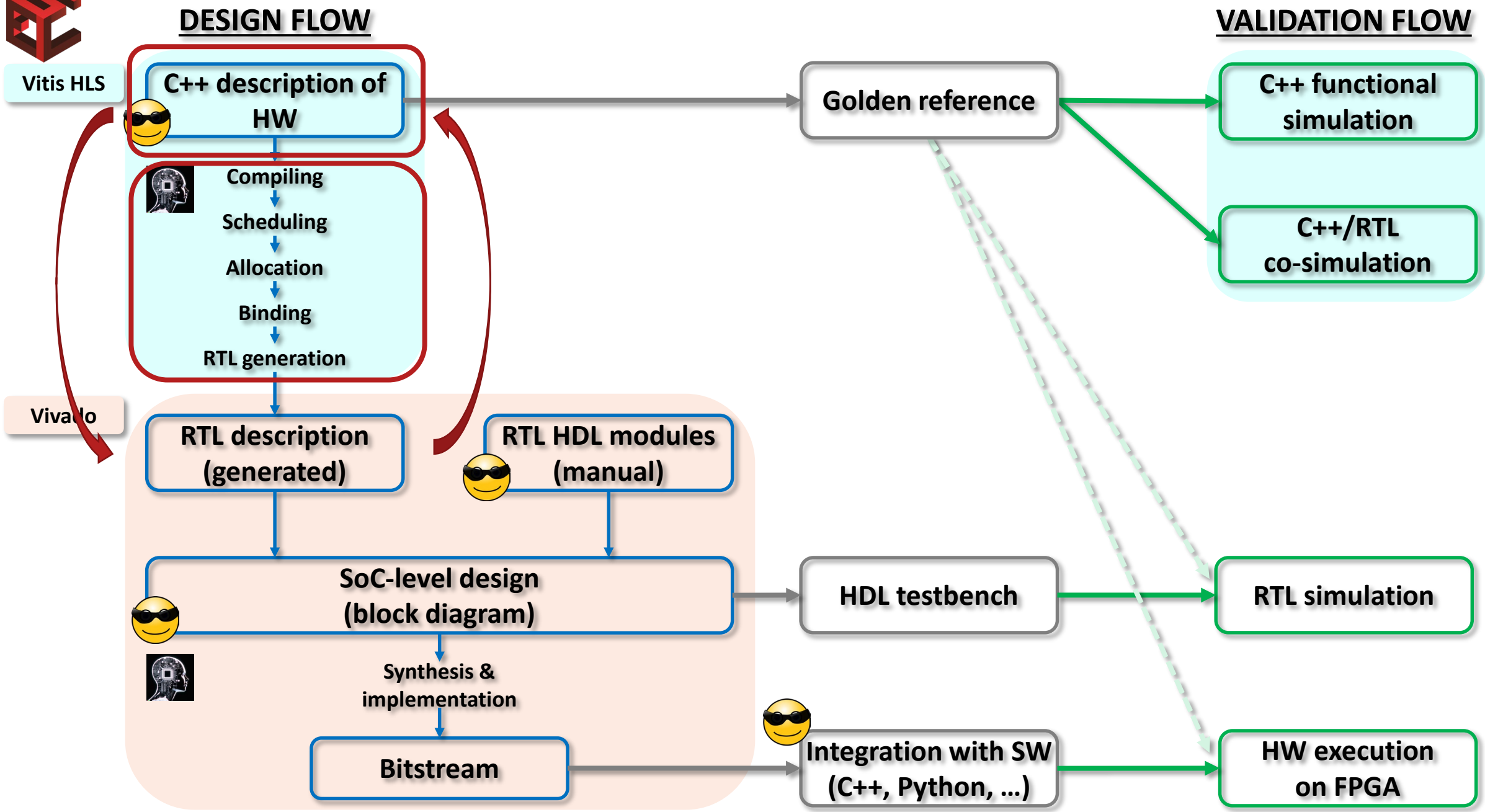
EPFL

Lab. On HW-SW Digital Systems Codesign
EE-390(a)

Session 3
Design exploration & optimization using HLS

Prof. David Atienza
Dr. Denisa Constantinescu, Dr. Miguel Peón-Quirós,
Mr. Rubén Rodríguez-Álvarez, Ms. Stasa Kostic, Mr. Karan Pathak

High-level synthesis (HLS) design flow for HW accelerators

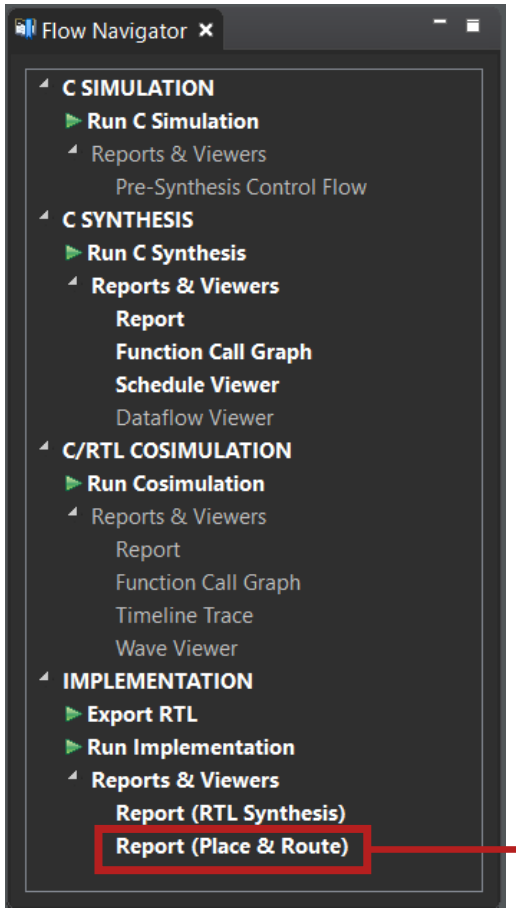


Learning Objectives

- How to perform a **Design Space Exploration** using Vitis HLS
- How to **optimize loops** in Vitis HLS
- Understand the **HLS workflow**: how C/C++ code is synthesized into RTL logic

Design Space Exploration (DSE)

- Metrics:
  - Latency
  - Throughput
  - Energy
  - Resources (Area)



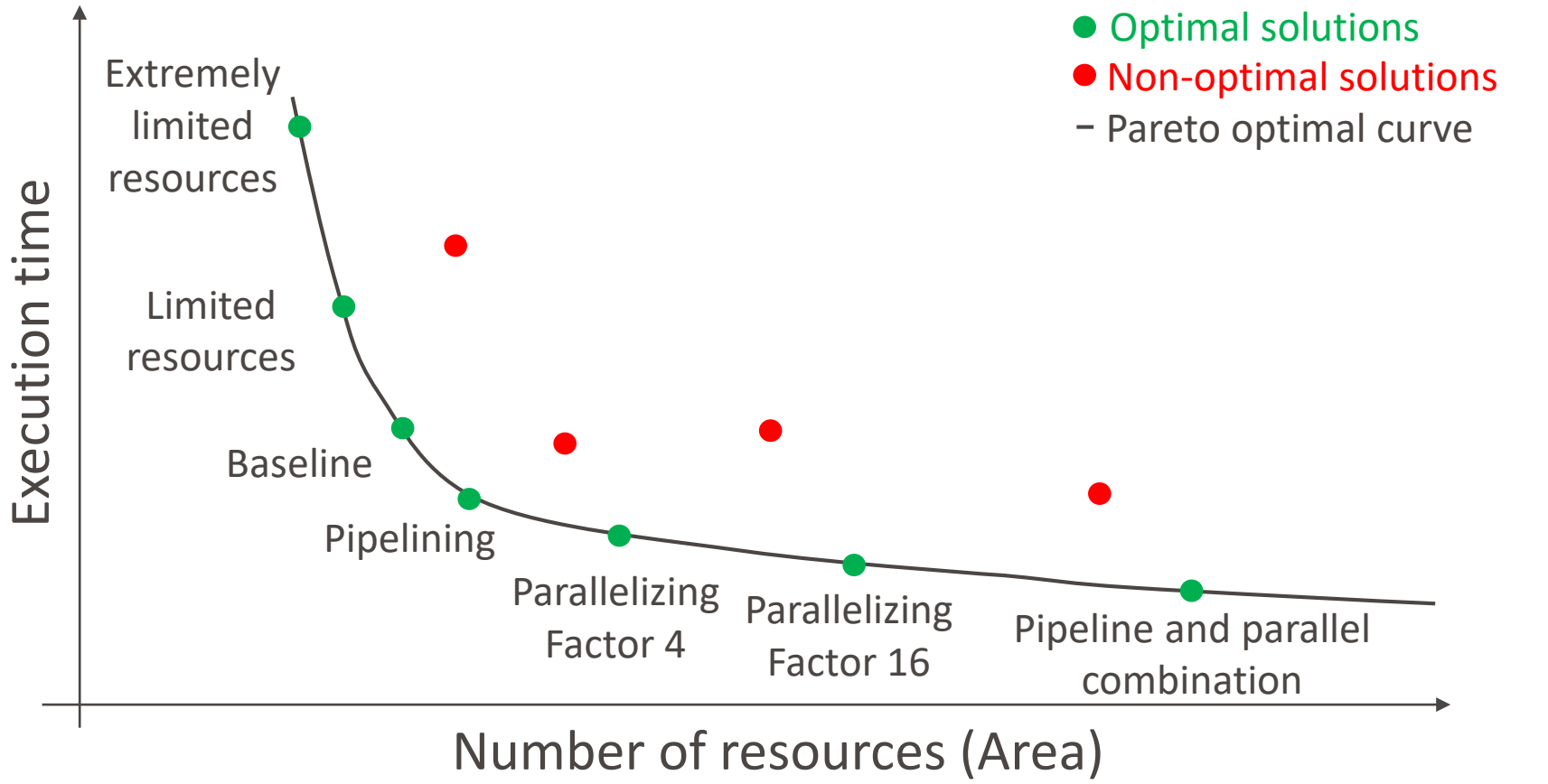
Resource Usage

	Verilog
SLICE	741
LUT	1634
FF	2592
DSP	0
BRAM	2
URAM	0
LATCH	0
SRL	142
CLB	0

Final Timing

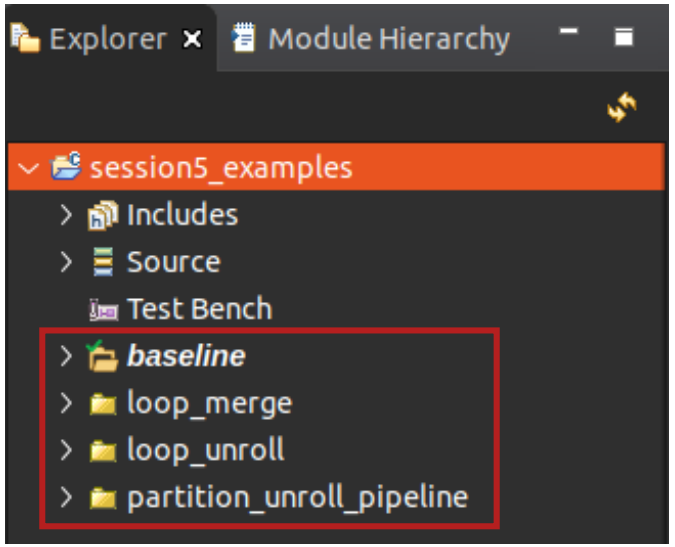
	Verilog
CP required	10.000
CP achieved post-synthesis	6.747
CP achieved post-implementation	7.151

Timing met

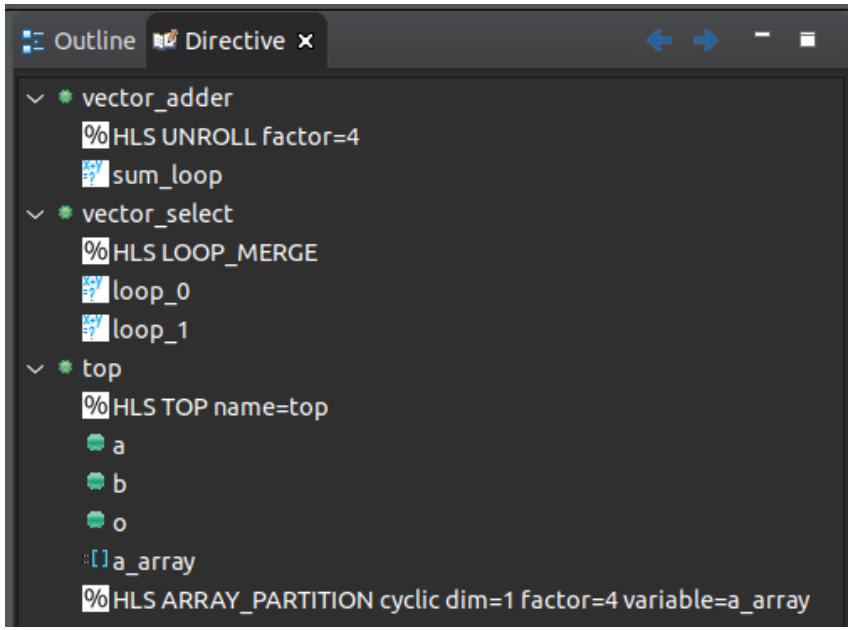


DSE in Vitis

Vitis HLS Solutions



Solution in Vitis HLS share the same code but apply different optimizations through the **directives**. These solutions can present resources-performance trade-off in the **Design Space**.



Post-synthesis reports:

- Resources: Implementation table
- Timings: Timings summary
- Solutions: Comparative report

Build the DSE

Optimization types

// source: accelerator.cpp
pragma OPTIMIZATION OPTION=2
e.g. INTERFACES

// source: directives.tcl
set_directive_optimization [OPTIONS]
e.g. Parallelization with different factors

// source: setup.tcl
set_command_optimization [OPTIONS]
e.g. Burst transaction width

HLS optimizations

- There is a full list of pragmas and directives optimizations.
- The most important ones are:
  - Loops optimizations
    - Pipelining
    - Unrolling
    - Merging
    - Flattening
  - Array optimizations (Next week with Denisa)
    - Partitioning
    - Reshaping
    - Type of data storage
  - Dataflow

Loops optimizations: Unrolling

Not unrolled (baseline)

```
Loop_1: for(int i = 0; i  < LENGTH; i++) {
   o[i] = a[i] + b[i];
}
```

Factor=2

```
Loop_1: for(int i = 0; i  < LENGTH; i+=2) {
   o[i] = a[i] + b[i];
   o[i+1] = a[i +1] + b[i +1];
}
```

Factor=4

```
Loop_1: for(int i = 0; i  < LENGTH; i+=4) {
   o[i] = a[i] + b[i];
   o[i+1] = a[i +1] + b[i +1];
   o[i+2] = a[i +2] + b[i +2];
   o[i+3] = a[i +3] + b[i +3];
}
```

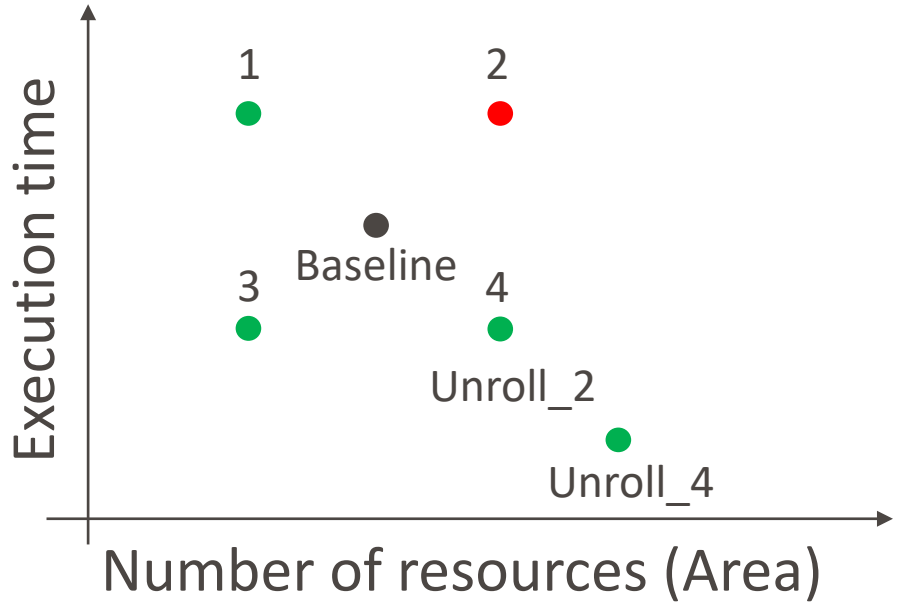
Timing

Latency

		baseline	loop_unroll_5	loop_unroll_10
Latency (cycles)	min	1002	202	102
	max	1002	202	102
Latency (absolute)	min	10.020 us	2.020 us	1.020 us
	max	10.020 us	2.020 us	1.020 us
Interval (cycles)	min	1003	203	103
	max	1003	203	103

Resources

	baseline	loop_unroll_5	loop_unroll_10
BRAM_18K	0	0	0
DSP	0	0	0
FF	45	97	137
LUT	272	1182	2294
URAM	0	0	0



Loops optimizations: Merge

Separated loops (baseline)

```
loop_0: for(int i = 0; i  < LENGTH; i++) {
    if(sel[i])
        o[i] = a[i] + b[i];
}
loop_1: for(int i = 0; i  < LENGTH; i++) {
    if(!sel[i])
        o[i] = a[i] - b[i];
}
```

Merged loops

```
loop_0: for(int i = 0; i  < LENGTH; i++) {
    if(sel[i])
        o[i] = a[i] + b[i];
     if(!sel[i])
        o[i] = a[i] - b[i];
}
```

Latency

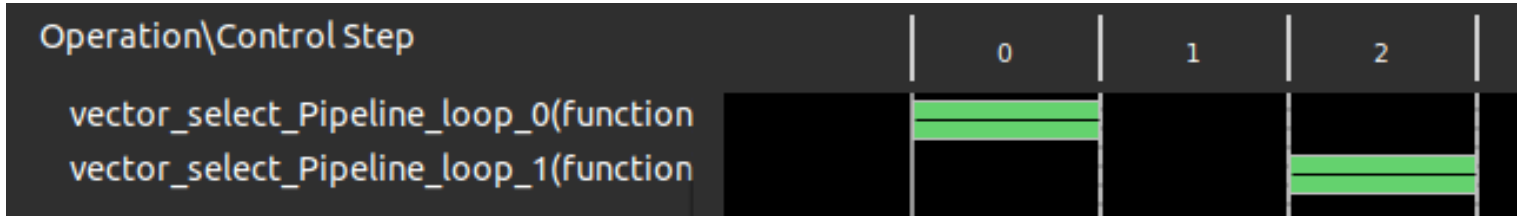
		baseline	loop_merge
Latency (cycles)	min	2009	1003
	max	2009	1003
Latency (absolute)	min	20.090 us	10.030 us
	max	20.090 us	10.030 us
Interval (cycles)	min	2010	1004
	max	2010	1004

Utilization Estimates

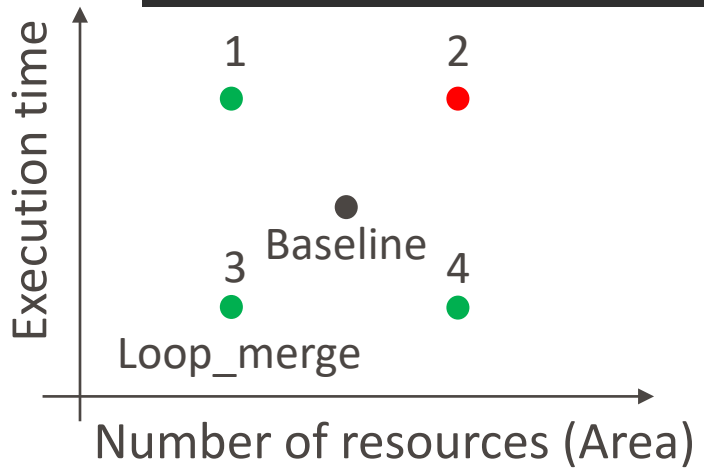
	baseline	loop_merge
BRAM_18K	0	0
DSP	0	0
FF	78	36
LUT	403	218
URAM	0	0

Scheduler view

baseline



Merged



Loops optimizations: Flattening

Only perfect and imperfect loops can be flattened

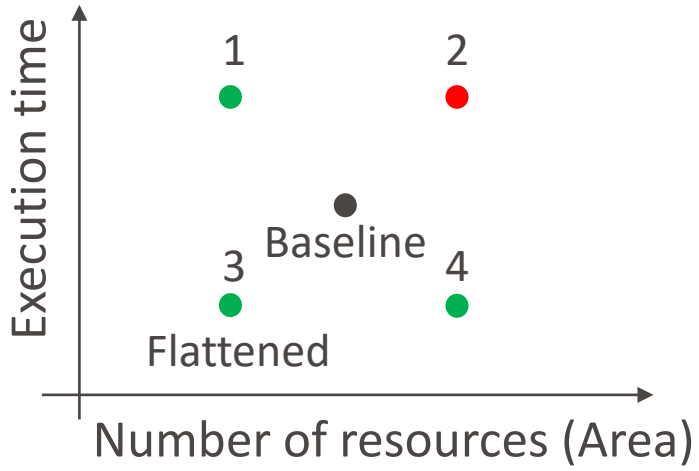
- Perfect nested loops
  - Logic code only on innermost loop
  - Loop bounds fixed
- Semi-perfect nested loops
  - Logic code only on innermost loop
  - Only outer loop bound variable (the rest are fixed)
- Imperfect Nested loops
  - Loops for which the previous conditions are not met

Separated loops (baseline)

```
loop_i: for(int i = 0; i  < LENGTH; i++) {
   loop_j: for(int j = 0; j  < WIDTH; j++) {
      acc += A[i * WIDTH + j];
   }
}
```

Flattened

```
loop_i_j: for(int i = 0; i  < LENGTH * WIDTH; i++) {
   acc += A[i];
}
```



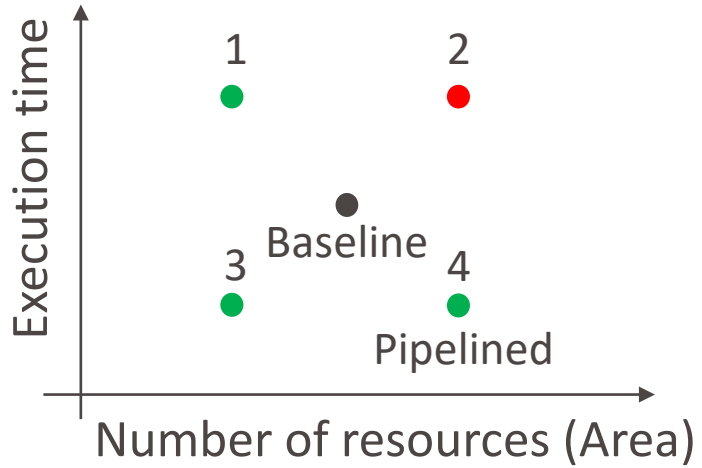
*Note: Use **LOOP_TRIPCOUNT** pragma or directive in **variable bounded loops** to allow Vitis HLS to report a minimum and maximum number of iterations. If not done, Vitis HLS will be unable to estimate the **total execution time**.*

Loops optimizations: Pipelining

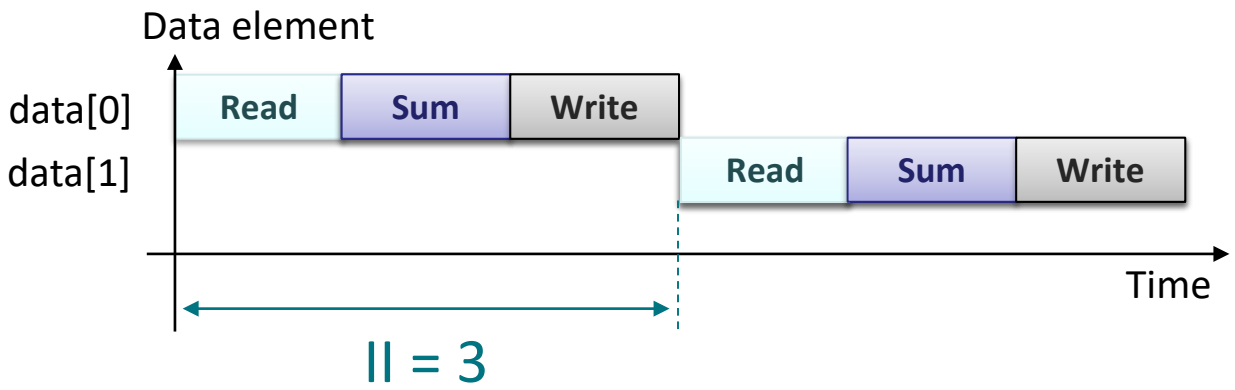
- The **Iteration Interval (II)** represents the number of cycles there is between two consecutives elements of data at the output. It measures the **throughput** of the function being analyzed.

Example code

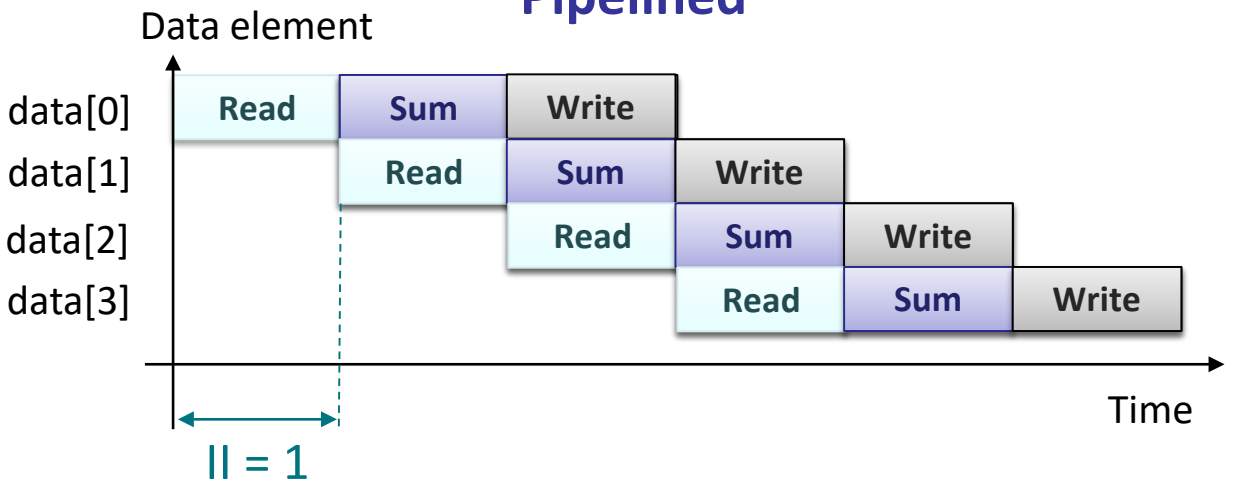
```
ap_uint<32> a[LENGTH];
ap_uint<32> b[LENGTH];
ap_uint<32> c[LENGTH];

loop_1: for(int i = 0; i  < LENGTH; i++) {
   o[i] = a[i] + b[i];
}
```



Not pipelined



Pipelined



Synthesis Flow

C++ description

```
int temp = 0;
for( int i=0 ; i<100 ; i++ ) {
   temp = (a[i] + b[i]) * temp + (b[i] * c [i]);
}
```

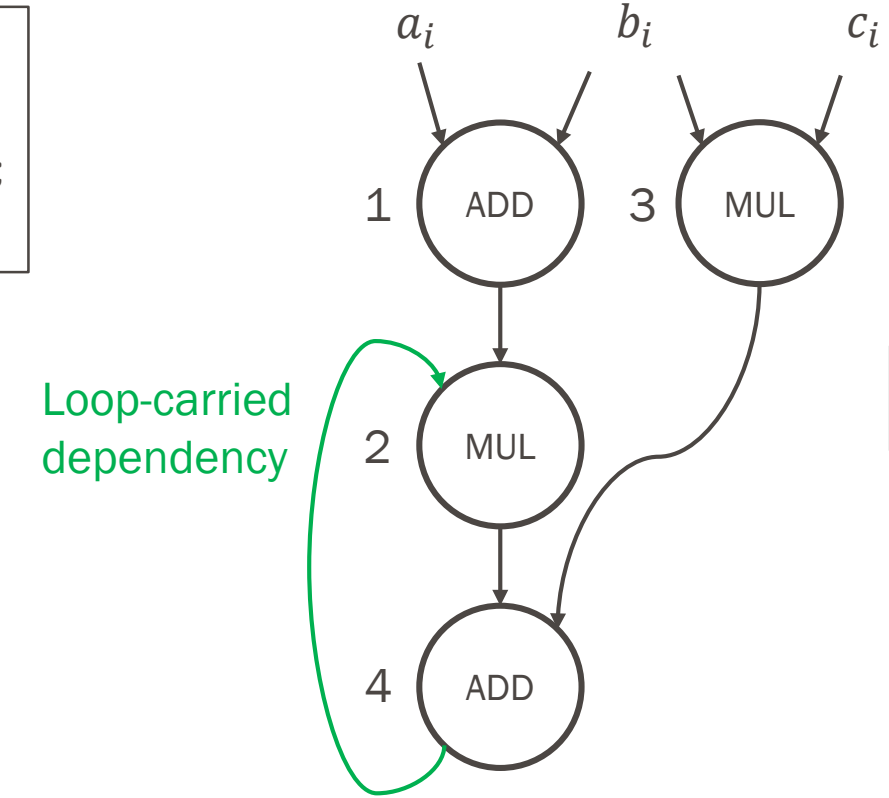
Assume:

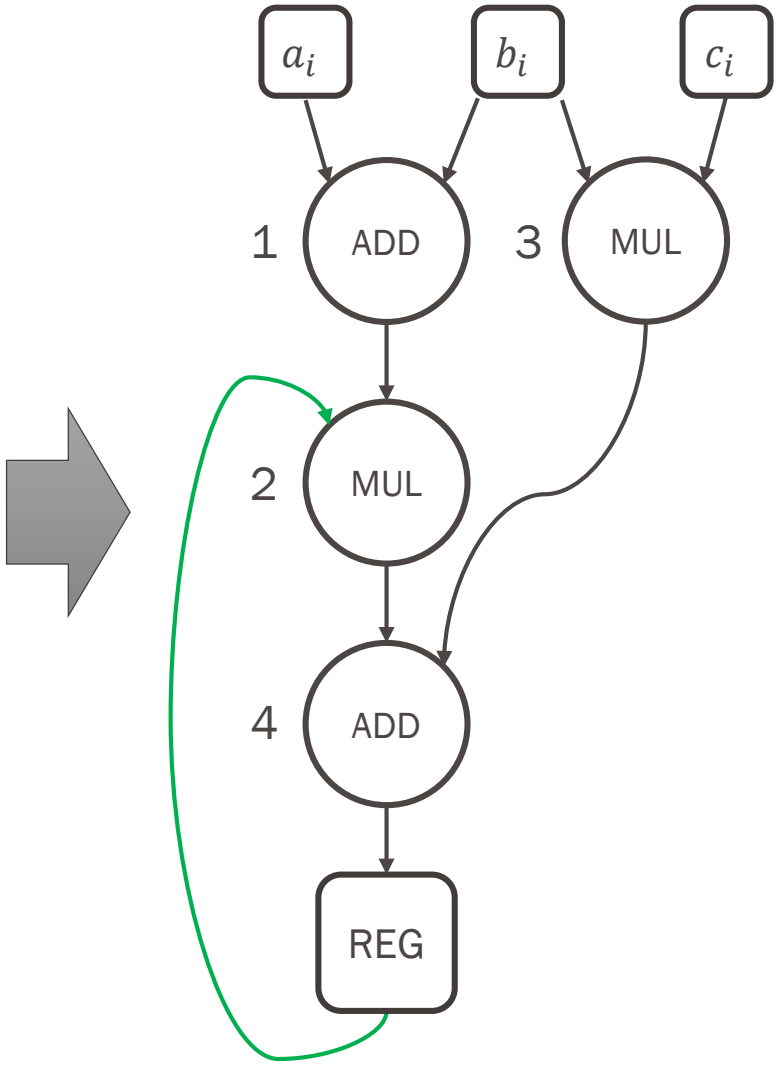
$t_A = 6ns$

$t_M = 9ns$

$t_R = 0ns$

Data Flow Graph





Critical path: t_{Path} =?

Iteration Latency: ?

Initiation Interval: ?

Synthesis Flow

C → DFG → Scheduling → Allocation → Binding → RTL

C++ description

```
int temp = 0;
for( int i=0 ; i<100 ; i++ ) {
   temp = (a[i] + b[i]) * temp + (b[i] * c [i]);
}
```

Assume:

$t_A = 6ns$

$t_M = 9ns$

$t_R = 0ns$

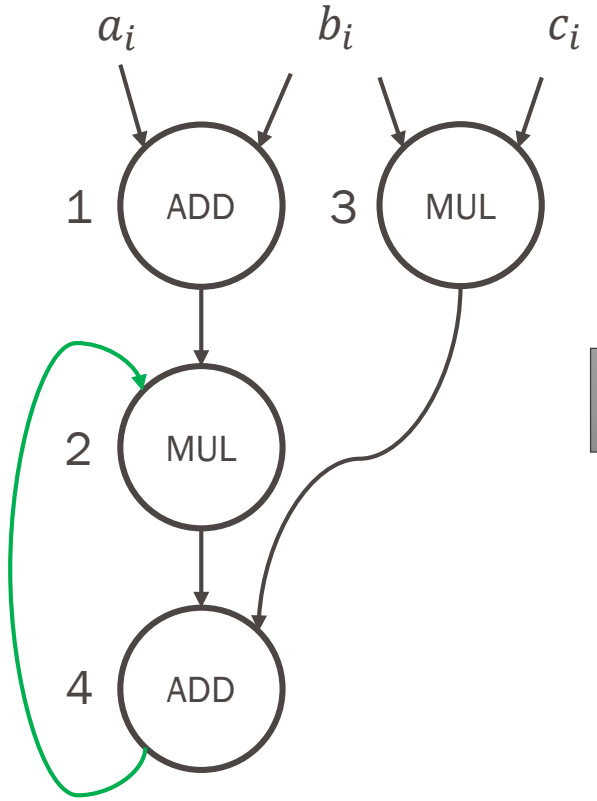
Timings model:

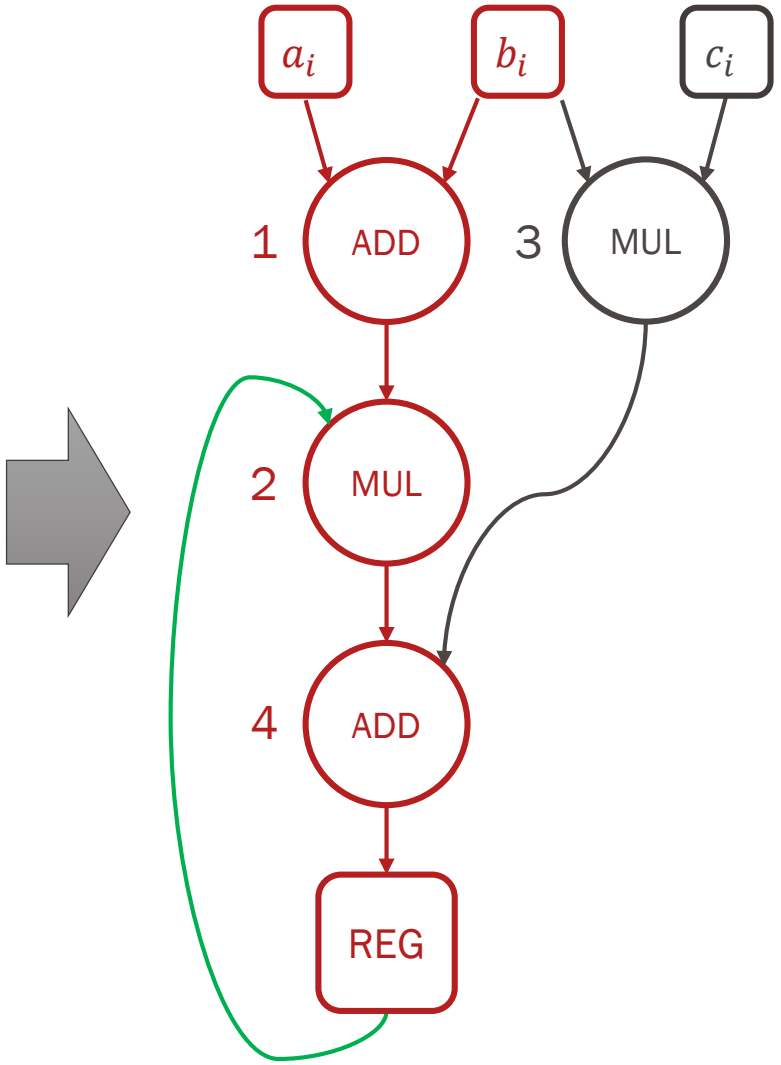
$T_L = \#iterations * II * T_{clk}$

Resources model:

$C_R = 2 * \#ADDs + 3 * \#MULs + 1 * \#REGs$

Data Flow Graph





Critical path: $t_{Path} = 21ns$

Iteration Latency: $L_I = 1\ cycle$

Initiation Interval: $II = 1\ cycle$

Solution: $L_I = 1\ cycle$, II = 1 cycle, $T_{clk} = 21ns$

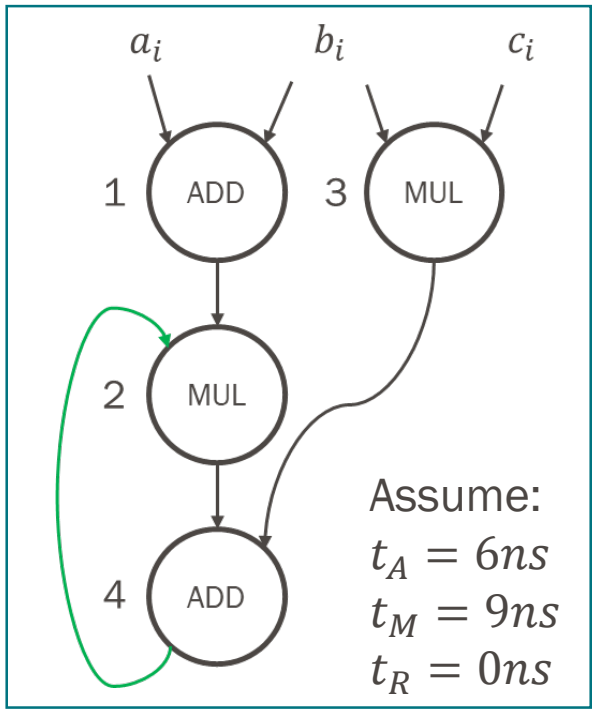
Total execution time	Resources cost
$T_L = 100 * 3 * 21ns = \mathbf{2100ns}$	$C_R = 2 * 2 + 3 * 2 + 1 * 1 = \mathbf{11}$

Synthesis Flow

C → DFG → Scheduling → Allocation → Binding → RTL

Solution: $T_{clk} = 10ns$ ASAP with $II = 3$

DFG



T_{clk}	10				
II	3				
ADD	1	MUL	1	REG	3
T_L	$100 * 3 * 10ns = \mathbf{3000ns}$				
C_R	$C_R = 2 * 1 + 3 * 1 + 1 * 3 = \mathbf{8}$				

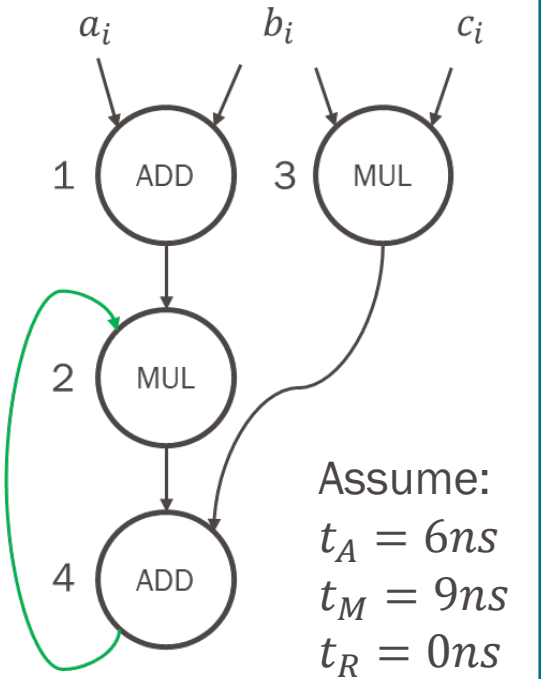
Step	Scheduling	Allocation		
		ADD	MUL	REG
S0	1 ADD, 3 MUL	1	1	3
S1	2 MUL	0	1	2
S2	4 ADD	1	0	1
S3	1 ADD, 3 MUL			
S4	2 MUL			
S5	4 ADD			

Synthesis Flow

C → DFG → Scheduling → Allocation → Binding → RTL

Solution: $T_{clk} = 10ns$ ALAP with $II = 3$

DFG



T_{clk}	10				
II	3				
ADD	1	MUL	2	REG	2
T_L	$100 * 3 * 10ns = \mathbf{3000ns}$				
C_R	$C_R = 2 * 1 + 3 * 2 + 1 * 2 = \mathbf{10}$				

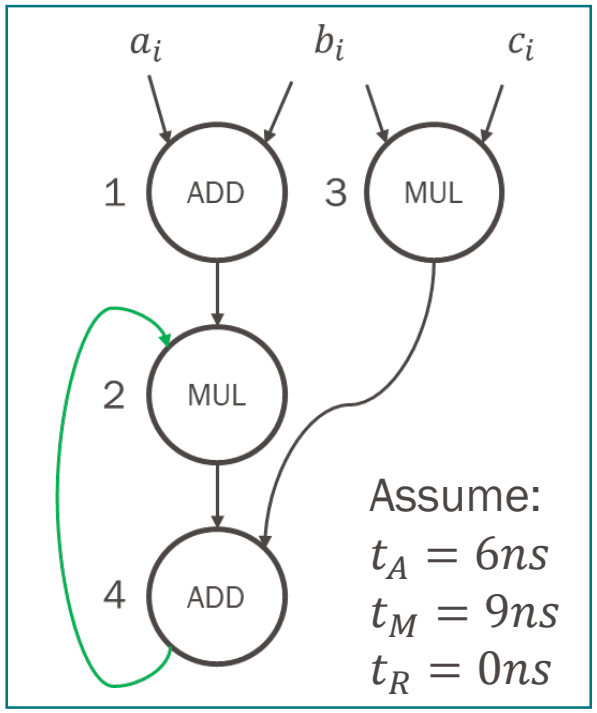
Step	Scheduling	Allocation		
		ADD	MUL	REG
S0	1 ADD (a_i, b_i)	1	0	2
S1	2 MUL (b_i, c_i); 3 MUL (b_i, c_i)	0	2	2
S2	4 ADD	1	0	1
S3	1 ADD (a_i, b_i)			
S4	2 MUL (b_i, c_i); 3 MUL (b_i, c_i)			
S5	4 ADD			

Synthesis Flow

C → DFG → Scheduling → Allocation → Binding → RTL

Solution: $T_{clk} = 10ns$ ASAP with $II = 2$

DFG



T_{clk}	10				
II	2				
ADD	2	MUL	1	REG	3
T_L	$100 * 2 * 10ns = \mathbf{2000ns}$				
C_R	$C_R = 2 * 1 + 3 * 2 + 1 * 2 = \mathbf{10}$				

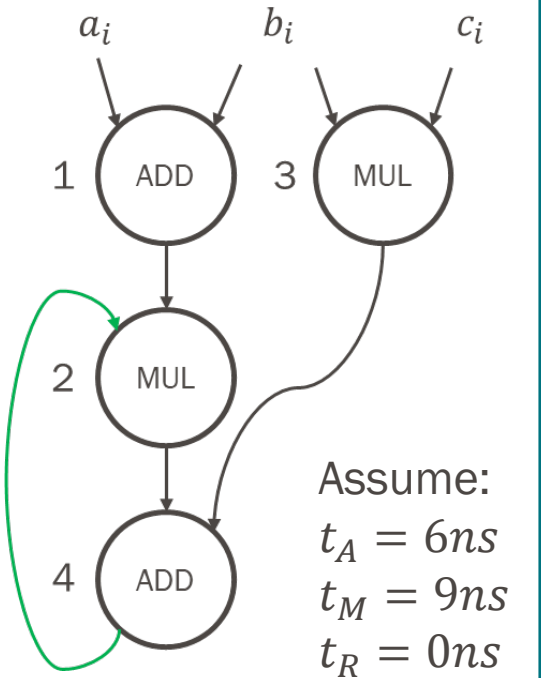
Step	Scheduling	Allocation		
		ADD	MUL	REG
S0	a_i b_i c_i — 1 ADD, 3 MUL			
S1	2 MUL	0	1	2
S2	4 ADD; a_i b_i c_i — 1 ADD, 3 MUL	2	1	3
S3	2 MUL			
S4	4 ADD			
S5				

Synthesis Flow

C → DFG → Scheduling → Allocation → Binding → RTL

Solution: $T_{clk} = 10ns$ ALAP with $II = 2$

DFG



T_{clk}	10				
II	2				
ADD	2	MUL	2	REG	2
T_L	$100 * 2 * 10ns = \mathbf{2000ns}$				
C_R	$C_R = 2 * 1 + 3 * 2 + 1 * 2 = \mathbf{12}$				

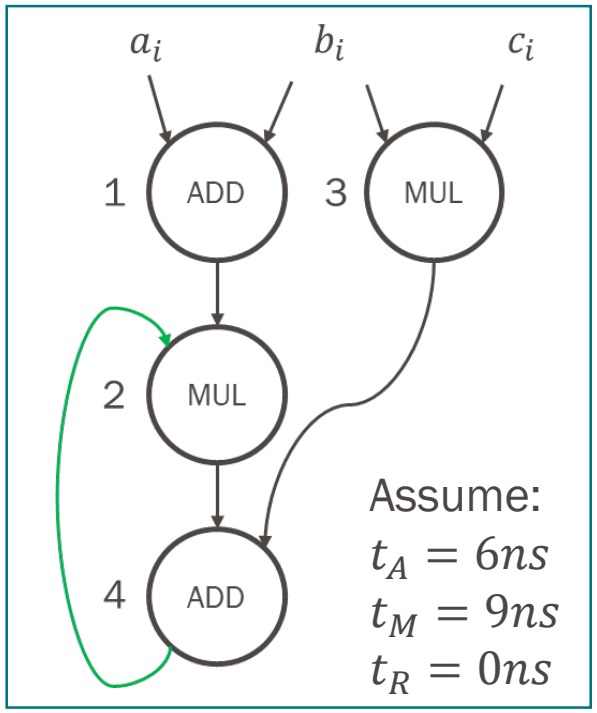
Step	Scheduling	Allocation		
		ADD	MUL	REG
S0	1 ADD (a_i, b_i)			
S1	2 MUL (b_i, c_i); 3 MUL (b_i, c_i)	0	2	2
S2	4 ADD; 1 ADD (a_i, b_i)	2	0	2
S3	2 MUL (b_i, c_i); 3 MUL (b_i, c_i)			
S4	4 ADD			
S5				

Synthesis Flow

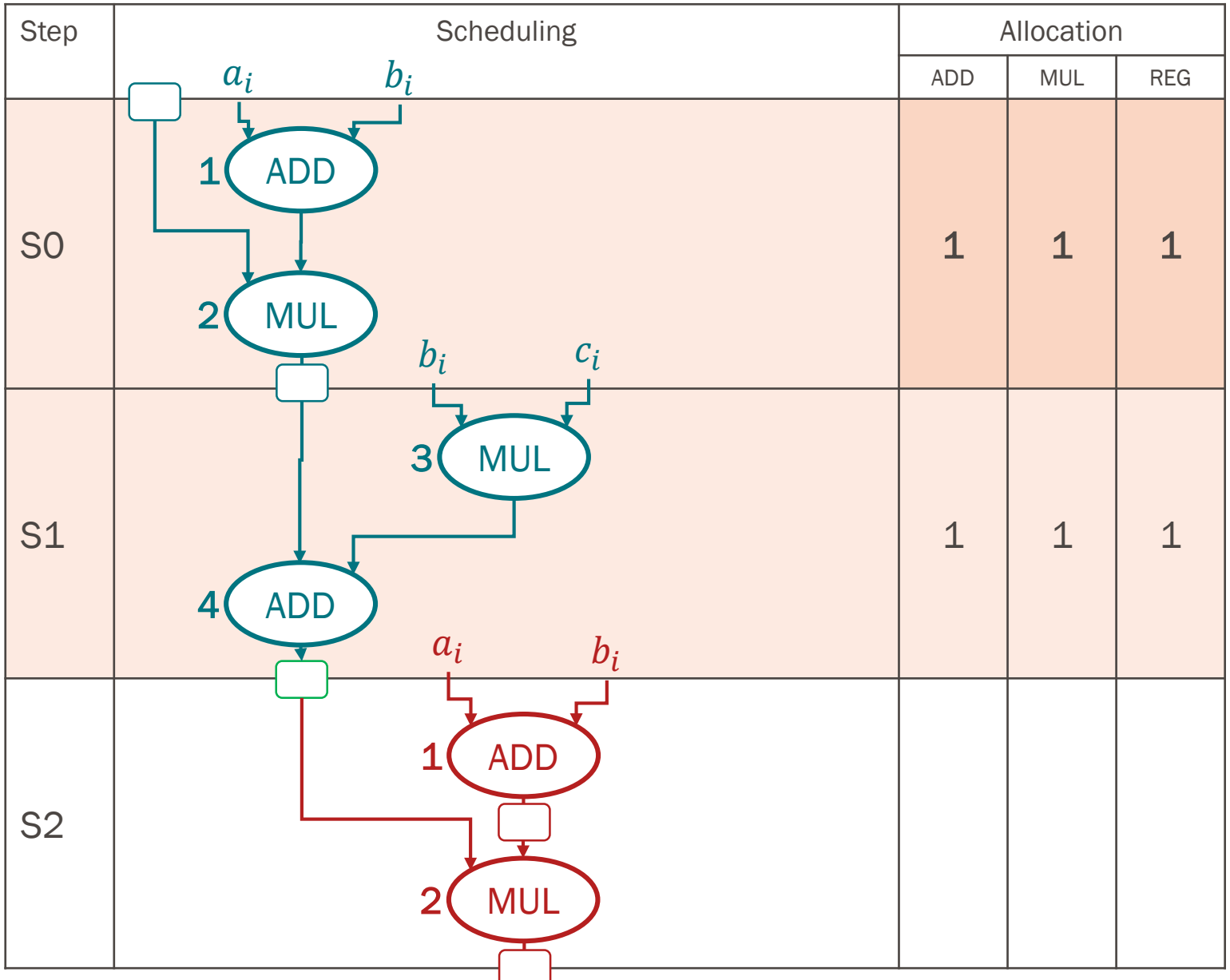
C → DFG → Scheduling → Allocation → Binding → RTL

Solution: $T_{clk} = 15ns$ with $II = 2$

DFG



T_{clk}	15				
II	2				
ADD	1	MUL	1	REG	1
T_L	$100 * 2 * 15ns = \mathbf{3000ns}$				
C_R	$C_R = 2 * 1 + 3 * 1 + 1 * 1 = \mathbf{6}$				

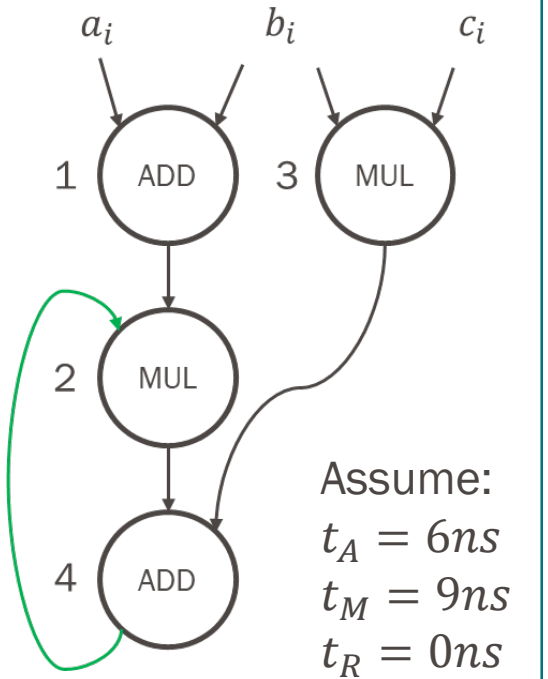


Synthesis Flow

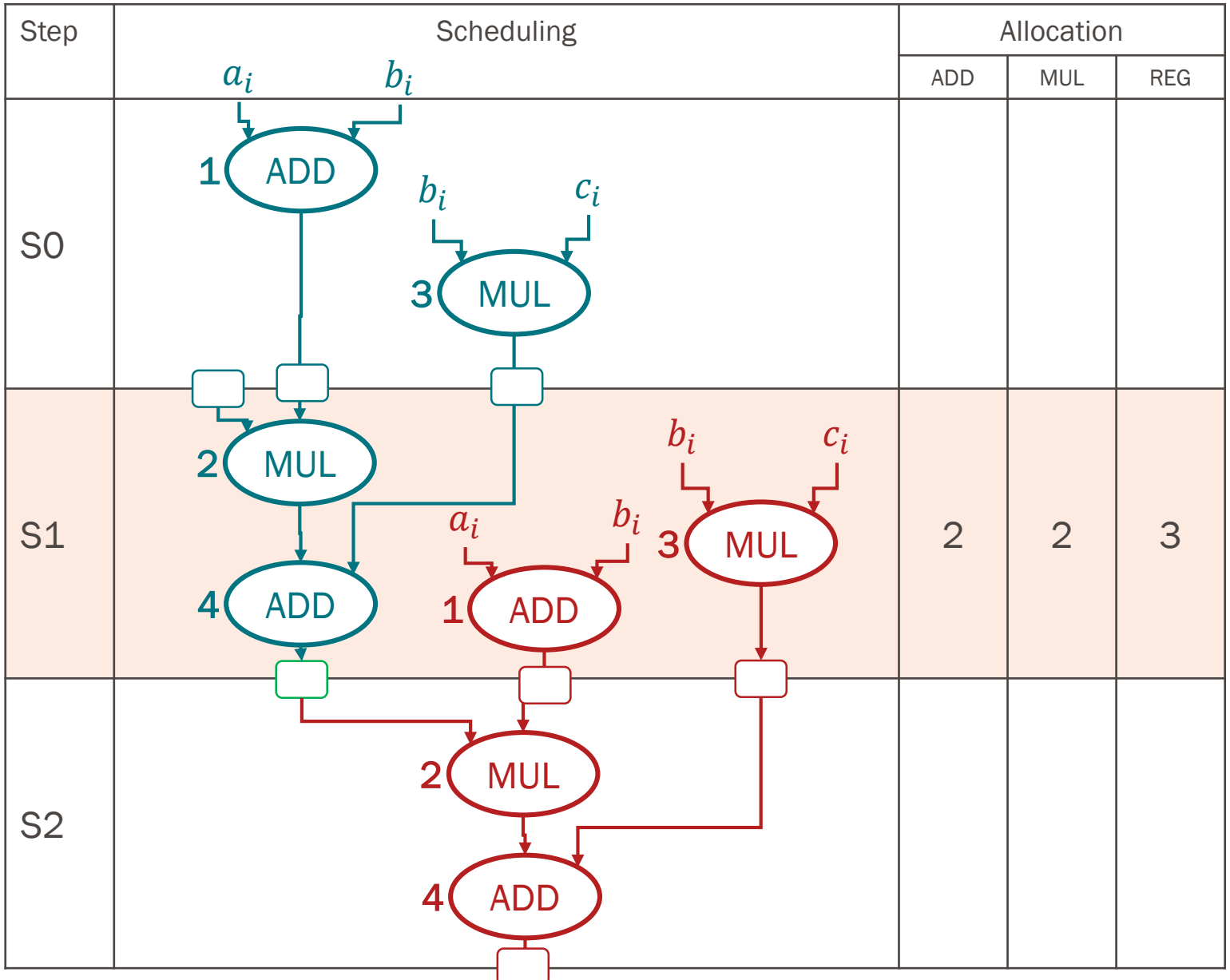
C → DFG → Scheduling → Allocation → Binding → RTL

Solution: $T_{clk} = 15ns$ with $II = 1$ (Thank you Pedro for this solution ☺)

DFG



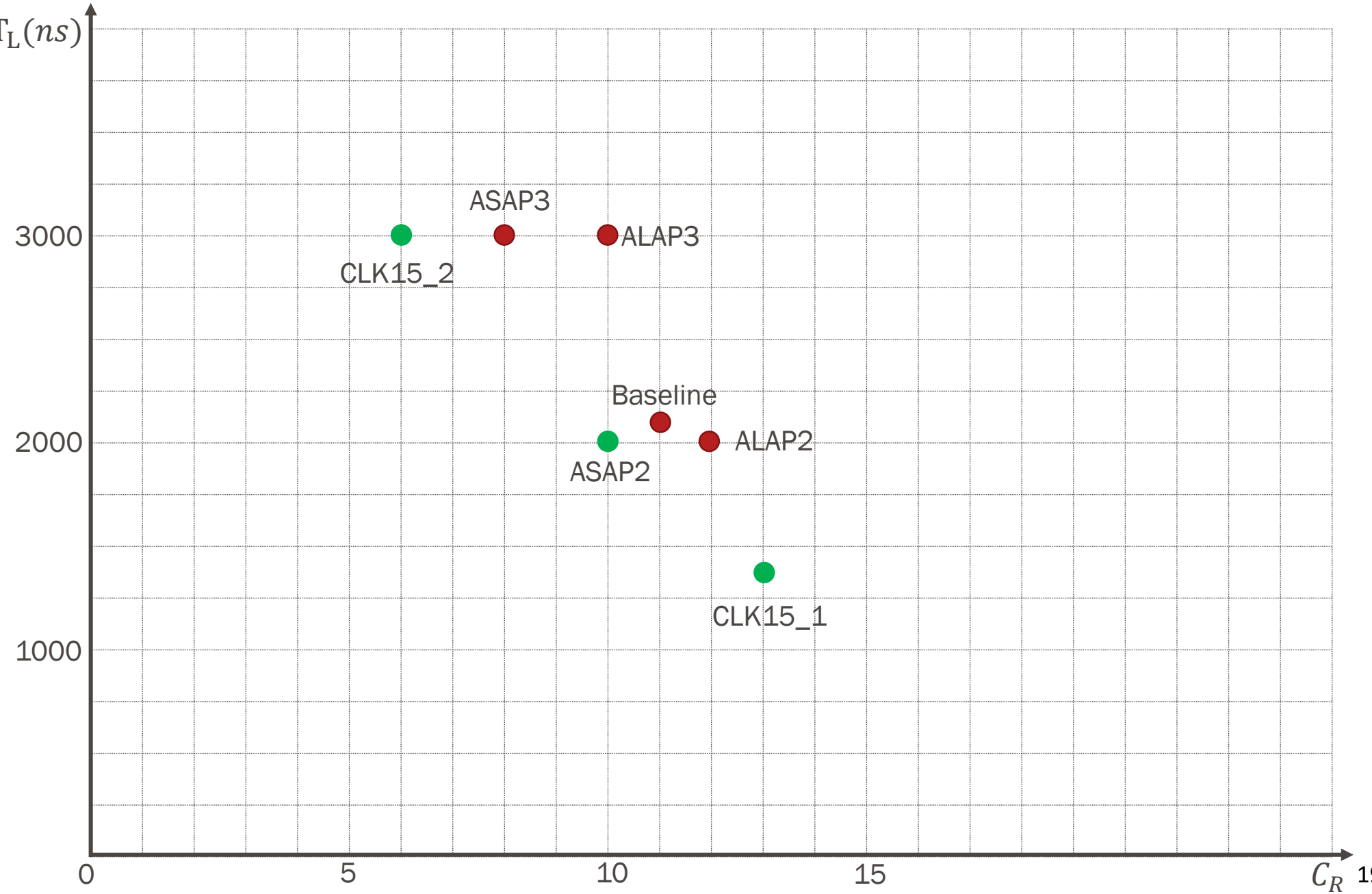
T_{clk}	15				
II	1				
ADD	2	MUL	2	REG	3
T_L	$100 * 1 * 15ns = \mathbf{1500ns}$				
C_R	$C_R = 2 * 2 + 3 * 2 + 1 * 3 = \mathbf{13}$				



DSE

Solutions table

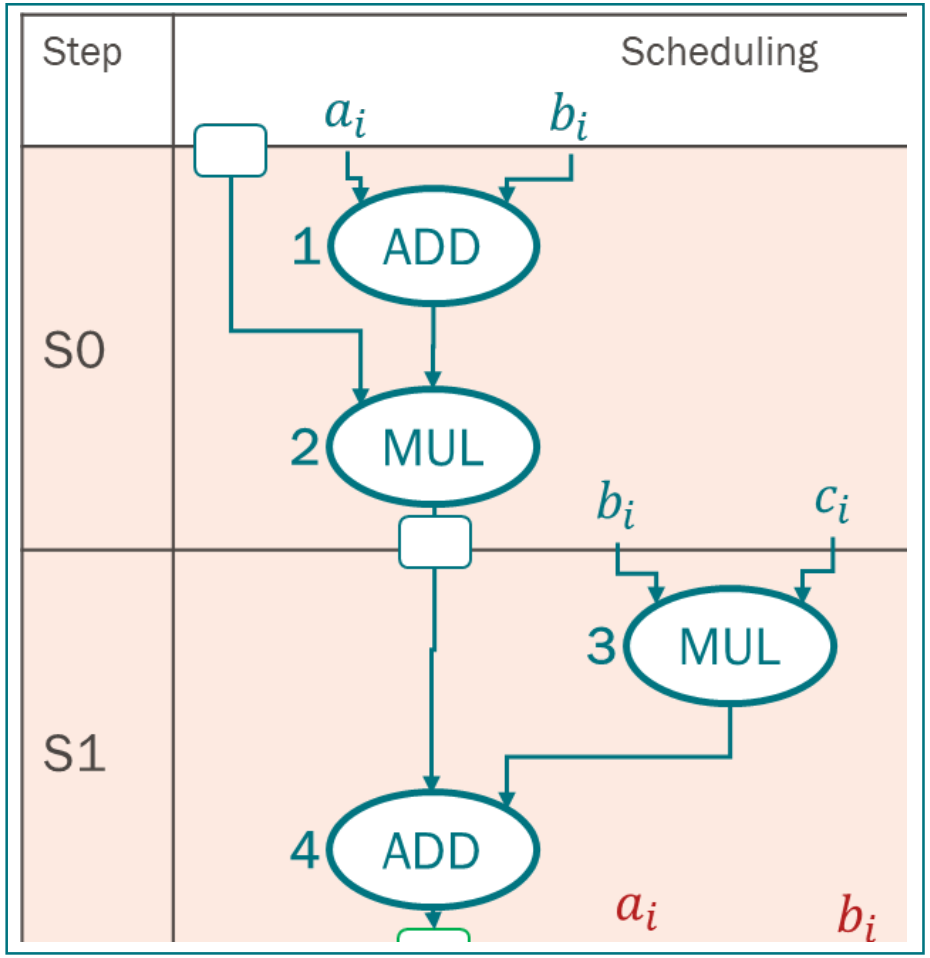
Solution	$T_L(ns)$	$C_R(ns)$
Baseline	2100	11
ASAP3	3000	8
ALAP3	3000	10
ASAP2	2000	10
ALAP2	2000	12
CLK15_2	3000	6
CLK15_1	1500	13



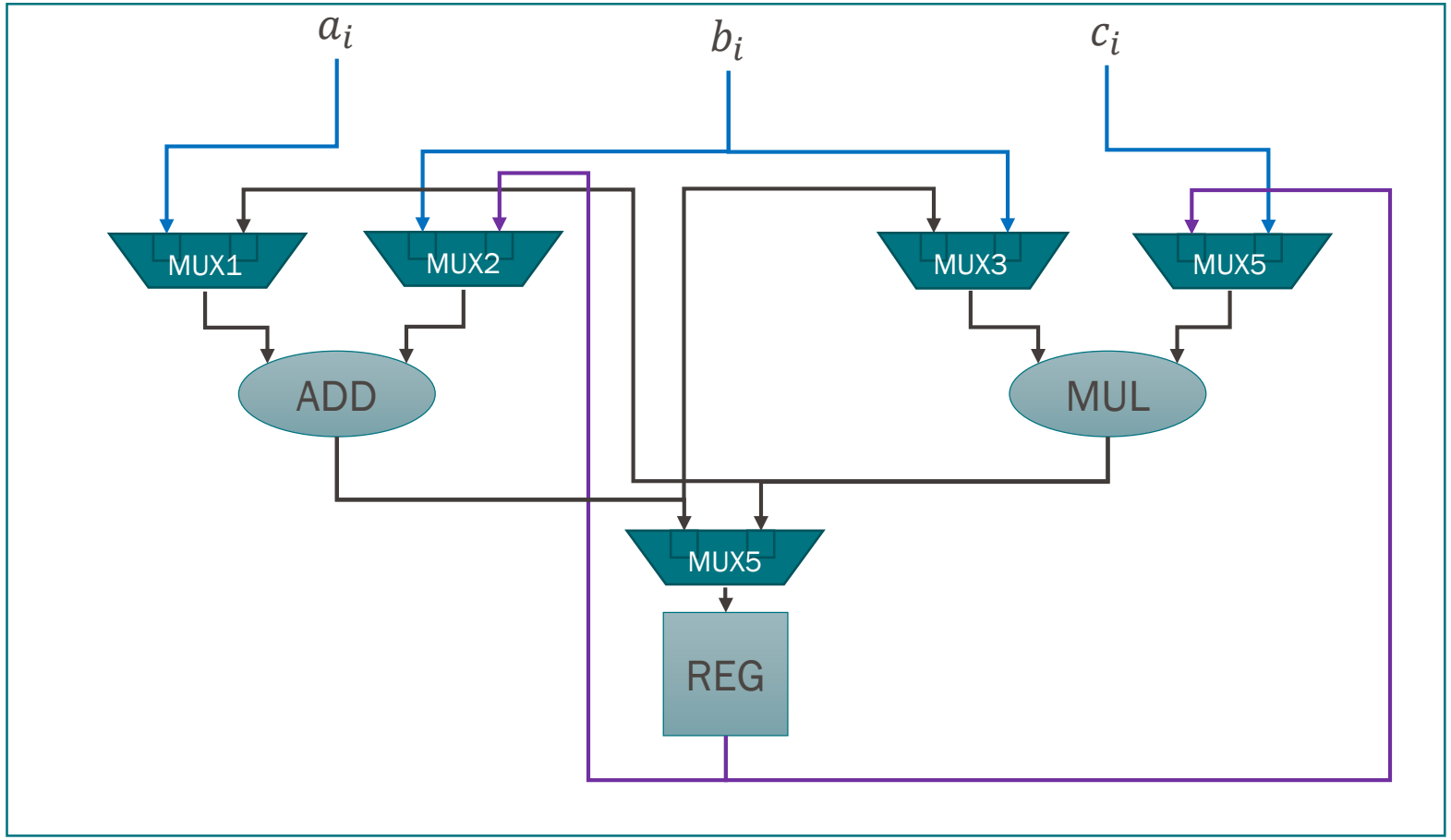
Synthesis Flow

C → DFG → Scheduling → Allocation → Binding → RTL

Scheduling



Binding



Allocated Resources

#ADD	1	#MUL	1	#REG	1

FSM	MUX1	MUX2	MUX3	MUX4	MUX5
S0	0	0	0	0	1
S1	1	1	1	1	0

Exercise

C++ description

```
int temp = 0;
for( int i=0 ; i<100 ; i++ ) {
    temp = (a[i] + b[i]) * temp + (b[i] + c [i]);
}
```

Assume:

$t_A = 6ns$

$t_M = 9ns$

$t_R = 0ns$

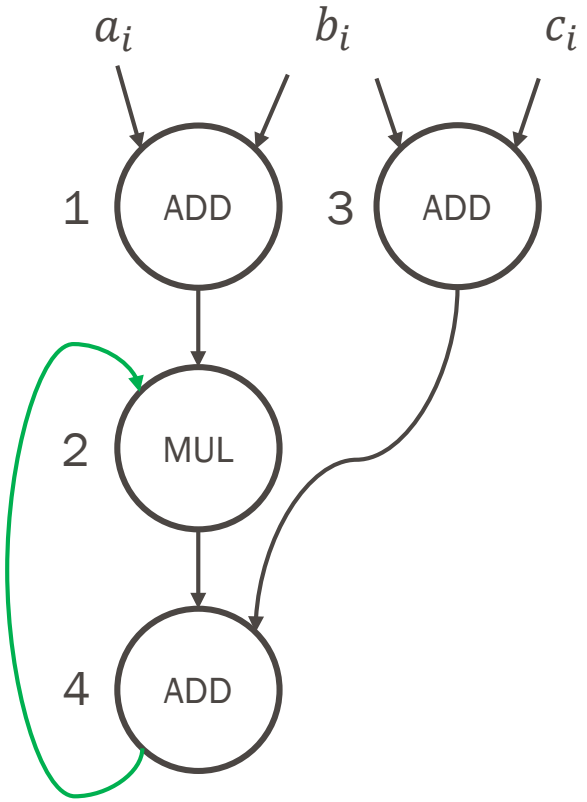
Timings model:

$T_L = \#iterations * II * T_{clk}$

Resources model:

$C_R = 2 * \#ADDs + 3 * \#MULs + 1 * \#REGs$

Data Flow Graph



Explore:

ASAP II=2 II=3

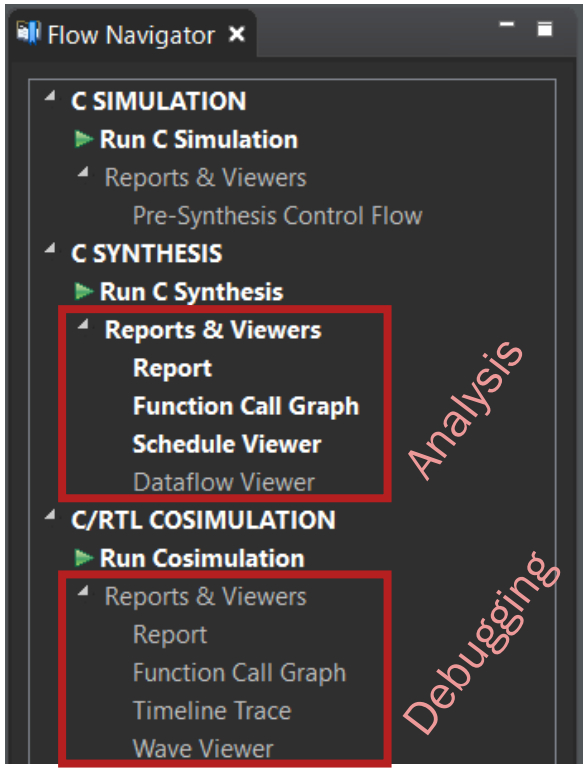
ALAP II=2 II=3

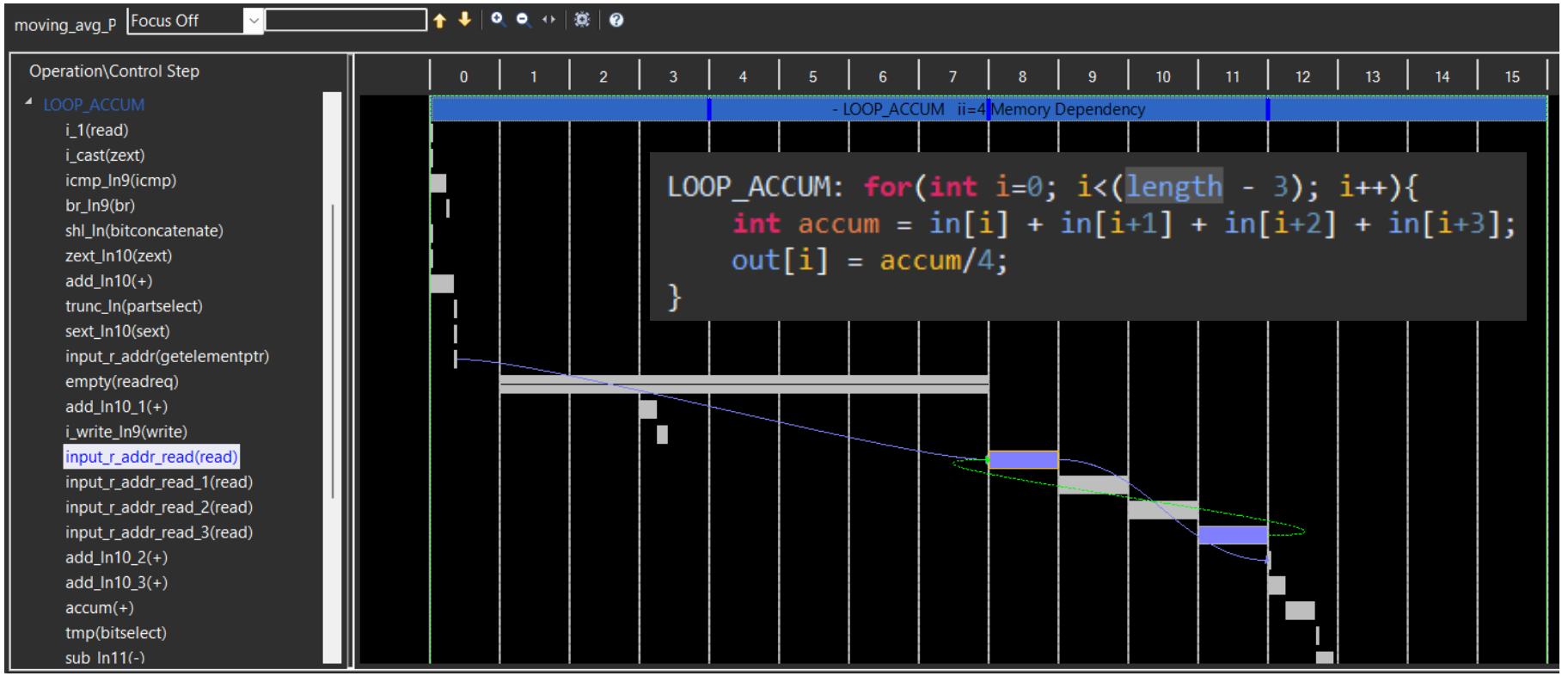
$T_{clk} = min$

$T_{clk} = 15ns$

Binding for any solution with $T_{clk} = min$

Synthesis Flow Analysis and Debugging





Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
moving_avg				-	183	1,830E3	-	184	-	no	0	0	2272	3788	0
moving_avg_Pipeline_LOOP_ACCUM	II Violation			-	175	1,750E3	-	175	-	no	0	0	423	512	0
LOOP_ACCUM	II Violation	Memory Dependency	1	-	173	1,730E3	15	4	40	yes	-	-	-	-	-

From C/C++ to HW

- **Always remember: you are not programming in C!**
- You are defining a behavior: The C++ description functionality matches the generated HW functionality
- Not all C++ features are available
- You can rewrite the C description to guide the compiler and synthesizer towards a desired design (dependencies)

HLS translation table

C++	Generated HW
Functions	Modules with I/Os
Arguments	Ports of modules
Private variables	Local signals
Static variables	Registers
Arrays	Storage units (Memories)
Arrays or pointers in arguments	Memory ports
Variables	Signals (Wires)
Function calls	Modules instantiation

C++ Unsupported Features
System Calls
Dynamic memory allocation
Pointer casting only for not C++ types
Pointers to pointers
Function pointers
Recursive functions
Undefined behaviors for conditions
Virtual functions and pointers

Over the next session

If statement example

```
if(x == 0) {
    a = ...
} if else (x < 5) {
    a = ...
} if else (x == 5) {
    a = ...
}
```

Undefined behavior

```
if(x == 0) {
    a = ...
} if else (x < 5) {
    a = ...
} if else (x == 5) {
    a = ...
} else {
    a = ...
}
```

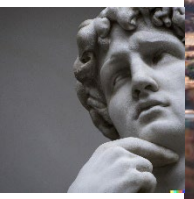
Define behavior for ALL cases

```
a = 0;

if(x == 0) {
    a = ...
} if else (x < 5) {
    a = ...
} if else (x == 5) {
    a = ...
}
```

Initialization of the variable

Summary

- How to perform a **Design Space Exploration** in HLS using:
  - Several **loop optimizations** pragmas: unrolling, merging, flattening, pipelining.
  - Adding **resources constraints**.
  - Changing **the target frequency**.
  - **C/C++ description rewriting** to help the interpreter understand the data dependencies.
- How HLS synthesizes C/C++ into HW through the **Scheduling, Allocation and Binding**.
- How the **Initiation Interval (II)** is directly related with the **throughput** and the **total execution time** and depends on the carried loop dependencies and the number of resources available.
- You have seen tools in **Vitis HLS** that will help us optimizing your HW:
  - Creating different **solutions** through the use of **directives**
  - **Solution summary reports** -> Giving us estimations of the **timings** and the **resources** consumption
  - **Analysis reports** -> How the tool is synthesizing the C/C++ description



Questions?

Rubén Rodríguez Álvarez

EPFL – Embedded Systems Laboratory
ruben.rodriguezalvarez@epfl.ch

EPFL

Lab. On HW-SW Digital Systems Codesign
EE-390(a)

Templates for Scheduling, Allocation and Binding

Prof. David Atienza
Dr. Denisa Constantinescu, Dr. Miguel Peón-Quirós,
Mr. Rubén Rodríguez-Álvarez, Ms. Stasa Kostic, Mr. Karan Pathak

Exercise

C++ description

```
int temp = 0;
for( int i=0 ; i<100 ; i++ ) {
    temp = (a[i] + b[i]) * temp + (b[i] + c [i]);
}
```

Assume:

$t_A = 6ns$

$t_M = 9ns$

$t_R = 0ns$

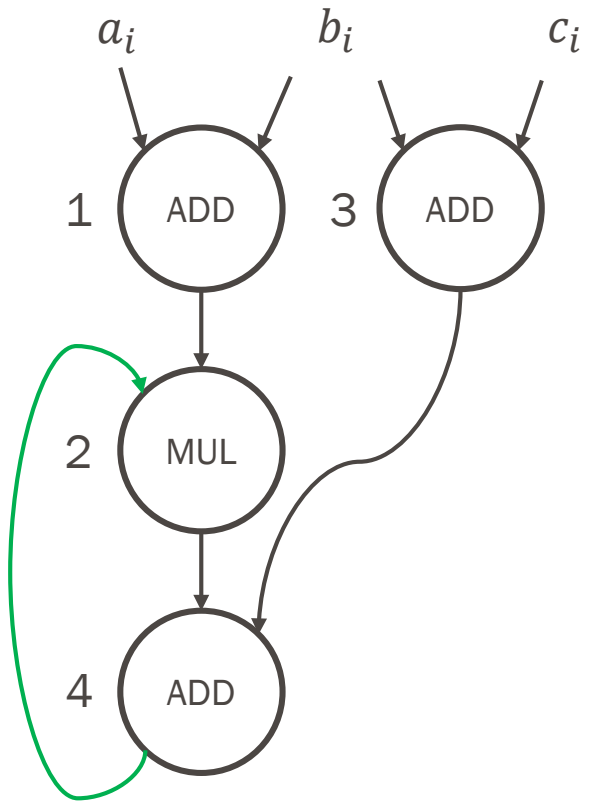
Timings model:

$T_L = \#iterations * II * T_{clk}$

Resources model:

$C_R = 2 * \#ADDs + 3 * \#MULs + 1 * \#REGs$

Data Flow Graph



Explore:

ASAP II=2 II=3

ALAP II=2 II=3

$T_{clk} = min$

$T_{clk} = 15ns$

Binding for any solution with $T_{clk} = min$

SOLUTION

DFG

T_{clk}					
II					
ADD		MUL		REG	
T_L					
C_R					

Step	Scheduling	Allocation		
		ADD	MUL	REG

SOLUTION

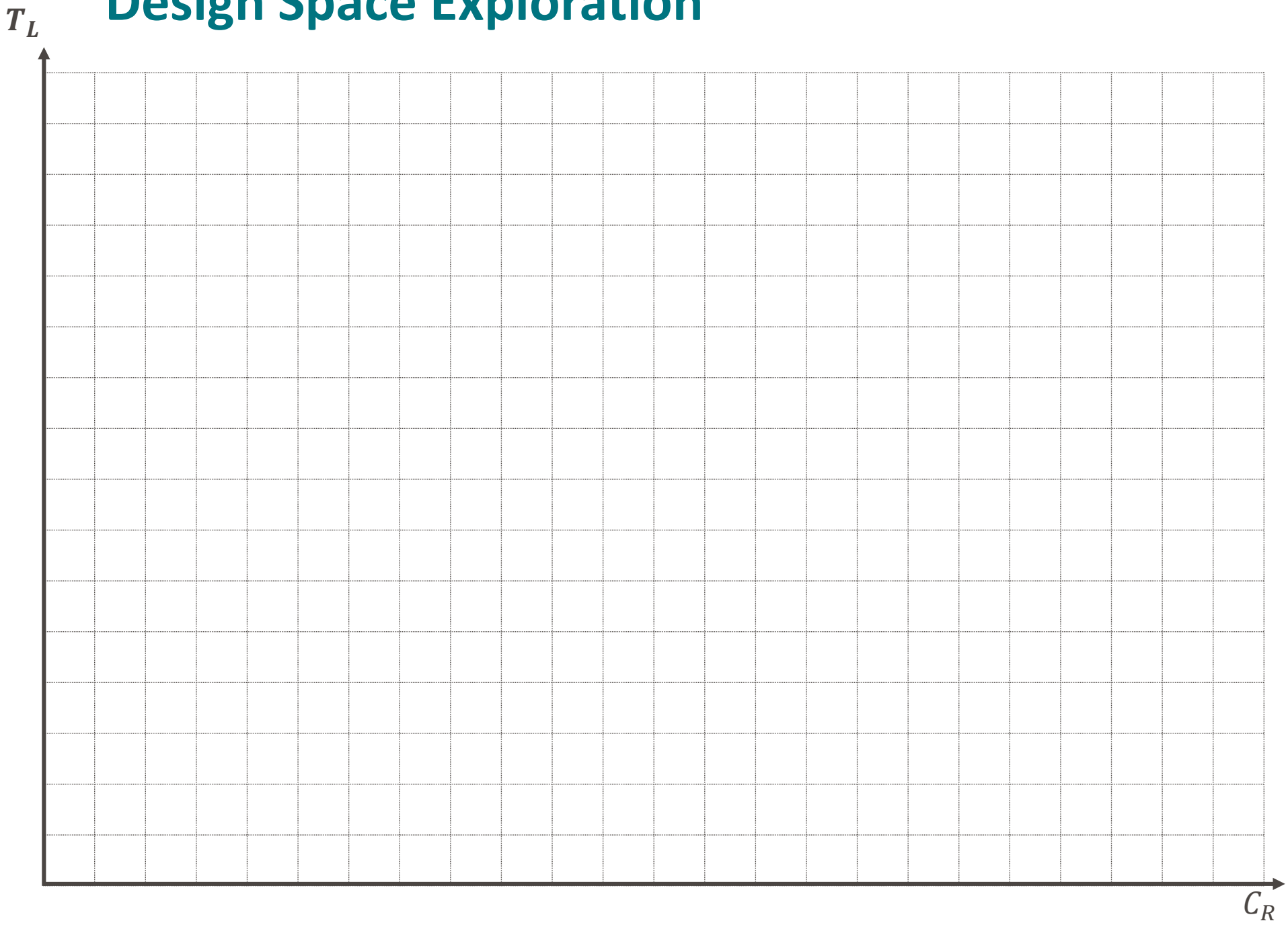
DFG

T_{clk}					
II					
ADD		MUL		REG	
T_L					
C_R					

Step	Scheduling	Allocation		
		ADD	MUL	REG

Design Space Exploration

Solution	T_L	C_R



SOLUTION

Scheduling

Binding

FSM	MUX1	MUX2	MUX3	MUX4	MUX5	MUX6	MUX7	MUX8	MUX9
S0									
S1									
S2									
S3									
S4									
S5									

Allocated Resources

#ADD		#MUL		#REG	