MECHTRON 3TB4
Optimization (Synthesis)

Tradeoffs in optimization during synthesis

- performance (latency)
  - here we consider the number of gate delays
- size
  - size is typically highly correlated with power consumption
  - we choose the number of gate inputs as a proxy for size (typically, there are two transistors for each gate input, thus this is a reasonable proxy)

Two-level minimization of a logic function

- represent logic function as a sum of products (or product of sums)
- gives best possible performance (at most two gate delays)
- if restricted to best possible performance, can we minimize size?

Example

a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Example (cont'd)

- direct cover: $f = ab\bar{c}\bar{d} + \bar{a}\bar{b}cd + \bar{a}bcd + a\bar{b}cd$
- this corresponds to four four-input AND gates and one four-input OR gate
- a total of 20 gate inputs (40 transistors)
- simple to compute
- can we do better (while maintaining two gate delays of latency)?

Minimum cover

- you already know how to minimize the size: use K-maps!
- need to find the minimum cover that is prime (cover all 1's with the minimum number of circles of largest size)
- I will leave you to do the K-map, but the result is: $f = ab\bar{c}\bar{d} + \bar{a}cd + \bar{b}cd$
- this corresponds to one four-input AND gate, two three-input AND gates, one three-input OR gate
- a total of 13 gate inputs (26 transistors)
- significant decrease over direct cover
- one significant problem: increased complexity

Heuristics

- using K-maps always yields the optimal solution
- however, functions with more than six inputs tend to be too complicated
- there are tabular methods that do find the optimal solution
- problem is that there are 2^n minterms for n inputs
- for example, 32 inputs have approximately four billion minterms
- result is exponential complexity
- synthesis tools use heuristics

Heuristics: iterative improvement

- start with original solution (direct cover, for example)
- repeatedly make modifications toward better solution
- common modifcations
  1. expand: replace each nonprime implicant with a prime implicant covering it (reminder: prime implicant corresponds to a circle on the K-map that cannot be made larger), then delete all implicants covered by new prime implicant
  2. reduce: opposite of expand
  3. reshape: expand one implicant while reducing another
  4. irredundant: select minimum number of implicants that cover from existing implicants
- synthesis tools differ in modifications used and the order that they are used

Multilevel logic minimization

- trade performance for size
- increase delay for lower number of gate inputs

Example

- suppose that we have a minimized two-level logic function: $f = adef + bdef + cdef + gh$
- corresponds to 18 gate inputs
- now, perform algebraic manipulation to get: $f = (a + b + c)def + gh$
- new expression has 11 gate inputs (22 transistors), but there is a sacrifice in performance, as the latency is three gate delays
- like low-level minimization, iterative improvement heuristic commonly used
- with trade-off, no clear optimal solution (need to define what is meant by optimal)

FSM synthesis

- we already discussed state minimization (merge equivalent states)
- some optimization may be possible by appropriate choice of state encoding
- most power consumed when flip-flops change state, so try to minimize the number of bit flips
- example given in lecture: four-state FSM, with transitions $A \to B \to C \to D \to A \to B \to \cdots$ (a cyclic structure)
- if we choose the state encoding of 00 for A, 01 for B, 10 for C, and 11 for D, there are two bit flips for the transitions from B to C and from D to A
- can do better with Gray code: 00 for A, 01 for B, 11 for C, 10 for D (one bit flip per transition)
- minimizing the (average) number of bit flips is not so easy for general FSMs: need to know the structure and the likelihood that each of the transitions are taken