ECE 697B (667)

Spring 2003

Synthesis and Verification of Digital Systems

Functional Decomposition

Slides adopted (with permission) from A. Mishchenko, 2003

Overview

- The concept of functional decomposition
- Two uses of BDDs for decomposition
  - as a computation engine to implement algorithms
  - as a representation that helps finding decompositions
- Two ways to direct decomposition using BDDs
  - bound set on top (Lai/Pedram/Vardhula, DAC'93)
  - free set on top (Stanion/Sechen, DAC'95)
  - other approaches
- Disjoint and non-disjoint decomposition
- Applications of functional decomposition:
  - Multi-level FPGA synthesis
  - Finite state machine design
  - Machine learning and data mining

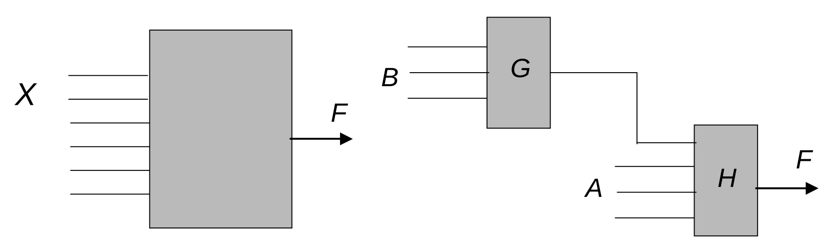
Functional Decomposition – previous work

- Ashenhurst [1959], Curtis [1962]
  - Tabular method based on cut: bound/free variables
  - BDD implementation:
    - Lai *et al*. [1993, 1996], Chang *et al*. [1996]
    - Stanion *et al*. [1995]
- Roth, Karp [1962]
  - Similar to Ashenhurst, but using cubes, covers
  - Also used by SIS
- Factorization based
  - SIS, algebraic factorization using cube notation
  - Bertacco *et al*. [1997], BDD-based recursive bidecomp.

Two-Level Curtis Decomposition

$F(X) = H(\,G(B), A\,),\;\; X = B \cup A$

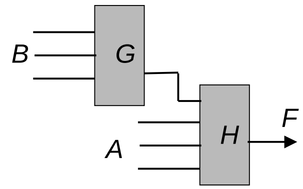
B = bound set   *A*= free set



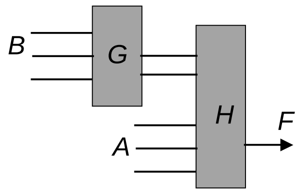
if $B \cap A = \varnothing$, this is *disjoint* decomposition

if $B \cap A \neq \varnothing$, this is *non-disjoint* decomposition

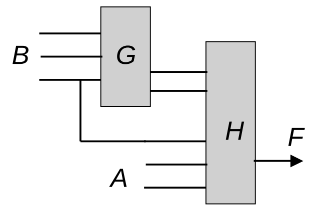
Decomposition Types



Simple disjoint decomposition
(Asenhurst)



Disjoint decomposition
(Curtis)



Non-disjoint decomposition

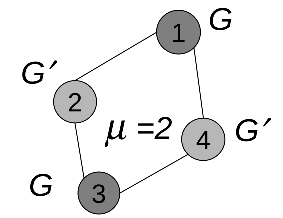
Decomposition Chart

Bound Set = *{a,b}*

Free Set = *{c,d}*

	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	1	0	1
10	0	0	0	0
	1	**2**	**3**	**4**

Incompatibility Graph



Definition 1: *Column Compatibility*

Two columns *i* and *j* are *compatible* if each element in *i* is equal to the corresponding element in j or the element in either *i* or *j* is not specified

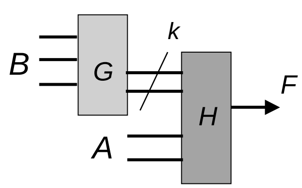
Definition 2: *Column Multiplicity* μ = the number of compatible sets (distinct column patterns)

Fundamental Decomposition Theorems

- Theorem (Asenhurst)
  Let *k* be the minimum number of compatible sets in the decomposition chart. Then function *H* must distinguish at least *k* values

- Theorem (Curtis)
  Let μ *(A | B)* denote column multiplicity under decomposition into bound set *B* and free set *A*. Then:

$$\mu(A \mid B) \le 2^k \Leftrightarrow F(B,A) = H(G_1(B), G_2(B), \ldots, G_k(B), A)$$



Asenhurst-Curtis Decomposition

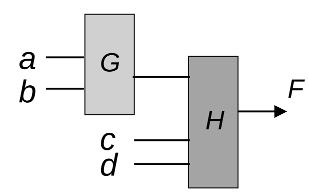
Bound Set = *{a,b}*

Free Set = *{c,d}*

	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	1	0	1
10	0	0	0	0
	1	**2**	**3**	**4**

Here $\mu = 2$, so function *H* must distinguish two values

- need 2 bits to encode inputs from *G*



$$F(a,b,c,d) = (a'b' + ab)c' + (a'b + ab')(cd + c'd')$$

$$G(a,b) = a'b' + ab \qquad H(G,c,d) = Gc' + G'(cd + c'd')$$

Multi-Level Curtis Decomposition

- Two-level decomposition is iteratively applied to new functions H_i and G_i, until smaller functions G_t and H_t are created, that are not further decomposable.
- One of the possible cost functions is *Decomposed Function Cardinality (DFC)*. It is the total cost of all blocks, where the cost of a binary block with n inputs and m outputs is $m * 2^n$.

Typical Decomposition Algorithm

- Find a set of partitions (B_i, A_i) of input variables X into bound set variables B_i and free set variables A_i
- For each partition, find decomposition

  $$F(X) = H_i(G_i(B_i), A_i)$$

  such that the column multiplicity is *minimal*, and compute DFC
- Repeat the process for all partitions until the decomposition with minimum DFC is found.

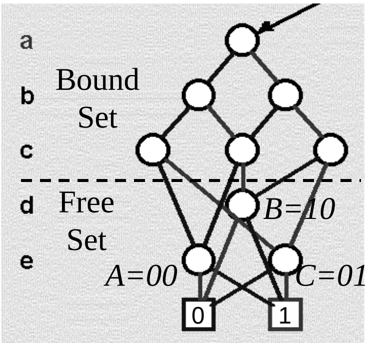
Uses of BDDs for Decomposition

- Whatever is the decomposition algorithm, BDDs can be used to store data and perform computation (using cubes, partitions, etc.)
- Alternatively, the algorithm may exploit the BDD structure of the function F to direct the decomposition in the bound set selection, column multiplicity computation, and deriving the decomposed functions G and H

BDD-Based Decomposition

- Bound set on top (Lai/Pedram/Vardhula, DAC'93)
- Free set on top (Stanion/Sechen, DAC'95)
- Bi-decomposition using 1-, 0-, and EXOR-dominators (Yang/Ciesielski, ICDD'99)
- Recursive decomposition (Bertacco/Damiani,ICCAD'97)
- Implicit decomposition (Wurth/Eckl/Legl,DAC'95)

Bound Set on Top (Function G)



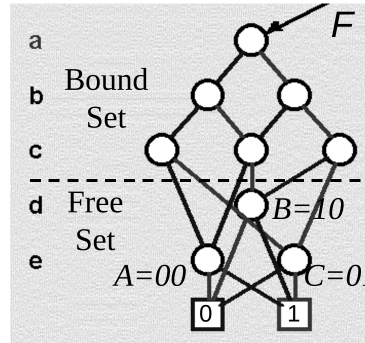
a	0	0	0	0	1	1	1	1
b	0	0	1	1	1	1	0	0
c	0	1	1	0	0	1	1	0
d e: 00	1	0	1	1	1	0	1	1
01	0	1	1	0	1	1	1	0
11	0	1	0	0	0	1	0	0
10	1	0	0	1	0	0	0	1
	A	C	B	A	B	C	B	A

= 0
= 1

$G=\{g_0,g_1\}$, $A=g_0'g_1'$, $B=g_0g_1'$, $C=g_0'g_1$

$g_0=a'bc+ab'c+abc'$, $g_1 = a'b'c+ abc$

Bound Set on Top (Function H)



a	0	0	0	0	1	1	1	1
b	0	0	1	1	1	1	0	0
c	0	1	1	0	0	1	1	0
d e: 00	1	0	1	1	1	0	1	1
01	0	1	1	0	1	1	1	0
11	0	1	0	0	0	1	0	0
10	1	0	0	1	0	0	0	1
	A	C	B	A	B	C	B	A

$F(a,b,c,d,e) = H(g_1(a,b,c), g_2(a,b,c), d, e)$

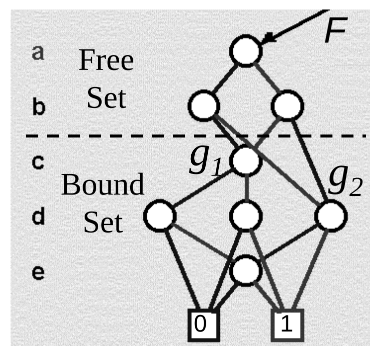
$H=g_0'g_1'e' + g_0g_1'd' + g_0'g_1e$

= 0
= 1

"Bound Set on Top" Algorithm

- Reorder variables in BDD for *F* and check column multiplicity for each bound set
- For the bound set with the smallest column multiplicity, perform decomposition :
  - Encode the *cut nodes* with minimum number of bits (*log* μ)
  - derive functions *G* and *H* (both depend on encoding)
- Iteratively repeat the process for functions *G* and *H* (typically, only *H*)
- *This algorithm can be modified to work for non-disjoint decompositions but does not work with DCs*

Free Set on Top (Function G)

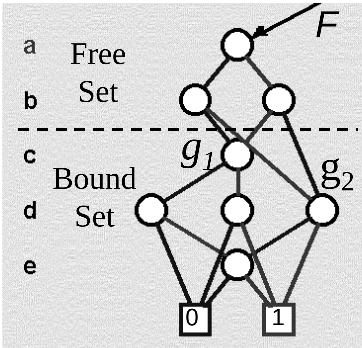


a	0	0	0	0	1	1	1	1
b	0	0	1	1	1	1	0	0
c	0	1	1	0	0	1	1	0
d e: 00	0	0	0	0	0	0	0	0
01	0	0	1	1	0	0	1	1
11	1	1	1	1	1	1	1	1
10	0	1	1	1	0	1	1	1

= 0
= 1

$G=\{g_1,g_2\}$, $g_1=c'de+cd$, $g_2=d+e$

Free Set on Top (Function *H*)



d e \ a b c	000	001	011	010	110	111	101	100
00	0	0	0	0	0	0	0	0
01	0	0	1	1	0	0	1	1
11	1	1	1	1	1	1	1	1
10	0	1	1	1	0	1	1	1

$$F(a,b,c,d,e) = H(\ a,\ b,\ g_1(c,d,e),\ g_2(c,d,e)\ )$$

$$H=(a'b'+ ab)\ g_1 + (a'b+ ab')\ g_2$$

= 0
= 1

"Free Set on Top" Algorithm

- Find good variable order
- Derive implicit representation of all feasible cuts on the BDD representing *F*
- Use some cost function to find the best bound set and perform decomposition
- Repeat the process for functions *G* and *H*
- This algorithms is faster than "*bound set on top*" but it does not work for non-disjoint decompositions and incompletely specified functins (with DCs).

Non-Disjoint Decomposition

- *Non-disjoint* decomposition can be reduced to *disjoint* decomposition by adding variables
- Bound Set = *{a,b,c}*, Free Set = *{c,d}*
  Disjoint decomposition can be generated by introducing variables $c_1=c_2=c$ instead of *c*
- In terms of the Karnaugh map, it is equivalent to introducing two variables instead of one in such a way that $c_1c_2'+c_1'c_2$ is a *don't care set*.

Why: $c_1 \equiv c_2 \ \Rightarrow c_1c_2'+c_1'c_2$

Non-Disjoint Decomposition Example

c d \ a b	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	1	1	1
10	1	0	0	0
	A	B	C	B

c2 d \ c1 a b	000	001	011	010	110	111	101	100
00	1	1	0	1	-	-	-	-
01	0	0	0	0	-	-	-	-
11	-	-	-	-	1	1	1	0
10	-	-	-	-	0	0	0	1
	A	A	B	A	B	B	B	A

There is no disjoint decomposition with any bound set; there is non-disjoint decomposition with bound set *{a,b,c}*