Multi-level Synthesis

Today

- Multilevel Synthesis/Optimization
  - Why
  - Transforms -- defined
  - Division/extraction
    - How we support transforms

Multi-level Logic

- General circuit netlist
- May have
  - sums within products
  - products within sum
  - arbitrarily deep
- \[ y = ((a \ (b+c)+e)fg+h)i \]

Why Multi-level Logic?

- \( ab(c+d+e)(f+g) \)
- \( abcf+abdf+abef+abcg+abdg+abeg \)
- 6 product terms \( \Rightarrow \) 23 2-input gates
- vs. 3 gates: and4, or3, or2 \( \Rightarrow \) 6 2-input gates
- Aside from Pterm sharing between outputs,
  - two level cannot share sub-expressions

Why Multilevel

- \( a \ xor \ b \)
  - \( a/b+/ab \)
- \( a \ xor \ b \ xor \ c \)
  - \( a/bc+/abc+/a/b/c+ab/c \)
- \( a \ xor \ b \ xor \ c \ xor \ d \)
  - \( a/bcd+/abcd+/a/b/cd+/ab/cd+/ab/cd+/abcd+ia/bcd \)

Compare

- \( a \ xor \ b \)
  - \( x1=a/b+/ab \)
- \( a \ xor \ b \ xor \ c \)
  - \( x2=x1/c+/x1*c \)
- \( a \ xor \ b \ xor \ c \ xor \ d \)
  - \( x3=x2/d+/x2*d \)
Why Multilevel

- $a \oplus b$
  - $x_1 = a/b + ab$
- $a \oplus b \oplus c$
  - $x_2 = x_1/c + x_1\cdot c$
- $a \oplus b \oplus c \oplus d$
  - $x_3 = x_2/d + x_2\cdot d$

- Multi-level
  - exploit common sub-expressions
  - linear complexity
- Two-level
  - exponential complexity

Goal

- Find the structure
- Exploit to minimize gates
  - Total (area)
  - In path (delay)

Multi-level Transformations

- Decomposition
- Extraction
- Factoring
- Substitution
- Collapsing

[copy these to board so stay up as we move forward]

Decomposition

- $F = abc + abd + a/c/d + b/c/d$
  - 4 3-input + 1 4-input \Rightarrow 11$ 2$-input gates
- $F = XY + X/Y$
- $X = ab$
- $Y = c + d$
  - 5 2-input gates

Note: use $X$ and $/X$, use at multiple places

Extraction

- $F = (a+b)cd + e$
- $G = (a+b)/e$
- $H = cde$
- $F = XY + e$
- $G = X/e$
- $H = Ye$
- $X = a+b$
- $Y = cd$
Extraction

- \( F = (a+b)cd + e \)
- \( G = (a+b)/e \)
- \( H = cde \)
- 2-input: 4
- 3-input: 2
- \( \Rightarrow 8 \) 2-input gates

Common sub-expressions over multiple output

Factoring

- \( F = ac + ad + bc + bd + e \)
- \( G = X/e \)
- \( H = Ye \)
- \( X = a+b \)
- \( Y = cd \)
- 2-input: 6
- 5-input

Substitution

- \( G = a+b \)
- \( F = a+bc \)
- Substitute \( G \) into \( F \)
  - \( F = G(a+c) \)
  - \( F = (a+b)(a+c) = aa + ab + ac + bc = a+b \)
- useful if also have \( H = a+c \), then \( F = GH \)

Collapsing

- \( F = Ga+/Gb \)
- \( G = c+d \)
- \( F = ac + ad + b/c/d \)
- opposite of substitution
  - sometimes want to collapse and refactor
  - especially for delay optimization [saw last time]

Moves

- These transforms define the "moves" we can make to modify our network.
- Goal is to apply, usually repeatedly, to minimize gates
  - ...then apply as necessary to accelerate design
- MIS/SIS
  - Applies to canonical 2-input gates
  - Then covers with target gate library
- Day 2
Division

- **Given**: function \( f \) and divisor \( p \)
- **Find**: quotient and remainder
  \[ f = pq + r \]

*E.g.*

\[ f = abc + abd + ef, \quad p = ab \]
\[ q = c + d, \quad r = ef \]

Algebraic Division

- Use basic rules of algebra, rather than full boolean properties
- Computationally simple
- Weaker than boolean division
- **Algebra**: not divisible
- **Boolean**: \( q = (a + c), \quad r = 0 \)

Algebraic Division Example

(adv to alg.; work ex on board)

- \( f = abc + abd + de \)
- \( p = ab + e \)

Algebraic Division

- **Given**: function \( f \) and divisor \( p \)
- **Find**: quotient and remainder
  \[ f = pq + r \]
- \( f \) and \( p \) are expressions (lists of cubes)
  - \( p = \{a_1, a_2, \ldots\} \)
- Define: \( h_i = \{c_j \mid a_i * c_j \in f\} \)
- \( f/p = h_1 \cap h_2 \cap h_3 \ldots \)
Algebraic Division Example

- \( f = abc + abd + de, \ p = ab + e \)
- \( p = \{ab, e\} \)
- \( h_1 = \{c, d\} \)
- \( h_2 = \{d\} \)
- \( h_1 \cap h_2 = \{d\} \)
- \( f/p = d \)
- \( r = f - p \cdot (f/p) = abc + abd + de - (ab + e)d \)
- \( r = abc \)

Algebraic Division Time

- \( O(|f||p|) \) as described
  - compare every cube pair
- Sort cubes first
  - \( O((|f|+|p|)\log(|f|+|p|)) \)

Primary Divisor

- \( f/c \) such that \( c \) is a cube
- \( f = abc + abde \)
- \( f/a = bc + bde \) is a primary divisor

Cube Free

- The only cube that divides \( p \) is 1
- \( c + de \) is cube free
- \( bc + bde \) is not cube free

Kernel

- Kernels of \( f \) are
  - cube free primary divisors of \( f \)
  - Informally: sums w/ cubes factored out
- \( f = abc + abde \)
- \( f/ab = c + de \) is a kernel
- \( ab \) is cokernel of \( f \) to \( (c + de) \)
  - cokernels always cubes

Factoring

- \( Gfactor(f) \)
  if \( \text{terms} == 1 \) return \( f \)
  \( p = \text{CHOOSE\_DIVISOR}(f) \)
  \( (h, r) = \text{DIVIDE}(f, p) \)
  \( f = Gfactor(h) \cdot Gfactor(p) + Gfactor(r) \)
  return \( f \) // factored
Factoring

- Trick is picking divisor
  - pick from kernels
  - goal minimize literals after resubstitution
- Re-express design using new intermediate variables
- Variable and complement

Kernel Extraction

- Kernel1(j,g)
  - $R=g$
  - $N$ max index in $g$
  - for $i=[j+1$ to $N$)
    - if ($l_i$ in 2 or more cubes)
      - $c_f=$largest cube divide $g/l_i$
    - if (forall $k \leq i$, $l_k \not\in c_f$)
      - $R=R \cup$ KERNEL1($i$, $g/(l_i \cap c_f)$)
  - return(R)

Consider each literal for cokernel once
(largest cokernels will already have been found)

Kernel Extract Example

- $f=abcd+abce+abef$
  - $c_f=ab$
  - $f/c_f=cd+ce+ef$
  - $R={cd+ce+ef}$
  - $N=6$
  - a, b not present
  - $(cd+ce+ef)/c=e+d$
  - largest cube 1

Kernel Extract Example

- $f=abcd+abce+abef$
  - $c_f=ab$
  - $f/c_f=cd+ce+ef$
  - $R={cd+ce+ef}$
  - only 1 d
  - $(d+ce+ef)/e=c+f$
  - $R=cd+ce+ef$
  - Recurse $c+f$

Extraction

1. Generate kernels for each function
2. select pair such that $k1 \cap k2$ is not a cube
   - Note: $k1=k2$ is simplest case of this
   - ... but intersection case is more powerful
   - Example to come
3. new variable from intersection
   - $v= k1 \cap k2$
4. update functions (resubstitute)
   - $f_i = f_i \cup (v \cap r_i)$
   - (similar for common cubes)
Extraction Example

• $X = ab(c(d+e)+f+g)+g$
• $Y = ai(c(d+e)+f+j)+k$

Extraction Example

• $X = ab(c(d+e)+f+g)+g$
• $Y = ai(c(d+e)+f+j)+k$
• $d+e$ kernel of both
• $L = d+e$
• $X = ab(cL+f+g)+h$
• $Y = ai(cL+f+j)+k$

Extraction Example

• $L = d+e$
• $X = ab(cL+f+g)+h$
• $Y = ai(cL+f+j)+k$
• kernels: $(cL+f+g), (cL+f+j)$
• extract: $M = cL+f$
• $X = ab(M+g)+h$
• $Y = ai(M+f)+h$

Extraction Example

• $L = d+e$
• $M = cL+f$
• $X = ab(M+g)+h$
• $Y = ai(M+j)+h$
• no kernels
• common cube: $aM$

Extraction Example

• $N = aM$
• $M = cL+f$
• $L = d+e$
• $X = b(N+ag)+h$
• $Y = i(N+aj)+k$

Resubstitution

• Also useful to try complement on new factors
• $f = ab+ac+b/cd$
• $X = b+c$
• $f = aX+/b/cd$
• $/X = /b/c$
• $f = aX+/Xd$
• ...extracting complements not a direct target
Summary

• Want to exploit structure in problems to reduce (contain) size
  – common sub-expressions
• Identify component elements
  – decomposition, factoring, extraction
• Division key to these operations
• Kernels give us divisors

Admin

• Everyone should have received Assignment 6 feedback in email
• Reading for Monday online
• Milestone Mondays…

Big Ideas

• Exploit freedom
  – form
• Exploit structure/sharing
  – common sub expressions
• Techniques
  – Iterative Improvement
  – Refinement/relaxation