

CS184a: Computer Architecture (Structure and Organization)

Day 2: January 5, 2003
Logic and FSM Review



Caltech CS184 Winter2005 -- DeHon

Last Time

- Computational Design as an Engineering Discipline
- Importance of Costs

Caltech CS184 Winter2005 -- DeHon

Today

- Simple abstract computing building blocks
 - gates, boolean logic
 - registers, RTL
- Logic in Gates
 - optimization
 - properties
 - costs
- Sequential Logic

Caltech CS184 Winter2005 -- DeHon

Stateless Functions/Comb. Logic

- Compute some “**function**”
 - $f(i_0, i_1, \dots, i_n) \rightarrow o_0, o_1, \dots, o_m$
- Each unique input vector
 - implies a particular, deterministic, output vector

Caltech CS184 Winter2005 -- DeHon

Specification in Boolean logic

- $o = a + b$
- $o = \neg(a * b)$
- $o = a * b$
- $o = a * b + b$
- $o = a * b + b * c + d * e + b * f + f * a + abcdef$

- $o = (a + b) / (b + c) + b * c$

Caltech CS184 Winter2005 -- DeHon

Implementation in Gates

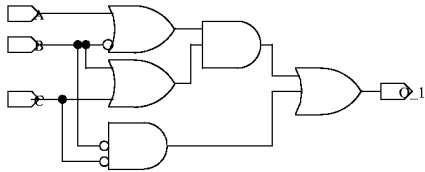
- **Gate:** small Boolean function
- **Goal:** assemble gates to *cover* our desired Boolean function

- Collection of gates should implement *same* function
- *I.e.* collection of gates and Boolean function should have same Truth Table

Caltech CS184 Winter2005 -- DeHon

Covering with Gates

– $o = (a + b)(b + c) + b^*c$



Caltech CS184 Winter2005 -- DeHon

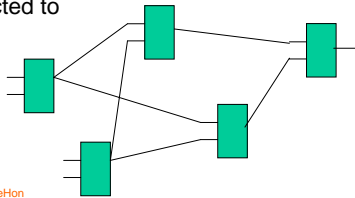
Equivalence

- There is a **canonical** specification for a Boolean function
 - its Truth Table
- Two expressions, gate netlists, a gate netlist and an expression -- are the same iff.
 - They have the same truth table

Caltech CS184 Winter2005 -- DeHon

Netlist

- **Netlist:** collection of interconnected gates
 - A list of all the gates and what they are connected to



Caltech CS184 Winter2005 -- DeHon

Caltech CS184 Winter2005 -- DeHon

Truth Table

• $o = a^*b^*c + a^*b^*/c + a^*b^*/c + a^*/b^*c$

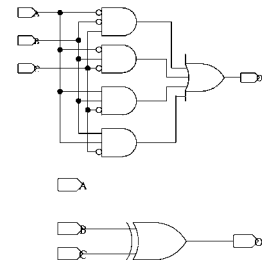
a	b	c	o
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Caltech CS184 Winter2005 -- DeHon

How many Gates?

• $o = a^*b^*c + a^*b^*/c + a^*b^*/c + a^*/b^*c$

a	b	c	o
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

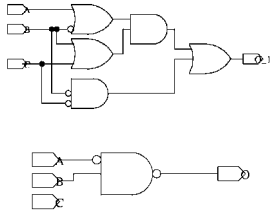


Caltech CS184 Winter2005 -- DeHon

How many gates?

$$o = (a+b)(b+c) + b^*c$$

a	b	c	o
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Caltech CS184 Winter2005 -- DeHon

Engineering Goal

- Minimize resources
 - area, gates
- Exploit *structure* of logic
- “An Engineer can do for a dime what everyone else can do for a dollar.”

Caltech CS184 Winter2005 -- DeHon

Sum of Products

- $o = a^*b^*c + a^*b^*c + a^*b^*c + a^*b^*c$
- $o = (a+b)(b+c)$
 - $a^*b + a^*c + b^*c$
- $o = (a+b)(b+c) + b^*c$
 - $a^*b + a^*c + b^*c + b^*c$

Caltech CS184 Winter2005 -- DeHon

Minimum Sum of Products

- $o = a^*b^*c + a^*b^*c + a^*b^*c + a^*b^*c$
 $\quad \quad \quad /b^*c + b^*c$

Caltech CS184 Winter2005 -- DeHon

Minimum Sum of Products

- $o = (a+b)(b+c)$
 - $a^*b + a^*c + b^*c$
 - $a^*b + a^*c + b^*c$
 - $a^*b + b^*c$

		ab			
		00	01	11	10
c	0	0	1	1	1
	1	0	0	0	1

Caltech CS184 Winter2005 -- DeHon

Redundant Terms

- $o = (a+b)(b+c)$
 - $a^*b + a^*c + b^*c$
 - $a^*b + b^*c$

		ab			
		00	01	11	10
c	0	0	1	1	1
	1	0	0	0	1

Caltech CS184 Winter2005 -- DeHon

There is a Minimum Area Implementation

- $o = (a+b)(b+c)$
 - $a^*b + a^*c + b^*c$
 - $a^*b + b^*c$

		ab			
		00	01	11	10
		0	0	1	1
c		1	0	0	1

Caltech CS184 Winter2005 -- DeHon

There is a Minimum Area Implementation

- Consider all combinations of less gates:
 - any smaller with same truth table?
 - There must be a smallest one.

Caltech CS184 Winter2005 -- DeHon

Not Always MSP

- $o = (a+b)(c+d)$
 - $a^*b + a^*c + b^*c + b^*d$
- Product of Sums smaller...
 - 3 2-input gates
 - 7 2-input gates

Caltech CS184 Winter2005 -- DeHon

Minimize Area

- Area minimizing solutions **depends** on the technology cost structure
- Consider:
 - I1: $((a^*b) + (c^*d))^*e^*f$
 - I2: $((a^*b^*e^*f) + (c^*d^*e^*f))$
- Area:
 - I1: $2^*A(\text{and2}) + 1^*A(\text{or2}) + 1^*A(\text{and3})$
 - I2: $2^*A(\text{and4}) + 1^*A(\text{or2})$

Caltech CS184 Winter2005 -- DeHon

Minimize Area

- I1: $((a^*b) + (c^*d))^*e^*f$
- I2: $((a^*b^*e^*f) + (c^*d^*e^*f))$
- Area:
 - I1: $2^*A(\text{and2}) + 1^*A(\text{or2}) + 1^*A(\text{and3})$
 - I2: $2^*A(\text{and4}) + 1^*A(\text{or2})$
- all gates take unit area:
 - $A(I2) = 3 < A(I1) = 4$
- gate size proportional to number of inputs:
 - $A(I1) = 2^*2 + 2 + 3 = 9 < A(I2) = 2^*4 + 2 = 10$

Caltech CS184 Winter2005 -- DeHon

Best Solution Depends on Costs

- This is a simple instance of the general point:
 - ... When technology costs change
 - the optimal solution changes.
- In this case, we can develop an automated decision procedure which takes the costs as a parameter.

Caltech CS184 Winter2005 -- DeHon

Don't Cares

- Sometimes will have incompletely specified functions:

a	b	c	o
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	x
1	0	0	x
1	0	1	0
1	1	0	0
1	1	1	0

Caltech CS184 Winter2005 -- DeHon

Don't Cares

- Will want to pick don't care values to minimize implementation costs:

a	b	c	o	a	b	c	o
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1
0	1	1	x	0	1	1	1
1	0	0	x	1	0	0	0
1	0	1	0	1	0	1	0
1	1	0	0	1	1	0	0
1	1	1	0	1	1	1	0

Caltech CS184 Winter2005 -- DeHon

NP-hard in General

- Logic Optimization
 - Two Level Minimization
 - Covering w/ reconvergent fanout
- Are NP-hard in general
 - ...but that's not to say it's not viable to find an optimal solution.
- Cover how to attack in CS137
 - can point you at rich literature
 - can find software to do it for you

Caltech CS184 Winter2005 -- DeHon

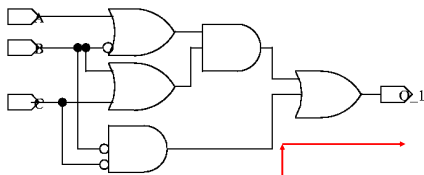
Delay in Gates

- Simple model:
 - each gate contributes a fixed delay for passing through it
 - can be different delay for each gate type
 - e.g.
 - inv = 50ps
 - nand2=100ps
 - nand3=120ps
 - and2=130ps

Caltech CS184 Winter2005 -- DeHon

Path Delay

- Simple Model: Delay along path is the sum of the delays of the gates in the path



$$\text{Path Delay} = \text{Delay}(\text{And3i2}) + \text{Delay}(\text{Or2})$$

Caltech CS184 Winter2005 -- DeHon

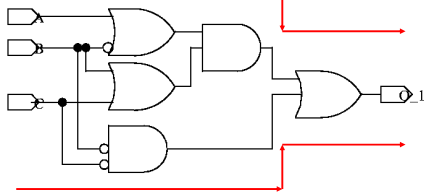
Critical Path

- Path lengths in circuit may differ
- Worst-case performance of circuit determined by the longest path
- Longest path designated **Critical Path**

Caltech CS184 Winter2005 -- DeHon

Multiple Paths

Path Delay = Delay(Or2i1)+Delay(And2)+Delay(Or2)

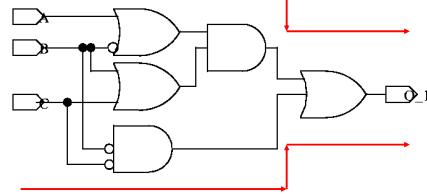


Path Delay = Delay(And3i2)+Delay(Or2)

Caltech CS184 Winter2005 -- DeHon

Critical Path = Longest

Path Delay = 3

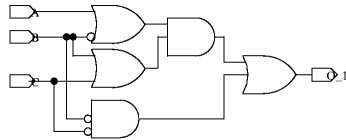


Path Delay = 2

Caltech CS184 Winter2005 -- DeHon

Critical Path

- There is always a set of critical paths
 - set such that the path length of the members is at least as long as any other path length
- May be many such paths



Caltech CS184 Winter2005 -- DeHon

Minimum Delay

- There is a minimum delay for a given function and technology cost.
- Like area:
 - consider all circuits of delay 1, 2,
 - Work from 0 time (minimum gate delay) up
 - stop when find a function which implements the desired logic function
 - by construction no smaller delay implements function

Caltech CS184 Winter2005 -- DeHon

Delay also depend on Costs

- Consider again:
 - I1: $((a*b) + (c*d))*e*f$
 - I2: $((a*b*e*f)+(c*d*e*f))$
- Delay:
 - I1: $D(\text{and2})+D(\text{or2})+D(\text{and3})$
 - I2: $D(\text{and4})+D(\text{or2})$

Caltech CS184 Winter2005 -- DeHon

Delay also depend on Costs

- Delay:
 - I1: $D(\text{and2})+D(\text{or2})+D(\text{and3})$
 - I2: $D(\text{and4})+D(\text{or2})$
- $D(\text{and2})=130\text{ps}$, $D(\text{and3})=150\text{ps}$, $D(\text{and4})=170\text{ps}$
 - $D(I2)=(170+D(\text{or2})) < D(I1)=(130+150+D(\text{or2}))$
- $D(\text{and2})=90\text{ps}$, $D(\text{and3})=100\text{ps}$, $D(\text{and4})=200\text{ps}$
 - $D(I2)=(200+D(\text{or2})) > D(I1)=(90+100+D(\text{or2}))$

Caltech CS184 Winter2005 -- DeHon

Delay and Area Optimum Differ

- I1: $((a*b) + (c*d))*e*f$
- I2: $((a*b*e*f)+(c*d*e*f))$
- $D(I2)=130ps$, $D(I3)=150ps$, $D(I4)=170ps$
 - $D(I2)<D(I1)$
- gate size proportional to number of inputs:
 - $A(I1)<A(I2)$
- **Induced Tradeoff** -- cannot always *simultaneously* minimize area and delay cost

Caltech CS184 Winter2005 -- DeHon

Delay in Gates make Sense?

- Consider a balanced tree of logic gates of depth (tree height) n .
- Does this have delay n ?
 - (unit delay gates)
- How big is it? (unit gate area)
- How long a side?
- Minimum wire length from input to output?

Caltech CS184 Winter2005 -- DeHon

Delay in Gates make Sense?

- (continuing example)
- How big is it? (unit gate area) 2^n
- How long a side? $\text{Sqrt}(2^n) = 2^{(n/2)}$
- Minimum wire length from input to output?
 - $2*2^{(n/2)}$
- Delay per unit length? (speed of light limit)
 - $\text{Delay} \propto 2^{(n/2)}$

Caltech CS184 Winter2005 -- DeHon

It's not all about costs...

- ...or maybe it is, just not always about a single, linear cost.
- Must manage complexity
 - Cost of developing/verifying design
 - Size of design can accomplish in fixed time
 - (limited brainpower)
- **Today:** human brainpower is most often the bottleneck resource limiting what we can build.

Caltech CS184 Winter2005 -- DeHon

Review Logic Design

- Input specification as Boolean logic equations
- Represent canonically
 - remove specification bias
- Minimize logic
- Cover minimizing target cost

Caltech CS184 Winter2005 -- DeHon

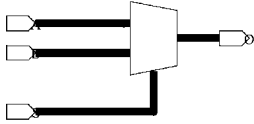
If's

- If $(a*b + /a*/b)$
 - $c=d$
- else
 - $c=e$
- $t=a*b+/a*/b$
- $c=t*d + /t*e$

Caltech CS184 Winter2005 -- DeHon

If → Mux Conversion

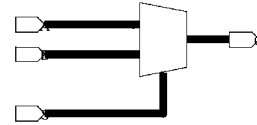
- Often convenient to think of IF's as Multiplexers
- If $(a*b + /a*/b)$
 $c=d$
- else
 $c=e$



Caltech CS184 Winter2005 -- DeHon

Muxes

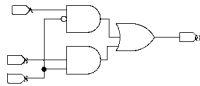
- Mux:
 – Selects one of two (several) inputs based on control bit



Caltech CS184 Winter2005 -- DeHon

Mux Logic

- Of course, Mux is just logic:
 – mux out = $/s*a + s*b$

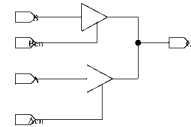


- Two views logically equivalent
 – mux view more natural/abstract when inputs are multibit values (datapaths)

Caltech CS184 Winter2005 -- DeHon

What about Tristates/busses?

- Tristate logic:
 – output can be 1, 0, or undriven
 – can wire together so outputs can share a wire



- Is this anything different?

Caltech CS184 Winter2005 -- DeHon

Tristates

- **Logically:**
 – No, can model correct/logical operation of tristate bus with Boolean logic
 – Bus undriven (or multiply driven) is Don't-Care case
 • no one should be depending on value
- **Implementation:**
 – sometimes an advantage in distributed control
 • don't have to build monolithic, central controller

Caltech CS184 Winter2005 -- DeHon

Finite Automata

- Recall from CS21
- A DFA is a quintuple $M=\{K,\Sigma,\delta,s,F\}$
 – K is finite set of states
 – Σ is a finite alphabet
 – $s \in K$ is the start state
 – $F \subseteq K$ is the set of final states
 – δ is a transition function from $K \times \Sigma$ to K

Caltech CS184 Winter2005 -- DeHon

Finite Automata

- Less formally:
 - Behavior depends not just on input
 - (as was the case for combinational logic)
 - Also depends on state
 - Can be completely different behavior in each state
 - Logic/output now depends on state and input

Caltech CS184 Winter2005 -- DeHon

Minor Amendment

- A DFA is a sextuple $M = \{K, \Sigma, \delta, s, F, \Sigma_o\}$
 - Σ_o is a finite set of output symbols
 - δ is a transition function from $K \times \Sigma$ to $K \times \Sigma_o$

Caltech CS184 Winter2005 -- DeHon

What power does the DFA add?

Caltech CS184 Winter2005 -- DeHon

Power of DFA

- Process unbounded input with finite logic
- State is a finite representation of what's happened before
 - finite amount of stuff can remember to synopsize the past
- State allows behavior to depend on past (on context)

Caltech CS184 Winter2005 -- DeHon

Registers

- New element is a state element
- Canonical instance is a register:
 - remembers the last value it was given until told to change
 - typically signaled by clock

Caltech CS184 Winter2005 -- DeHon

Issues of Timing...

- ...many issues in detailed implementation
 - glitches and hazards in logic
 - timing discipline in clocking
 - ...
- We're going to work above that level for the most part this term.
- Watch for these details in CS181

Caltech CS184 Winter2005 -- DeHon

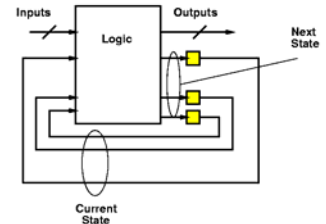
Same thing with registers

- Logic becomes:
 - if (state=s1)
 - boolean logic for state 1
 - (including logic for calculate next state)
 - else if (state=s2)
 - boolean logic for state2
 - ...
 - if (state=sn)
 - boolean logic for state n

Caltech CS184 Winter2005 -- DeHon

Finite-State Machine (FSM)

- Logic core
- Plus registers to hold state



Caltech CS184 Winter2005 -- DeHon

State Encoding

- States not (necessarily) externally visible
- We have *freedom* in how to encode them
 - assign bits to states
- Usually want to exploit freedom to minimize implementation costs
 - area, delay, energy
- (again, algorithms to attack -- cs137)

Caltech CS184 Winter2005 -- DeHon

Multiple, Interacting FSMs

- What do I get when I wire together more than one FSM?

Caltech CS184 Winter2005 -- DeHon

Multiple, Interacting FSMs

- What do I get when I wire together more than one FSM?
- Resulting composite is also an FSM
 - Input set is union of input alphabets
 - State set is product of states:
 - e.g. for every sa_i in A.K and sb_j in B.K there will be a composite state (sa_i, sb_j) in AB.K
 - Think about concatenating state bits

Caltech CS184 Winter2005 -- DeHon

Multiple, Interacting FSMs

- In general, could get product number of states
 - $|AB.K| = |A|^*|B|$... can get large fast
- All composite states won't necessarily be reachable
 - so real state set may be $< |A|^*|B|$

Caltech CS184 Winter2005 -- DeHon

Multiple, Interacting FSMs

- Multiple, “independent” FSMs
 - often have implementation benefits
 - localize inputs need to see
 - simplify logic
 - decompose/ease design
 - separate into small, understandable pieces
 - can sometimes obscure behavior
 - not clear what composite states are reachable

Caltech CS184 Winter2005 -- DeHon

FSM Equivalence

- Harder than Boolean logic
- Doesn't have unique canonical form
- Consider:
 - state encoding not change behavior
 - two “equivalent” FSMs may not even have the same number of states
 - can deal with **infinite** (unbounded) input
 - ...so cannot enumerate output in all cases

Caltech CS184 Winter2005 -- DeHon

FSM Equivalence

- What matters is external observability
 - FA accepts and rejects same things
 - FSM outputs same signals in response to every possible input sequence
- Possible?
 - Finite state suggests there is a finite amount of checking required to verify behavior

Caltech CS184 Winter2005 -- DeHon

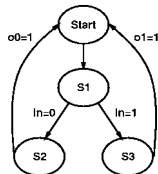
FSM Equivalence Flavor

- Given two FSMs A and B
 - consider the composite FSM AB
 - Inputs wired together
 - Outputs separate
- Ask:
 - is it possible to get into a composite state in which A and B output different symbols?
- There is a literature on this

Caltech CS184 Winter2005 -- DeHon

FSM Specification

- St1: goto St2
- St2:
 - if (I==0) goto St3
 - else goto St4
- St3:
 - output o0=1
 - goto St1
- St4:
 - output o1=1
 - goto St2
- Could be:
 - behavioral language
 - computer language
 - state-transition graph



Caltech CS184 Winter2005 -- DeHon

Systematic FSM Design

- Start with specification
- Can compute boolean logic for each state
 - If conversion...
 - including next state translation
 - Keep state symbolic (s1, s2...)
- Assign states
- Then have combinational logic
 - has current state as part of inputs
 - produces next state as part of outputs
- Design comb. Logic and add state registers

Caltech CS184 Winter2005 -- DeHon

Admin: Reminder

- No class **this** Friday
- Next class is Monday
 - Logic assignment due Monday

Caltech CS184 Winter2005 -- DeHon

Big Ideas [MSB Ideas]

- Can implement any Boolean function in gates
- Can implement any FA with gates and registers

Caltech CS184 Winter2005 -- DeHon

Big Ideas [MSB-1 Ideas]

- Canonical representation for combinational logic
- Transformation
 - don't have to implement the input literally
 - only have to achieve same semantics
 - trivial example: logic minimization
- There is a minimum delay, area
- Minimum depends on cost model

Caltech CS184 Winter2005 -- DeHon