ESE535: 
Electronic Design Automation

Day 22: April 15, 2015
Multi-level Synthesis

Today

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

Behavioral
(C, MATLAB, …)
Arch. Select
Schedule
RTL
FSM assign
Two-level
Multilevel opt.
Covering
Retiming
Gate Netlist
Placement
Routing
Layout
Masks

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?

• \( ab(c+d+e)(f+g) \)
• \( abc+abdf+abef+abcg+abdg+abeg \)
• 6 product terms \( \to \) 23 2-input gates
• vs. 3 gates: and4, or3, or2 \( \to \) 6 2-input gates

• Aside from Pterm sharing between outputs,
  – two level cannot share sub-expressions

Why Multilevel

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

Why 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 \text{xor} b
  \quad x1 = a/b + ab
• a \text{xor} b \text{xor} c
  \quad x2 = x1/c + x1c
• a \text{xor} b \text{xor} c \text{xor} d
  \quad x3 = x2/d + x2d

• Multi-level
  \quad \text{exploit common sub-expressions}
  \quad \text{linear complexity}

Multi-level

• Two-level
  \quad \text{exponential complexity}

Harder than Two-Level

• Two-level already \text{NP-hard}
  \quad \text{has canonical representation}
  \quad \text{clean formulation}
  \quad \text{observed can limit to Primes}
  \quad \text{identified opportunities for pruning}

• Multi-level has more flexibility
  \quad \rightarrow \text{larger space to explore}
  \quad \text{Not formulated cleanly}

• Solution more heuristic \ldots \text{art}
  \quad \ldots\text{all problems start this way, some stay…}

Goal

• Find the structure
• Exploit to minimize gates
  \quad \text{Total (area)}
  \quad \text{In path (delay)}

Multi-level Transformations

• Decomposition
• Extraction
• Factoring
• Substitution
• Collapsing

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

Decomposition

• \( F = abc + abd + a/c + d/b + c/d \)
• \( F = XY + X/Y \)
• \( X = ab \)
• \( Y = c + d \)

Decomposition

• \( F = abc + abd + a/c + d/b + c/d \)
  \quad 4 \text{ 3-input} + 1 \text{ 4-input} \rightarrow 11 \text{ 2-input gates}
• \( F = XY + X/Y \)
• \( X = ab \)
• \( Y = c + d \)
  \quad 5 \text{ 2-input gates}

\text{Note: use} \ X \text{ and } /X, \text{ 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

- \( F = ac + ad + bc + bd + e \)
- \( F = (a+b)(c+d) + e \)

Factoring

- \( F = ac + ad + bc + bd + e \)
  - 4 2-input, 1 5-input \( \Rightarrow 8 \) 2-input gates
  - 9 literals
- \( F = (a+b)(c+d) + e \)
  - 4 2-input
  - 5 literals

Substitution

- \( G = a+b \)
- \( F = a+bc \)
- Substitute \( G \) into \( F \)
- \( F = G(a+c) \)
  - (verify) \( F = (a+b)(a+c) = aa + ab + ac + bc = a + bc \)
- 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 [next lecture]
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

Division

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

*E.g.*

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

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: \( h_i = \{ c_j \mid a_i \ast c_j \in f \} \)
- \( f/p = h_1 \cap h_2 \cap h_3 \ldots \)
Algebraic Division

- f and p are expressions (lists of cubes)
- p = {a₁, a₂, ...}
- \( h₁ = \{c_j | a₁ \cdot c_j \in f\} \)
- \( f/p = h₁ \cap h₂ \cap h₃ \ldots \)

Algebraic Division Example

- \( f = abc +abd +de, \ p = ab + e \)
- \( p = \{ab, e\} \)
- \( h₁ = \{c,d\} \)
- \( h₂ = \{d\} \)
- \( h₁ \cap h₂ = \{d\} \)
- \( f/p = d \)
- \( r = f - p \cdot (f/p) \)
- \( r = abc +abd +de-(ab+e)d \)
- \( r = abc \)

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 (terms==1) return(f)
  
  p=CHOOSE_DIVISOR(f)
  
  (h,r)=DIVIDE(f,p)
  
  f=Gfactor(h)*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**

- Find $c_f = \text{largest cube factor of } f$
  
  $K=\text{Kernel1}(0,f/c_f)$
  
  if (f is cube-free)
    
    return(f}$∈$K
  
  else
    
    return(K)

**Kernel Extract Example**

(ex. on board; adv to return to alg.)

- $f=abcd+abce+abef$

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

**Kernel Extract Example**

(stay on prev. slide, ex. on board)

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

**Kernel Extract Example**

(stay on prev. slide, ex. on board)

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

**Kernel Extract Example**

(stay on prev. slide, ex. on board)

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

Identify cube-free expressions in many functions
(common sub expressions)
1. Generate kernels for each function
2. Select pair such that $k_1 \cap k_2$ is not a cube
   - Note: $k_1 \cap k_2$ is simplest case of this
   - ...but intersection case is more powerful
   - Example to come
3. New variable from intersection
   - $v = k_1 \cap k_2$
4. Update functions (resubstitute)
   - $f_i = v \cdot (f_i / v) + r_i$
   - (similar for common cubes)

Extraction Example

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

- $L = d+e$
- $X = ab(cL+f+g)+h$
- $Y = ai(cL+f+j)+h$

- No kernels
- Common cube: $aM$

- Can collapse
  - $L$ into $M$ into $N$
  - Only used once
- Get larger common kernel $N$
  - Maybe useful if components becoming too small for efficient gate implementation
**Resubstitution**

- Also useful to try complement on new factors
- \( f = ab + ac + /b/cd \)
- \( X = b + c \)
- \( f = aX + /b/cd \)
- \( /X = /b/c \)
- …extracting complements not a direct target

**Multilevel Optimization**

- Unlike Two-level, very heuristic
- Not clear when done
- Goal find common terms to share
- Often start with two-level optimization
  - Identifies product term sharing
- Identify kernels and cubes
- Factor them out
- If can be used many places, get benefit
- SIS included common recipes
- More after timing analysis

---

**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

**Big Ideas**

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

---

**Admin**

- Reading for Monday on canvas
- Updated energy model: update_model.tar
- Milestone Thursday