ECE 697B (667)

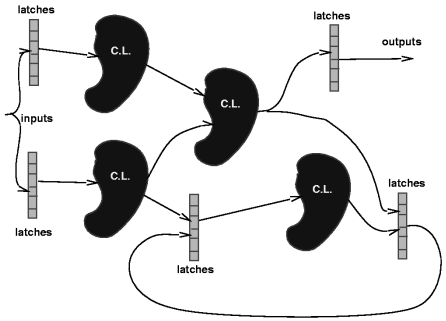
Spring 2003

Synthesis and Verification of Digital Systems

Multi-level Minimization

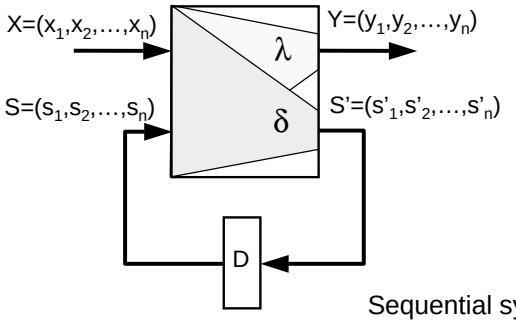
Slides adopted (with permission) from A. Kuehlmann, UC Berkeley 2003

General Logic Structure



- Combinational optimization
  - keep latches/registers at current positions, keep their function
  - optimize combinational logic in between
- Sequential optimization
  - change latch position/function (retiming)

Basic Model of Sequential ckt: FSM



M(X,Y,S,S_0,δ,λ):
- X: Inputs
- Y: Outputs
- S: Current State
- S_0: Initial State(s)
- δ: $X \times S \rightarrow S$ (next state function)
- λ: $X \times S \rightarrow Y$ (output function)

Sequential synthesis:

find (multi-level) implementation of δ(X) and λ(X) that minimize its cost (area, delay, power)

Delay elements:
- Clocked: synchronous
  - single-phase clock, multiple-phase clocks
- Unclocked: asynchronous

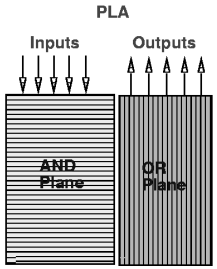
Optimization Criteria for Synthesis

The optimization criteria for multi-level logic is to *minimize* some function of:

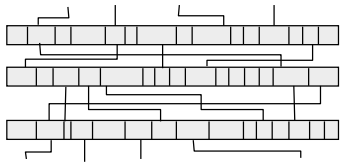
1. Area occupied by the logic gates and interconnect (approximated by literals = transistors in technology independent optimization)
2. Critical path delay of the longest path through the logic
3. Degree of testability of the circuit, measured in terms of the percentage of faults covered by a specified set of test vectors for an approximate fault model (e.g. single or multiple stuck-at faults)
4. Power consumed by the logic gates
5. Noise Immunity
6. Place-ability, Wire-ability

while simultaneously satisfying upper or lower bound constraints placed on these physical quantities

Two-Level (PLA) vs. Multi-Level



E.g. Standard Cell Layout



PLA

control logic
constrained layout
highly automatic
technology independent
multi-valued logic
input, output, state encoding
Very predictable

Multi-level Logic

all logic
general (e.g. standard cell, regular blocks,..)
automatic
partially technology independent
some ideas
part of multi-level logic
Very hard to predict

Transformation-based Synthesis

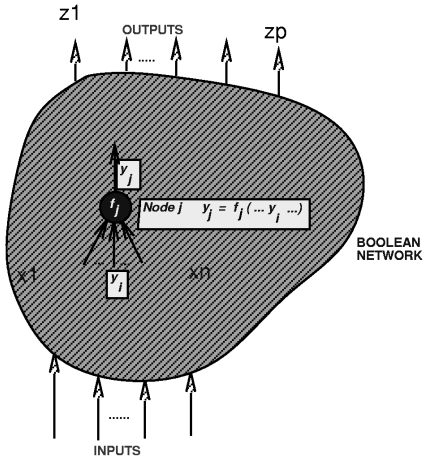
- All modern synthesis systems are build that way
  - set of transformations that change network representation
    - work on uniform network representation
  - "script" of "scenario" that can combine those transformations to a overall greedy
- Transformations differ in:
  - the scope they are applied
    - local scope versus global restructuring
  - the domain they optimize
    - combinational versus sequential
    - timing versus area
    - technology independent versus technology dependent
  - the underlying algorithms they use
    - BDD based, SAT based, structure based

Network Representation

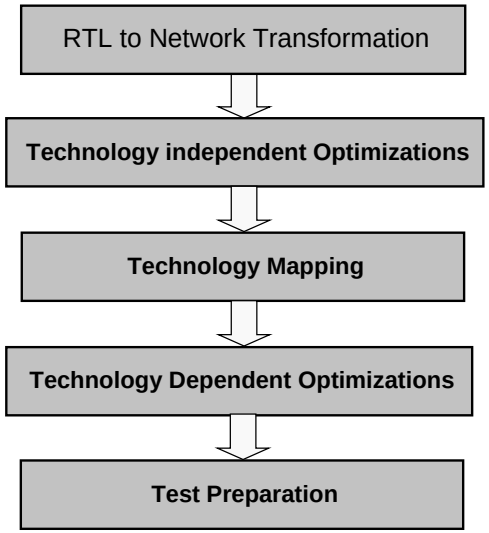
Boolean network:

- directed acyclic graph (DAG)
- node logic function representation $f_j(x,y)$
- node variable y_j: $y_j = f_j(x,y)$
- edge (i,j) if f_j depends explicitly on y_i

Inputs $x = (x_1, x_2, \ldots, x_n)$
Outputs $z = (z_1, z_2, \ldots, z_p)$
External don't cares: $d_1(x), \ldots, d_p(x)$



Typical Synthesis Scenario



- read HDL
- control/data flow analysis

- basic logic restructuring
- crude measures for goals

- use logic gates from target cell library

- timing optimization
- physically driven optimizations

- improve testability
- test logic insertion

Local versus Global Transformations

- Local transformations optimize the function of one node of the network
  - smaller area
  - bettter performance
  - map to a particular set of cells (library)
- Global transformations restructure the entire network
  - merging nodes
  - spitting nodes
  - removing/changing connections between nodes
- Node representation:
  - SOP, POS
  - BDD
  - Factored forms
  - keep size bounded to avoid blow-up of local transformations

Sum of Products (SOP)

Example:

abc'+a'bd+b'd'+b'e'f *(sum of cubes)*

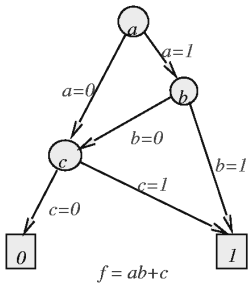
Advantages:

- easy to manipulate and minimize
- many algorithms available (e.g. AND, OR, TAUTOLOGY)
- two-level theory applies

Disadvantages:

- Not representative of logic complexity. For example
  f=ad+ae+bd+be+cd+ce f'=a'b'c'+d'e'
  These differ in their implementation by an inverter.
- hence not easy to estimate logic; difficult to estimate progress during logic manipulation

Reduced Ordered BDDs



- Like factored form, represents both function and complement
- Like network of muxes, but restricted since controlled by primary input variables
  - not really a good estimator for implementation complexity
- Given an ordering, reduced BDD is canonical, hence a good replacement for truth tables
- For a good ordering, BDDs remain reasonably small for complicated functions (e.g. not multipliers)
- Manipulations are well defined and efficient
- True support (dependency) is displayed

Factored Forms

Example:

(ad+b'c)(c+d'(e+ac'))+(d+e)fg

Advantages

- good representative of logic complexity
  f=ad+ae+bd+be+cd+ce f'=a'b'c'+d'e' ⇒ *f=(a+b+c)(d+e)*
- in many designs (e.g. complex gate CMOS) the implementation of a function corresponds directly to its factored form
- good estimator of logic implementation complexity
- doesn't blow up easily

Disadvantages

- not as many algorithms available for manipulation
- hence often just convert into SOP before manipulation