Ανάλυση και Σύνθεση Πολύπλοκων Ηλεκτρονικών Συστημάτων

Εργασία αναφορικά με την ανίχνευση ακμών
Edge-Detection

Καραγιάννης Καρούσος Δημήτρης, 57412
Κεφαλάς Γεώργιος, 57406

Περιεχόμενα

1. Εισαγωγή

Η παρακάτω εργασία υλοποιήθηκε στα πλαίσια του μαθήματος “Ανάλυση και Σύνθεση Πολύπλοκων Ηλεκτρονικών Συστημάτων” του 8ου εξαμήνου από τους προπτυχιακούς φοιτητές Καραγιάννη-Καρούσο Δημήτρη και Κεφαλά Γεώργιο. Αφορά την ανάπτυξη αλγορίθμου για επεξεργασία εικόνας με SystemVerilog και πιο συγκεκριμένα για την ανίχνευση ακμών μιας εικόνας (edge-detection). Κατά τη μελέτη έγινε ανάπτυξη κώδικα που υλοποιεί τον αντίστοιχο αλγόριθμο καθώς επίσης και περιβάλλον στο οποίο ελέγχεται η λειτουργία του κώδικα. Σε αυτό το έγγραφο γίνεται η περιγραφή της εν λόγω μελέτης η οποία πραγματοποιήθηκε, η περιγραφή του αλγορίθμου, του κώδικα που αναπτύχθηκε και του περιβάλλοντος όπου έγινε ο έλεγχος του κώδικα ώστε να λειτουργεί σωστά καθώς επίσης και αναφορά σε προβλήματα που αντιμετωπίστηκαν και σημεία για τα οποία υπάρχουν προοπτικές για περαιτέρω βελτίωση.

2. Περιγραφή αλγορίθμου

Ο αλγόριθμος που χρησιμοποιήθηκε περιγράφεται αναλυτικά στην βιβλιογραφία. Η βασική ιδέα που ακολουθείται είναι το γεγονός ότι εφόσον μία ακμή αποτελείται από μια σειρά συνεχόμενων μη κενών pixel (pixel με τιμή 1), τότε μπορεί να μπορεί να εξαχθεί συμπέρασμα για το αν ένα συγκεκριμένο pixel είναι ακμή αναλόγως των τιμών που έχουν τα pixel στην γειτονιά του. Ο αλγόριθμος υλοποιείται σε δύο βήματα. Στο πρώτο βήμα ελέγχεται αν υπάρχουν τρία συναφή pixel, στην γειτονιά Moore του υπό μελέτη pixel, τα οποία έχουν την τιμή 1. Έπειτα ελέγχεται αν τα συμμετρικά τους, ως προς το υπό μελέτη pixel, έχουν την τιμή 0. Αν ισχύουν οι παραπάνω προϋποθέσεις το pixel διατηρεί την τιμή του, αλλιώς του εκχωρείται η τιμή 0. Η διαδικασία αυτή επιτυγχάνει μια βασική εκτίμηση για την θέση των ακμών στην εικόνα, αλλά οι ακμές που προκύπτουν είναι μη ομαλές, “θορυβώδεις”. Την “λείανση” αυτών των ακμών για μια πιο καθαρή εικόνα που αποτυπώνει τις ακμές επιτυγχάνει το δεύτερο βήμα. Αυτό το βήμα εφαρμόζεται στην εικόνα που προέκυψε από το πρώτο βήμα και όχι στην αρχική. Εάν στη γειτονιά Von Neumann του υπό μελέτη pixel υπάρχουν ακριβώς δύο γείτονες με την τιμή 1 και όλα τα υπόλοιπα pixel της γειτονιάς Moore εχουν την τιμή 0, τότε το pixel διατηρεί την τιμή του, αλλιώς του εκχωρείται η τιμή 0. Αυτή η διαδικασία αφαιρεί pixel τα οποία μπορεί να είναι κατάλοιπα του εσωτερικού των σχημάτων που αποτυπώνονται στην αρχική εικόνα, όπως για παράδειγμα τα εσωτερικά pixel σε έναν μαύρο κύκλο. Το πρώτο βήμα εφαρμόζεται μία φορά στην αρχική εικόνα, ενώ το δεύτερο μπορεί να εφαρμοστεί επανειλημμένα, έως ότου επιτευχθεί η επιθυμητή ευκρίνεια στο περίγραμμα της εικόνας.

3. Υλοποίηση κώδικα

❑ Πρώτο βήμα του αλγορίθμου

Σύμφωνα με τον αλγόριθμο που αναλύθηκε και κατά το προηγούμενο κεφάλαιο στο πρώτο βήμα με λίγα λόγια γίνεται έλεγχος της γειτονιάς Moore έως ότου εντοπιστούν τρία pixel στη σειρά που έχουν την τιμή 1 καθώς επίσης και τα αντιδιαμετρικά τους και αντίστοιχα διατηρείται η όχι η τιμή του pixel. Στο πρώτο αρχείο που υλοποιήθηκε*(edge_detection_first_step.sv)* έγινε αυτή η διεργασία σύμφωνα με την οποία δίνονται σαν είσοδοι το κεντρικό μας pixel μαζί με την γειτονιά Moore του και επιστρέφεται η νέα του τιμή είτε όταν αυτή διατηρείται είτε όταν μηδενίζεται. Παρακάτω μπορείτε να δείτε και τις αντίστοιχες εισόδους που δηλώνονται στην αρχή του κώδικα μας.

```
1   module edge_detection_first_step(
2       input logic clk,
3       input logic rst,
4
5       input logic neighbors_state[8:0],
6       // |0  1  2|
7       // |7  8  3|
8       // |6  5  4|
9
10      output logic state
11  );
12
```

Αρχικοποίηση εισόδων και εξόδων του module σε System Verilog

Στη συνέχεια ξεκινάει η υλοποίηση του αλγορίθμου για το πρώτο βήμα η οποία πραγματοποιήθηκε με συνδυαστική λογική. Αρχικά, δηλώνονται όλες οι λογικές μεταβλητές που θα χρειαστούν και έπειτα παίρνονται όλες οι περιπτώσεις κατά τις οποίες 3 συνεχόμενα pixel της γειτονιάς Moore είναι ‘1’. Αυτό υλοποιείται με πύλες AND για κάθε μια από τις περιπτώσεις που μπορούμε να έχουμε. Να σημειωθεί ότι οι γειτονιά δηλώθηκε με την μορφή πίνακα 9 λογικών τιμών ενός bit όπου το όγδοο κελί του πίνακα αντιστοιχεί στο κεντρικό μας pixel και τα υπολοιπα από το 0 μέχρι το 7 για τα γειτονικά pixel.

```
13  //First Step
14  //Find the 3 contiguous neighbors
15
16  logic line1,line2; // line1 = 0 1 2, line2 = 4 5 6
17  logic col1,col2; //col1 = 0 7 6, col2 = 2 3 4
18  logic corner1,corner2,corner3,corner4; // corner1 = 7 0 1,corner2 = 1 2 3, corner3 = 3 4 5,corner4 = 5 6 7
19
20  assign line1 = (neighbors_state[0] & neighbors_state[1] & neighbors_state[2]);
21  assign line2 = (neighbors_state[4] & neighbors_state[5] & neighbors_state[6]);
22
23  assign col1 = (neighbors_state[0] & neighbors_state[7] & neighbors_state[6]);
24  assign col2 = (neighbors_state[2] & neighbors_state[3] & neighbors_state[4]);
25
26  assign corner1 = (neighbors_state[7] & neighbors_state[0] & neighbors_state[1]);
27  assign corner2 = (neighbors_state[1] & neighbors_state[2] & neighbors_state[3]);
28  assign corner3 = (neighbors_state[3] & neighbors_state[4] & neighbors_state[5]);
29  assign corner4 = (neighbors_state[5] & neighbors_state[6] & neighbors_state[7]);
30
```

Πύλες AND για κάθε μια από τις περιπτώσεις όπου έχουμε 3 άσσους στη γειτονιά Moore

Στη συνέχεια, θέλοντας να εξετάσουμε τα αντιδιαμετρικά pixel της γειτονιάς χρησιμοποιούμε πύλες NOR αφού πρώτα έχουμε δηλώσεις τις αντίστοιχες λογικές μεταβλητές ώστε να γνωρίζουμε το ποια θα πρέπει να είναι η τελική τιμή που πρέπει να πάρει το κεντρικό μας pixel μετά το πρώτο στάδιο. Παρακάτω μπορείτε να δείτε και την υλοποίηση των πυλών NOR.

```
31  logic nor_line1,nor_line2;
32  logic nor_col1,nor_col2;
33  logic nor_corner1,nor_corner2,nor_corner3,nor_corner4;
34
35  assign nor_line1 = ~(neighbors_state[0] | neighbors_state[1] | neighbors_state[2]);
36  assign nor_line2 = ~(neighbors_state[4] | neighbors_state[5] | neighbors_state[6]);
37
38  assign nor_col1 = ~(neighbors_state[0] | neighbors_state[7] | neighbors_state[6]);
39  assign nor_col2 = ~(neighbors_state[2] | neighbors_state[3] | neighbors_state[4]);
40
41  assign nor_corner1 = ~(neighbors_state[7] | neighbors_state[0] | neighbors_state[1]);
42  assign nor_corner2 = ~(neighbors_state[1] | neighbors_state[2] | neighbors_state[3]);
43  assign nor_corner3 = ~(neighbors_state[3] | neighbors_state[4] | neighbors_state[5]);
44  assign nor_corner4 = ~(neighbors_state[5] | neighbors_state[6] | neighbors_state[7]);
45
```

Πύλες NOR για τις τιμές των αντιδιαμετρικών pixel της γειτονιάς Moore

Το τελευταίο μέρος του πρωτου βηματος αφορά τον τελικό υπολογισμό της επόμενης κατάστασης που πρέπει να έχει το κεντρικό pixel μας με τη χρήση μια λογικής συνθήκης που ελέγχει όλες τις περιπτώσεις ώστε να διατηρεί ή όχι την αντίστοιχη τιμή.

```
45
46  //If the symetrical contiguous neighbors are 0 then the state stays the same else it is zero.
47  assign state = neighbors_state[8] & ((line1 & nor_line2) | (line2 & nor_line1) | (col1 & nor_col2) | (col2 & nor_col1) | ...
48                                       ...  (corner1 & nor_corner3) | (corner3 & nor_corner1) | (corner2 & nor_corner4) | (corner4 & nor_corner2));
49
50  endmodule;
```

Τελική τιμή της επόμενης κατάστασης

Έτσι τελικά υλοποιείται το πρώτο μέρος του αλγορίθμου μας για να συνεχίσουμε με το επόμενο.

❑ Δεύτερο βήμα του αλγορίθμου

Στη συνέχεια θα ασχοληθούμε με την περιγραφή της υλοποίησης του δεύτερου βήματος όπως περιγράφηκε και προηγουμένως*(edge_detection_second_step.sv)*. Αυτό το βήμα αφορά την γειτονια Von Neumann στην οποία ψάχνουμε ακριβώς 2 άσσους από τις τέσσερις τιμές που έχει ενώ θα θέλαμε όλες οι άλλες περιοχές να έχουν την τιμή ‘0’. Για να το υλοποιήσουμε αυτό χρησιμοποιούμε μια συνάρτηση (module) με τις ίδιες εισόδους και εξόδους με το προηγούμενο βήμα. Στη συνέχεια, εξετάζουμε με τη χρήση τριών λογικών πυλών την ύπαρξη ακριβώς 2 άσσων. Αυτό επιτυγχάνεται αρχικά με μια πύλη XOR τεσσάρων εισόδων η οποία μας επιστρέφει την τιμή ‘0’ όταν οι είσοδοι έχουν είτε κανέναν άσσο, είτε 2 είτε 4. Γι αυτόν ακριβώς το λόγο γίνεται χρήση ακόμα δύο πυλών, πιο συγκεκριμένα μιας AND για την περίπτωση των τεσσάρων άσσων και μιας πύλης OR για την περίπτωση των τεσσάρων μηδενικών, οι οποίες χρησιμοποιούνται για να μας βοηθήσουν στο να εντοπίσουμε μόνο την περίπτωση των 2 άσσων που χρειαζόμαστε. Παρακάτω μπορείτε να δείτε την υλοποίηση της συνδυαστικής λογικής του κώδικά μας.

```
1   module edge_detection_second_step(
2       input logic clk,
3       input logic rst,
4
5       input logic neighbors_state[8:0],
6       // |0  1  2|
7       // |7  8  3|
8       // |6  5  4|
9
10      output logic state
11  );
12
13  //Second Step
14
15  //count exactly 2 ones in Von Neuman neighborhood.
16  logic xor_neuman;
17  assign xor_neuman = neighbors_state[1] ^ neighbors_state[3] ^ neighbors_state[5] ^ neighbors_state[7];
18  logic and_neuman;
19  assign and_neuman = neighbors_state[1] & neighbors_state[3] & neighbors_state[5] & neighbors_state[7];
20  logic or_neuman;
21  assign or_neuman = neighbors_state[1] | neighbors_state[3] | neighbors_state[5] | neighbors_state[7];
22
```

Υλοποίηση συνδυαστικής λογικής για ανίχνευση δύο ‘1’ στην γειτονιά Von Neumann

Έπειτα, γίνεται ακόμα μια χρήση συνδυαστικής λογικής η οποία έχει ως σκοπό τον έλεγχο των υπόλοιπων γειτονικών pixel για τα οποία πρέπει να έχουμε τέσσερα ‘0’. Αυτό γίνεται επίσης με μια πύλη NOR με 4 εισόδους, μια για κάθε pixel. Τέλος έχοντας ολοκληρώσει το κομμάτι της συνδυαστικής λογικής που χρειαζόμασταν, γίνεται χρήση μιας λογικής συνθήκης κατά την οποία όταν το κεντρικό pixel μας είναι ‘1’ και στην γειτονιά Von Neumann έχουμε ακριβώς δύο ‘1’ (με βάση τις λογικές τιμές που ελέγχουμε) και δεν υπάρχουν άλλοι ‘1’ στην γειτονιά Moore, τότε το κεντρικό pixel διατηρείται ως ακμή διαφορετικά παίρνει την τιμή μηδέν όπως φαίνεται και στο κομμάτι του κώδικα παρακάτω.

```
23  //check if all other Moore Neighborhood's pixels are zero.
24  logic moore_eq_zero;
25  assign moore_eq_zero = ~(neighbors_state[0] | neighbors_state[2] | neighbors_state[4] | neighbors_state[6]);
26
27  //if both of the previous conditions are true then the cell is edge.
28  assign state = (neighbors_state[8] && (~xor_neuman && ~and_neuman && or_neuman && moore_eq_zero)) ? 1 : 0;
```

Έλεγχος μηδενικών στη γειτονιά Moore και υπολογισμός της τελικής τιμής της εξόδου μας βάση του αλγορίθμου.

Έτσι ολοκληρώνεται και το 2ο και τελευταίο μέρος του αλγορίθμου για την ανίχνευση ακμών το οποίο είναι σημαντικό να αναφερθεί πως επαναλαμβάνεται αρκετές φορές ώστε να επιτευχθεί μεγαλύτερη ακρίβεια στο τελικό αποτέλεσμα ανίχνευσης όπως θα εξεταστεί στο επόμενο κεφάλαιο.

4. Μοντελοποίηση αλγορίθμου

❑ Verification κώδικα

Θέλοντας να εξετάσουμε το πόσο αξιόπιστος είναι ο κώδικας των δύο αρχείων που δημιουργήσαμε, κάναμε ένα test bench κατά το οποίο δίνοντας διάφορες τιμές για τις περιπτώσεις που έχουμε κατά τα 2 αυτά στάδια εξετάζοντας τις εξόδους ώστε να συνάδουν με τις επιθυμητές. Αυτό το πετύχαμε με ακόμα ένα αρχείο που δημιουργήσαμε με το όνομα “edge_detection_tb.sv”. Στο πρόγραμμα αυτό, αρχικά δηλώνουμε τις απαραίτητες μεταβλητές που απαιτούνται ενώ στη συνέχεια όπως φαίνεται και παρακάτω αρχικοποιούμε τα module με τους προηγούμενους κώδικες στο αρχείο του testbench.

```
1   module edge_detection_tb;
2
3       // |0  1  2|
4       // |7  8  3|
5       // |6  5  4|
6
7       logic clk;
8       logic rst;
9       logic pixels[8:0];
10      logic pixels_after_step_1;
11      logic pixels_after_step_2;
12
13      always begin
14          clk = 1;
15          #1ns;
16          clk = ~clk;
17          #1ns;
18      end
19
20      edge_detection_first_step first_step(
21          .clk                (clk),
22          .rst                (rst),
23          .neighbors_state    (pixels),
24          .state              (pixels_after_step_1)
25      );
26      edge_detection_second_step second_step(
27          .clk                (clk),
28          .rst                (rst),
29          .neighbors_state    (pixels),
30          .state              (pixels_after_step_2)
31      );
```

Αρχικοποίηση μεταβλητών και προηγούμενων module.

Έπειτα ξεκινάει η διαδικασία του verification κατά το οποίο εξετάζουμε 6 περιπτώσεις δίνωντας όπως ήδη αναφέρθηκε τις απαραίτητες τιμές στις εισόδους των module.

- 1η Περίπτωση: Τρεις συνεχόμενοι “1” στη γειτονιά Moore με τα αντιδιαμετρικά τους pixel να είναι “0”, με αποτέλεσμα το κεντρικό pixel να κρατάει την τιμή “1”.
- 2η Περίπτωση: Τρεις συνεχόμενοι “1” στη γειτονιά Moore με τα αντιδιαμετρικά τους pixel να μην είναι “0”, με αποτέλεσμα το κεντρικό pixel αλλάζει την κατάσταση του σε “0”.
- 3η Περίπτωση: Ακριβώς 2 “1” στη γειτονιά Von Neumann με αποτέλεσμα να διατηρείται ο “1” του κεντρικού pixel.

- 4η Περίπτωση: Ακριβώς 2 “1” στη γειτονιά Von Neumann ενώ αυτή τη φορά τα υπόλοιπα γειτονικά pixel να μην είναι όλα “0” με αποτέλεσμα το κεντρικό pixel να επιστρέφη στην τιμή “0”.
- 5η Περίπτωση: Τρεις συνεχόμενοι “1” στη γειτονιά Moore με τα αντιδιαμετρικά τους pixel να είναι “0”, αλλά το κεντρικό pixel είναι ίσο με “0”, με αποτέλεσμα το κεντρικό pixel να κρατάει την τιμή “0”.
- 6η περίπτωση: Ακριβώς 2 “1” στη γειτονιά Von Neumann, αλλά το κεντρικό pixel είναι ίσο με “0”, με αποτέλεσμα να διατηρείται το “0” του κεντρικού pixel.

Παρακάτω μπορείτε να δείτε τα αποτελέσματα έπειτα από την εκτέλεση του αρχείου που περιγράφηκε σε αυτό το κεφάλαιο στο περιβάλλον του Modelsim.

```
VSIM 6> run -all
# Starting Test Bench
# Pixel : 1
#
# Pixel : 0
#
# Pixel : 1
#
# Pixel : 0
#
# Pixel : 0
#
# Pixel : 0
#
# Finishing test
```

Αποτελέσματα verification.

❑ Top level

Τα επιμέρους βήματα του αλγορίθμου, που περιγράφονται παραπάνω συγκροτούν το top level του module. Το αρχείο αυτό δέχεται σαν όρισμα μία δυαδική εικόνα μεγέθους 7x7 για χάρη ευκολίας στον έλεγχο, αλλά οι διαστάσεις αυτές δύναται να αλλάξουν. Το πρόγραμμα αρχικά δημιουργεί μία επαυξημένη εικόνα βάσει της αρχικής, περιβάλλοντάς την απλά με pixel που έχουν την τιμή 0. Το τελικό αποτέλεσμα είναι μια εικόνα 9x9 που περιέχει όλη την πληροφορία της αρχικής. Σε αυτήν την εικόνα εφαρμόζεται το πρώτο βήμα του αλγορίθμου. Η εικόνα που προκύπτει επεξεργάζεται επαναληπτικά από το δεύτερο βήμα του αλγορίθμου μου, έως ότου επιτευχθεί το επιθυμητό sharpness. Η τελική εικόνα προκύπτει αφότου αφαιρεθούν τα μηδενικά που εισήλθαν κατά την επαύξηση.

5. Αντιμετώπιση προβλημάτων

Όσον αφορά τα προβλήματα που είχαμε κατά τη διάρκεια υλοποίησης της εργασίας το κυριότερο αποτελεί το γεγονός ότι δεν καταφέραμε να εξετάσουμε των αλγόριθμο και τους κώδικες που αναπτύξαμε σε πραγματικές εικόνες. Αυτό οφείλεται κυρίως στο γεγονός ότι δεν υπηρχε η απαραίτητη γνώση αναφορικά με το περιβάλλον της γλώσσας SystemVerilog. Σε γενικές γραμμές όπως είδατε και στο προηγούμενο κεφάλαιο, για την αντιμετώπιση του εν λόγω προβλήματος εξετάστηκαν μεμονωμένα οι 2 κώδικες που αναπτύχθηκαν καθώς επίσης έγινε και η ανάπτυξη ενός Top Level κώδικα που υλοποιεί ιδανικά και τα δύο αρχεία που δημιουργήσαμε.

6. Προοπτικες για βελτίωση

Το πρόγραμμα θα μπορούσε να λειτουργήσει και για την ανίχνευση ακμών σε ασπρόμαυρες εικόνες που αποτελούνται από μη δυαδικά pixel εκχωρώντας την τιμή 0 σε όλα τα pixel τα οποία έχουν τιμή κάτω από ένα ορισμένο κατώφλι και την τιμή 1 σε αυτά που το ξεπερνούν, πριν εφαρμοστεί η επεξεργασία.
Ανάλογα με τις τιμές της γειτονιάς των pixel που βρίσκονται στις άκρες της εικόνας θα μπορούσαν να συμπληρωθούν τιμές διάφορες του 0 στην επαυξημένη εικόνα που δημιουργεί εσωτερικά ο αλγόριθμος, προκειμένου να επιτευχθεί μεγαλύτερη ακρίβεια στον εντοπισμό ακμών σε αυτές τις περιοχές.

7. Βιβλιογραφία

Cellular Automata in Image Processing and Geometry - Paul Rosin · Andrew Adamatzky Xianfang Sun