DESIGN, AUTOMATION & TEST IN EUROPE
9 – 13 March, 2015 · Grenoble · France
The European Event for Electronic System Design & Test

Profiling-Driven Multi-Cycling in FPGA High-Level Synthesis

Stefan Hadjis[1], Andrew Canis[1], Ryoya Sobue[2], Yuko Hara-Azumi[3], Hiroyuki Tomiyama[2], Jason Anderson[1]

[1]University of Toronto,
[2]Ritsumeikan University,
[3]Tokyo Institute of Technology

legup.eecg.utoronto.ca

Motivation

- High-Level Synthesis:
  - Synthesize hardware from software
  - Raises hardware design abstraction
- However, HLS-generated hardware may be slower and consume more area/power
- Our work boosts the speed of HLS circuits using multi-cycling of combinational paths

LegUp High-Level Synthesis

- High-Level Synthesis Tool from University of Toronto
  - http://legup.eecg.toronto.edu
- Input: C software program
- Synthesizes a hybrid target architecture:
  - Processor (a soft-core MIPS or hardened ARM)
  - **Custom hardware accelerators**

    → **Focus of this talk**
- Open source and freely downloadable
  - 1300+ downloads by groups worldwide

Outline

- Multi-Cycle Paths in FPGA High-Level Synthesis
- Multi-Cycle Path Static Analysis
- Profiling-Driven Multi-Cycle Path Analysis
- Experimental Results

Outline

- **Multi-Cycle Paths in FPGA High-Level Synthesis**
- Multi-Cycle Path Static Analysis
- Profiling-Driven Multi-Cycle Path Analysis
- Experimental Results

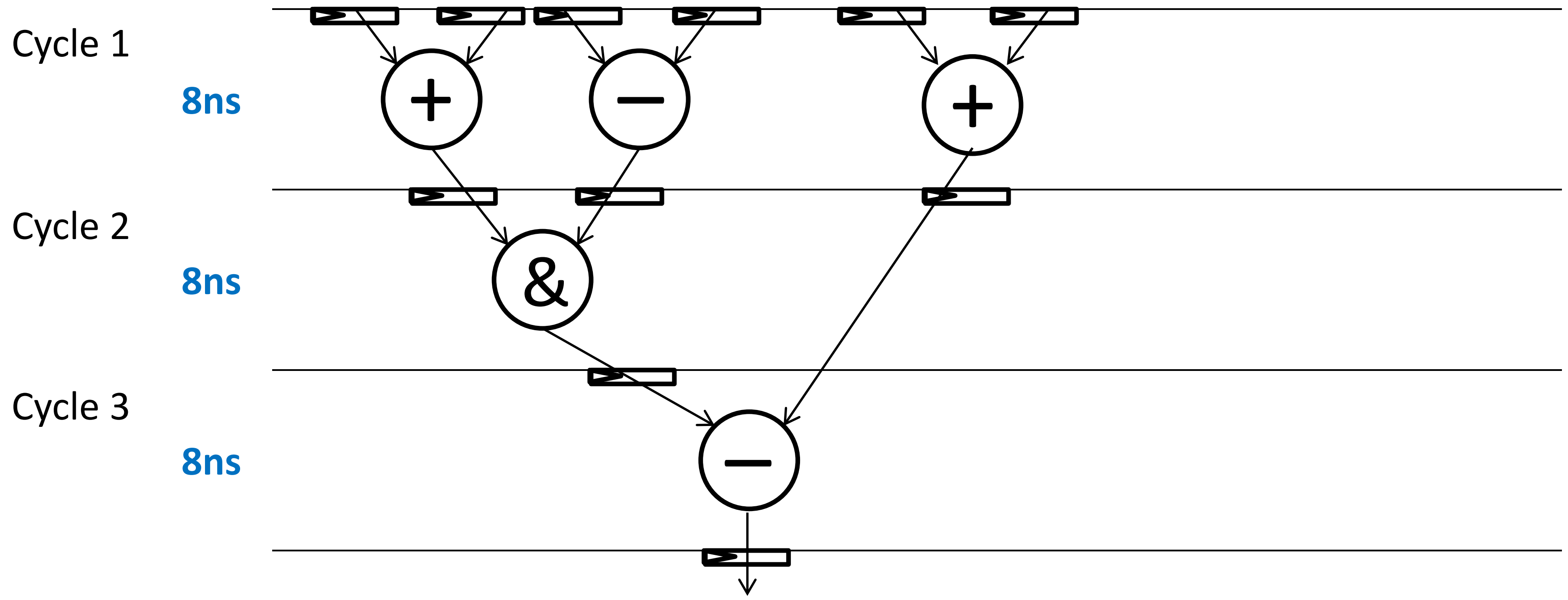
Multi-Cycle Paths

- Register-to-register path delays usually cannot exceed the clock period constraint
- Multi-cycling permits selected paths to have longer delays than the clock period
- A multi-cycle path is a register-to-register path which is permitted > 1 cycle to complete
- Terminology: A multi-cycle path that takes **N** clock cycles to complete has a **slack** of **N**
  - N is an integer > 1

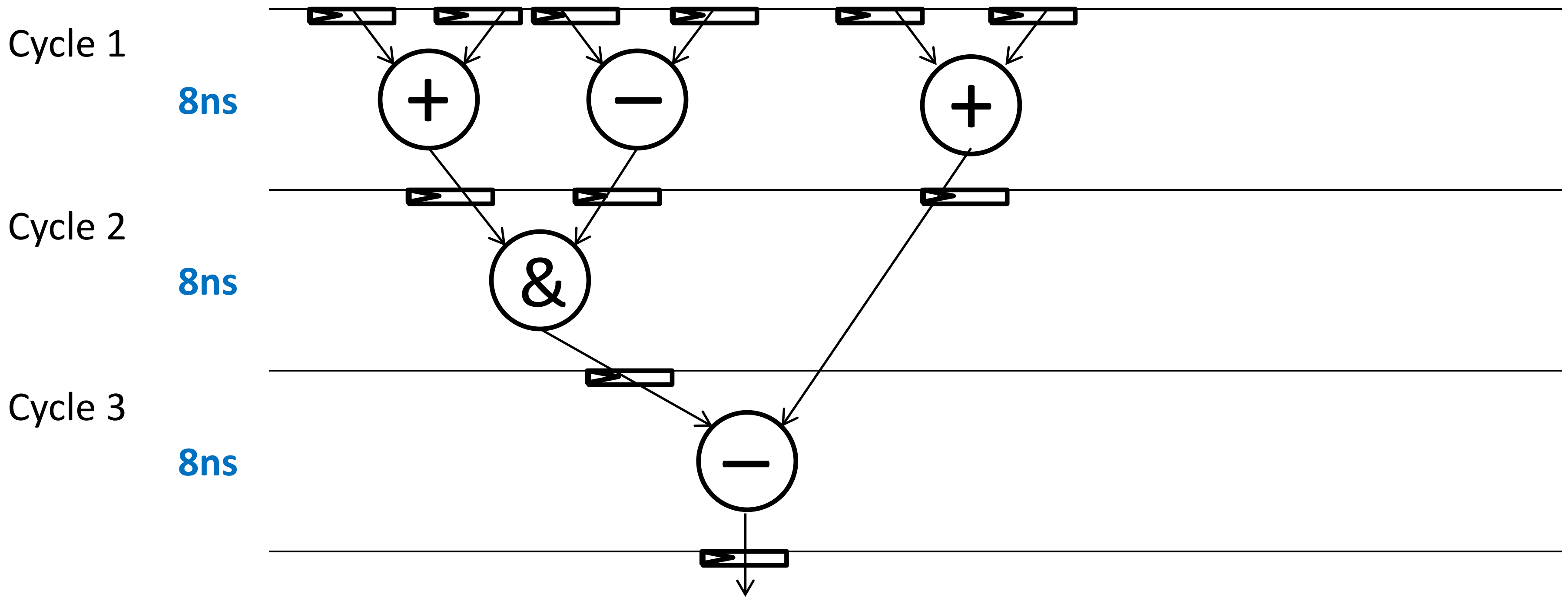
Multi-Cycle Paths

- A multi-cycle path with slack N has these constraints:
  1. Registered inputs can be held constant for N cycles
  2. Outputs are not used in these subsequent N cycles

Multi-Cycle Paths

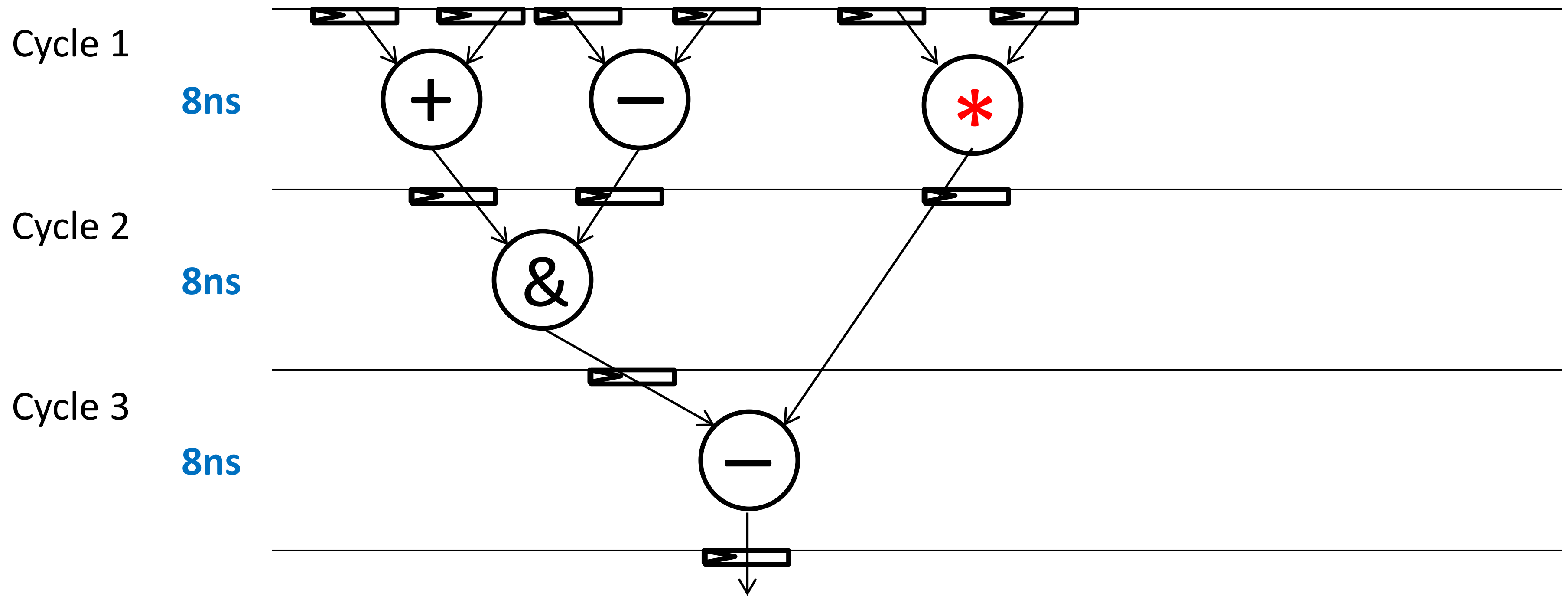


Multi-Cycle Paths



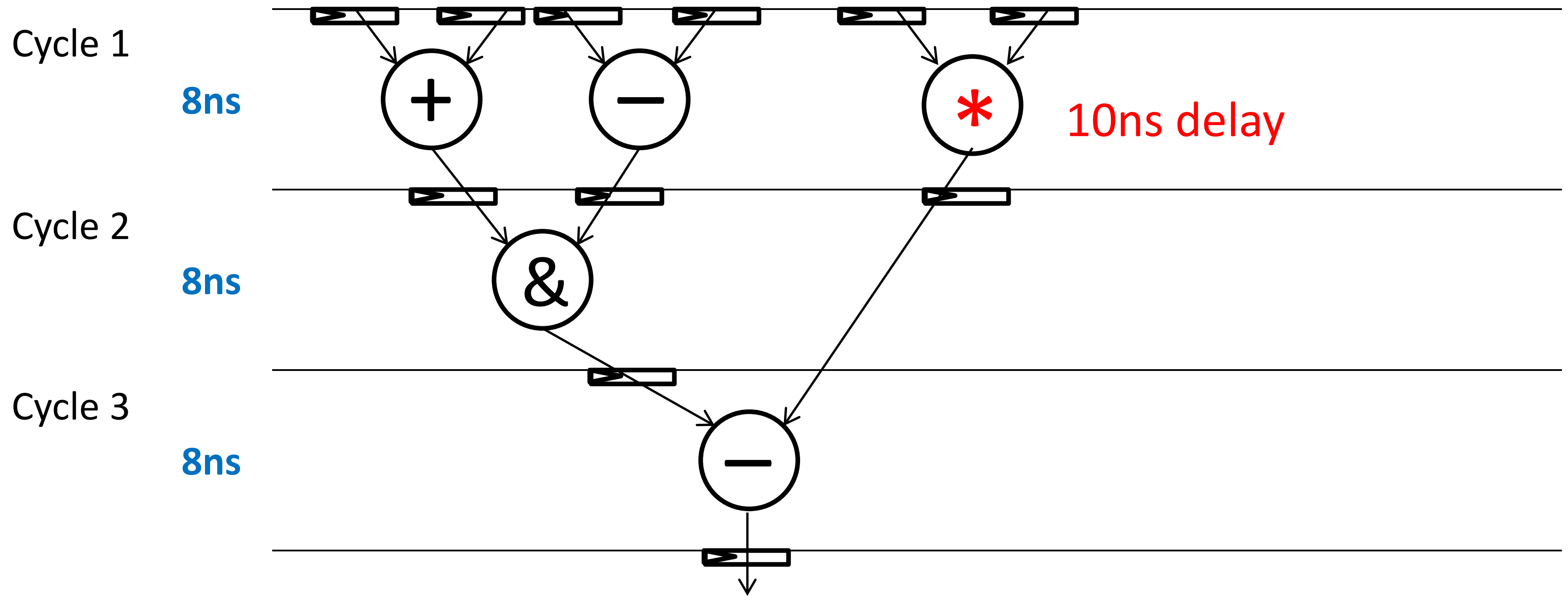
Fmax: 125 MHz

Multi-Cycle Paths



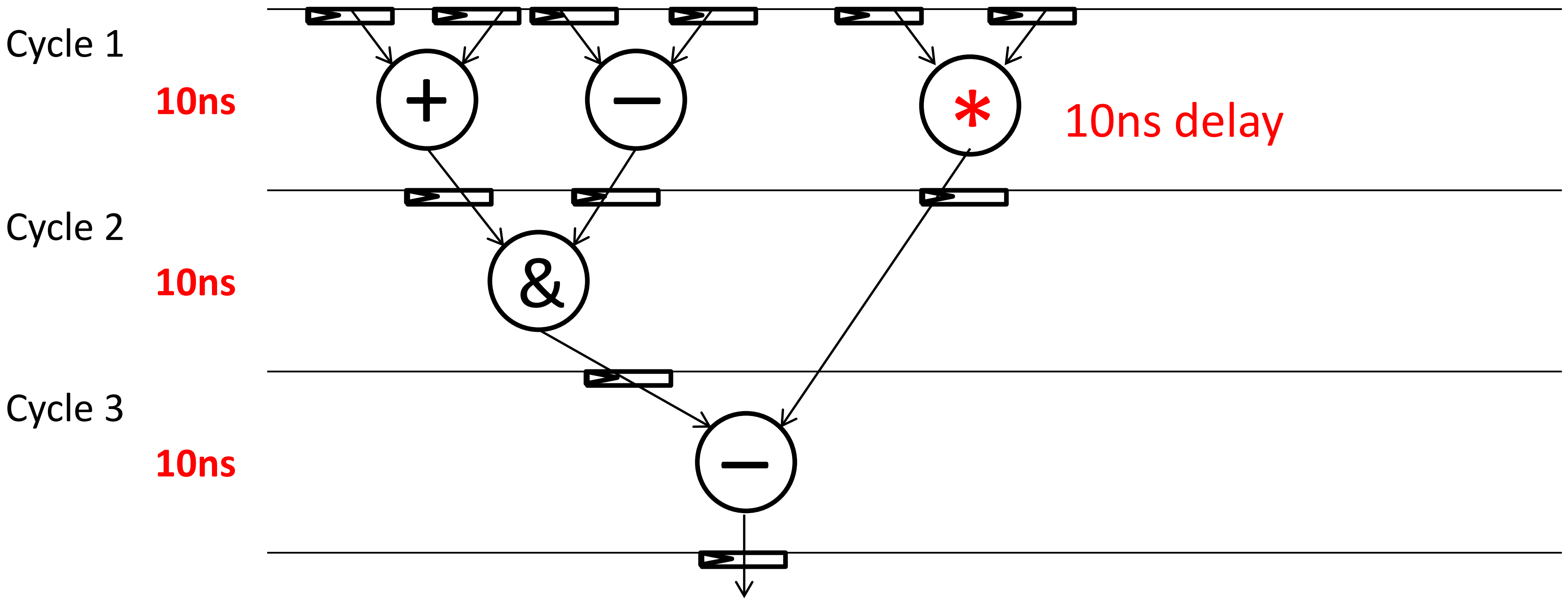
Fmax: 125 MHz

Multi-Cycle Paths



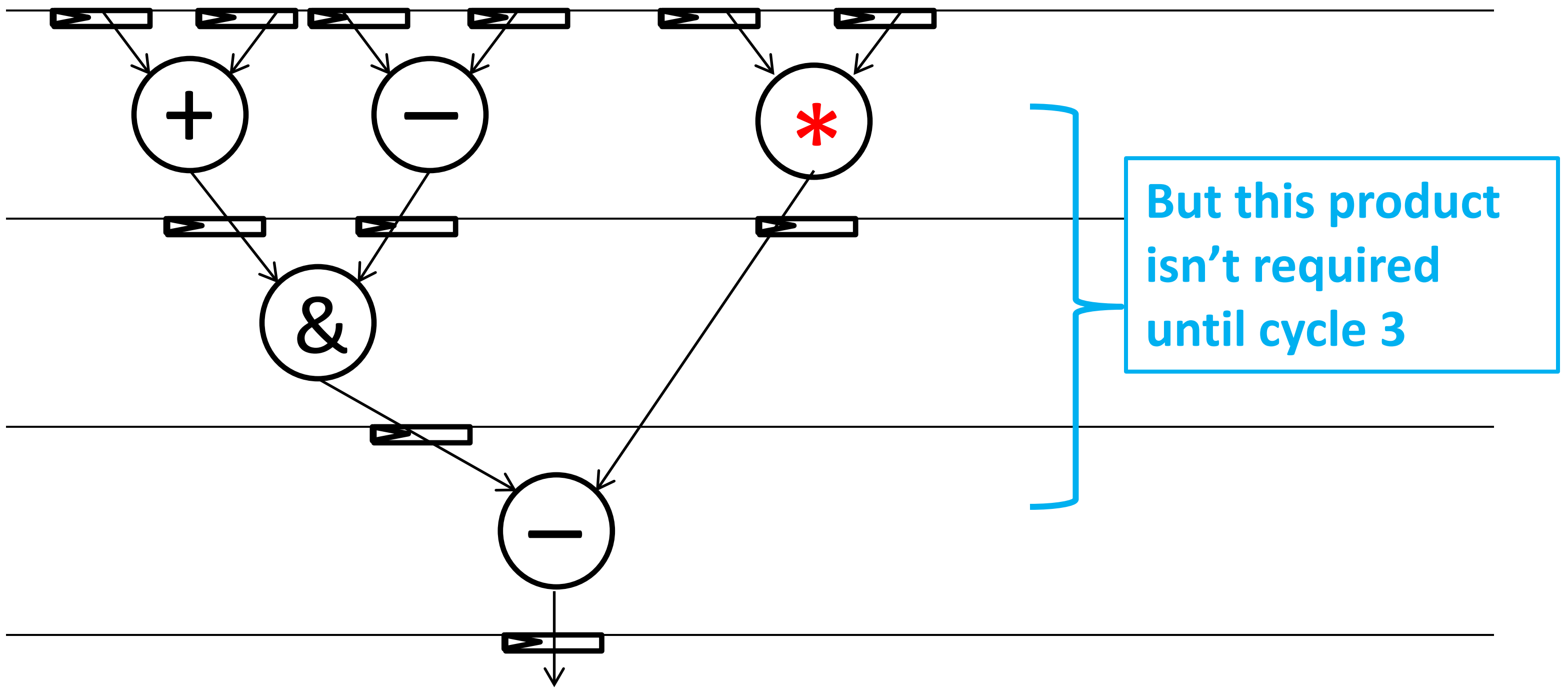
Fmax: 125 MHz

Multi-Cycle Paths



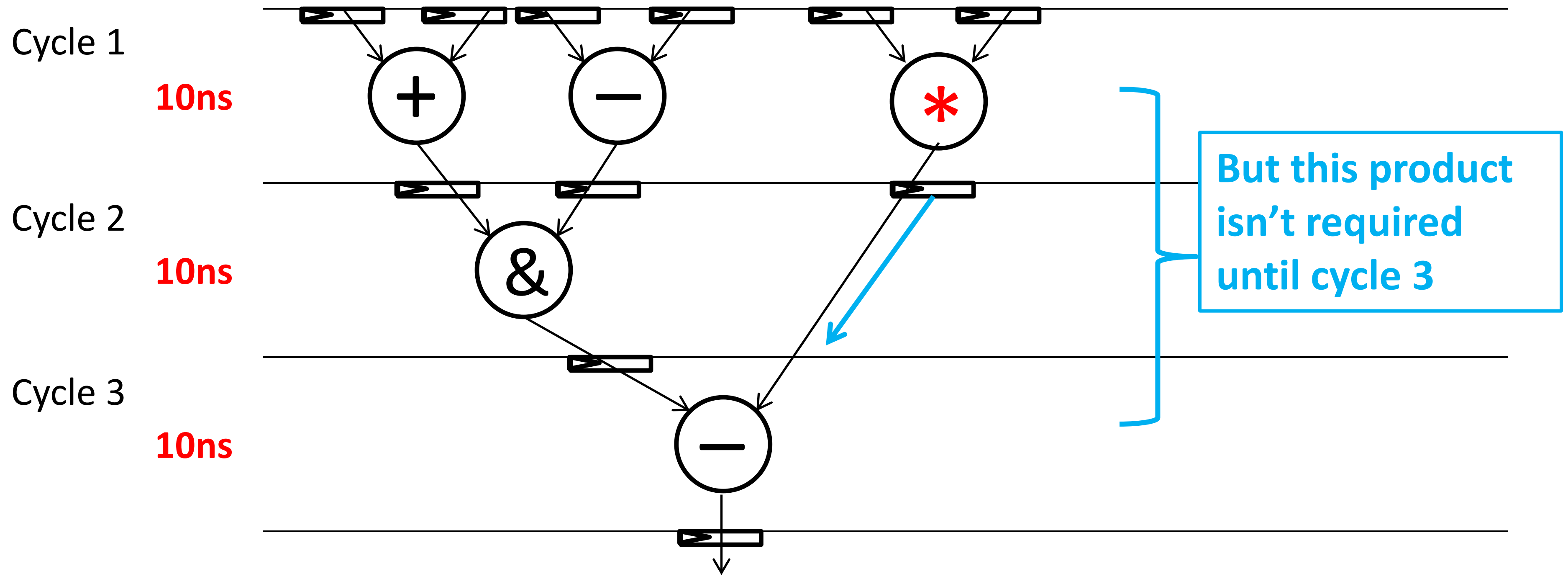
Fmax: 100 MHz

Multi-Cycle Paths



Fmax: 100 MHz

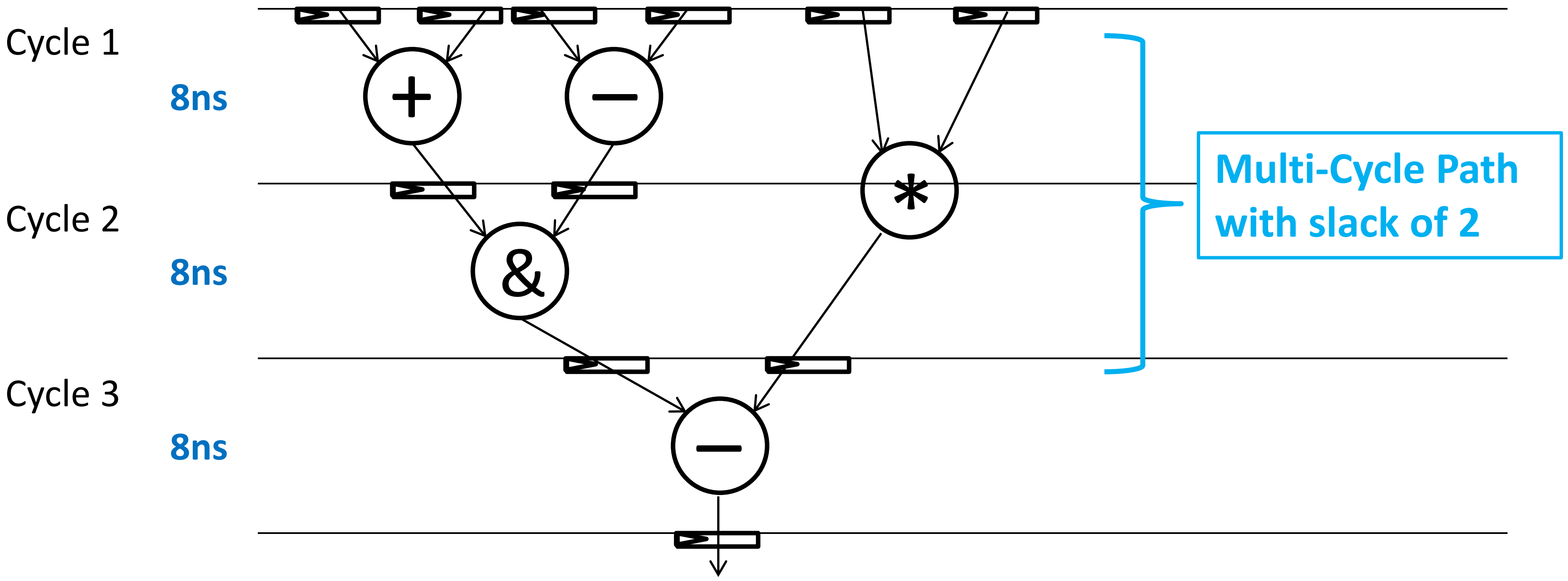
Multi-Cycle Paths



But this product isn't required until cycle 3

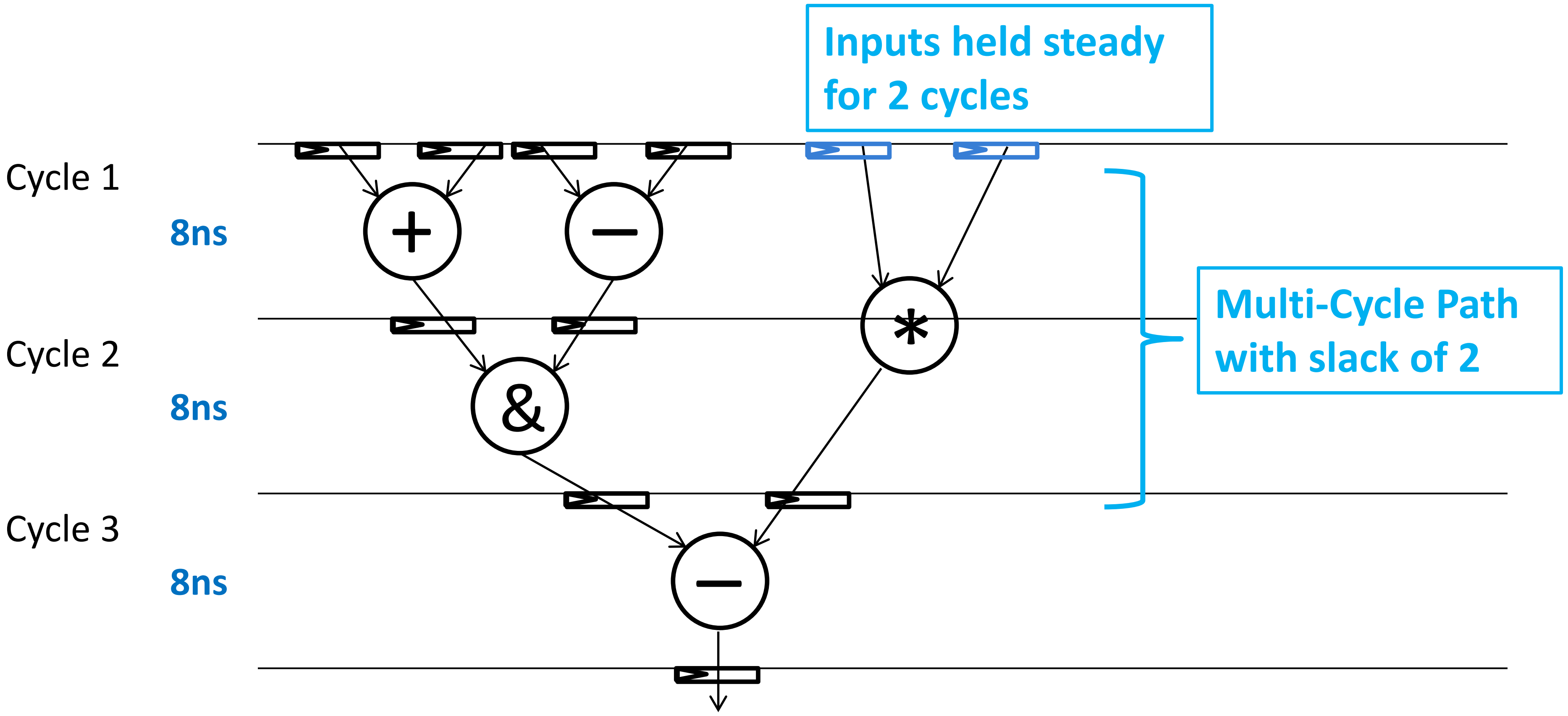
Fmax: 100 MHz

Multi-Cycle Paths



Fmax: 125 MHz

Multi-Cycle Paths



Inputs held steady for 2 cycles
Cycle 1
8ns
Cycle 2
8ns
Cycle 3
8ns
+
−
*
&
−
Multi-Cycle Path with slack of 2

Fmax: 125 MHz

Multi-Cycle Paths

- Multi-cycled paths cannot be pipelined with initiation interval 1
- However, in HLS datapaths do not always benefit from pipeline parallelism
  - Portions of C algorithm can be sequential
- Paths with such cycle slack can be multi-cycled without increasing cycle latency

Benefits of Multi-Cycling vs. Pipelining

- Fewer register-to-register delays (Tsu, Tcq, clock skew)
- Synthesis tools optimize across register boundaries
- Data-path delays are difficult to predict in HLS (pre-routing), making scheduling pessimistic
  - Multi-cycle paths remove this speculation
- Can improve clock period
- Fewer registers

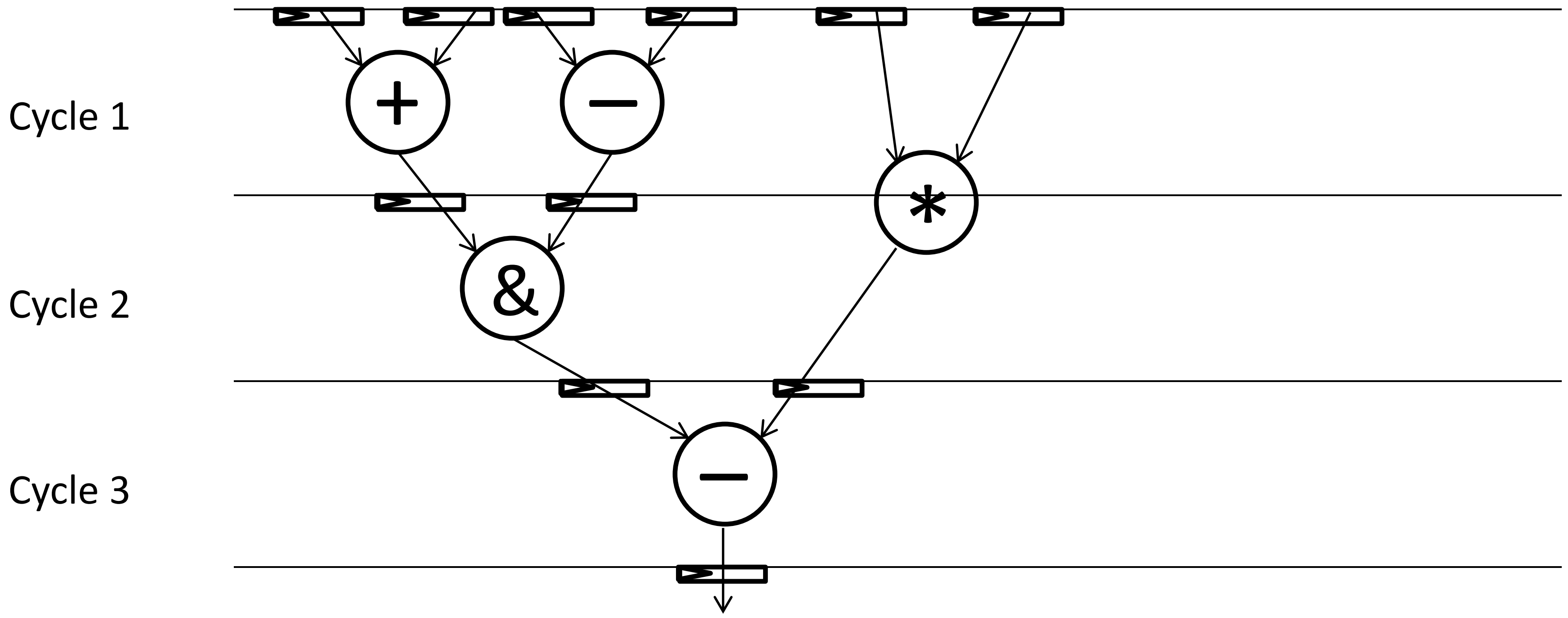
Why do this in HLS?

- HLS is an opportune stage of the flow to discover/ create multi-cycling paths
  - HLS schedules computations into states of a finite-state machine (FSM)
  - FSM state and dependencies of all operations are known
  - Can determine all paths whose computation is not needed in subsequent cycles
  - Much harder to do for a circuit described in RTL

LegUp Finite State Machine

- LegUp uses an FSM to schedule operations
  - Data-flow is directed by FSM next-state logic
  - Registers only enabled in certain FSM states

LegUp Finite State Machine

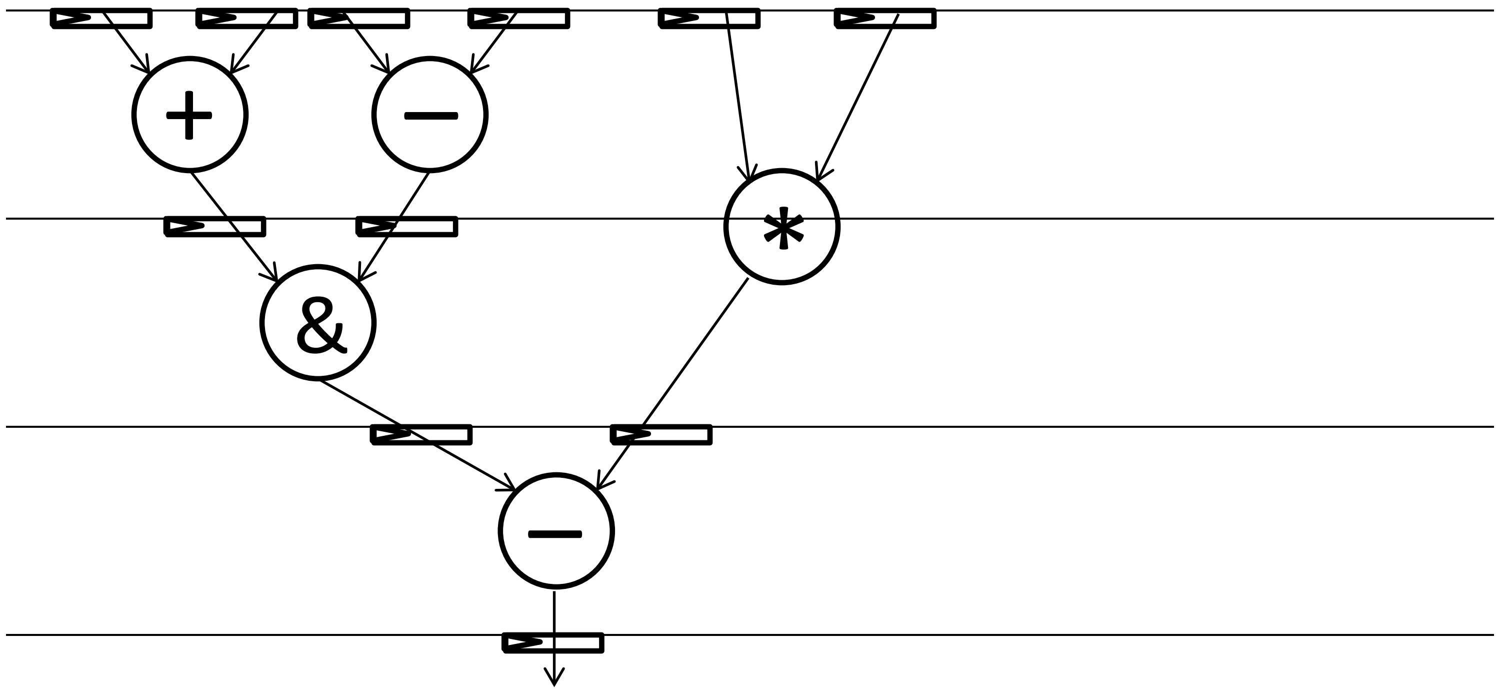


LegUp Finite State Machine

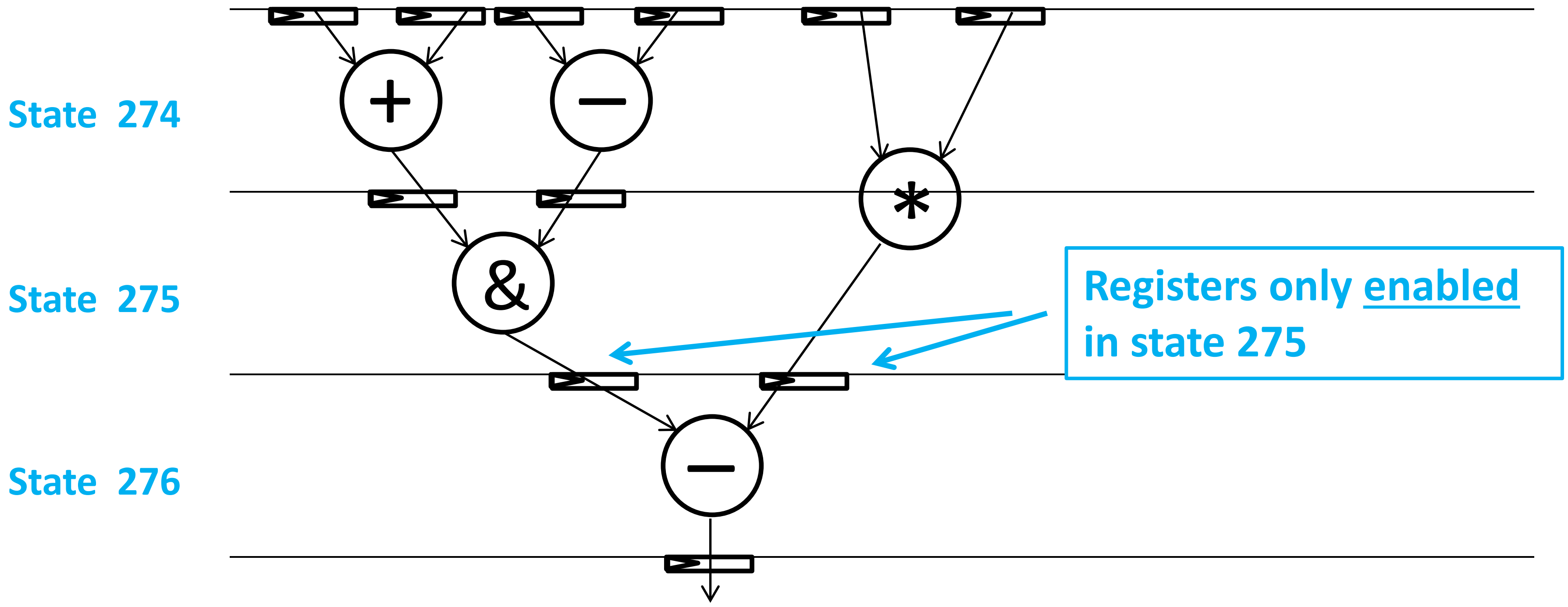
State 274

State 275

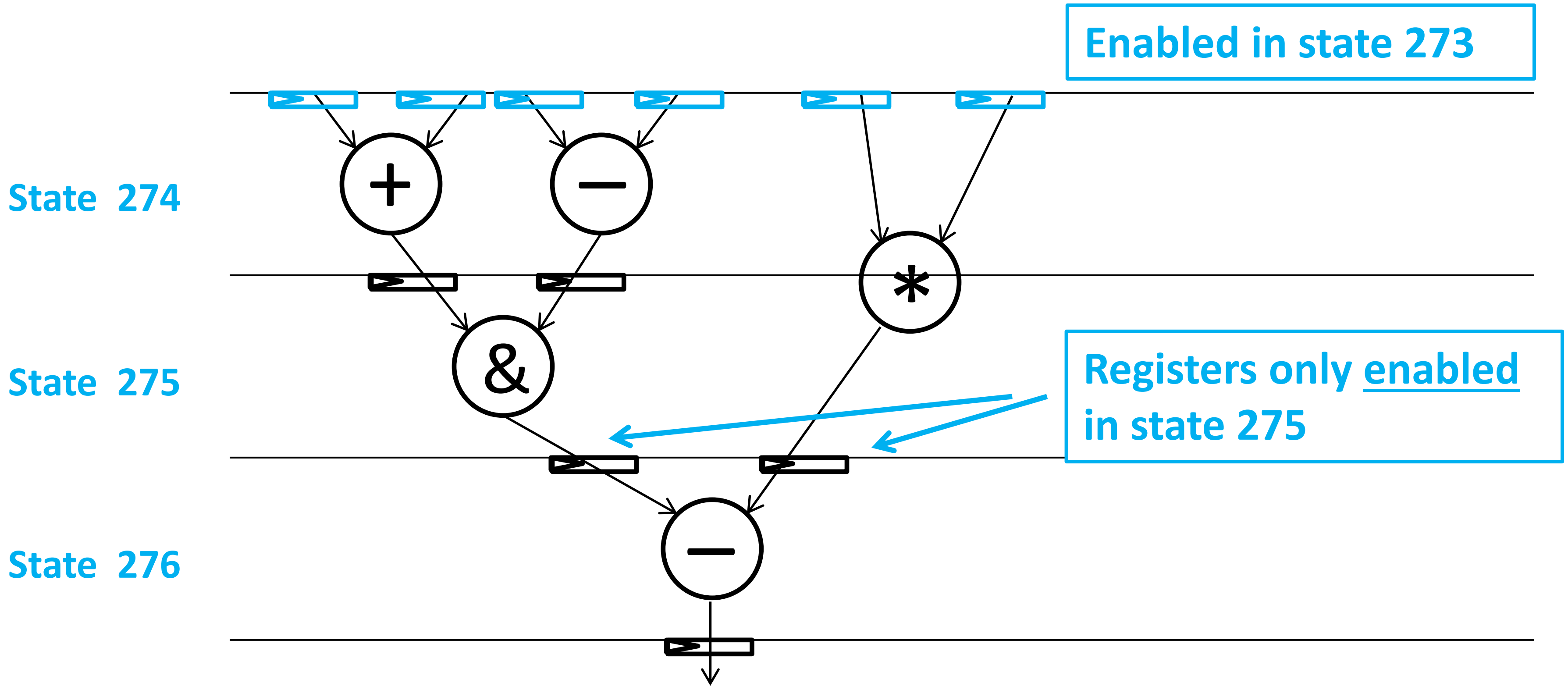
State 276



LegUp Finite State Machine

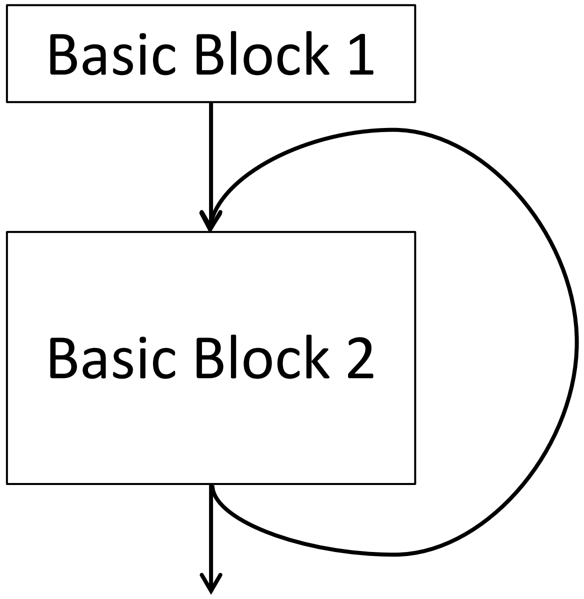


LegUp Finite State Machine



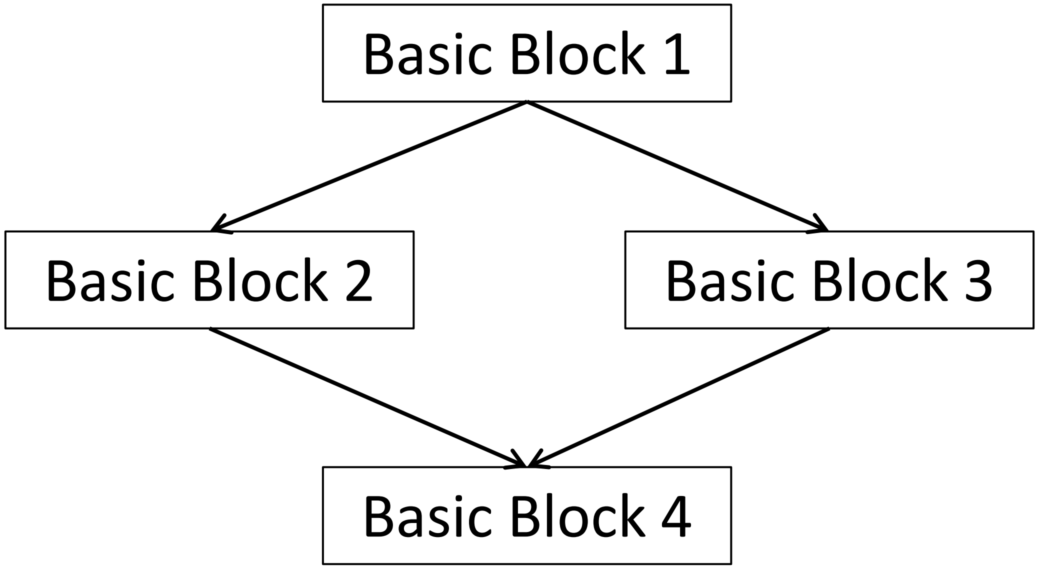
Two Important Compiler Concepts

1. **Basic Block**: a straight-line segment of code with a single entry point and a single exit point
   – For Loop:



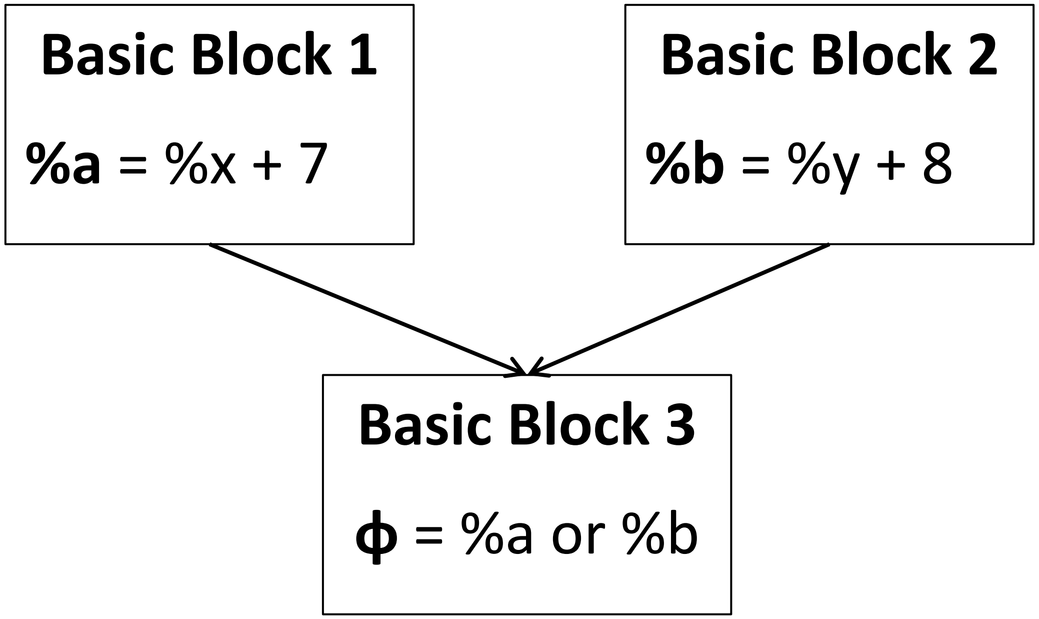
Two Important Compiler Concepts

1. **Basic Block**: a straight-line segment of code with a single entry point and a single exit point
   – Conditional branch:



Two Important Compiler Concepts

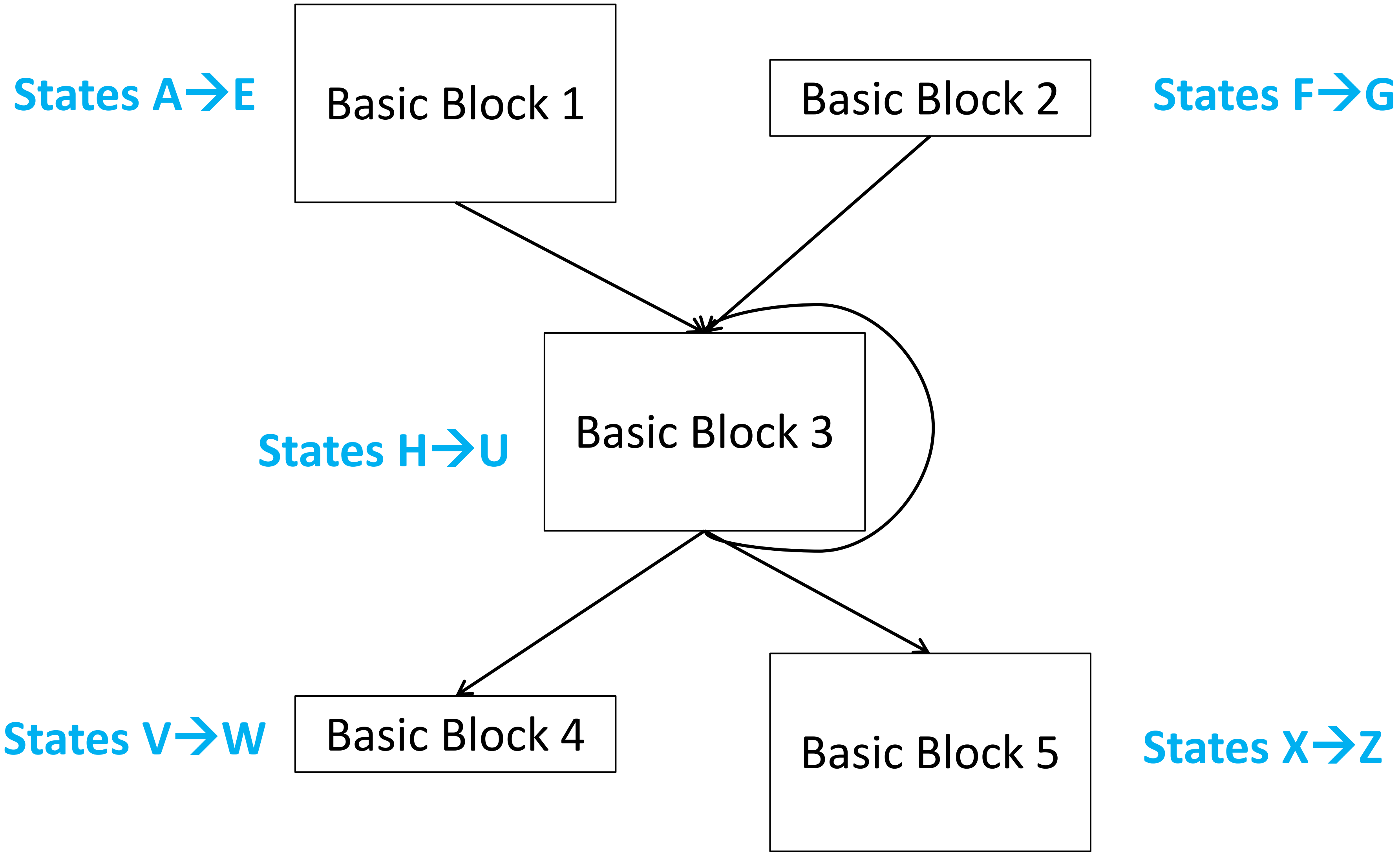
2. **PHI instruction**: control-flow instruction that selects a value depending on the previously executed basic block
   – Implemented with a mux in hardware



FSM Control Flow

- LegUp schedules operations (instructions) from the C program into FSM states
- Within basic blocks, the FSM state proceeds in order
- Once basic blocks finish, the FSM state jumps to the beginning of the next basic block
  - Since Basic Block transitions are only known dynamically, FSM state can jump in any order

FSM Control Flow



Outline

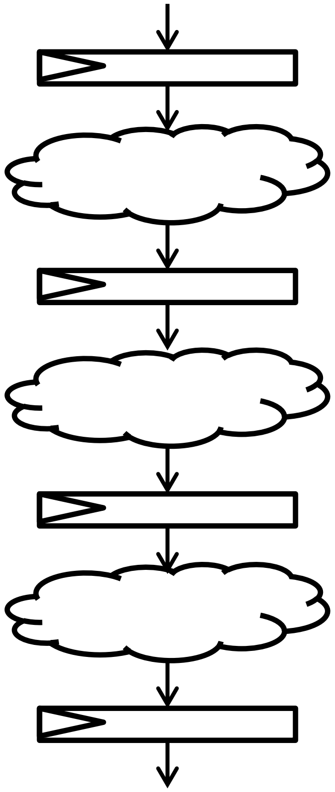
- Multi-Cycle Paths in FPGA High-Level Synthesis
- **Multi-Cycle Path Static Analysis**
- Profiling-Driven Multi-Cycle Path Analysis
- Experimental Results

Static Analysis

- After HLS scheduling we analyze scheduled operations to find all instances of multi-cycle slack
- Generate timing constraints for synthesis tools
- We can also modify the circuit to create additional multi-cycle paths
- De-pipeline datapaths and designate them as multi-cycle paths of equivalent latency

De-Pipelining Data Paths

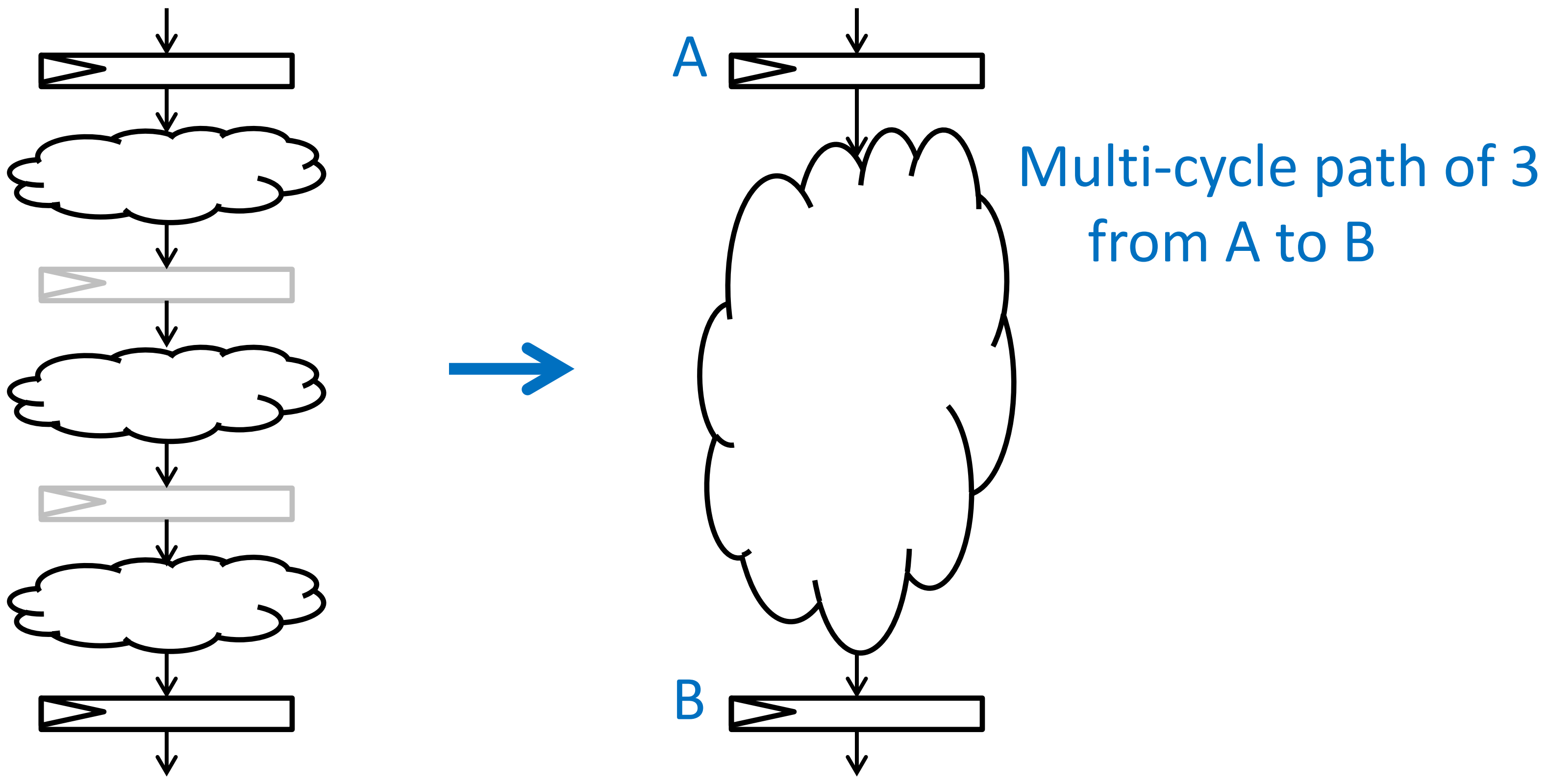
- If there isn't new input data every cycle:



pipelined data path

De-Pipelining Data Paths

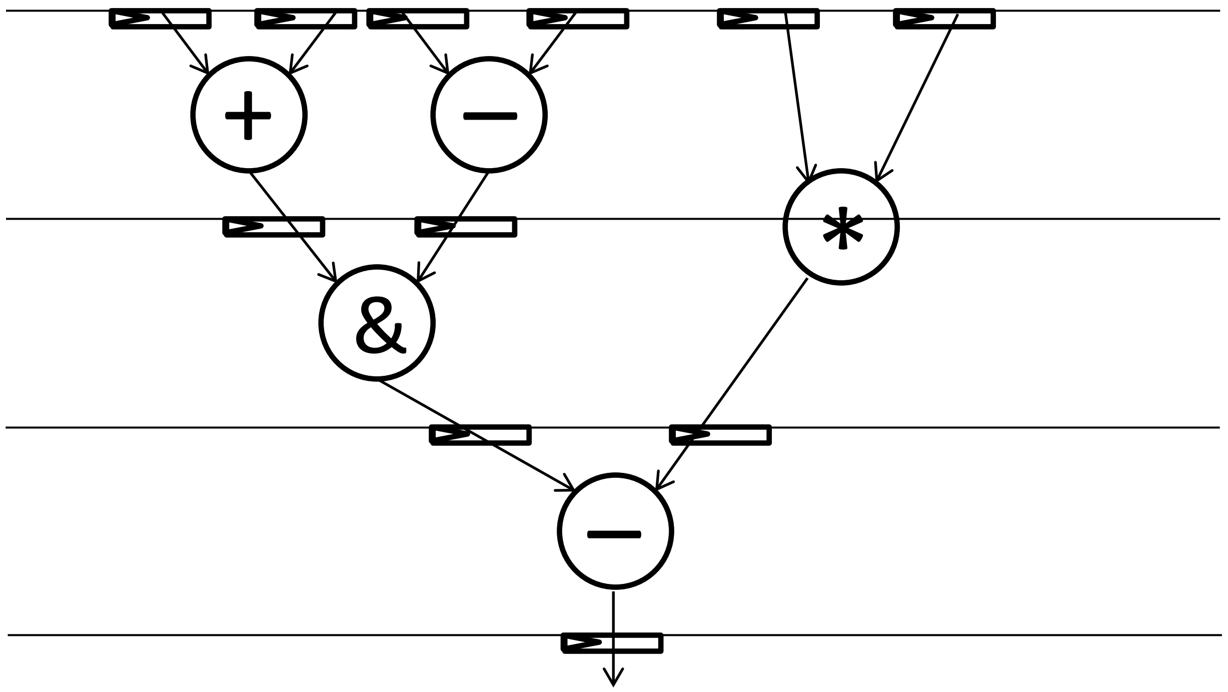
- If there isn't new input data every cycle :



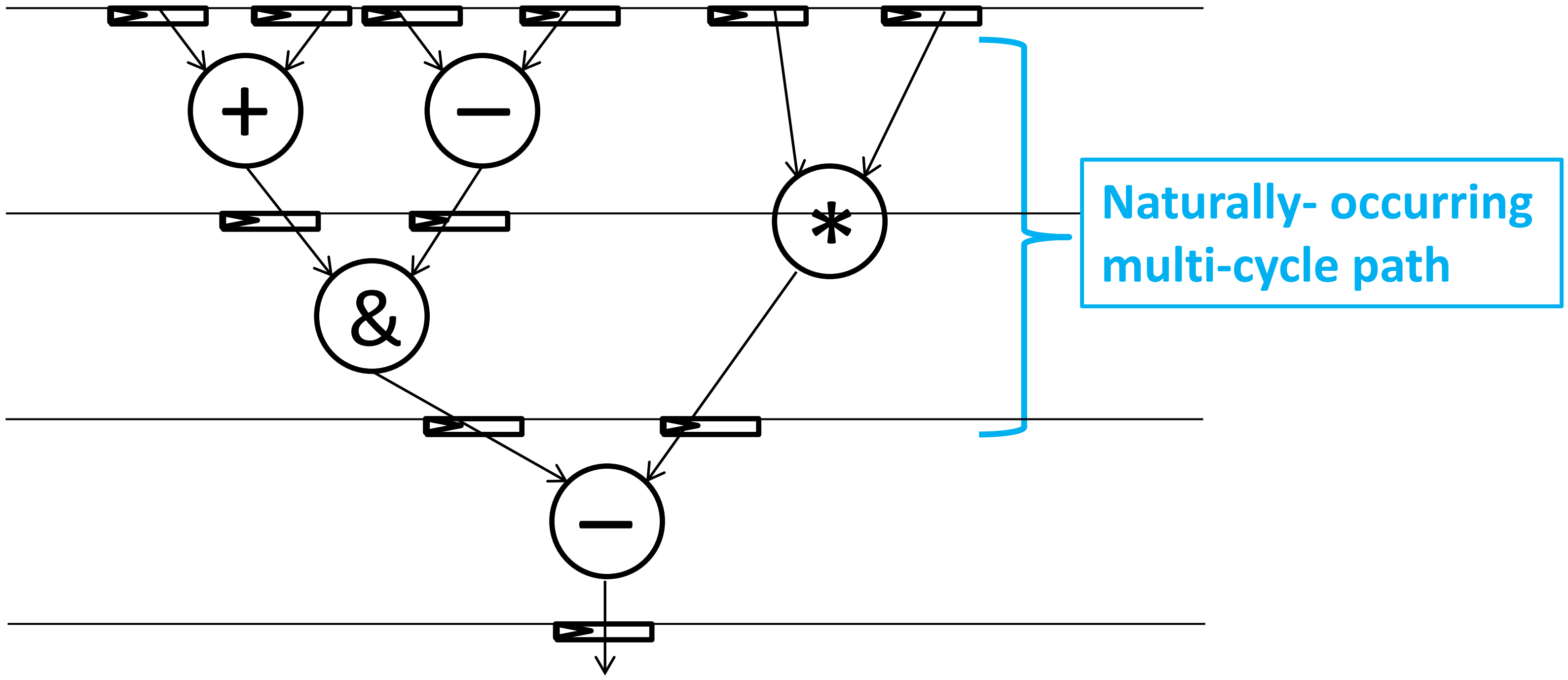
pipelined data path

path with multi-cycle constraint

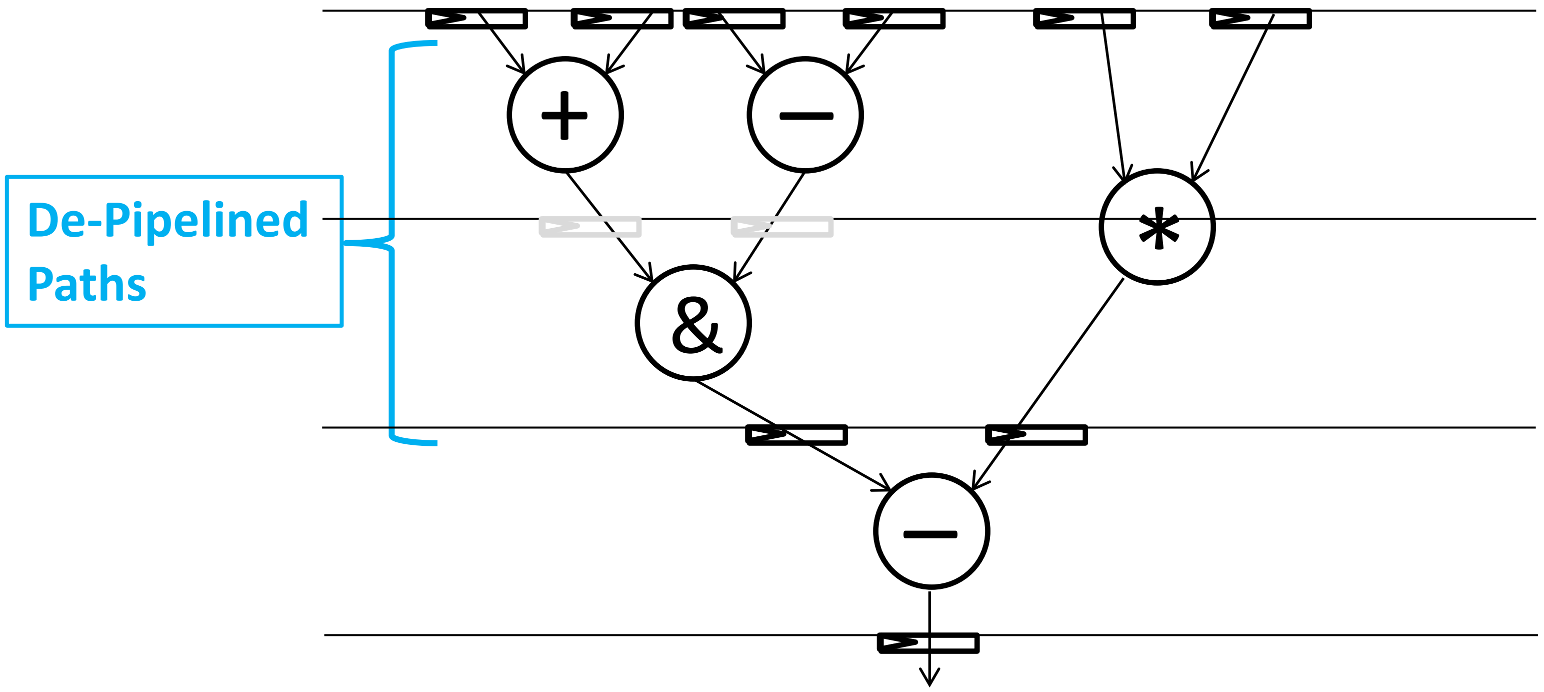
Another Example



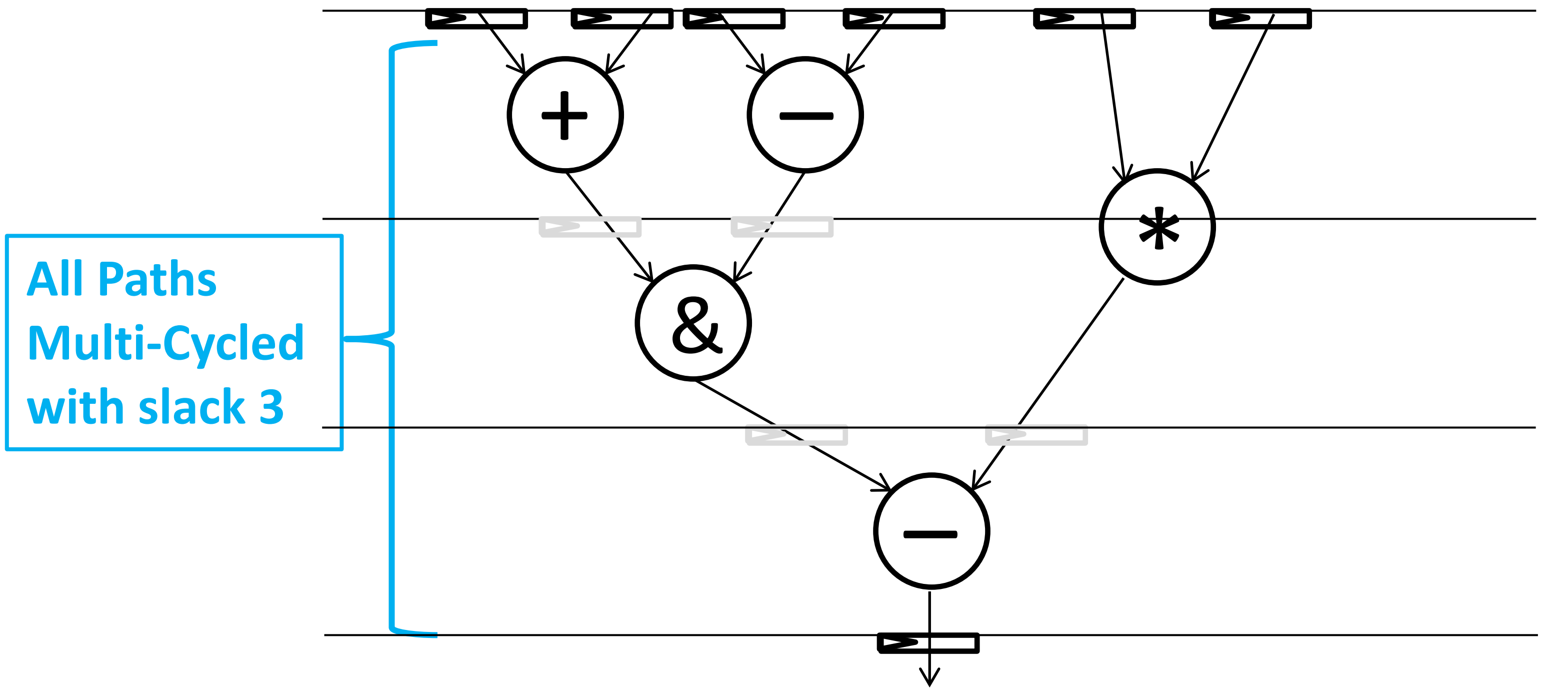
Another Example



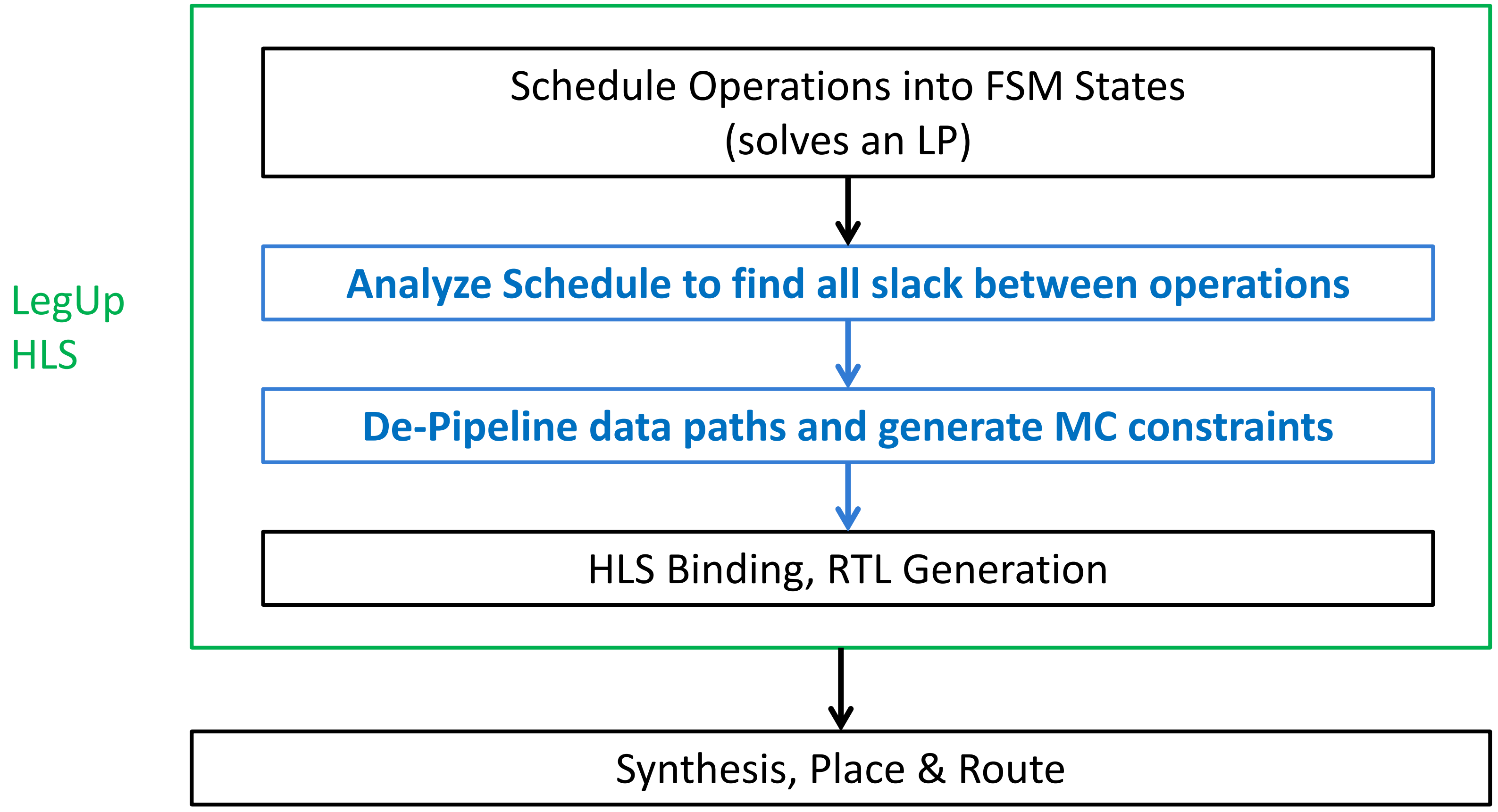
Another Example



Another Example



Static Analysis



LegUp HLS
Schedule Operations into FSM States (solves an LP)
Analyze Schedule to find all slack between operations
De-Pipeline data paths and generate MC constraints
HLS Binding, RTL Generation
Synthesis, Place & Route

De-Pipelining Algorithm

<u>Step 1</u>: identify “path separators”

- Not all registers can be removed
  - Both for correctness and speed
- Certain operations keep their registers
  - Call these **Path Separators**
  - These define start/end points of multi-cycle paths

De-Pipelining Algorithm

Step 1: identify "path separators"

1. Block RAMs
   – Loads from memory are MC path sources
   – Stores to memory are MC path destinations
2. FSM State Registers
3. Function Calls
   – Currently, functions in LegUp have registered inputs

De-Pipelining Algorithm

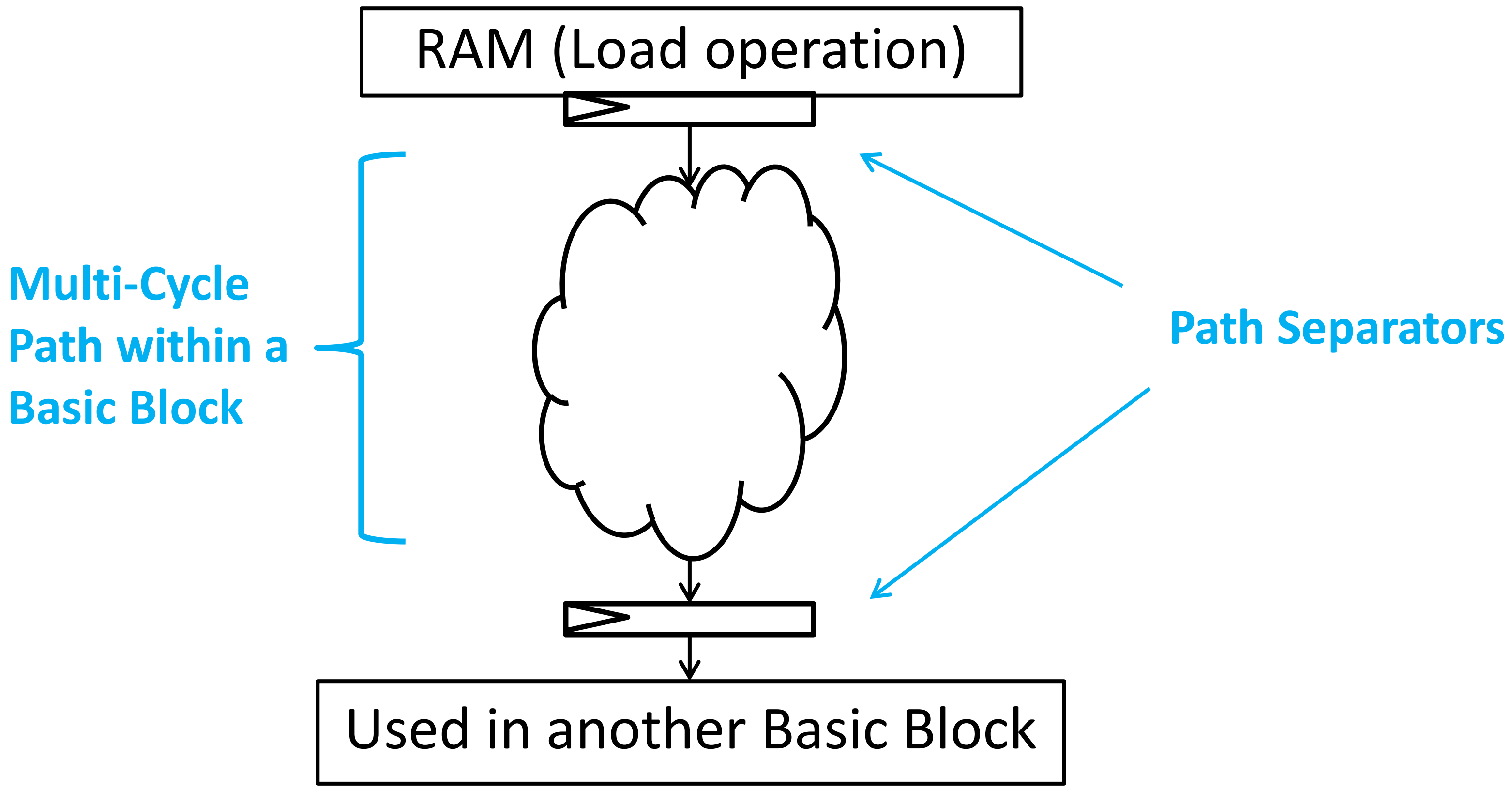
Step 1: identify "path separators"

4. Basic Block Boundaries
   - PHI operations have register inputs in LegUp
   - Computations used in a different basic block from their definition
5. Pipelined Hardware
   - E.g. dividers are still pipelined

De-Pipelining Algorithm

Step 1: identify “path separators”

- Example:


RAM (Load operation)
Multi-Cycle Path within a Basic Block
Path Separators
Used in another Basic Block

De-Pipelining Algorithm

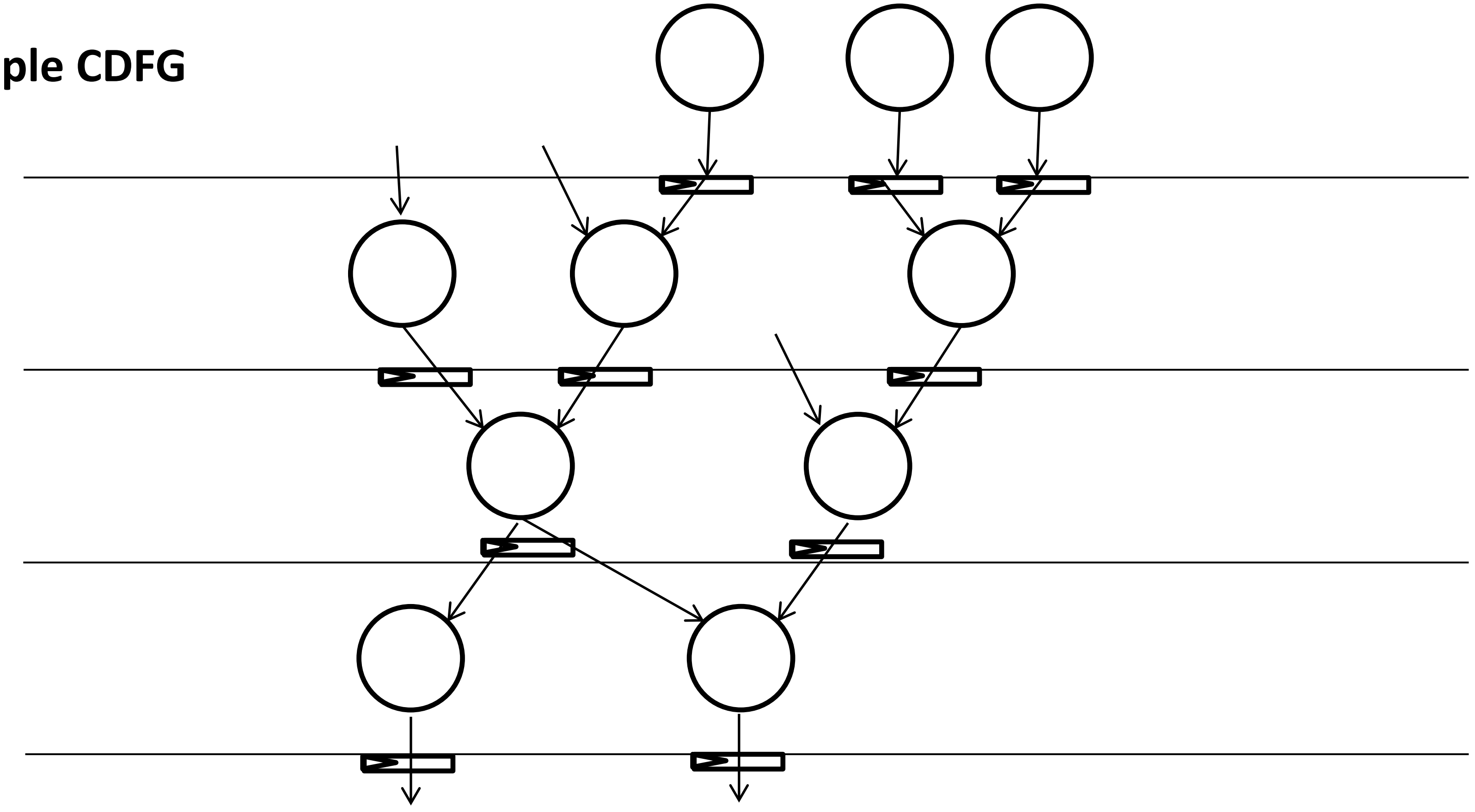
Step 2: find all paths with multi-cycle slack

- Once all separators are found, traverse CDFG to find all separator-to-separator paths
  - Control Data Flow Graph (CDFG) represents circuit
  - Each node of the CDFG is an operation
  - Some nodes are separators, rest are not
  - Use DFS to find all paths between separators
    - Algorithm 1 in paper
  - Remove registers during traversal

De-Pipelining Algorithm

Step 2: find all paths with multi-cycle slack

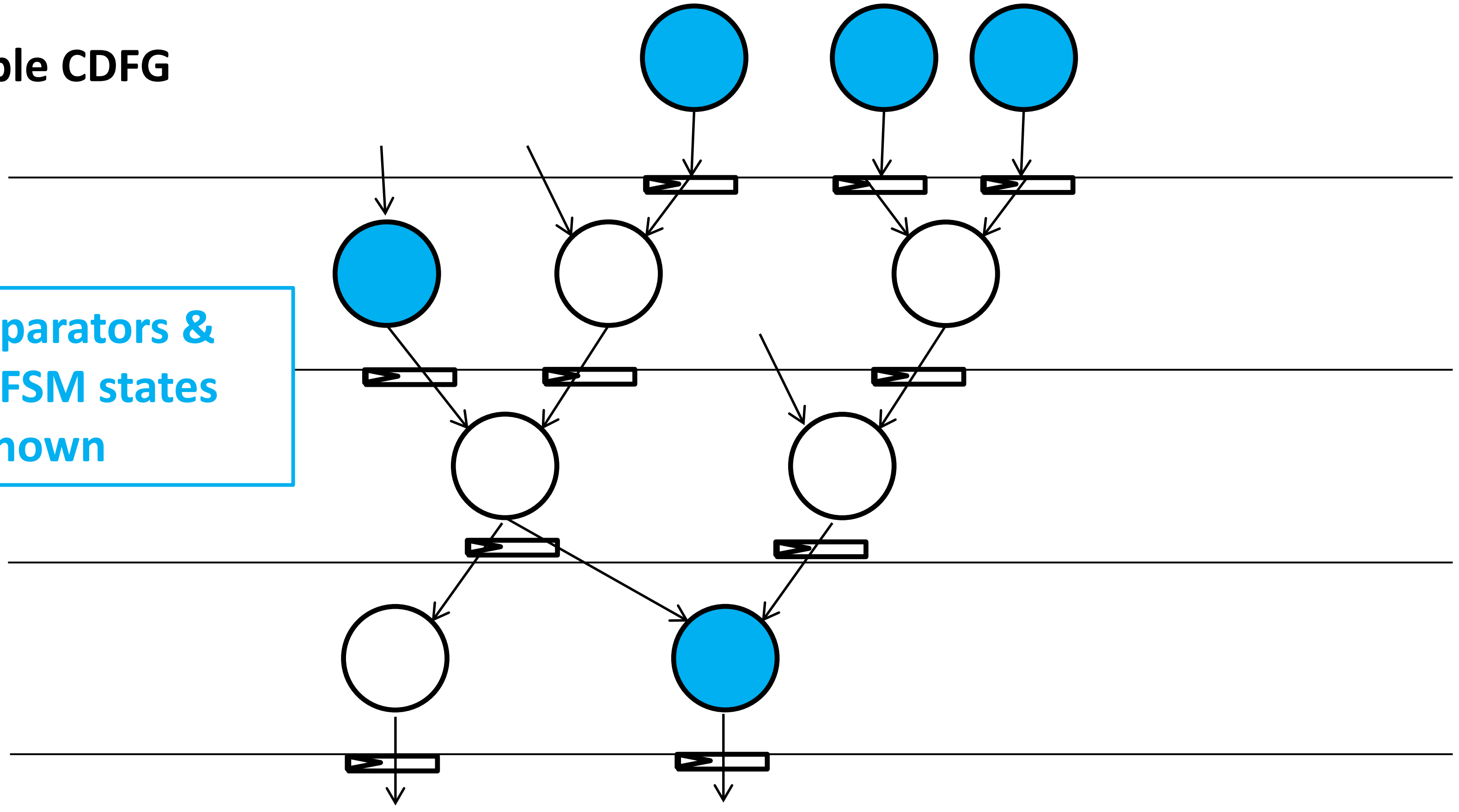
Example CDFG



De-Pipelining Algorithm

Step 2: find all paths with multi-cycle slack

Example CDFG

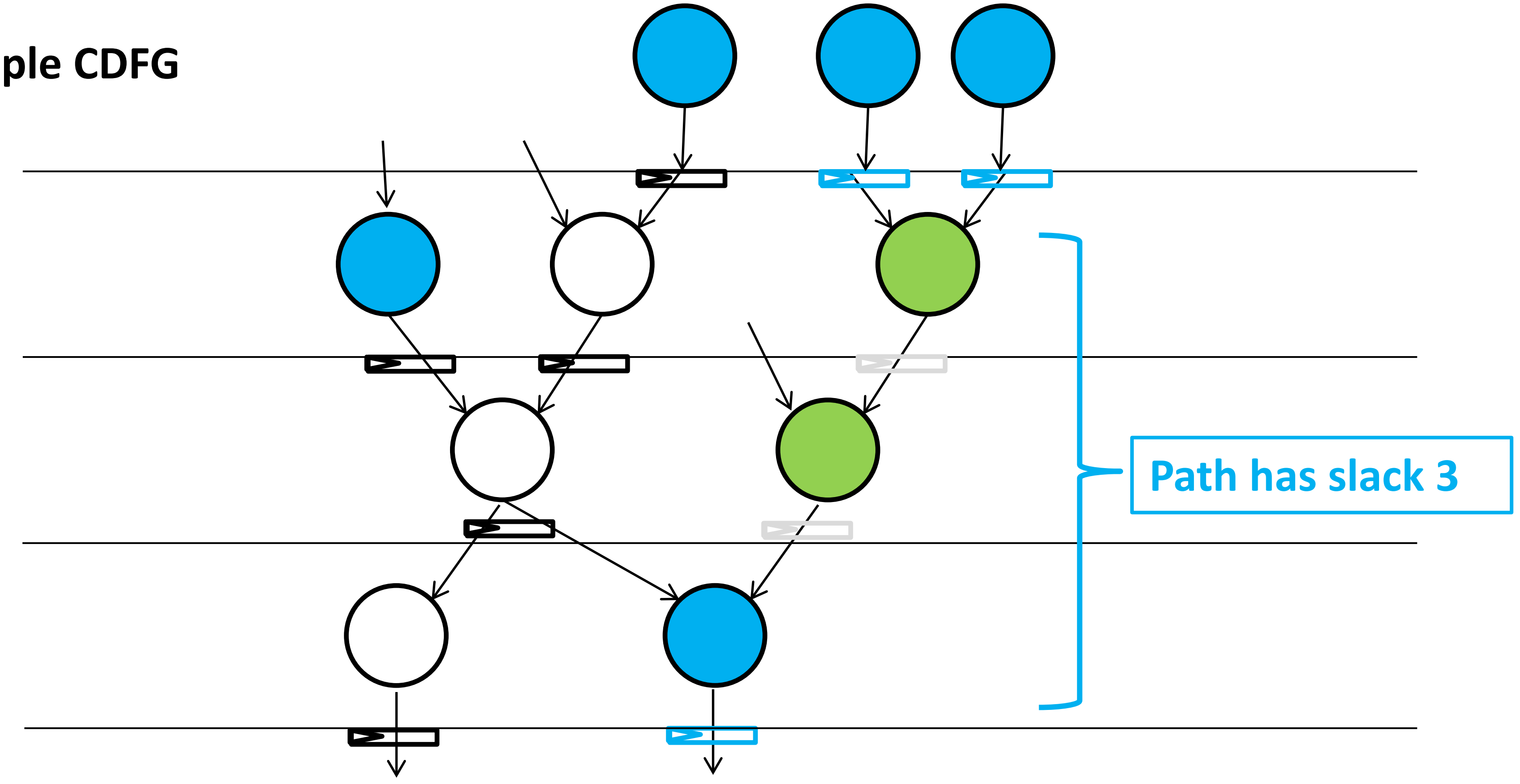


All separators & their FSM states are known

De-Pipelining Algorithm

<u>Step 2</u>: find all paths with multi-cycle slack

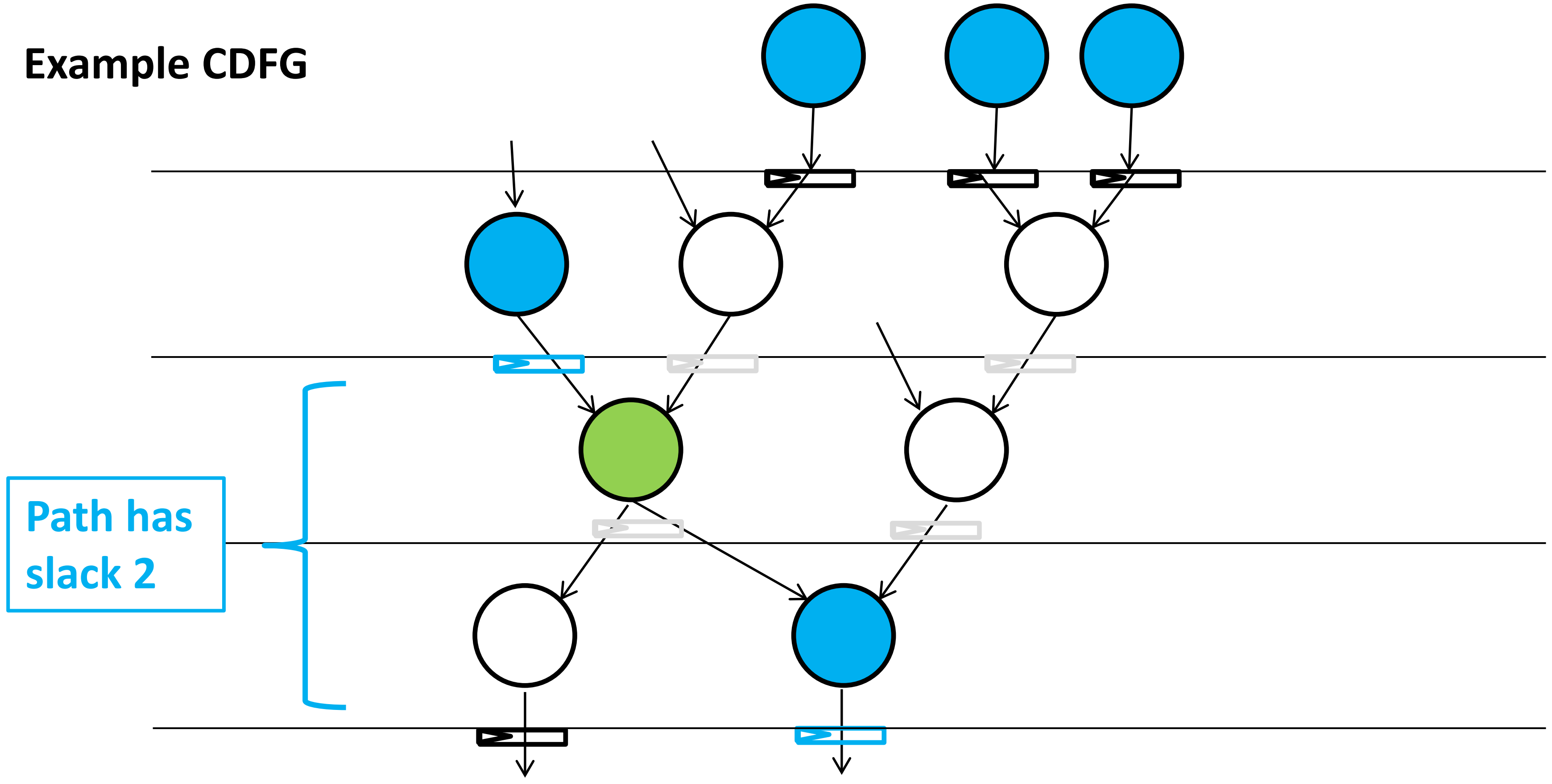
Example CDFG



De-Pipelining Algorithm

Step 2: find all paths with multi-cycle slack

Example CDFG



De-Pipelining Algorithm

Step 3: print timing constraints for all paths

For a path with multi-cycle slack of N,

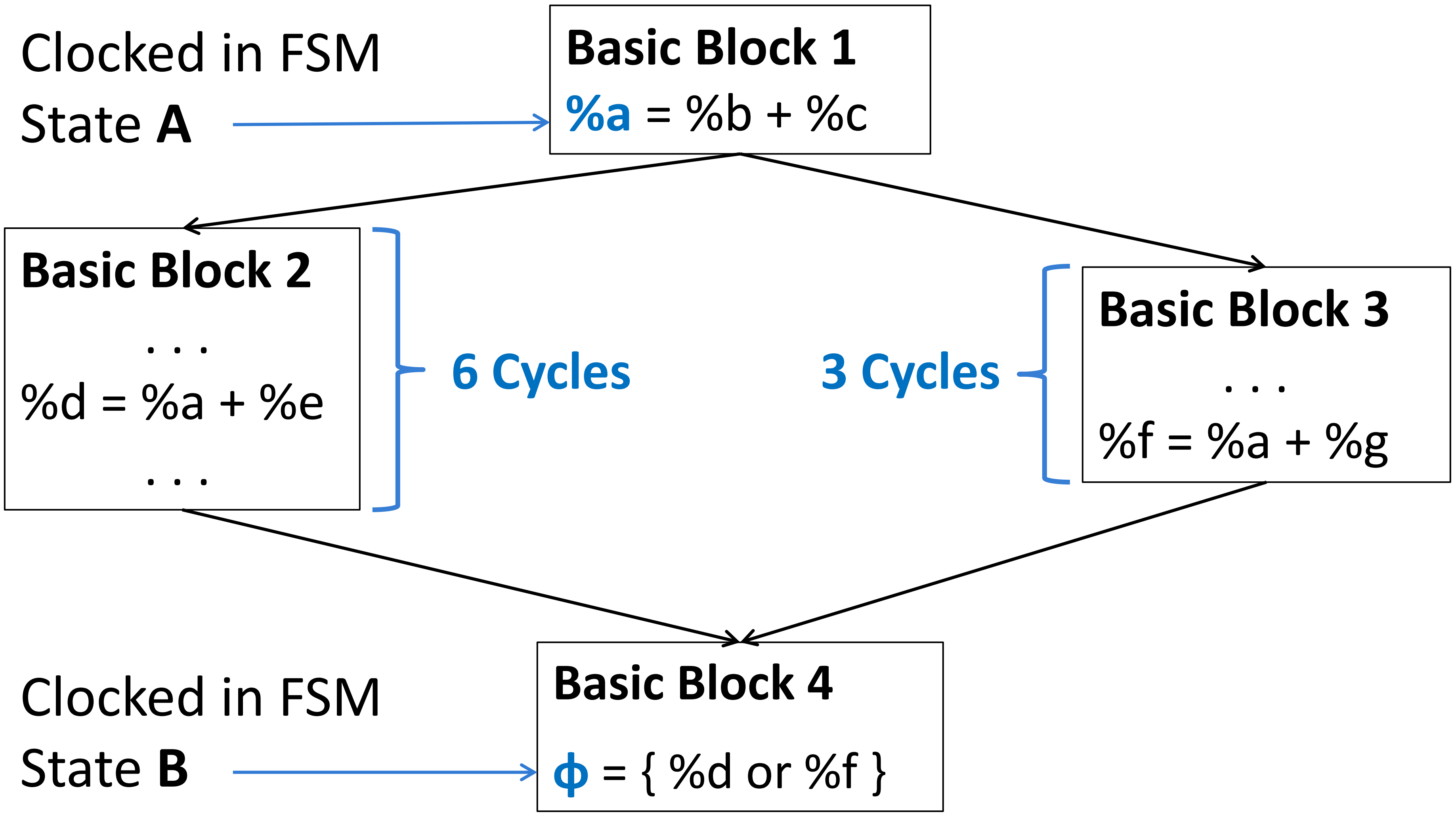
- Setup slack = N cycles
  - Take Nth edge as capturing edge[1]
- Hold slack = N-1 cycles
  - Move hold check back to the launch edge[1]

[1] *J. Bhasker and R. Chadha, Static Timing Analysis for Nanometer Designs, Springer, 2009, pp. 260–272*

Unbalanced Path Latencies

- Greatest speedups come from multi-cycle paths spanning basic block boundaries
- However, this can cause multiple paths between two separators
  - Why some registers are needed at basic block boundaries (for PHIs)
- These paths can have different slacks
- Must use minimum slack between 2 separators
  - Possible solutions discussed at length in the paper, e.g. timing constraints that specify –through signals

Unbalanced Path Latencies



Outline

- Multi-Cycle Paths in FPGA High-Level Synthesis
- Multi-Cycle Path Static Analysis
- **Profiling-Driven Multi-Cycle Path Analysis**
- Experimental Results

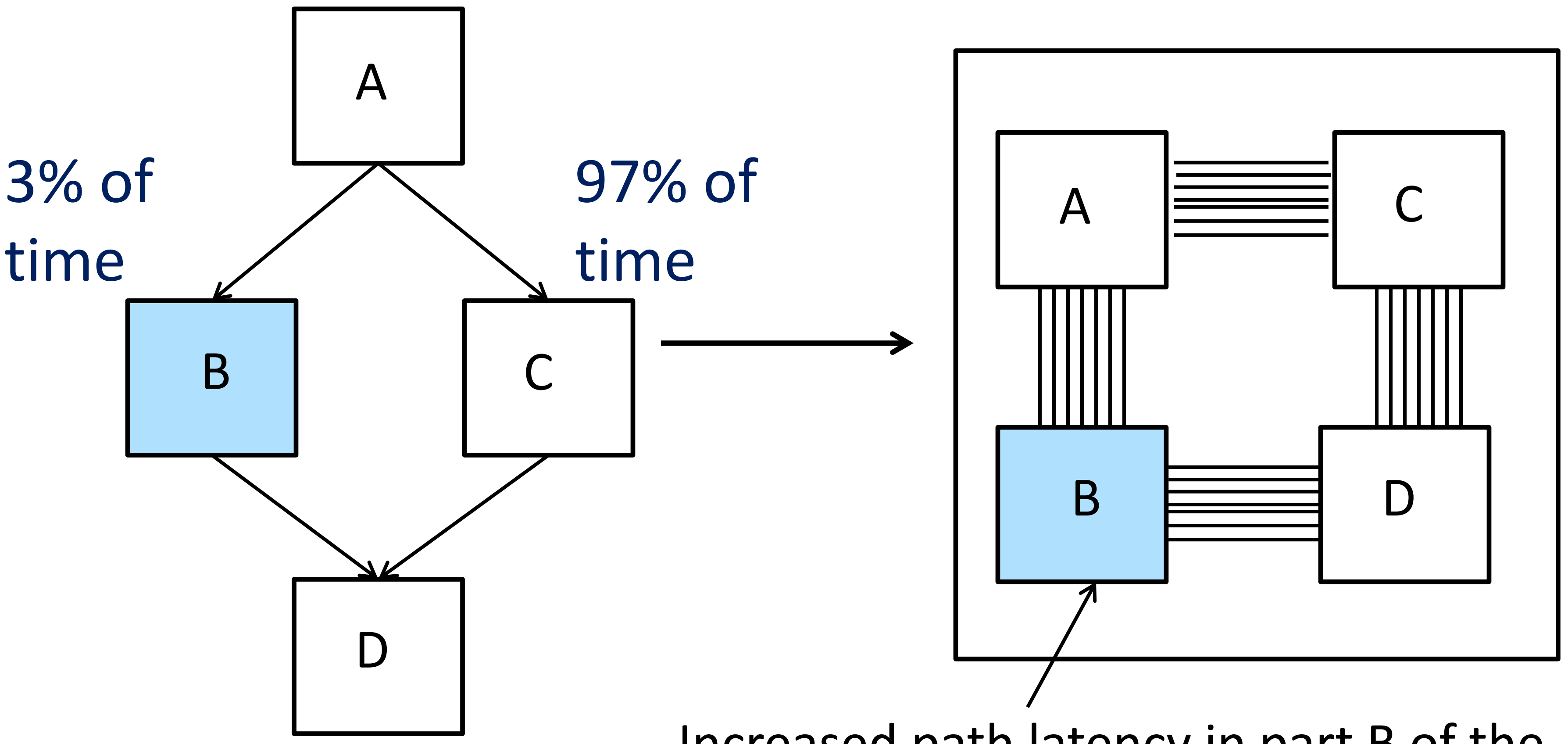
Changing the Schedule

- All analysis so far took place *after* scheduling
  - HLS schedules operations to FSM states, and then algorithm finds paths and their slacks
- We can also modify the schedule to create more multi-cycle paths

Changing the Schedule

- Extend the latency of near-critical paths
  - Common technique in manual circuit design
- Improves clock frequency
- But if paths execute too often, increased latency in one path can significantly increase latency in overall circuit
- **Only extend path latency when Fmax increase > Latency increase in the overall circuit**

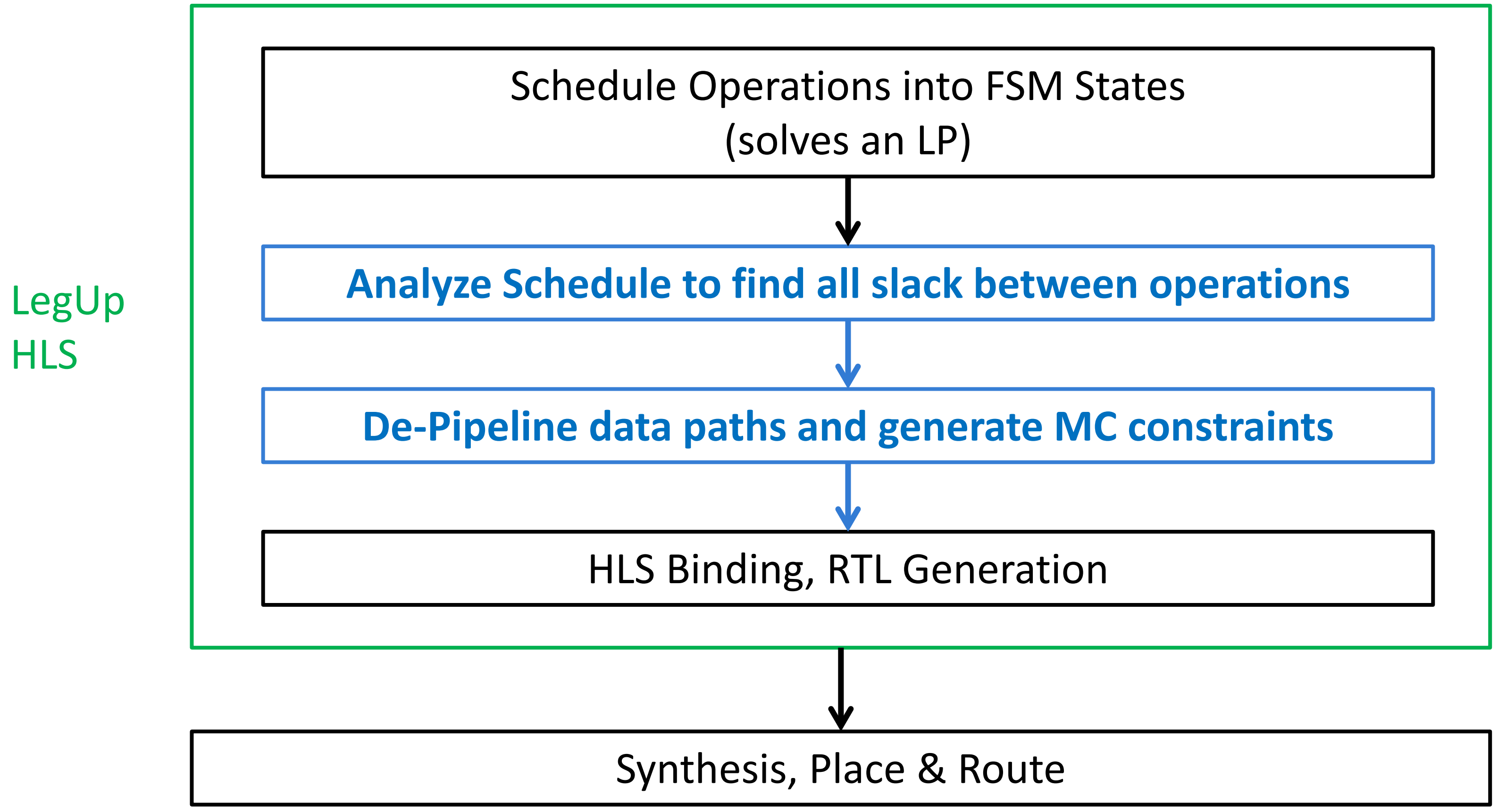
Changing the Schedule



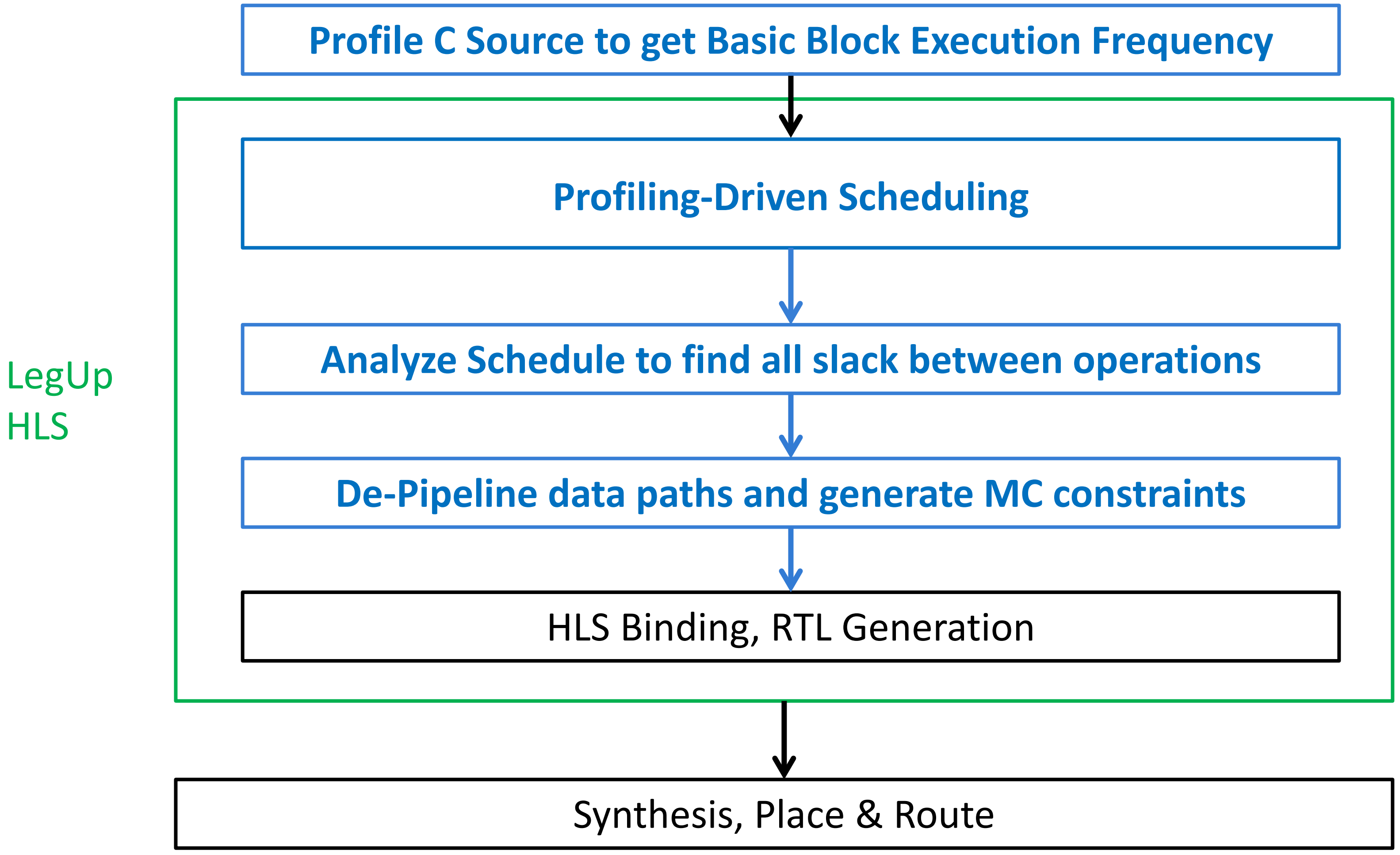
(a) Software

(b) Hardware

Static Analysis


LegUp
HLS
Schedule Operations into FSM States
(solves an LP)
Analyze Schedule to find all slack between operations
De-Pipeline data paths and generate MC constraints
HLS Binding, RTL Generation
Synthesis, Place & Route

Dynamic Analysis

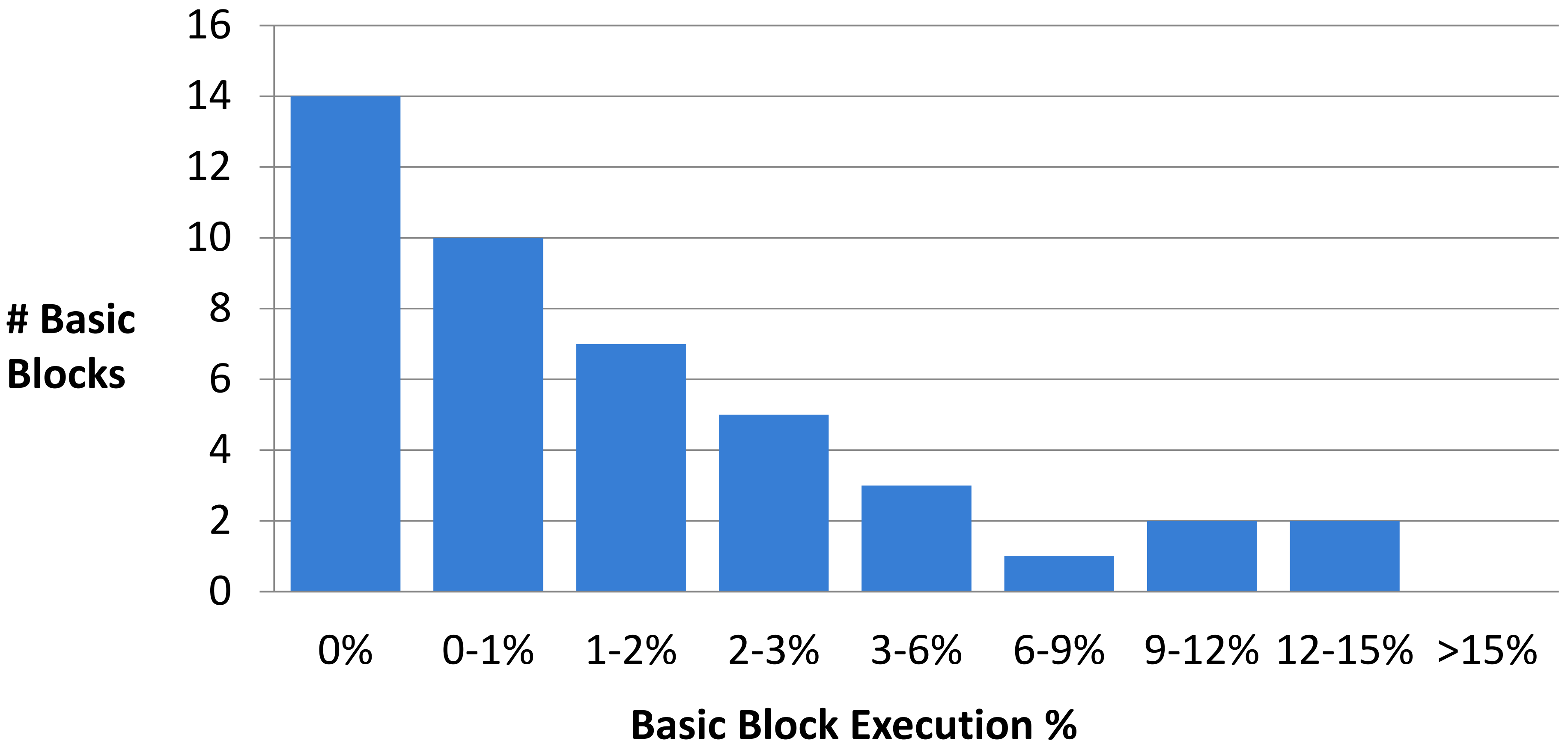

Profile C Source to get Basic Block Execution Frequency
Profiling-Driven Scheduling
Analyze Schedule to find all slack between operations
De-Pipeline data paths and generate MC constraints
HLS Binding, RTL Generation
LegUp HLS
Synthesis, Place & Route

Profiling-Driven Scheduling

- Basic Block execution frequency for an execution:

$$freq = \frac{\text{\# executions of Basic Block}}{\text{\# executions of all Basic Blocks}}$$

Basic Block Frequency Distribution



dfmul benchmark

Profiling-Driven Scheduling

- Perform initial scheduling by solving a Linear Program (LP)
  - Schedule operations into FSM states
  - Goal = minimize total # FSM states
  - Constrained by operation dependencies and combinational delay
  - "System of Difference Constraints" [1]

[1] J. Cong and Z. Zhang, "An efficient and versatile scheduling algorithm based on SDC formulation," in IEEE/ACM DAC, 2006, pp. 433–438

Profiling-Driven Scheduling

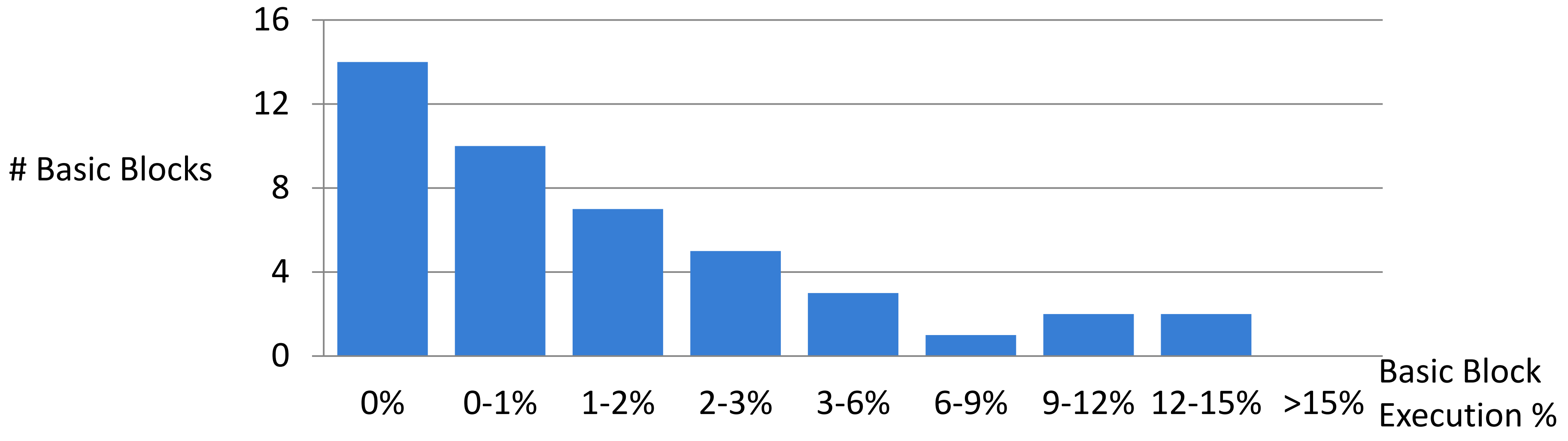
- Once initial schedule is obtained:

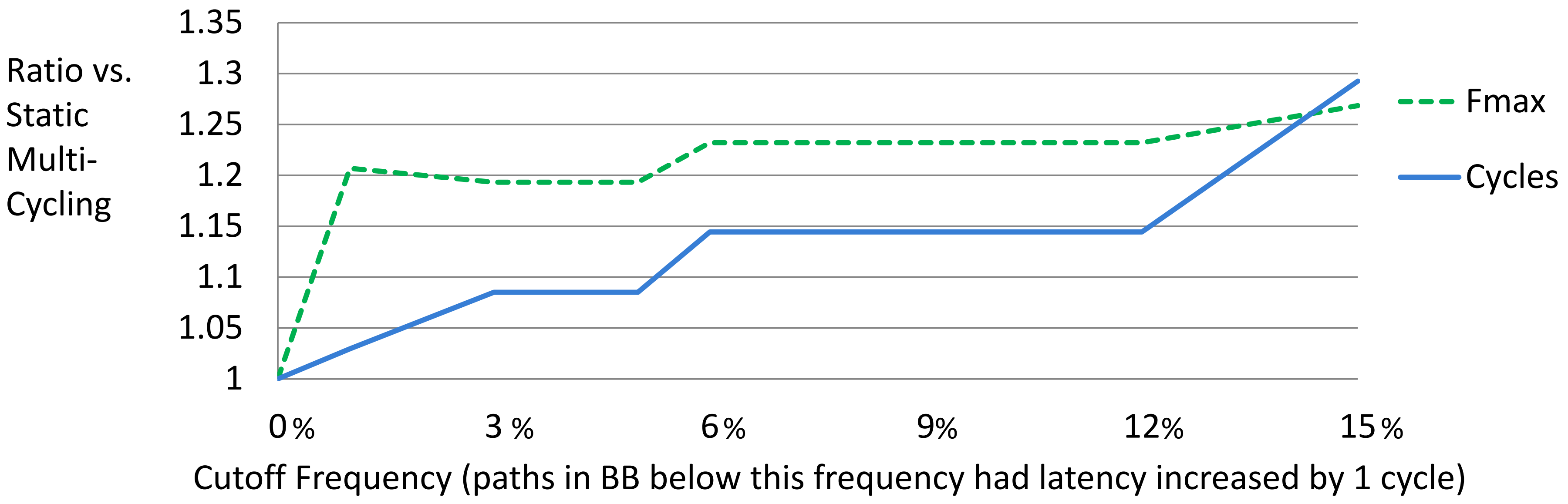
```
for all path separators S do
    B = the basic block containing S
    freq = get_execution_frequency(B)
    if freq < FREQ_THRESHOLD then
        new_state = calculate_delayed_state(S, freq)
        Add LP constraint: state[S] $\geq$ new_state
    end
end
Solve the new LP
```

Profiling-Driven Scheduling

- Key idea:
  Extend latencies of paths in infrequently executed BB
- How much to extend latency?
  - Adding 1 cycle of latency for all paths below frequency threshold worked best
  - Additional latency gave clock frequency speedups but overall execution time got worse
- What frequency threshold to use?

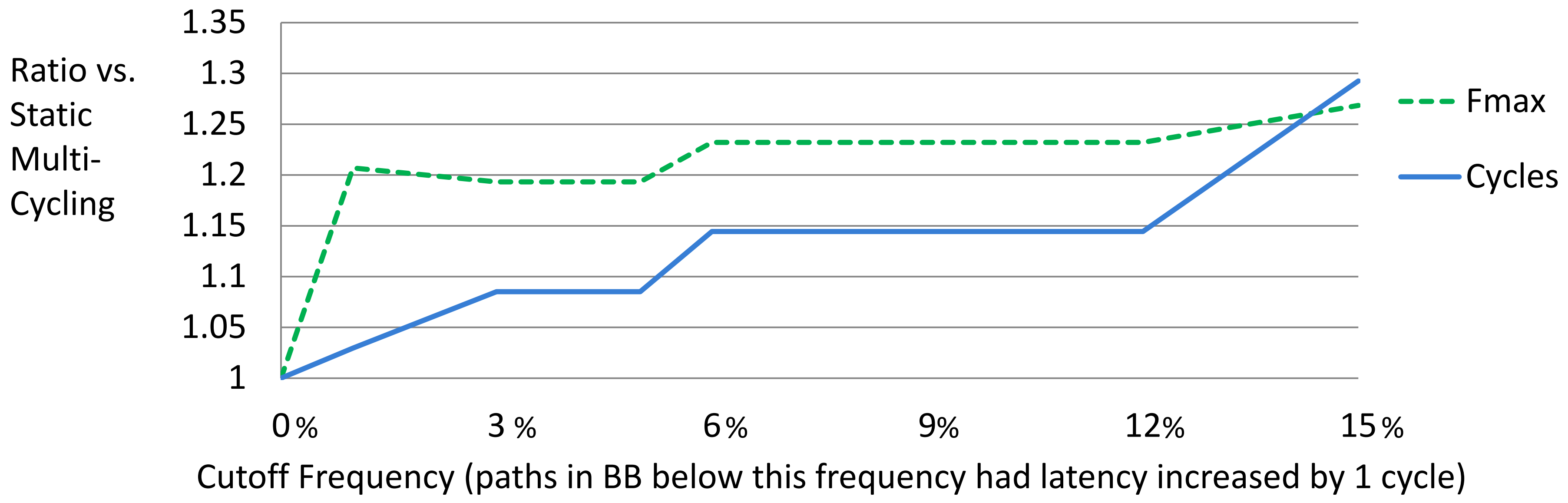
Profiling-Driven Scheduling





Profiling-Driven Scheduling

- Notice the largest "gap" occurs at lowest cutoff
- Profiling works by achieving significant frequency speedups with insignificant increase to cycle latency



Profiling-Driven Scheduling

- Frequency cutoff of 1%, 2% or 3% works best
  - Depends on circuit
  - >3% increases latency too much
  - Future work: automate cutoff frequency parameter

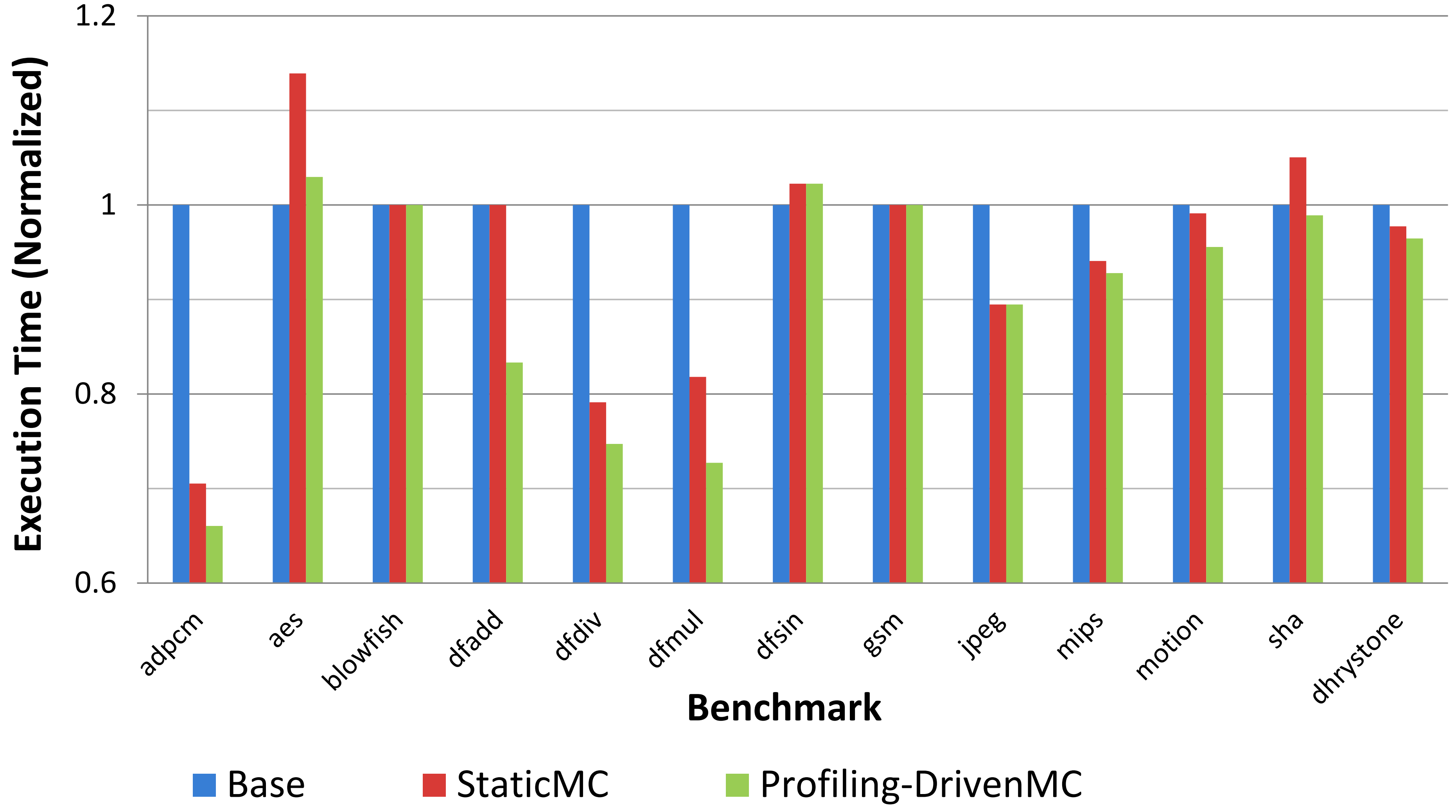
Outline

- Multi-Cycle Paths in FPGA High-Level Synthesis
- Multi-Cycle Path Static Analysis
- Profiling-Driven Multi-Cycle Path Analysis
- **Experimental Results**

Experimental Results

- Altera Stratix IV, Quartus II v. 11.1
- HLS scheduler had target clock period of 6ns
  - Experimentally gave lowest area-delay product
- Comparison:
  - **Baseline**: No multi-cycling
  - **StaticMC**: Static Multi-Cycle Analysis (no profiling)
  - **Profiling-DrivenMC**: Static and Profiling-Driven multi-cycling

Normalized Execution Time

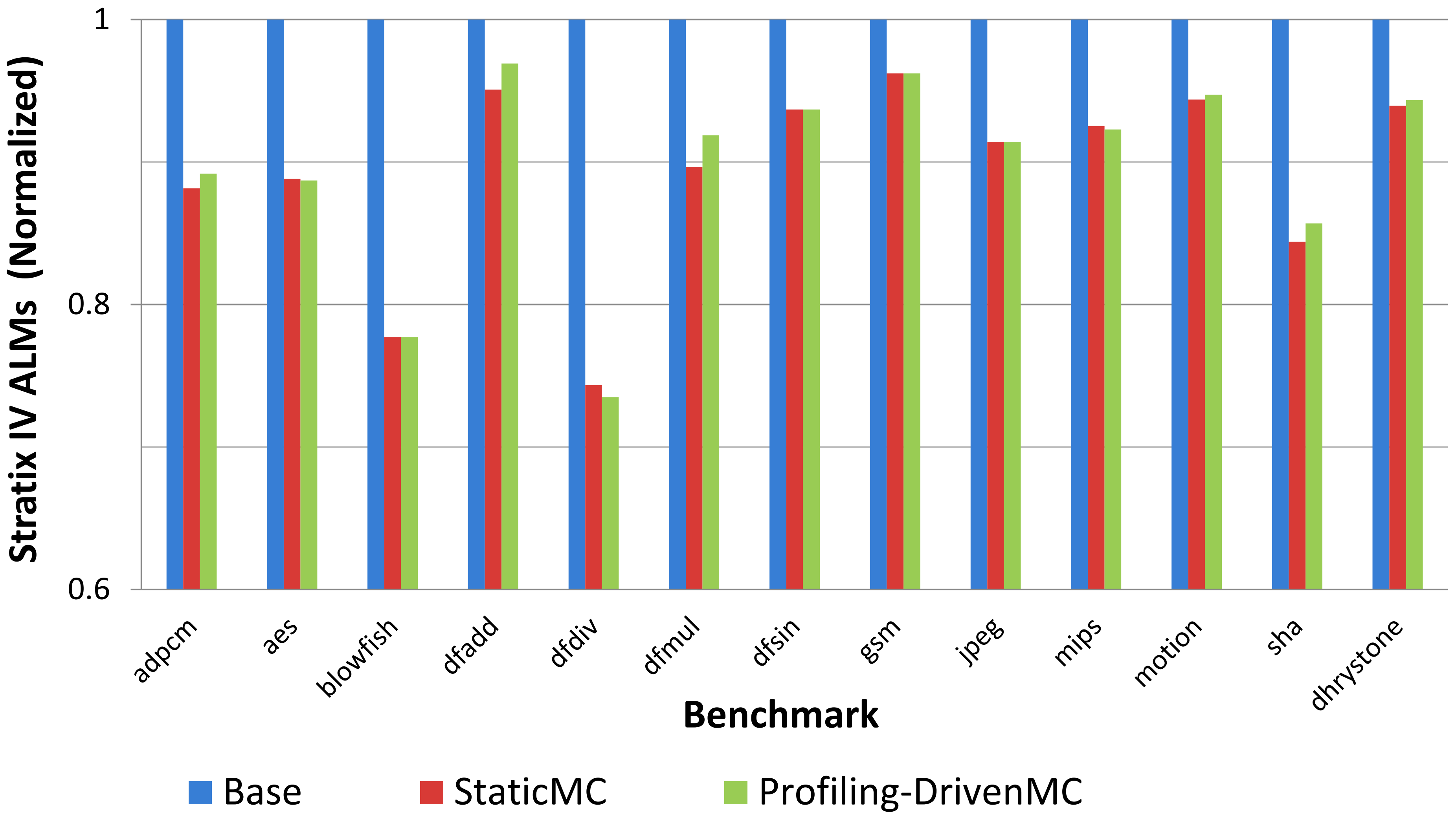


Geomean:	**1.0**	**0.945**	**0.898**

Execution Time

- As expected, not all circuits benefit from multicycling
  - If critical path is not in the datapath, adding latency slows circuits down
  - In these circuits a cutoff frequency of 0% was chosen (falling back to static multi-cycling)
- Datapath-critical circuits speed up as much as 30% from multi-cycling and an additional 17% from profiling-driven multi-cycling
- Some circuits (e.g. aes, sha) slow down due to unbalanced path latencies, this is partially fixed by profiling-driven multi-cycling

Normalized Circuit Area



	Base	StaticMC	Profiling-DrivenMC
Geomean:	**1.0**	**0.890**	**0.894**

Circuit Area

- Total area reduction of 11% (Stratix IV ALMs)
  - Register usage decreased by 26% due to de-pipelining
  - Combinational logic remains flat (decreases by 1%)
- Profiling-driven multi-cycling uses 0.4% more ALMs than static mult-cycling, due to additional FSM logic
- Total area-delay product reduced by 20% over baseline with profiling-driven multi-cycling

Conclusion

- Profiling-driven multi-cycling provides significant speedups in datapath-critical circuits
- Total area-delay product reduced by 20% over baseline with profiling-driven multi-cycling
- See full implementation at:
  - http://legup.eecg.utoronto.ca/git
- Questions?