Mapping-Aware Constrained Scheduling for LUT-Based FPGAs

Mingxing Tan, Steve Dai, Udit Gupta, Zhiru Zhang

School of Electrical and Computer Engineering
Cornell University

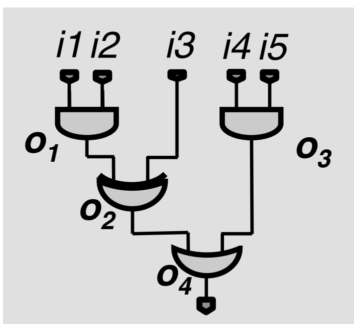
Cornell University

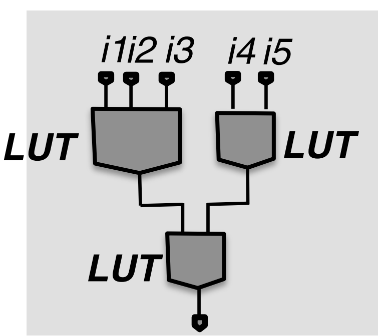
CSL

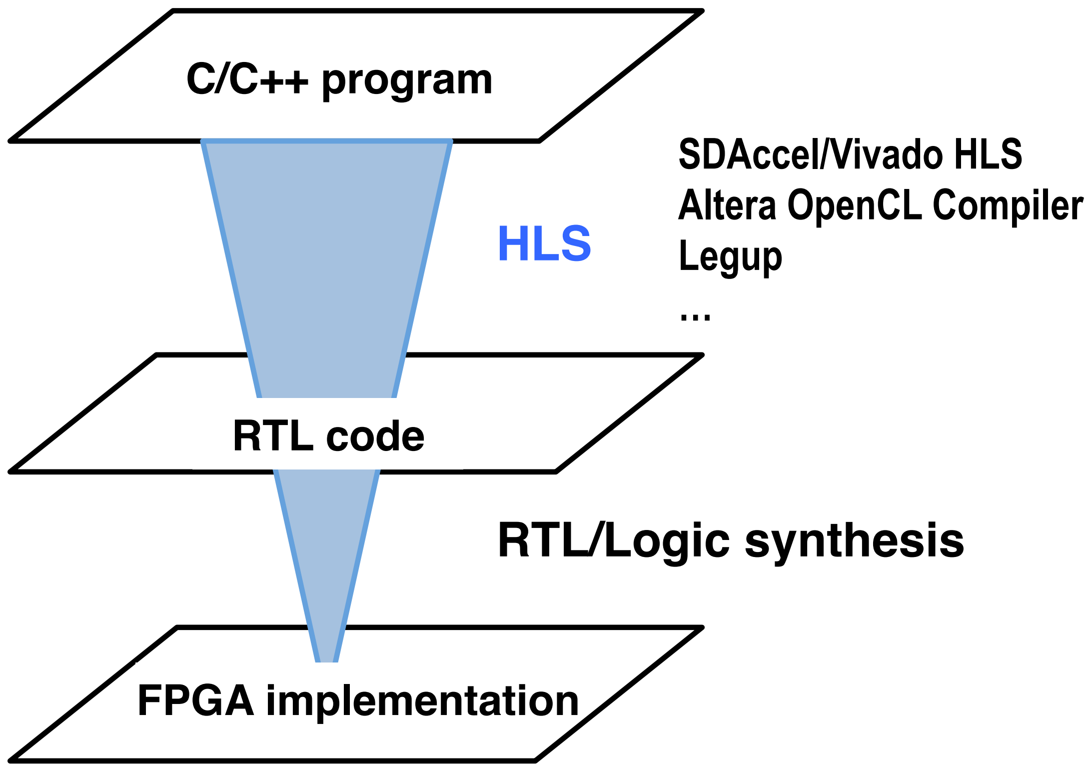
High-Level Synthesis (HLS) for FPGAs

- HLS has become increasingly important to achieve higher design productivity & quality

```
out = ((i1 & i2) ^ i3)
       ^ (i4 & i5)
```







A Typical HLS Flow

High-level Programming Languages (C/C++, OpenCL, SystemC Java, Matlab, Python ...)

→

```
if (condition) {
  ...
} else {
  t1 = a + b;
  t2 = c * d;
  t3 = e + f;
  t4 = t1 * t2;
  z = t4 – t3;
}
```

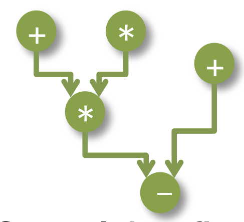
Untimed high-level description

Compilation

↓

Transformations

↓ →



Control data flow graph (CDFG)

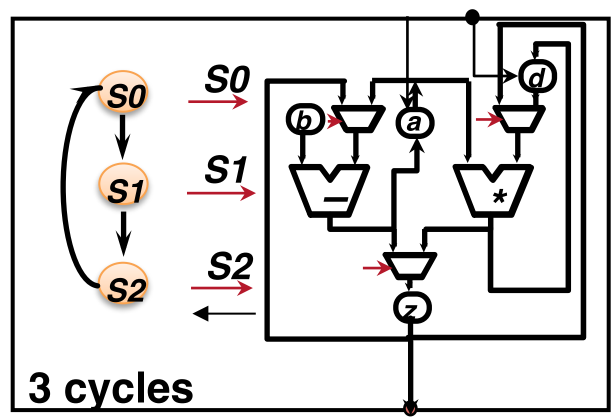
Scheduling

↓ ↑

Resource sharing

↓ →

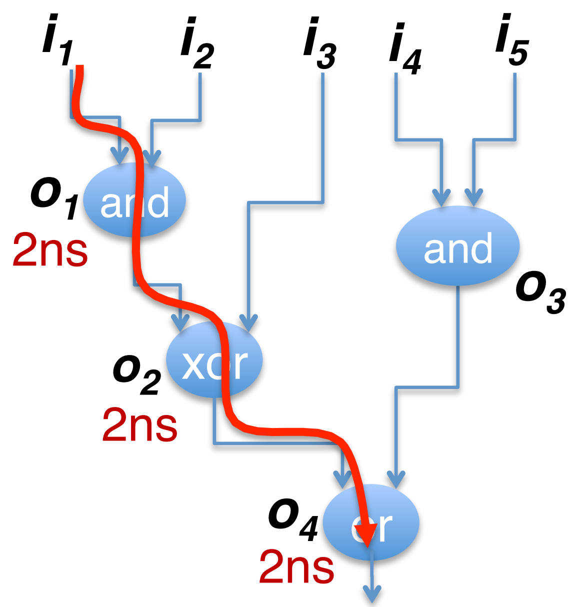
RTL generation



Timed design with clock cycles

SDC-Based Scheduling

- Scheduling based on system of difference constraints (SDC) formulation [Cong and Zhang, DAC'06]



Total delay = 6ns > 5ns

- Target clock period: 5ns
- Delay estimate: 2ns

Let $\mathbf{s_i}$ be the schedule variables

- Dependency constraints

  $\langle o_1, o_2 \rangle : s_1 - s_2 \leq 0$

  $\langle o_2, o_4 \rangle : s_2 - s_4 \leq 0$

  $\langle o_3, o_4 \rangle : s_3 - s_4 \leq 0$

- Cycle time constraint

  $o_1 \sim o_4 : s_1 - s_4 \leq -1$

- Latency constraints
- Resource constraints
- …

SDC-Based Scheduling

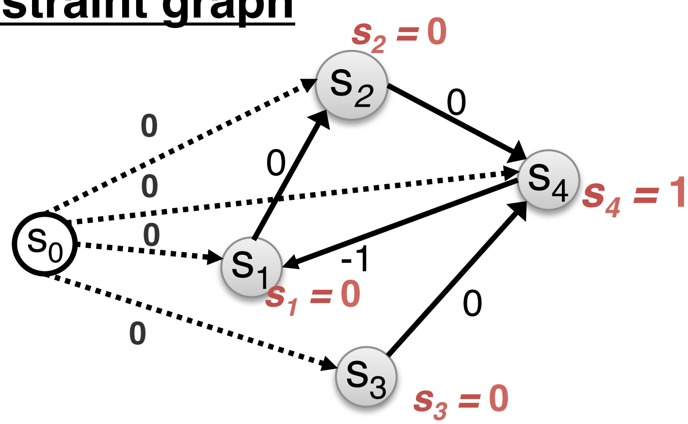
- Representing SDC constraints with constraint graph

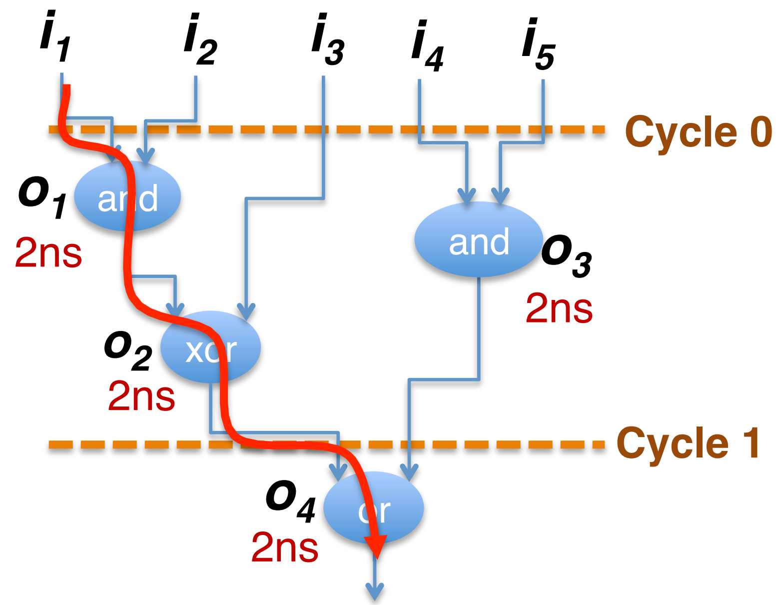
SDC constraints

s: $\{s_1, s_2, s_3, s_4\}$

$s_1 - s_2 \leq 0$
$s_2 - s_4 \leq 0$
$s_3 - s_4 \leq 0$
$s_1 - s_4 \leq -1$

Constraint graph



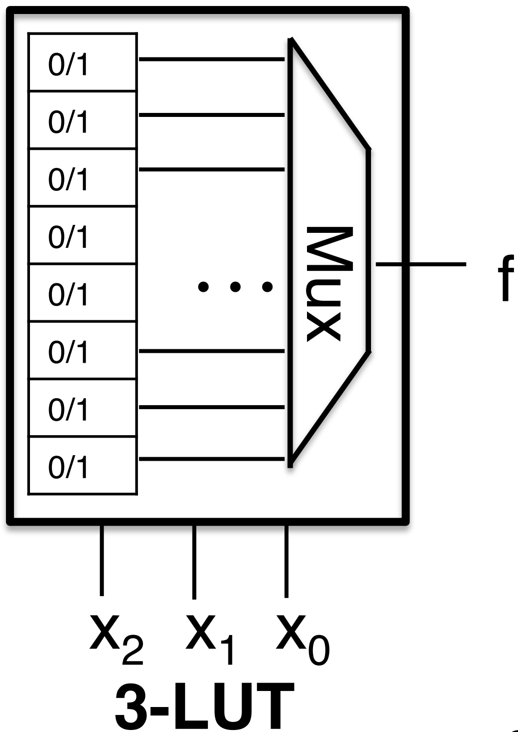


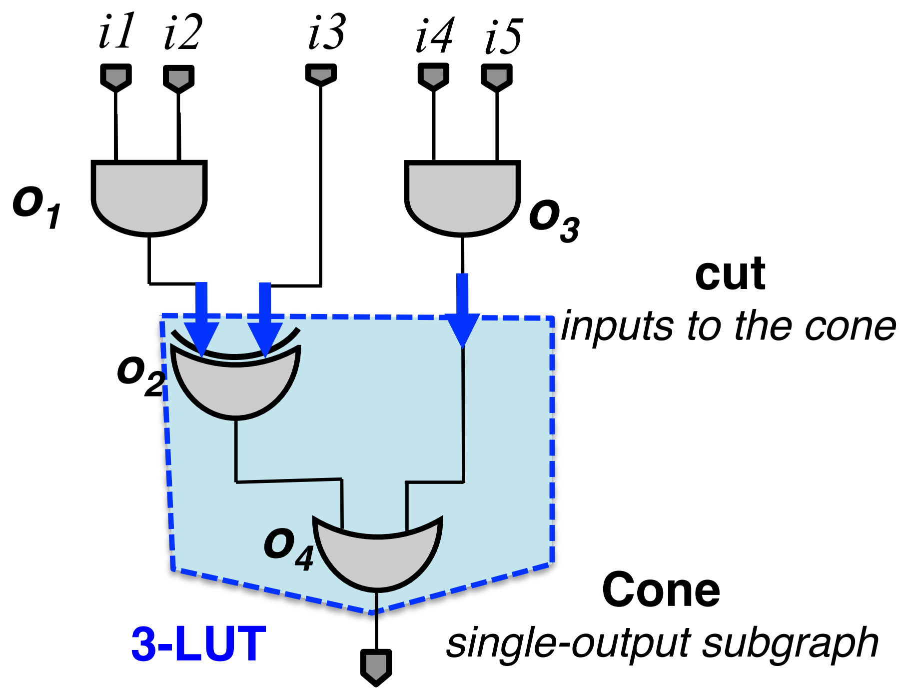
Total delay = 6ns > 5ns

Price of abstraction? Pre-characterized delay estimation for individual operation is often too pessimistic for logic operations

Lookup Tables (LUTs)

- A k-input LUT (**k-LUT**) can be configured to implement any k-input 1-output combinational logic
  - Delay is a constant for all K-LUTs

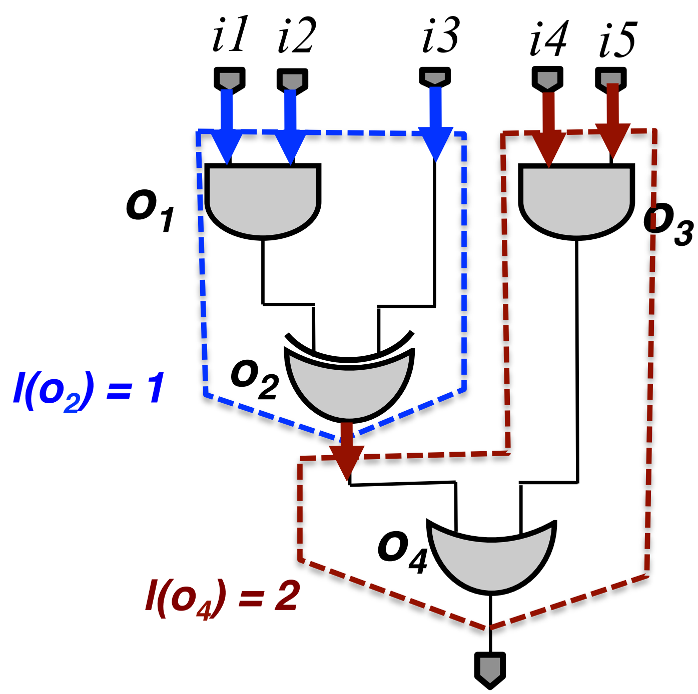




Cones and cuts are K-feasible when # of inputs ≤ K

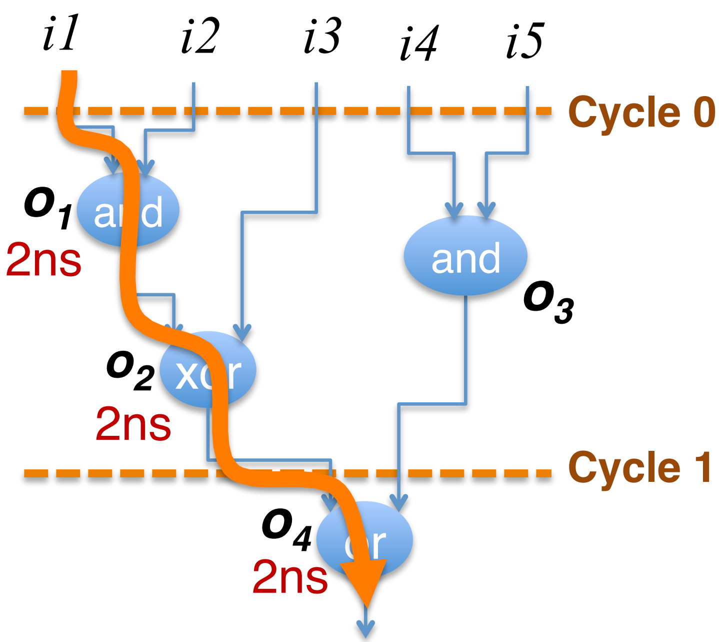
Mapping Logic Gates into LUTs

- Mapping enables more aggressive chaining by packing more operations into each cycle
  - **LUT level *l(v)* : arrive time in depth of LUTs**



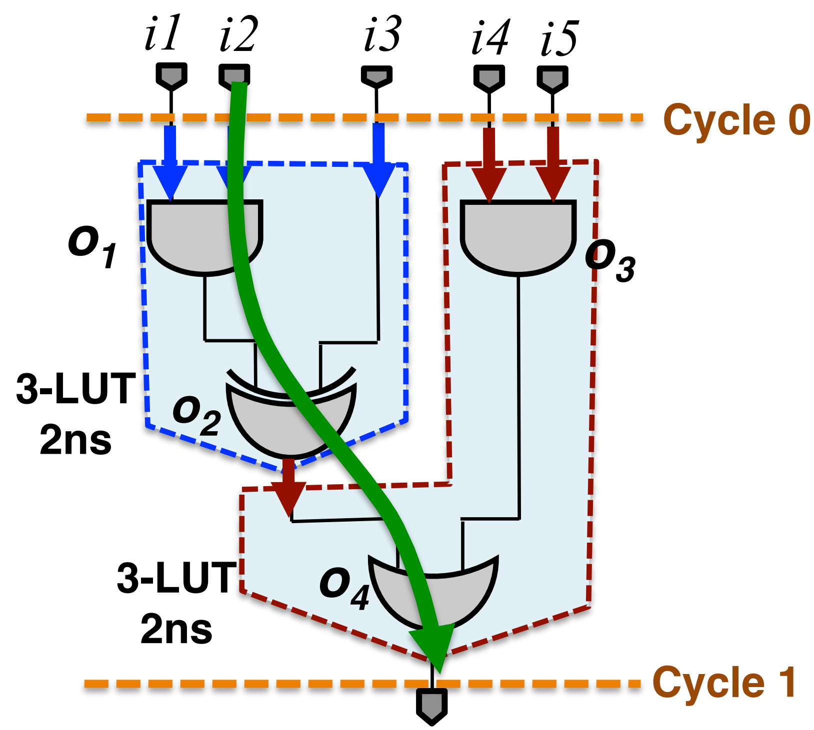
Considering Mapping in Scheduling

Conventional scheduling



Estimated delay = 6ns > 5ns, 2 cycles

Considering LUT mapping



Estimated delay = 4ns < 5ns, 1 cycle

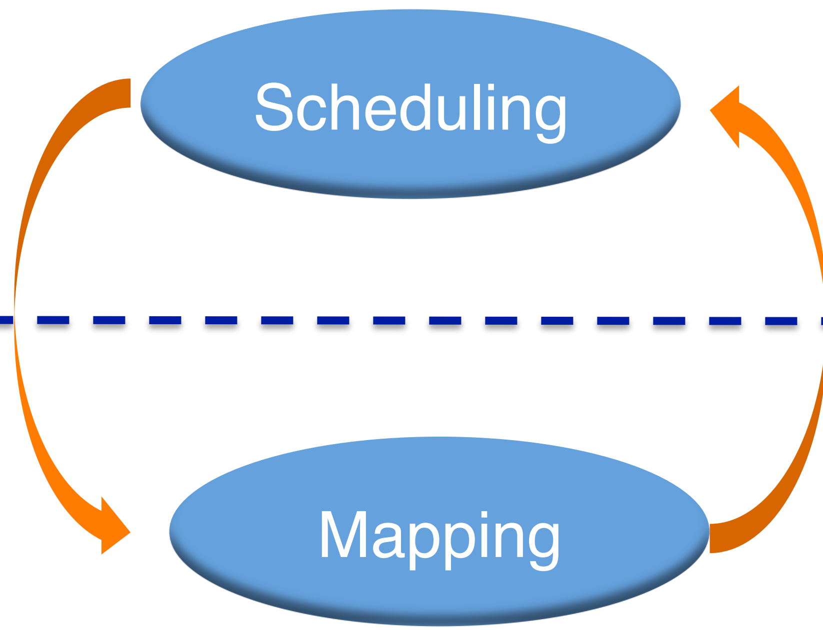
A better schedule needs to consider mapping!

Scheduling and Mapping Interdependence

HLS

Determine register boundaries

⇒ Prefer mapping information for more accurate delay estimation



Logic Synthesis

Determine LUT mapping

⇒ Typically occurs between register boundaries

MAPS: Mapping-Aware Constrained Scheduling

Idea

- Considering mapping in scheduling to enable more aggressive chaining

Contributions

- An algorithm that finds the minimum-latency schedule under SDC constraints considering LUT mapping

Results

- Significant latency reduction over a range of logic-intensive applications

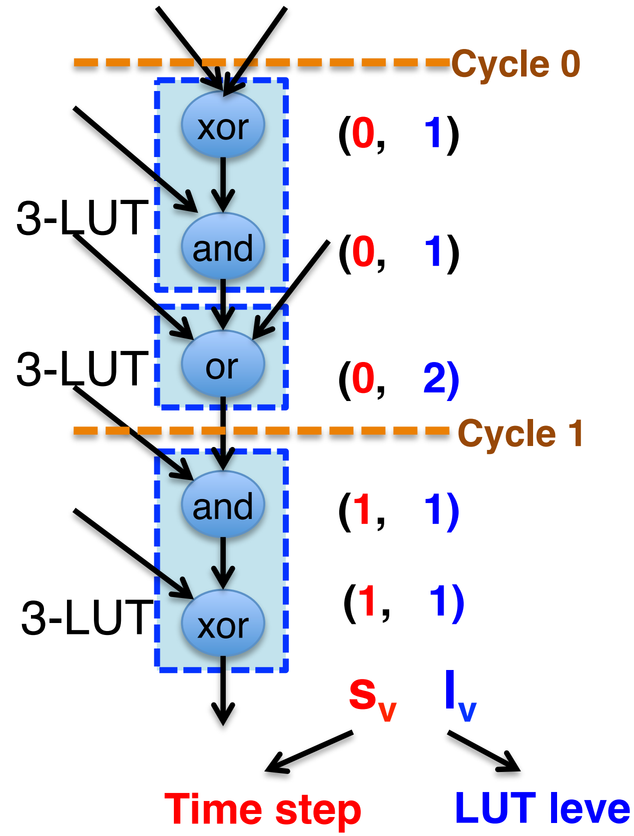
L-Values for MAPS

- We introduce *L-values* to represent the integrated scheduling and mapping information

$$L_v = (s_v,\ l_v)$$

Time step — **Inter-cycle** scheduling information

LUT level — **Intra-cycle** mapping Information



Goal: find a legal schedule with minimum L-value for each operation

Relaxation-Based Labeling

- We keep refining the lower-bound of L-values by relaxation
  - A generalization of Bellman-Ford shortest-path algorithm

Step1: **Initialize** the L-value as (0, 0), an obvious lower-bound without considering any constraint

Step2: **Iteratively** improve the L-values as follows until convergency

Each iteration

For each node on constraint graph

(1) Mapping constraints: choose the best cut with minimal L-value

$$f_v = \min_{\forall C \in CUT_v} \max_{\forall u \in C} \{L_u + (0, Delay_v)\}$$

(2) Scheduling constraints: decide new L-values according to input edges

$$g_v = \max_{\forall u \rightarrow v \in E} \{L_u + (Lat_{u \rightarrow v}, 0) + (0, Delay_v)\}$$

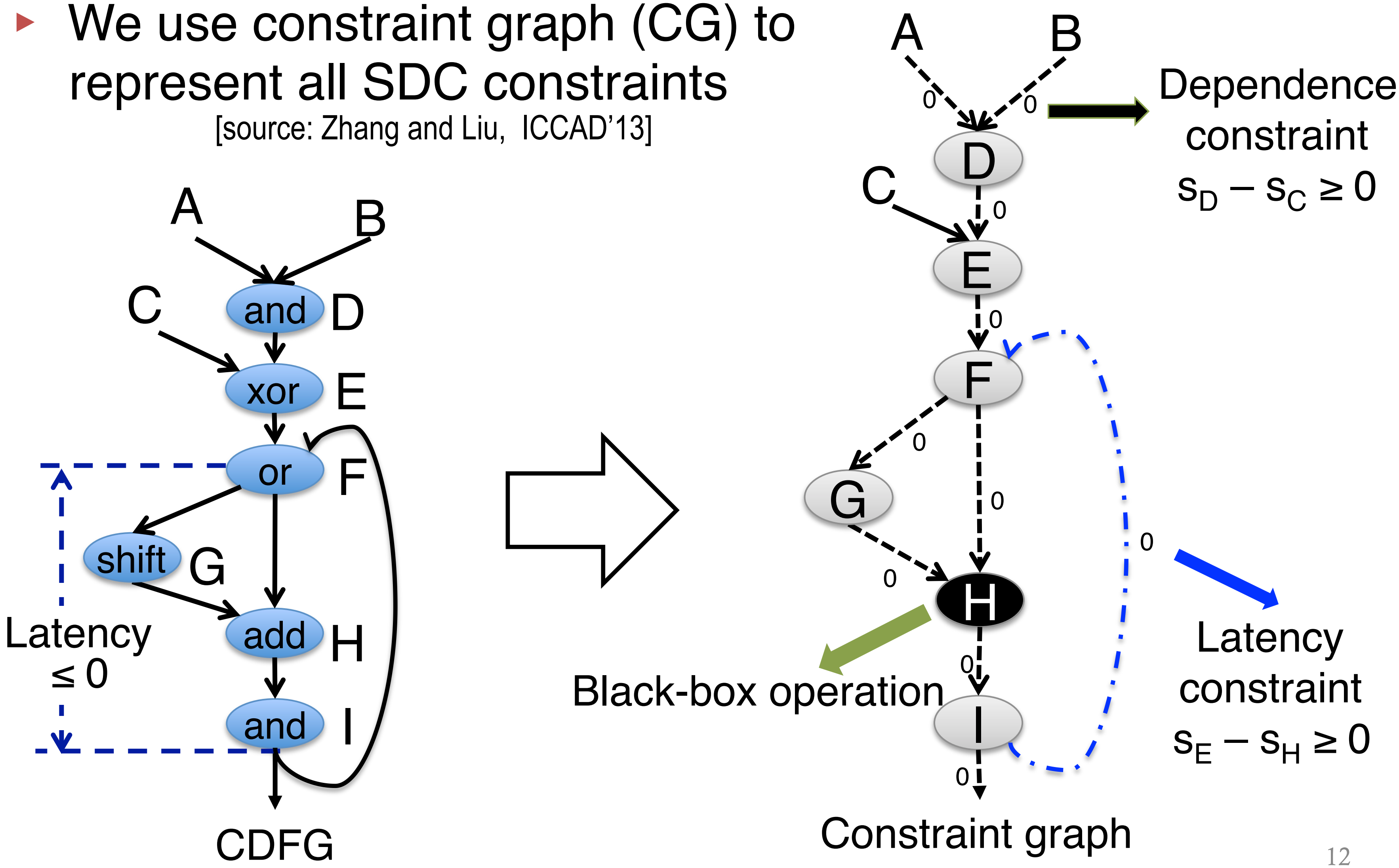
(3) Update the L-value based on mapping and scheduling constraints

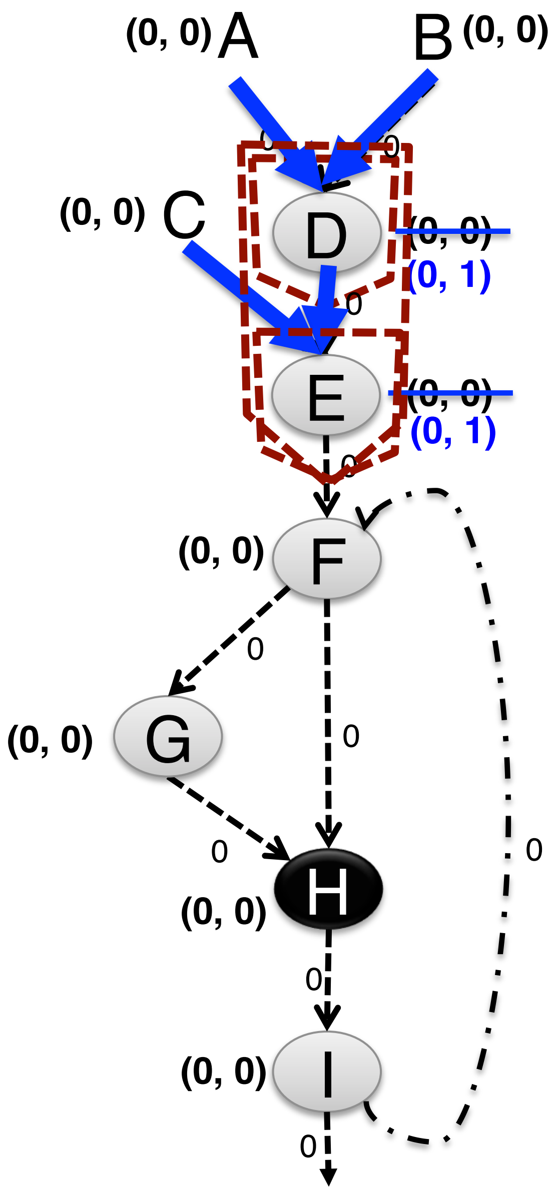
$$L_v = \max\{f_v, g_v\}$$

Return a legal schedule when the algorithm converges

From CDFG to Constraint Graph

- We use constraint graph (CG) to represent all SDC constraints
  [source: Zhang and Liu, ICCAD'13]





Constraint graph

Relaxation-Based Labeling

Assuming 3-LUTs and LUT level ≤ 3

Step1: Initialize the L-value as (0,0) for each node

Step2: Iteratively update L-values by relaxation

Iteration 1

Node D

Propagate L-values for mapping constraints

LUT level

$$f_v = \min_{\forall C \in CUT_v} \max_{\forall u \in C} \{L_u + (0, Delay_v)\}$$

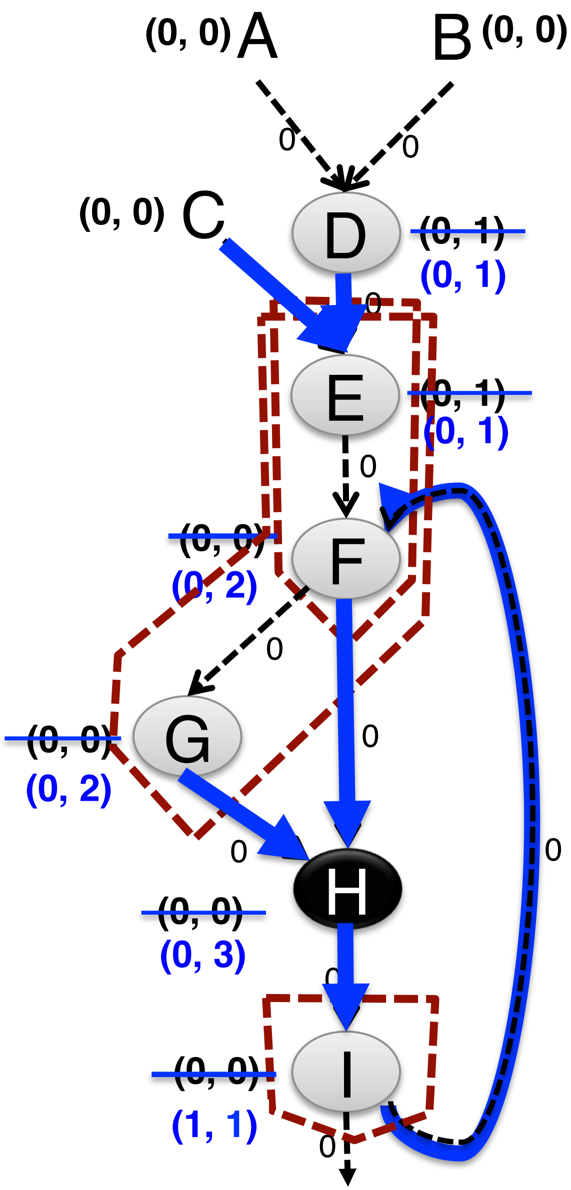
L_B: (0, 0)

Cut{A, B} L_D: (0, 1)

Cut1{C, D} L_E: (0, 2)

Cut2 {C, A, B} L_E: (0, 1)

Choose the best cut with minimal L-value



Constraint graph

Relaxation-Based Labeling

Assuming 3-LUTs and LUT level≤ 3

Step1: Initialize the L-value as (0,0) for each node

Step2: Iteratively update L-values by relaxation

Iteration 1

Node I

Propagate L-values for mapping constraints

Black-box operation H has only trivial cut

Maximum LUT level is restricted by cycle time

Time step LUT level

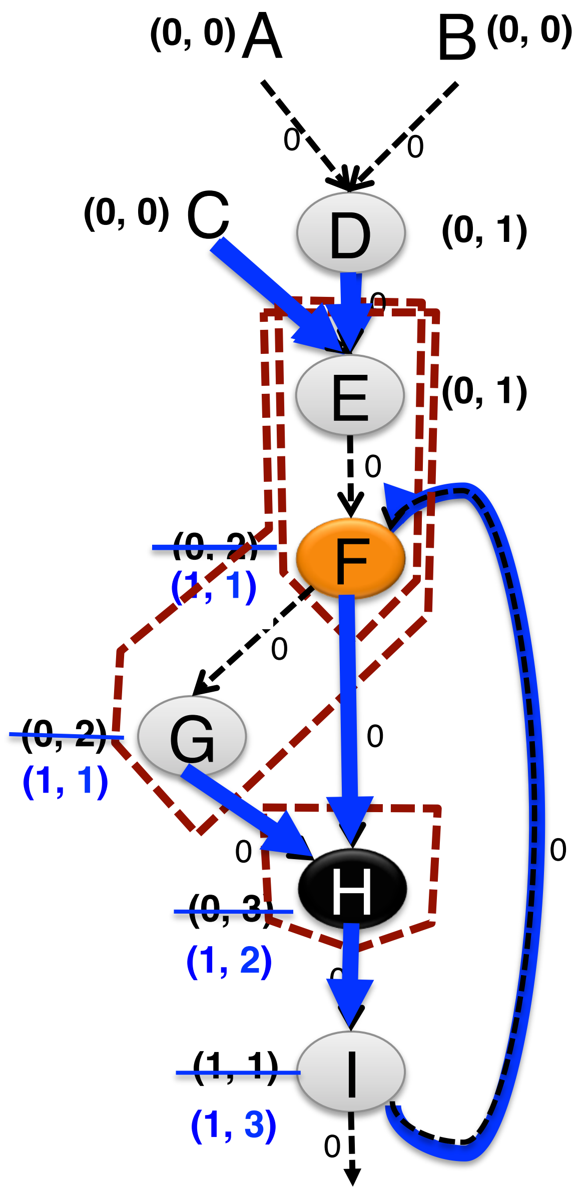
L_H: (0, 3)

L_I: ~~(0, 4)~~ (1, 1)

LUT level≤ 3

Relaxation-Based Labeling

Assuming 3-LUTs and LUT level≤ 3



Constraint graph

Step1: Initialize the L-value as (0,0) for each node

Step2: Iteratively update L-values by relaxation

Iteration 1 ⇨ **Iteration 2**

Node I

Scheduling constraints

$$g_v = \max_{\forall u \to v \in E} \{L_u + (Lat_{u \to v}, 0) + (0, Delay_v)\}$$

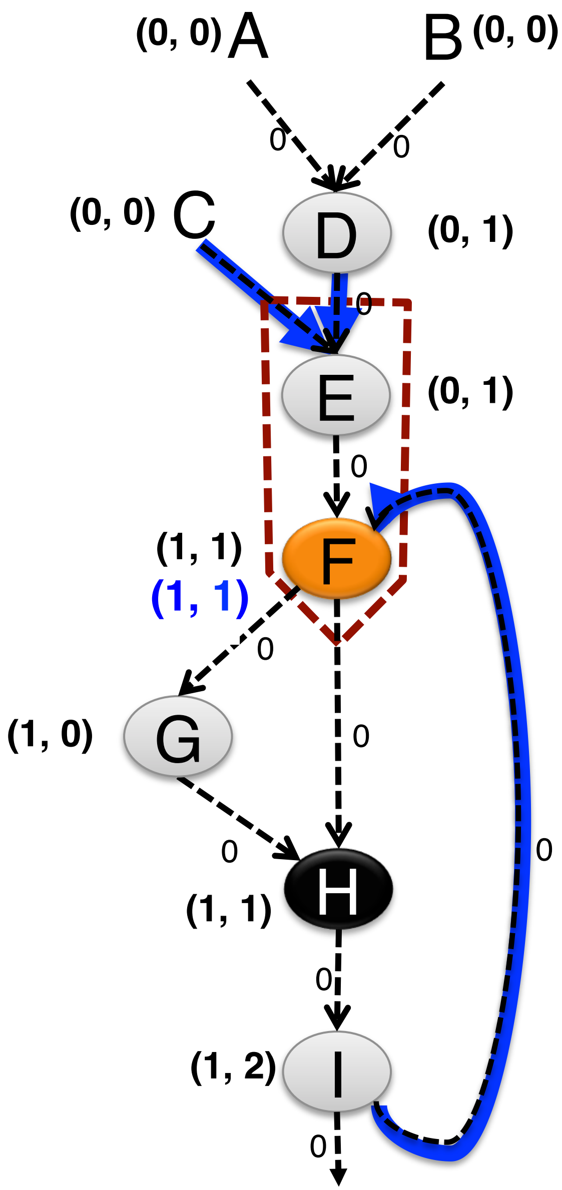
Back edge from I to F: maximum latency constraint
F and I must be in the same cycle

L_I: (1, 1)

L_F: (1, 1) L_G: (1, 1)

L_H: (1, 2)

L_I: (1, 3)



Constraint graph

Relaxation-Based Labeling

Assuming 3-LUTs and LUT level≤ 3

Step1: Initialize the L-value as (0,0) for each node

Step2: Iteratively update L-values by relaxation

Iteration 1 ⇨	**Iteration 2** ⇨	**Iteration 3**

Node F

Mapping constraints: satisfied

Scheduling constraints: F and I are in the same cycle

Every node reaches its minimum legal L-value!

The algorithm converges

Optimality of MAPS Labeling

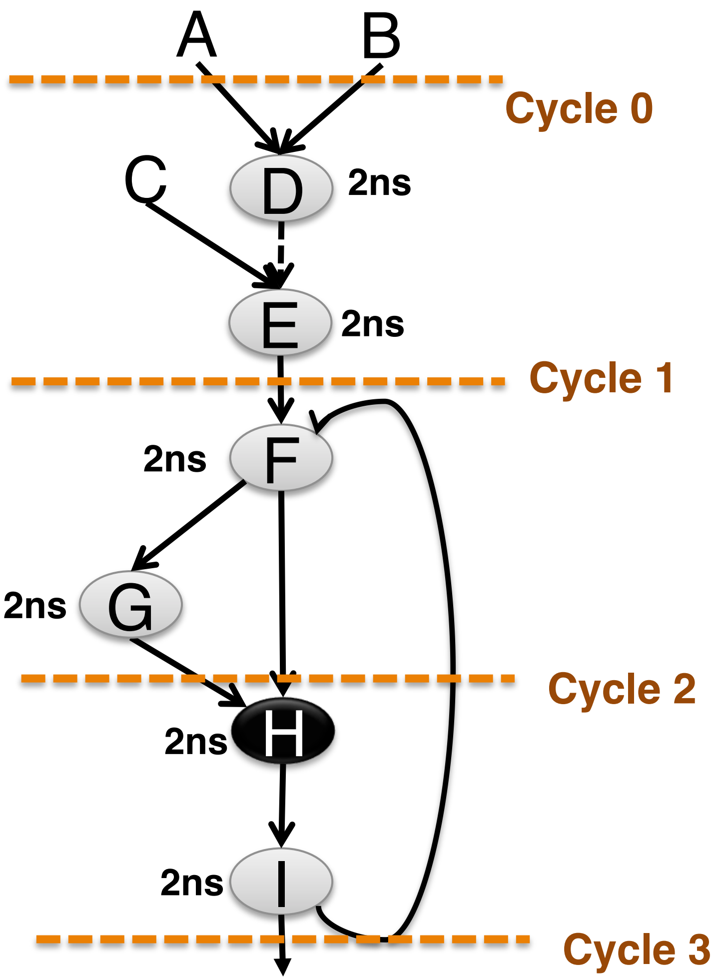
- Proof by induction that MAPS labeling algorithm always maintains the **lower bound** of L-values

Base Case: All L-values are initialized as (0,0), which are the lower bound without considering any constraints;

Induction: assume **iteration k** maintain the lower bound of L-values,
=> L-values in iteration **(k+1)** are also lower bound,
because our algorithm only monotonically increases minimal L-values that satisfy a part of the given constraints

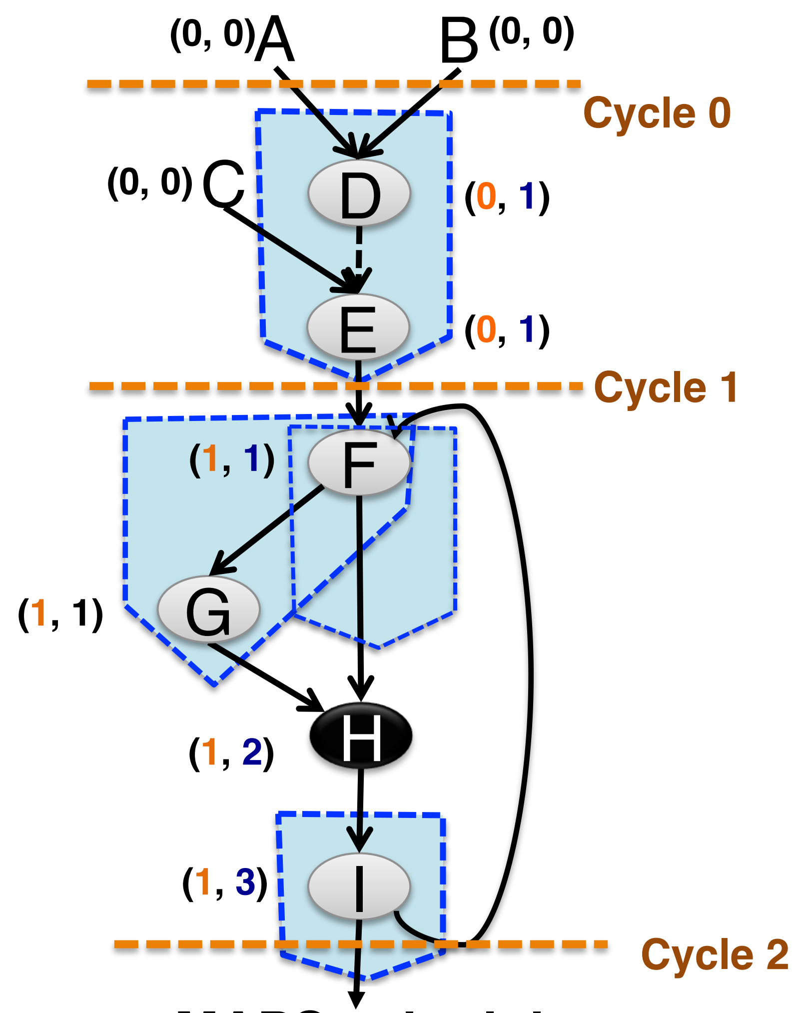
Upon convergence, MAPS returns a legal schedule with a minimum L-value for each node

Conventional vs. MAPS schedule



Conventional schedule

3 cycles; 2 cycle/iteration



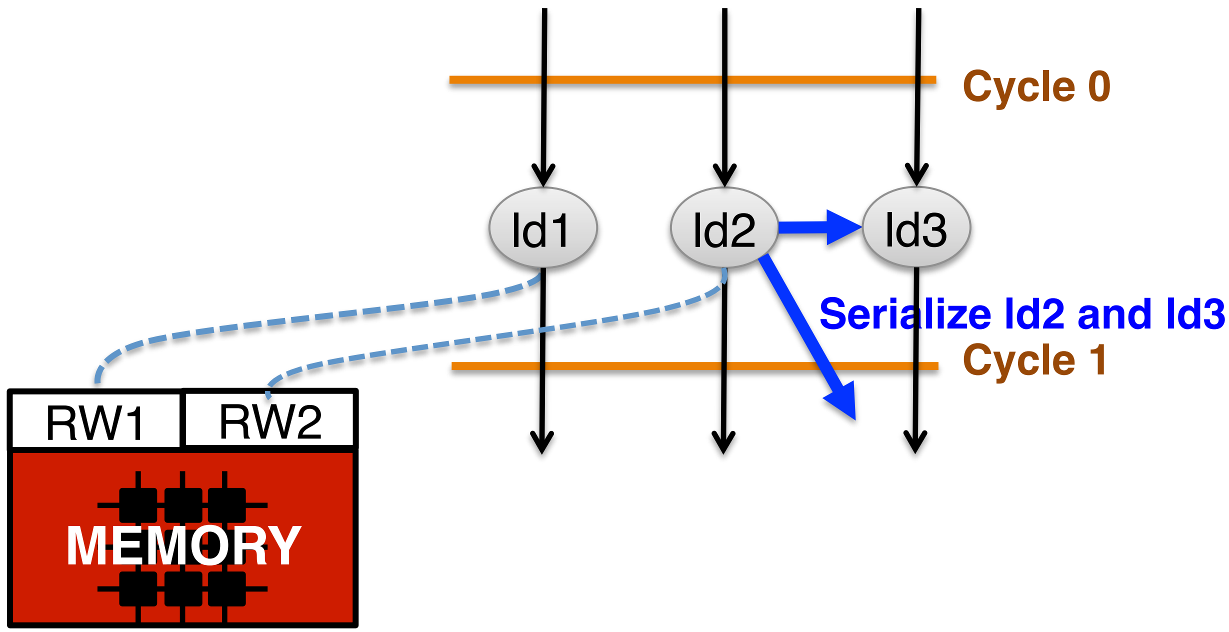
MAPS schedule

Total 2 cycles; 1 cycle/iteration

Incremental Scheduling for Resource Constraints

- Resource constraints for black-box operations
  - e.g. memory port limits, hardened multipliers

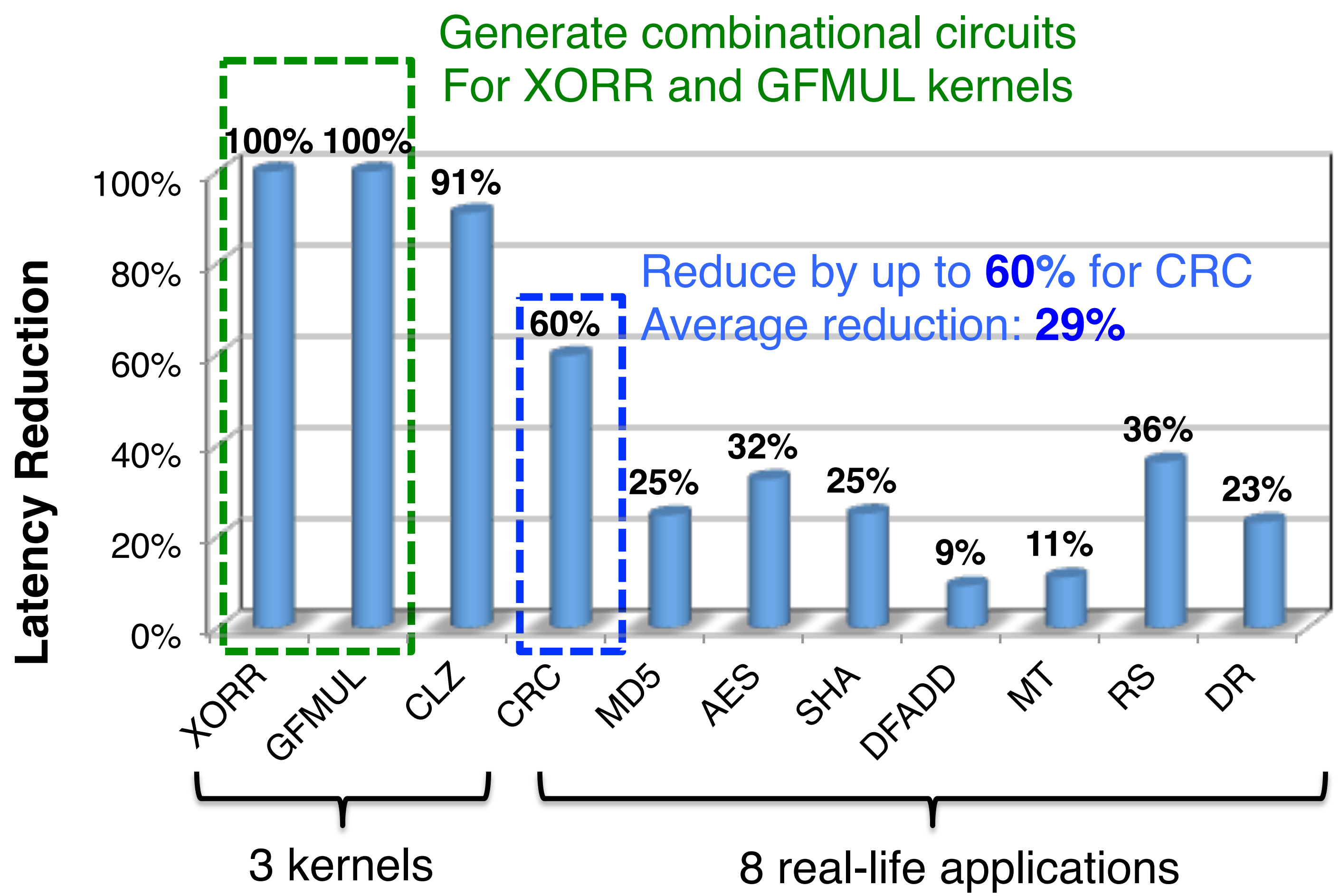
- Incremental scheduling heuristic
  - Legalize the initial solution from the labeling step
  - Gradually serialize resource-constrained operations



Experimental Results

- Setup
  - A state-of-the-art commercial HLS
    - MAPS is implemented an LLVM pass
    - We leverage the commercial HLS as the back end for RTL generation
    - We use the same commercial HLS tool as baseline
  - Target device: Virtex-7 FPGA with 6-LUTs
    - 5ns target clock period
- Benchmarks
  - 3 kernels: XORR, GFMUL, CLZ
  - 8 logic-intensive applications from MiBench and CHStone
    - Communication: CRC, Reed-Solomon decoder (RS)
    - Cryptography: MD5, AES, SHA
    - Scientific Computing: DFADD, Mersenne twister (MT)
    - Machine Learning: Digit recognition (DR)

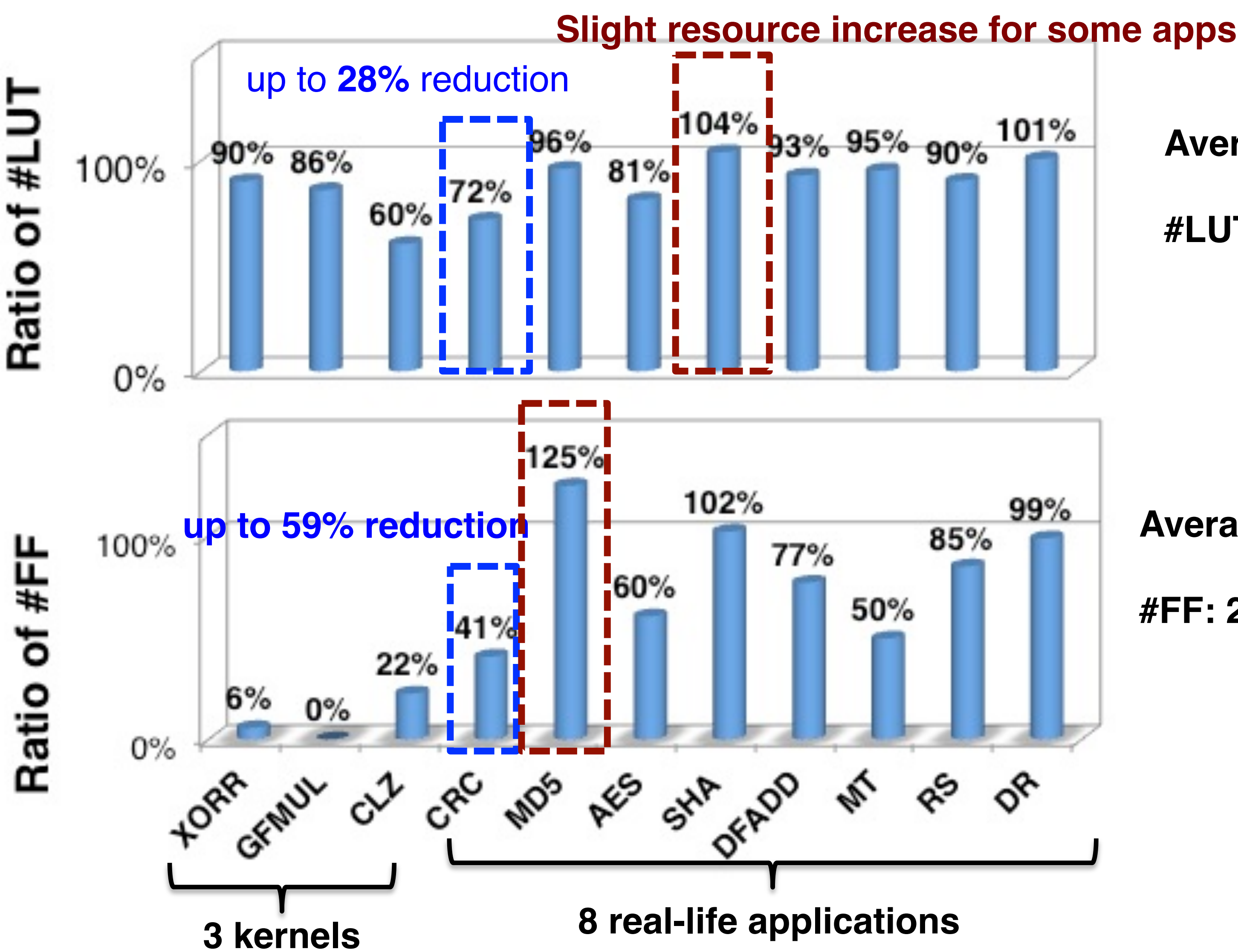
Latency Reduction



5ns target clock period is met for all designs

MAPS significantly reduces latency by enabling more aggressive chaining

Resource Usage Comparison



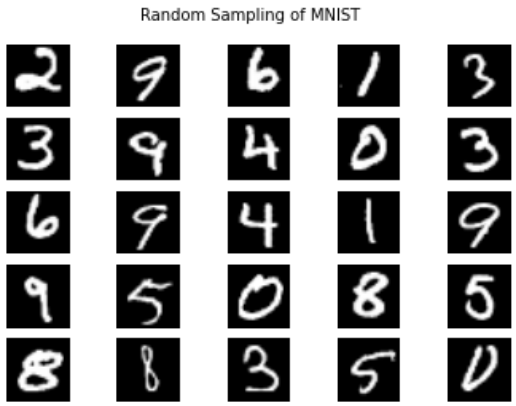
Average reduction

#LUT: 9%

Average reduction

#FF: 25%

Case Study for Digit Recognition (DR)

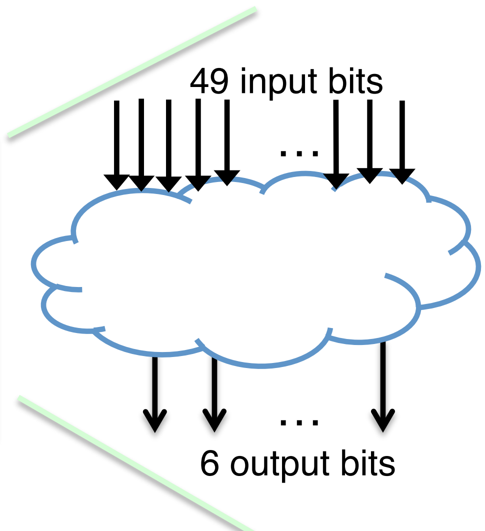



```
0000000
0011100
0110110
0110010
0110110
0011100
0000000
```

(a) Binary string in 2D array

(b) Binary image

```
void count_set_bit
    (bit49 input, bit6 &ones)
{
  for (int i=0; i<49; i++)
    ones += input[i];
}
```



Target clock period = 5ns

Baseline:

7 levels of operations

2 cycles

MAPS:

3 levels of 6-LUTs

1 cycles

23% latency reduction for the entire DR app

Conclusions

- Cross-layer optimizations that integrate different steps of the FPGA flow can enable next leap in QoR improvement for HLS

- MAPS: a mapping-aware constrained scheduling algorithm
  - Elegantly integrate LUT mapping information into scheduling
  - Achieve latency-optimal schedule under SDC constraints
  - Significantly improve performance and reduce hardware resource

THANKS!

QUESTIONS?

Complexity of MAPS Algorithm

- Each iteration will traverse each node and edge once
  - Complexity for a single iteration: $O(|V|^K+|E|)$
- MAPS labeling converges within at most $D*|V|$ iterations
  - Each iteration will monotonically increase the L-value by at least 1
  - The upper bound of each L-value is $D*|V|$, where D denotes the maximum delay for any edge; D is usually a small constant.
- Total complexity of MAPS Labeling is $O(D*|V|*(|V|^K+|E|))$, which is **polynomial** when K and D are small constants

MAPS labeling guarantees to obtain a legal schedule with optimal L-value for each node in pseudo-polynomial time

Runtime Evaluation for MAPS

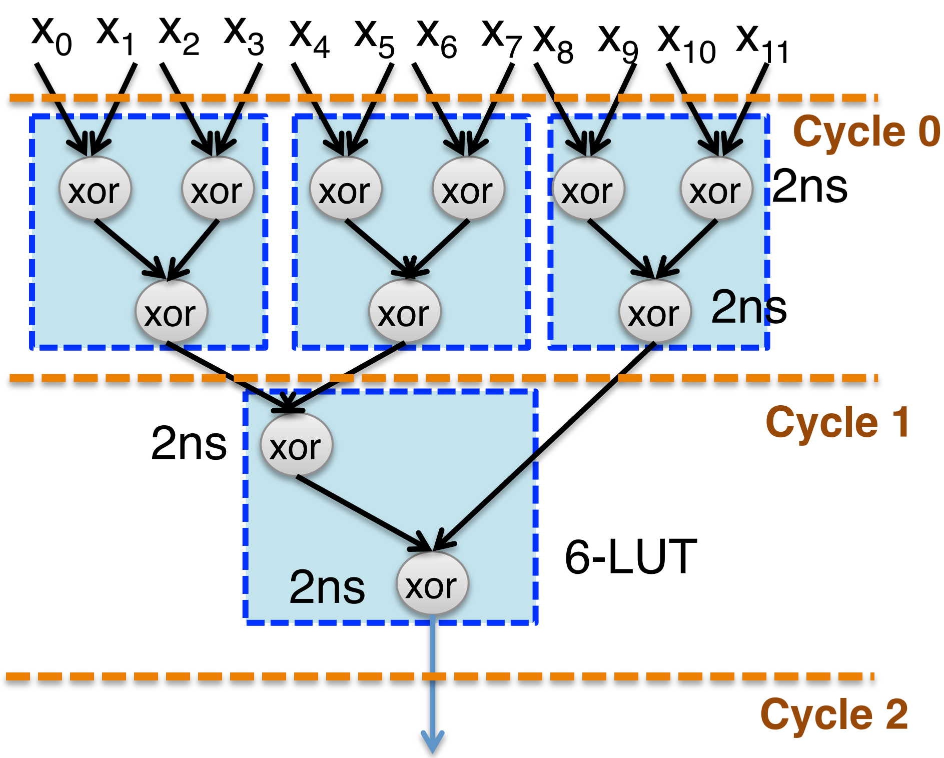
Synthesis time (seconds)

	Baseline	MAPS
PC	23.0	23.0
XORR	56.0	64.7
GFMUL	4.3	11.1
CLZ	24.0	29.7
CRC	3.9	11.8
MD5	15.6	28.8
AES	20.5	61.9
SHA	8.9	19.6
DFADD	9.3	11.1
MT	36.5	193.5
RS	23.0	24.6
DR	44.5	50.5

Design	Approach	Target Clock Period = 5ns			
		CP(ns)	LAT	LUT	FF
XORR	baseline	2.88	1	133	17
	MAPS	2.28	0 (-100%)	120 (-10%)	0 (-100%)
GFMUL	baseline	2.93	2	50	27
	MAPS	1.68	0 (-100%)	43 (-14%)	0 (-100%)
CLZ	baseline	2.93	11	177	169
	MAPS	2.93	1 (-91%)	107 (-40%)	38 (-78%)
CRC	baseline	2.93	161	57	310
	MAPS	2.93	65 (-60%)	41 (-28%)	126 (-59%)
MD5	baseline	4.39	126	9175	6747
	MAPS	4.24	95 (-25%)	8812 (-4%)	8417 (+25%)
AES	baseline	4.78	197	4895	5855
	MAPS	4.44	133 (-32%)	3989 (-19%)	3540 (-40%)
SHA	baseline	4.21	561	2916	3196
	MAPS	3.87	421 (-25%)	3032 (+4%)	3263 (+2%)
DFADD	baseline	4.81	11	5950	2735
	MAPS	4.80	10 (-9%)	5528 (-7%)	2106 (-23%)
MT	baseline	3.96	146	3617	4630
	MAPS	4.03	130 (-11%)	3447 (-5%)	2295 (-50%)
RS	baseline	4.23	124370	1710	974
	MAPS	4.30	79222 (-36%)	1546 (-10%)	828 (-15%)
DR	baseline	3.70	520021	625	432
	MAPS	3.80	400021 (-23%)	630 (+1%)	427 (-1%)
AVERAGE			-29%	-9%	-25%

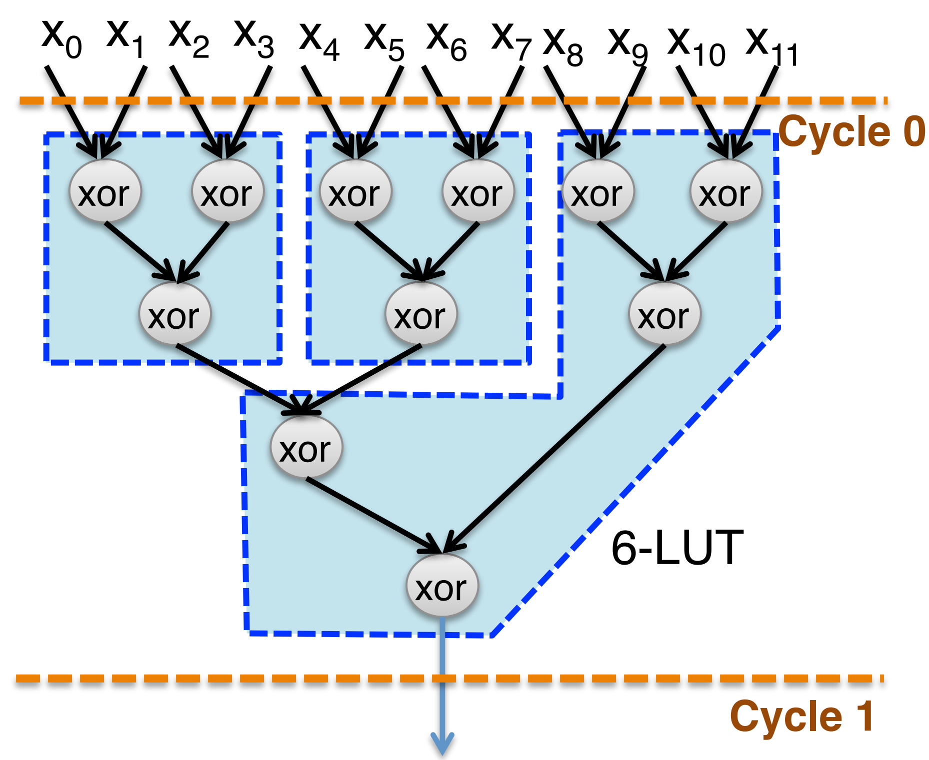
Kernel: Xor Reduction for Bit Vector

Target clock period is **5ns**, each one-bit addition has **2ns** latency



Original Schedule

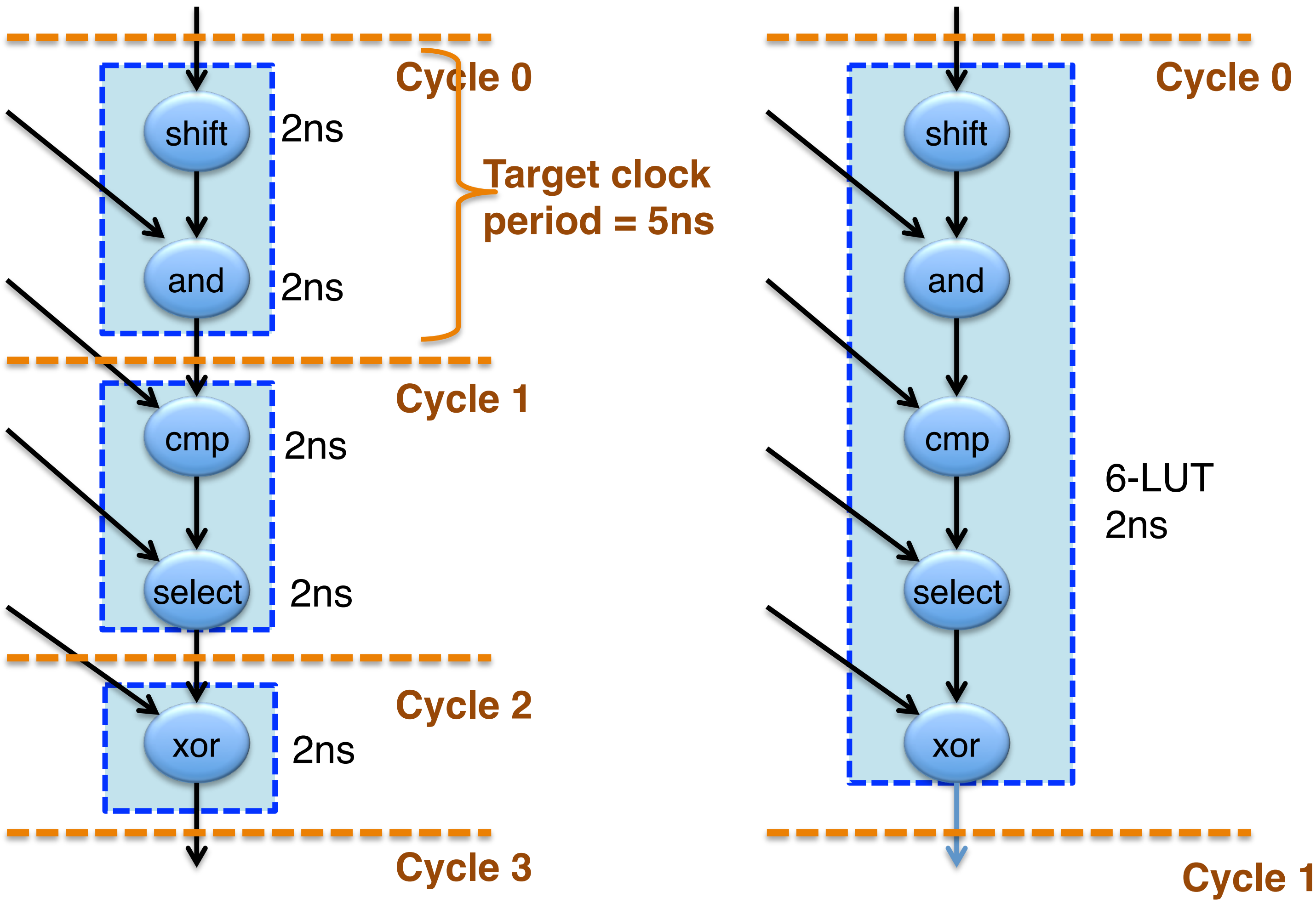
2 cycles, 4 LUTs



MAPS Schedule

1 cycles, 3 LUTs

Kernel Example: Galois Field Multiplication (GFMUL)



Original Schedule

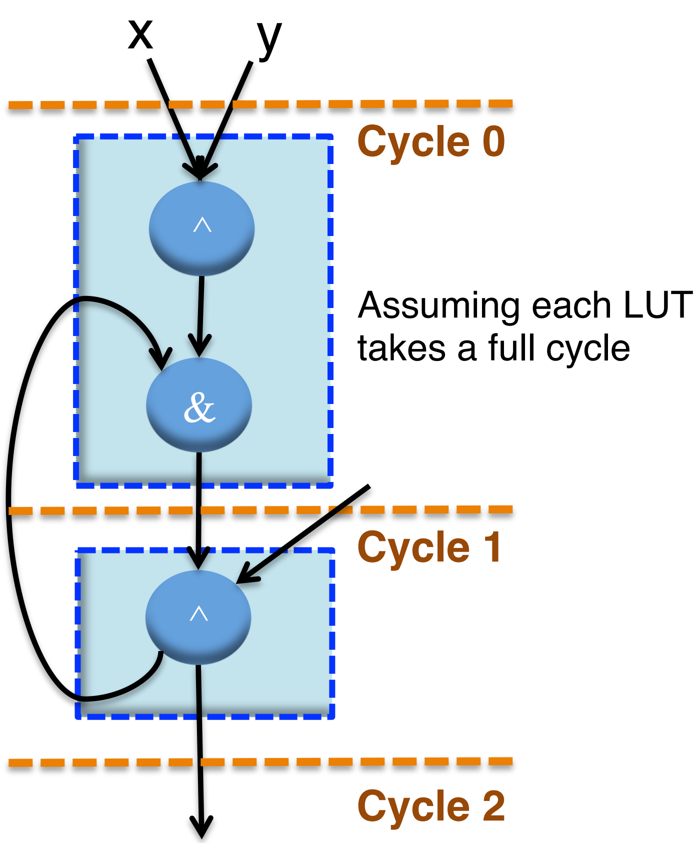
4 cycles, 3 LUTs

MAPS Schedule

1 cycle, 1 LUT

Separate Mapping and Scheduling

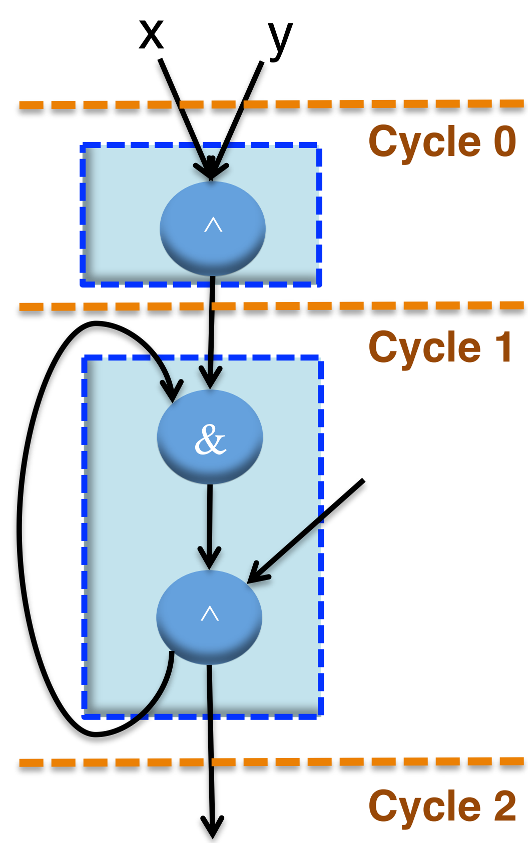
- How about performing mapping before scheduling?



Mapping + Scheduling

2 cycles per iteration

Total Latency = 2*N for N iterations



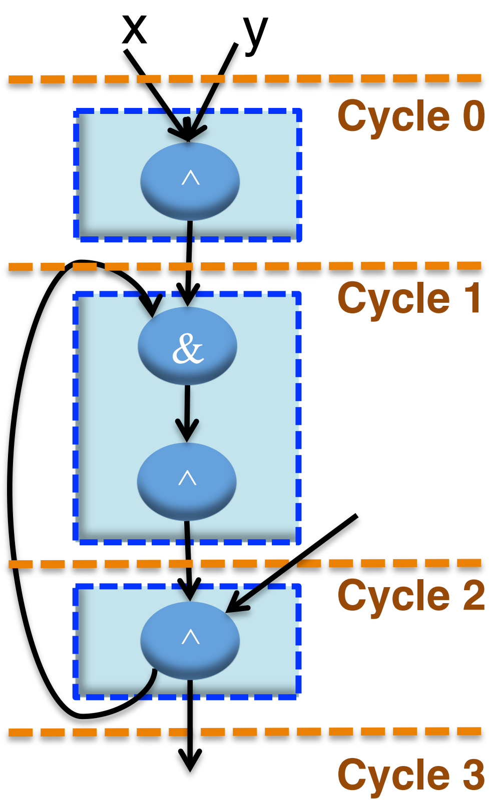
Optimal schedule

1 cycle per iteration

Total Latency = N for N iterations

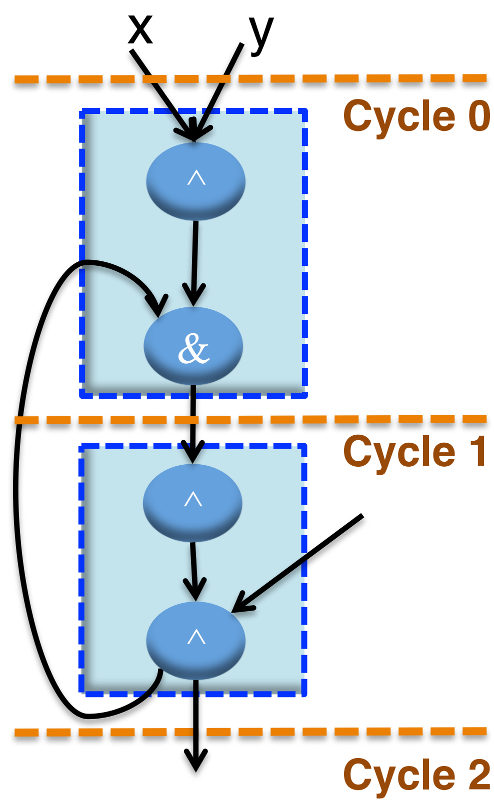
Loop-Prioritized Mapping and Scheduling

- How about prioritizing mapping for loops



Mapping + Scheduling

3 cycles, 3 LUTs

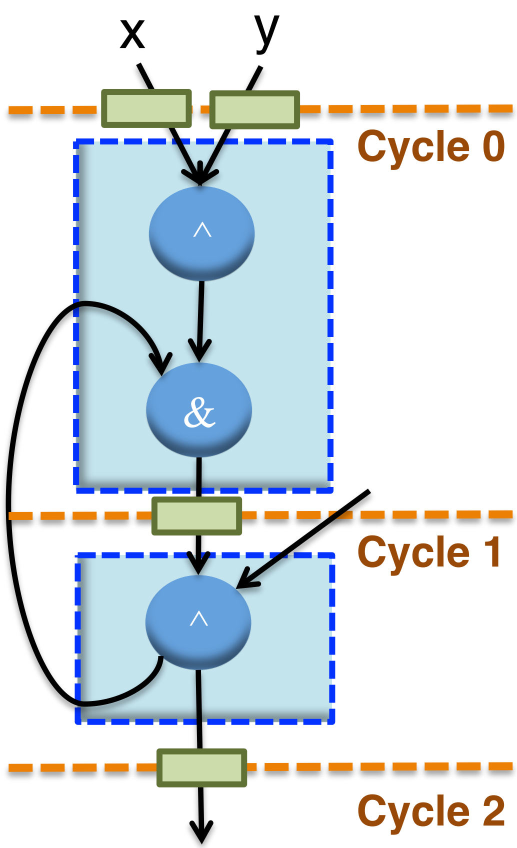


Optimal schedule

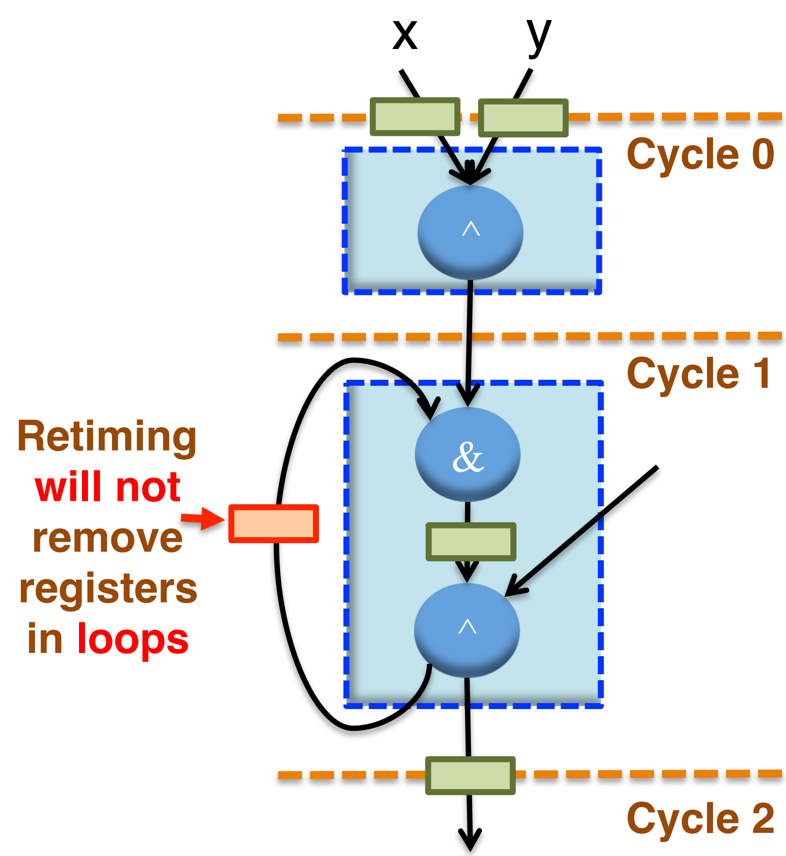
2 cycles, 2 LUTs

Retiming Based Mapping and Scheduling

- Can we address the problem using retiming?



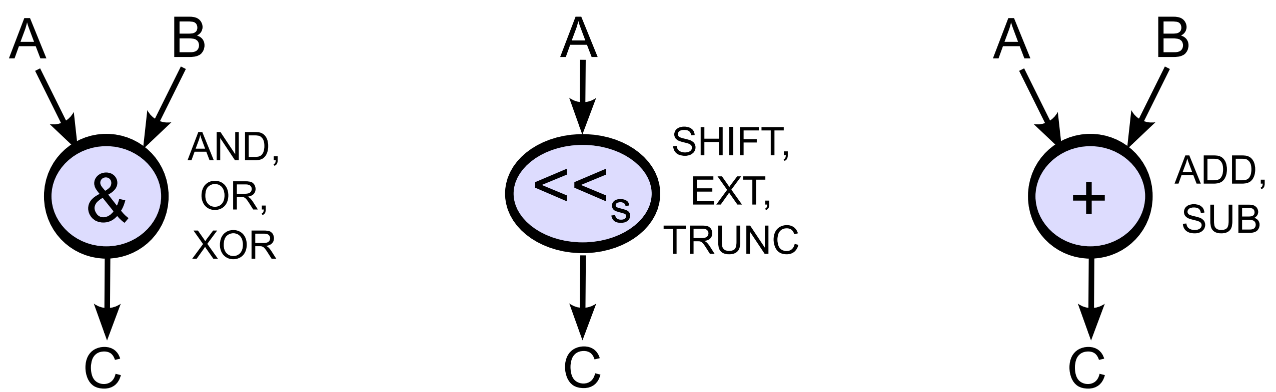
Mapping + Scheduling
2 cycles per iteration
Total Latency = 2*N for N iterations



Mapping + Scheduling + Retiming
Still 2 cycles per iteration

Word-Level Tracking

- Bit-Level Dependence Tracking



inputs(C) = {A, B}

inputs(C)={A_S}

inputs(C)=
{A_0, A_1, ..., B_0, B_1, ...}