AAHLS LabA Report

D10943018 戴妤珊

-Briefly introduction to the algorithm or overall system

The sample design used in the lab exercise is a **matrix multiplier function**, which contains two 3x3 input arrays, a and b, and 3x3 output array res. The design goal is to process a new sample every clock period (II=1) and implement the interface as streaming data interfaces. In **lab1**, the analysis includes a comparison of methods that optimizes at the **loop** level and ones optimizes at the **function** level. In b, code optimization is introduced to facilitate **streaming** design.

-Explain the original code/system/pragmas and how you implemented it

The source code for the matrix multiplier is shown below. The goal of this code is to implement matrix multiplication of the two 3x3 input arrays, a and b. The matrix multiplication process is done by 3 loops, Row, Col, and Product. The final results are saved in a 3x3 array, res.

```
48  void matrixmul(
49          mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
50          mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
51          result_t res[MAT_A_ROWS][MAT_B_COLS])
52  {
53    // Iterate over the rows of the A matrix
54    Row: for(int i = 0; i < MAT_A_ROWS; i++) {
55       // Iterate over the columns of the B matrix
56       Col: for(int j = 0; j < MAT_B_COLS; j++) {
57          res[i][j] = 0;
58          // Do the inner product of a row of A and col of B
59          Product: for(int k = 0; k < MAT_B_ROWS; k++) {
60             res[i][j] += a[i][k] * b[k][j];
61          }
62       }
63    }
64
65  }
```

-If possible, share how you optimize the design and the trade-off you make

Lab1

There are 6 solutions implemented in lab1, and I will elaborate them in the following contexts.

1. **Solution1** is the original version, which contains no optimization. In each iteration of the Product loop, it takes 5 cycles, which are composed of load(1)+mul(2)+add(1)+branch(1). Since there are 3 iterations, the total latency of Product is 15. For each iteration of the Col loop, there are 17 cycles, which are branch(1)+15(Product)+branch(1). Since there are 3 iterations, the total latency of Col is 51. For each iteration of the Row loop, there are 53 cycles, which are

branch(1)+51(Row)+branch(1). Since there are 3 iterations, the total latency of Col is 159.

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	1.818 ns	3.60 ns

Latency

Summary

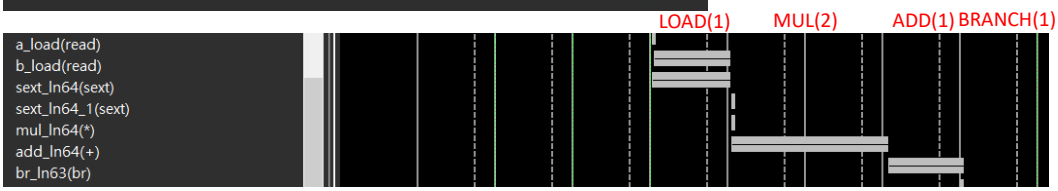
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
160	160	2.133 us	2.133 us	161	161	no

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row	159	159	53	-	-	3	no
+ Col	51	51	17	-	-	3	no
++ Product	15	15	5	-	-	3	no



Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	101	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	85	-
Register	-	-	46	-	-
Total	0	1	46	186	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	0	~0	~0	~0	0
Utilization SLR (%)	0	~0	~0	~0	0

However, due to the tool version difference, the arithmetic operations (mul + add) are done in one cycle in the tutorial version. Thus, the resulted latency is different from my implementation.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
79	79	79	79	none

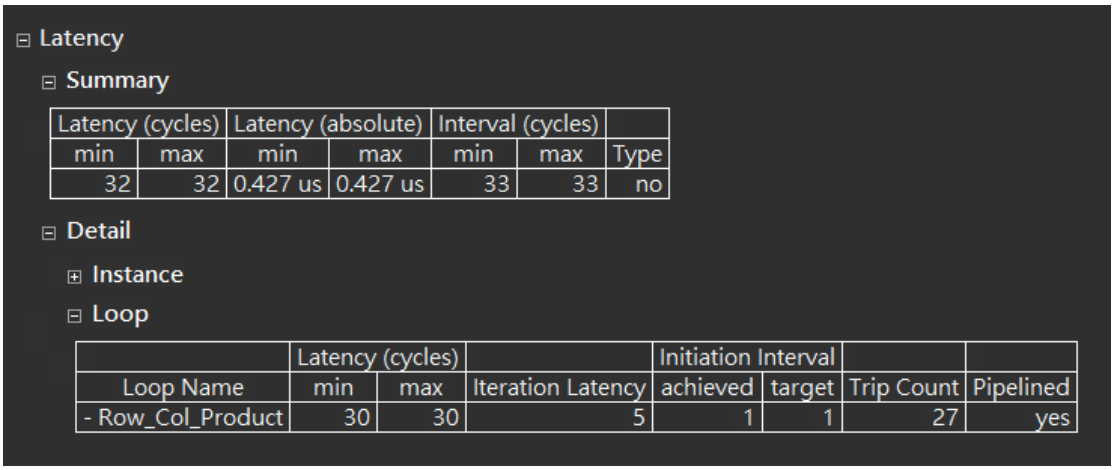
Detail

Instance

Loop

	Latency			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

2. **Solution2** adds **pipeline in the Product loop**. This synthesis report shows the **top-level loop is not pipelined** since loop flattening only occurred on loop **Row_Col**. Due to the code sets res to 0 (line 57) before the Product loop, the Product loop can't be flattened into the Row_Col loop. The warning messages show there is **II violation (II=2)** due to carried dependency, which occurred if two operations in different iterations of the same loop conflict. Specifically, there is a += operator in line 60, which needs loading and writing at the same address of res. This problem needs **code re-writing** and would be optimized in lab2.
   However, since each iteration of Product takes 5 cycles not 1 as in tutorial, the cycle of writing 0 to res is not followed by writing the result to res. I don't encounter the II violation mentioned-above.



Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
32	32	0.427 us	0.427 us	33	33	no

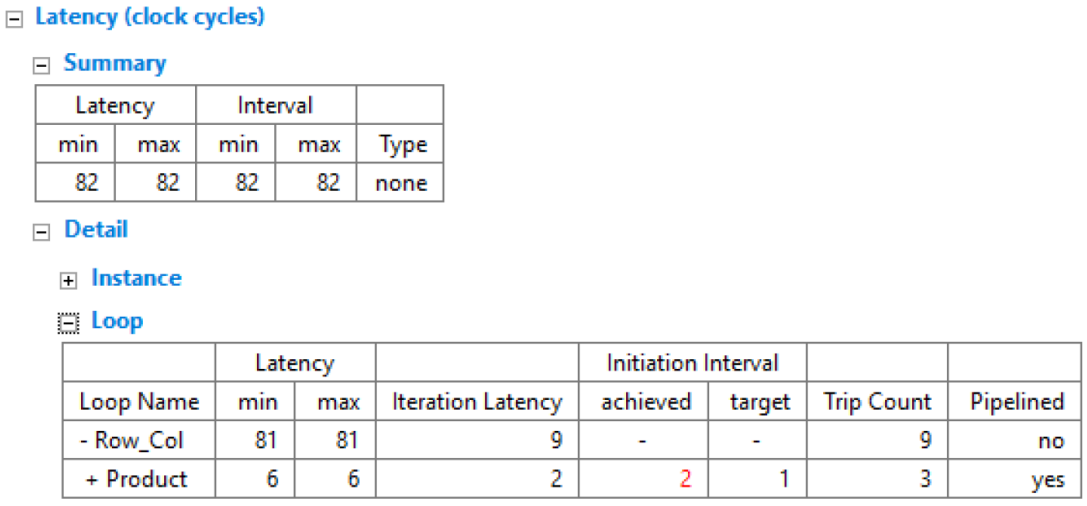
Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col_Product	30	30	5	1	1	27	yes

The results of tutorial are shown below, which show II violation.



Latency (clock cycles)

Summary

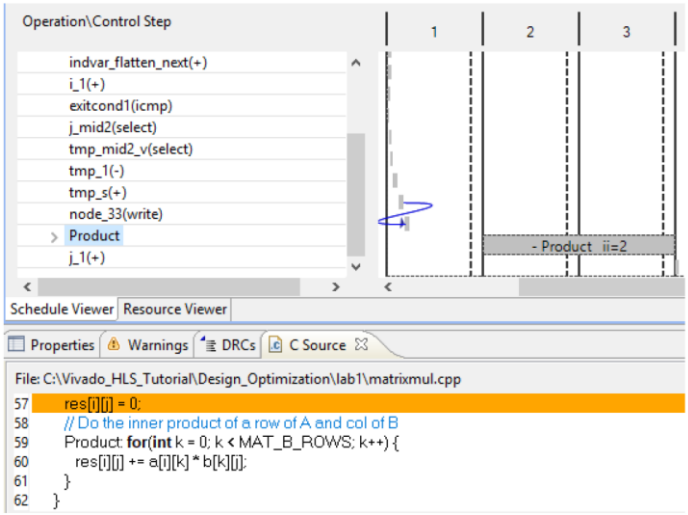
Latency		Interval		
min	max	min	max	Type
82	82	82	82	none

Detail

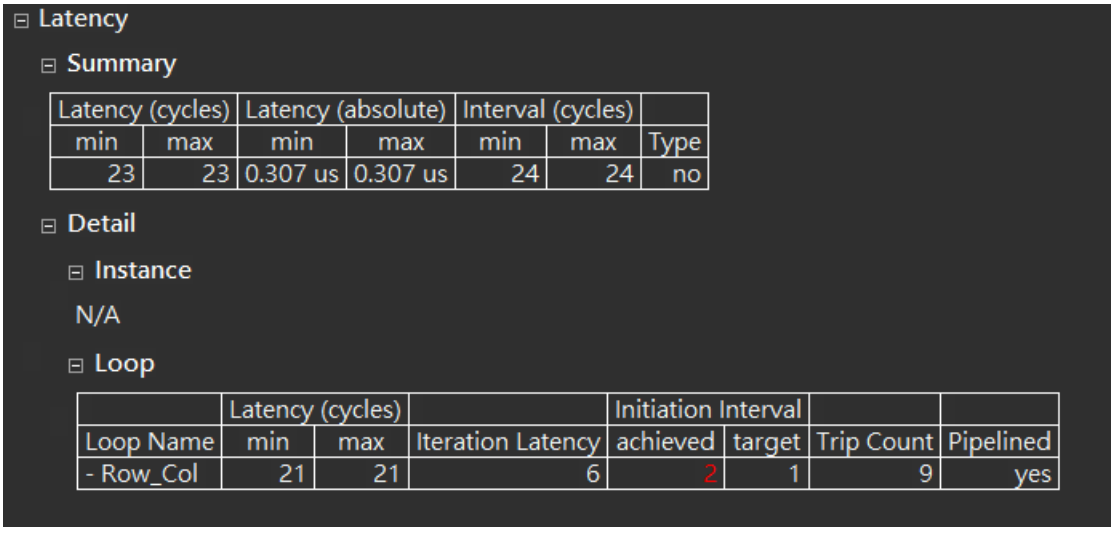
Instance

Loop

	Latency			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



3. **Solution3** adds **pipeline in the Col loop**. The warning messages are *WARNING: [HLS 200-885] The II Violation in module 'matrixmul' (loop 'Row_Col'): Unable to schedule 'load' operation ('a_load_1', matrixmul.cpp:58) on array 'a' due to **limited memory ports (II = 1)**. Please consider using a memory core with more ports or partitioning the array 'a'.* For dual-port block RAM, it can only have a maximum of two ports, and therefore fails to read all three values in one cycle. To solve this issue, array reshaping is introduced in Solution4.



Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
23	23	0.307 us	0.307 us	24	24	no

Detail

Instance

N/A

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	21	21	6	2	1	9	yes

4. **Solution4** applies **array reshaping**. Since the loop index of the Product loop is k, both arrays should be partitioned along their respective k dimensions, which is 2 for a and 1 for b. This design reaches **II=1**.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
15	15	0.200 us	0.200 us	16	16	no

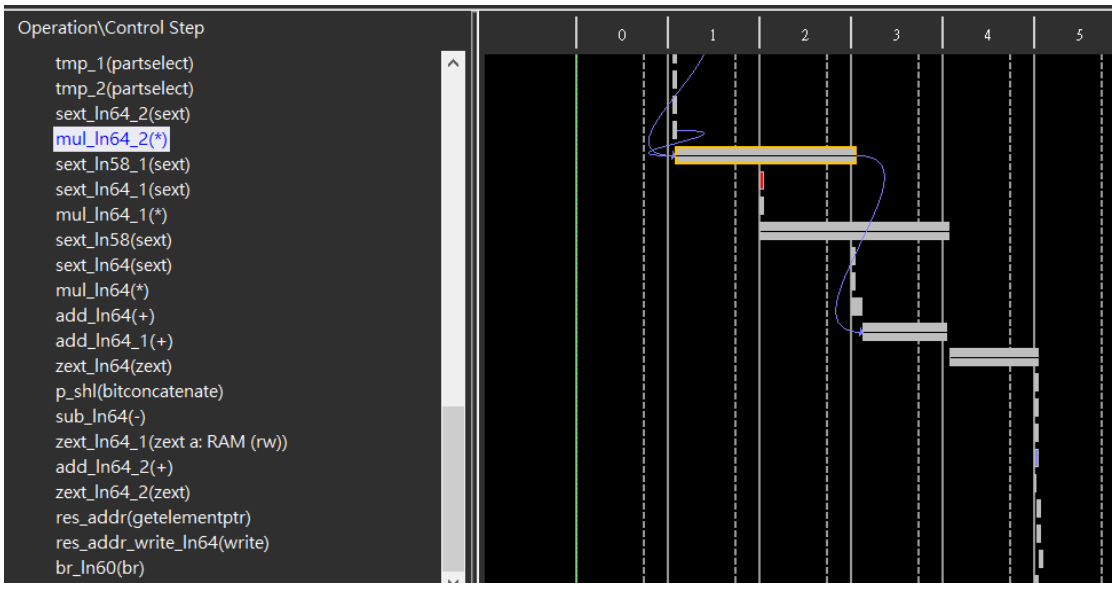
Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	13	13	6	1	1	9	yes

We con observe there are two multiplication operators run parallelly from scheduler viewer.



The results of tutorial are shown below.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
11	11	11	11	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	9	9	2	1	1	9	yes

5. **Solution5** applies **FIFO interface**. The warning messages are *WARNING: [HLS 214-142] Implementing stream: may cause mismatch if read and write accesses are not*

in sequential order on port 'res' (matrixmul.cpp:52:0) Since line 58 and line 60 results in four **consecutive writes** to the same address, the design doesn't constitute a streaming access pattern. This problem needs **code re-writing** and would be optimized in lab2.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
37	37	0.493 us	0.493 us	38	38	no

Detail

Instance

		Latency (cycles)		Latency (absolute)		Interval (cycles)		
Instance	Module	min	max	min	max	min	max	Type
grp_matrixmul_Pipeline_Col_fu_64	matrixmul_Pipeline_Col	9	9	0.120 us	0.120 us	9	9	no

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row	36	36	12	-	-	3	no

6. **Solution6** applies **pipeline on the Function**. This design leads to fewer clocks (9 sample every 5 cycles) than prior ones, however, the **area** and **resource** have **increased significantly** due to **all loops were unrolled**. Since the design goal is II=1, we don't need to exceed the requirement and thus pipeline on loop is preferred.
   The comparison of the performance estimation of solution4 and solution6 is shown below.

Performance Estimates

Timing

Clock		solution4	solution6
ap_clk	Target	13.33 ns	13.33 ns
	Estimated	2.521 ns	2.521 ns

Latency

		solution4	solution6
Latency (cycles)	min	15	10
	max	15	10
Latency (absolute)	min	0.200 us	0.133 us
	max	0.200 us	0.133 us
Interval (cycles)	min	16	5
	max	16	5

Utilization Estimates

	solution4	solution6
BRAM_18K	0	0
DSP	2	18
FF	201	632
LUT	269	553
URAM	0	0

The results of tutorial are shown below.

compare reports

Performance Estimates

Timing (ns)

Clock		solution4	solution6
ap_clk	Target	13.33	13.33
	Estimated	7.566	7.566

Latency (clock cycles)

		solution4	solution6
Latency	min	11	5
	max	11	5
Interval	min	11	5
	max	11	5

Utilization Estimates

	solution4	solution6
BRAM_18K	0	0
DSP48E	2	18
FF	18	343
LUT	187	565
URAM	0	0

Lab2

In Lab2, the source code is modified as follows:

```
46  #include "matrixmul.h"
47
48  void matrixmul(
49          mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
50          mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
51          result_t res[MAT_A_ROWS][MAT_B_COLS])
52  {
53  #pragma HLS ARRAY_RESHAPE variable=b complete dim=1
54  #pragma HLS ARRAY_RESHAPE variable=a complete dim=2
55  #pragma HLS INTERFACE ap_fifo port=a
56  #pragma HLS INTERFACE ap_fifo port=b
57  #pragma HLS INTERFACE ap_fifo port=res
58    mat_a_t a_row[MAT_A_ROWS];
59    mat_b_t b_copy[MAT_B_ROWS][MAT_B_COLS];
60    int tmp = 0;
61
62    // Iterate over the rowa of the A matrix
63    Row: for(int i = 0; i < MAT_A_ROWS; i++) {
64      // Iterate over the columns of the B matrix
65      Col: for(int j = 0; j < MAT_B_COLS; j++) {
66  #pragma HLS PIPELINE rewind
67        // Do the inner product of a row of A and col of B
68        tmp=0;
69        // Cache each row (so it's only read once per function)
70        if (j == 0)
71          Cache_Row: for(int k = 0; k < MAT_A_ROWS; k++)
72            a_row[k] = a[i][k];
73
74         // Cache all cols (so they are only read once per function)
75       if (i == 0)
76                Cache_Col: for(int k = 0; k < MAT_B_ROWS; k++)
77                  b_copy[k][j] = b[k][j];
78
79        Product: for(int k = 0; k < MAT_B_ROWS; k++) {
80          tmp += a_row[k] * b_copy[k][j];
81        }
82        res[i][j] = tmp;
83      }
84    }
85  }
```

The directives of Lab1 are included in the code, which specified as pragmas. There are some differences from the code in Lab1.

1. Introduce for-loop to **cache** the row (line 71) and column reads (line 76). Since **pipeline** pragma is added in the Col for-loop (line 66), the cache for-loops are unrolled to avoid multiple cycles.
2. A temporary variable **tmp** is used for the **accumulation** (line 80) and its values are assigned to **res** when the accumulation **completes**.

After the above modification, the optimized design can support streaming FIFO interfaces. The whole IO process is:

(1) Read a[0][0], a[0][1], a[0][2]
(2) Read b[0][0], b[1][0], b[2][0]
(3) Write res[0][0]
(4) Read b[0][1], b[1][1], b[2][1]
(5) Write res[0][1]
(6) Read b[0][2], b[1][2], b[2][2]
(7) Write res[0][2]
(8) Read a[1][0], a[1][1], a[1][2]
(9) Write res[1][0]
(10) Write res[1][1]
(11) Write res[1][2]
(12) Read a[2][0], a[2][1], a[2][2]
(13) Write res[2][0]
(14) Write res[2][1]
(15) Write res[2][2]

In the above process, each address is only read/write once.

The following figures show the latency and utilization estimation. The latency is slightly lower than the solution4 in lab1, however, the resource requirements are more. Since stream design needs additional caches, it costs more FF(399) and LUT(453).

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
14	15	0.187 us	0.200 us	9	9	loop rewind stp(delay=0 clock cycles(s))

⊟ Detail

⊟ Instance

N/A

⊟ Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	14	14	7	1	1	9	yes

Utilization Estimates

Summary

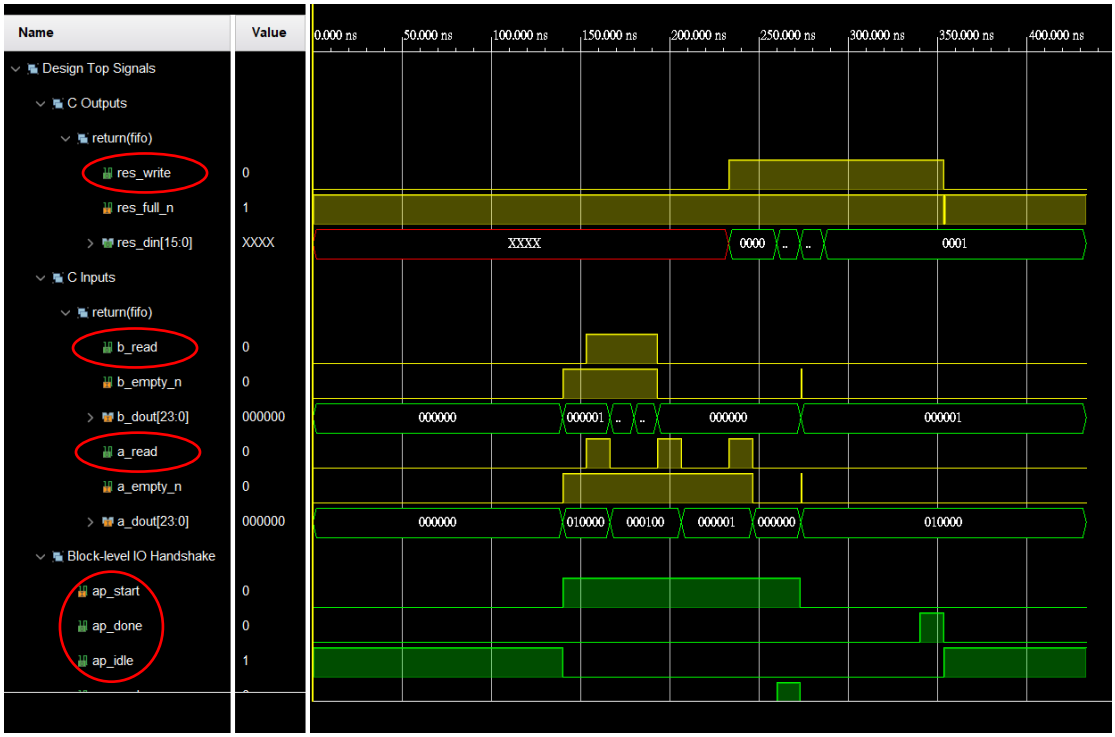
Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	-	0	85	-
FIFO	-	-	-	-	-
Instance	-	0	0	40	-
Memory	0	-	48	3	0
Multiplexer	-	-	-	197	-
Register	-	-	351	128	-
Total	0	2	399	453	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	0	~0	~0	~0	0
Utilization SLR (%)	0	~0	~0	~0	0

After co-simulation, the performance estimation is shown below.

Performance Estimates

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
matrixmul				15	15	15
Row_Col				16	16	16

From the wave figure, we can observe the stream pattern as mentioned previously, each value read or write only once. We can also notice the handshake mechanism, the system run idle -> start -> done -> idle.



-Analyze the timing/performance/utilization

The detailed analysis is included in the last problem, and in this part, I just summarize all the optimization methods done in this lab. In original version, the latency is 160 cycles with 1 DSP, 46 FF, and 186 LUT. Among different optimization methods, the solution4 with array reshape in lab1 strike a balance between performance and resource costs, whose latency is 16 cycles with 2 DSP, 201 FF, 269 LUT. Since the limited bandwidth is the bottleneck of the original design, this modification can effectively improve the latency. Although function pipeline design in solution6 achieve lower latency than solution4, it exceeds the requirement of II=1 and introduce huge resource requirements due to whole loop unrolling.

-Explain what you observed and learned

Lab1

1. I learned how to calculate the function latency, and the calculation details are specified in the precious contexts.
2. Observe the synthesis log files, we can observe there are INFO messages and sometimes some warning messages. INFO message demonstrates the what synthesis do, such as pipeline, unrolling. As for warning, it displays violations and their possible causes.
3. Dual-port block RAM can only have a maximum of two ports, so we need ARRAY_RESHAPE if the number of iterations of a loop is larger than 2. By array reshaping, array is partitioned into k arrays, where is the loop index for the loop writing output values. Thus, the access to array can be modified.
4. If the load/write operations in the source code are not a streaming access pattern (random access), then it can't use FIFO interfaces and needs code re-writing.
5. Pipeline at function level results in lower latency and higher throughput, but at the sacrifice of huge resource requirement due to whole loop unrolling. For the trade-off between latency and resource usage, II=1 is enough good and thus pipeline at loop is preferred.

Lab2

1. In a stream/FIFO interface, the values must be accessed in sequential order.
2. Observe the Figure 7.20, we can notice in the original design there are multiple accesses to the same address (blue blocks). However, to build a streaming pattern, the port accesses can only be set as in the red blocks, each address is only read/write once.

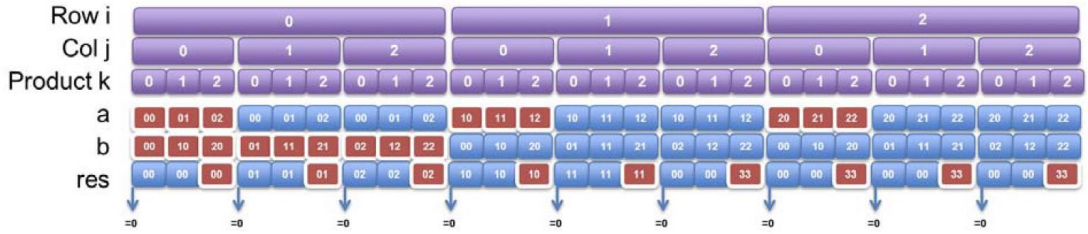


Figure 7-20: **Matrix Multiplier Address Accesses**

3. To achieve the streaming pattern, cache buffer is introduced. As shown in the Figure 7.20, the values appeared in blue blocks should be cached for later use. Therefore, for array a, a cache array of size 3 is used. As for array b, it needs a cache array of size 9. For res, it needs a buffer of size 1 to save the temporary results and receives the final results after all accumulation of this address is done.
4. I learn how to compare the latency and resource usage to evaluate a design. In original version, the latency is 160 cycles with 1 DSP, 46 FF, and 186 LUT. Among different optimization methods, the solution4 in lab1 strike a balance between performance and resource costs, whose latency is 16 cycles with 2 DSP, 201 FF, 269 LUT. Although function pipeline design in solution6 achieve lower latency than solution4, it exceeds the requirement of II=1 and introduce huge resource requirement due to whole loop unrolling.

-Explain what problem you encountered and how you solved it

1. In Vitis HLS 2022.1, simple loops and inner loops (for nested loops) are automatically pipelined by the tool. Therefore, I need to turn off the optimization manually to get the initial version (without optimization).

-Attach the GitHub link in your report

https://github.com/Shan-handsome/2022_AAHLS_Design_Optimization