Γιαννακοπούλου Λαμπρινή - 57776
Εργασία 4η

Σύνθεση Υψηλού επιπέδου για τη σχεδίαση ψηφιακών ολοκληρωμένων κυκλωμάτων

Μέρος 1: Υλοποίηση ομαδοποίησης k-means

Ο κώδικας C++ για την εργασία βρίσκεται στο github μου: https://github.com/lamprini19/High-Level-Synthesis και περιλαμβάνει το αρχείο kmeans.cpp (που περιέχει τη συνάρτηση kmeans και την main) και το types.h (που περιέχει τη δήλωση της δομής Point).

Συνάρτηση k-means

Ο αλγόριθμος k-means υλοποιήθηκε με τη συνάρτηση kmeans όπως ζητείται στην εκφώνηση της εργασίας.

Αρχικά, για κάθε σημείο (βρόχος POINTS) διαβάζουμε τα x και y από τον πίνακα points και αρχικοποιούμε τη μεταβλητή min_distance στη μέγιστη δυνατή τιμή. Έπειτα για κάθε cluster (βρόχος DIST), υπολογίζουμε την απόσταση Manhattan του σημείου από το αντίστοιχο κέντρο. Αν η απόσταση αυτή είναι μικρότερη από τις αποστάσεις από τα κέντρα των προηγούμενων ομάδων, το ID του σημείου αλλάζει. Αυτό το κομμάτι φαίνεται παρακάτω:

```
// calculate manhattan distance
ac_math::ac_abs(x-center_x, abs_x);
ac_math::ac_abs(y-center_y, abs_y);
distance = abs_x + abs_y;

if (distance < min_distance) {
     min_distance = distance;
     ID[i] = j;
}
```

Αφού ομαδοποιήσουμε τα σημεία, πρέπει να υπολογίσουμε το νέο κέντρο κάθε cluster, και να ελέγξουμε αν τα νέα κέντρα είναι ίδια με τα παλιά.

Έτσι, για κάθε ομάδα (βρόχος NEW_CENT), υπολογίζουμε τα μέσα x και y των σημείων που ανήκουν σε αυτή, δηλαδή τα νέα κέντρα (μεταβλητές new_cent_x, new_cent_y).
Στη συνέχεια ελέγχουμε αν τα κέντρα άλλαξαν. Αν έχουν αλλάξει, κάνουμε την bool μεταβλητή true (από false που είναι το default), και ανανεώνουμε τα κέντρα όπως φαίνεται εδώ:

```
// check if centers changed
ac_int<16, false> diff_x, diff_y;
ac_math::ac_abs(center[j].x-new_cent_x, diff_x);
ac_math::ac_abs(center[j].y-new_cent_y, diff_y);
if ( diff_x >= 1 || diff_y >= 1) {
    change = true;
    center[j].x = new_cent_x;
    center[j].
y = new_cent_y;
}
```

Κλήση από τη main

Αρχικά παράγουμε N τυχαία σημεία-δείγματα τύπου Point, με τη συνάρτηση rand() και την τωρινή ώρα σαν seed. Με τον ίδιο τρόπο αρχικοποιούμε τυχαία και τα κέντρα των τριών ομάδων.

Μετά η συνάρτηση kmeans καλείται επαναληπτικά μέχρι να επιστρέψει τιμή false (δηλαδή να σταματήσουν να αλλάζουν τα κέντρα).

Αποτελέσματα

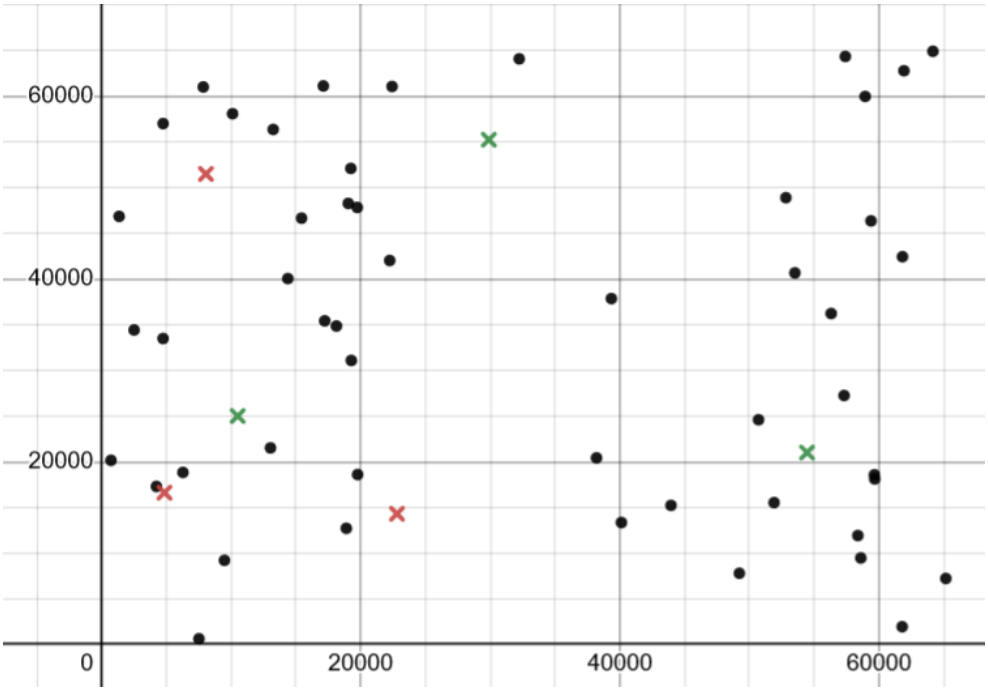
Για να επαληθεύσουμε ότι ο κώδικας λειτουργεί σωστά, τον δοκιμάζουμε για διάφορους αριθμούς σημείων. Ενδεικτικά, για M=3 clusters και N=50 σημεία, το πρόγραμμα τυπώνει για τα τυχαία αρχικά κέντρα:

```
# Initial Centers:
# 8051 51524
# 22795 14327
# 4849 16643
```

Και για τα τελικά:

```
# The 3 centers are:
# 29886 55280
# 54448 21031
# 10498 25035
```

Τα αποτελέσματα φαίνονται επίσης στο παρακάτω γράφημα, όπου με μαύρο είναι τα τυχαία samples, με κόκκινο τα αρχικά κέντρα, και με πράσινο τα τελικά. Είναι εμφανές ότι ενώ τα κέντρα ξεκινούν από σημεία σχετικά κοντινά μεταξύ τους, τελικά παίρνουν τις κατάλληλες θέσεις.



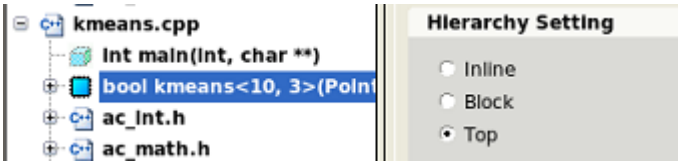
Μέρος 2: Σύνθεση

Αρχικά τροποποιούμε τη συνάρτηση ώστε να μπορεί να γίνει το cosimulation. Αυτό περιλαμβάνει στην περίπτωσή μας 3 αλλαγές:

- `bool CCS_BLOCK(kmeans)` αντί για `bool kmeans` στη δήλωση της συνάρτησης kmeans(),
- `CCS_MAIN(...)` αντί του `int main(...)`,
- `CCS_RETURN(0);` αντί για `return 0;`
- και φυσικά την προσθήκη της γραμμής `#include "mc_scverify.h"`

Κάνουμε τη σύνθεση, με M=3 όπως πάντα και N=10. Το N επιλέχθηκε επίτηδες τόσο μικρό, για να μην κρατήσει πάρα πολύ η σύνθεση.

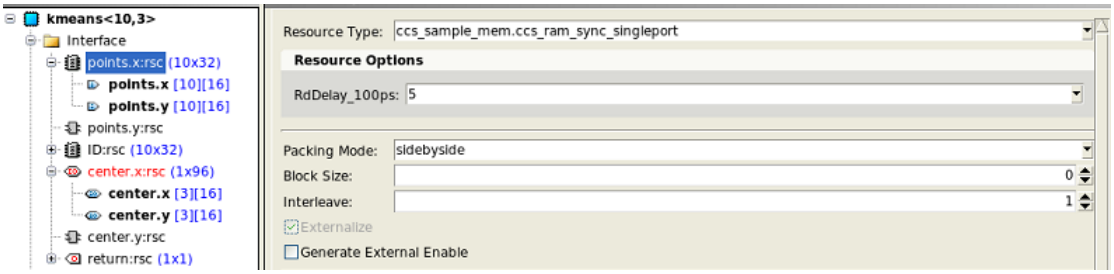
Επιλέγουμε ως top module τη συνάρτηση kmeans() όπως φαίνεται εδώ.



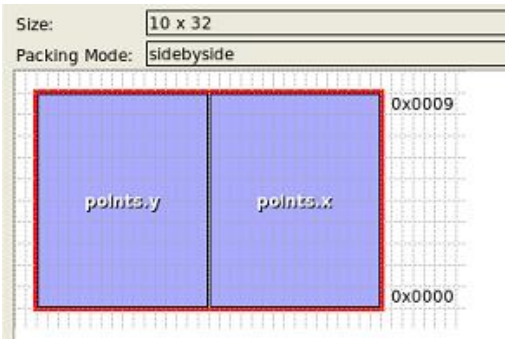
Αφού γίνει αυτό, στο πεδίο Mapping επιλέγω συχνότητα ρολογιού 500MHz.

<u>Επιλογή αρχιτεκτονικής</u>

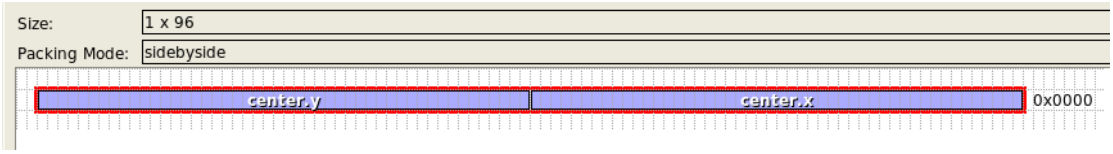
Για να βρίσκονται τα points.x και points.y στην ίδια θέση μνήμης όπως ζητάει η εκφώνηση, στο πεδίο Architecture ορίζω το points.x ως singleport ram μνήμη, με Packing Mode “sidebyside”.



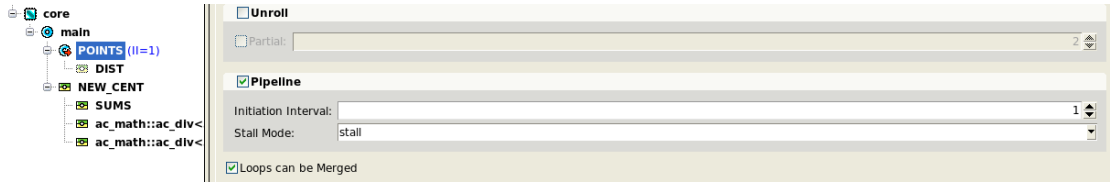
Αυτή η επιλογή, σε συνδυασμό με την αλλαγή του μεγέθους λέξης όπως περιγράφεται στην εκφώνηση, έχουν ως αποτέλεσμα να μπορούν να διαβαστούν τα x και y στον ίδιο κύκλο ρολογιού, και η μνήμη points.x να μοιάζει κάπως έτσι:



Κάτι αντίστοιχο κάνουμε και για την είσοδο-έξοδο center.x, που μετά τις αλλαγές είναι:



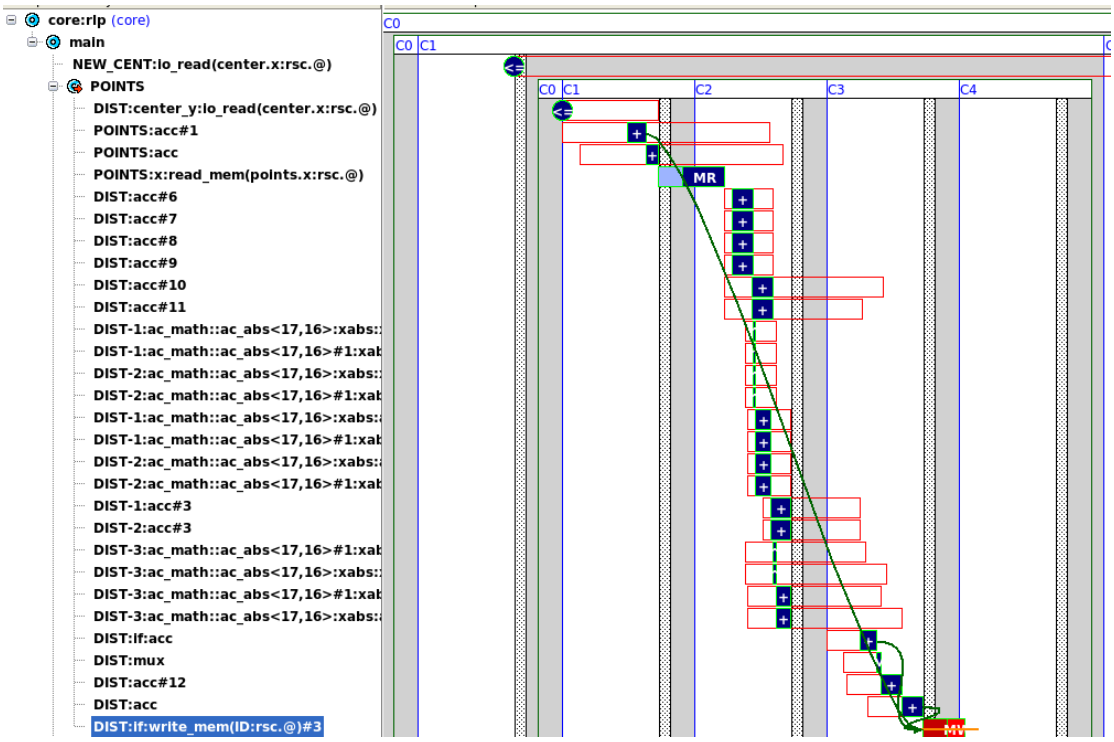
Πέρα από τις ρυθμίσεις του interface πρέπει να κάνουμε και τις κατάλληλες ρυθμίσεις στους βρόχους, δηλαδή pipeline με initiation interval 1 για τον βρόχο POINTS όπως φαίνεται στην εικόνα, και loop unrolling για τους υπόλοιπους βρόχους (DIST, SUMS, και τις χρήσεις του ac_div).



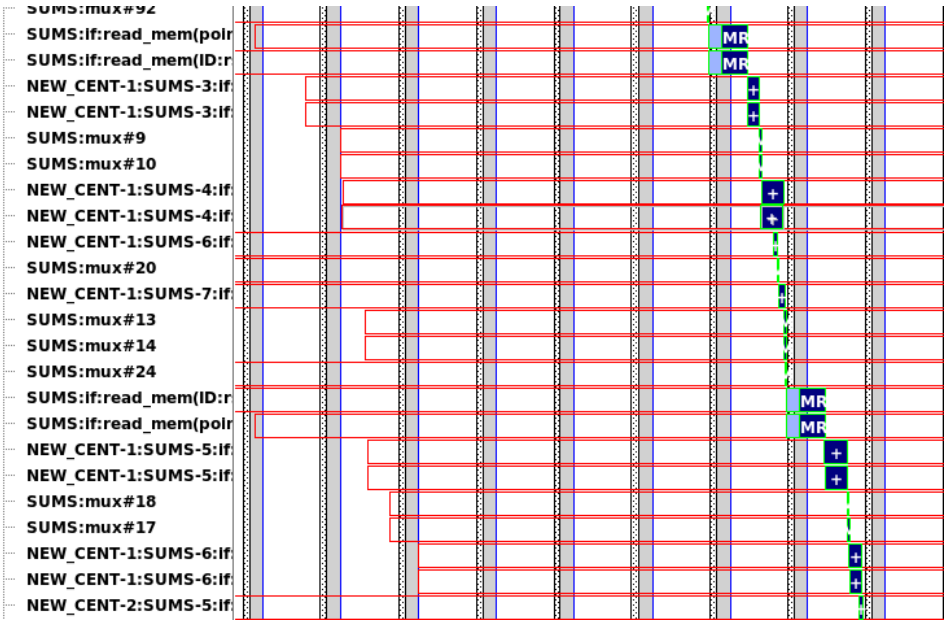
Χρονοπρογραμματισμός

Προχωρώντας στο πεδίο Schedule, βλέπουμε ότι:

Ο βρόχος POINTS εκτελείται σε pipeline, ενώ ο DIST έχει γίνει unrolled. Παρατηρούμε ότι υπολογισμοί για τις αποστάσεις και από τα κέντρα και των 3 clusters γίνονται παράλληλα, λόγω του unrolling. Χρειάζονται 3 (?) κύκλοι ρολογιού για κάθε επανάληψη πέρα από την πρώτη.



Μετά περνάμε στο scheduling των nested loops NEW_CENT και SUMS. Υπάρχουν φυσικά και οι διαιρέσεις με ac_div, που επίσης είναι βρόχοι. Για αυτούς τους 3 βρόχους επιλέξαμε να γίνει unrolling, οπότε όπως είναι λογικό το εργαλείο σύνθεσης εκτελεί πράξεις «ανακατεμένα» από διάφορες επαναλήψεις των 3 loops, με τον τρόπο που θεωρεί βέλτιστο. Παρακάτω φαίνεται ένα τμήμα του χρονοπρογραμματισμού αυτού.



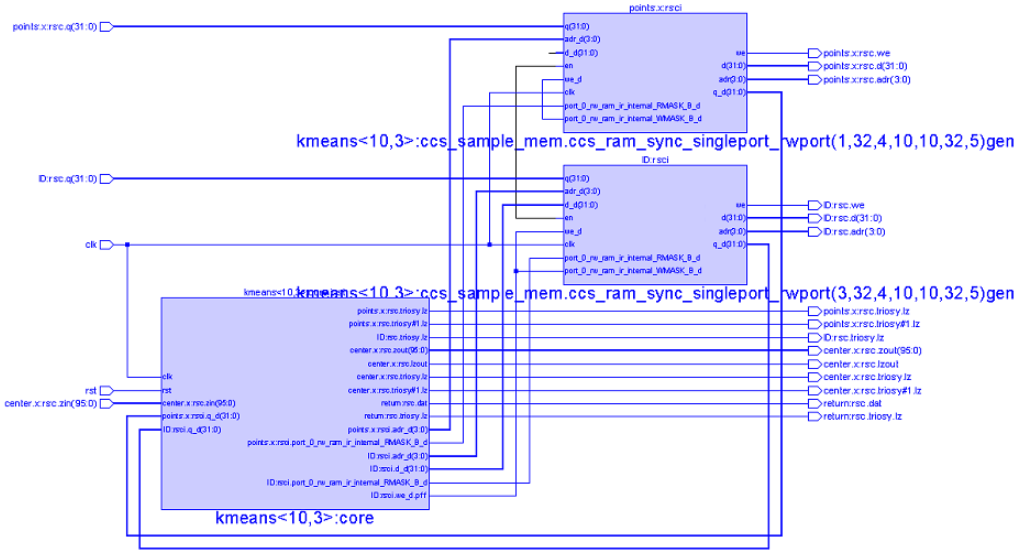
Κάτι που ίσως αξίζει να αναφερθεί ότι οι υπάρχουν ακριβώς N=10 προσπελάσεις στη μνήμη ID για τον έλεγχο στο κομμάτι:

```
NEW_CENT: for (int j = 0; j < M; j++) {
    . . .
        SUMS: for (int i = 0; i < N; i++) {
            if (ID[i] == j) {
                  . . .
            }
        }
```

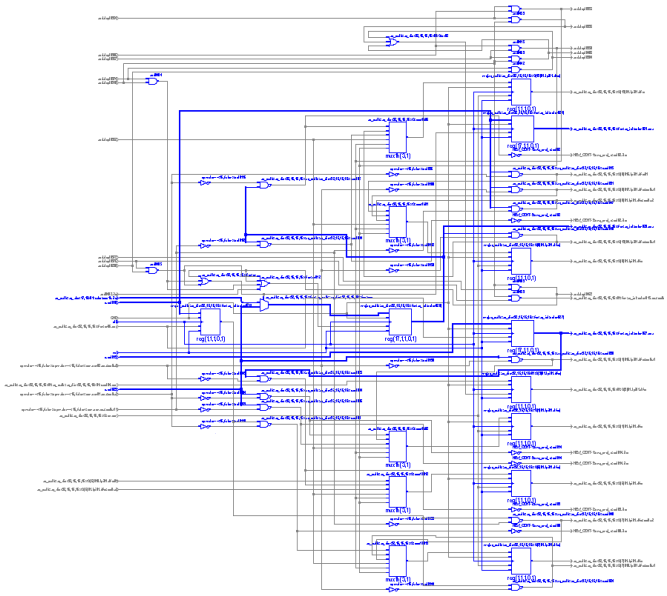
Αυτό μπορεί να φαίνεται παράδοξο γιατί στο software οι προσπελάσεις θα ήταν N*M, αλλά στο hardware λόγω της παραλληλίας δεν υπάρχει αυτή η ανάγκη και αρκεί να διαβαστεί κάθε τιμή μία φορά.

RTL

Πάταμε RTL, και επιλέγουμε να εμφανιστεί το RTL Schematic που φαίνεται στην παρακάτω εικόνα. Μπορούμε να δούμε τις 2 μνήμες points.x και ID, καθώς και το top module kmeans.

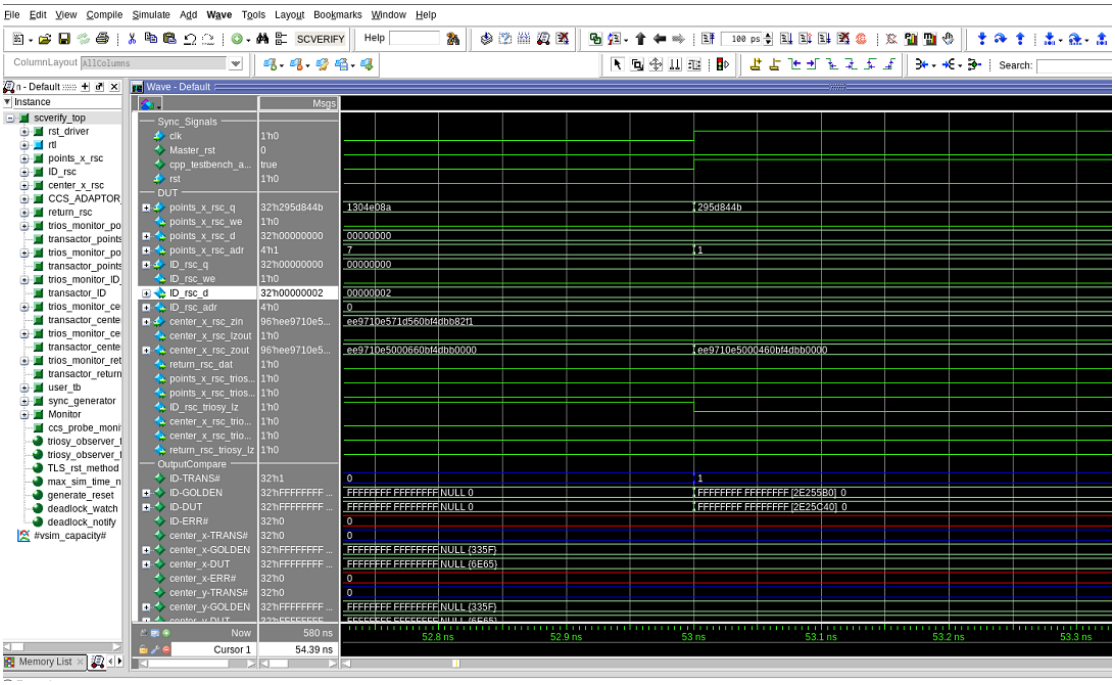


Επιλέγοντας το kmeans μπορούμε να το δούμε με περισσότερη λεπτομέρεια, σε επίπεδο πυλών και καταχωρητών:



Co-simulation

Έπειτα ανοίγουμε το QuestaSIM, και τρέχουμε την προσομοίωση, που φαίνεται εδώ:



Εκτός από την προσομοίωση της σύνθεσης σε υλικό, το QuestaSIM συγκρίνει τα αποτέλεσματα αυτής της προσομοίωσης με τα αποτελέσματα της C++.

Σε αυτό το screenshot φαίνονται τα αποτελέσματα των συγκρίσεων. Οι τιμές «error count» για όλες τις μεταβλητές (που αλλάζουν) και για το return είναι 0, πράγμα που σημαίνει ότι δεν υπάρχουν λάθη στη σύνθεση.

```
# Checking results
# 'ID'
#    capture count       = 3
#    comparison count    = 3
#    ignore count        = 0
#    error count         = 0
#    stuck in dut fifo   = 0
#    stuck in golden fifo = 0
# 'center_x'
#    capture count       = 3
#    comparison count    = 3
#    ignore count        = 0
#    error count         = 0
#    stuck in dut fifo   = 0
#    stuck in golden fifo = 0
# 'center_y'
#    capture count       = 3
#    comparison count    = 3
#    ignore count        = 0
#    error count         = 0
#    stuck in dut fifo   = 0
#    stuck in golden fifo = 0
# 'return'
#    capture count       = 3
#    comparison count    = 3
#    ignore count        = 0
#    error count         = 0
#    stuck in dut fifo   = 0
#    stuck in golden fifo = 0
#
```