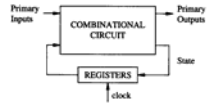


## Sequential Logic Optimization

11/29/2006 1

## Synchronous logic circuits

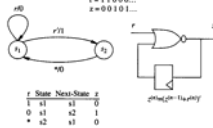
- Interconnection of:
  - Combinational logic gates
  - Synchronous delay elements
    - Edge-trigger or Master-Slave registers
- Assumptions
  - No direct combinational feedback
  - Single-phase clocking
- Modeling synchronous circuits
  - State-based models
  - Structural models



CS220: Synthesis of Digital System, Fall 06 2

## Modeling synchronous circuits

- State-based model:
  - Model circuits as finite-state machines
  - Represent by state tables/diagrams
  - Extracted SLN
  - Apply exact/heuristic algorithms for:
    - State minimization
    - State encoding
  - Minimize area and clock period
- Structural model:
  - Represent circuit by synchronous logic network
  - Encoded FSM
  - Apply
    - Retiming
    - Logic transformations
  - Optimize performance

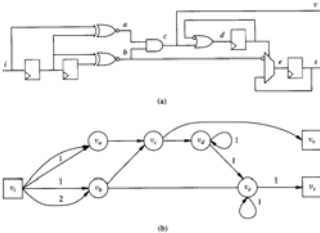


x	State	Next-State	z
0	s1	s1	0
0	s1	s2	1
1	s2	s1	0

CS220: Synthesis of Digital System, Fall 06 3

## Synchronous network graph

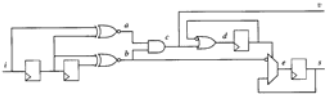
- Vertices
  - Equations
    - I/O, gates
- Edges
  - Dependencies
    - Nets
- Weights
  - Synchronous delays
    - registers



CS220: Synthesis of Digital System, Fall 06 4

## Synchronous logic circuit modeling

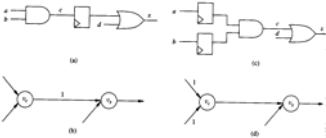
- State-based model
  - Transition diagrams or tables
  - Explicit notion of state
  - Implicit notion of area and delay
- Structural model
  - Synchronous logic network
  - Implicit notion of state
  - Explicit notion of area and delay



CS220: Synthesis of Digital System, Fall 06 5

## Retiming

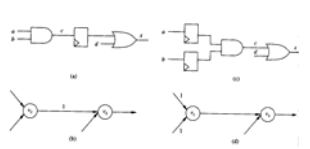
- Move register position
- Do not modify combinational logic
- Preserve network structure
  - Modify weights



CS220: Synthesis of Digital System, Fall 06 6

### Retiming

- Global optimization technique [Leiserson]
- Changes register positions
  - Affect area
    - Change register count
  - Affects cycle-time
    - Changes path delays between register pairs
- Solvable in polynomial time

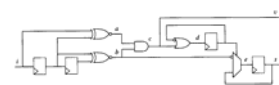


7

CS220: Synthesis of Digital System, Fall 06

### Assumptions

- Vertex delay is constant
  - No fanout delay dependency
    - Lead to inaccuracy
- Graph topology is invariant
  - No logic transformations
    - Lead to suboptimality
- Synchronous implementation
  - Cycles have positive weights
  - Edges have non-negative weights
- Consider topological paths
  - No false path analysis
    - Lead to inaccuracy



8

CS220: Synthesis of Digital System, Fall 06

### Retiming

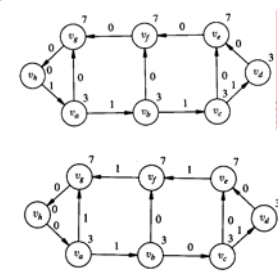
- Retiming of a vertex
  - Integer
  - Registers moved from output to input
- Retiming of a network
  - Vector of vertex retiming
- A family of equivalent networks are specified by
  - The original network
  - A retiming vector

9

CS220: Synthesis of Digital System, Fall 06

### Example

- Synchronous network
  - Close system



10

CS220: Synthesis of Digital System, Fall 06

### Definition and properties

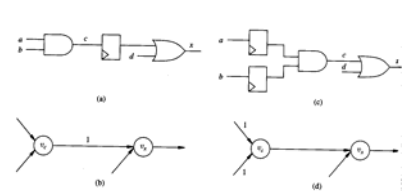
- Definitions
  - $d(v_i)$  – delay of node  $v_i$
  - $w(v_i, v_j)$  – weight of edge  $(v_i, v_j)$
  - $(v_i, \dots, v_j)$  – path from  $v_i$  to  $v_j$
  - $d(v_i, \dots, v_j)$  – path delay from  $v_i$  to  $v_j$  (inclusive)
  - $w(v_i, \dots, v_j)$  – path weight from  $v_i$  to  $v_j$
- Properties
  - Retiming of an edge  $(v_i, v_j)$   $\tilde{w}_{ij} = w_{ij} + r_j - r_i$
  - Retiming of a path  $(v_i, \dots, v_j)$   $\tilde{w}(v_i, \dots, v_j) = w(v_i, \dots, v_j) + r_j - r_i$
  - Cycle weights are invariant

11

CS220: Synthesis of Digital System, Fall 06

### Example

- Retiming vector  $\mathbf{r} = [1 \ 0]$ 
  - $r_c = 1, r_x = 0$



12

CS220: Synthesis of Digital System, Fall 06

### Example

- Retiming vector  $\mathbf{r} = -[11222100]^T$
- Delays goes from 24 to 13

13

CS220: Synthesis of Digital System, Fall 06

### Calculating Retiming

- A relaxation based method
- Calculate  $|V|$  time
- Guarantee to find a feasible solution
  - If one exists

```

FEAS( $G_{in}(V, E, W), \phi$ )
Set  $r = \emptyset$ 
Set  $\bar{G}_{in}(V, E, \bar{W}) = G_{in}(V, E, W)$ ;
for ( $k = 1$  to  $|V|$ ) {
  Compute the set of data-ready times;
  if ( $\max_{v_i \in V} t_i \leq \phi$ ) /* All path delays are bounded by  $\phi$  */
    return ( $\bar{G}_{in}(V, E, \bar{W})$ );
  else /* Some path delays exceed  $\phi$  */
    foreach vertex  $v_i$  such that  $t_i > \phi$ 
       $r_i = r_i + 1$ ; /* Retime vertices with excessive delay */
     $\bar{G}_{in}(V, E, \bar{W}) = G_{in}(V, E, W)$  retimed by  $r$ ;
}
return ( FALSE );
    
```

14

CS220: 1

### Retiming by relaxation

15

CS220: Synthesis of Digital System, Fall 06

### Library Binding

- A.k.a. Technology Mapping
- A.k.a. Technology dependent optimization

16

11/29/2006

### Library binding

- Given an unbound logic network and a set of library cells
  - Transform into an interconnection of instances of library cells
  - Optimize area, (under delay constraints)
  - Optimize delay, (under area constraints)
  - Optimize power, (under delay constraints)
- Called also technology mapping
  - Method used for re-designing circuits in different technologies

17

CS220: Synthesis of Digital System, Fall 06

### Library models

- Combinational elements
  - Single-output functions
    - E.g. AND, OR, AOI
  - Compound cells: e.g. adders, encoders
- Sequential elements
  - Registers counters
- Miscellaneous
  - Schmitt triggers

18

CS220: Synthesis of Digital System, Fall 06

### Heuristic algorithms

- Decomposition
  - Cast network and library in standard form
  - Decompose into base functions
  - Example: NAND2 and INV
- Partitioning
  - Break network into cones
  - Reduce to many multi-input single-output subnetworks
    - Subject graphs
- Covering
  - Cover each subnetwork by library cells

19

CS220: Synthesis of Digital System, Fall 06

### Heuristic algorithms

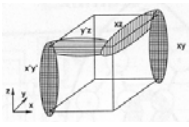
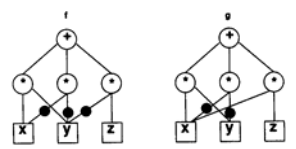
- Matching
  - Given two single-output combinational functions  $f(x)$  and  $g(y)$  with the same number of support variables,  $f$  matches  $g$  if there exists a permutation matrix  $P$  such that  $f(x) = g(Py)$  is a tautology
- Structural approach
  - Model functions by patterns
    - Example: trees, DAGs...
  - Rely on pattern matching techniques
- Boolean approach
  - Use Boolean models
  - Solve tautology problem
  - More powerful

20

CS220: Synthesis of Digital System, Fall 06

### Boolean vs structural

- $f = xy + x'y' + y'z$
- $g = xy + x'y' + xz$
- Function equality is a tautology
  - Boolean matched
- Patterns do not match
  - Structural match can not be found

21

CS220: Synthesis of Digital System, Fall 06

### Structural matching and covering

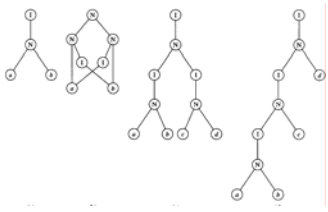
- Expression patterns
  - Represented by DAGs
- Identify pattern DAGs in network
  - Sub-graph isomorphism
- Simplification
  - Use tree patterns

22

CS220: Synthesis of Digital System, Fall 06

### Patterns

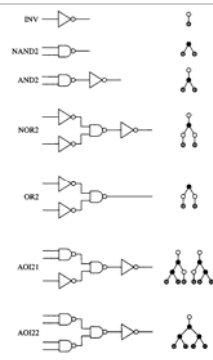
- Library elements are represented by pattern graph
  - AND, XOR, AOI
- Base function same as subject graph
  - NAND, INV



23

CS220: Synthesis of Digital System, Fall 06

### Library



24

CS220: Synthesis of Digital System, Fall 06

### Tree-based matching

- Network
  - Partitioned and decomposed
    - NOR2 (or NAND2) + INV
    - Generic base function
  - Subject tree
- Library
  - Represented by trees
  - Possibly more than one tree per cell
- Pattern recognition
  - Simple binary tree match
  - Aho-Corasick automaton

25

### Tree matching

```

MATCH(u, v) {
  if (u is a leaf) return (TRUE);           /* Leaf of the pattern graph reached */
  else {
    if (v is a leaf) return (FALSE);        /* Leaf of the subject graph reached */
    if (degree(v) != degree(u)) return(FALSE); /* Degree mismatch */
    if (degree(v) == 1) {                   /* One child each: visit subtree recursively */
      u1 = child of u; v1 = child of v;
      return (match(u1, v1))
    }
    else {                                  /* Two children each: visit subtrees recursively */
      u1 = left-child of u; u2 = right-child of u;
      v1 = left-child of v; v2 = right-child of v;
      return (MATCH(u1, v1) + MATCH(u2, v2))
    }
  }
}
    
```

26

### Tree covering

- Dynamic programming
  - Visit subject tree bottom-up
- At each vertex
  - Attempt to match
    - Locally rooted subtree
    - All library cells
- Optimum solution, for the subtree

27

### Dynamic Programming

- Principle of optimality
  - An optimum solution must contain optimum solution to subproblem
- May be VERY efficient
- E.g. Tree covering problem
  - At each vertex, match locally rooted subtree to all library cells
  - $O(V)$

SUBJECT TREE

PATTERN TREES

28

### Dynamic Programming

- Tree\_Cover(T(V,E))
  - Set the cost of the internal vertices to -1;
  - Set the cost of the leaf vertices to 0;
  - While (some vertex has negative weight) do {
    - Select a Vertex  $v \in V$  whose children have all nonnegative cost;
    - $M =$  set of all matching pattern trees at vertex  $v$ ;
    - $Cost(v) = \min_{m \in M(v)} (cost(m) + \sum_{u \in L(m)} cost(u))$ ;

SUBJECT TREE

PATTERN TREES

29

### Dynamic Programming

- Tree\_Cover(T(V,E))
  - Set the cost of the internal vertices to -1;
  - Set the cost of the leaf vertices to 0;
  - While (some vertex has negative weight) do {
    - Select a Vertex  $v \in V$  whose children have all nonnegative cost;
    - $M =$  set of all matching pattern trees at vertex  $v$ ;
    - $Cost(v) = \min_{m \in M(v)} (cost(m) + \sum_{u \in L(m)} cost(u))$ ;

SUBJECT TREE

PATTERN TREES

30

### Dynamic Programming

- **Tree\_Cover(T(V,E))**
  - Set the cost of the internal vertices to -1;
  - Set the cost of the leaf vertices to 0;
  - While (some vertex has negative weight) do {
    - Select a Vertex  $v \in V$  whose children have all nonnegative cost;
    - $M$  = set of all matching pattern trees at vertex  $v$ ;
  - $Cost(v) = \min_{m \in M(v)} (cost(m) + \sum_{u \in L(m)} cost(u))$ ;

**SUBJECT TREE**

**PATTERN TREES**

CS220: Synthesis of Digital System, Fall 06

### Dynamic Programming

- **Tree\_Cover(T(V,E))**
  - Set the cost of the internal vertices to -1;
  - Set the cost of the leaf vertices to 0;
  - While (some vertex has negative weight) do {
    - Select a Vertex  $v \in V$  whose children have all nonnegative cost;
    - $M$  = set of all matching pattern trees at vertex  $v$ ;
  - $Cost(v) = \min_{m \in M(v)} (cost(m) + \sum_{u \in L(m)} cost(u))$ ;

**SUBJECT TREE**

**PATTERN TREES**

CS220: Synthesis of Digital System, Fall 06

### Dynamic Programming

- **Tree\_Cover(T(V,E))**
  - Set the cost of the internal vertices to -1;
  - Set the cost of the leaf vertices to 0;
  - While (some vertex has negative weight) do {
    - Select a Vertex  $v \in V$  whose children have all nonnegative cost;
    - $M$  = set of all matching pattern trees at vertex  $v$ ;
  - $Cost(v) = \min_{m \in M(v)} (cost(m) + \sum_{u \in L(m)} cost(u))$ ;

**SUBJECT TREE**

**PATTERN TREES**

CS220: Synthesis of Digital System, Fall 06

### Dynamic Programming

- **Tree\_Cover(T(V,E))**
  - Set the cost of the internal vertices to -1;
  - Set the cost of the leaf vertices to 0;
  - While (some vertex has negative weight) do {
    - Select a Vertex  $v \in V$  whose children have all nonnegative cost;
    - $M$  = set of all matching pattern trees at vertex  $v$ ;
  - $Cost(v) = \min_{m \in M(v)} (cost(m) + \sum_{u \in L(m)} cost(u))$ ;

**SUBJECT TREE**

**PATTERN TREES**

CS220: Synthesis of Digital System, Fall 06

### Dynamic Programming

- **Tree\_Cover(T(V,E))**
  - Set the cost of the internal vertices to -1;
  - Set the cost of the leaf vertices to 0;
  - While (some vertex has negative weight) do {
    - Select a Vertex  $v \in V$  whose children have all nonnegative cost;
    - $M$  = set of all matching pattern trees at vertex  $v$ ;
  - $Cost(v) = \min_{m \in M(v)} (cost(m) + \sum_{u \in L(m)} cost(u))$ ;

**SUBJECT TREE**

**PATTERN TREES**

CS220: Synthesis of Digital System, Fall 06

### Dynamic Programming

- **Tree\_Cover(T(V,E))**
  - Set the cost of the internal vertices to -1;
  - Set the cost of the leaf vertices to 0;
  - While (some vertex has negative weight) do {
    - Select a Vertex  $v \in V$  whose children have all nonnegative cost;
    - $M$  = set of all matching pattern trees at vertex  $v$ ;
  - $Cost(v) = \min_{m \in M(v)} (cost(m) + \sum_{u \in L(m)} cost(u))$ ;

**SUBJECT TREE**

**PATTERN TREES**

CS220: Synthesis of Digital System, Fall 06

### Covering to minimize area

- Cost
  - INV: 2
  - NAND2: 3
  - AND2: 4
  - AOI21: 6
- Optimum solution: area = 9

Network	Subject graph	Vertex	Match	Gate	Cost
		x	i2	NAND2(a,c)	NAND2
		y	i1	INV(a)	INV
		z	i2	NAND2(a,d)	2 NAND2
		w	i2	NAND2(c,d)	3 NAND2+INV
		o	i1	INV(w)	3 NAND2+2INV
		i3		AND2(c,d)	2 NAND2+AND2+INV
		#6B		AOI21(c,d,a)	NAND2+AOI21

37

CS220: Synthesis of Digital System, Fall 06

### Covering to minimize area

- Cost
  - INV: 2
  - NAND2: 3
  - AND2: 4
  - AOI21: 6
- Optimum solution: area = 9

Network	Subject graph	Vertex	Match	Gate	Cost
		x	i2	NAND2(a,c)	NAND2
		y	i1	INV(a)	INV
		z	i2	NAND2(a,d)	2 NAND2
		w	i2	NAND2(c,d)	3 NAND2+INV
		o	i1	INV(w)	3 NAND2+2INV
		i3		AND2(c,d)	2 NAND2+AND2+INV
		#6B		AOI21(c,d,a)	NAND2+AOI21

38

CS220: Synthesis of Digital System, Fall 06

### Minimum delay cover

- Dynamic programming approach
- Cost related to gate delay
- Delay modeling
  - Constant gate delay
    - Straightforward
  - Load-dependent delay
    - Load fanout unknown
    - Binning techniques

39

CS220: Synthesis of Digital System, Fall 06

### Minimum delay cover constant delays

- The cell pattern tree and the rooted subtree are isomorphic
  - The vertex is labeled with the cell delay
- The cell tree is isomorphic to a subtree with leaves L
  - The vertex is labeled with the cell cost plus the maximum of the labels of L

40

CS220: Synthesis of Digital System, Fall 06

### Example

- Input data-ready times are 0 except  $t_d = 6$
- Constant delays: INV(2); NAND2(4); AND2(5); AOI21(10)
- Compute data-ready times bottom-up

Network	Subject graph	Vertex	Match	Gate	Cost
		x	i2	NAND2(a,c)	4
		y	i1	INV(a)	2
		z	i2	NAND2(a,d)	6 + 4 + 10
		w	i2	NAND2(c,d)	10 + 4 + 14
		o	i1	INV(w)	14 + 2 + 16
		i3		AND2(c,d)	10 + 5 + 15
		#6B		AOI21(c,d,a)	10 + 6 + 16

41

CS220: Synthesis of Digital System, Fall 06

### Example

- Input data-ready times are 0 except  $t_d = 6$
- Constant delays: INV(2); NAND2(4); AND2(5); AOI21(10)
- Compute data-ready times bottom-up

Network	Subject graph	Vertex	Match	Gate	Cost
		x	i2	NAND2(a,c)	4
		y	i1	INV(a)	2
		z	i2	NAND2(a,d)	6 + 4 + 10
		w	i2	NAND2(c,d)	10 + 4 + 14
		o	i1	INV(w)	14 + 2 + 16
		i3		AND2(c,d)	10 + 5 + 15
		#6B		AOI21(c,d,a)	10 + 6 + 16

42

CS220: Synthesis of Digital System, Fall 06

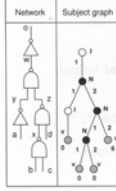
### Minimum delay cover load-dependent delays

- Model
  - Assume a finite set of load values
- Dynamic programming approach
  - Compute an array of solutions for each possible load
  - For each input to a matching cell the best match for any load is selected
- Optimum solution, when all possible loads are considered

43

### Example

- Inputs data-ready times are 0 except for  $t_d=6$
- Load-dependent delays
  - INV: 1+load
  - NAND2: 3+load
  - AND2: 4+load
  - AOI21:9+load
- SINV: 1+0.5load

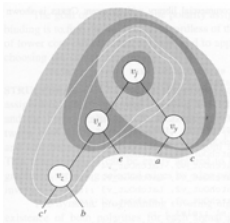


Network	Subject graph	Vertex	Match	Gate	Cost		
					Leads <sup>1</sup>	Leads <sup>2</sup>	Leads <sup>3</sup>
X	02	NAND2(a,b)	4	5	8		
Y	01	INV(a)	2	3	4		
Z	02	NAND2(a,b)	10	11	14		
W	02	NAND2(a,b)	14	15	18		
D	01	INV(a)			20		
E	03	AND2(a,b)			19		
H	03	AOI21(a,b)			20		
SINV	03	SINV(a)					18.5

44

### Boolean covering

- Decompose network into base functions
- When considering vertex  $v_i$ :
  - Construct clusters by local elimination
  - Several functions associated with  $v_i$
- Limit size and depth of clusters
- E.g.
  - $f_{j,1}=xy$
  - $f_{j,2}=x(a+c)$
  - $f_{j,3}=(e+z)y$
  - $f_{j,4}=(e+z)(a+c)$
  - $f_{j,5}=(e+c'+d)y$
  - $f_{j,6}=(e+c'+d)(a+c)$



45

### Boolean matching p-equivalence

- Cluster function  $f(x)$ : sub-network behavior
- Pattern function  $g(y)$ : cell behavior
- P-equivalence
  - Exists a permutation operator  $P$ , such that  $f(x)=g(Py)$  is a tautology
- Approaches
  - Tautology check over all input permutations
  - Multi-rooted pattern ROBDD capturing all permutations

46