Report

Briefly Introduction

這個主題主要在學習Vitis HLS GUI的dataflow viewer的使用以及如何利用其對dataflow做優化。首先第一個lab是在用viewer查看4個function之間的結構與dataflow，並且學會看一些基本的參數，像是process的Latency、II、Cosim Stalling Time等等以及channel的Cosim Stall No Start/Continue、Depth、Cosim Max Depth等等。同時也考察了它做C/RTL Co-simulation後的waveform。第二個lab則是再講如何利用此工具去做FIFO size上的調整從而優化performance，也避免Deadlock的問題。主要介紹了2種手動的方法以及1種automation的方法來解決因Deadlock在Co-sim時出現的error。

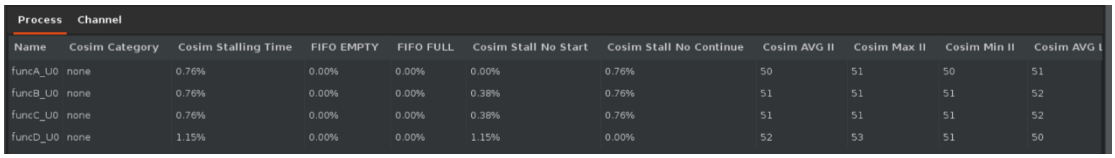
Code Explain

第一個lab的code很簡單，就是function傳輸N個3的非負整倍數的數據給function B跟function C，function B把數據都加25輸出，function C把數據都乘2輸出，function D最後再把function B的輸出加上兩倍function C的輸出從而得到Output。

第二個lab的code的流程則是分為2個processor，並且每個processor都有2個channel。其中第一個processor的第一個channel讀取A的data並與第二個channel做累加，然後從第二個channel輸出兩條channel的data給第二個processor，後者類似地，先一個channel讀寫再給另一個channel累加讀寫，最終得到輸出。

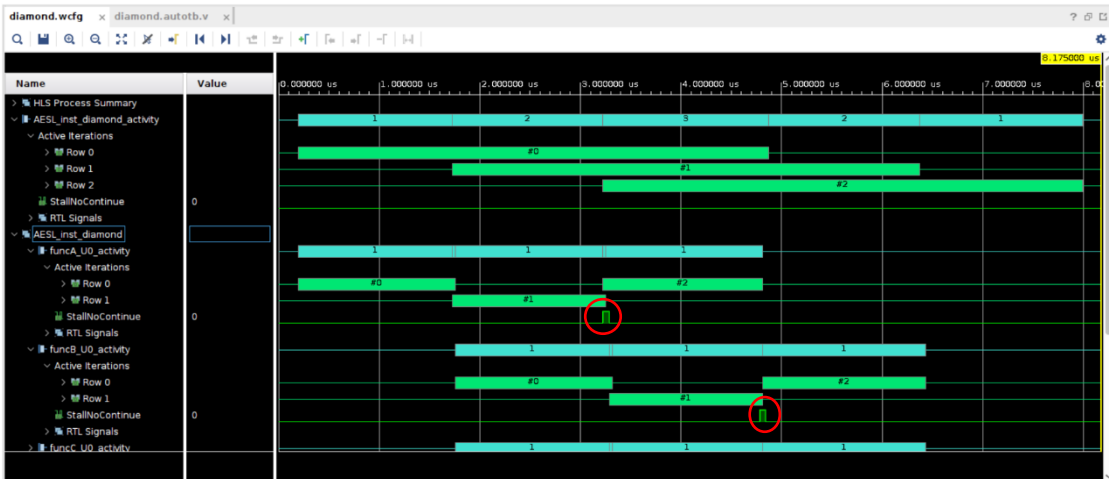
Performance Parameters & Waveform Analysis

LAB1



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 L
funcA_U0	none	0.76%	0.00%	0.00%	0.00%	0.76%	50	51	50	51
funcB_U0	none	0.76%	0.00%	0.00%	0.38%	0.76%	51	51	51	52
funcC_U0	none	0.76%	0.00%	0.00%	0.38%	0.76%	51	51	51	52
funcD_U0	none	1.15%	0.00%	0.00%	1.15%	0.00%	52	53	51	50



Cosim Stalling Time指用於此特定過程的停頓所花費的模擬時間的百分比，可以看到佔比很小卻仍存在。Cosim Stall No Start和Cosim Stall No Continue分別表示forward與back pressure：前者指無法通過the block handshaking protocols開始執行另一個迭代；後者意味著consumer process仍在處理該進程已產生的數據，尚未為下一組數據做好準備。可以看到這些參數指標其比例都很小，但這些指標可以反映一些潛在問題，如果當這些指標數值過高時，可能會導致性能降低甚至deadlock。從waveform圖中也可以看到紅色圓圈圈起來的impulse就可能代表了這些地方出現了stalling的問題。

LAB2

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 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

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	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

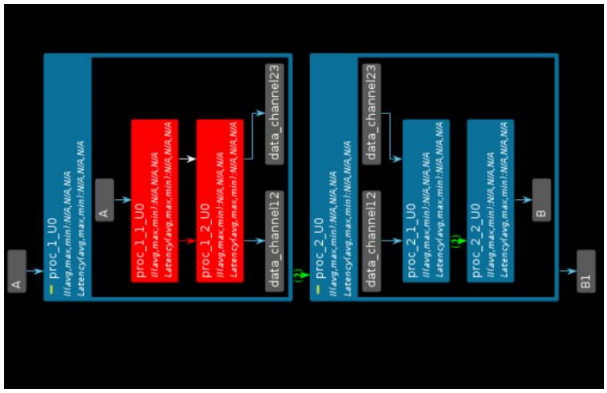
這裡我特地放一張有error的channel參數表格作對照，可以清楚地看到如果沒有做dataflow的優化或是Depth沒有調整好，在Co-sim時就會出錯，呈現如圖中的紅色數字一般，並且呈現出來的FIFO empty也會要么爆高，要么爆低。但是在做了Depth的調整之後，如第二張圖，很明顯error就消失了，並且參數值也變得正常。

How To Optimize

首先LAB 1主要是操作與觀察型的，沒有什麼值得優化的地方。那我就直接講LAB 2中我做優化的過程，這裡主要就是手動優化Channel Depth。剛開始，我先預設所有Depth都為4，可以看到proc_1_1與proc_1_2之間出現了deadlock，如下圖所示。

Process Channel

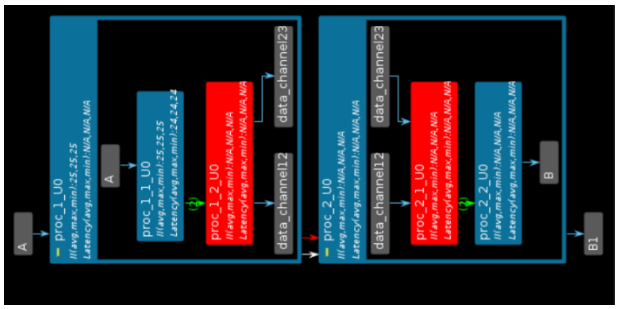
Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel2	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel1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



然後我把data_channel1的write_block Depth與data_channel2的read_block Depth改為10，結果這次deadlock轉移到proc_1_2與proc_2_1之間的channel了，如下圖所示。

Process Channel

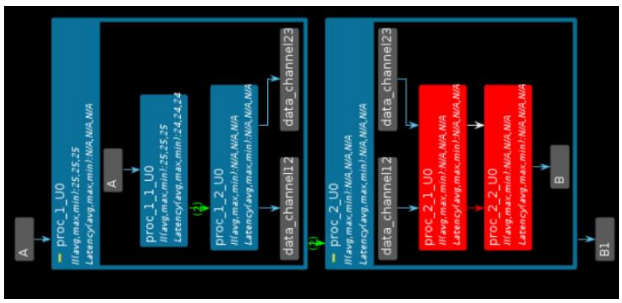
Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block and write_block	1.36%	98.15%	2	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel2	read_block	99.61%	0.00%	0	2	FIFO	Stream	32	proc_1_U0	proc_2_U0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



於是這次我直接把 data_channel1 的 read_block Depth 改為 10，結果這次是 proc_2_1 和 proc_2_2 之間出現 deadlock，如下圖所示。

Process Channel

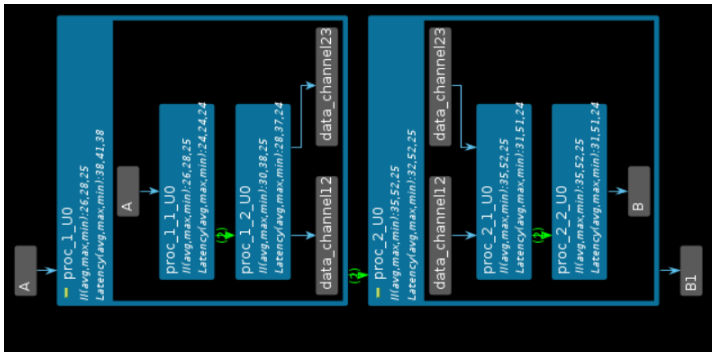
Name	Cosim Category	FIFO EMPTY	FIFO FULL	Cosim Max Depth	Depth	Type	Sub-Type	BitWidth	Producer	Consumer	Cosim Distribution Graph
data_channel1	read_block	1.34%	0.00%	10	10	FIFO	Stream	32	proc_1_U0	proc_2_U0	Link
data_channel2	read_block and write_block	2.59%	96.73%	2	2	FIFO	Stream	32	proc_1_U0	proc_2_U0	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	1.25%	0.00%	10	10	FIFO	Stream	32	proc_1_1_U0	proc_1_2_U0	Link
data_channel1	read_block and write_block	2.69%	96.83%	2	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link
data_channel2	read_block	99.62%	0.00%	0	2	FIFO	Stream	32	proc_2_1_U0	proc_2_2_U0	Link



最後我乾脆把所有的 Depth 都設為 10，終於 Co-sim 沒有再出現 error 了。並且仔細觀察，比較 Cosim Max Depth 與 Depth，也發現實際上只要三個 channel 深度為 10 就可以解決問題，其它是不需要改變的。如下圖所示:

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	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



Trade-off: 雖然我這樣手動調整 channel size 有些繁瑣，但對於剛入門者，卻可以更加透徹的瞭解 size 優化是怎麼運作的，並且也能及時對症下藥修改 size，一步一步一點點地改也省總資源。但確實這種手工的方法存在一些缺點: 1. 需要重複的做 Co-sim，可能要好長一段時間才能收斂。 2. 在某些情況下它甚至不會收斂。所以這時候用 FIFO size automation 的方法調整就方便許多。

What I Learned

The Dataflow viewer enables the following throughput analysis tasks:

1. The graph shows the overall topology of the DATAFLOW region and shows what type of channels (FIFO/PIPO) were inferred for communication between the tasks in the DATAFLOW region. Analyzing each channel and process can be useful to resolve issues such as deadlock or poor throughput because of bad FIFO sizing.
2. The co-simulation data helps with the FIFO sizing problem by tracking the maximum size of the FIFO during the course of the simulation and thereby giving the user a good reference point on how to size the FIFOs. In addition, when running Co-simulation, automatic deadlock detection can highlight the processes and channels involved in the deadlock allowing the user to quickly narrow the focus and fix the issue.
3. In addition to FIFO sizing, the data reported after Co-simulation also provides, on a per process and channel basis, the time spent stalling either waiting for input or blocked from writing output. The graph helps the user understand such issues and manage how the channels are sized to accommodate slow producers versus fast

consumers and/or vice-versa. In addition, the graph is useful in understanding how reading from the input in the middle of a DATAFLOW region can impact performance. This is a fairly common scenario where performance can be impacted.

Problem Encountered & Solution

在做LAB2時，用範例的程式來執行他的流程會出現一行錯誤：error: [hls 200-1023] part 'virtex7' is not installed。原來他的set_part是用virtex7，這主要應該跟platform或board有關，看來工作站的Vitis HLS中可能沒有或不支援這個。於是我上網查找資料，發現這個東西在安裝Vivado平台的時候就會預設安裝了，而FPGA工作站的Vivado竟然沒有，有點匪夷所思。嘗試了很多辦法，在不動到工作站環境配置或是重新安裝Vivado的前提下，我已經無解。然後我想到這裡的流程只是要跑到HW-emu而已，所以其實platform是啥也不重要，於是我模仿LAB 1，使用它所用的{xcvu9p-flga2104-2-i}就順利完成LAB2了。

Github Link

https://github.com/Barry-Sung/HLS_LABA_Dataflow-Debug-and-Optimization