Το εργαλείο λογικής σύνθεσης SIS

Ελεύθερο προς κατέβασμα από :

http://embedded.eecs.berkeley.edu/Alumni/pchong/sis.html

Βασική Δημοσίευση:

SIS: A System for Sequential Circuit Synthesis (1992)(357 citations)
Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R. Stephan, Robert K. Brayton, et al.

http://citeseer.ist.psu.edu/sentovich92sis.html

1

Προεπισκόπηση

- Μορφή αρχείων εισόδου και εξόδου
- Βασικές Εντολές
- Scripts για βελτιστοποίηση κυκλωμάτων
- Προσθήκη νέων εντολών

2

Μορφή αρχείων εισόδου και εξόδου

- Equation format (.eqn)
  - F = a*b*(c+d')
    - + λογικό Η
    - * λογικό ΚΑΙ
    - ` αντιστροφή
    - != αποκλειστικό Η
    - == έλεγχος ισότητας
    - !() αντιστροφή των στοιχείων εντός παρένθεσης
    - Με κεφαλαία γράμματα σημειώνουμε τις εξόδους
    - Με μικρά γράμματα τις εισόδους

3

Μορφή αρχείων εισόδου κι εξόδου

- Berkeley Logic Interchange format (.blif)
  - Αναλυτική περιγραφή : http://www.bddportal.org/docu/blif/blif.html
    - Υποστηρίζει:
      - Κυβικές μορφές εξισώσεων(models)
      - Ορισμό πυλών(gates)
      - Ορισμό εξωτερικών συνθηκών αδιαφορίας(exdc)
      - Χρήση Flip – Flops, Latches
      - Αναφορά σε πύλες της βιβλιοθήκης τεχνολογίας
      - Ορισμό Μηχανών Πεπερασμένων Καταστάσεων
      - Χρήση περιορισμών για καθυστερήσεις

4

Μορφή αρχείων εισόδου κι εξόδου

- Αρχείο βιβλιοθήκης τεχνολογίας
  - GATE <cell-name> <cell-area> <cell-logic-function>
    <pin-info>
    ..
    <pin-info>
    - PIN <pin-name> <phase> <input-load> <max-load>
      <rise-block-delay> <rise-fanout-delay>
      <fall-block-delay> <fall-fanout-delay
  - Παράδειγμα πύλης
    GATE inv1 1 O=!a;
    PIN * INV 1 999 0.9 0.3 0.9 0.3

5

Βασικές εντολές

- read_eqn <filename>
  - Διαβάζει ένα αρχείο εισόδου μορφής Εξισώσεων.
- read_blif <filename>
  - Διαβάζει ένα αρχείο εισόδου μορφής BLIF.
- write_eqn <filename>
  - Δημιουργεί ένα νέο αρχείο και γράφει την τρέχουσα κατάσταση του δικτύου σε μορφή Εξισώσεων.
- write_blif <filename>
  - Δημιουργεί ένα νέο αρχείο και γράφει την τρέχουσα κατάσταση του δικτύου σε μορφή BLIF.

6

Βασικές εντολές

- print
  - Τυπώνει την τρέχουσα κατάσταση του κυκλώματος.
  - Παράδειγμα

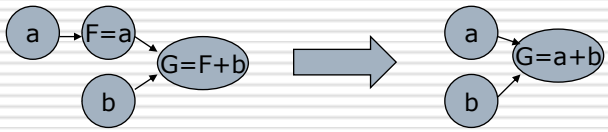
```
sis>read_eqn myFile.eqn
sis>print
F = G + c
G = a + b
```

myFile.eqn :
F = a + b + c
G = a + b

7

Βασικές εντολές

- sweep
  - Αφαιρεί όλους τους κόμβους του δικτύου που
    - έχουν μονάχα μία είσοδο
    - έχουν σταθερή τιμή 1 ή 0
  - Παράδειγμα



8

Βασικές εντολές

- eliminate <όριο>
  - Αφαιρεί από το δίκτυο όλους τους κόμβους των οποίων η ΤΙΜΗ είναι μικρότερη απο το <όριο>
    - ΤΙΜΗ ενός κόμβου είναι το fanout του.(Για την ακρίβεια είναι ο αριθμός των φορών που χρησιμοποιείται στην παραγοντοποιημένη μορφή των κόμβων-fanouts του)
  - Παράδειγμα

```
sis>read_eqn myFile.eqn
sis>eliminate 1
sis>print
{G} = a + b
F = G + c
H = G + d
sis>eliminate 3
sis>print
F = a + b + c
H = a + b + d
```

myFile.eqn :
F = a + b + c
H = a + b + d

9

Βασικές εντολές

- simplify
  - Σε κάθε κόμβο του δικτύου εφαρμόζεται ο αλγόριθμος ESPRESSO
    - -no_comp Δεν υπολογίζεται ολόκληρο το OFFSET του κάθε κόμβου. Χρησιμοποιείται για γρηγορότερα αποτελέσματα.
- full_simplify
  - Ομοίως με την εντολή simplify αλλά χρησιμοποιώντας μεγαλύτερο σύνολο συνθηκών αδιαφορίας.
- Παράδειγμα

```
sis>read_eqn myFile.eqn
sis>print
F = a + a'b + c
sis>simplify
sis>print
F = a + b + c
```

myFile.eqn :
F = a + a'*b + c

10

Βασικές εντολές

- resub
  - Ελέγχει ανά ζεύγη όλους τους κόμβους τους δικτύου για την περίπτωση που ο πρώτος ή το συμπλήρωμα του μπορεί να αντικαταστήσει τμήμα του δεύτερου.
  - -a χρησιμοποιείται αγεβρική διαίρεση. Διαιρείται ο πρώτος από τον δεύτερο και στην περίπτωση που έχουμε μείωση συνολικού εμβαδού γίνεται η αντικατάσταση.
  - Παράδειγμα

```
sis>read_blif myFile.eqn
sis>resub –a
sis>print
F = a * b
G = F + c
H = F + e
```

myFile.eqn :
F = a * b
G = a * b + c
H = a * b + e

11

Βασικές εντολές

- gcx
  - Αναζητά τους καλύτερους κύβους – αλγεβρικούς διαιρέτες του δικτύου και δημιουργεί επιπλέον κόμβους για αυτούς.
  - -b : βρίσκει τον καλύτερο(περισσότερο κοινό) κύβο σε κάθε βήμα του αλγορίθμου.
  - -t : ένας μέγιστος αριθμός ο οποίος όταν ξεπεραστεί σταματάνε οι επαναλήψεις του αλγορίθμου.
  - Παράδειγμα

```
sis> read_eqn myFile.eqn
sis> print
f = a b c e + d
{g} = a b c + f
sis> gcx –bt 10
sis> print
f = [1] e + d
{g} = [1] + f
[1] = a b c
```

myFile.eqn
f = a*b*c*e + d;
g = a*b*c + f;

12

Βασικές εντολές

- gkx
  - Όπως και η gcx μόνο που αναζητούνται οι κοινοί διαιρέτες με περισσότερους του ενός κύβους.
  - Επιπλέον παράμετροι:
    - -a : αναζητούνται όλοι οι δυνατοί πυρήνες επιπέδου 0.
    - -l : ο αλγόριθμος περνάει μια μόνο φορά από το δίκτυο. Διαφορά ο αλγόριθμος επαναλαμβάνεται και στους νέους κόμβους.
  - Παράδειγμα

```
sis> read_eqn myFile.eqn
sis> print
{f} = a b c e + a b d
{g} = a c e + a d
sis> gkx
sis> print
{f} = [2] a b
{g} = [2] a
[2] = c e + d
```

myFile.eqn :
f = a*b*c*e + a*b*d;
g = a*c*e+a*d;

13

Βασικές Εντολές

- decomp
  - Για κάθε κόμβο του δικτύου :
    - Αν είναι παραγοντοποιημένος δημιουργεί καινούριους κόμβους για διαιρεταίο, διαιρέτη και υπόλοιπο.
    - Αν δεν είναι παραγοντοποιημένος πρώτα τον παραγοντοποιεί χρησιμοποιώντας τον πυρήνα που θα βρει.
  - -g : βρίσκει τον καλύτερο δυνατό πυρήνα για να αποσυνθέσει τον κάθε κόμβο του δικτύου.
  - -q : χρησιμοποιεί τον πρώτο πυρήνα που θα υπολογίσει για να αποσυνθέσει τον κάθε κόμβο του δικτύου.
  - Παράδειγμα

```
sis> read_eqn myFile.eqn
sis> factor -g *
sis> print
{f} = a b c e + a b d
{g} = a c e + a d
sis> decomp
sis> print
{f} = [3] a b
{g} = [3] a
[3] = c e + d
```

myFile.eqn :
F = a + b*c + d
G = a*d + b*c*d + e

14

Βασικές εντολές

- fx
  - Δημιουργεί νέους ενδιάμεσους κόμβους από τμήματα(ένα ή δύο κύβους) κόμβων. Θα πρέπει να ακολουθείται από την εντολή resub η οποία θα κρατήσει τους νέους κόμβους που μειώνουν το εμβαδόν και θα αφαιρέσει όσους δε χρειάζονται.
  - Παράδειγμα

```
sis>read_blif myFile.blif
sis>fx
sis>print
[1] = a + bc
F = [1] + d
G = [1]*d + e
```

myFile.eqn :
F = a + b*c + d
G = a*d + b*c*d + e

15

Scripts για την βελτιστοποίηση κυκλωμάτων

- script.rugged

```
sweep; eliminate -1;
simplify –m nocomp;
eliminate -1;

sweep; eliminate 5;
simplify –m nocomp;
resub –a

fx; resub –a; sweep;

eliminate -1; sweep;
full_simplify –m nocomp;
```

Στο αρχικό πέρασμα από το δίκτυο αφαιρούνται όσοι κόμβοι δεν είναι απαραίτητοι και γίνονται κάποιες πρώτες απλοποιήσεις με τον ESPRESSO

Αφαιρούμε κόμβους και απλοποιούμε πάλι με τον ESPRESSO αλλά κάνουμε και αλγεβρική παραγοντοποίηση με τη resub χρησιμοποιώντας τους υπάρχοντες κόμβους

Επεκτείνουμε το δίκτυο μας με την fx δημιουργώντας νέους ενδιάμεσους κόμβους, εφαρμόζουμε αλγεβρική διαίρεση σε όλους τους κόμβους και κρατάμε όσους είναι απαραίτητοι

Σαν τελικό βήμα περνάμε από όλους τους κόμβους και εφαρμόζουμε τη καλύτερη δυνατή βελτιστοποίηση ESPRESSO

16

Scripts για τη βελτιστοποίηση κυκλωμάτων

```
Script.algebraic
sweep
eliminate 5
simplify -m nocomp -d
resub -a

gkx -abt 30
resub -a; sweep
gcx -bt 30
resub -a; sweep

gkx -abt 10
resub -a; sweep
gcx -bt 10
resub -a; sweep

gkx –ab
resub -a; sweep
gcx -b
resub -a; sweep

eliminate 0
decomp -g *
```

Αρχικά απαλείφονται οι ενδιάμεσοι κόμβοι που προσθέτουν εμβαδο στο κύκλωμα και απλοποιούνται με τον ESPRESSO όλοι οι κόμβοι.

Έπειτα για τρεις φορές
1. υπολογίζεται ο καλύτερος πυρήνας
2. δημιουργούνται νέοι κόμβοι για τα αποτελέσματα της διαίρεσης
3. υπολογίζεται ο καλύτερος κύβος
4. διαιρούνται όσοι κόμβοι είναι δυνατόν με τον κύβο

Στο τέλος όλοι οι κόμβοι διαιρούνται με τον καλύτερο πυρήνα που είναι δυνατό να διαιρεθούν

17

Προσθήκη νέων εντολών

- Δίνεται η δυνατότητα να μπορεί κάποιος να προσθέτει νέες εντολές οι οποίες μπορούν να χρησιμοποιούν τις υπάρχουσες δομές (π.χ struct network node)
- Είναι διαθέσιμο πλήθος επικουρικών συναρτήσεων για τις συνηθέστερες πράξεις στη λογική σύνθεση.(π.χ struct network node *cofactor(struct network node *);
- Υπάρχει γενικά ένα μικρό πρόβλημα στη συμβατότητα του εργαλείου με τις τελευταίες διανομές του Linux αλλά από την έκδοση 1.3.6(Ανεπίσιμη) κι έπειτα δεν υπάρχει πρόβλημα.

18