Lab A Report--Chapter 7 Design Optimization

R10943119 蔡岳峰

Introduction

因為 2022.1 版本會自己做除了 pipeline 之外的一些優化，vitis 會自動使用一些優化將 ii 變為 1。因此無法只單純地把 pipeline 功能關掉去分析每一步優化之後的結果。若是使用 2022.1 的話會無法跑出一個合理的結果，因此我後面的步驟都是用與 tutorial 相同的 2020.1 去做的實驗。板子一樣是 xc7z007s-clg400-1。

Lab 1

1. Scratch

   一開始是什麼優化都不做的寫法。

```
void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
      // Iterate over the columns of the B matrix
      Col: for(int j = 0; j < MAT_B_COLS; j++) {
         res[i][j] = 0;
         // Do the inner product of a row of A and col of B
         Product: for(int k = 0; k < MAT_B_ROWS; k++) {
            res[i][j] += a[i][k] * b[k][j];
         }
      }
   }

}
```

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	8.702 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
79	79	1.053 us	1.053 us	79	79	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row	78	78	26	-	-	3	no
+ Col	24	24	8	-	-	3	no
++ Product	6	6	2	-	-	3	no

因為最內層的迴圈需要兩個 cycle 才能完成，考慮到 trip count=3，總共需要 6 個 cycle 完成最內圈的運算。考慮到進入 for loop 以及跳出 for loop 分別需要再

一個 cycle，因此 col iteration latency 為 8。並且 trip count 為 3，總共 24 個 cycle。一樣加上進出 for loop，row iteration latency 變為 26。因此最後需要 78 個 cycle 完成整個矩陣乘法運算。

2. Product pipeline

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	11.024 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
82	82	1.093 us	1.093 us	82	82	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	81	81	9	-	-	9	no
+ Product	6	6	2	2	1	3	yes

將最內層的 product pipeline 則會跳出以下 warning。

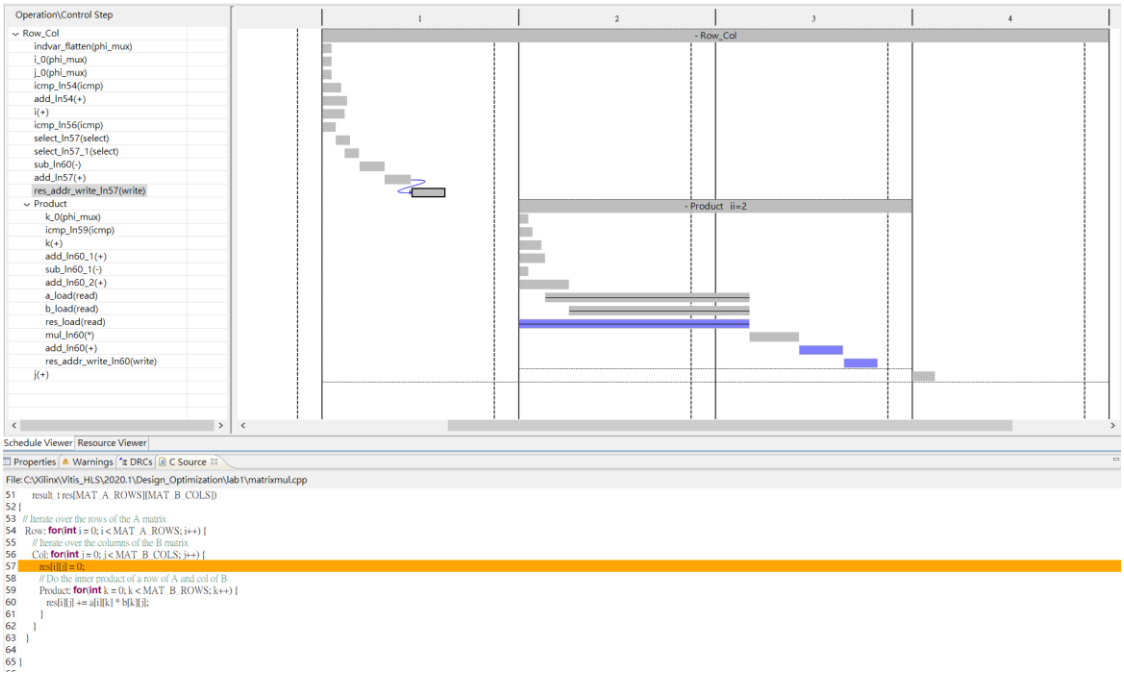
```
INFO: [SCHED 204-11] Starting scheduling ...
INFO: [SCHED 204-61] Pipelining loop 'Product'.
WARNING: [SCHED 204-68] The II Violation in module 'matrixmul' (Loop: Product): Unable to enforce a carried dependence constraint (II = 1, distance = 1, offset = 1)
   between 'store' operation ('res_addr_write_ln60', matrixmul.cpp:60) of variable 'add_ln60', matrixmul.cpp:60 on array 'res' and 'load' operation ('res_load', matrixmul.cpp:60) on array 'res'.
INFO: [SCHED 204-61] Pipelining result : Target II = 1, Final II = 2, Depth = 2.
INFO: [SCHED 204-11] Finished scheduling.
```

代表說在跑矩陣運算時，store operation 會與 load operation 產生 data dependency 的問題。造成 II 無法變成 1。

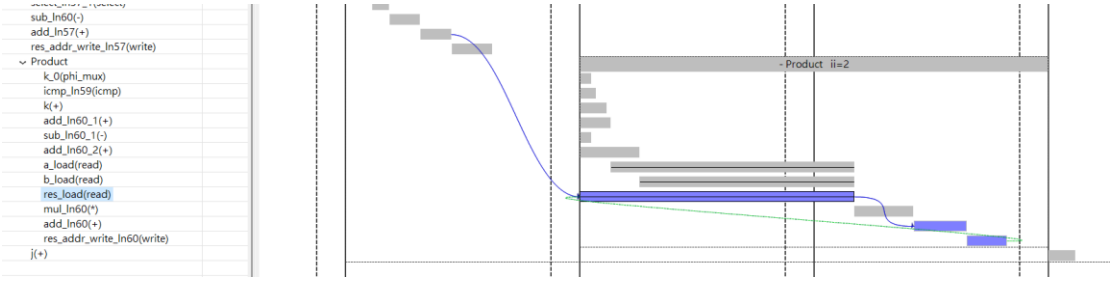
read		Write		
		read		write

(產生 conflict)

從 data 時序圖中可以看到。



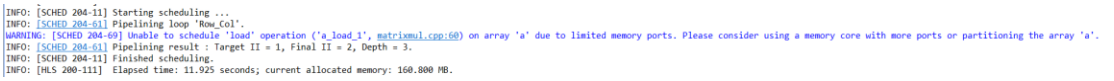
會造成無法將 for flatten 的原因是因為有 res[i][j] = 0 這行，res 是定義在 top-level function 的變數，對他寫入就等於在 RTL 上對 IO 做寫入。這個動作是無法被最佳化掉的。且有+=這個運算命令，使得需要先讀 res 後加完再寫回去。
從這張圖可以看到因為 data dependency 而使得 II 必須大於 2。



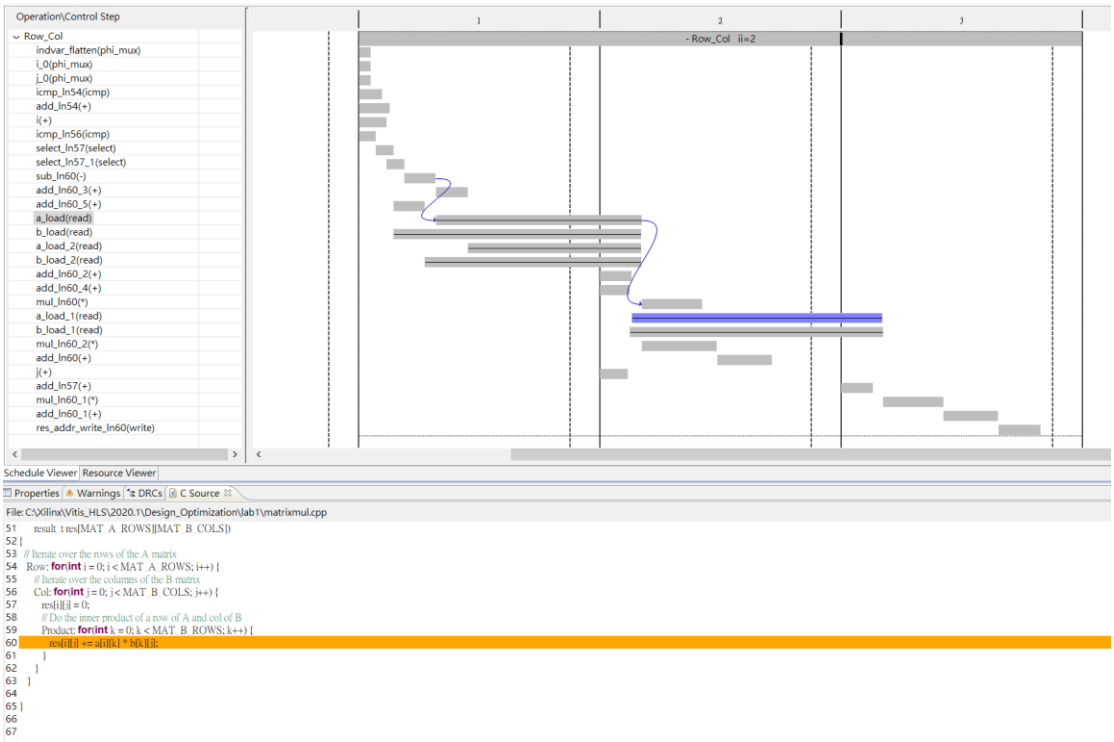
這個 data dependency 問題只能使用 pipeline 更外層或是 data reshape/partition 來解決。

3. Column Pipeline

   這邊使用了更外層 pipeline 的解法。但出現了另外一個問題: memory port 不夠。

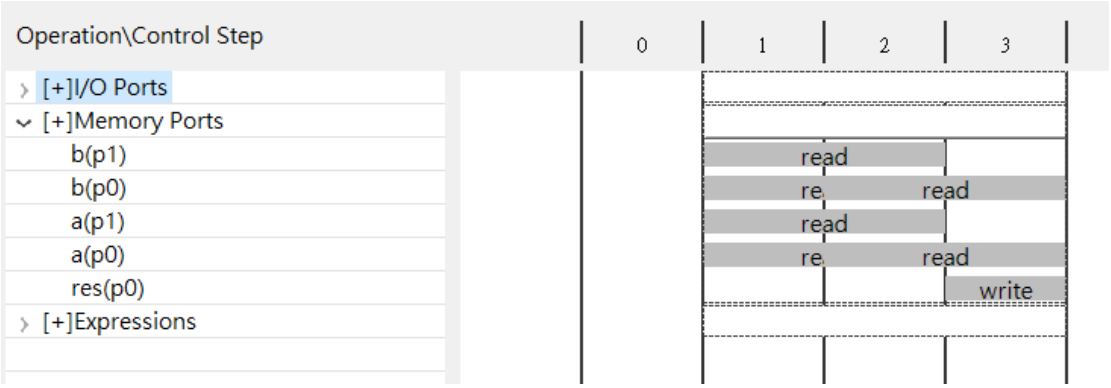


從 schedule flow 中可以看到，在不同的 stage 分別都有 read array a，總共需要 read 三次。必須要在連續的三個 cycle 做出讀取，pipeline 後則是每個 cycle 都有 3 個 read。



但是 BRAM 是 daul-port block RAM，一個 cycle 只能讀寫兩個值。因此無法同時

做出三個 Read operation。



從 memory port usage 中可以看到若是照著做 pipeline 則會發生 memory resource 不夠的問題，也就是 structural hazard。
因此最後 II 還是只能為 2。

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	11.024 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
20	20	0.267 us	0.267 us	20	20	none

Detail

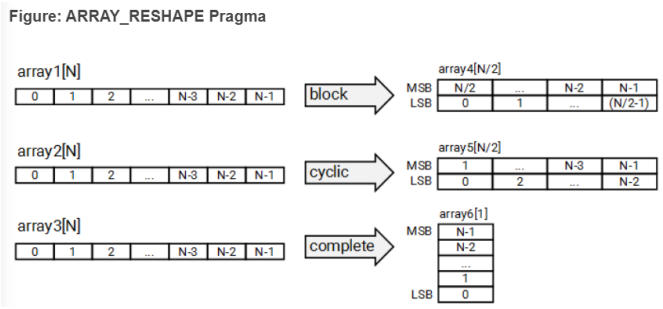
Instance

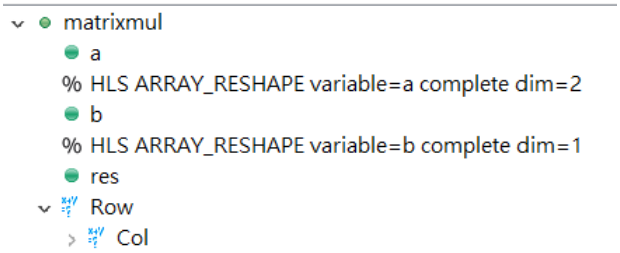
Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	18	18	3	2	1	9	yes

4. Reshape array

   另一種方法是使用 Array reshape 的技巧。我們要在 array a/b 中給定他們 reshape 的方式，而因為矩陣乘法的方向的關係，兩個矩陣的 dimension 也會不同。Block 會將矩陣分割為連續記憶體，按照輸入順序與維度進行分切。而 cycle 會以循環的方式將原本的順序打亂，依照分切的記憶體。而 complete 會將記憶體全部切成獨立各自的小塊，最消耗資源，但也最容易解決 data dependency 的問題。






```
############################################################
set_directive_pipeline "matrixmul/Col"
set_directive_array_reshape -type complete -dim 2 "matrixmul" a
set_directive_array_reshape -type complete -dim 1 "matrixmul" b
```

可以達到 II 等於 1。

Row_col loop 需要 3 個 cycle (iteration latency)。而因為 II 為 1，可以每個 cycle 不斷的讀值。總共需要 9+3=12 個 cycle 完成所有運算。

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	9.512 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
12	12	0.160 us	0.160 us	12	12	none

Detail

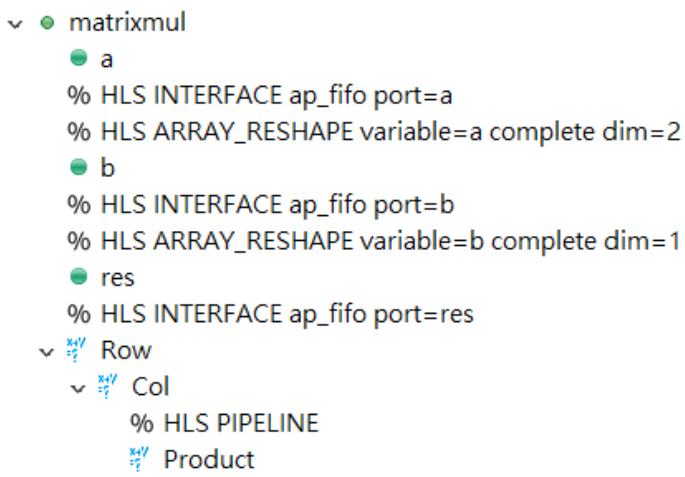
Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	10	10	3	1	1	9	yes

5. Apply FIFO interface

   這邊想做的是改變 RAM 的 interface 來做 data streaming。給出這些 directives。




```
void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
      // Iterate over the columns of the B matrix
      Col: for(int j = 0; j < MAT_B_COLS; j++) {
         res[i][j] = 0;
         // Do the inner product of a row of A and col of B
         Product: for(int k = 0; k < MAT_B_ROWS; k++) {
            res[i][j] += a[i][k] * b[k][j];
         }
      }
   }

}
```

可以看到 res 會一直不斷地在同一個 address 重複做寫入。因此不符合 streaming access 的特性，而是較偏向 random access。因此無法直接使用 FIFO interface，會跳出 warning。

```
INFO: [HLS 200-10] Checking synthesizability ...
ERROR: [SYNCHK 200-91] Port 'res' (matrixmul.cpp:48) of function 'matrixmul' cannot be set to a FIFO
ERROR: [SYNCHK 200-91] as it has both write (matrixmul.cpp:60:13) and read (matrixmul.cpp:60:13) operations.
INFO: [SYNCHK 200-10] 1 error(s), 0 warning(s).
ERROR: [HLS 200-70] Synthesizability check failed.
command 'ap_source' returned error code
    while executing
"source C:/Xilinx/Vitis_HLS/2020.1/Design_Optimization/lab1/matrixmul_prj/solution4_FIFO_int/csynth.tcl"
    invoked from within
"hls::main C:/Xilinx/Vitis_HLS/2020.1/Design_Optimization/lab1/matrixmul_prj/solution4_FIFO_int/csynth.tcl"
    ("uplevel" body line 1)
    invoked from within
"uplevel 1 hls::main {*}$args"
    (procedure "hls_proc" line 5)
    invoked from within
"hls_proc $argv"
Finished C synthesis.
```

6. Pipeline the function

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	10.210 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
6	6	79.998 ns	79.998 ns	5	5	function

Detail

Instance

Loop

在這裡，pipeline function 比起 pipeline loop 是為一個更好的方案。

All Compared Solutions

solution3_reshape_array: xc7z007sclg400-1

solution5_pipeline_func: xc7z007sclg400-1

Performance Estimates

Timing

Clock		solution3_reshape_array	solution5_pipeline_func
ap_clk	Target	13.33 ns	13.33 ns
	Estimated	9.512 ns	10.210 ns

Latency

		solution3_reshape_array	solution5_pipeline_func
Latency (cycles)	min	12	6
	max	12	6
Latency (absolute)	min	0.160 us	79.998 ns
	max	0.160 us	79.998 ns
Interval (cycles)	min	12	5
	max	12	5

Utilization Estimates

	solution3_reshape_array	solution5_pipeline_func
BRAM_18K	0	0
DSP48E	2	18
FF	56	487
LUT	190	574
URAM	0	0

可以發現 pipeline function 之後的 latency 可以比起 array reshape 還要更短。但是 area 使用就會上升，這是因為有同時做了 loop unrolling。Function pipeline 之後可以做到每 5 個 cycle process 9 samples。Function pipeline 可以有最好的 performance。

```
INFO: [XFORM 203-502] Unrolling all loops for pipelining in function 'matrixmul' (matrixmul.cpp:49).
INFO: [HLS 200-489] Unrolling loop 'Row' (matrixmul.cpp:54) in function 'matrixmul' completely with a factor of 3.
INFO: [HLS 200-489] Unrolling loop 'Col' (matrixmul.cpp:56) in function 'matrixmul' completely with a factor of 3.
INFO: [HLS 200-489] Unrolling loop 'Product' (matrixmul.cpp:59) in function 'matrixmul' completely with a factor of 3.
INFO: [XFORM 203-131] Reshaping array 'a' (matrixmul.cpp:49) in dimension 2 completely.
INFO: [XFORM 203-131] Reshaping array 'b' (matrixmul.cpp:50) in dimension 1 completely.
```

目前的 c code 寫法會讓 HLS 無法達到 streaming interface。原因如下:

1) Streaming 時 access 的順序必須是連續的

2) 每次的 Access 都是 port access，讓 HLS 無法針對這地方去做優化。

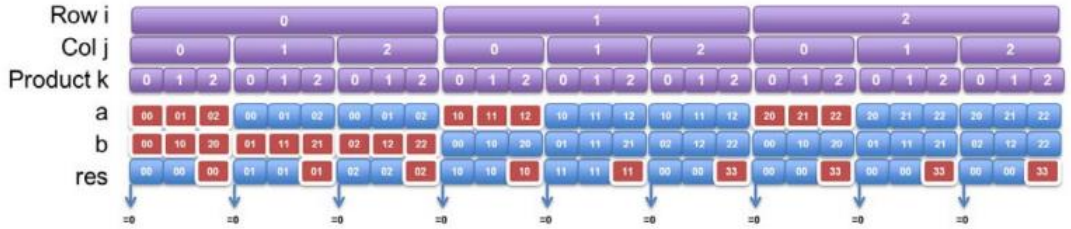


Figure 7-20: **Matrix Multiplier Address Accesses**

7-20 代表了 IO access pattern for read A, B, res。紅色代表 port access。在 read port 來說，資料必須 cached 在內部避免 re-read；write port 來說，資料必須存在暫存變數中。
原版的 code 並沒有這樣的編排。

Lab2

本題將使用 FIFO 作為 input/output。輸入輸出都需要遵照 FIFO 之規則，不可進行隨機存取。本 code 針對資料讀取有較好的優化，B 則一開始就讀取完畢並存入暫存器中。避免重複從 port 讀取，浪費資源與時間。

```
void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
#pragma HLS ARRAY_RESHAPE variable=b complete dim=1
#pragma HLS ARRAY_RESHAPE variable=a complete dim=2
#pragma HLS INTERFACE ap_fifo port=a
#pragma HLS INTERFACE ap_fifo port=b
#pragma HLS INTERFACE ap_fifo port=res
  mat_a_t a_row[MAT_A_ROWS];
  mat_b_t b_copy[MAT_B_ROWS][MAT_B_COLS];
  int tmp = 0;

  // Iterate over the rowa of the A matrix
  Row: for(int i = 0; i < MAT_A_ROWS; i++) {
    // Iterate over the columns of the B matrix
    Col: for(int j = 0; j < MAT_B_COLS; j++) {
#pragma HLS PIPELINE rewind
      // Do the inner product of a row of A and col of B
      tmp=0;
      // Cache each row (so it's only read once per function)
      if (j == 0)
        Cache_Row: for(int k = 0; k < MAT_A_ROWS; k++)
          a_row[k] = a[i][k];

       // Cache all cols (so they are only read once per function)
     if (i == 0)
           Cache_Col: for(int k = 0; k < MAT_B_ROWS; k++)
              b_copy[k][j] = b[k][j];

     Product: for(int k = 0; k < MAT_B_ROWS; k++) {
       tmp += a_row[k] * b_copy[k][j];
     }
     res[i][j] = tmp;
    }
  }
}
```

Store data in register

Register renaming

後面則使用了 register renaming 的技巧，避免產生 data dependency，解決 data hazard，達到更高的 pipeline throughput。

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	10.014 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
10	11	0.133 us	0.147 us	9	9	loop rewind(delay=0 initiation interval(s))

Detail

Instance

Loop

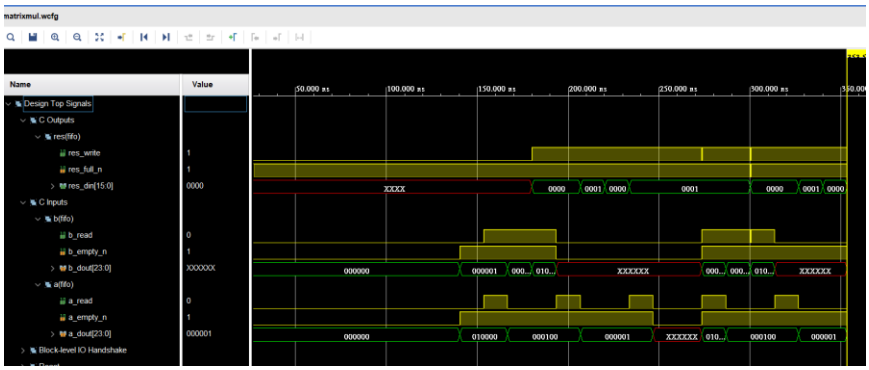
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	10	10	3	1	1	9	yes

Utilization Estimates

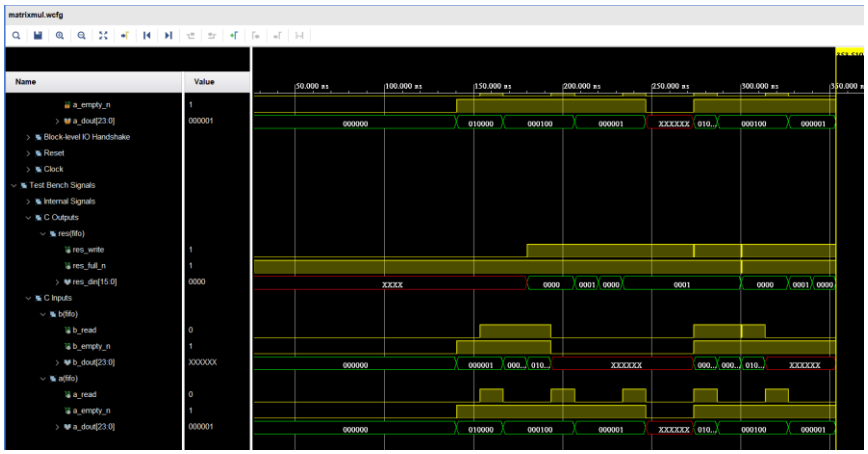
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	249	-
FIFO	-	-	-	-	-
Instance	-	-	0	45	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	180	-
Register	-	-	159	-	-
Total	0	2	159	474	0
Available	100	66	28800	14400	0
Utilization (%)	0	3	~0	3	0

Design 之波型圖



Testbench 之波型圖



從波型圖中可以確認與測資提供的數值相同。
總體不同方法之間的比較，可以發現 reshape array 可以達到比 column pipeline 更好的速度。但還是比不上 loop unroll 的 pipeline function 的結果。相對的 pipeline function 付出的是硬體面積的代價。
就 FIFO 來說，因為可以達到更好的 throughput，data streaming 可以做得更好，可以比起 reshape 再更加進步。且硬體使用率不會跟 pipeline function 比起來不會使用那麼多。

Performance Estimates

Timing

Clock		solution1_scratch	solution2_product_pipe	solution3_col_pipe	solution4_reshape_array	solution6_pipeline_func
ap_clk	Target	13.33 ns	13.33 ns	13.33 ns	13.33 ns	13.33 ns
	Estimated	8.702 ns	11.024 ns	11.024 ns	9.512 ns	10.210 ns

Latency

		solution1_scratch	solution2_product_pipe	solution3_col_pipe	solution4_reshape_array	solution6_pipeline_func
Latency (cycles)	min	79	82	20	12	6
	max	79	82	20	12	6
Latency (absolute)	min	1.053 us	1.093 us	0.267 us	0.160 us	79.998 ns
	max	1.053 us	1.093 us	0.267 us	0.160 us	79.998 ns
Interval (cycles)	min	79	82	20	12	5
	max	79	82	20	12	5

Utilization Estimates

	solution1_scratch	solution2_product_pipe	solution3_col_pipe	solution4_reshape_array	solution6_pipeline_func
BRAM_18K	0	0	0	0	0
DSP48E	1	1	2	2	18
FF	44	36	46	56	487
LUT	184	232	302	190	574
URAM	0	0	0	0	0

GitHub link:
https://github.com/yuehfeng1114/2022HLS-LAB-A-DesignOptimization.git