18-643 Lecture 8: C-function-to-IP HLS (Vitis HLS IP-Flow)

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today: learn how to tell Vitis what you want and understand what Vitis tells you back
- Notices
  - Handout #5: lab 2, due noon, 10/9
  - Project status report due each Friday
- Readings (see lecture schedule online)
  - Ch 15, The Zynq Book (skim Ch 14)
  - https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/.html (see Lab 2 handout for reading recommendations)
  - for lab2, C. Zhang, et al., ISFPGA, 2015

Vitis C-to-RTL HLS

- Function-to-IP, not Program-to-HW
  - never mind all of C (what's main( )? what malloc?)
  - never mind all usages of allowed subset (all loops okay, but static ones actually work well)
  - what else beyond C might a HW designer need (types, interface, structural hints)

You can use it as a better RTL

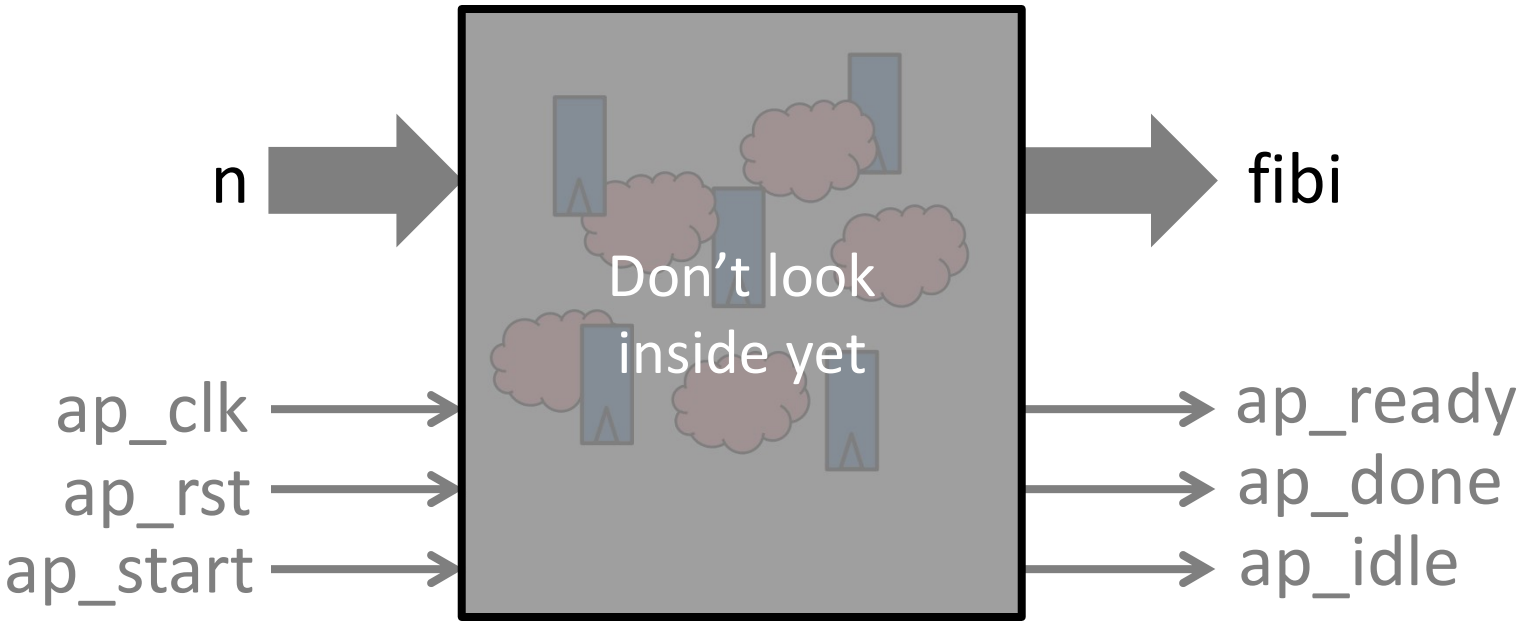
- Designer still in charge (garbage in, garbage out)
  - specify functionality as algorithm (in C)
  - specify structure as pragmas (beyond C)
  - set optimization constraints (beyond C)

Offload bit- and cycle-level design/opt. to tools

What does Vitis HLS see?

```
int fibi(int n) {
  int last=1; int lastlast=0; int temp;

  if (n==0) return 0;
  if (n==1) return 1;

  for(;n>1;n--) {
    temp=last+lastlast;
    lastlast=last;
    last=temp;
  }

  return temp;
}
```

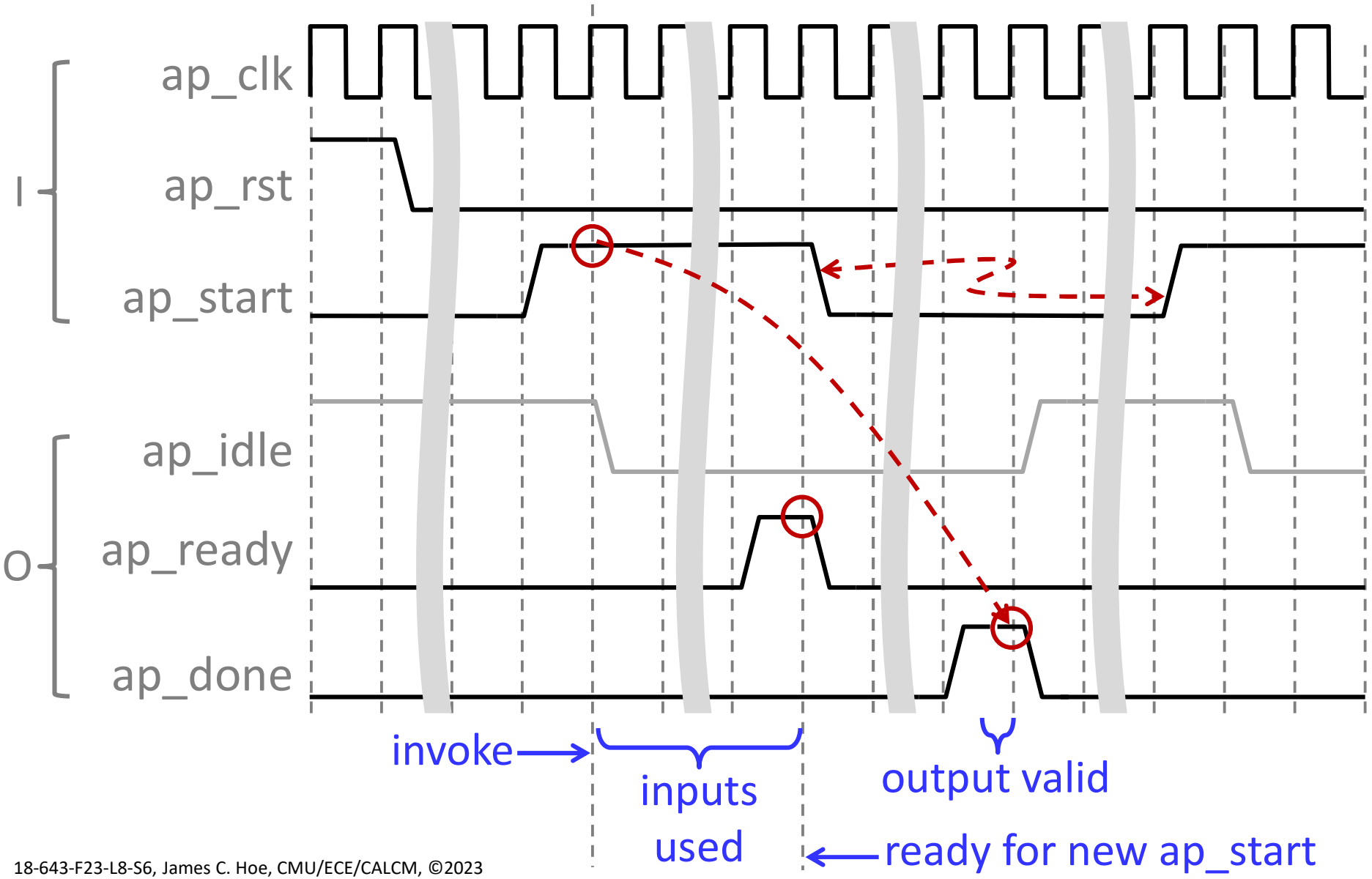
Function to IP Block



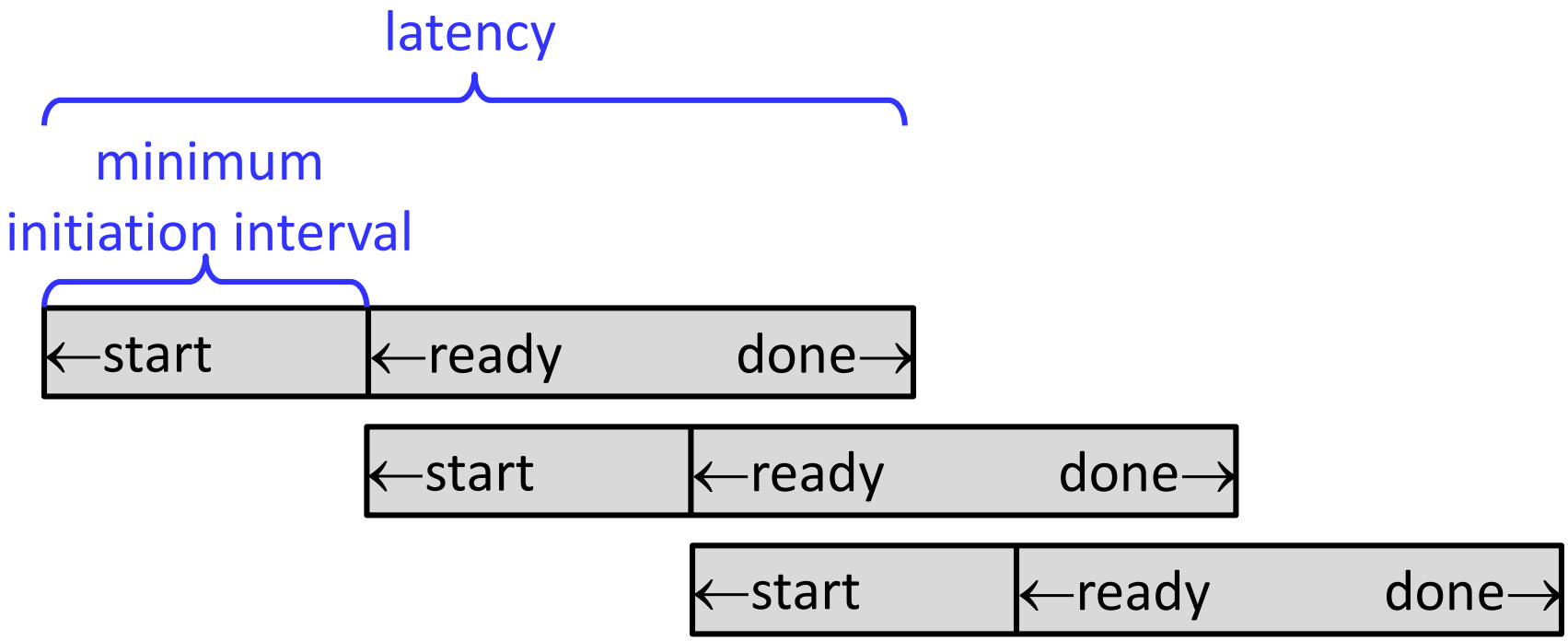
Later—what if you want 2 outputs?

```
int fibi(int n) {
  . . . .
  return ...;
}
```

AP_CTRL_HS Block Protocol



Function Invocation: Latency vs Throughput

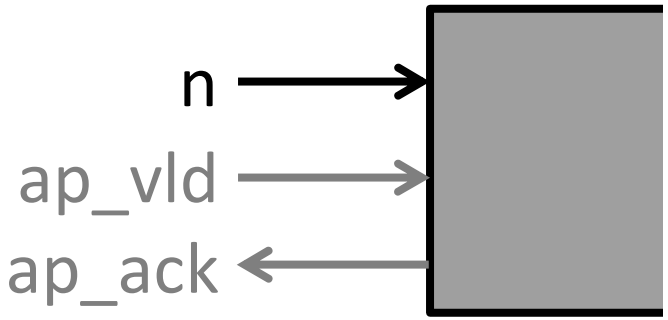


Other Block Control Options

- ap_ctrl_chain
  - separate input producer and output consumer
  - ap_continue: driven by the consumer to backpressure the block and producer
  - IF a block reaches "done" AND ap_continue is deasserted, the block will hold ap_done and keep output valid until ap_continue is asserted
- AXI compatible memory-mapped control
  - software on ARM interacts with the block using fxn-call-like interfaces (input, output, start, etc.)
  - IP-specific .h and routines generated automatically

Scalar I/O Port Timing

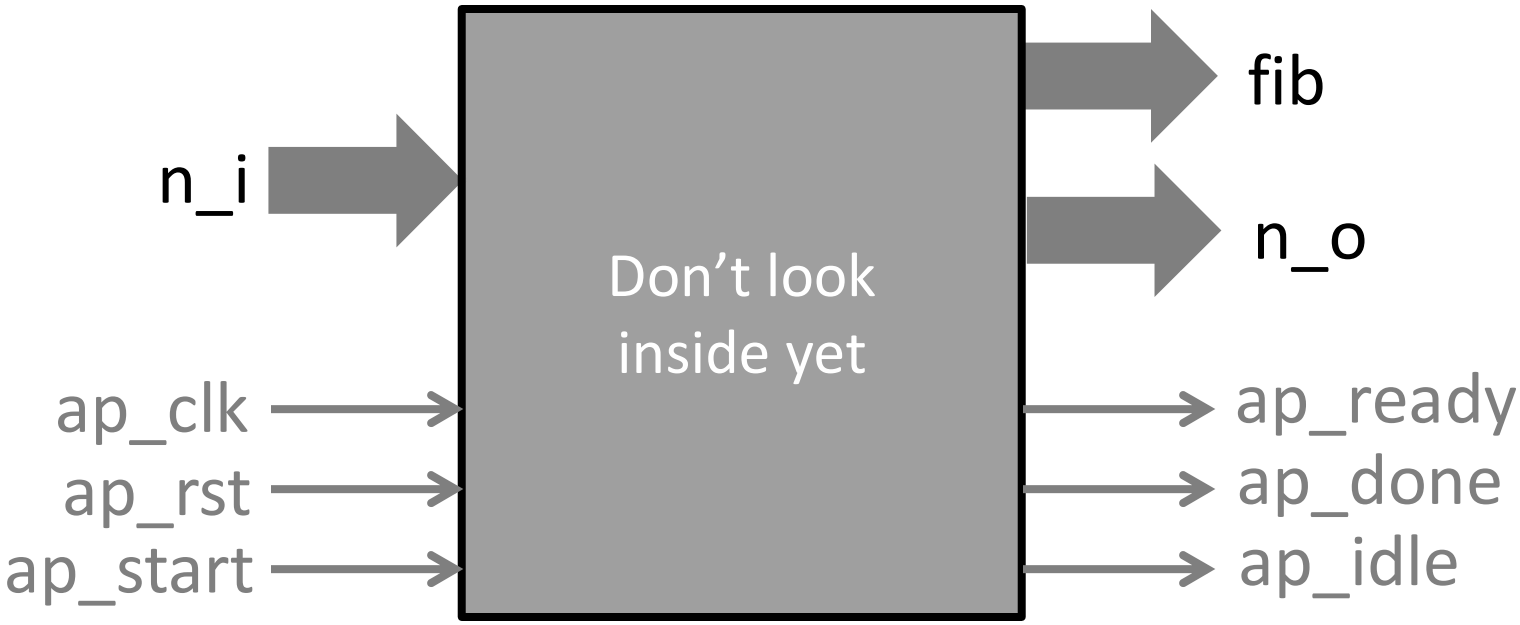
- By default (ap_none)
  - input ports should be stable between ap_start and ap_ready
  - output port is valid when ap_done
- 3 asynchronous handshake options on input
  - ap_vld only: consumes only if input valid
  - ap_ack only: signals back when input consumed
  - ap_hs: ap_vld + ap_ack
- HLS's job to follow protocol



Pass-by-Reference Arguments

```
void fibi(int *n, int *fib) {
  int last=1; int lastlast=0; int temp;
  int nn=*n;

  if (nn==0) { *fib=0; *n=0; return; }
  if (nn==1) { *fib=1; *n=0; return; }
  for(;nn>1;nn--) {
    temp=last+lastlast;
    lastlast=last;
    last=temp;
  }

  *fib=last; *n=lastlast;
}
```

Pass-by-Pointer & Reference



They are not really "pointers"

- do not evaluate *(fib+1) or fib
- except to pretend to be a fifo

```
void fibi(int *n, int *fib) {
   . . . .
   *n and *fib assigned to;
   *n in RHS before assigned;
   *fib in RHS after assigned;
   . . . .
}
```

All I/O Options

Argument Type	Scalar		Array			Pointer or Reference		
Interface Mode	Input	Return	I	I/O	O	I	I/O	O
ap_ctrl_none								
ap_ctrl_hs		D						
ap_ctrl_chain								
axis								
s_axilite								
m_axi								
ap_none	D					D		
ap_stable								
ap_ack								
ap_vld								D
ap_ovld							D	
ap_hs								
ap_memory			D	D	D			
bram								
ap_fifo								
ap_bus								

Supported D = Default Interface Not Supported

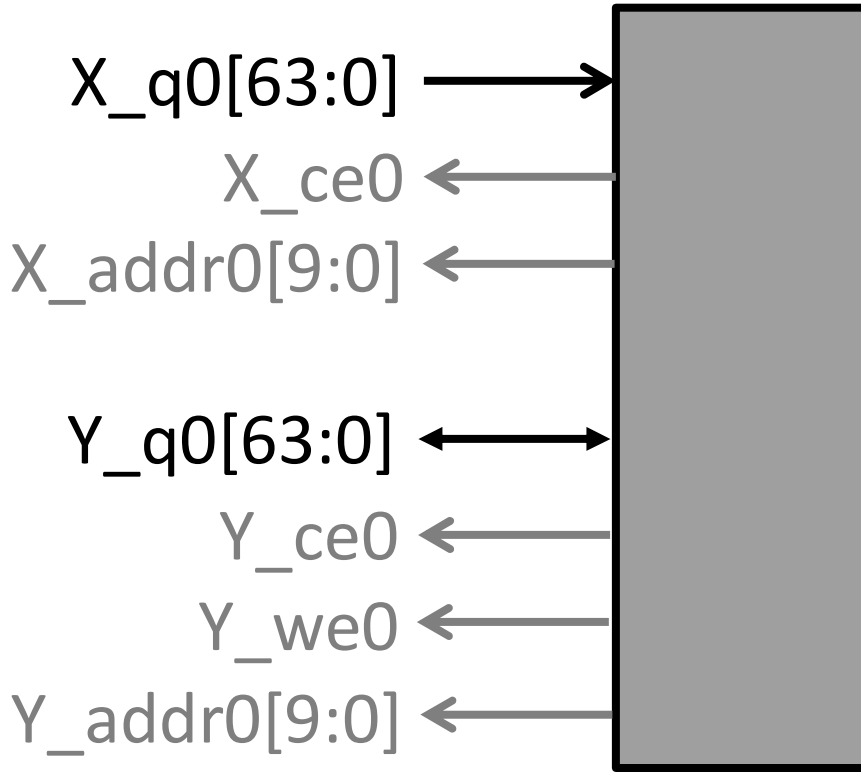
X14293

pointer I/O can output before fxn ends and multiple times

Fig 1-49, Vivado Design Suite User Guide: High-Level Synthesis

Array Arguments

```
#define N (1<<10)
void D2XPY (double Y[N], double X[N]) {
      for(i=0; i<N; i++) {
            Y[i]=2*X[i]+Y[i];
      }
}
```



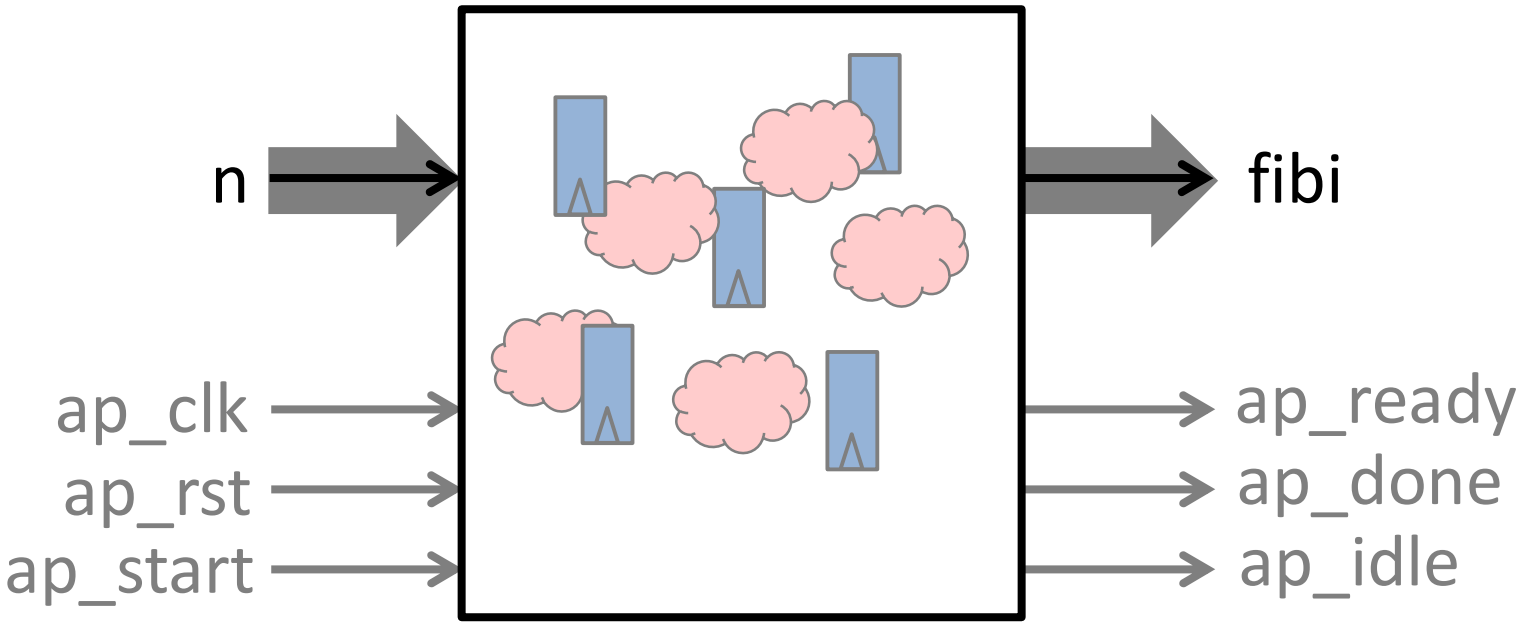
*could ask to use separate read and write ports

Array Arg Options

- By default, array args become BRAM ports
  - array must be fixed size
  - can use 2 ports for bandwidth or split read/write
- If array arg is accessed always consecutively AND only either read or written
  - can become ap_fifo port
  - i.e., no address wires, just push or pop
- Array args can also become AXI or a generic bus master ports

Scheduler handles port sharing and dynamic delays

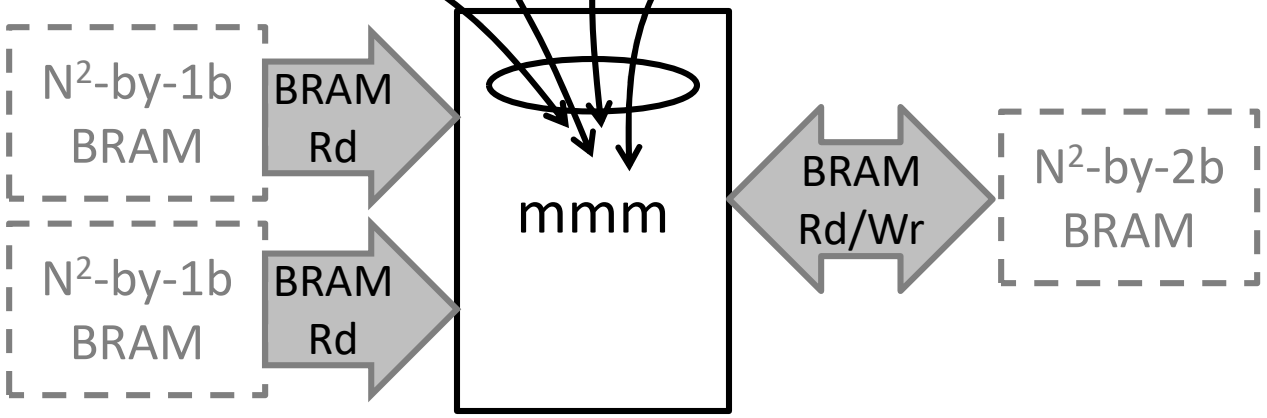
Time to Look Inside



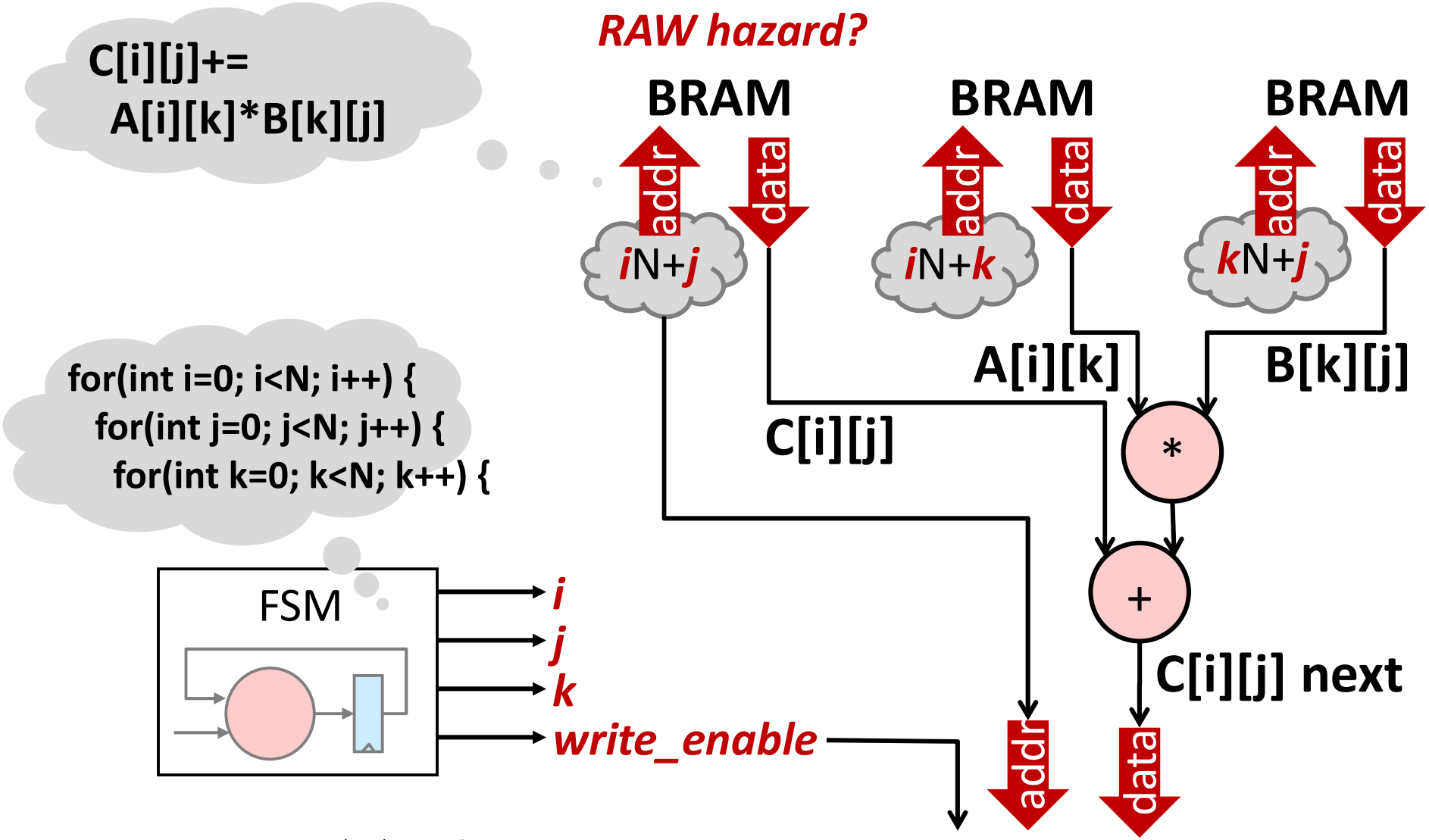
MMM (yet again)

```
void mmm(char A[N][N], char B[N][N], short C[N][N) {

   Row: for(int i=0; i<N; i++) {
      Col: for(int j=0; j<N; j++) {
         C[i][j]=0;
         Product: for(int k=0; k<N; k++) {
               C[i][j] += A[i][k]*B[k][j];
         }
      }
   }
}
```

keep it simple



If you want to be literal



[Vitis HLS 2020.2]

If you want to be literal

matrix_mult.cpp × | Synthesis Summary(nopipe) | Schedule Viewer(nopipe

```
#include "matrix_mult.h"

void matrix_mult(
      mat_a a[IN_A_ROWS][IN_A_COLS],
      mat_b b[IN_B_ROWS][IN_B_COLS],
      volatile mat_prod prod[IN_A_ROWS][IN_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < IN_A_ROWS; i++) {
      // Iterate over the columns of the B matrix
      Col: for(int j = 0; j < IN_B_COLS; j++) {
         prod[i][j] = 0;
         // Do the inner product of a row of A and col of B
         Product: for(int k = 0; k < IN_B_ROWS; k++) {
            prod[i][j] += a[i][k] * b[k][j];
         }
      }
   }

}
```

for 5x5 matrices

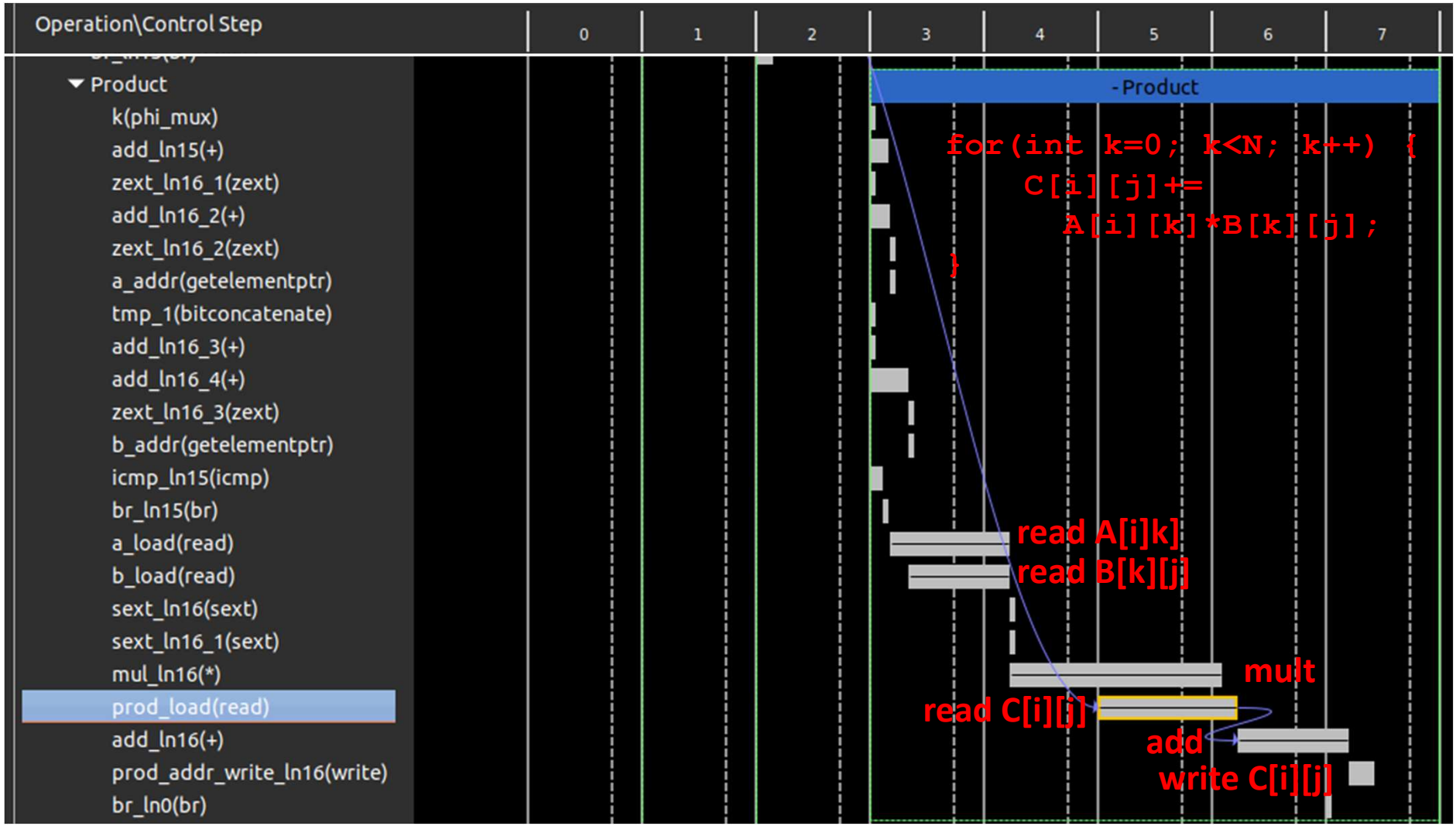
Outline | Directive ×

- matrix_mult
  - HLS TOP name=matrix_mult
  - a
  - b
  - prod
  - Row
    - HLS PIPELINE off
    - Col
      - HLS PIPELINE off
      - Product
        - HLS PIPELINE off

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
matrix_mult		-	686	6.860E3	-	687	-	no
Row		-	685	6.850E3	137	-	5	no
Col		-	135	1.350E3	27	-	5	no
Product		-	25	250.000	5	-	5	no

[Vitis HLS 2020.2]

If you want to be literal (continued)



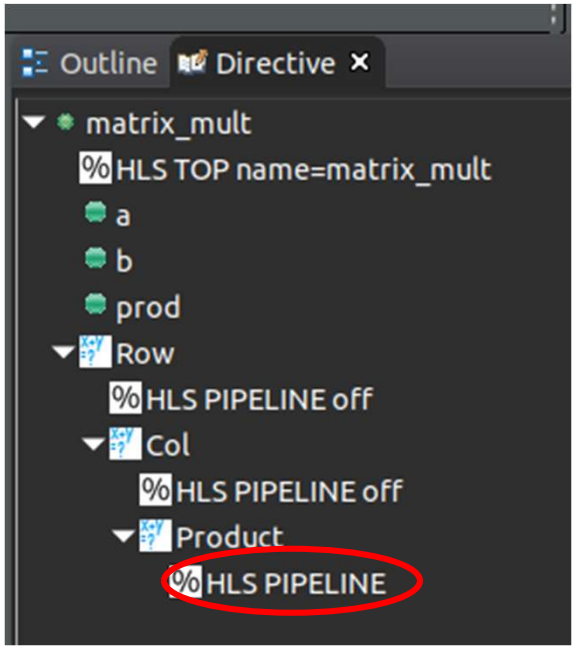
[Vitis HLS 2020.2]

Let's Try Pipelining

matrix_mult.cpp × | Synthesis Summary(nopipe) | Schedule Viewer(proc

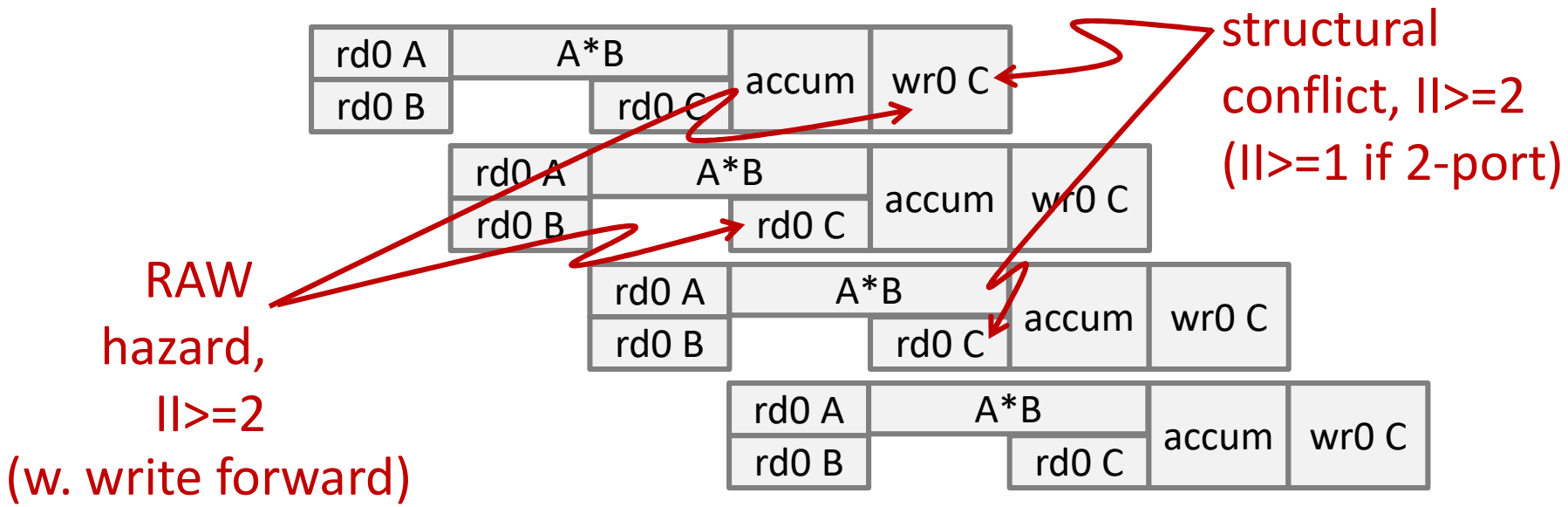
```
#include "matrix_mult.h"

void matrix_mult(
      mat_a a[IN_A_ROWS][IN_A_COLS],
      mat_b b[IN_B_ROWS][IN_B_COLS],
      volatile mat_prod prod[IN_A_ROWS][IN_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < IN_A_ROWS; i++) {
      // Iterate over the columns of the B matrix
      Col: for(int j = 0; j < IN_B_COLS; j++) {
         prod[i][j] = 0;
         // Do the inner product of a row of A and col of B
         Product: for(int k = 0; k < IN_B_ROWS; k++) {
            prod[i][j] += a[i][k] * b[k][j];
         }
      }
   }

}
```

for 5x5 matrices



Structural Pragma: Pipelining

- Find minimum "iteration interval (II)" schedule
  - II >= num stages a resource instance is used
  - II >= RAW hazard distance
- E.g., to pipeline `C[i][j]+=A[i][k]*B[k][j];`



[Vitis HLS 2020.2]

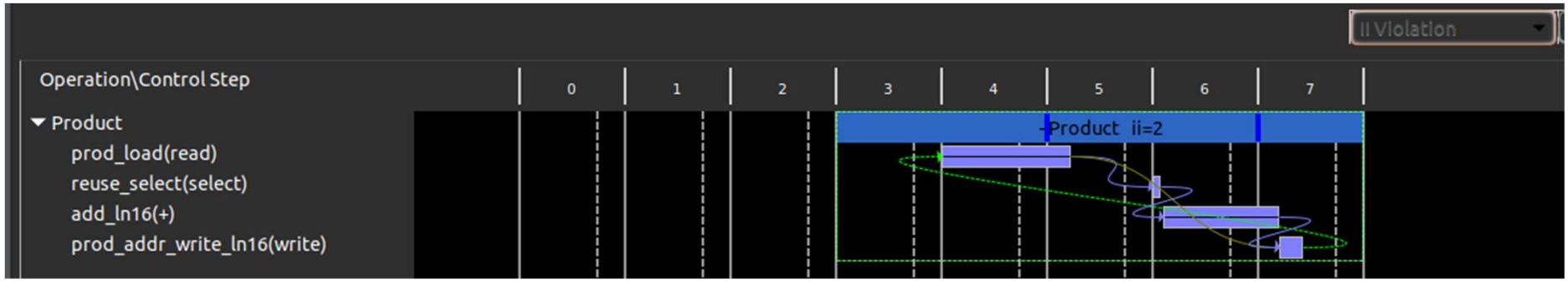
What Vitis HLS tells you . . .

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
matrix_mult	II Violation	-	376	3.760E3	-	377	-	no
Row_Col		-	375	3.750E3	15	-	25	no
Product	II Violation	-	12	120.000	5	2	5	yes

SCHEDULE

[HLS 200-880] LINK The II Violation in module 'matrix_mult' (loop 'Product'): Unable to enforce a carried dependence constrair operation ('prod_addr_write_ln16', ../matrix_mult.cpp:16) of variable 'add_ln16', ../matrix_mult.cpp:16 on on array 'prod'.

[HLS 200-1470] Pipelining result : Target II = 1, Final II = 2, Depth = 5, loop 'Product'



[Vitis HLS 2020.2]

Removing "volatile"

matrix_mult.cpp × | Synthesis Summary(prod_pipeline)

```
#include "matrix_mult.h"

void matrix_mult(
      mat_a a[IN_A_ROWS][IN_A_COLS],
      mat_b b[IN_B_ROWS][IN_B_COLS],
      mat_prod prod[IN_A_ROWS][IN_B_COLS])
{
  // Iterate over the rows of the A matrix
   Row: for(int i = 0; i < IN_A_ROWS; i++) {
      // Iterate over the columns of the B matrix
      Col: for(int j = 0; j < IN_B_COLS; j++) {
         prod[i][j] = 0;
         // Do the inner product of a row of A and col of B
         Product: for(int k = 0; k < IN_B_ROWS; k++) {
            prod[i][j] += a[i][k] * b[k][j];
         }
      }
   }

}
```

for 5x5 matrices

Outline | Directive ×

- matrix_mult
  - HLS TOP name=matrix_mult
  - a
  - b
  - prod
  - Row
    - HLS PIPELINE off
    - Col
      - HLS PIPELINE off
      - Product
        - HLS PIPELINE

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
matrix_mult		-	130	1.300E3	-	131	-	no
Row_Col_Product		-	128	1.280E3	5	1	125	yes

[Vitis HLS 2020.2]

Inferring Accumulation Register


```
int temp; // carries depedency
for(int cnt=0; cnt<125; cnt++) {
   int i=cnt/25, j=(cnt/5)%5, k=cnt%5;
   if (k==0) temp=C[i][j];
   temp += A[i][k]*B[k][j];
   C[i][j]=temp;
}
```

safe if no one else changing C[][]

[Vitis HLS 2020.2]

Letting Vitis HLS just do its own thing

matrix_mult.cpp × | Synthesis Summary(prod_pipeline)

```
1
2  #include "matrix_mult.h"
3
4  void matrix_mult(
5        mat_a a[IN_A_ROWS][IN_A_COLS],
6        mat_b b[IN_B_ROWS][IN_B_COLS],
7        mat_prod prod[IN_A_ROWS][IN_B_COLS])
8  {
9    // Iterate over the rows of the A matrix
10   Row: for(int i = 0; i < IN_A_ROWS; i++) {
11      // Iterate over the columns of the B matrix
12      Col: for(int j = 0; j < IN_B_COLS; j++) {
13         prod[i][j] = 0;
14         // Do the inner product of a row of A and col of B
15         Product: for(int k = 0; k < IN_B_ROWS; k++) {
16            prod[i][j] += a[i][k] * b[k][j];
17         }
18      }
19   }
20
21 }
22
```

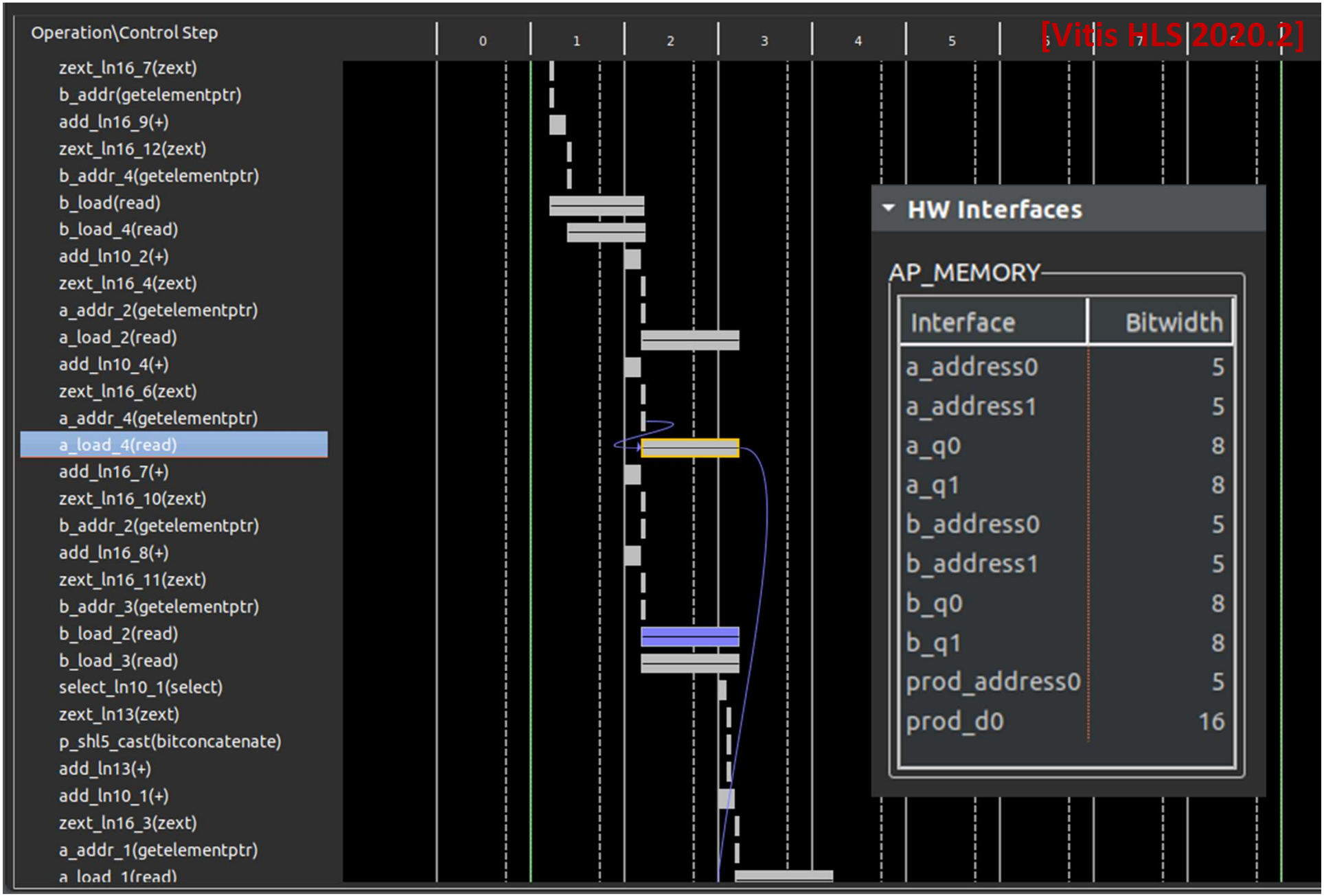
for 5x5 matrices

Outline | Directive ×

- matrix_mult
  - HLS TOP name=matrix_mult
  - a
  - b
  - prod
  - Row
    - Col
      - Product

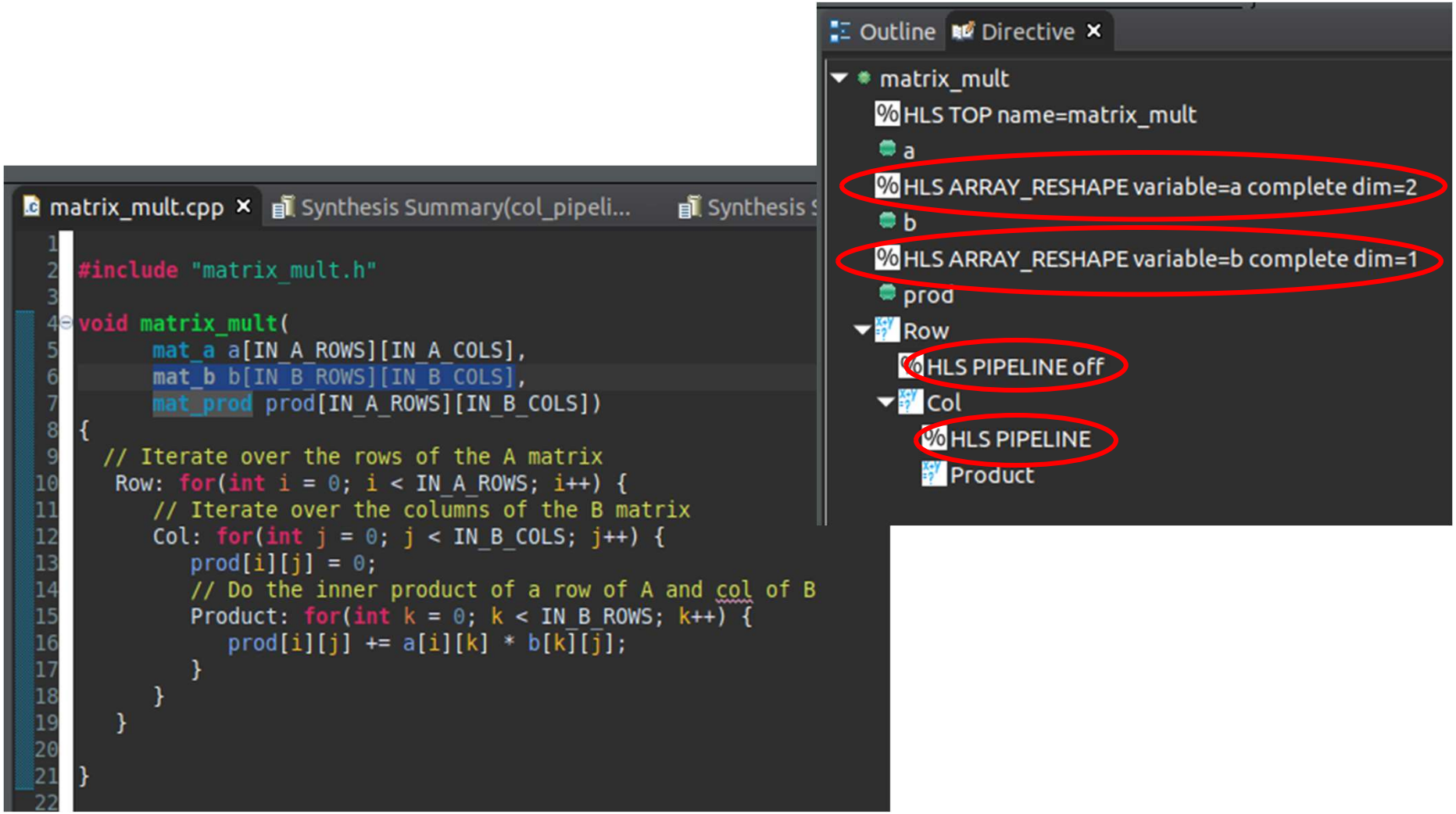
Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
matrix_mult	II Violation	-	80	800.000	-	81	-	no
Row_Col	II Violation	-	78	780.000	7	3	25	yes

product?


Operation\Control Step
0
1
2
3
4
5
zext_ln16_7(zext)
b_addr(getelementptr)
add_ln16_9(+)
zext_ln16_12(zext)
b_addr_4(getelementptr)
b_load(read)
b_load_4(read)
add_ln10_2(+)
zext_ln16_4(zext)
a_addr_2(getelementptr)
a_load_2(read)
add_ln10_4(+)
zext_ln16_6(zext)
a_addr_4(getelementptr)
a_load_4(read)
add_ln16_7(+)
zext_ln16_10(zext)
b_addr_2(getelementptr)
add_ln16_8(+)
zext_ln16_11(zext)
b_addr_3(getelementptr)
b_load_2(read)
b_load_3(read)
select_ln10_1(select)
zext_ln13(zext)
p_shl5_cast(bitconcatenate)
add_ln13(+)
add_ln10_1(+)
zext_ln16_3(zext)
a_addr_1(getelementptr)
a_load_1(read)
[Vitis HLS 2020.2]
HW Interfaces
AP_MEMORY
Interface
Bitwidth
a_address0 5
a_address1 5
a_q0 8
a_q1 8
b_address0 5
b_address1 5
b_q0 8
b_q1 8
prod_address0 5
prod_d0 16

[Vitis HLS 2020.2]

Optimizing Memory Layout



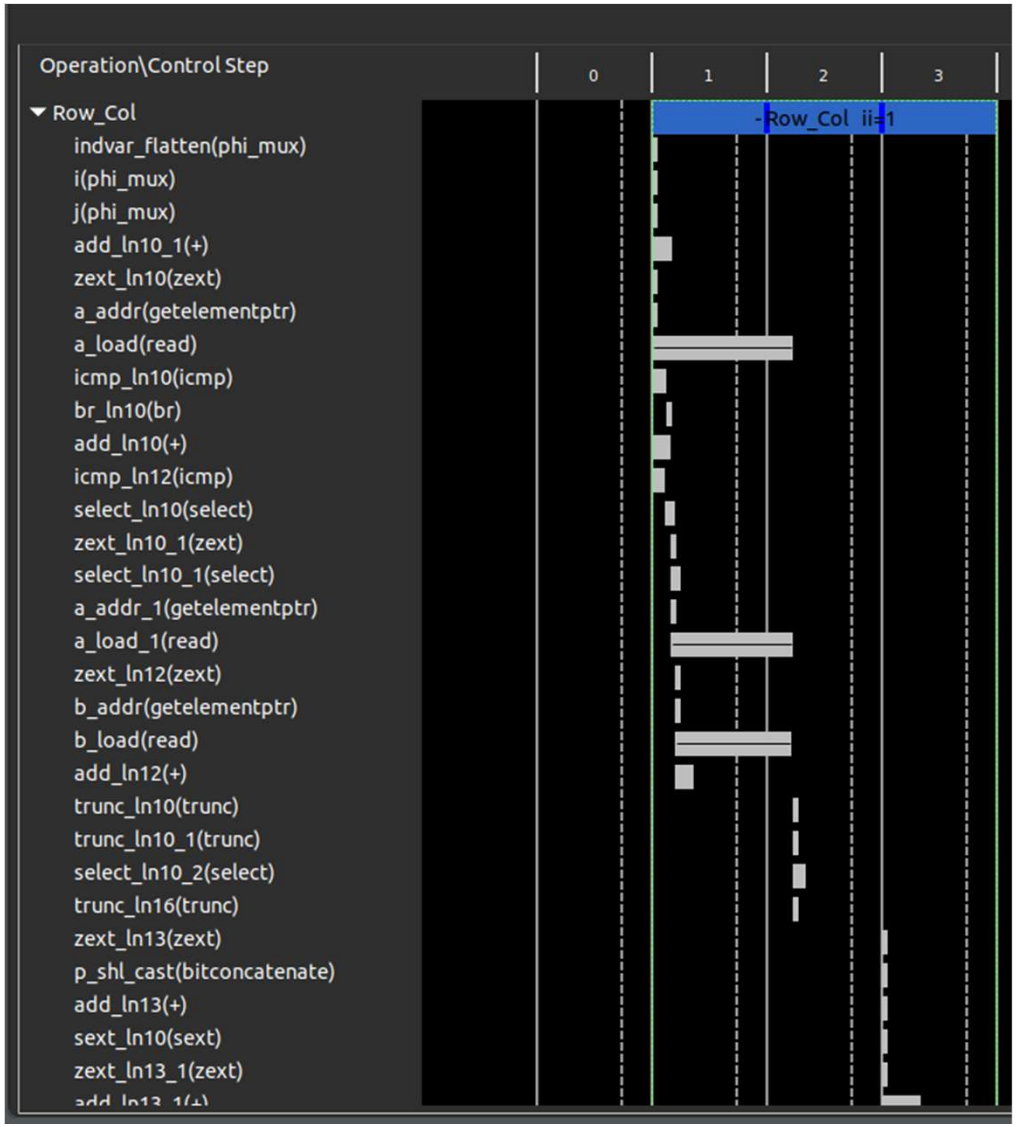
Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
matrix_mult	-		28	280.000	-	29	-	no
Row_Col	-		26	260.000	2	1	25	yes

HW Interfaces

AP_MEMORY

Interface	Bitwidth
a_address0	3
a_address1	3
a_q0	64
a_q1	64
b_address0	3
b_q0	64
prod_address0	5
prod_d0	16

?



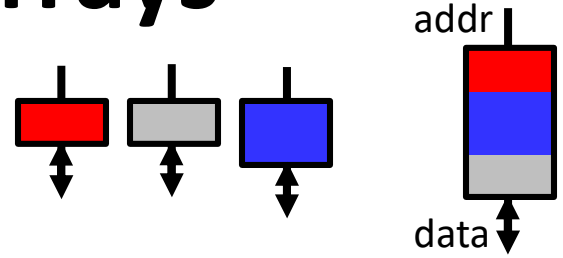
[Vitis HLS 2020.2]

- 25x speedup off same code

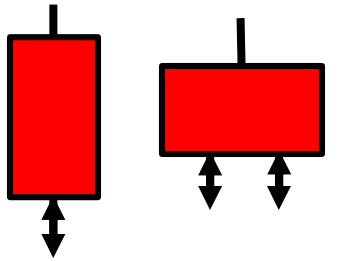
- No new functional bugs!!

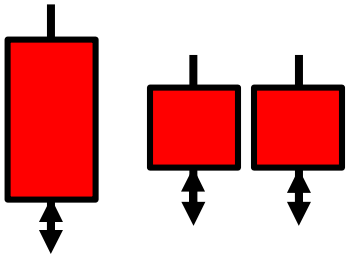
- Can you do better by RTL?

Pragma Crib Sheet: Arrays



- Map
  - multiple arrays in same BRAM
  - no perf loss if no scheduling conflicts
- Reshape
  - change BRAM aspect ratio to widen ports
  - change linear address to location mapping
  - higher bandwidth on consecutive locations
- Partition
  - map 1 array to multiple BRAMs
  - multiple independent ports if no bank conflicts

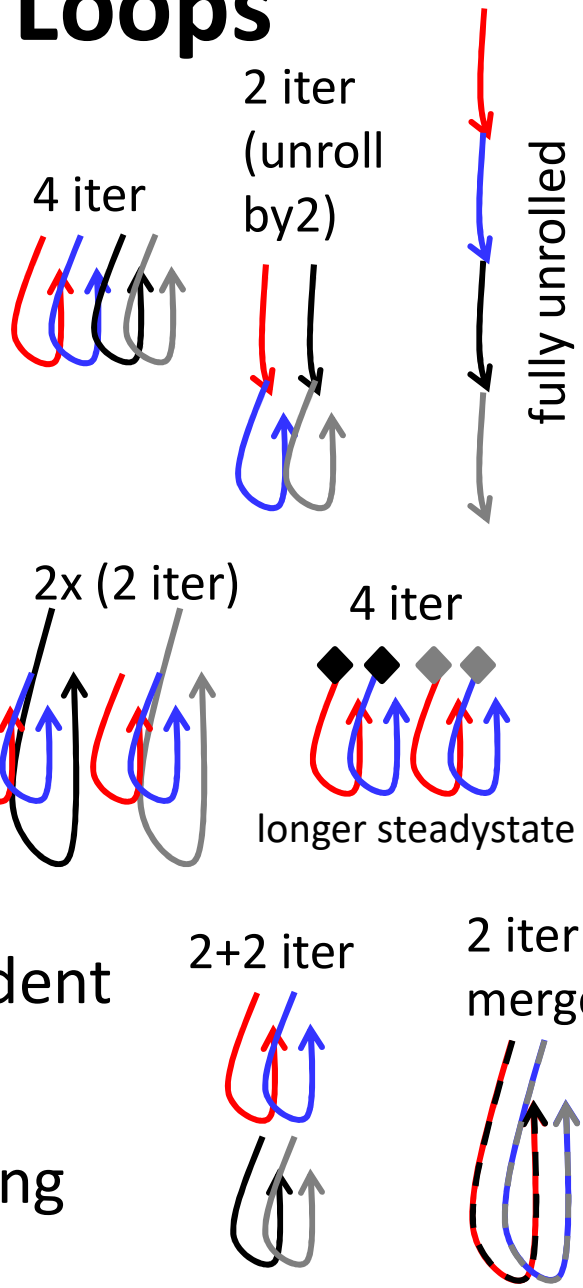




A lot more you can control; must read UG902

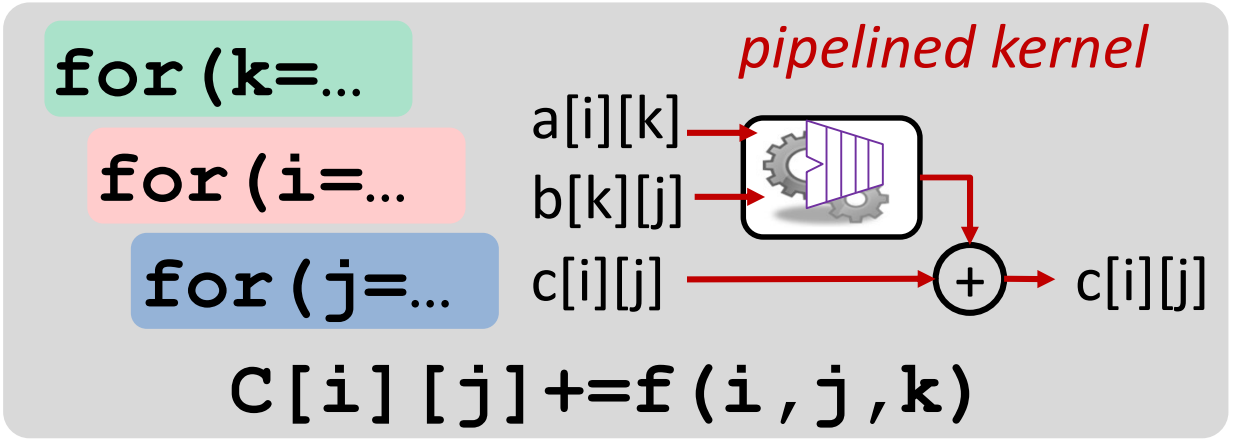
Pragma Crib Sheet: Loops

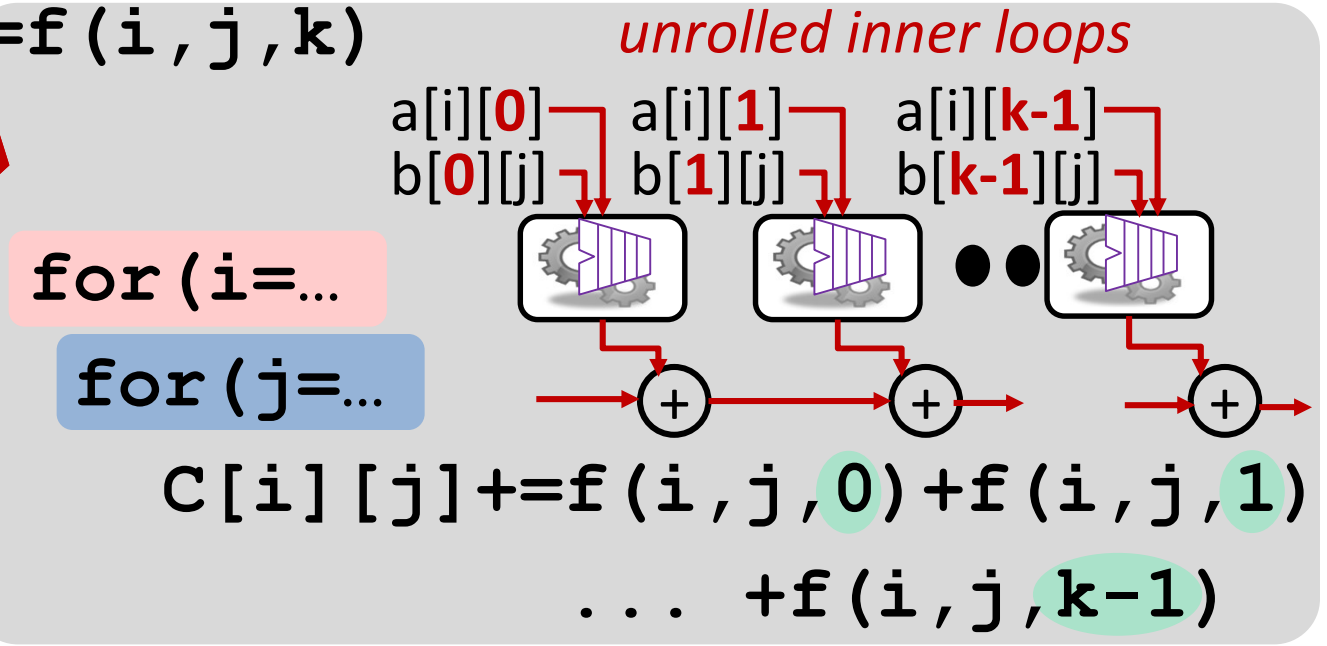
- Loop Unroll (full and partial)
  - amortize loop control overhead
  - increase loop-body size, hence "ILP" and scheduling flexibility
- Loop Flatten
  - streamline loop-nest control
  - reduce start/finish stutter
- Loop Merge
  - combine loop-bodies of independent loops of same control
  - improve parallelism and scheduling



Additional Control thru Code Structure

```
for(i=…
  for(j=…
    for(k=…
      C[i][j]+=f(i,j,k)
```





Last Time

Don't Forget HW Basics

- Literal (forced)
  686cyc, 6860ns (1x)

Target	Estimated	Uncertainty	BRAM	DSP	FF	LUT
10.00 ns	1.878 ns	2.70 ns	0	1	39	214

- Vitis Default

  80cyc, 800ns (8.6x)

Target	Estimated	Uncertainty	BRAM	DSP	FF	LUT
10.00 ns	3.772 ns	2.70 ns	0	3	128	467

- Pragma Directed

  28cyc, 280ns (25x)

Target	Estimated	Uncertainty	BRAM	DSP	FF	LUT
10.00 ns	4.195 ns	2.70 ns	0	0	24	190

- Clock go fast . . .

  29cyc, 116ns (60x)

Target	Estimated	Uncertainty	BRAM	DSP	FF	LUT
4.00 ns	2.170 ns	1.08 ns	0	0	132	231

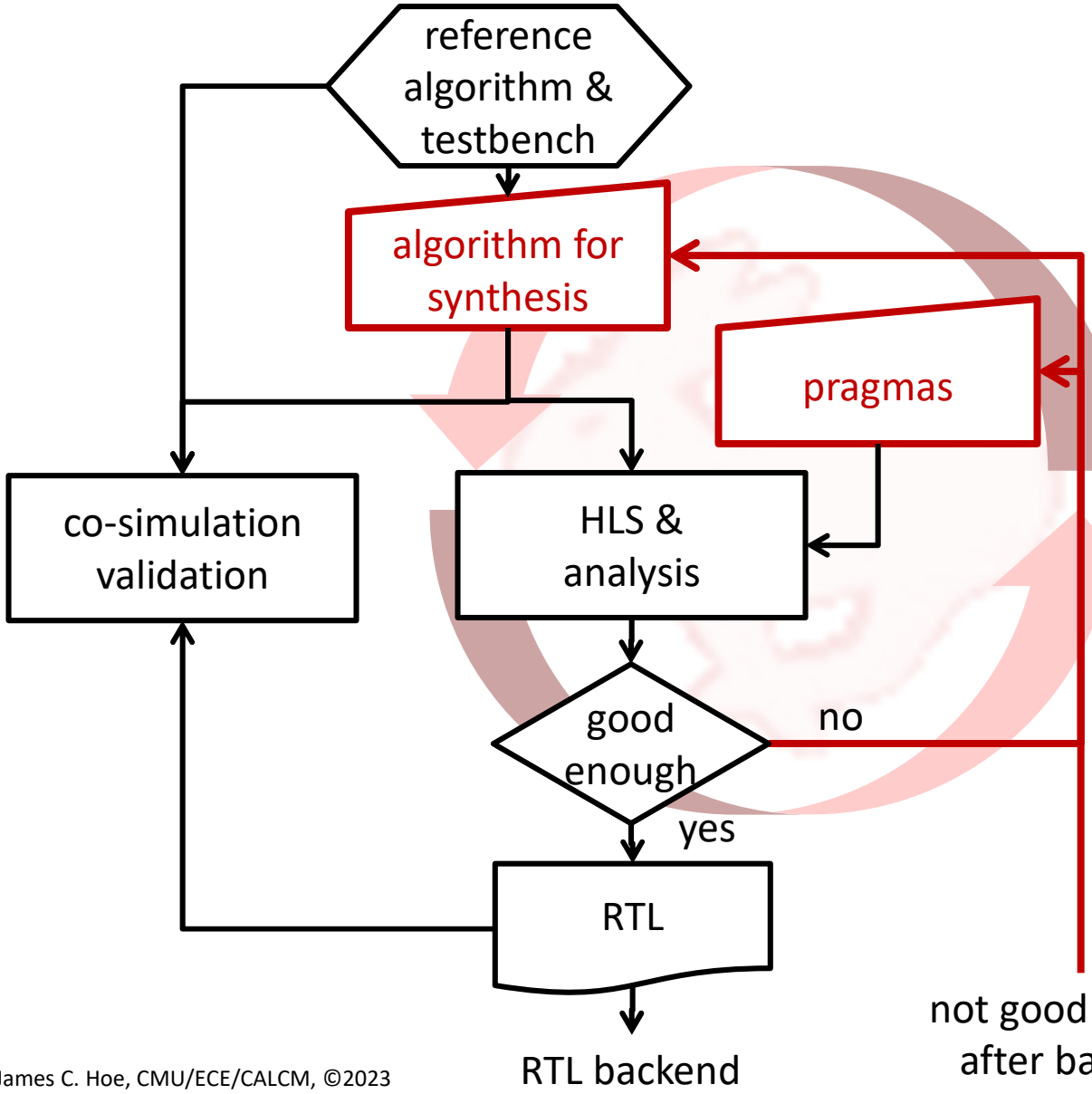
29cyc, 87ns (79x)

Target	Estimated	Uncertainty	BRAM	DSP	FF	LUT
3.00 ns	2.170 ns	0.81 ns	0	0	250	295

57cyc, 114ns (60x)

Target	Estimated	Uncertainty	BRAM	DSP	FF	LUT
2.00 ns	1.287 ns	0.54 ns	0	1	211	263

Design by Exploration



When this takes only minutes, a little trial-and-error is okay (just a little!!!!)

Putting it in context (from last time)

- For you to produce "good" structural RTL
  - identify suitable "temporal and spatial pattern"
  - flesh out concrete datapath (bit/cycle exact)
  - develop correct and efficient control sequencing
- C-to-HW (i.e., C-to-RTL) compiler bridges the gap between functionality and implementation
  - extract parallelism from a sequential specification
  - fill in the details below the functional abstraction
  - make good decisions when filling in the details

Vitis HLS does its part (under your direction)

fast and without mistakes

Parting Thoughts

- Vitis HLS doesn't turn program into HW
- Vitis HLS doesn't turn programmer into HW designer
- Multifaceted benefits to HW designer
  - algo. development/debug/validate in SW
  - pragma steering (no RTL hacking, machine tuning)
  - fast analysis and visualization
  - data type support

    it is about more than adding "double" to Verilog
  - built-in, stylized IP interfaces
  - integration with the rest of Vitis and Zynq!!

Can we turn HPC programmers into HW designers?