_2_U0	read_block	33.96%	33.96%	0.00%	0.00%	0.00%	55	79	43	52	79	43

Process Channel

Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block	6.60%	0.00%	10	10	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel2	read_block and write_block	33.96%	16.98%	1	1	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel1	none	0.00%	0.00%	10	10	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel2	read_block and write_block	23.11%	26.89%	1	1	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel1	read_block	18.87%	0.00%	10	10	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link
data_channel2	read_block and write_block	77.36%	16.98%	1	1	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link

depth=1

Process Channel

Name	Cosim Category	Cosim Stalling Time	FIFO EMPTY	FIFO FULL	Cosim Stall No Start	Cosim Stall No Continue	Cosim AVG II	Cosim Max II	Cosim Min II	Cosim AVG Latency	Cosim Max Latency	Cosim Min Latency
proc_1_1_U0	none	0.00%	0.00%	0.00%	0.00%	0.00%	26	28	25	24	24	24
proc_1_2_U0	read_block	11.94%	11.94%	0.00%	0.00%	0.00%	30	38	25	28	37	24
proc_1_U0	none	2.24%	0.00%	0.00%	2.24%	0.00%	26	28	25	38	41	38
proc_2_1_U0	none	22.39%	22.39%	0.00%	0.00%	0.00%	35	52	25	31	51	24
proc_2_2_U0	read_block	64.18%	64.18%	0.00%	0.00%	0.00%	35	52	25	31	51	24
proc_2_U0	read_block	22.39%	22.39%	0.00%	0.00%	0.00%	35	52	25	32	52	25

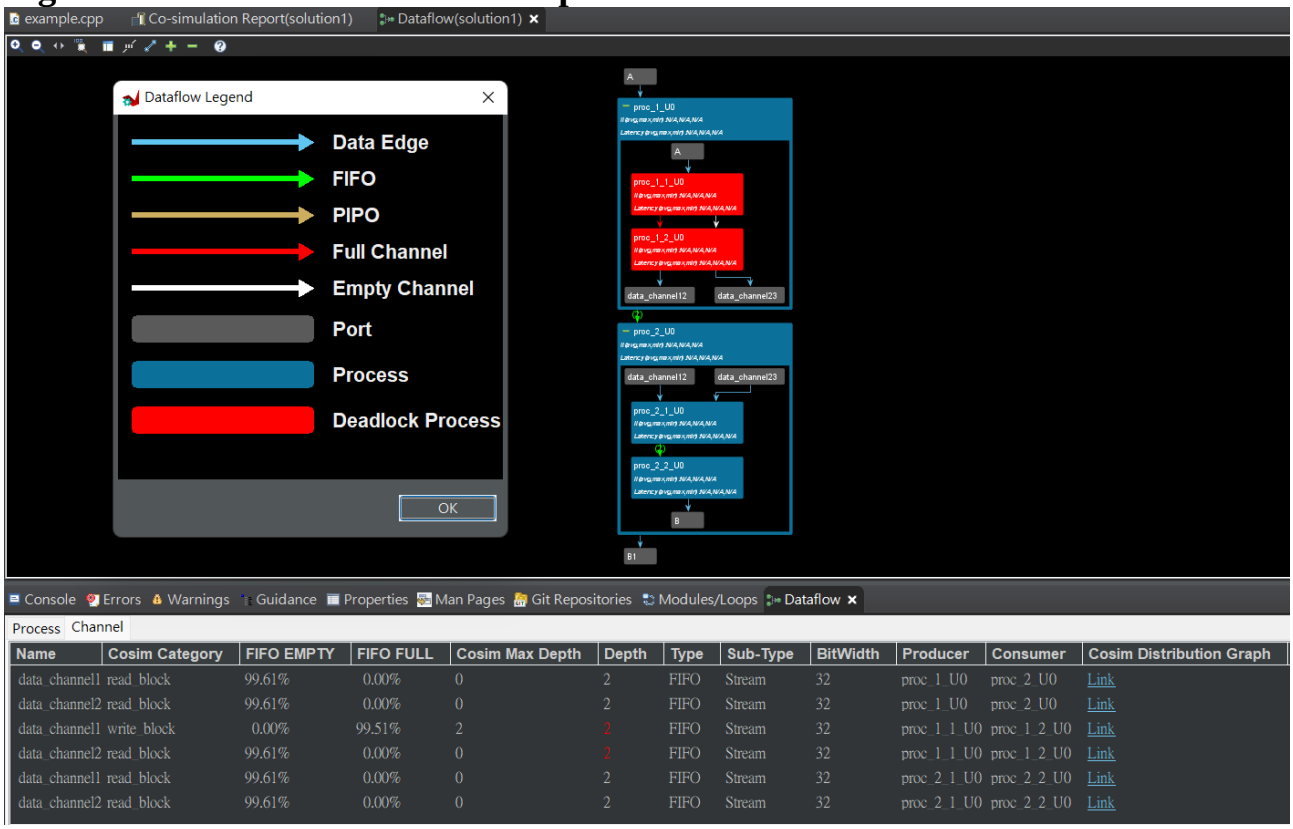
Process Channel

Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block	10.45%	0.00%	10	10	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel2	read_block	22.39%	0.00%	1	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel1	none	0.00%	0.00%	10	10	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel2	read_block	11.94%	0.00%	1	2	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel1	read_block	25.37%	0.00%	10	10	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link
data_channel2	read_block	64.18%	0.00%	1	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link

depth=2

I think this may result from the decrease of FIFO Full situation.

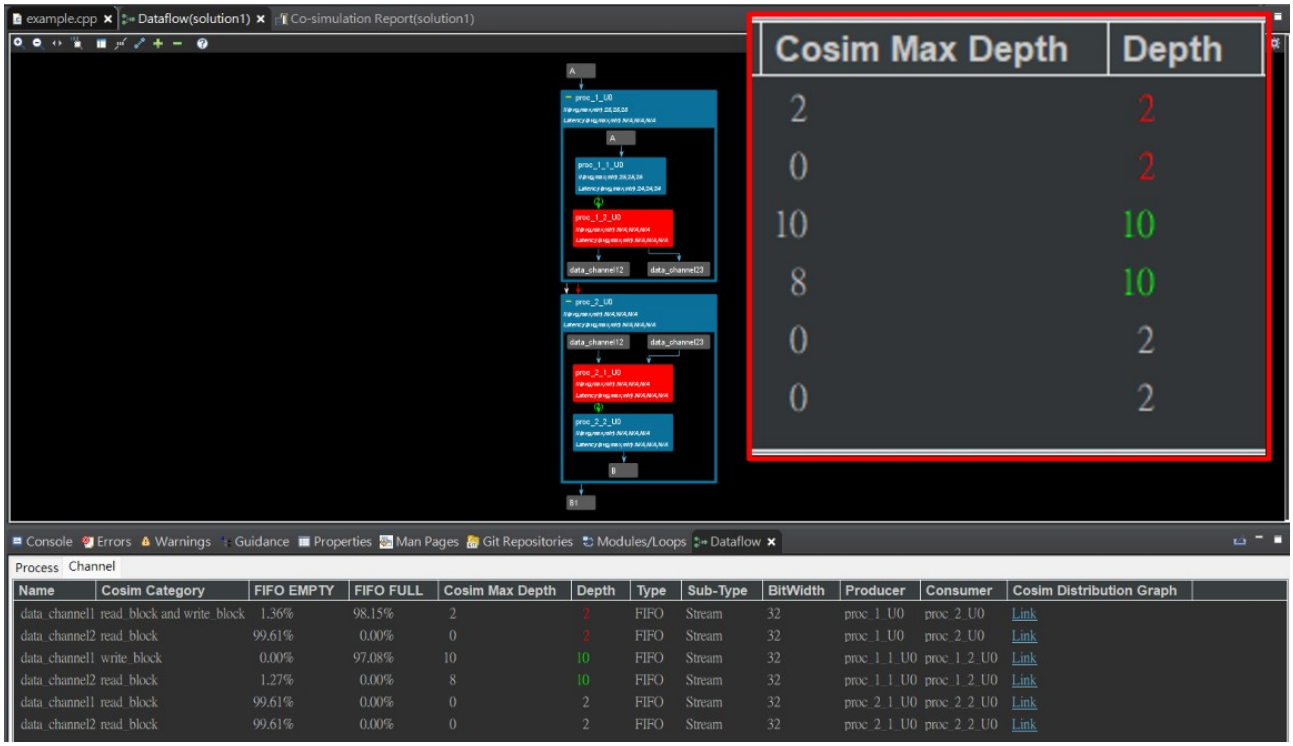
Figures for Deadlock and Their Explanation



Process Channel

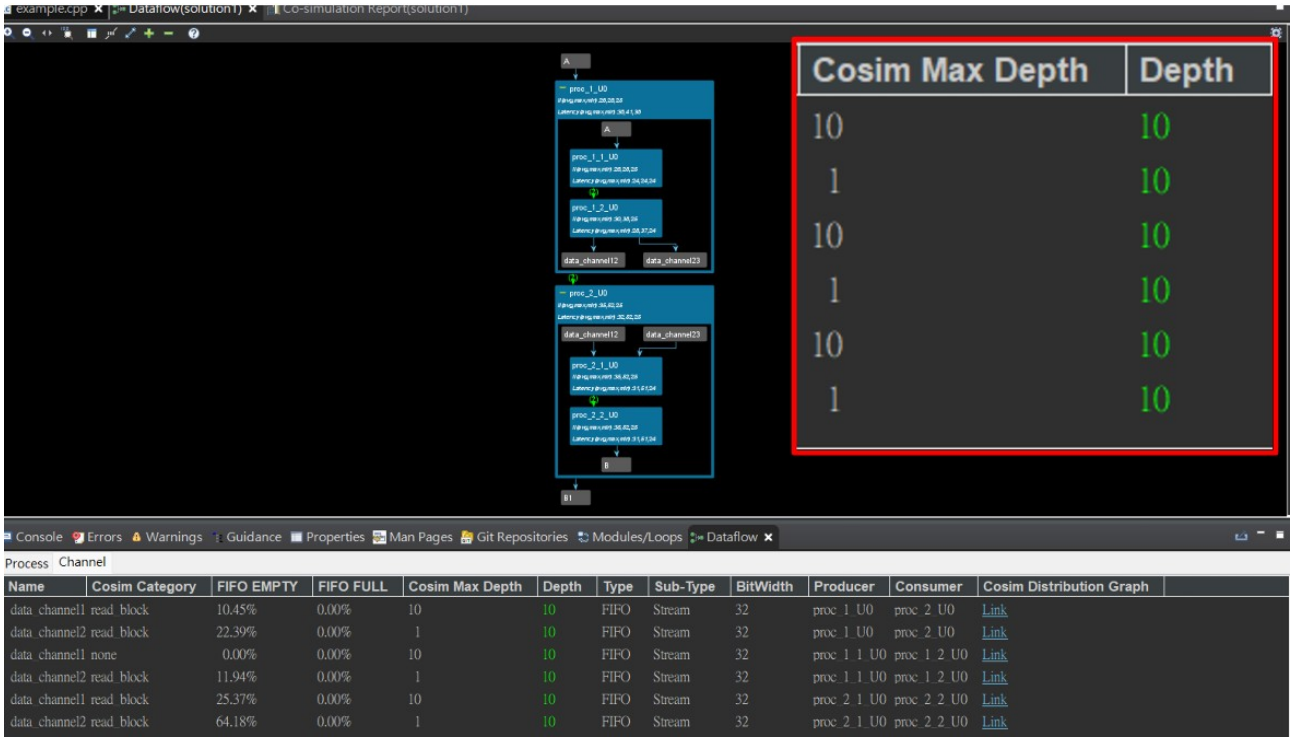
Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel2	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel1	write_block	0.00%	99.51%	2	2	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel2	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel1	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link
data_channel2	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link

Dataflow Viewer after Cosimulation: the Depth written in red indicates the first blocked channels, and we will first modify their depth using Manual FIFO Sizing.



Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block and write_block	1.36%	98.15%	2	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel2	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel1	write_block	0.00%	97.08%	10	10	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel2	read_block	1.27%	0.00%	8	10	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel1	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link
data_channel2	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link

Dataflow Viewer after Manual FIFO Sizing: new blocked channels may appear after we resolved original blocked channels. Manual FIFO Sizing is an iterative process.

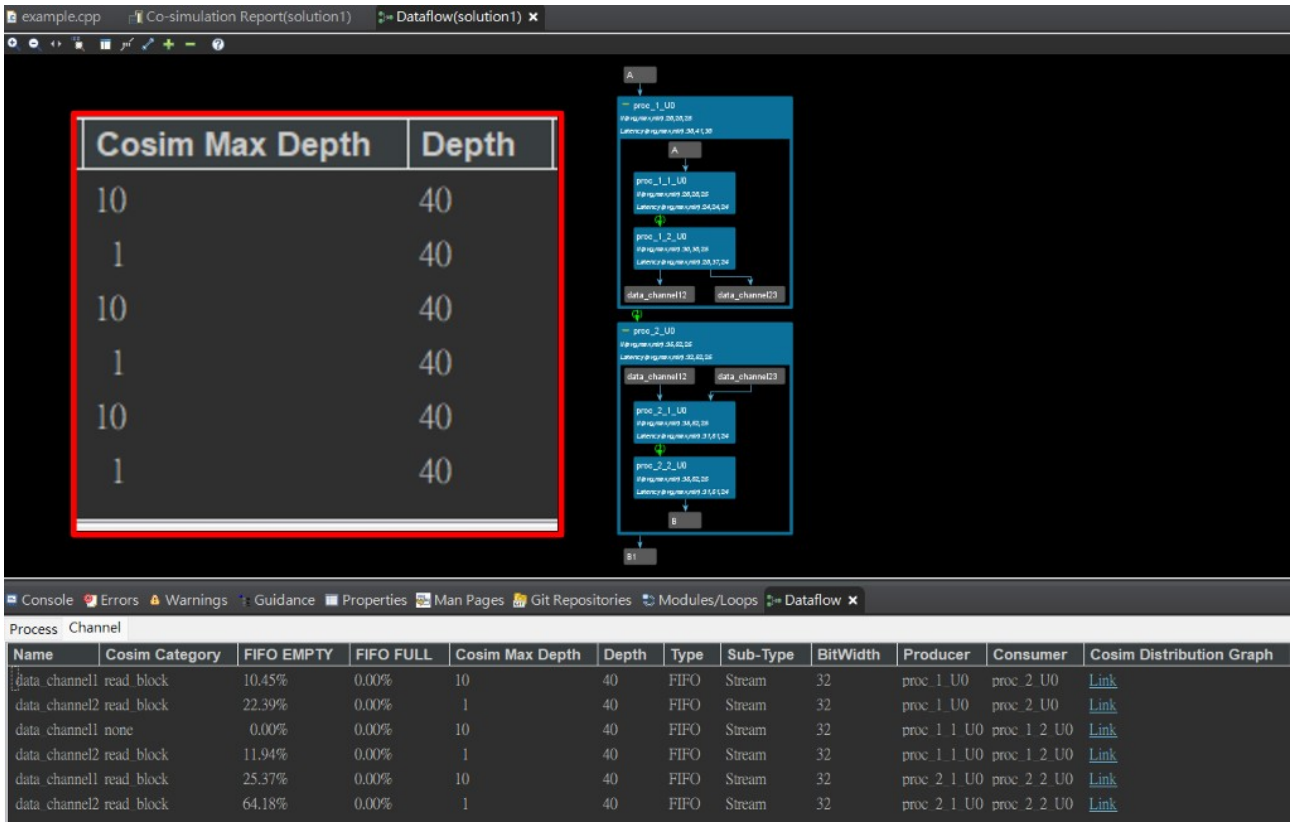


Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block	10.45%	0.00%	10	10	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel2	read_block	22.39%	0.00%	1	10	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel1	none	0.00%	0.00%	10	10	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel2	read_block	11.94%	0.00%	1	10	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel1	read_block	25.37%	0.00%	10	10	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link
data_channel2	read_block	64.18%	0.00%	1	10	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link

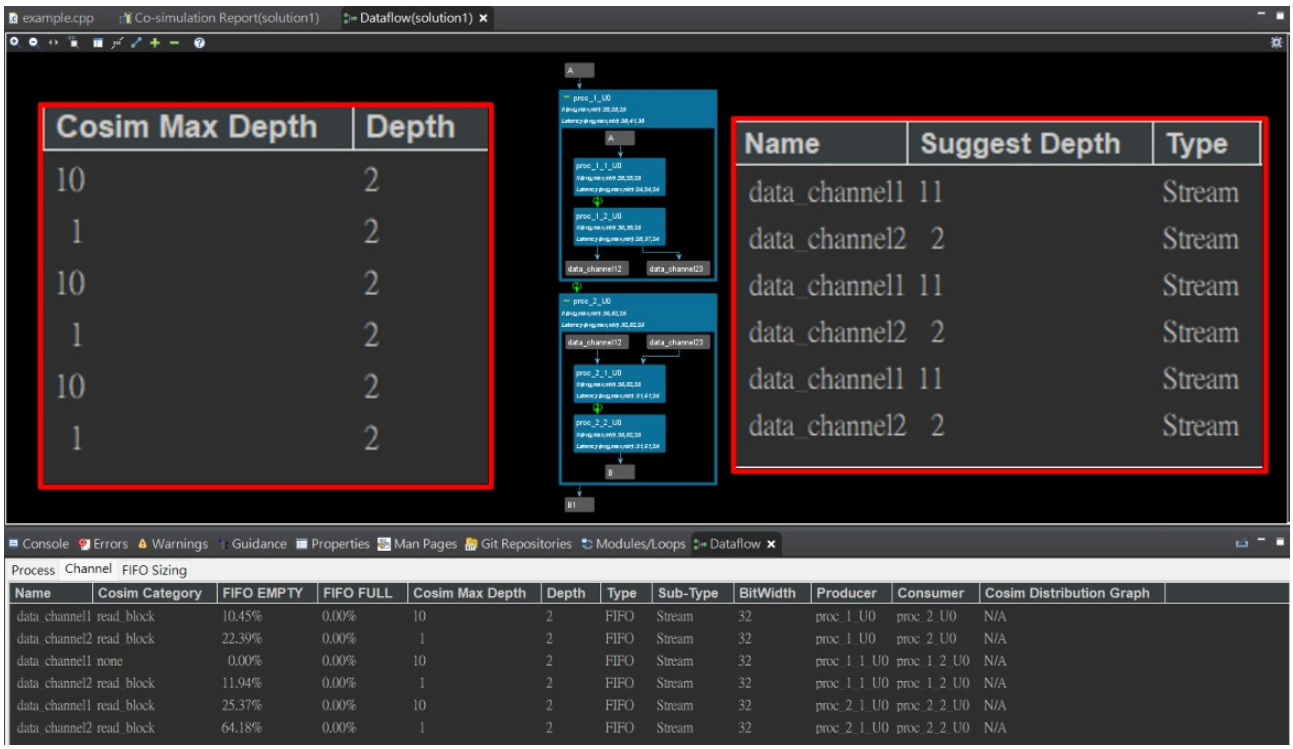
Dataflow Viewer after All Manual FIFO Sizing: Cosim Max Depth shows that we do not need depth equal to ten for data_channel2.

```cpp
void example(hls::stream<int>& A, hls::stream<int>& B){
#pragma HLS dataflow
#pragma HLS INTERFACE ap_fifo port=&A
#pragma HLS INTERFACE ap_fifo port=&B
    hls::stream<int> data_channel1;
    #pragma HLS stream depth=10 variable=data_channel1 // Modified by Vitis HLS
    hls::stream<int> data_channel2;

    proc_1(A, data_channel1, data_channel2);
    proc_2(data_channel1, data_channel2, B);
}

void proc_1(hls::stream<int>& A, hls::stream<int>& B, hls::stream<int>& C){
#pragma HLS dataflow
    hls::stream<int> data_channel1;
    #pragma HLS stream depth=10 variable=data_channel1 // Modified by Vitis HLS
    hls::stream<int> data_channel2;

    proc_1_1(A, data_channel1, data_channel2);
    proc_1_2(B, C, data_channel1, data_channel2);
}
```

New Pragma Modified by Vitis after Back-Annotations: no matter which method of FIFO Sizing we used, we need to write the depth in Cosimulation back into the code with pragma. The Back-Annotate step would help us do this automatically.



Dataflow Viewer after Global FIFO Sizing: Global FIFO Sizing makes all channels have the same depth.



Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block	10.45%	0.00%	10	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	N/A
data_channel2	read_block	22.39%	0.00%	1	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	N/A
data_channel1	none	0.00%	0.00%	10	2	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	N/A
data_channel2	read_block	11.94%	0.00%	1	2	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	N/A
data_channel1	read_block	25.37%	0.00%	10	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	N/A
data_channel2	read_block	64.18%	0.00%	1	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	N/A

Dataflow Viewer and Suggest Depth after Automated FIFO Sizing: we can see that the Suggest Depth is not equal to Cosim Max Depth.

GitHub: https://github.com/b07901181/AAHLS-Lab-A-5