ESE535: Electronic Design Automation

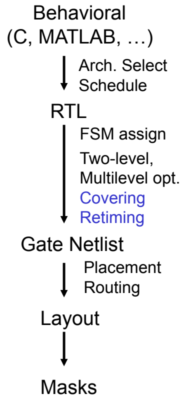
Day 25: April 17, 2013
Covering and Retiming

Previously

- Cover (map) LUTs for minimum delay
  - solve optimally for delay → flowmap
- Retiming for minimum clock period
  - solve optimally

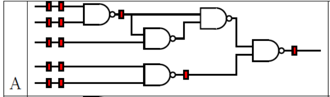
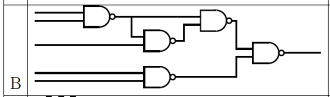
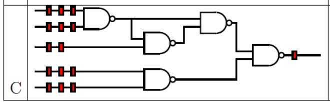
Today

- Solving cover/retime separately **not** optimal
- Cover+retime

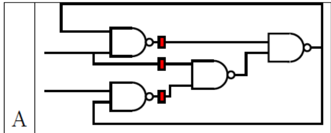
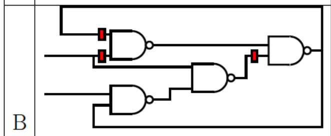


Behavioral (C, MATLAB, …)
Arch. Select
Schedule
RTL
FSM assign
Two-level,
Multilevel opt.
Covering
Retiming
Gate Netlist
Placement
Routing
Layout
Masks

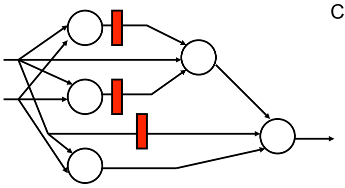
Preclass 1

	Circuit	3-LUTs?	critical path
A			
B			
C			

Preclass 2

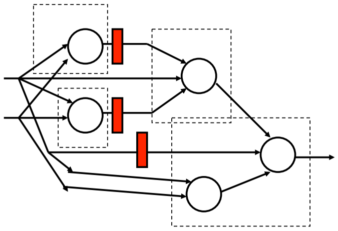
	Circuit	3-LUTs?	critical path
A			
B			

Example

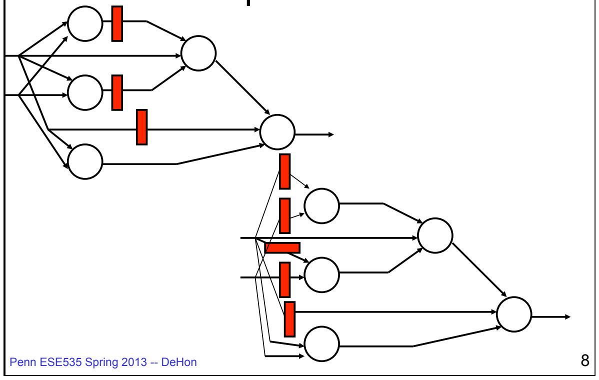


Cover with 4-LUTs

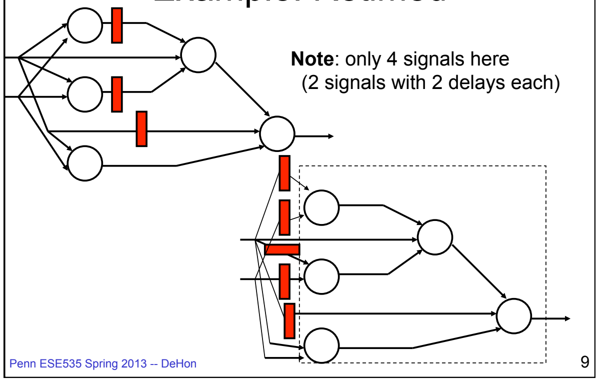
Example



Example: Retimed

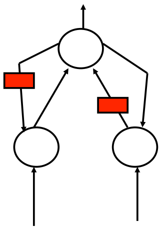


Example: Retimed



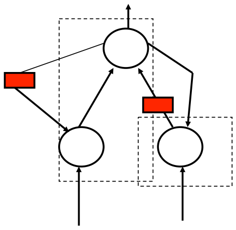
Note: only 4 signals here
(2 signals with 2 delays each)

Example 2



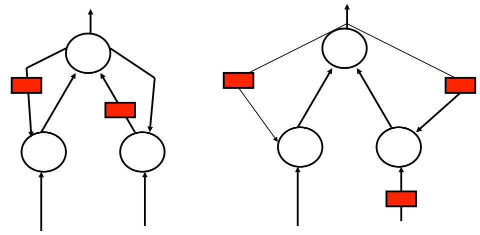
Cover with 4-LUTs

Example 2



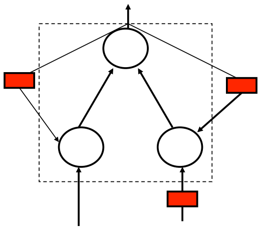
Cycle Bound: 2

Example 2: retimed



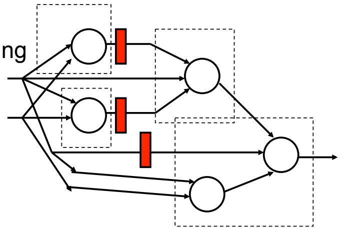
Example 2: retimed

Cycle Bound: 1



Basic Observation

- Registers break up circuit, limiting coverage
  - – fragmentation
  - – prevent grouping

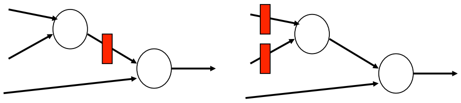


Phase Ordering Problem

- General problem
  - – don’t know effect of other mapping step
  - – Have seen this many places
- Here
  - – don’t know delay if retime first
    - don’t know what can be packed into LUT
  - – If we do not retime first
    - fragmentation: forced breaks at bad places

Observation #1

- Retiming flops to input of (fanout free) subgraph is trivial (and always doable)

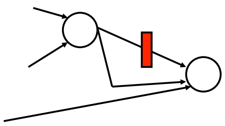


- Does not change I/O into subgraph

Observation #1: Consequence

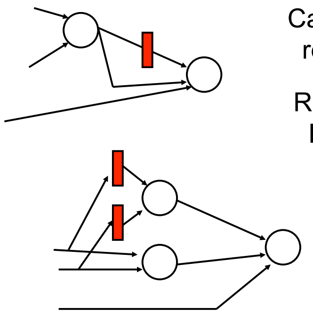
- Can cover *ignoring* flop placement
- Then retime flops to input of gates

Fanout Problem?



Can we use the same trick here?

Fanout Problem?



Cannot retime without replicating.

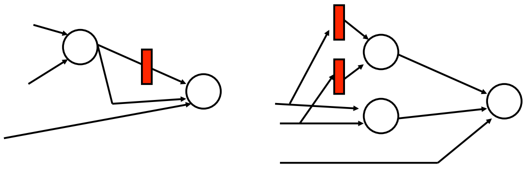
Replicating increases I/O (so cut size).

…but I/O is what defined feasible covers for LUTs

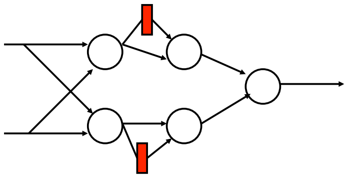
3 cut → 5 cut

Observation #2

- Retiming flops to input of a subgraph with fanout may change the subgraph I/O

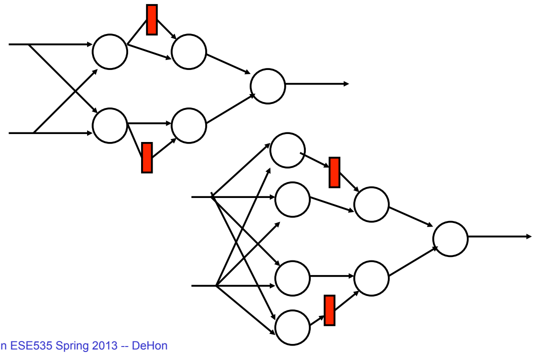


Different Replication Problem

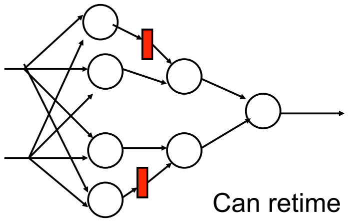


What does this do to I/O?

Different Replication Problem



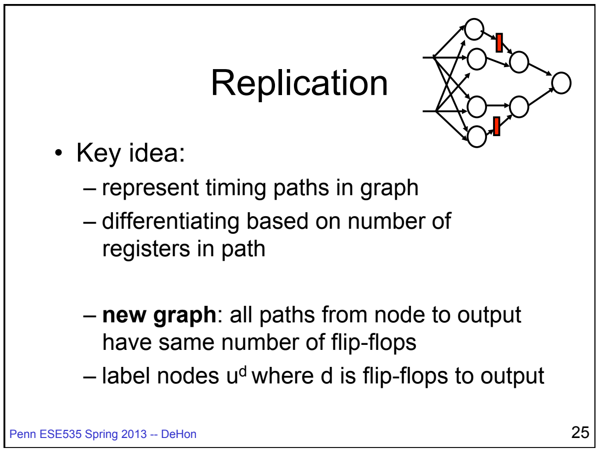
Different Replication Problem



Can retime and cover with single 4-LUT.

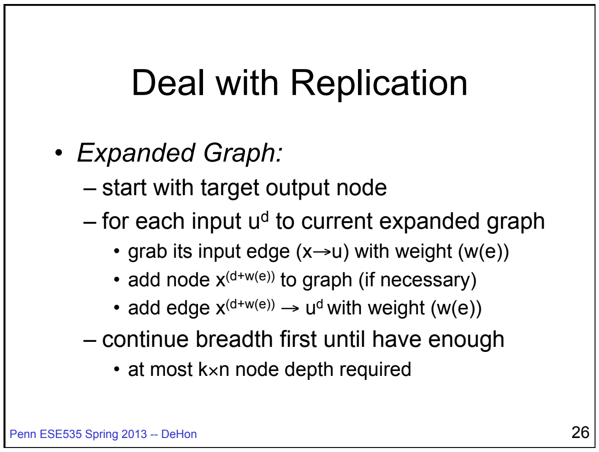
Replication

- Once add registers
  - can't just grab max flow and get replication
    - (compare flowmap)
- Or, can't just ignore flop placement when have reconvergent fanout through flop



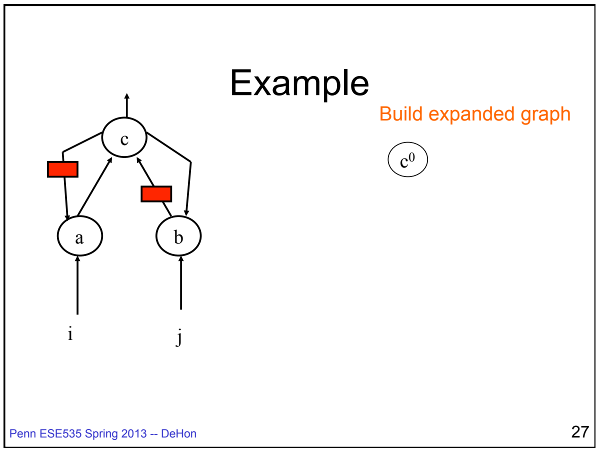
Replication

- Key idea:
  - represent timing paths in graph
  - differentiating based on number of registers in path
  - **new graph**: all paths from node to output have same number of flip-flops
  - label nodes u^d where d is flip-flops to output



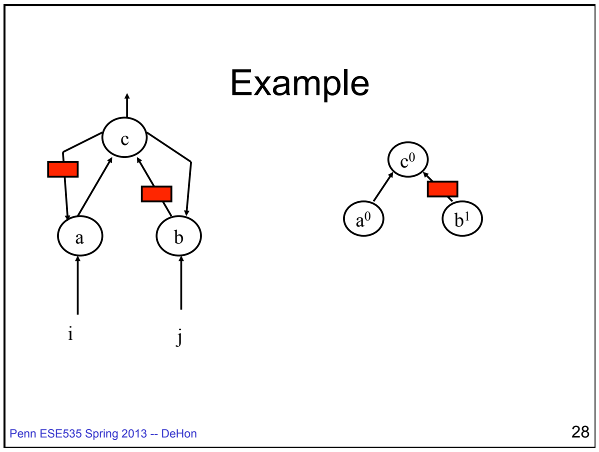
Deal with Replication

- *Expanded Graph:*
  - start with target output node
  - for each input u^d to current expanded graph
    - grab its input edge ($x \rightarrow u$) with weight ($w(e)$)
    - add node $x^{(d+w(e))}$ to graph (if necessary)
    - add edge $x^{(d+w(e))} \rightarrow u^d$ with weight ($w(e)$)
  - continue breadth first until have enough
    - at most k×n node depth required

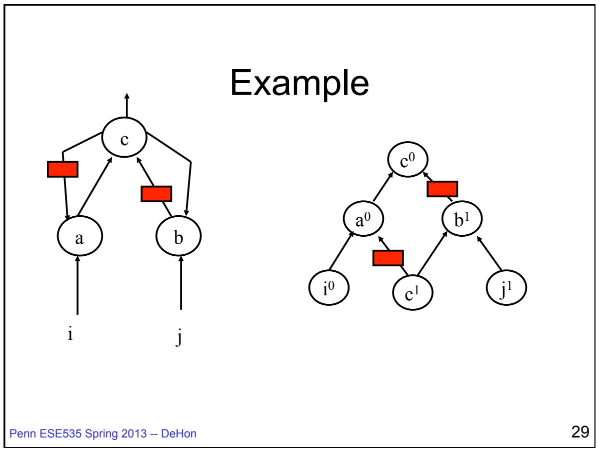


Example

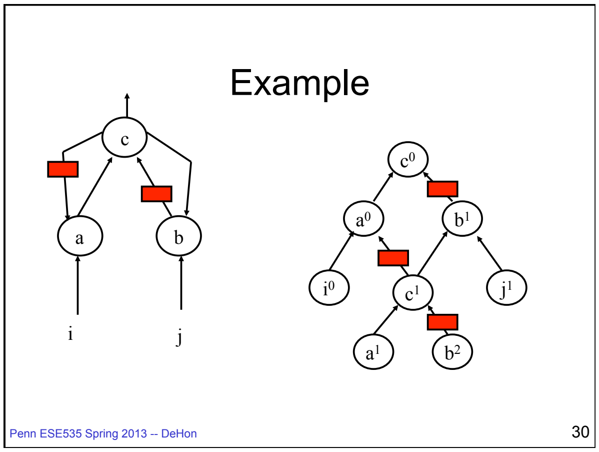
Build expanded graph



Example

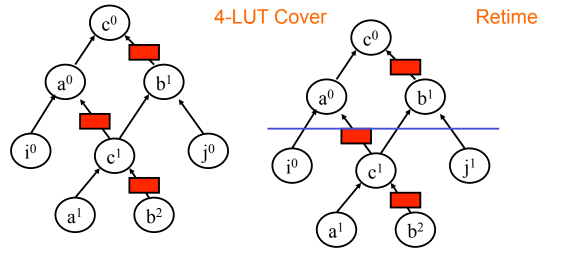


Example

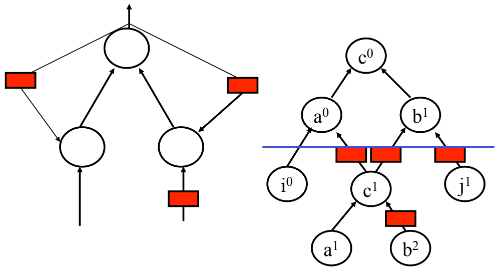


Example

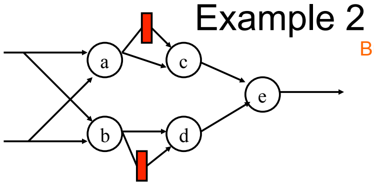
Example



Example

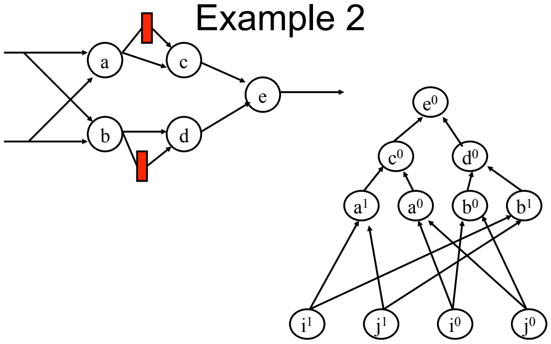


Example 2



Build expanded graph

Example 2



Expanded Graph

- Expanded graph does not have fanout of different flip-flop depths from the *same* node.
  - Captures IO after register retiming
- Can now cover ignoring flip-flops and trivially retime.

Intuition on Solution

- Phase ordering problem arise form
  - need to capture I/O effects before covering
  - but also need to model delay for register movement
    - But don't know register movement until after covering
- So, break retime into two pieces
  1. Expanded graph (capture I/O)
  2. Actual retime (moves registers)
- Do expanded graph piece before cover and register movement after

Intuition on Solution

- Break retime into two pieces
  1. Expanded graph (capture I/O)
  2. Actual retime (moves registers)
- Do expanded graph piece before cover and register movement after
- Not quite that simple since how much of expanded graph need depends on covering
  - So really doing just-in-time expansion in the middle of covering…
    - Before each cover/cut computation

Labeling

- Key idea #1:
  - compute distances/delay like flowmap
    - Try collapse and compute flow cut
    - Dynamic programming to compute min delay covers
- Key idea #2:
  - count distance from register
    - like G-1/c graph

Labeling: Edge Weights

- To target clock period c
  - use graph G-1/c
  - paper:
    - assign weight $-c*w(e)+1$
    - (same thing scaled by c and negated)

Labeling: Edge Weight Idea

- same idea:
  - will need register ever c LUT delays
  - credit with registers as encounter
  - charge a fraction (1/c) every LUT delay
  - know net distance at each point
  - if negative (delays > c*registers)
    - cannot distribute to achieve c
  - otherwise
    - labeling tells where to distribute

Labeling: Flow cut

- Label node as before (flowmap)
  - $L(v)=\min\{l(u)+d \mid \exists\, u \rightarrow v\}$
  - trivially can be $L(v)-1/c$ == new LUT
    - Correspond to flowmap case: $L(v)+1$
    - note min vs. max and -1/c vs. +1 due to rescaling to match retiming formulation and G-1/c graph
    - in this formulation, a combinational circuit of depth 4 would have $L(v)=-4/c$
  - if can put this and all L(v)'s in one LUT
    - this can be L(v)
    - construct and compute flow cut to test

LUT Map and Retime

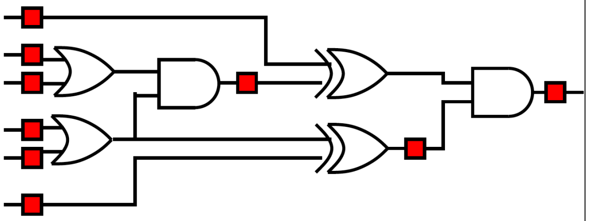
- Start with outputs
- Cover with LUT based on cut
  - move flip-flops to inputs of LUT
    - don't have meaningful labels for covered nodes
    - Know can do this by expanded graph construction
- Recursively cover inputs
- Use label to retime

$$r(v)=\lceil l(v)\rceil-1$$

Target Clock Period c

- As before (retiming)
  - binary search to find optimal c

Example



Summary

- Can optimally solve
  - LUT map for delay
  - retiming for minimum clock period
- But, solving separately does not give optimal solution to problem
- Can solve problems together
  - Account for registers on paths
  - Label based on register placement and (flow) cover ignoring registers
  - Labeling gives delay, covering, retiming

Today's Big Ideas

- Exploit freedom
- Cost of decomposition
  - benefit of composite solution
- Technique:
  - dynamic programming
  - network flow

Admin

- Monday reading online
- HW7 final due Monday