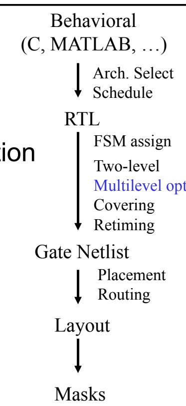
ESE535: Electronic Design Automation

Day 21: April 6, 2011
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 → 23 2-input gates
- vs. 3 gates: and4,or3,or2 → 6 2-input gates
- Aside from Pterm sharing between outputs,
  - two level cannot share sub-expressions

Why Multi-level Logic

- 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/c/d+a/b/c/d+abc/d+/a/bc/d

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/c/d+a/b/c/d+abc/d+/a/ bc/d

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 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
- 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

- F=XY+/X/Y
- X=ab
- Y=c+d

Decomposition

- F=abc+abd+/a/c/d+/b/c/d
  - 4 3-input + 1 4-input ➔ 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

➔ 8 2-input gates

- F=XY+e
- G=X/e
- H=Ye
- X=a+b
- Y=cd

- 2-input: 6

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 ➔ 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 [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

Division

- **Given**: function (f) and divisor (p)
- **Find**: quotient and remainder

  f=pq+r

E.g.

f=abc+abd+ef, p=ab

q=c+d, r=ef

Algebraic Division

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

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 (adv to alg.; work ex on board)

- f=abc+abd+de
- p=ab+e

Algebraic Division

- f and p are expressions (lists of cubes)
- $p=\{a_1,a_2,\ldots\}$
- $h_i=\{c_j \mid a_i * c_j \in f\}$
- $f/p = h_1 \cap h_2 \cap h_3 \ldots$

Algebraic Division Example

- f=abc+abd+de, p=ab+e

- p={ab,e}
- h1={c,d}
- h2={d}
- h1 ∩ h2={d}
- f/p=d

- r=f- p *(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

Must be to Generate Non-trivial kernel

- Find c_f = largest cube factor of f
- K=Kernel1(0,f/c_f)
- if (f is cube-free)
  - return(f∪K)
- else
  - return(K)

- 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≤i, $l_k \notin c_f$)
        - » R=R ∪ 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
(ex. on board; adv to return to alg.)

- f=abcd+abce+abef

Kernel Extraction

Must be to Generate Non-trivial kernel

- Find c_f = largest cube factor of f
- K=Kernel1(0,f/c_f)
- if (f is cube-free)
  - return(f∪K)
- else
  - return(K)

- 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≤i, $l_k \notin c_f$)
        - » R=R ∪ 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
(stay on prev. slide, ex. on board)

- 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
- Recurse→e+d
- R={cd+ce+ef,e+d}
- only 1 d
- (d+ce+ef)/e=c+f
- Recurse→c+f
- R={cd+ce+ef,e+d,c+f}

Extraction

Identify cube-free expressions in many functions (common sub expressions)

1. Generate kernels for each function
2. select pair such that k1∩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∩k2
4. update functions (resubstitute)
   - $f_i = v^*(f_i /v)+ r_i$
   - (similar for common cubes)

Extraction Example

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

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

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

Extraction Example

- N=aM
- M=cL+f
- L=d+e
- X=b(N+ag)+h
- Y=I(N+aj)+k

- 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
- 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