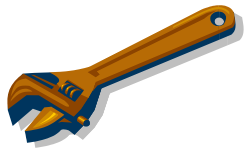
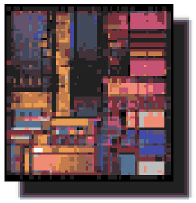


ECE 595Z
Digital VLSI Design Automation

Module 7 (Lectures 24-26): Sequential Logic Optimization

Lecture 24



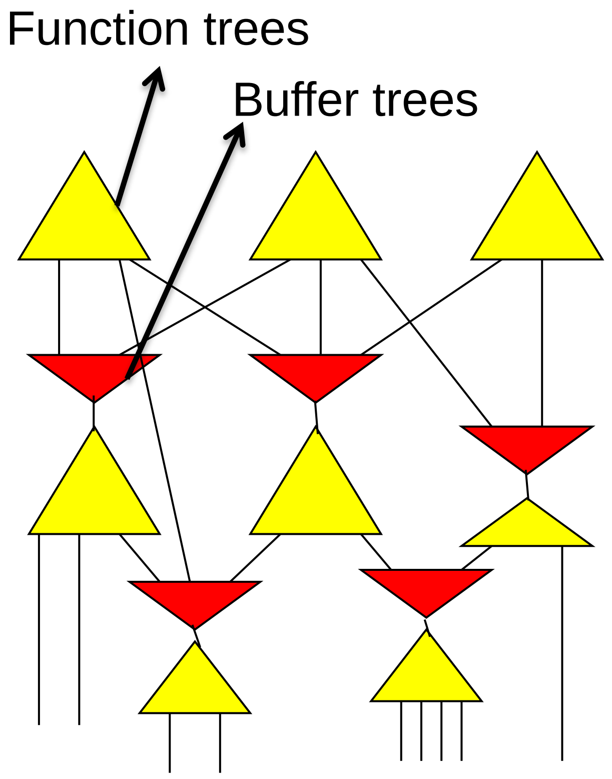
Anand Raghunathan

MSEE 318

raghunathan@purdue.edu

Timing Optimization in Logic Synthesis

- **Circuit re-structuring**
  - Get a good global structure for the circuit
  - Possible to do this during technology-independent optimization or after technology mapping
    - E.g., Balanced tree vs. chain
- Technology mapping for delay
  - We have already seen how tree mapping works
  - Delay budgeting across trees and design hierarchy
- Fanout optimization (implementation of buffer trees)
- Circuit sizing

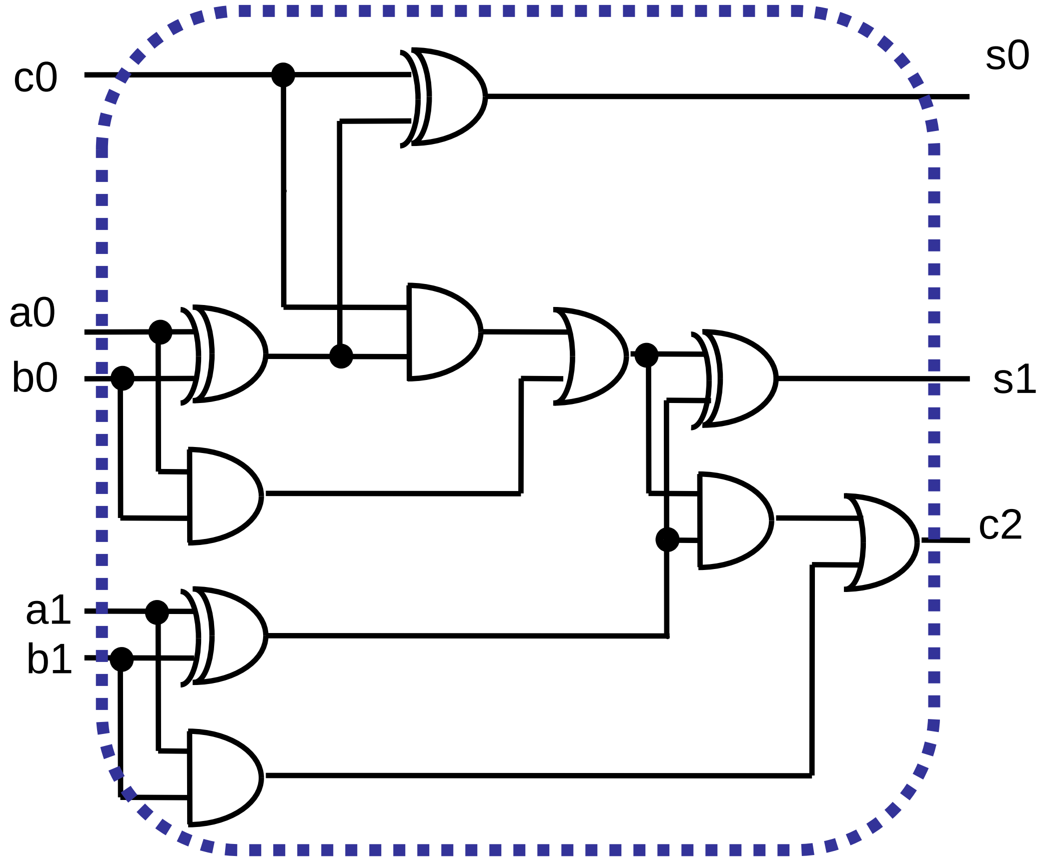


Circuit Re-structuring for Delay

- Techniques used for circuit re-structuring can be viewed as generalizations of fast adder architectures
  - **Carry lookahead → Collapsing and re-structuring / height reduction**
  - Carry select → Generalized select transform
  - Carry bypass → Generalized bypass transform

Ripple Carry → Carry Lookahead Adder

- Recall how you designed a carry lookahead adder?



2-bit Ripple carry adder

Equations for carry directly in terms of Propagate and Generate bits

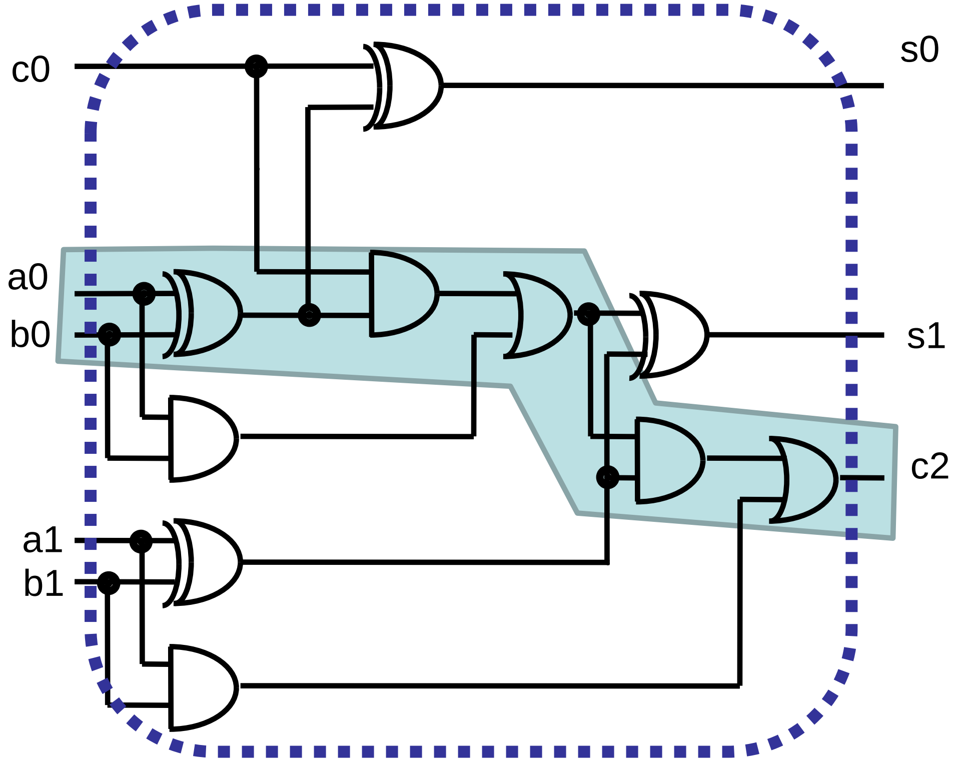
$P0 = a0 \oplus b0$
$G0 = a0b0$

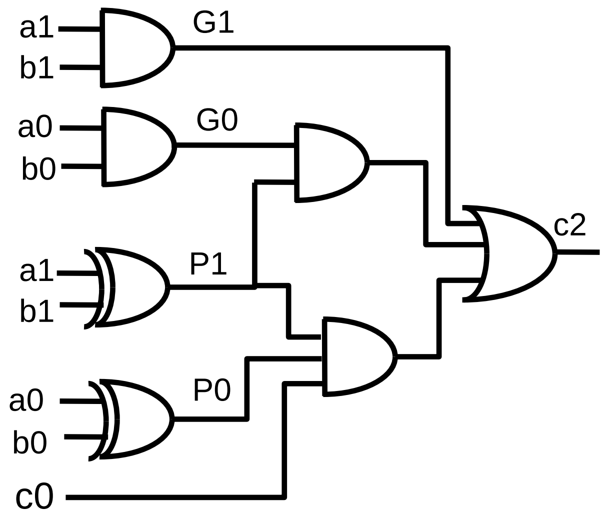
$G1 = a1b1$
$P1 = a1 \oplus b1$

$c1 = G0 + c0P0$
$c2 = G1 + G0P1 + c0P0P1$
$c3 = G2 + G1P2 + G0P1P2 + c0P0P1P2$
…

Ripple Carry → Carry Lookahead Adder

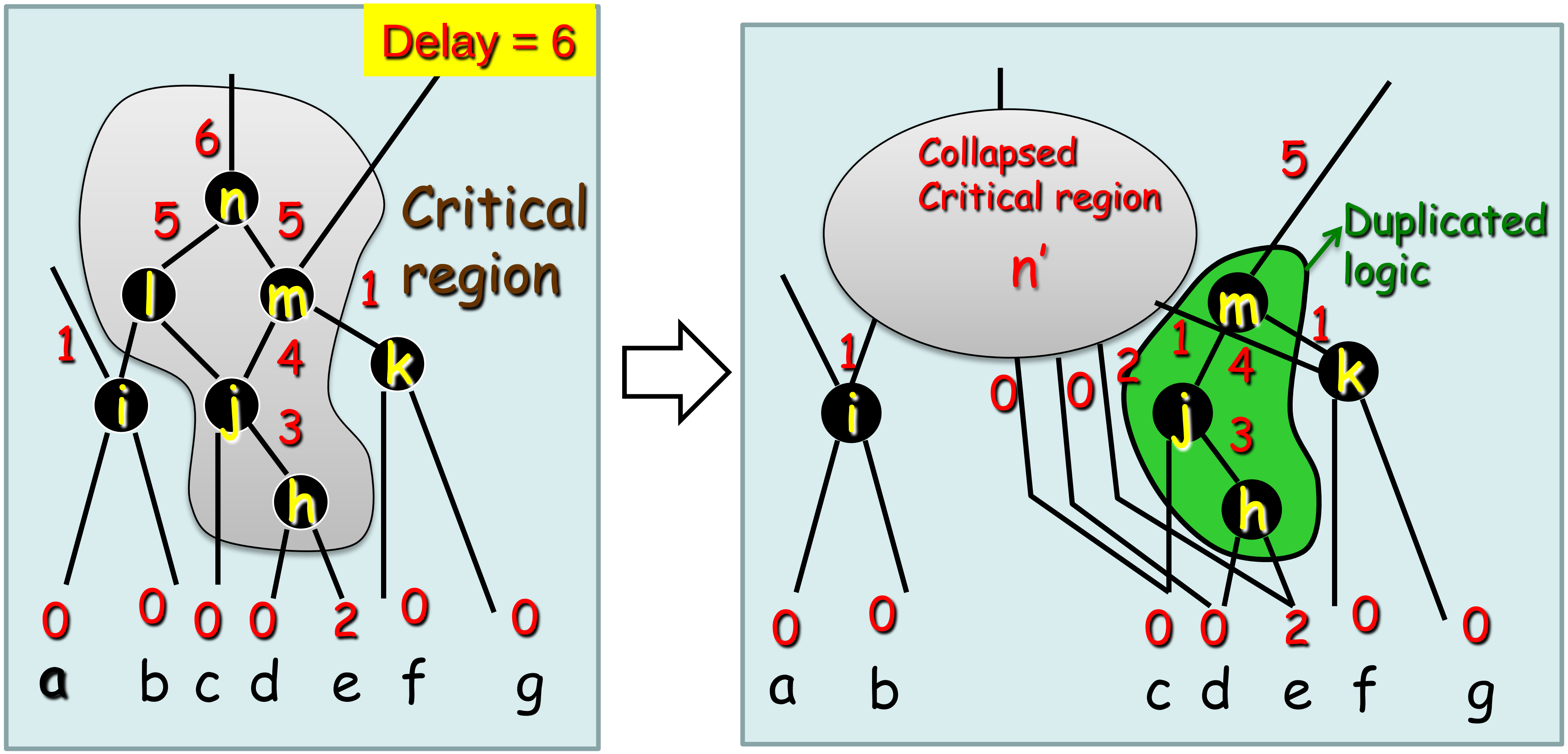
- What are we really doing here?
  - Collapsing the critical path and re-structuring it!





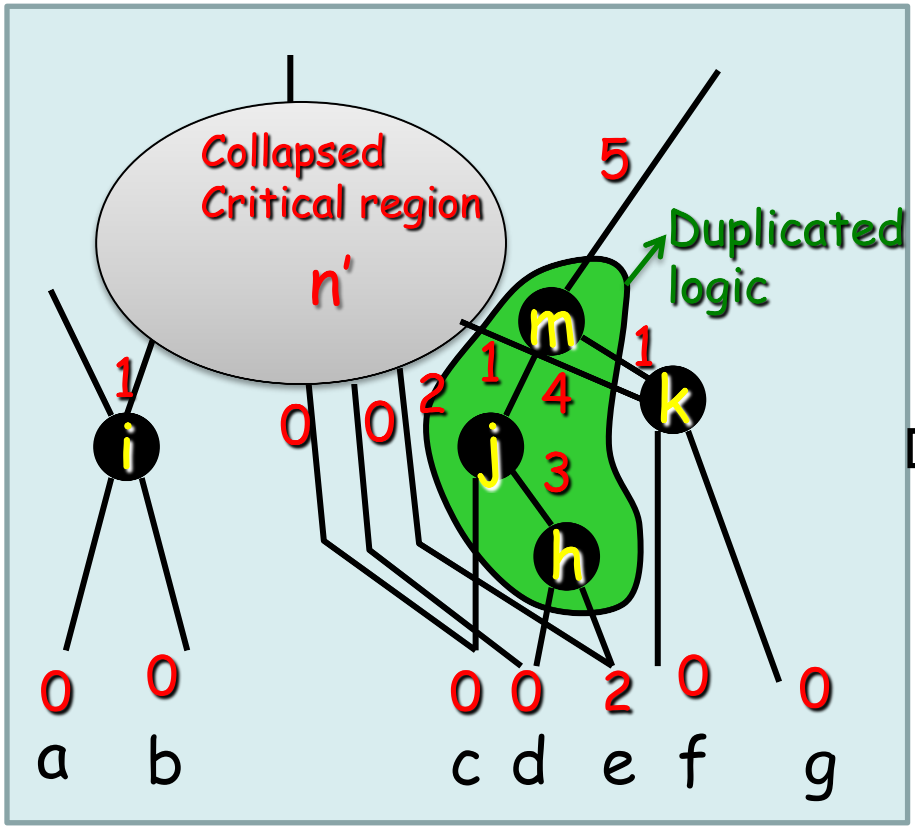
Re-structuring for Delay: Height Reduction

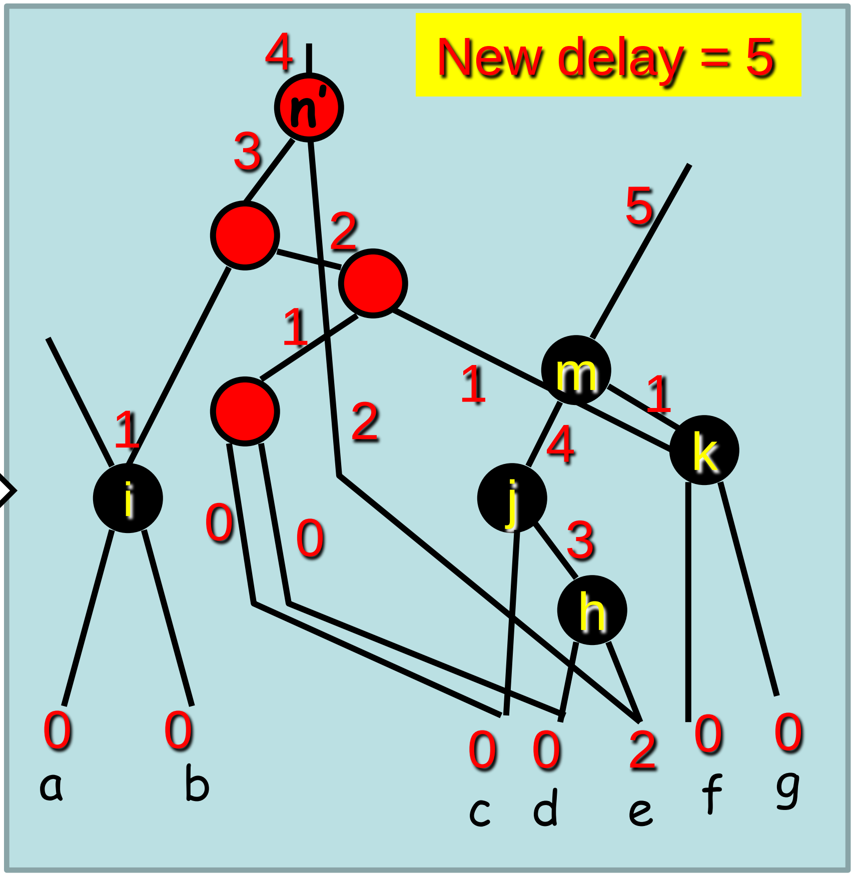
- The same concept can be applied to arbitrary Boolean networks!



Re-structuring for Delay: Height Reduction

- **Key idea**: Collapsing a critical region and re-structuring based on its input/output constraints





Collapsing and re-structuring methodology

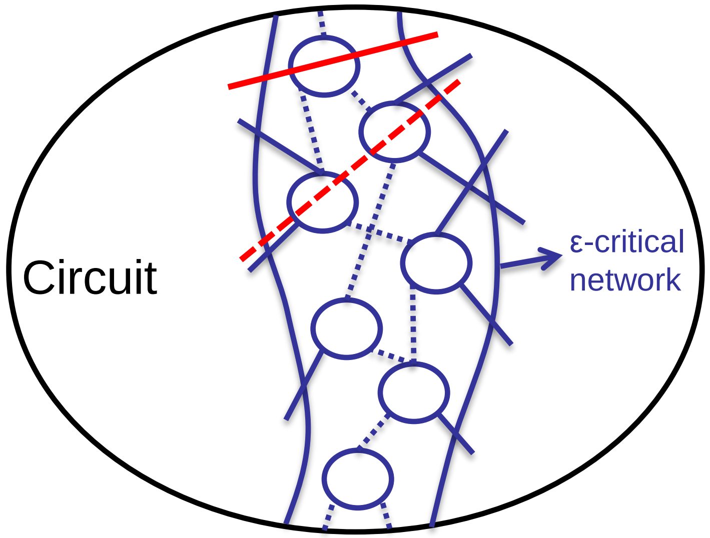
```
while(circuit timing improves) {
        select logic region to transform;
        collapse selected region;
        re-synthesize for better timing;
}
```

- Questions
  - Which regions to re-structure?
  - How to re-synthesize?

K. J. Singh, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, “Timing optimization of combinational logic”, Proc. Int. Conf. on Computer-Aided Design, pp. 282-285,1988.
H. J. Touati, H. Savoj, R. K. Brayton, “Delay Optimization of Combinational Logic Circuits By Clustering and Partial Collapsing,” Proc. Int. Conf. on Computer-Aided Design, pp. 188-191, 1991

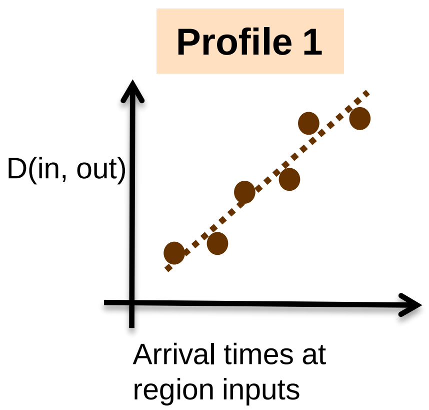
Identifying Regions to Re-structure

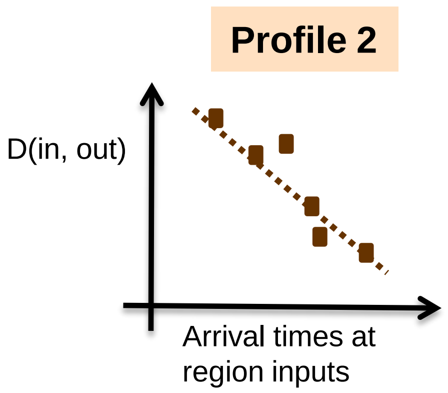
- Start by identifying the ε-critical network
- **Key idea**: **Cut-set** (or separator-set) of a graph
  - Set of vertices which, if removed, partition the graph into disjoint sub-graphs
- Find a minimum-cost cut-set of the ε-critical network and collapse nodes in the cut-set
  - Cost computed based on
    - Potential for delay improvement
    - Estimate of logic duplication
- Min-cut / Max-flow algorithm can be used to identify minimum cost cut-set

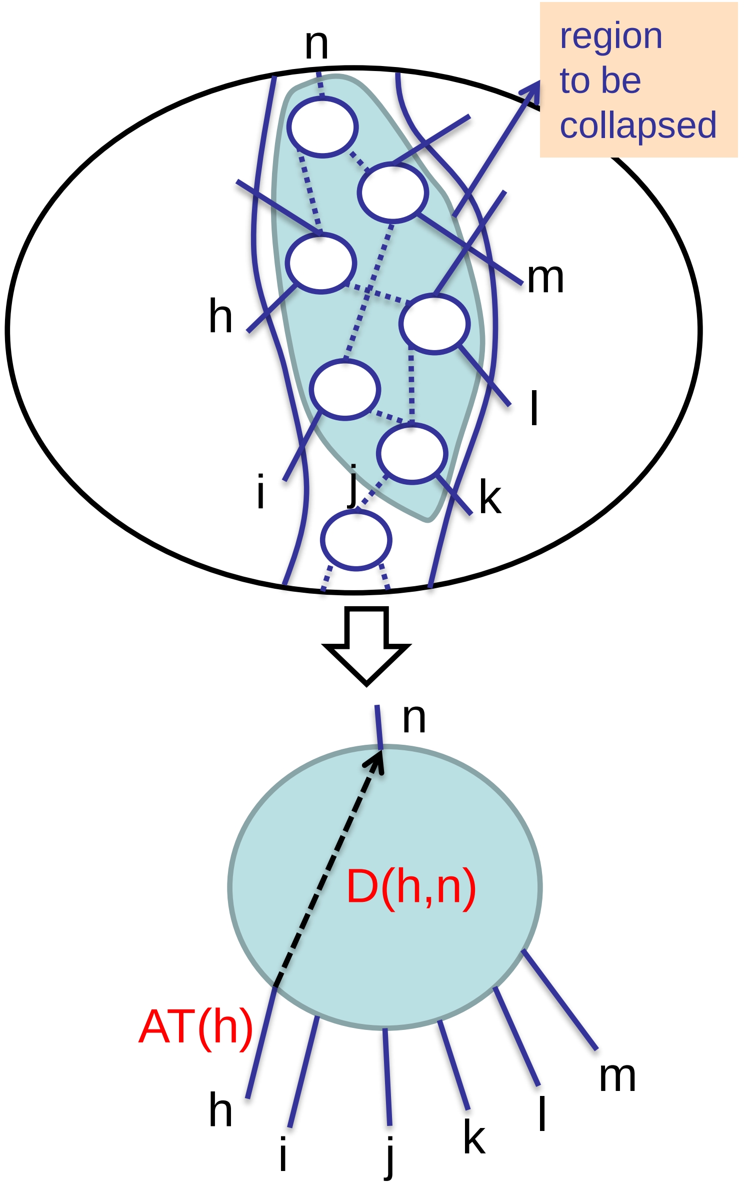


Identifying Regions to Re-structure

- How can we tell whether there is potential for delay optimization at a node?
  - Look at the delay profile of the candidate region that is to be collapsed
    - Arrival time at input vs. delay through from the input to the region output





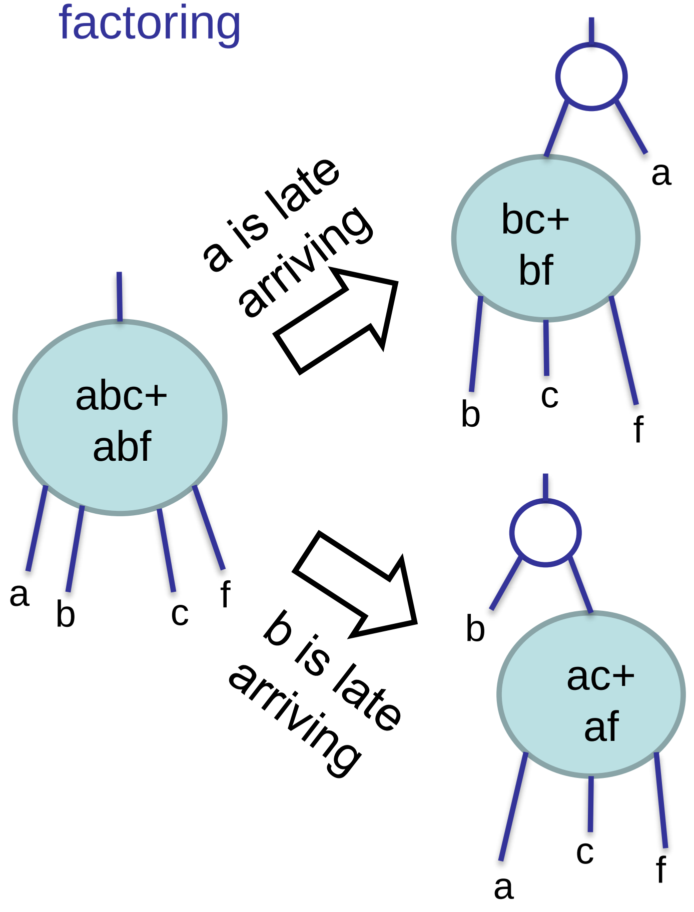


Question: Which of the above profiles offers higher potential for delay optimization?

Timing-driven Re-structuring

- Collapse the nodes in the cut-set with their transitive fanins (up to some logic depth)
- Re-synthesize collapsed region
  - Factoring
  - Decomposition
- **Key idea**: Later arriving inputs are used closer to the outputs, and hence experience lower delay through the region

Example of timing-driven factoring



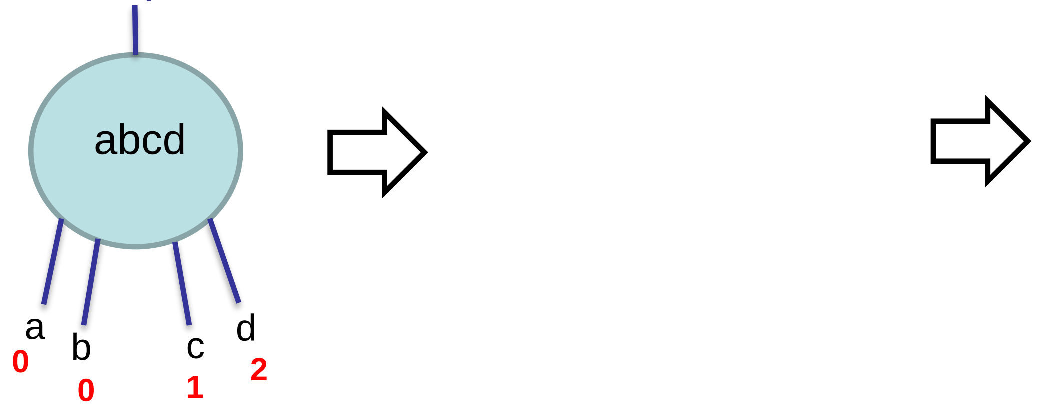
Timing-driven Re-structuring

- Timing-driven de-composition
  - Efficient (greedy) algorithm to do this for AND, OR trees

```
timing_driven_decomp(f) {
    while(|inputs(f)| > 2) {
        select two earliest arriving inputs $l_i, l_j$
        create a gate g with inputs $l_i, l_j$
        substitute g into f
        compute delay at g
    }
}
```

Note: This algorithm is provably optimal!

Example

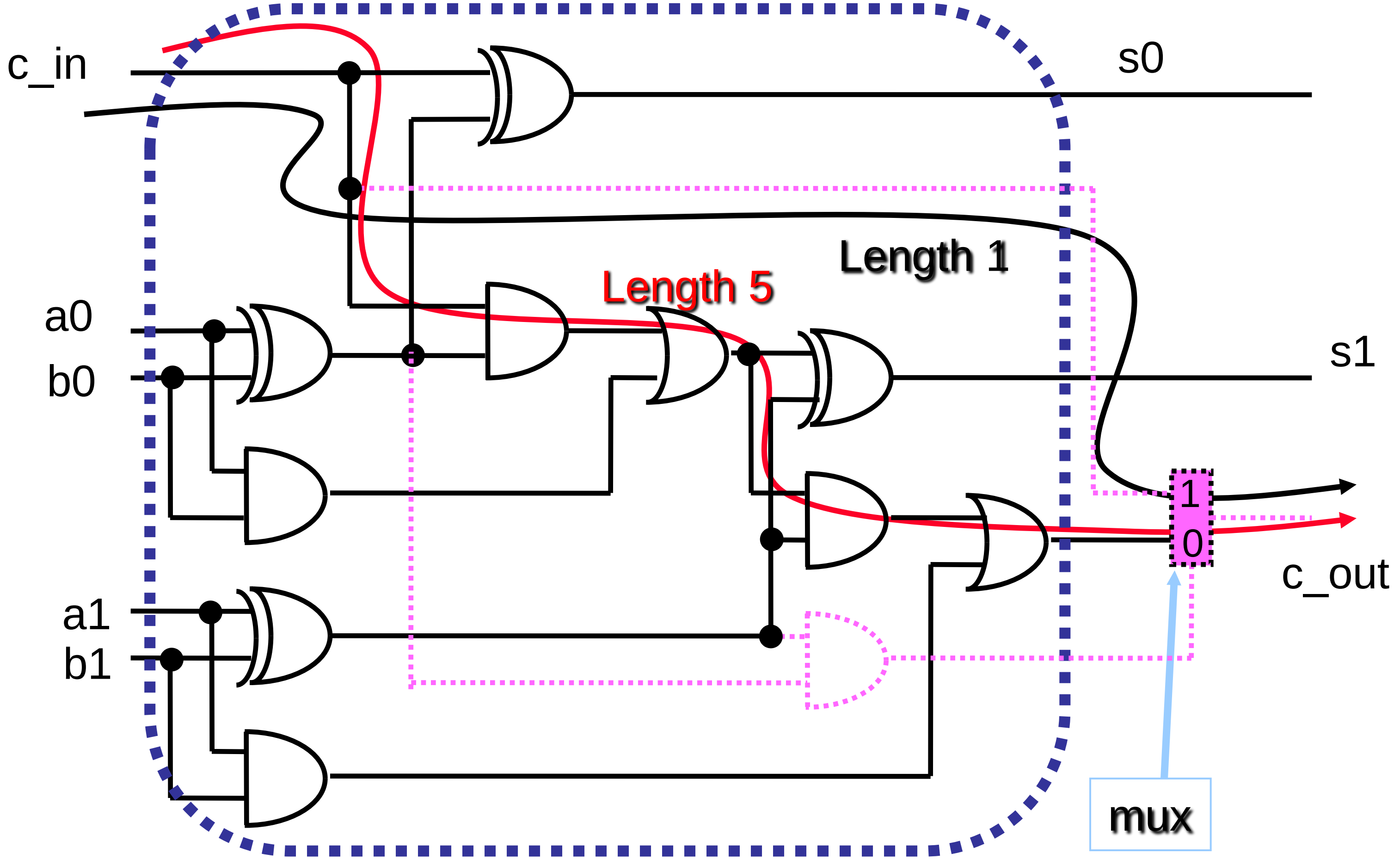


Circuit Re-structuring for Delay

- Techniques used for circuit re-structuring can be viewed as generalizations of fast adder architectures
  - Carry lookahead ⟶ Collapsing and re-structuring / height reduction
  - **Carry bypass → Generalized bypass transform**
  - Carry select → Generalized select transform

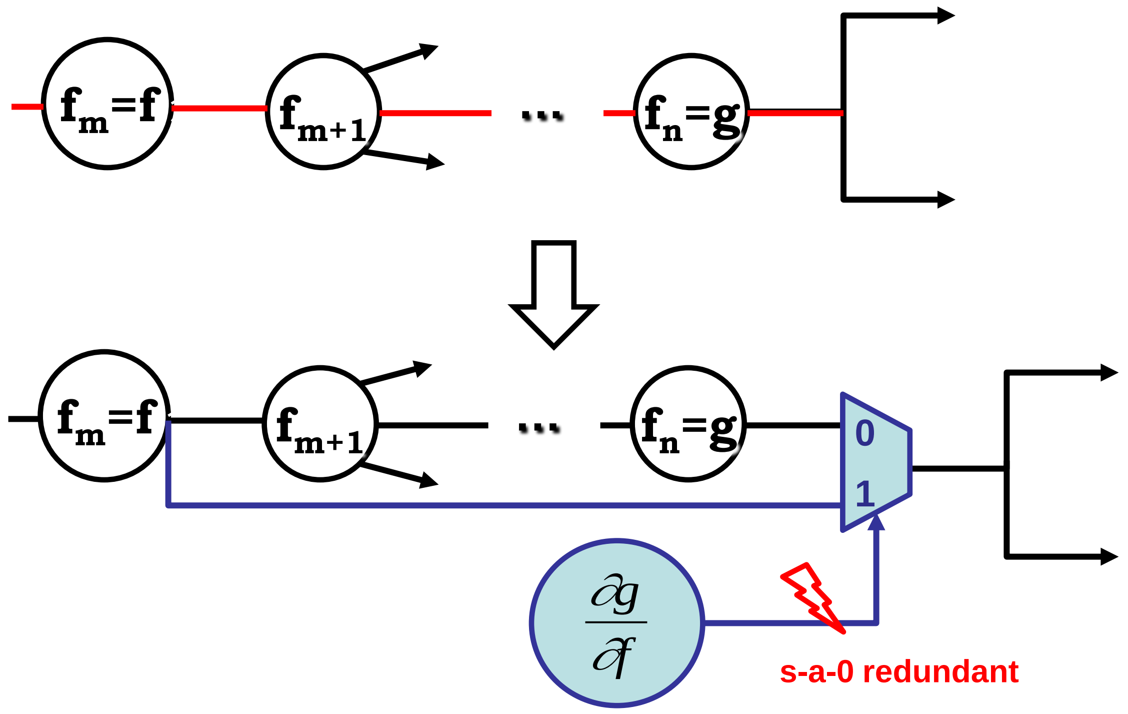
Ripple Carry → Carry Bypass Adder

- How can we generalize this?



Generalized Bypass Transform (GBX)

- **Key Idea**: Make the longest path false
  - Through the introduction of a "bypass" path



Note: Assumes path from f to g is non-inverting

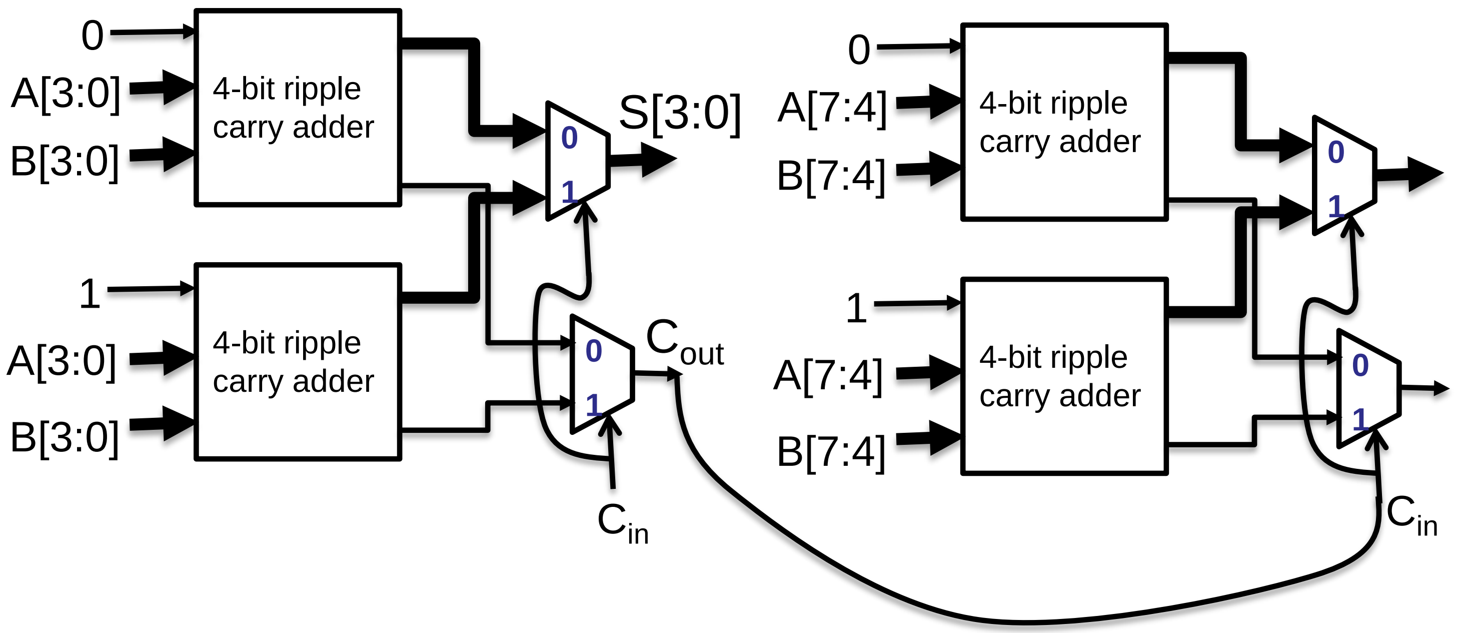
Patrick C. McGeer, Robert K. Brayton, Alberto L. Sangiovanni-Vincentelli, Sartaj Sahni, "Performance Enhancement through the Generalized Bypass Transform," Int. Conf. on Computer-Aided Design, 1991, pp. 184-187.

Circuit Re-structuring for Delay

- Techniques used for circuit re-structuring can be viewed as generalizations of fast adder architectures
  - Carry lookahead $\longrightarrow$ Collapsing and re-structuring / height reduction
  - Carry bypass $\rightarrow$ Generalized bypass transform
  - **Carry select $\rightarrow$ Generalized select transform**

Carry Select Adder

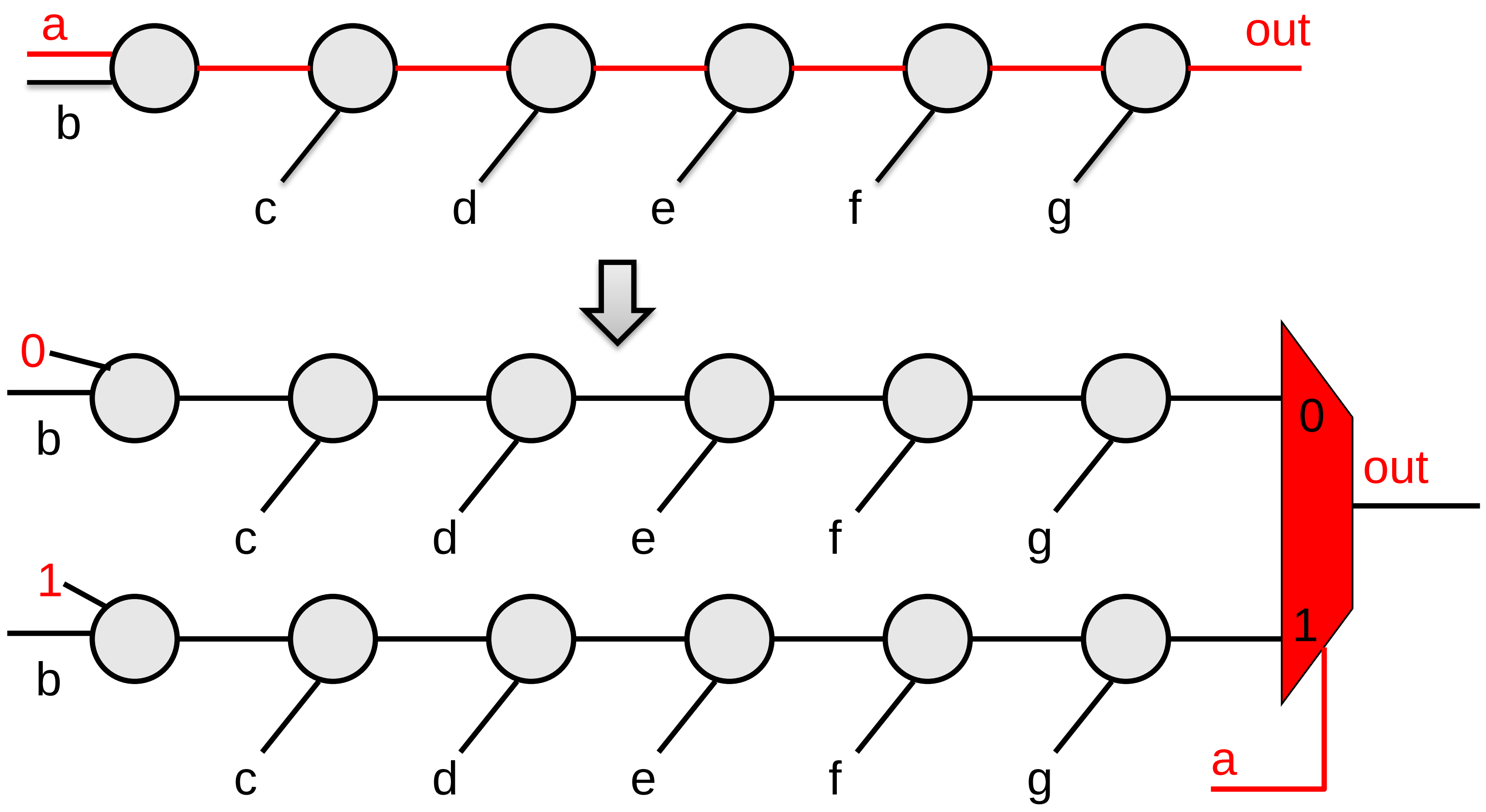
- Compute output assuming carry = 0 and carry = 1, and select correct value once carry is available
  - Benefit: Carry-in sees only MUX delay



How do we generalize this?

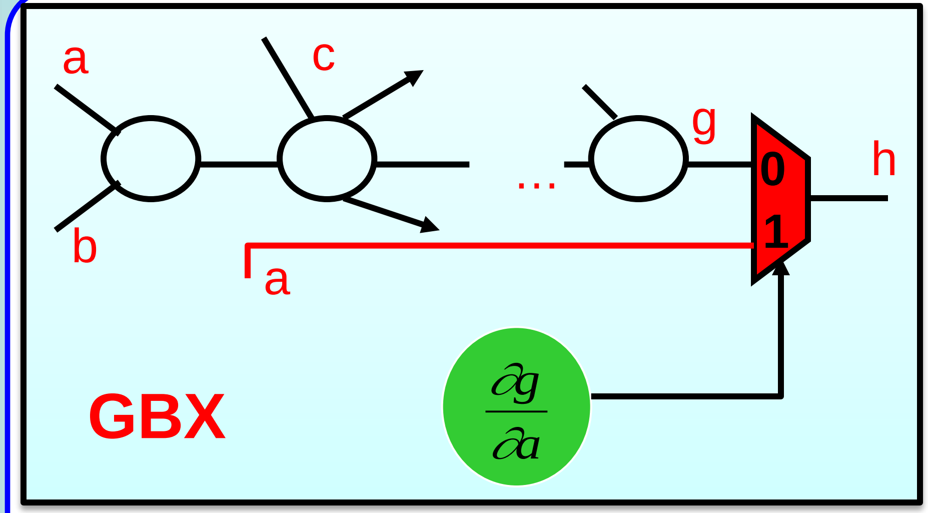
Generalized Select Transform (GST)

- **Key idea**: Co-factor w.r.t. late arriving signal

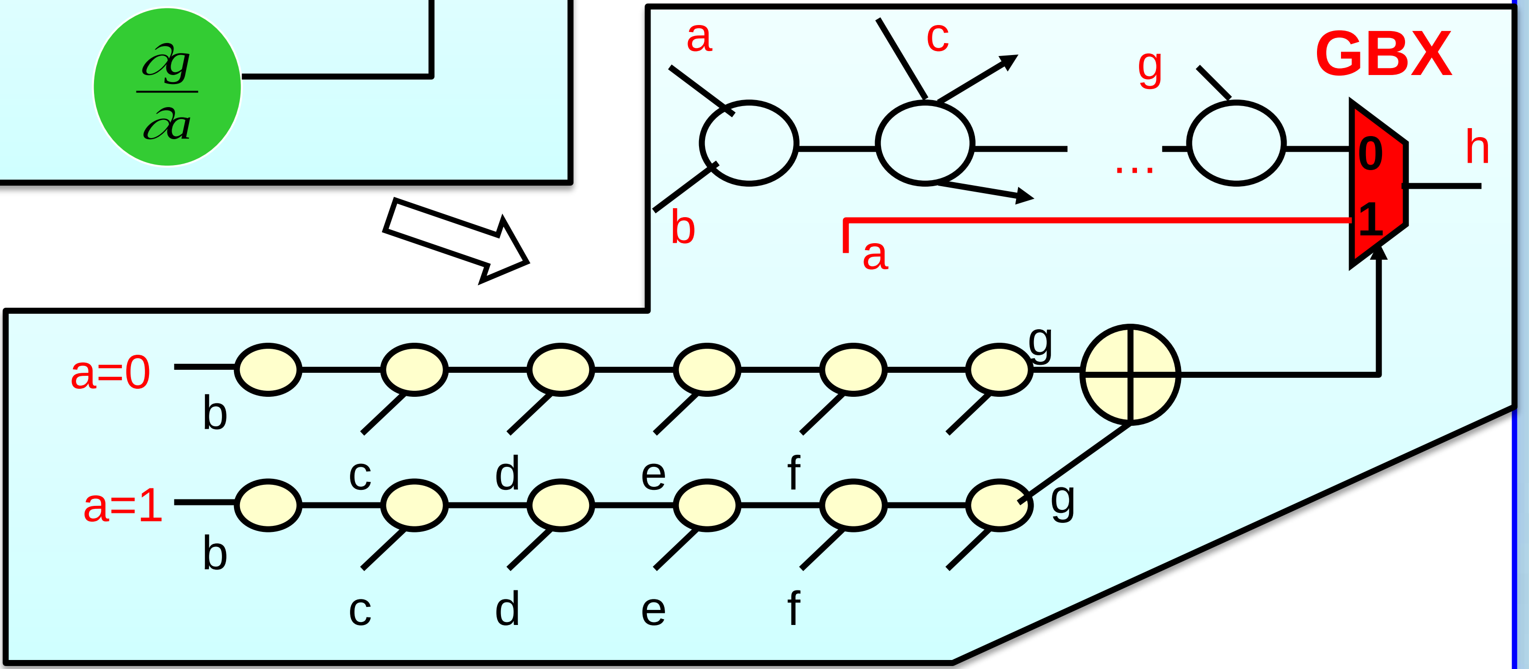


C.L. Berman, D.J. Hathaway, A.S. LaPaugh, L.H. Trevillyan, “Efficient techniques for timing correction,” IEEE International Symposium on Circuits and Systems, 1990, pp. 415-419 .

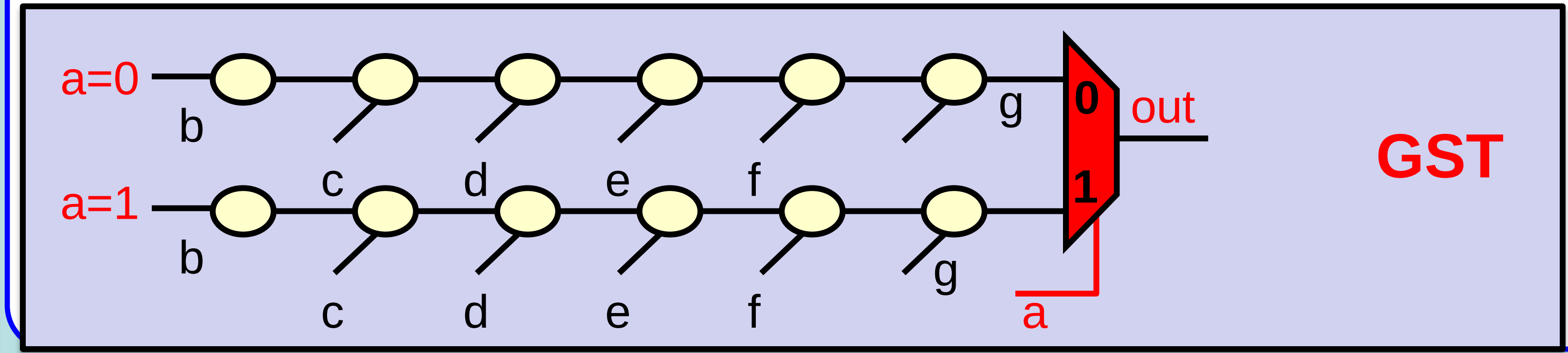
GBX vs. GST



a
b
c
g
...
0
1
h
a
GBX
∂g/∂a



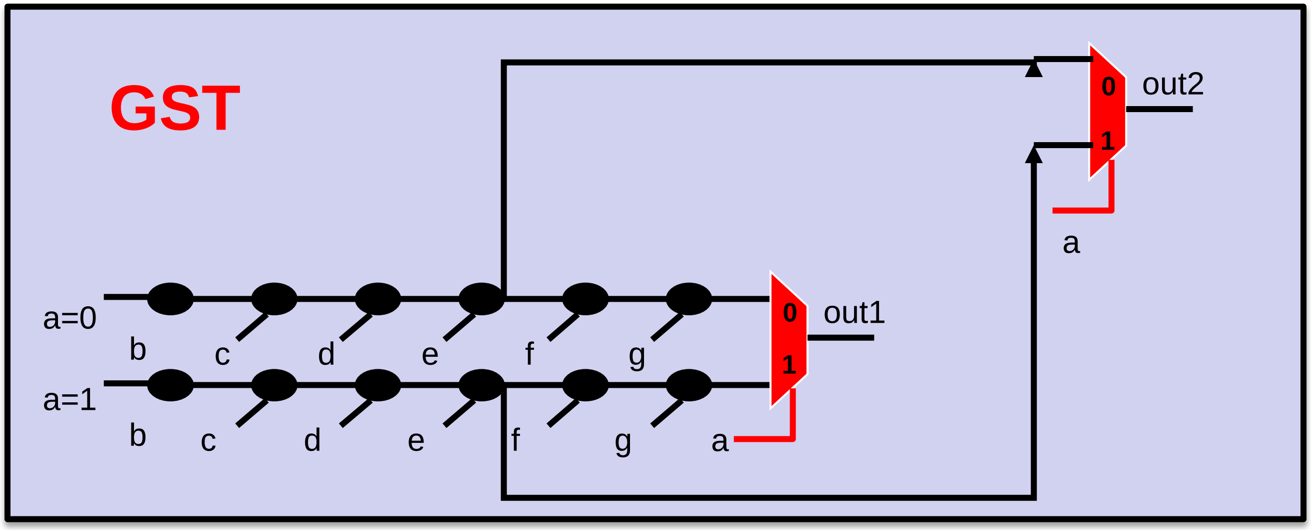
a
b
c
g
...
0
1
h
a
GBX
a=0
b
g
a=1
b
c
d
e
f
g
c
d
e
f



a=0
b
g
0
out
1
a=1
b
c
d
e
f
g
c
d
e
f
a
GST

GBX vs. GST

- Select transform appears to be more area efficient, but..
  - Boolean difference often more efficiently formed in practice
- Both transforms have similar impact on delay
- Can reuse parts of the duplicated logic for multiple fanouts in GST
  - Need one MUX per fanout



Summary: Timing Optimization

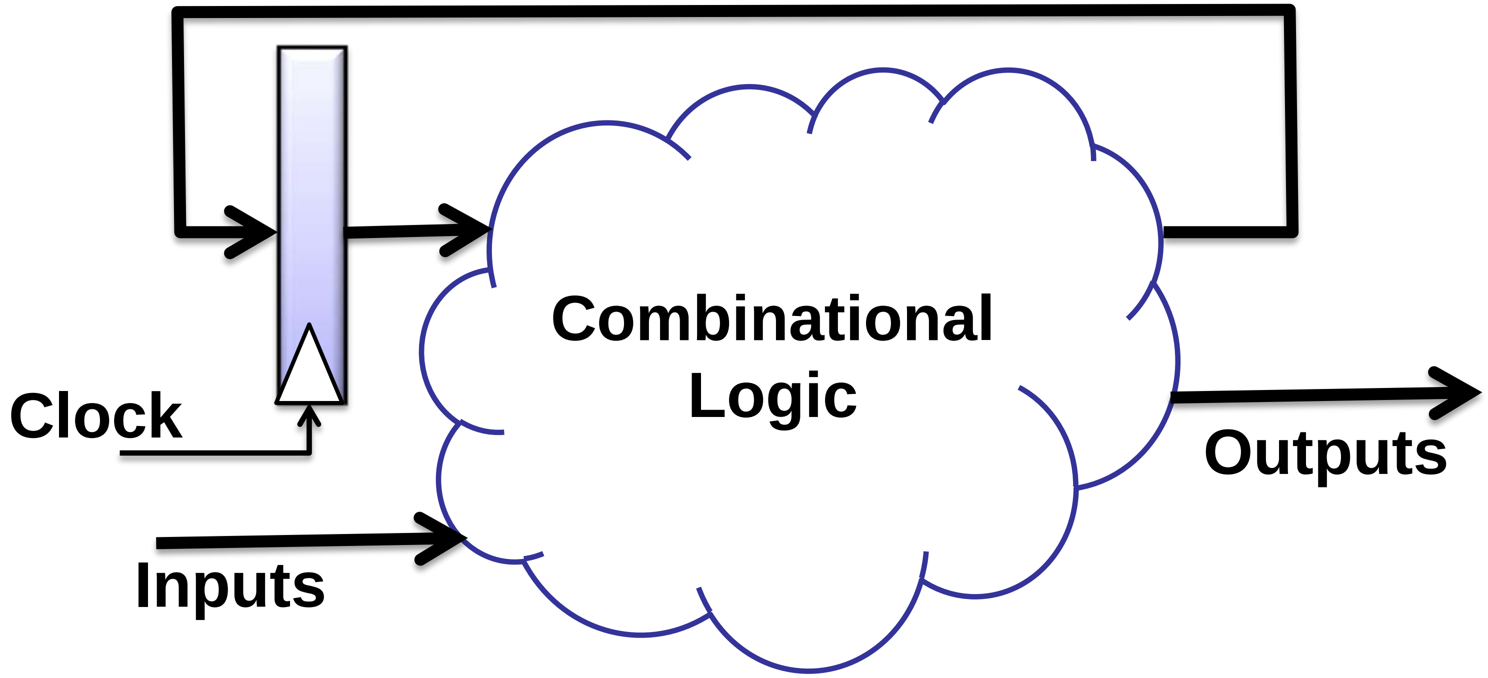
- Variety of methods for delay optimization
- No single technique dominates
- When applied to ripple-carry adder get
  - Carry-lookahead adder (height reduction)
  - Carry-bypass adder (Generalized Bypass Transform)
  - Carry-select adder (Generalized Select Transform)

Suggested Reading

- De Micheli, Chapter 8.6
- Selected publications (available on blackboard)
  - H. Chen, and N. Du, “Path sensitization in critical path problem,” IEEE Transactions on Computer-Aided Design, vol. 12, no. 2, pp. 196 – 207, Feb. 1993.
  - S. Devadas, K. Keutzer, and S. Malik, “Delay Computation in Combinational Logic Circuits: Theory and Algorithms”, Proc. Int. Conf. on Computer-Aided Design, pp. 176-179, 1991.
  - Patrick C. McGeer, Alexander Saldanha, Paul R. Stephan, Robert K. Brayton, Alberto L. Sangiovanni-Vincentelli: Timing Analysis and Delay-Fault Test Generation using Path-Recursive Functions. ICCAD 1991
  - Patrick C. McGeer , Robert K. Brayton, Integrating Functional and Temporal Domains in Logic Design: The False Path Problem and Its Implications, Kluwer Academic Publishers, Norwell, MA, 1991
  - K. J. Singh, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, “Timing optimization of combinational logic”, Proc. Int. Conf. on Computer-Aided Design, pp. 282-285,1988.
  - H. J. Touati, H. Savoj, R. K. Brayton, “Delay Optimization of Combinational Logic Circuits By Clustering and Partial Collapsing,” Proc. Int. Conf. on Computer-Aided Design, pp. 188-191, 1991
  - K. J. Singh, “Performance optimization of digital circuits”, Ph.D. thesis, U. C. Berkeley, 1991.

Why?

- Virtually all digital circuits encountered in practice are sequential
- Techniques that we have learnt thus far only optimize the combinational logic
  - How do we go beyond the clock boundary?

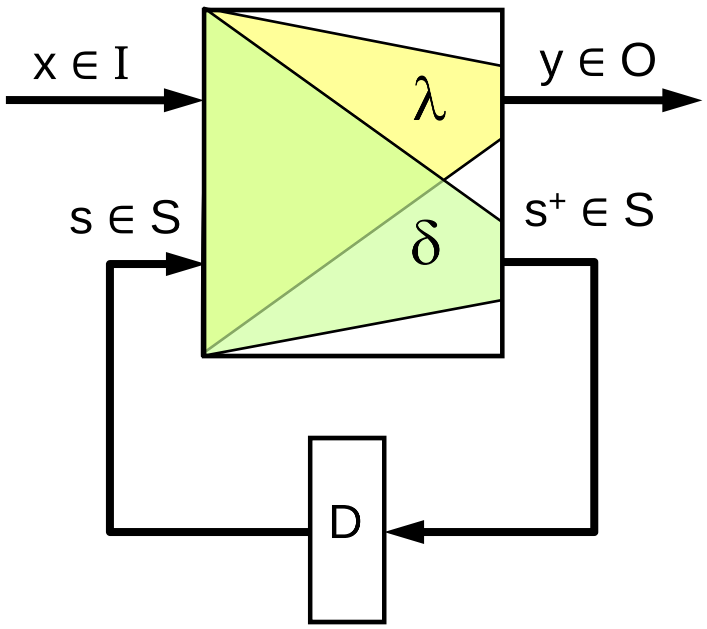


Functional Specification of Sequential Circuits

- For combinational circuits, we used
  - Truth table
  - SOP / POS
  - Boolean network
  - …
- For sequential circuits
  - **Finite State Machines (FSMs)**
    - Mealy vs. Moore
    - Completely vs. Incompletely Specified
  - Finite Automata
    - Abstraction of computing systems
    - Similar to FSMs, except that they just accept input sequences
    - Deterministic vs. Non-deterministic

Finite State Machines (FSMs)

- Formally defined as a 5-tuple (I,O,S,δ, λ)
  - I: Set of input labels
  - O: Set of output labels
  - S: Set of states
  - δ: Transition relation
    - Determines next state given input and present state
    - Mapping: I X S → S
  - λ: Output relation
    - Determines output given input and present state
    - Mapping: I X S → O (Mealy machine), S → O (Moore machine)



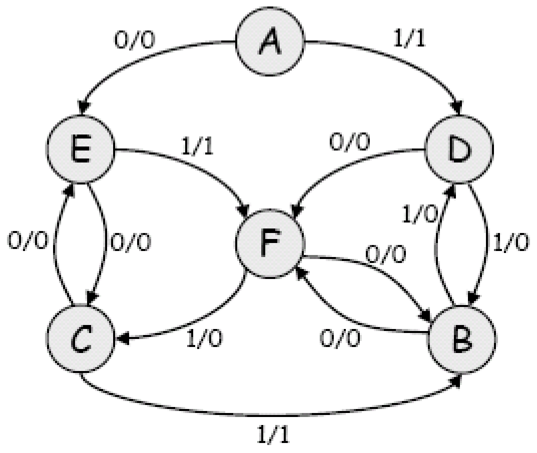
Delay element D:

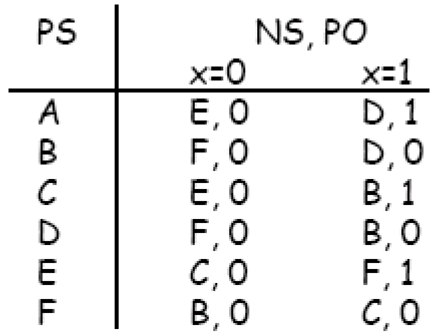
- Clocked: synchronous
- Unclocked: asynchronous

Can also specify an initial state $s_0 \in S$

Representing FSMs

- State Transition Graph (STG)
  - Directed graph (vertices = states, edges = state transitions)
- State Transition Table
  - Tabular representation of next-state and output functions



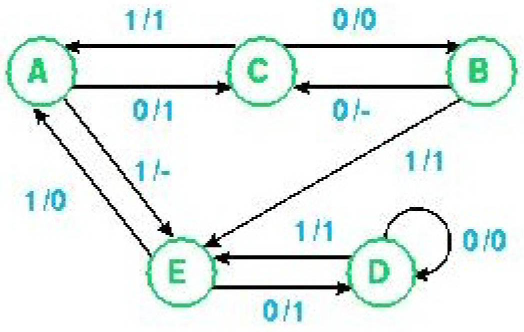


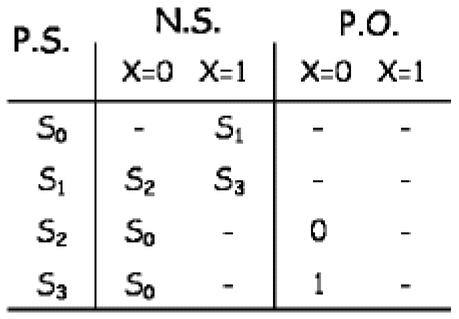
PS	NS, PO	
	x=0	x=1
A	E, 0	D, 1
B	F, 0	D, 0
C	E, 0	B, 1
D	F, 0	B, 0
E	C, 0	F, 1
F	B, 0	C, 0

Completely vs. Incompletely Specified FSMs

- Completely specified FSMs
  - δ must specify next state for all possible input and present state combinations
  - λ must specify output for all possible input and present state combinations
- Incompletely specified FSMs
  - Under some conditions, next-state or output values may be don't cares

Examples of incompletely specified FSMs

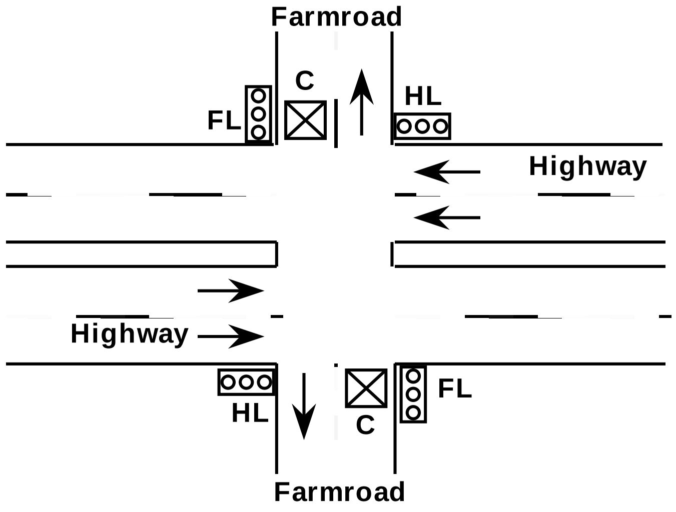




P.S.	N.S.		P.O.	
	X=0	X=1	X=0	X=1
S_0	-	S_1	-	-
S_1	S_2	S_3	-	-
S_2	S_0	-	0	-
S_3	S_0	-	1	-

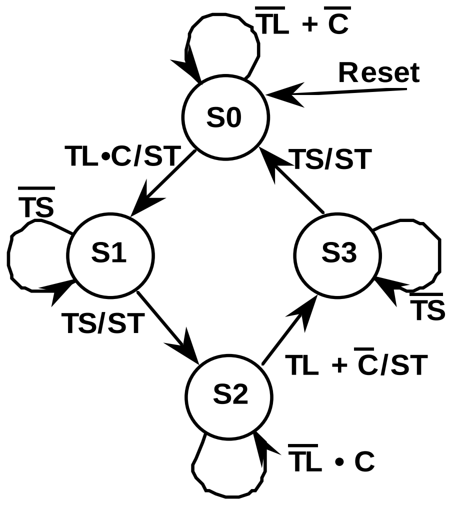
Example: Traffic Light Controller

- **Specification**: A busy highway is intersected by a little used farm road. Detectors sense the presence of cars waiting on the farm road. With no car on farm road, the lights remains green in the highway direction. If vehicles are present on the farm road, highway lights go from Green to Yellow to Red, allowing the farm road lights to become green. These stay green only as long as a farm road car is detected but never longer than a set interval. When these conditions are met, the farm road lights transition from Green to Yellow to Red, allowing the highway lights to return to green. Even if farm road vehicles are waiting, the highway gets at least a set interval as green. Assume you have an interval timer that generates a short time pulse (TS) and a long time pulse (TL) in response to a set (ST) signal. TS is to be used for timing yellow lights and TL for green lights.



Input Signal	*Description*
Reset	place FSM in initial state
C	detect vehicle on farmroad
TS	short time interval expired
TL	long time interval expired

Output Signal	*Description*
HG, HY, HR	assert green/yellow/red highway lights
FG, FY, FR	assert green/yellow/red farmroad lights
ST	start timing a short or long interval



S0: HG,FR

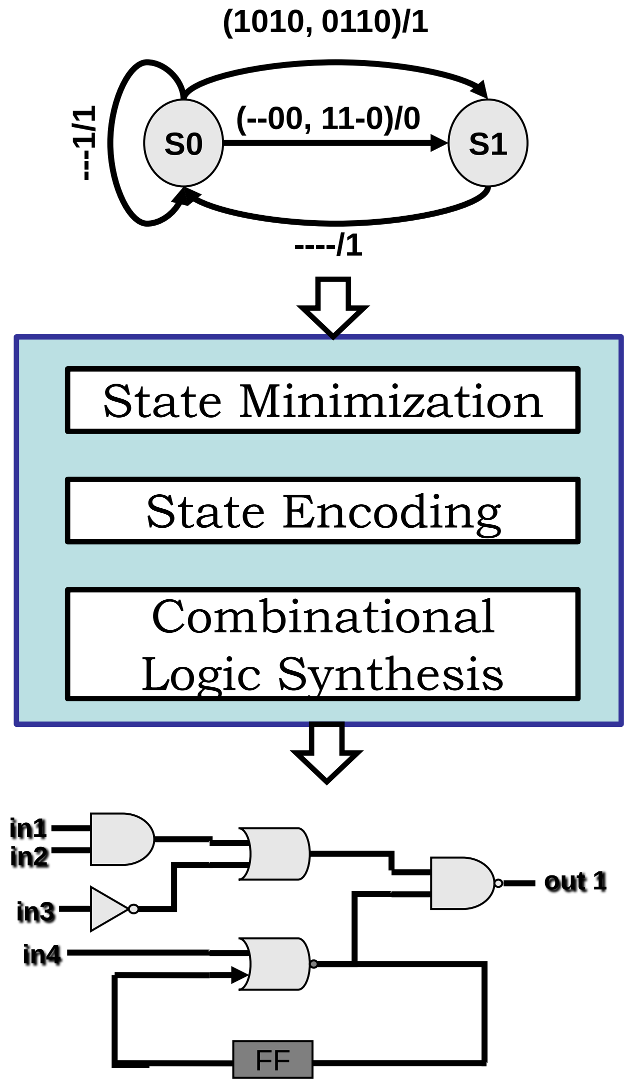
S1: HY,FR

S2: FG,HR

S3: FY,HR

FSM Synthesis

- Given FSM specification, synthesize optimized implementation (gates + FFs)
  - State minimization
  - State encoding
  - Derive next-state, output functions & apply combinational logic minimization techniques



(1010, 0110)/1
---1/1
S0
(--00, 11-0)/0
S1
----/1
State Minimization
State Encoding
Combinational Logic Synthesis
in1
in2
in3
in4
out 1
FF