Application Acceleration with High-Level-Synthesis

UG871 Chapter 7: Design Optimization

Speaker: F08943009 張承洋

Advisor: 賴瑾 教授

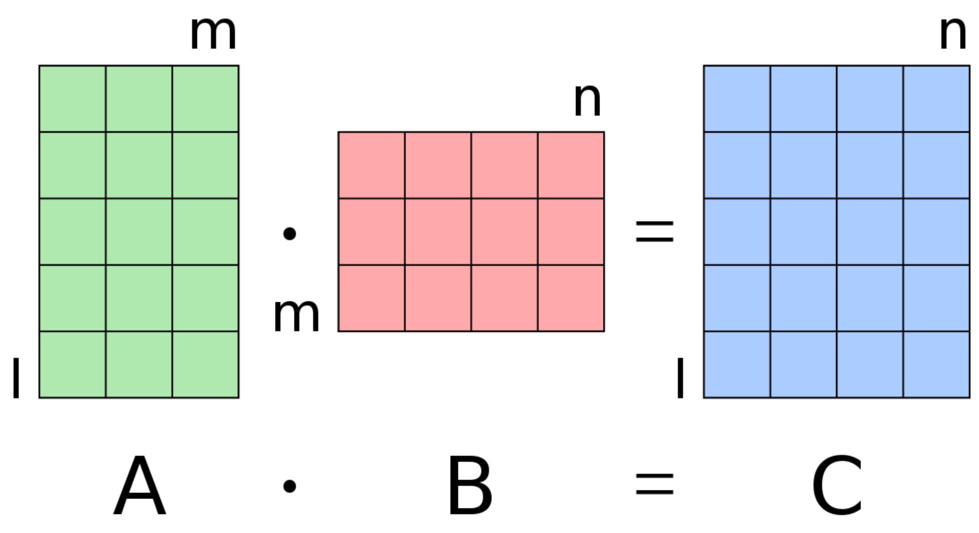
Date: 2022/10/20

Outline

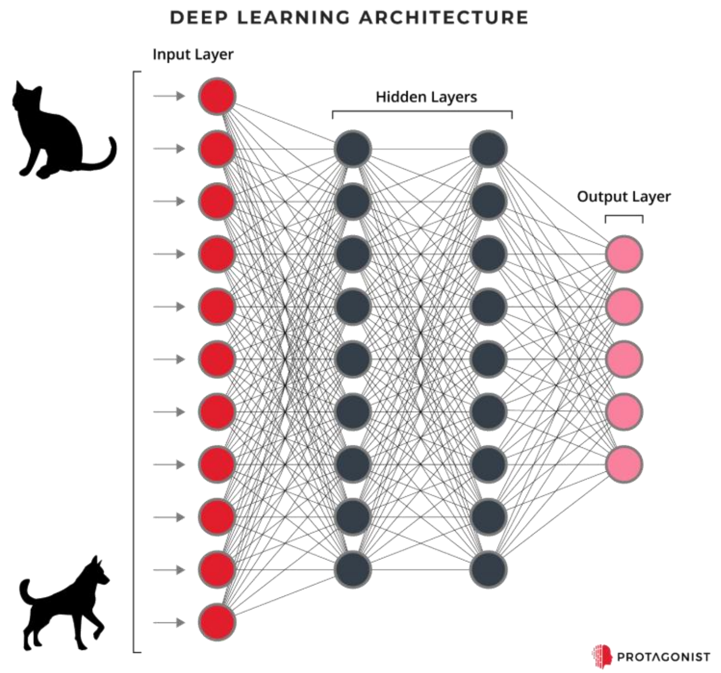
- Background of Matrix Multiplier
- Lab 1: Optimizing a Matrix Multiplier
  - @ Case: No pragma
  - @ Case: Pipelining Product Loop
  - @ Case: Pipelining Col Loop
  - @ Case: Pipelining Col Loop and RESHAPE
  - @ Case: Pipelining the Function
  - Cannot Apply FIFO
- Lab 2: C-Code Optimized for I/O Accesses
  - @ Case: FIFO Interface with Pipeline Rewind
- Questions

Background of Matrix Multiplier

- **Matrix multiplication** is a canonical example in parallel computing
  - Applied to linear algebra, statistics, machine learning, etc.
  - The computational pattern is amenable to <u>hardware acceleration</u> with **parallelization**

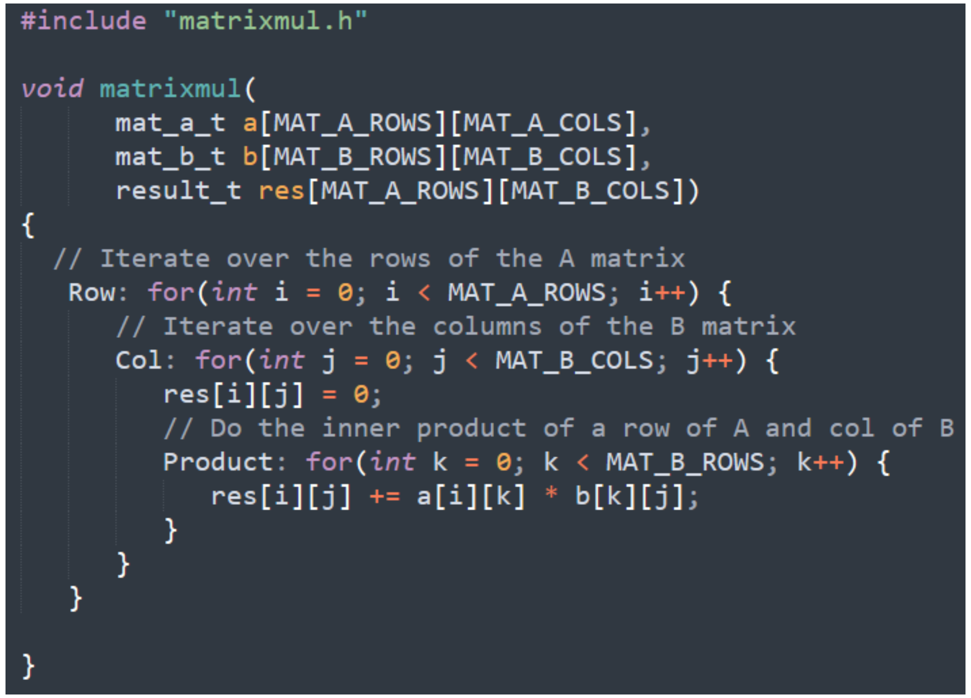


(l=m=n=3 in this lab)

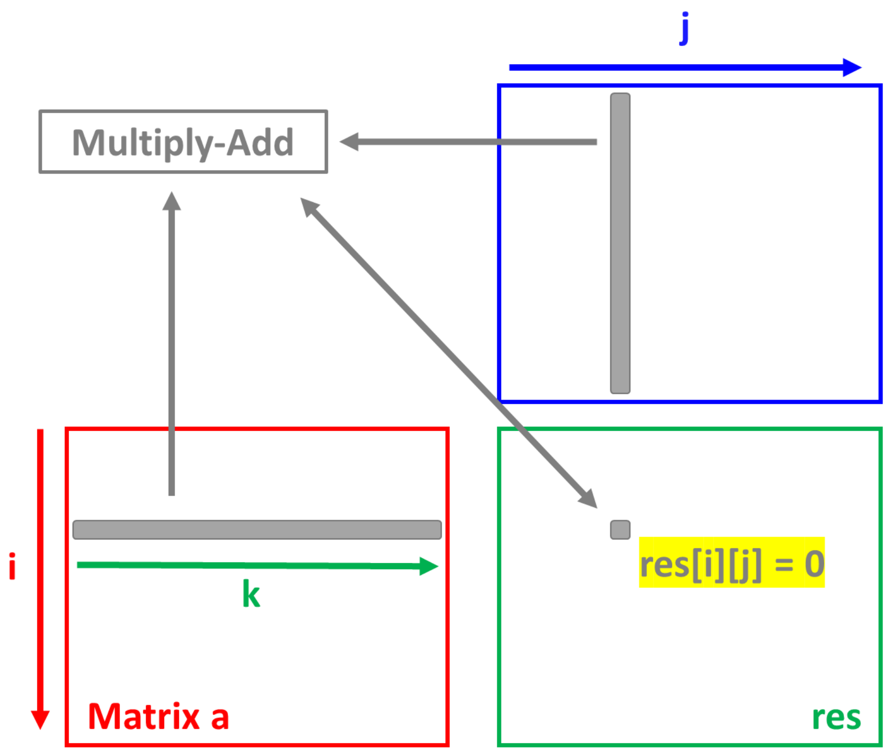


Lab1: Code Snippets

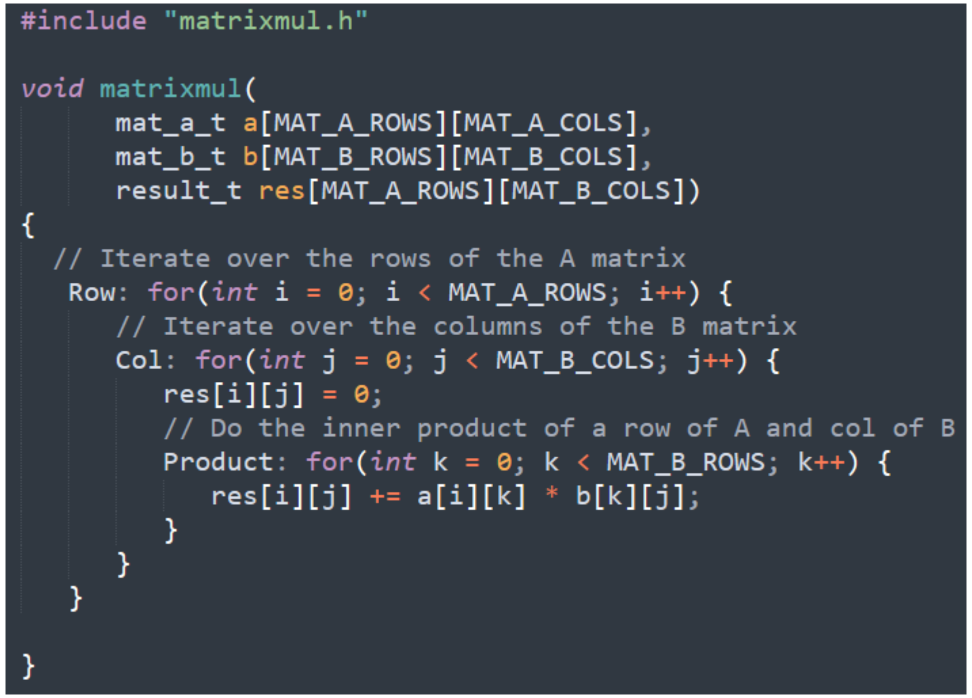
- The baseline method uses three loops (Row, Col, Product)
  - Before each **Product Loop**, the associated **res[i][j] is set to zero**



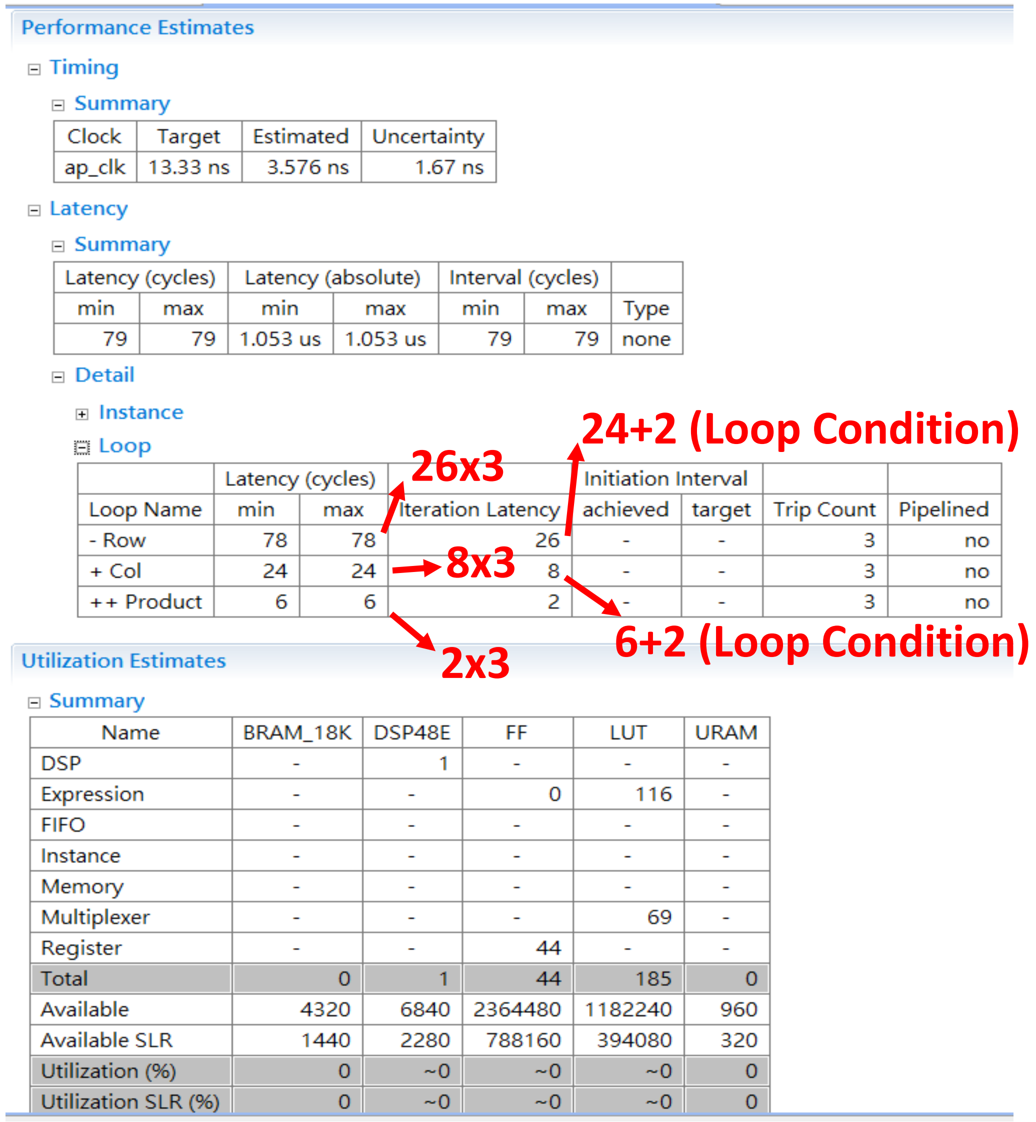
```c
#include "matrixmul.h"

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];
         }
      }
   }

}
```



@ Case: No pragma



```
#include "matrixmul.h"

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	3.576 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

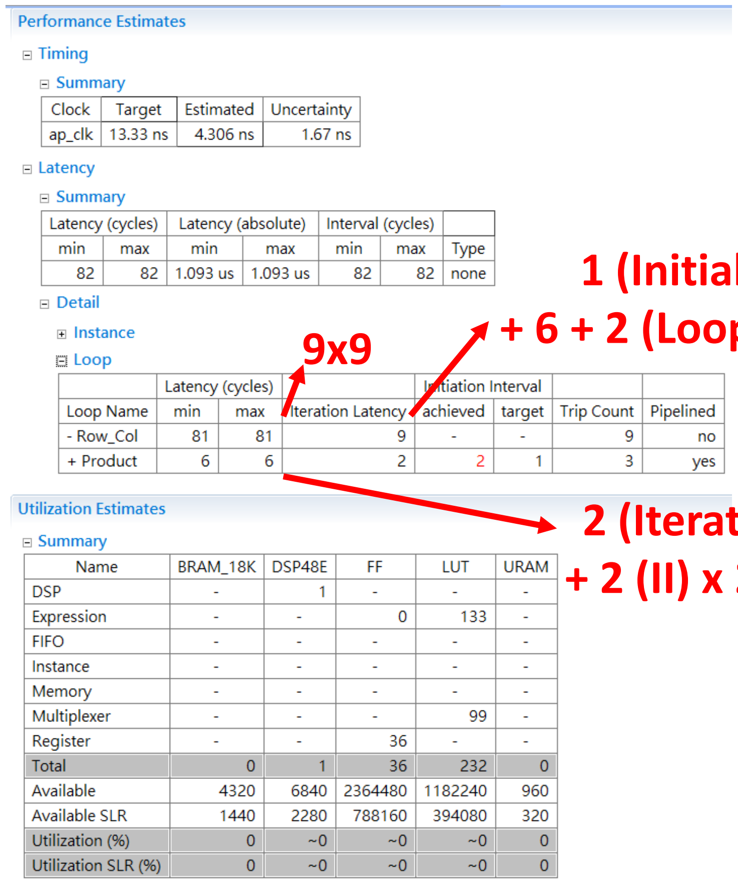
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	116	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	69	-
Register	-	-	44	-	-
Total	0	1	44	185	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

@ Case: Pipelining Product Loop

- Pipelining inner loops should lead to **Loop Flattening** of outer loops
  - Product Loop is **NOT flattened** into Row_Col loop
  - Product Loop cannot achieve **II = 1**



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	4.306 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

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	133	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	99	-
Register	-	-	36	-	-
Total	0	1	36	232	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

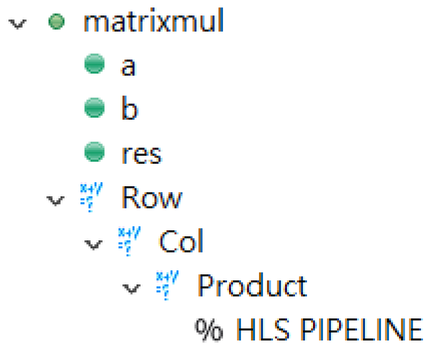
9x9

1 (Initialization)
+ 6 + 2 (Loop Condition)

2 (Iteration Latency)
+ 2 (II) x 2 (Trip Count)

Issue 1.
I/O-restricted Flattening

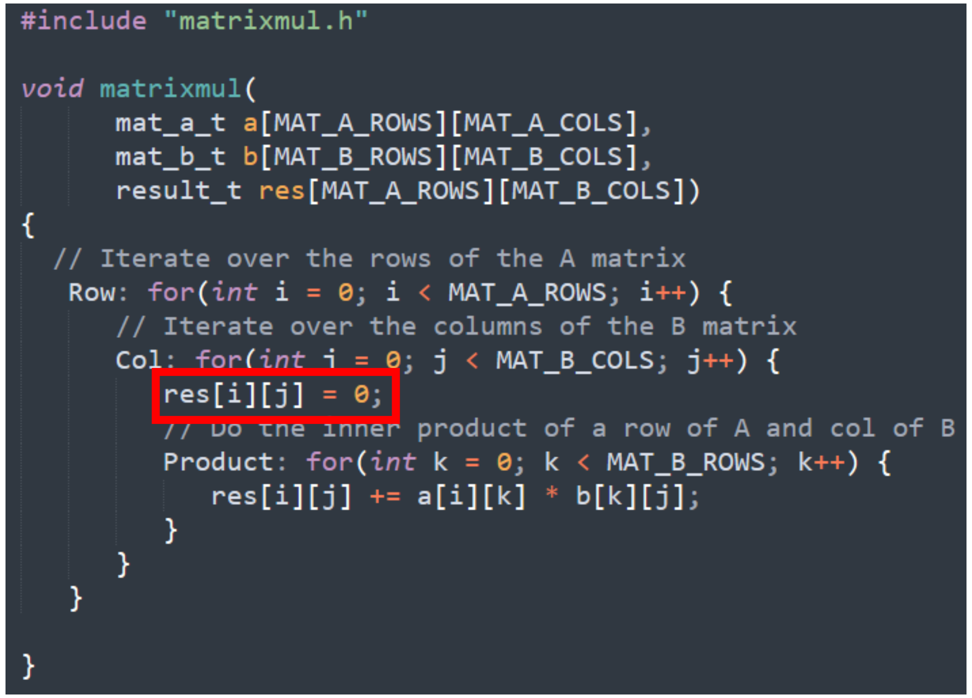
Issue 2.
Carried Dependency



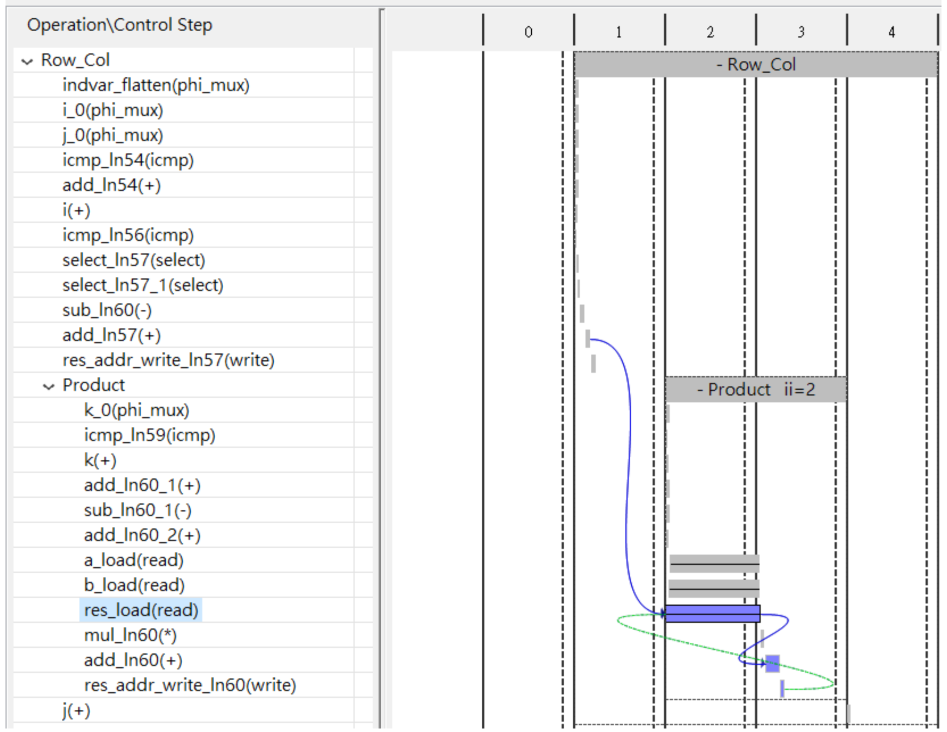
```
matrixmul
    a
    b
    res
    Row
        Col
            Product
                % HLS PIPELINE
```

Issue 1. I/O-restricted Flattening

- Before each **Product Loop**, the associated **res[i][j] is set to zero**
- Since **res** is a <u>top-level</u> function argument → RTL-level I/O behavior
  - Not an internal operation



```
#include "matrixmul.h"

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];
         }
      }
   }

}
```

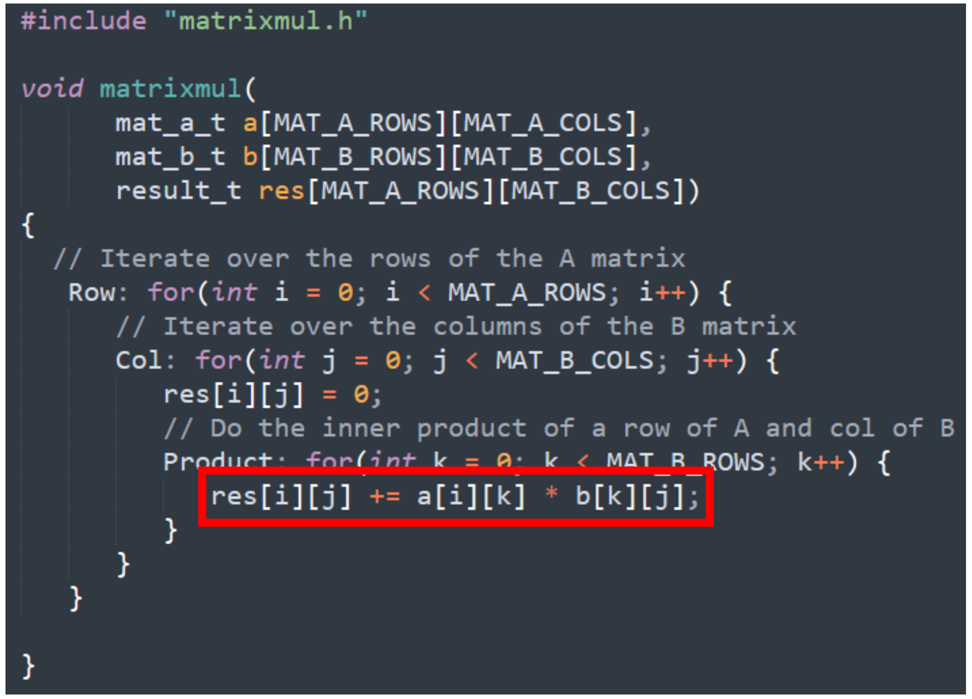


```
Operation\Control Step        0   1   2   3   4
Row_Col                           - Row_Col
  indvar_flatten(phi_mux)
  i_0(phi_mux)
  j_0(phi_mux)
  icmp_ln54(icmp)
  add_ln54(+)
  i(+)
  icmp_ln56(icmp)
  select_ln57(select)
  select_ln57_1(select)
  sub_ln60(-)
  add_ln57(+)
  res_addr_write_ln57(write)
  Product                          - Product  ii=2
    k_0(phi_mux)
    icmp_ln59(icmp)
    k(+)
    add_ln60_1(+)
    sub_ln60_1(-)
    add_ln60_2(+)
    a_load(read)
    b_load(read)
    res_load(read)
    mul_ln60(*)
    add_ln60(+)
    res_addr_write_ln60(write)
  j(+)
```

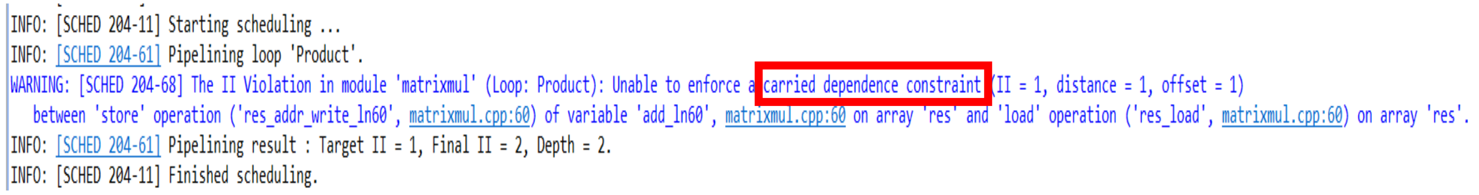
Prevents the Product loop from being flattened into the Row_Col loop
→ It costs a <u>clock cycle</u> to move between loops in the loop hierarchy

Issue 2. Carried Dependency

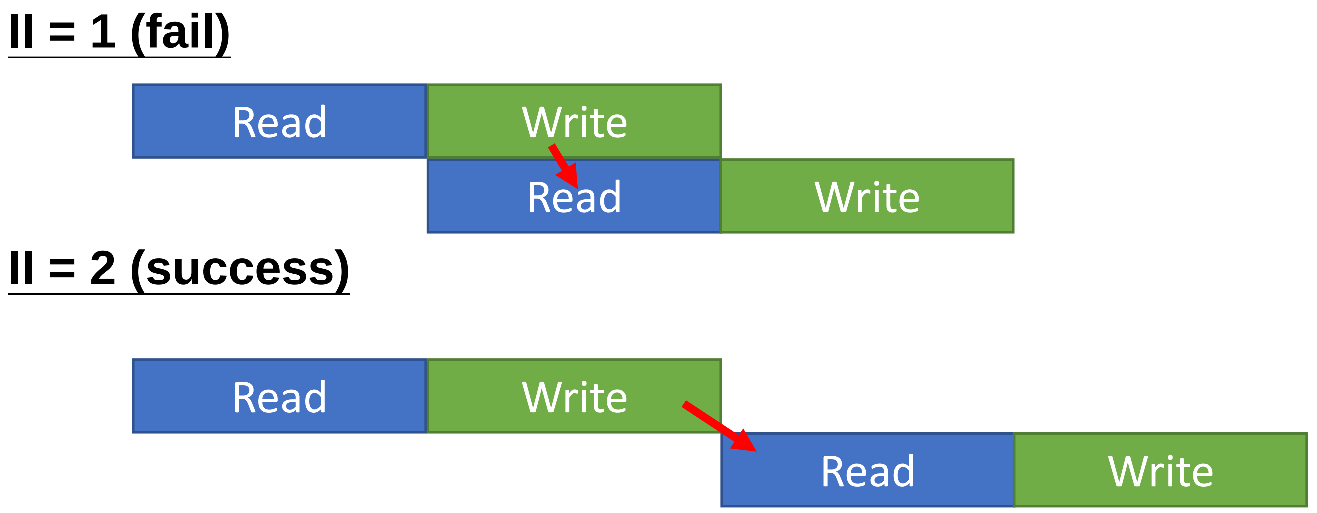
- Dependency between different iterations of the same loop
  - e.g., k=1 and k=2
- RAW (Read After Write) Dependency
  - The 2nd read cannot occur until the 1st write has finished



```
#include "matrixmul.h"

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];
         }
      }
   }

}
```



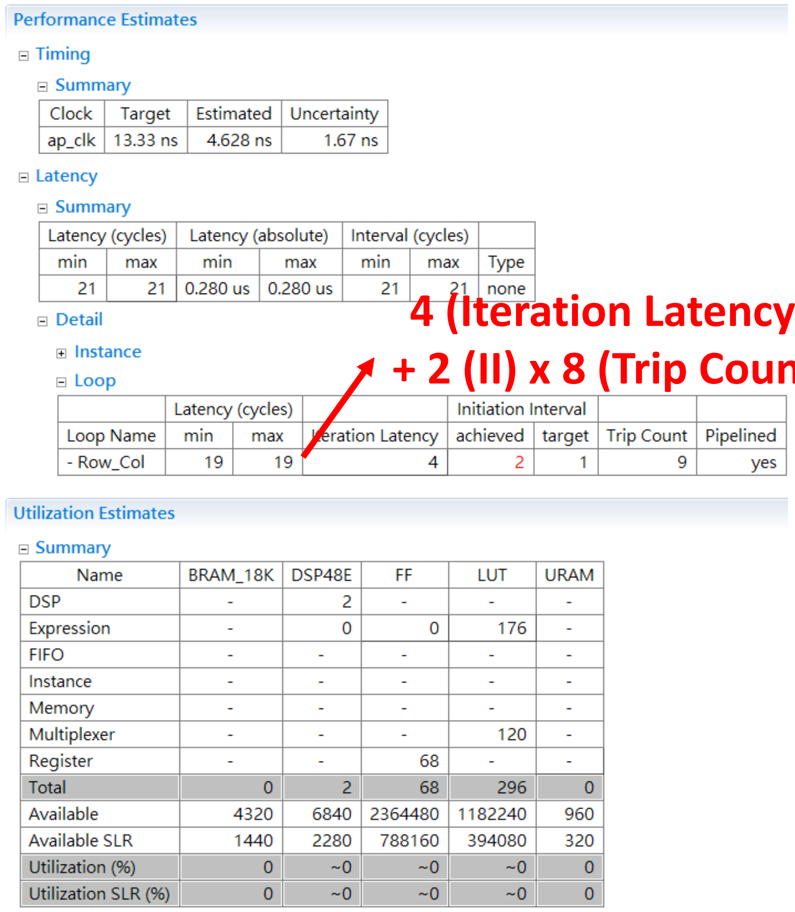
```
INFO: [SCHED 204-11] Starting scheduling ...
INFO: [SCHED 204-61] Pipelining loop 'Product'.
WARNING: [SCHED 204-68] The II Violation in module 'matrixmul' (Loop: Product): Unable to enforce a carried dependence constraint (II = 1, distance = 1, offset = 1)
   between 'store' operation ('res_addr_write_ln60', 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.
```



@ Case: Pipelining Col Loop

- Pipelining outer loops should lead to **Loop Unrolling** of inner loops
  - Product Loop is **unrolled**
  - Product Row_Col cannot achieve **II = 1**

Issue.
Resource limitations on array a and b



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	4.628 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
21	21	0.280 us	0.280 us	21	21	none

Detail

Instance

Loop

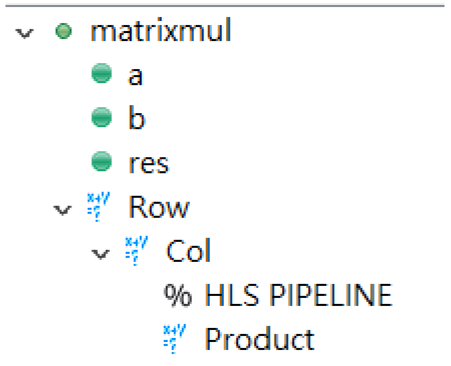
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	19	19	4	2	1	9	yes

4 (Iteration Latency)
+ 2 (II) x 8 (Trip Count)

Utilization Estimates

Summary

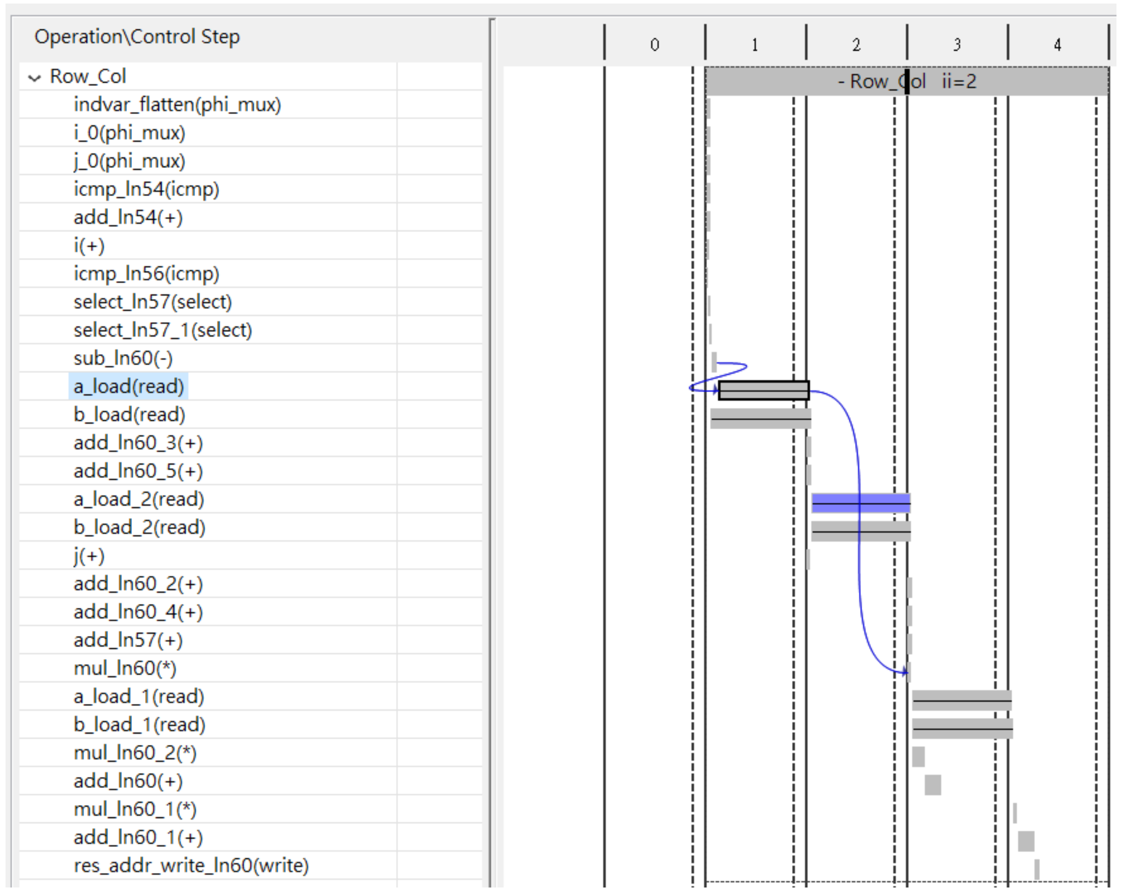
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	176	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	120	-
Register	-	-	68	-	-
Total	0	2	68	296	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	0	~0	~0	~0	0
Utilization SLR (%)	0	~0	~0	~0	0



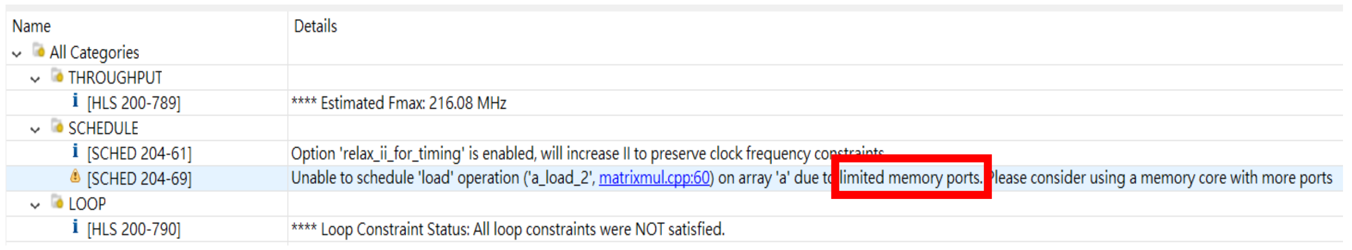
```
matrixmul
    a
    b
    res
    Row
        Col
            % HLS PIPELINE
            Product
```

Issue. Resource Limitations on Arrays

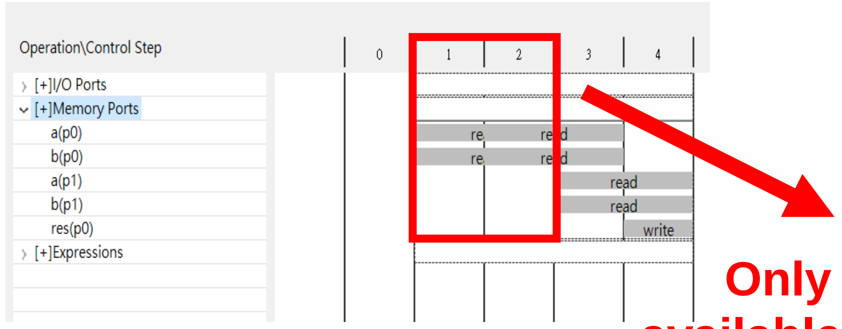
- There are three read operations on arrays a and b
  - Unrolling Product loop leads to **3x** parallelism



```
Row_Col
    indvar_flatten(phi_mux)
    i_0(phi_mux)
    j_0(phi_mux)
    icmp_ln54(icmp)
    add_ln54(+)
    i(+)
    icmp_ln56(icmp)
    select_ln57(select)
    select_ln57_1(select)
    sub_ln60(-)
    a_load(read)
    b_load(read)
    add_ln60_3(+)
    add_ln60_5(+)
    a_load_2(read)
    b_load_2(read)
    j(+)
    add_ln60_2(+)
    add_ln60_4(+)
    add_ln57(+)
    mul_ln60(*)
    a_load_1(read)
    b_load_1(read)
    mul_ln60_2(*)
    add_ln60(+)
    mul_ln60_1(*)
    add_ln60_1(+)
    res_addr_write_ln60(write)
```



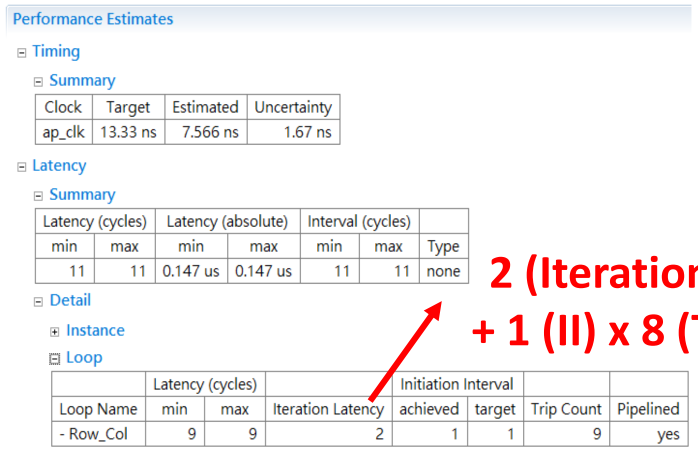
Name	Details
All Categories	
THROUGHPUT	
[HLS 200-789]	**** Estimated Fmax: 216.08 MHz
SCHEDULE	
[SCHED 204-61]	Option 'relax_ii_for_timing' is enabled, will increase II to preserve clock frequency constraints.
[SCHED 204-69]	Unable to schedule 'load' operation ('a_load_2', matrixmul.cpp:60) on array 'a' due to limited memory ports. Please consider using a memory core with more ports
LOOP	
[HLS 200-790]	**** Loop Constraint Status: All loop constraints were NOT satisfied.



Only 2 read operations available with dual-port BRAM

@ Case: Pipelining Col Loop and RESHAPE

- Reshaping arrays a and b allows one wide array (port) to be created
  - Product Row_Col can achieve **II = 1**



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	7.566 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
11	11	0.147 us	0.147 us	11	11	none

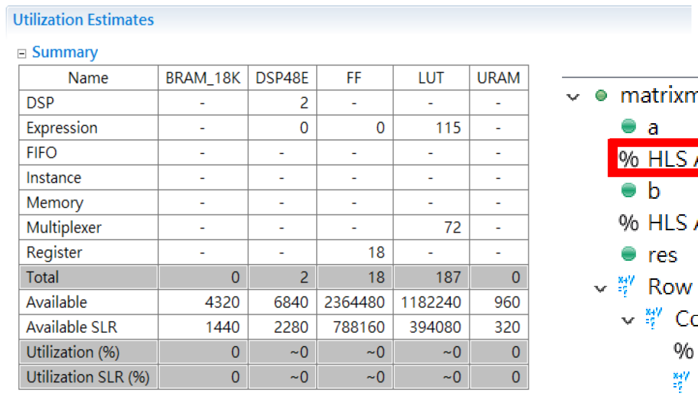
Detail

Instance

Loop

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

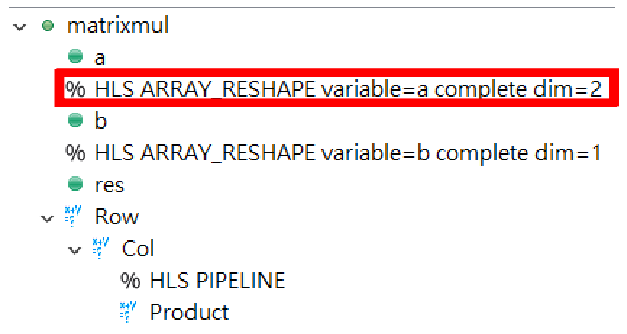
2 (Iteration Latency)
+ 1 (II) x 8 (Trip Count)



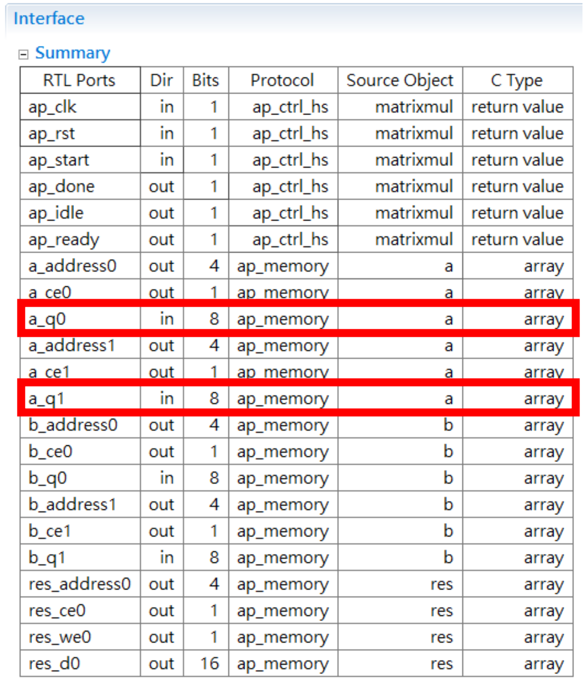
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	115	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	72	-
Register	-	-	18	-	-
Total	0	2	18	187	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



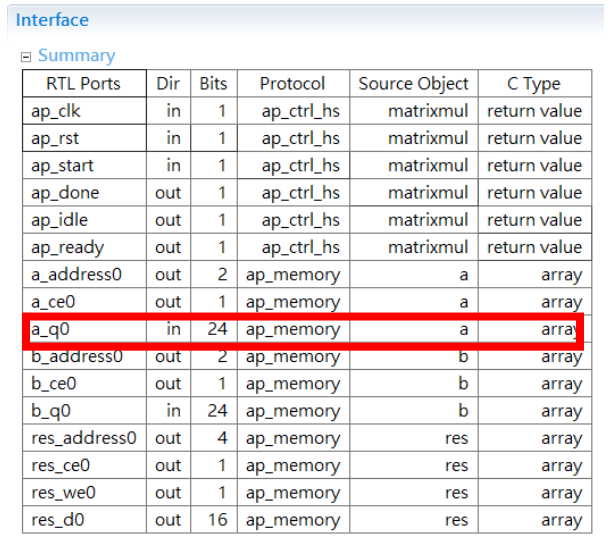
```
matrixmul
    a
    % HLS ARRAY_RESHAPE variable=a complete dim=2
    b
    % HLS ARRAY_RESHAPE variable=b complete dim=1
    res
    Row
        Col
            % HLS PIPELINE
            Product
```



Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	matrixmul	return value
ap_rst	in	1	ap_ctrl_hs	matrixmul	return value
ap_start	in	1	ap_ctrl_hs	matrixmul	return value
ap_done	out	1	ap_ctrl_hs	matrixmul	return value
ap_idle	out	1	ap_ctrl_hs	matrixmul	return value
ap_ready	out	1	ap_ctrl_hs	matrixmul	return value
a_address0	out	4	ap_memory	a	array
a_ce0	out	1	ap_memory	a	array
a_q0	in	8	ap_memory	a	array
a_address1	out	4	ap_memory	a	array
a_ce1	out	1	ap_memory	a	array
a_q1	in	8	ap_memory	a	array
b_address0	out	4	ap_memory	b	array
b_ce0	out	1	ap_memory	b	array
b_q0	in	8	ap_memory	b	array
b_address1	out	4	ap_memory	b	array
b_ce1	out	1	ap_memory	b	array
b_q1	in	8	ap_memory	b	array
res_address0	out	4	ap_memory	res	array
res_ce0	out	1	ap_memory	res	array
res_we0	out	1	ap_memory	res	array
res_d0	out	16	ap_memory	res	array



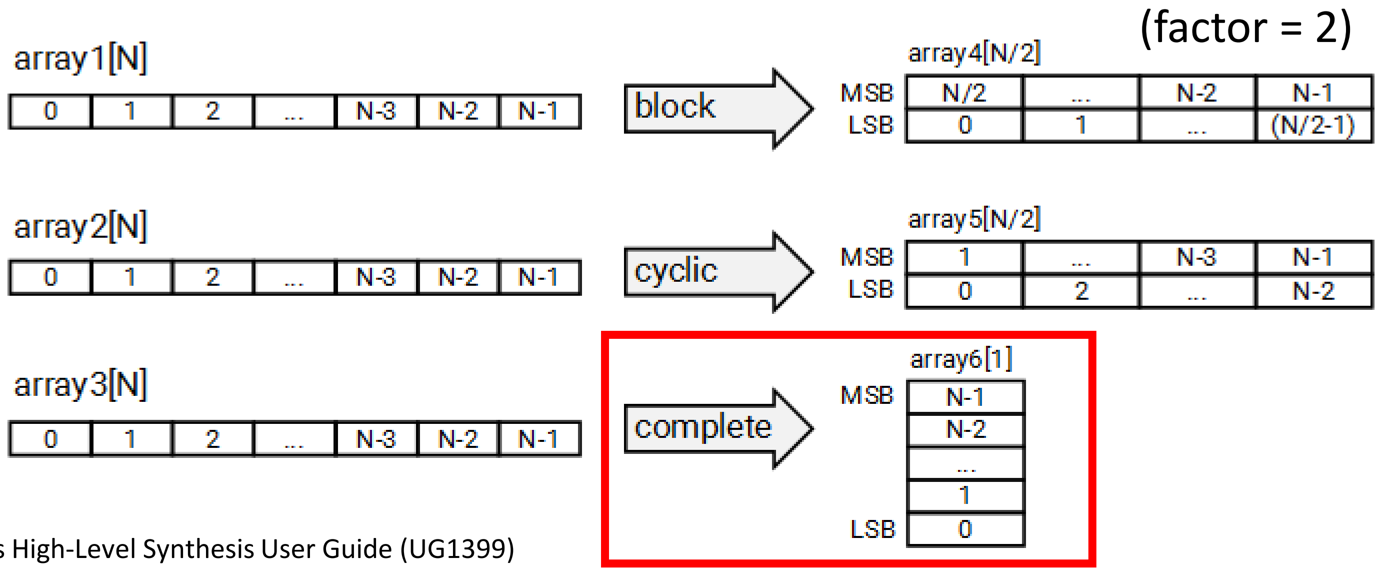
Interface

Summary

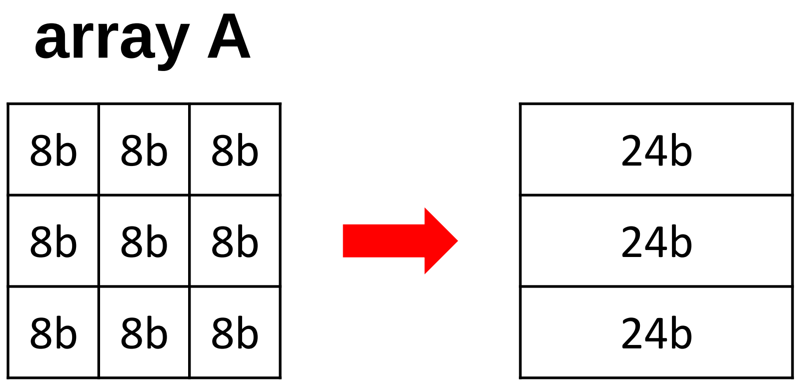
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	matrixmul	return value
ap_rst	in	1	ap_ctrl_hs	matrixmul	return value
ap_start	in	1	ap_ctrl_hs	matrixmul	return value
ap_done	out	1	ap_ctrl_hs	matrixmul	return value
ap_idle	out	1	ap_ctrl_hs	matrixmul	return value
ap_ready	out	1	ap_ctrl_hs	matrixmul	return value
a_address0	out	2	ap_memory	a	array
a_ce0	out	1	ap_memory	a	array
a_q0	in	24	ap_memory	a	array
b_address0	out	2	ap_memory	b	array
b_ce0	out	1	ap_memory	b	array
b_q0	in	24	ap_memory	b	array
res_address0	out	4	ap_memory	res	array
res_ce0	out	1	ap_memory	res	array
res_we0	out	1	ap_memory	res	array
res_d0	out	16	ap_memory	res	array

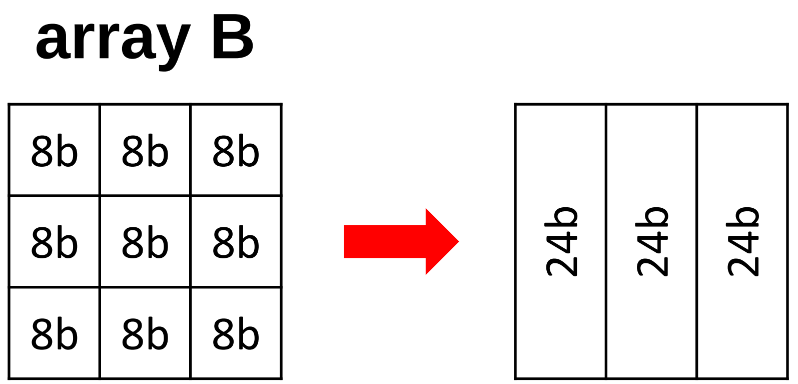
ARRAY_RESHAPE Methods

- ARRAY_RESHAPE: Creates a new array with fewer elements (1/N) but with greater bit-width (N times)
  - 3x8b → 24b in this case



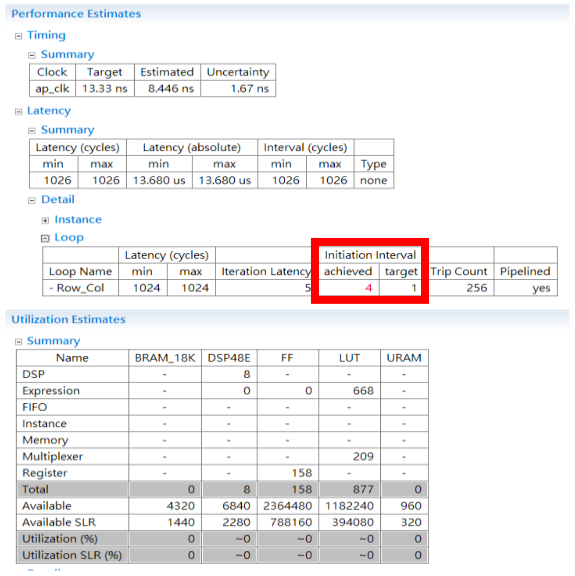
Vitis High-Level Synthesis User Guide (UG1399)





Comparison of ARRAY_RESHAPE Methods

- While BLOCK method is not often used, it can achieve the same performance as the COMPLETE method (II = 1)
  - Carefully choose the **block factor**



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	8.446 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1026	1026	13.680 us	13.680 us	1026	1026	none

Detail

Instance

Loop

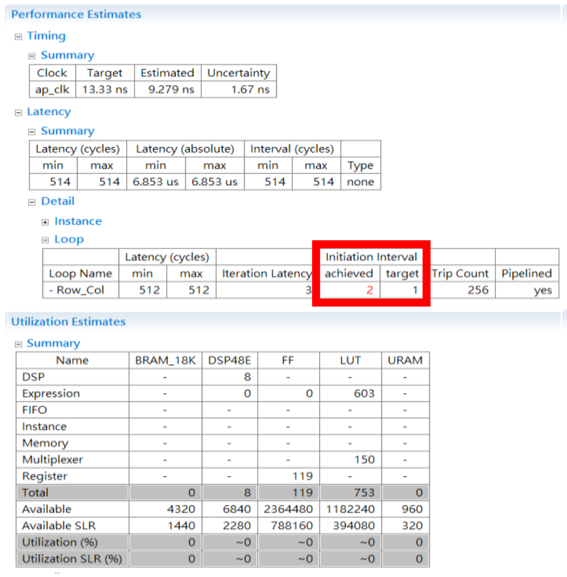
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	1024	1024	5	4	1	256	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	8	-	-	-
Expression	-	0	0	668	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	209	-
Register	-	-	158	-	-
Total	0	8	158	877	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

(BLOCK factor = 2)



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	9.279 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
514	514	6.853 us	6.853 us	514	514	none

Detail

Instance

Loop

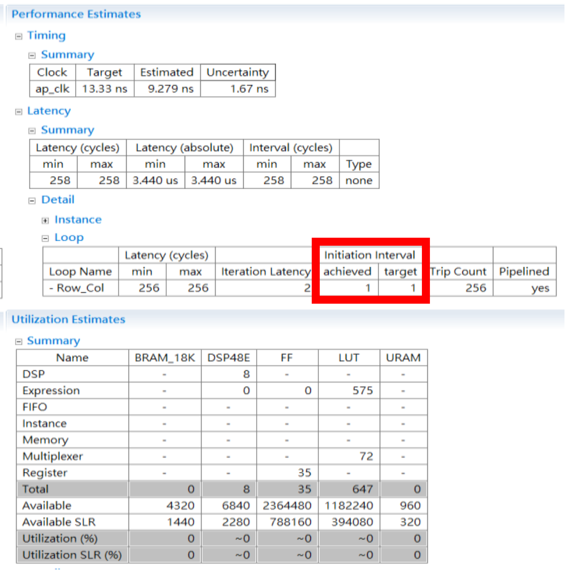
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	512	512	3	2	1	256	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	8	-	-	-
Expression	-	0	0	603	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	150	-
Register	-	-	119	-	-
Total	0	8	119	753	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

(BLOCK factor = 4)



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	9.279 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
258	258	3.440 us	3.440 us	258	258	none

Detail

Instance

Loop

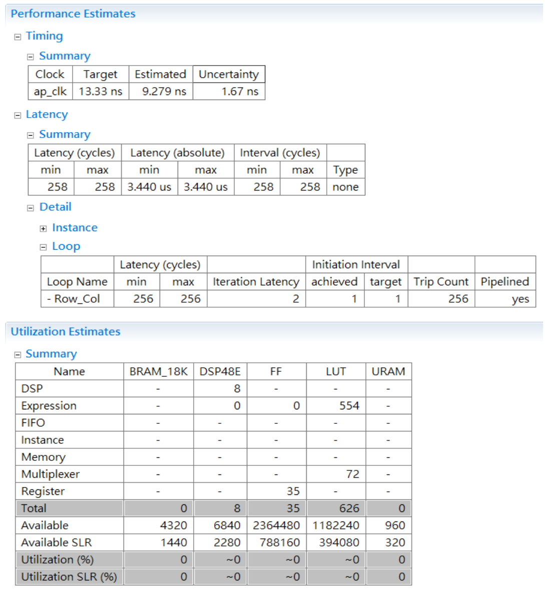
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	256	256	2	1	1	256	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	8	-	-	-
Expression	-	0	0	575	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	72	-
Register	-	-	35	-	-
Total	0	8	35	647	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

(BLOCK factor = 8)



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	9.279 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
258	258	3.440 us	3.440 us	258	258	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	256	256	2	1	1	256	yes

Utilization Estimates

Summary

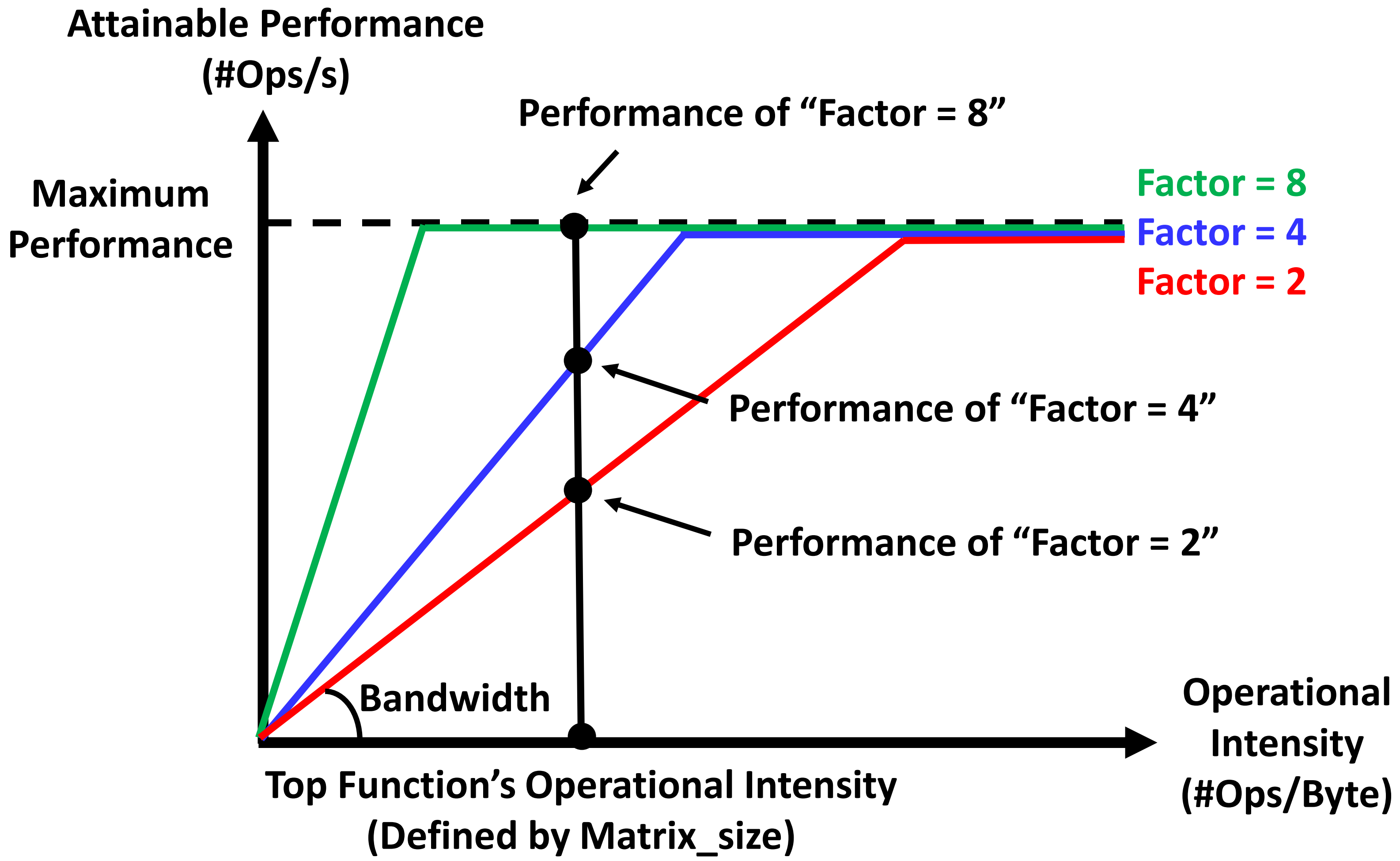
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	8	-	-	-
Expression	-	0	0	554	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	72	-
Register	-	-	35	-	-
Total	0	8	35	626	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	0	~0	~0	~0	0
Utilization SLR (%)	0	~0	~0	~0	0

COMPLETE method

Matrix_size = 16 → Need 8x bandwidth using dual-port BRAM

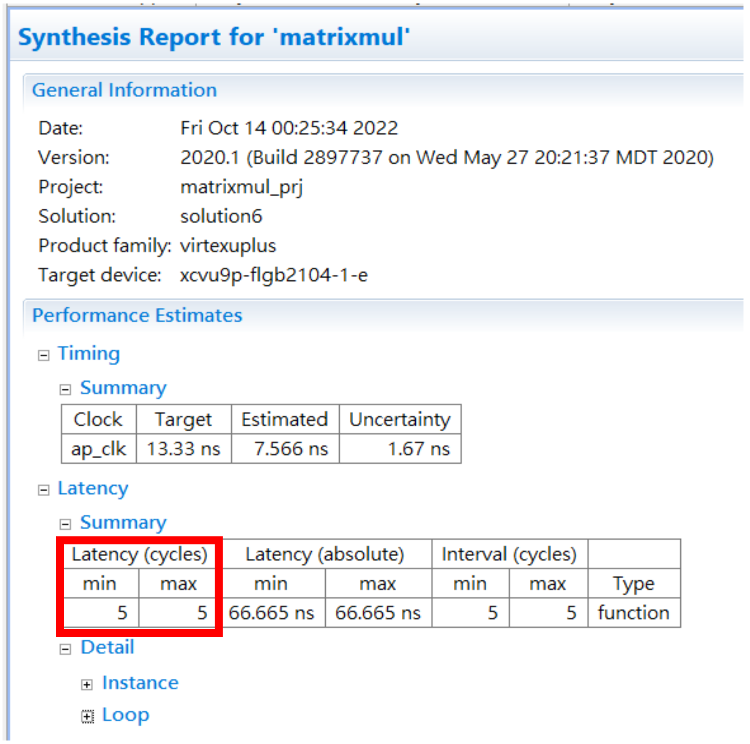
Explanation with Roofline Model

- Increasing the **block factor** is equivalent to adjusting the slope
  - The **slope** in the roofline model represents the HW bandwidth


Attainable Performance (#Ops/s)
Performance of "Factor = 8"
Maximum Performance
Factor = 8
Factor = 4
Factor = 2
Performance of "Factor = 4"
Performance of "Factor = 2"
Bandwidth
Operational Intensity (#Ops/Byte)
Top Function's Operational Intensity (Defined by Matrix_size)

@ Case: Pipelining the Function

- Pipelining a function causes <u>all loops unrolled</u>
- Outperform @ **Case: Pipelining Col Loop and RESHAPE** (II=1) at the cost of much higher HW costs (**DSP 9x, FF 19x, LUT 3.02x**)
  - Only **2x** speedup → not a good design trade-off



Synthesis Report for 'matrixmul'

General Information

Date:	Fri Oct 14 00:25:34 2022
Version:	2020.1 (Build 2897737 on Wed May 27 20:21:37 MDT 2020)
Project:	matrixmul_prj
Solution:	solution6
Product family:	virtexuplus
Target device:	xcvu9p-flgb2104-1-e

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	7.566 ns	1.67 ns

Latency

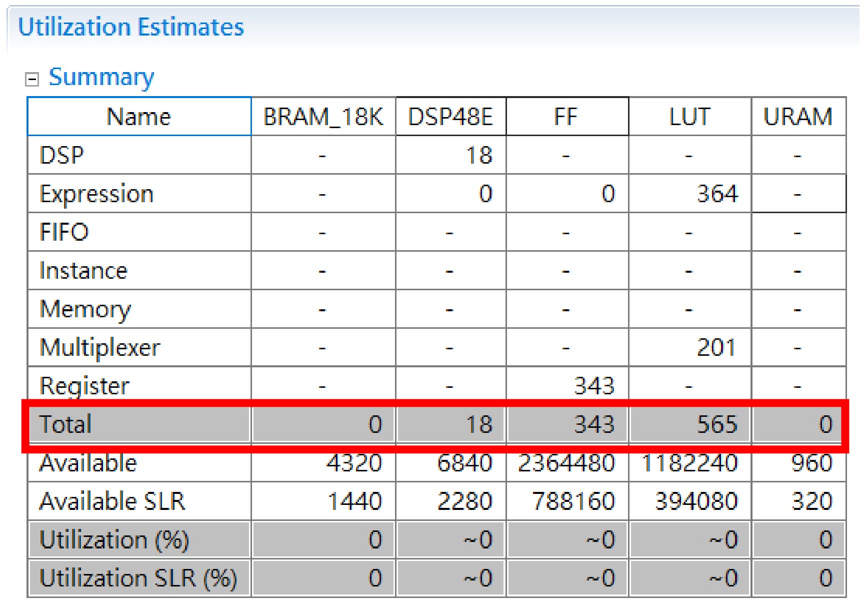
Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
5	5	66.665 ns	66.665 ns	5	5	function

Detail

Instance

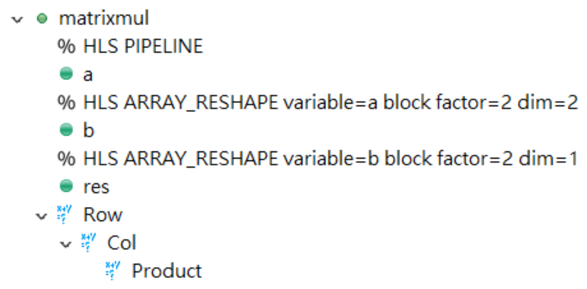
Loop



Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	18	-	-	-
Expression	-	0	0	364	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	201	-
Register	-	-	343	-	-
Total	0	18	343	565	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

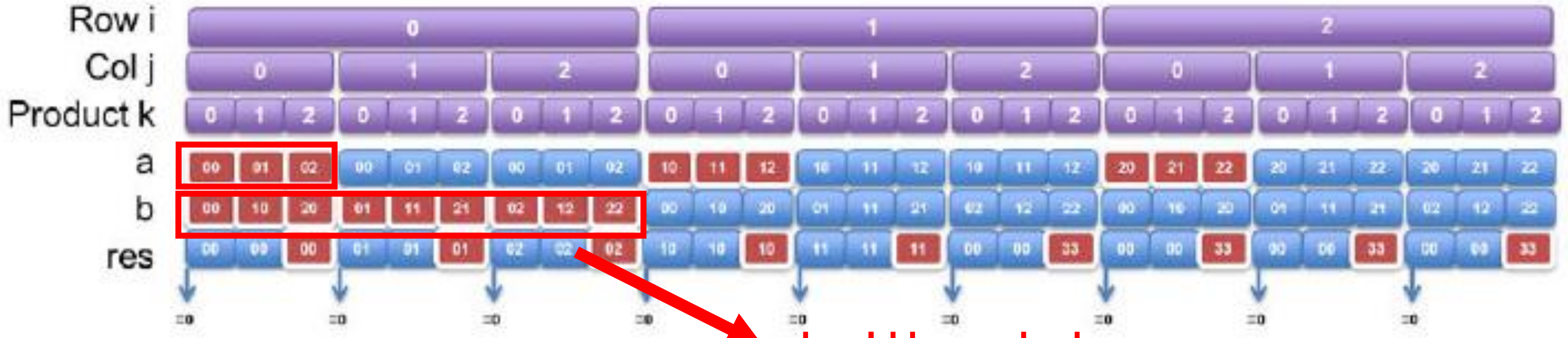


```
matrixmul
  % HLS PIPELINE
  a
  % HLS ARRAY_RESHAPE variable=a block factor=2 dim=2
  b
  % HLS ARRAY_RESHAPE variable=b block factor=2 dim=1
  res
  Row
    Col
      Product
```

Cannot Apply FIFO

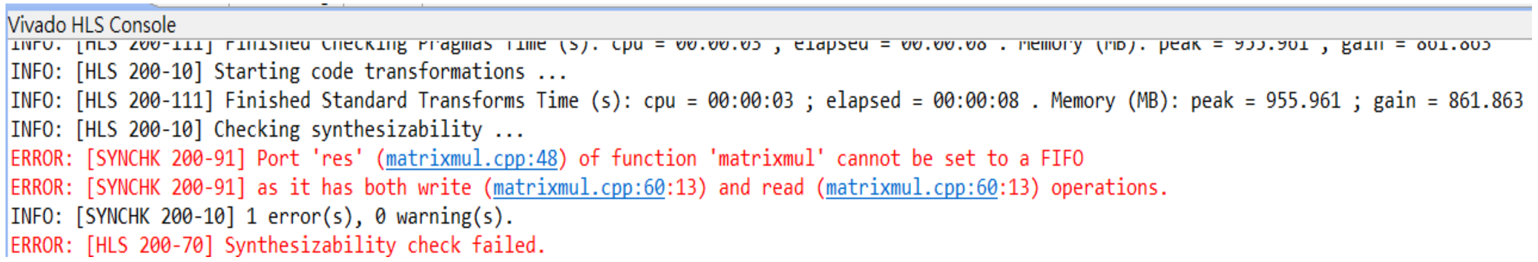
- In a FIFO interface, the values must be accessed **sequentially**
- The code currently enforces a certain order of reads and writes.
  - The optimization directives are inadequate

The ports accesses can only be those shown highlighted in red



array A

00	01	02
10	11	12
20	21	22

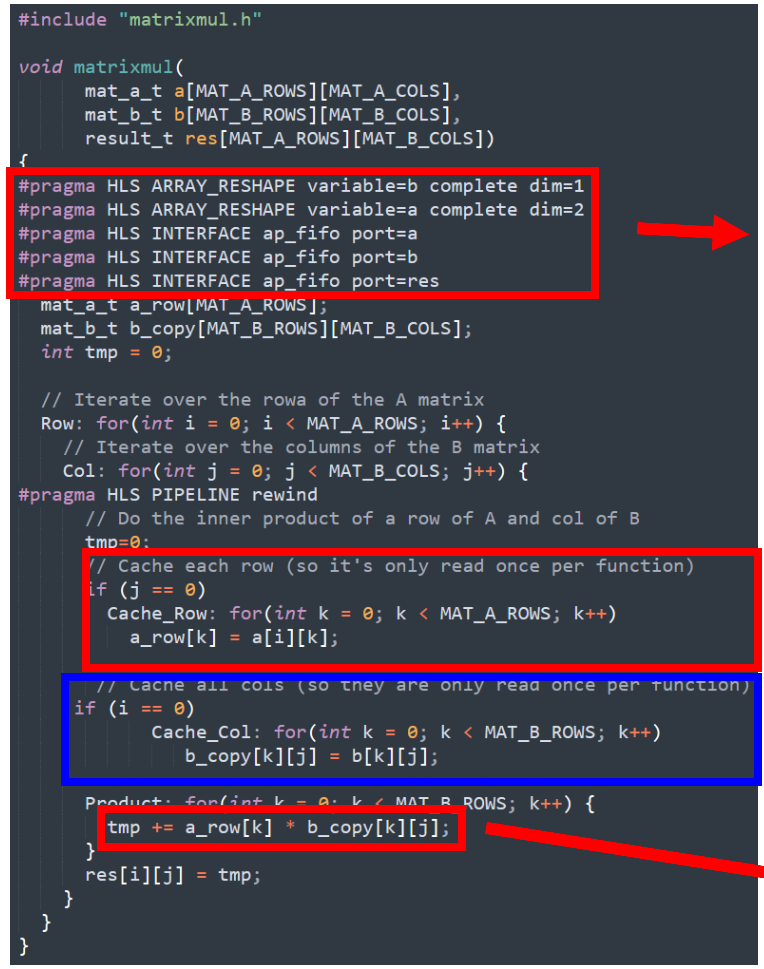


```
Vivado HLS Console
INFO: [HLS 200-111] Finished Checking Pragmas Time (s): cpu = 00:00:03 ; elapsed = 00:00:08 . Memory (MB): peak = 955.961 ; gain = 861.863
INFO: [HLS 200-10] Starting code transformations ...
INFO: [HLS 200-111] Finished Standard Transforms Time (s): cpu = 00:00:03 ; elapsed = 00:00:08 . Memory (MB): peak = 955.961 ; gain = 861.863
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.
```

Re-writing the code is required

Lab2: C-Code Optimized for I/O Accesses

- The modified code **caches arrays** if necessary
  - Store temporary data for future reuse



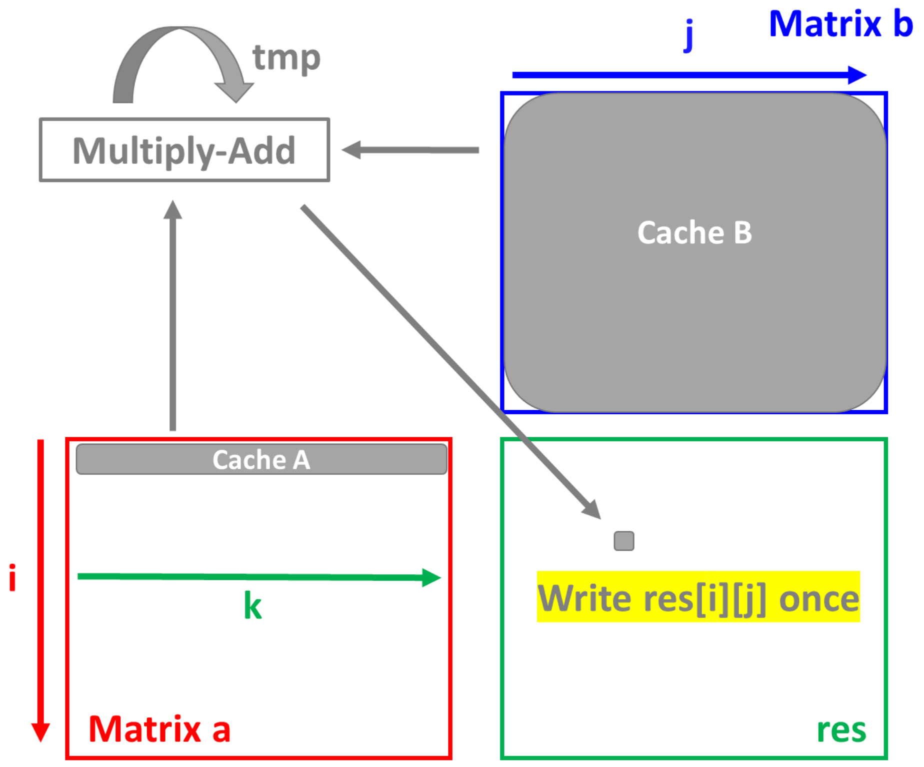
```c
#include "matrixmul.h"

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;
    }
  }
}
```

Setup
(use ARRAY_RESHAPE
to ensure bandwidth)

Cache A

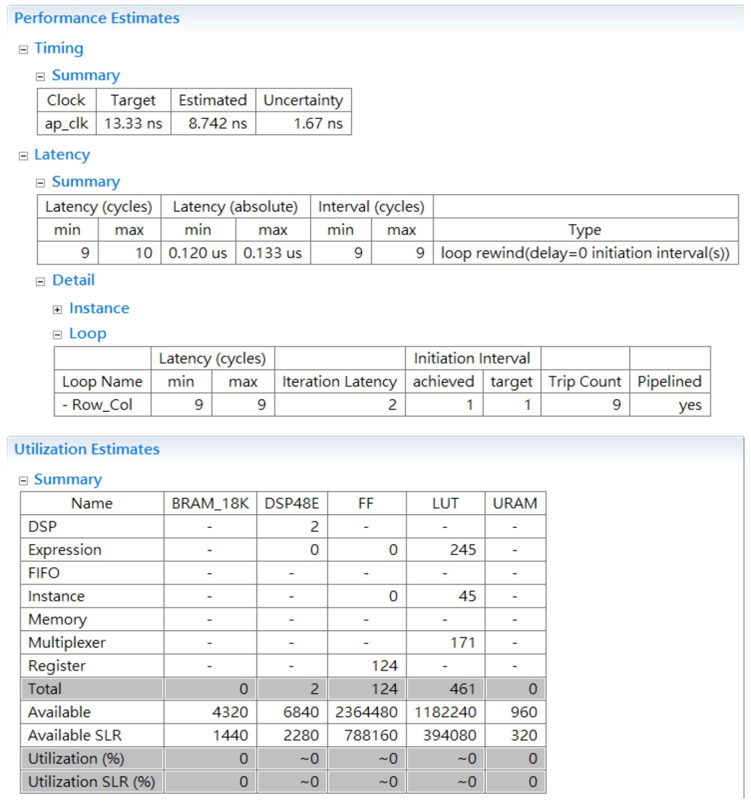
Cache B

Use variable
rather than res



@ Case: FIFO Interface with Pipeline Rewind

- Apply pipeline to Col loop to unsure that the **cache read** and **Product loop** are paralleled
  - Compared with **@ Case: Pipelining Col Loop and RESHAPE** (II=1) , the HW costs increase (**FF 6.8x, LUT 2.46x**)



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	8.742 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
9	10	0.120 us	0.133 us	9	9	loop rewind(delay=0 initiation interval(s))

Detail

Instance

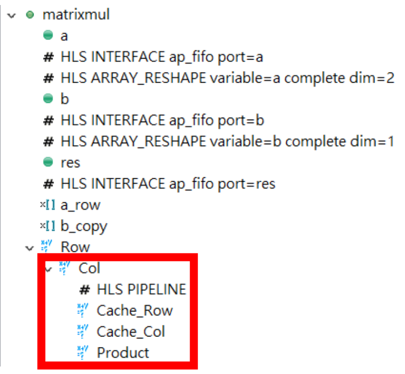
Loop

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

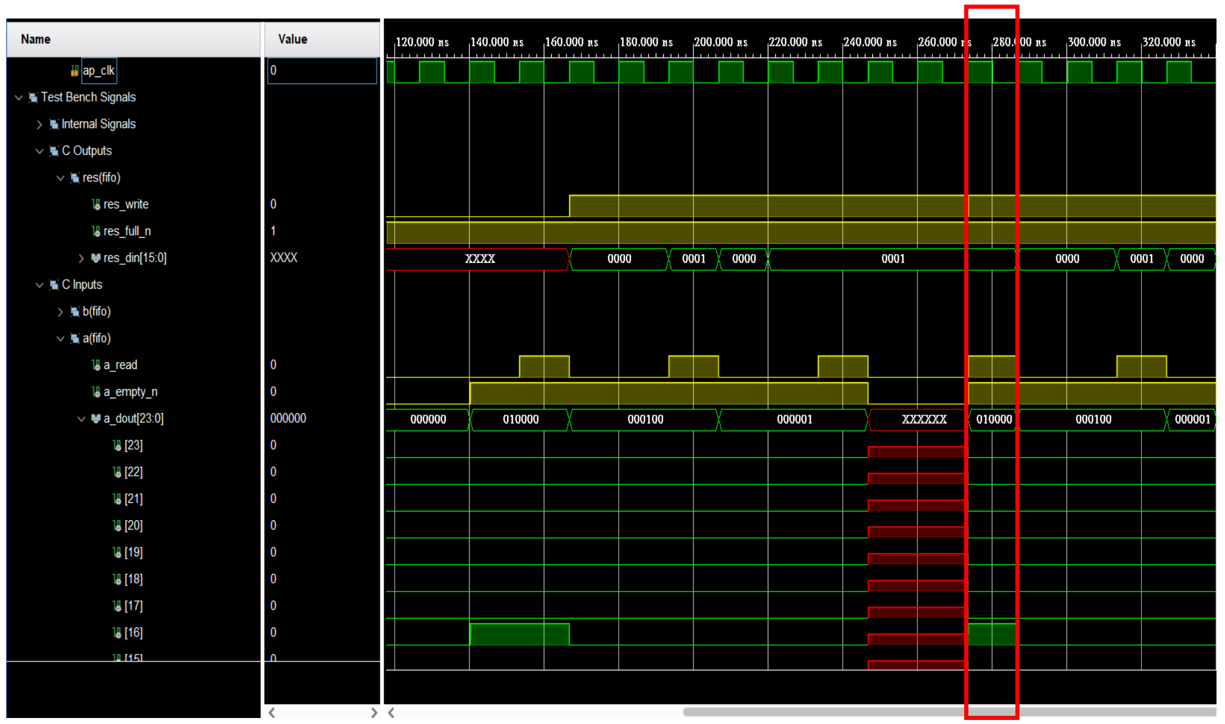
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	245	-
FIFO	-	-	-	-	-
Instance	-	-	0	45	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	171	-
Register	-	-	124	-	-
Total	0	2	124	461	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



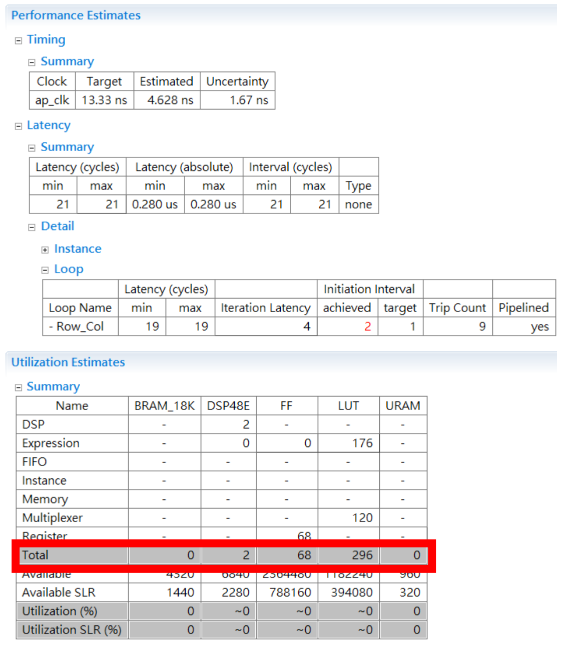
```
matrixmul
  a
  # HLS INTERFACE ap_fifo port=a
  # HLS ARRAY_RESHAPE variable=a complete dim=2
  b
  # HLS INTERFACE ap_fifo port=b
  # HLS ARRAY_RESHAPE variable=b complete dim=1
  res
  # HLS INTERFACE ap_fifo port=res
  a_row
  b_copy
  Row
    Col
      # HLS PIPELINE
      Cache_Row
      Cache_Col
      Product
```



Pre-load A arrays for the next Loop

Questions

- In @ **Case: Pipelining Col Loop**, why are the costs of FF and LUT higher than those of @ **Case: Pipelining Col Loop and RESHAPE?**
  - Need FF and LUT to store the temporary data (like cache)
- In @ **Case: FIFO Interface with Pipeline Rewind**, why do we have to cache a row of A array, but **the whole B array?**
  - It depends on the loop order (row-major → column-major)



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	4.628 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
21	21	0.280 us	0.280 us	21	21	none

Detail

Instance

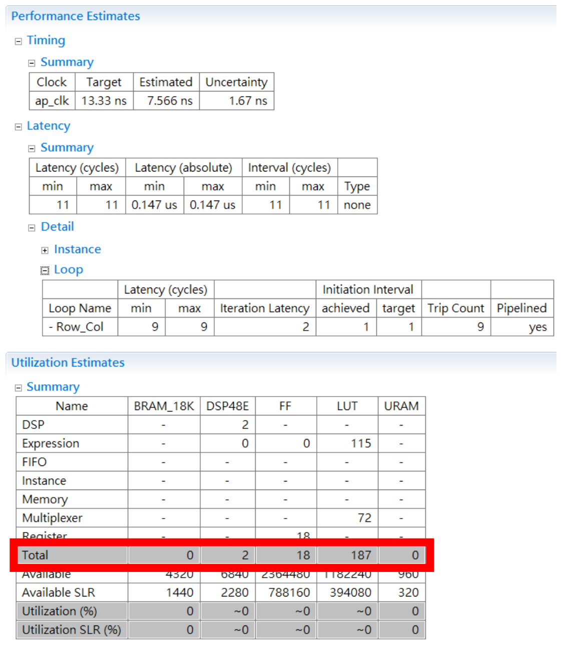
Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	19	19	4	2	1	9	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	176	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	120	-
Register	-	-	68	-	-
Total	0	2	68	296	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



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	13.33 ns	7.566 ns	1.67 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
11	11	0.147 us	0.147 us	11	11	none

Detail

Instance

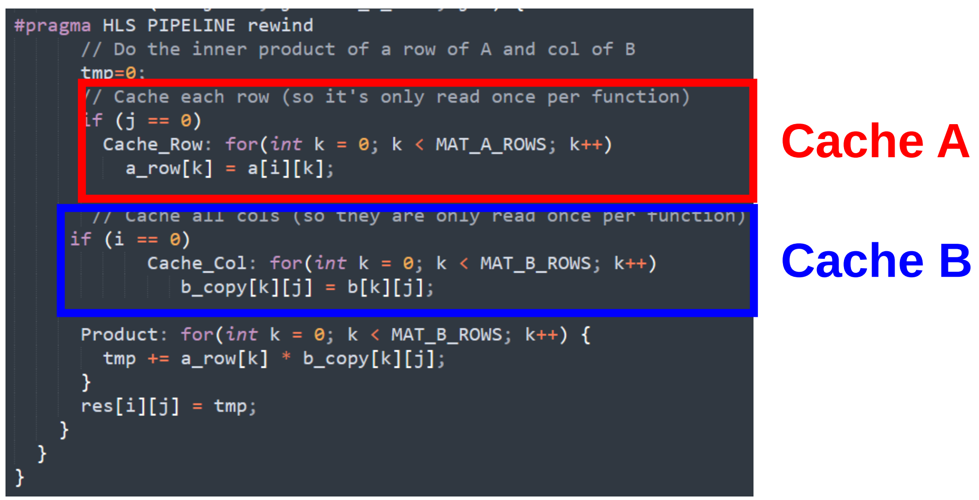
Loop

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

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	115	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	72	-
Register	-	-	18	-	-
Total	0	2	18	187	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



```
#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;
    }
  }
}
```

Thanks for Listening