

האוניברסיטה הפתוחה

המחלקה למתמטיקה ולמדעי המחשב

עבודה מסכמת בנושא

**"אלגוריתמים מקביליים לזיהוי פגמים ביצור מעגלים
משולבים"**

העבודה המסכמת הוגשה כחלק מהדרישות לקבלת תואר

"מוסמך למדעים" M.Sc. במדעי המחשב

באוניברסיטה הפתוחה

החטיבה למדעי המחשב

על ידי

ליאור יחיאלי

ת"ז 022945646

העבודה הוכנה בהדרכתו של ד"ר עמי מרובקה

2009

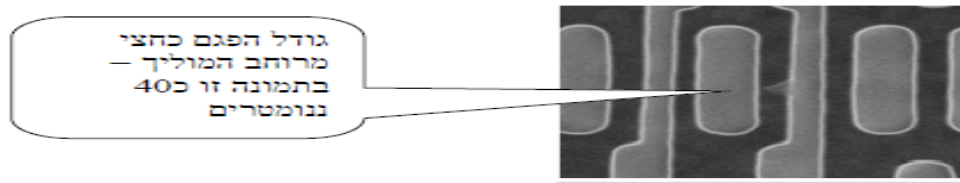
תוכן עניינים

4	תקציר
6	1 מבוא
6	1.1 תהליך וטכניקות ייצור השבבים
7	1.1.1 חפירת התעלות
8	1.1.2 המסכה
9	1.1.3 שלבים כימיים וטכניקות בתהליך ייצור השבב
9	1.2 תהליך בקרת האיכות בייצור השבבים
10	1.2.1 מכונת Wafer Inspection
10	1.2.2 מכונת SEM Defect Review
10	1.2.3 מכונת Mask Inspection
10	1.2.4 מכונת CD
10	1.2.5 תרשים בקרת התהליך ודוגמאות לפגמים
11	1.3 הפן הכלכלי
13	2 מערכות לגילוי פגמים
13	2.1 מבוא
14	2.1.1 אופן רכישת התמונה ועיבוד
16	3 מודל כללי עבור מערכת עיבוד תמונה במערכות inspection
16	3.1 הדרישות ממערכת עיבוד תמונה עבור WI
17	3.2 סקירת הדרישות מארכיטקטורות חישוב למערכות עיבוד תמונה
18	3.2.1 ארכיטקטורת מערכת עיבוד תמונה מבוססת DSP
20	3.2.1.1 שיקולים לבחירת ה- DSP
21	3.2.1.2 תיאור כללי של תצורת מערכת עיבוד התמונה מבוססת DSP
23	4 אלגוריתמי השוואה כפולה מבוססים die2die למציאת פגמים
27	4.1.1 מדידת ההיסט בין תמונות (Registration Measurement)
30	4.1.2 תיקון ההיסט בין תמונות עוקבות (Registration Correction)
32	4.1.3 מדידה ותיקון של תופעת ה- Color Variation
39	4.1.4 בחינת ביצועי אלגוריתמי השוואה מבוססת Die2Die
44	5 אלגוריתמי השוואה Cell to Cell למציאת פגמים
44	5.1 תיאור אלגוריתם השוואה מבוססת Cell2Cell
46	5.2 בחינת ביצועי אלגוריתמי השוואה מבוססים Cell2Cell
49	6 אלגוריתמים להשוואת תא מחזורי למציאת פגמים
49	6.1 ניתוח ועיבוד התמונה המחזורית במישור התדר
53	6.2 ניתוח התמונה המחזורית במרחב מישור התמונה
56	6.3 התאמות למערכות WI וניתוח התוצאות
61	7 סיכום ומבט לעתיד

63	נספחים
63	נספח 1 - תהליך ייצור הטרנזיסטור
63	מבנה הטרנזיסטור
63	המבנה הגבישי של מוליך למחצה
63	צומת PN וטכנולוגית CMOS
64	איך מייצרים טרנזיסטור
66	נספח 2 - מערכות BF מבוססות הארת לייזר לעומת הארת מנורה
66	השוואת מערכות BF מבוססות הארת מנורה לעומת הארת לייזר
67	יתרונות וחסרונות בין הארת מנורה להארת לייזר
68	נספח 3- מבוא וסקירה היסטורית לאבולוציית מערכות עיבוד תמונה WI
69	ארכיטקטורת Intel Multi-Core
70	ארכיטקטורת מערכת עיבוד תמונה מבוססת IBM-Cell
71	ארכיטקטורת מערכת עיבוד תמונה מבוססת GPU
71	ארכיטקטורת מערכת עיבוד תמונה מבוססת FPGA
73	נספח 4 - תיאור יחידת עיבוד Processing Node
73	תיאור הליבה הנבחרת C64+
73	ליבה בארכיטקטורת VLIW
74	ליבה C64+
76	מבנה ה-pipeline בליבה C64+
80	תיאור הפקודות הווקטוריות הנפוצות בעיבוד תמונה ב-C64+
80	תיאור פקודות pack
82	פקודות multiply -I Dot-Product
82	פקודת חיבור חיסור וקטור מקבילי
83	פקודות minimum,maximum -I avg
83	פקודת טעינת ואחסון וקטור
83	פקודות השוואה ובניית מסכה
86	בניית לולאת PIPELINE יעילה
90	תיאור Mega-Module והפעלת ה-EDMA
92	מימוש פונקציות עיבוד תמונה אופייניות
93	דוגמא תיאור הקוד המנהל את העתקות ה-EDMA
95	מימוש מסנן 3x3
101	מימוש פונקציית השוואה
104	תוצאות ריצה אלגוריתם D2D בסיסי
105	סיכום מגבלות במימוש אלגוריתמים למכונת WI
106	נספח 5 - עקרונות תיאורטיים בעיבוד תמונה
110	ביבליוגרפיה
111	רשימת האיורים:
114	רשימת הטבלאות:

תקציר

בעולם הטכנולוגי כיום מעגלים משולבים (IC) integrated circuits הם חלק בלתי נפרד מכל מכשיר אלקטרוני החל במכשירים פשוטים והמשך עם טלפונים סלולאריים, מחשבים, טלוויזיות וכדומה. המבנה הבסיסי של כל מעגל משולב הוא הטרנזיסטור. תעשיית המוליכים למחצה עוקבת אחר חוק Moore הידוע הטוען שכל שנתיים עד שלוש ניתן להכפיל את מספר הטרנזיסטורים ליחידת שטח. חוק Moore משמש כמעין מפת דרכים לתעשייה ששואפת כל הזמן ליצור מעגלים משולבים עם טכנולוגיה גבוהה שהמאפיין העיקרי שלהם הוא מזעור הרכיבים המודפסים על שבב הסיליקון דבר המאפשר להאיץ את קצבי החישוב ולהגדיל זמינות ונפח זיכרון. תעשיית המיקרו אלקטרוניקה מצליחה לייצב תהליכי ייצור משופרים כל כשנתיים-שלוש ושומרת על קצב המזעור המתאים לחוק מור (Moore) דהיינו קצב אקספוננציאלי ריבועי של הכפלת מספר הטרנזיסטורים על שטח השבב כל שנתיים. כיום מייצרים שבבים בגודל מוליכים של 45 ננומטרים ומפתחים יכולת עתידית למזער ל- 22 ננומטרים. מכונות בקרת תהליכים נדרשות לגלות פגמים המסכנים את השבב. ניתן להניח שסטייה גאומטרית או חלקיק בגודל של חצי מרוחב המוליך תהווה סיכון ויש לגלותה.



איור 1 דוגמא לפגם בשבב סיליקון

חישוב מהיר מראה שעל פרוסת סיליקון בקוטר של 300 מילימטרים יש לסרוק ולעבד כמות יחידות שטח בגודל 30×30 ננומטרים העולה על 10^{14} . בהנחה שכל יחידה היא פיקסל נקבל בקירוב 3132Gbyte. יצרני הרכיבים מבקשים בקרת תהליך שתאפשר החלטה על תקינות התהליך תוך פחות משעה (כדי למנוע אובדן חומר יקר). וכך נקבל שקצב עיבוד הנתונים הדרוש הוא 25Gbyte/second-50Gbyte/second שהוא לכל הדעות קצב עיבוד אדיר. בנוסף לקצב העצום של עיבוד הנתונים, האלגוריתמים הדרושים לצורך מציאת הפגמים הופכים למורכבים ומסובכים יותר ויותר והסיבה לכך היא שכל שעולים ברזולוציה (גודל פיקסל יותר קטן) התמונה המתקבלת היא באיכות ירודה מכיוון שיחס האות לרעש קטן ומקשה על יצירת אלגוריתם פשוט למציאת הפגמים (חלק נרחב בעבודה זו יתאר בעיות אלו). יחס אות לרעש הקטן בנוסף לצורך לעבד קצבי מידע גבוהים, שגם העיבוד שלהם דורש כוח מחשוב חזק גורם למצב שלמרות שבמקרים מסוימים ישנו אלגוריתם ראוי למציאת הפגמים, הוא עלול להיות יקר במונחים של סיבוכיות החישוב ולכן בהחלט יכול להיות שהוא איננו ישים. המכונות שתומכות בבקרת איכות של תהליך ייצור המעגלים המשולבים המפותחים על מצע של פרוסת סיליקון (wafer) נחלקות לשתי קבוצות עיקריות:

- **מכונות פיקוח Wafer Inspection (WI)** - מכונות אלו אמורות לסרוק במהירות המרבית את ה-Wafer כולו ולהצביע על מיקומים של תקלות ב-Wafer. מכונות אלו אמורות להיות אמינות, יחסית זולות ומהירות ועם זאת לספק דרגת רגישות (sensitivity) גבוהה ואמינות גבוהה, כלומר יחס דיווח פגמים אמיתיים לעומת שיקריים גבוה.
- **מכונות Review** - מכונות אלו מסוגלות לקבל מיקומים של פגמים ממכונות הפיקוח ולתת תמונות ברזולוציה ובאיכות גבוהה של הפגמים. ניתן לתאר מכונות אלו כסוג של מיקרוסקופ אלקטרוני בכל רזולוציה גבוהה מאוד היכולה להגיע עד ל-0.5nm לפיקסל. עם התקדמות הטכנולוגיה והיכולת למזער יותר ויותר את המעגלים המשולבים (מעבדים זיכרונות וכדומה) הפכה משימת בניית מכונות ה-Wafer Inspection למורכבת מאוד הן מההיבט הפיזיקלי מכיוון שצריך לבצע איסוף תמונה ברזולוציות גבוהות מאוד.

נתון זה משפיע באופן ישיר על סיבוכיות מערכות עיבוד התמונה הצריכות לקלוט את התמונות המגיעות בסריקה ה-Wafer בזמן אמת לעבד אותם אלגוריתמית ולשלוח את רשימות הפגמים והתמונות שלהם לנקודות איסוף מרכזית (כגון מכונות ה-Review).

מערכת עיבוד תמונה כזו צריכה לעמוד בדרישה של עיבוד זמן אמת (Real Time Processing) של מידע בקצב (Data Rate) גבוה מאוד בסדרי גודל שבין 4-25Gbyte/sec.

כתוצאה מכך מערכות אלו מציבות אתגר טכנולוגי גדול בפני מתכנני מערכת עיבוד התמונה הן מהיבט החמרי, האלגוריתמי, והן מצד המתכנת המממש את האלגוריתמים.

קיימים מספר מודלים למימוש מערכות זמן אמת לעיבוד תמונה כגון מערכות מבוססות DSP, מערכות מבוססות FPGA או שילוב שלהם ו/או מבוססות מעבדי Pentium.

המשותף לכל המודלים הללו הוא שמערכת עיבוד התמונה מבוססת על מספר רב של ליבות חישוב מהסוג שהוזכר לעיל. מערכות מרובות ליבות אלו צריכות לעמוד בדרישות זמן אמת מחמירות (כלומר לעמוד באילוצי זמן של מספר מחזורי מכונה (cycles) המוגדרים מראש במוצא עיבוד כל פיקסל) ובנוסף עליהם להיות ניתנות להרחבה בצורה יחסית קלה, כלומר הוספת כוח מחשוב יתרחש ביחס ישר להוספת ליבות חישוב למערכת, כלומר המערכת צריכה להיות ניתנת להרחבה.

בעבודה זו נתבסס על מעבד DSP כאבן הבניין במערכת מרובת הליבות. האלגוריתמים יצטרכו להיות כאלה שניתן לממש אותם ביעילות על מערכות מבוססות DSP וכן שיבטיחו גילוי פגמים עם רמת רגישות מספיקה בהתחשב בסביבה המכנית, הפיזיקלית והאופטית שבה פועלות מכונות ה-Wafer Inspection (WI).

במערכת WI קיימים שני אזורים עיקריים המרכיבים את ה-Wafer הנסרק:

הראשון אזור Logic שבד"כ מכיל תבנית כלשהי של קווים אבל לא דווקא מחזורית. השני אזור Array המכיל בעיקר אזורים מחזוריים הנקראים תאים (Cells שמהם בונים אזורי זיכרון) התחומים בתוך אזורים לא מחזוריים הנקראים Stitch (שבהם ממומשת הלוגיקה הבוררת של רכיבי הזיכרון).

קיימים גם אזורים ריקים יחסית מתבנית (bare או ליתר דיוק bar wafer) שעליהם ארזיב פחות מכיוון שהבעיה של מציאת פגמים ב-wafers אלה יחסית קלה בסדרי גודל אחד או שניים יחסית למציאת פגמים ב - logic, array.

מכיוון שישנם טכנולוגיות פיסיקליות שונות להארת ה-Wafer (העיקריות שבהם לייזר קוהרנטי או אור רחב פס) הפתרון והאלגוריתם עלולים להיות מושפעים כתוצאה מהטכנולוגיה הפיסיקלית. בעבודה זו אתייחס למכונות המאירות בלייזר קוהרנטי מונו-כרומטי.

ישנם שתי טכניקות עיקריות לגילוי פגמים:

- השוואת Die ל-Die הסמוך לו. מכיוון ששני ה-dies זהים במבנה שלהם ניתן ע"י ההשוואה בניהם לגלות פגמים.
- גילוי ומציאת המחזוריות בתוך ה-Die ושימוש בעובדה שהמידע מחזורי כדי לגלות פגמים.

קיימים אלגוריתמים רבים לגילוי פגמים על משטחים מחזוריים (ראה מאמרים [6],[7],[8],[9],[10],[11],[12],[13],[14],[15],[16],[17],[18],[19],[20],[21],[22],[23],[24],[25],[26],[27],[28],[29],[30],[31]).

כאמור מטרת העבודה היא לחקור אלגוריתמים אלו ולנסות לבדוק את מידת התאמתם ואת דרך מימושם (ובמידת הצורך לשנותם או להתאימם) למערכות WI מסוג BF כך שיהיו יעילים הן מבחינת רמת רגישות זיהוי הפגמים, ומצד שני ירוצו בזמן או במספר המחזוריים המוגדר כדי לקיים את דרישות זמן האמת החמורות הקיימות במכונה זו.

יש להדגיש שחלק מהאלגוריתמים שאחזקור מנסים לנצל את תכונת המחזוריות של התמונות, אבל לא תמיד יש התייחסות להשפעות הפיזיקליות שיש לסביבה על התמונות המתקבלות וכן אין התייחסות למימוש מקבילי של האלגוריתם.

מימוש האלגוריתמים בעבודה זו יעשה בערכת הערכה EVMDM 6437 של חברת TI הכוללת כרטיס המאפשר פיתוח אב טיפוס של אפליקציות DSP ומבוסס על ליבת ה-DSP tms320dm6437.

1 מבוא

כאמור מעגלים משולבים נכללים ביותר ויותר מוצרים, והשוק דורש שהם יהיו במחיר נמוך, עם ביצועים גבוהים וימשיכו לצמוח ולתת ביצועים גבוהים יותר עם אותה רמת מחירים. כאמור המבנה הבסיסי של כל מעגל משולב מבוסס על מיליארדי טרנזיסטורים המחוברים בניהם בעזרת מוליכים זעירים. תהליך ייצור השבבים הינו מורכב ומסובך וכולל מספר שלבים ומבוצע ע"י מכוונות שיוצרות את התהליך. אולם תהליך זה מתקדם ונהיה מורכב ככל שהטכנולוגיה מתקדמת ולכן דורש מערכות מתוחכמות יותר לבקרת התהליך. ניתן להגדיר רכיבים אלה כרכיבים ננומטרים מכיוון שגודל הרכיבים הבסיסיים המוטבע בהם הוא בסדרי גודל של עשרות עד ננומטרים בודדים. גודלו של השבב המיוצר נע בין 1 – 5 סמ"ר ומכיל כאמור מספר עצום (ממיליונים עד מיליארדים) שכולם מיוצרים בבת אחת ובאותו תהליך יצור. גודל הפרט הקטן ביותר הנמצא והמותר בשבב נקרא כלל העיצוב (design rule). לדוגמא שבב שכלל העיצוב שלו הוא 100 ננומטר יכול פרטים בגודל 100 ננומטר ולא קטנים מכך. מעבר לכך כלל העיצוב קובע את עובי החוטים שהם בד"כ פסי נחושת המחוברים בין חלקיו. כלל העיצוב של שבב נגזר מטכנולוגית הייצור בה משתמשים לייצורו והוא המגדיר את גבולות הגודל של הטכנולוגיה הנתונה. אם ננסה לייצר בטכנולוגיה כלשהי פרטים קטנים מכלל העיצוב נסתכן בכישלון התהליך ואנו עלולים לייצר שבב פגום. בתעשיית המוליכים למחצה נהוגים היום כללי עיצוב שבין 1.5 מיקרון ל- 0.032 מיקרון (כלומר מ- 1500 ועד 32 ננומטרים). כאמור בתהליך הייצור של השבב מוטבעים בו כל רכיביו כולל החוטים הנדרשים (חיבורי החוטים ומיקומם) כך שבסוף התהליך מתקבל שבב גולמי שרכיביו מחוברים ויש לו ממשק יציאות זעיר לעולם החיצון. שבב זה מוכנס לתושבות המחוברות את חוטי הממשק הזעירים לרגלי מתכת שגודלן נע בין 0.5-1 מילימטר והיא המעניקה לו את המראה המזכיר חרק.

1.1 תהליך וטכניקות ייצור השבבים

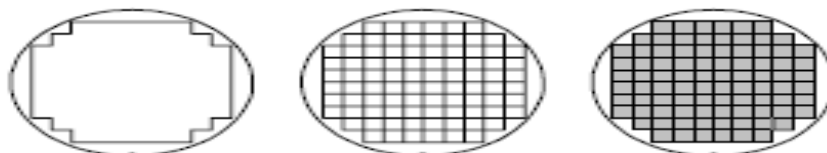
רוב הנתונים בסעיף זה מבוססים על [34] שבמקורות. תהליך הייצור נפתח בפרוסת סיליקון (צורן) גולמית המשמשת מצע ליצירת השבבים. יש בעולם כ- 4-5 מפעלים שבהם מיוצרים מוטות סיליקון גליליים ארוכים בקוטר של 20 או 30 ס"מ שאותם פורסים לפרוסות דקות בקוטר 20 או 30 ס"מ. על כל פרוסה מיוצרים בין 50 ל- 500 שבבים, תלוי בגודל השבב. " פרוסות ערומות " המכונות bare wafer מובאות כחומר גלם למפעלי הייצור (FABs) השונים.



איור 2 גליל סיליקון בצורתו הגולמית המובא ל-FAB

ישנם חמישה שלבים עיקריים בייצור מעגל משולב:

- הכנת ה-Wafer. הסיליקון מטוהר ומנוקה ונחתך ל-wafers.
- שבבים (micro chips) מיוצרים ב-Wafer FABs בתהליך שיתואר בהמשך פרק זה. תהליכים עיקריים בשלב זה הם עריכת השכבות המרכיבות את השבב, ניקוי, יצור התבניות המתאימות ו-doping (הגברת המוליכות של הסיליקון ע"י סוג של זיהום).



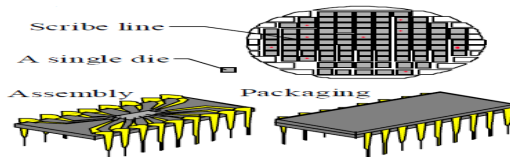
איור 3 ייצור השבבים על הפרוסה.

- בדיקת ומיון של ה-Wafer. כל die (שבב) ספציפי ובצורה אינדיווידואלית נבחן ונבדק בבחינות אלקטרוניות כדי למיין את השבבים הטובים והפסולים.



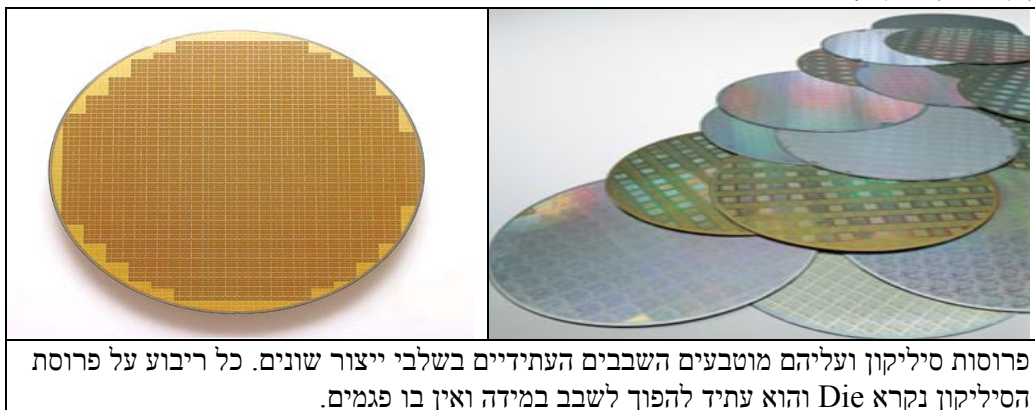
איור 4 בדיקה ומיון של כל die.

- הרכבה ואריזה. כל die מורכב לתוך אריזתו האלקטרונית. ה-wafer נחתך לאורך קווים המפרידים בין כל שבב Scribe lines כדי להפריד כל Die. חיבור מתכתי מתבצע והשבב נארז.



איור 5 חיתוך ואריזה

- בדיקה אלקטרונית סופית שעובר כל IC כשהוא כבר ארוז באריזתו הסופית ומבטיחה שהשבב עובר את הבדיקה האלקטרונית והסביבתית.
- השבב הסופי הוא תוצאה של תהליך מסובך שיתואר בפרק זה וכולל מספר לא קטן של שכבות כך שכל שכבה מיוצרת בתהליך אחר ולפעמים עם כלים שונים. השכבות השונות המרכיבות את השבב מופרדות בחומר מבודד ורק הנקודות הדרושות חיבור בין שכבה לשכבה מחוברות דרך חיבורים מיוחדים הנקראים via.
- ההבדל בין השכבות לרוב הוא בשלב הליתוגרפיה שיתואר בהמשך בכך שבכל שכבה משתמשים במסכה שונה (שגם תפקידה יוסבר) המתארת את המסלולים המיועדים לאותה שכבה בלבד או את נקודות החיבור בין שכבה נוכחית לזו שמתחתיה.
- בכל שכבה יוטבעו רק המסלולים שלה שמתחברים רק בנקודות המגע הקבועות מראש לשכבה שמתחתיה וכך שלב אחר שלב גדלים ומיוצרים השבבים על הפרוסה.
- תהליך זה אורך במוצא חודש עד שהשבבים על הפרוסה מוכנים לחיתוך ואריזה כדי להפוך לשבבים המוכנים לשימוש.



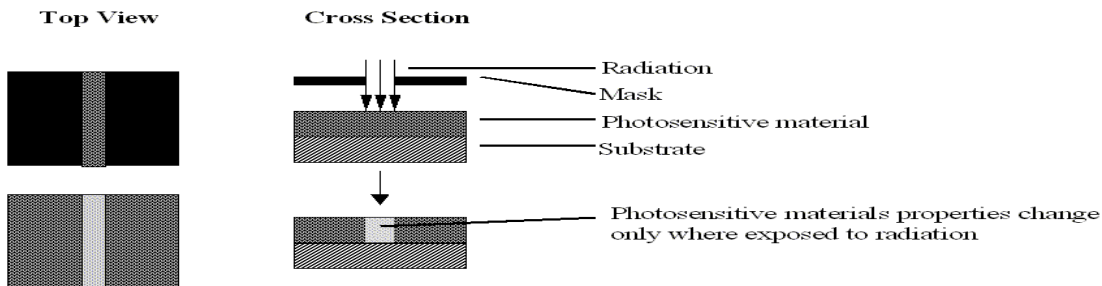
פרוסות סיליקון ועליהם מוטבעים השבבים העתידיים בשלבי ייצור שונים. כל ריבוע על פרוסת הסיליקון נקרא Die והוא עתיד להפוך לשבב במידה ואין בו פגמים.

איור 6 דוגמא ל-Wafers שונים

1.1.1 חפירת התעלות

חפירת תעלות ננו-מטריות היא שלב מרכזי בייצור שבבי סיליקון (micro chips) שהם לב ליבו של המעבד, רכיב הזיכרון ו/או כל רכיב חכם אחר במכשירים המודרניים.

כאמור על שבב אופייני מקובצים בין עשרות אלפים למיליוני רכיבים זעירים שמחוברים בניהם על ידי מספר דומה של חוטים זעירים שכמובן אין אפשרות להחזיקם באוויר הפתוח כי הם דקים ושבריים מדי. לכן מטביעים אותם בתוך מצע של חומר מבודד שגם עוזר לשמר אותם מפגיעה מכנית וגם מאפשר את ייצורם. דבר זה נעשה בעזרת התעלות הננו-מטריות. לוקחים את פרוסת הסיליקון שעליה רוצים לייצר את החוטים ומצפים אותה בשכבת חומר מבודד שבו בשלב מאוחר יותר חופרים תעלות ננו-מטריות במקומות שבהם רוצים שהחוטים יעברו. אחרי ה"חפירה" מצפים את הפרוסה בשכבה של מתכת (אלומיניום או נחושת). על רוב הפרוסה נוצרת שכבה מתכת דקה ורק התעלות מתמלאות מתכת ושם השכבה עבה יותר. אחר כך באמצעים מכניים וכימיים מלטשים את הפרוסה, מורידים ממנה את שכבת המתכת הדקה וכך נשארת מתכת רק בתוך התעלות. ניתן לתאר את התעלות כמוליכים של השכבה הנ"ל בפרוסת הסיליקון. כדי ליצור את התעלות משתמשים בחומצה כדי לאכל את החומר ולחפור את התעלה. כדי שהחומצה תאכל רק את המקומות המיועדים לתעלות משתמשים ב**תהליך הליתוגרפיה** המתואר להלן. ראשית מורחים את הפרוסה בחומר שמזכיר פילם (של מצלמה) שמשנה את תכונותיו הכימיות בתלות באור. לאחר מכן מקרינים עליו אור דרך מסיכה שמכילה תרשים של התעלות (כזו שחוסמת אור בכל מקום חוץ מקווים שקופים באזורי התעלות). ואז במקום שבו פוגע האור משתנה החומר על הפרוסה והופך רגיש לחומצה בעוד שבשאר הפרוסה הוא עמיד לה. לאחר ביצוע הליתוגרפיה אפשר לבצע את חפירת התעלות כלומר את איכול המקומות הרגישים לחומצה בלבד.



איור 7 תיאור עקרוני של תהליך הליתוגרפיה

1.1.2 המסכה

כאמור בתהליך ייצור השבבים משתמשים במסכות בתהליך הליתוגרפיה. בעזרת המסכה והחומר הרגיש לאור נקבעת צורת הרכיבים על הפרוסה. חשיבות רבה יש לעובדה שהמסכה תהיה מדויקת שאם לא כאשר נקריין דרכה את האור התבנית על פרוסת הסיליקון תצא שגויה והתעלות לא ייצרו במקום ובצורה הנכונים. כשם שהרכיבים שעל השבב מיוצרים כולם יחד במהלך ייצור השבב כך גם השבבים עצמם מיוצרים בו זמנית בכמות גדולה על פרוסה אחת של סיליקון. מרבית שלבי הייצור: פיזור החומר המקבע, שפיכת החומצה, הוספת המתכת והליטוש להסרת עודפי המתכת נעשים בו זמנית על הפרוסה כולה ובכך היתרון העיקרי של ייצור הרבה שבבים על פרוסה אחת. אך השלב הקריטי של הליתוגרפיה נעשה על כל שבב או על קבוצות קטנות של שבבים בנפרד. המכשיר לביצוע התהליך נקרא סטפר (stepper). בפעולתו הוא קודם מתיישר על הפרוסה כך שהצירים שלו מקבילים לחלוטין לצירי השבבים (שהם תמיד מלבניים) ואז, כשכבר הוכנסה לתוכו המסכה, הוא מתמקד בשבב הראשון ולרגע נדלק הבזק אור המוקרן דרך המסכה על השבב. אחר כך כבה האור והסטפר זז שבב אחד ימינה או שמאלה או למטה או למעלה וחוזר חלילה. עד שהוא מסיים את טיולו על הפרוסה כל השבבים הוקרנו ושלב הליתוגרפיה הסתיים. כעת עוברת הפרוסה לשלב הבא בייצור – הוספת החומר המקבע ולמכונת הליתוגרפיה מוכנסת הפרוסה הבאה. מכיוון שהמסכה צריכה להיות בדיוק רב מאוד והיא מכילה פרטים בגודל כלל העיצוב ברור שיש בעיה טכנולוגית ליצור אותה מכיוון שהיא מכילה כביכול פרטים קטנים ברמת הפרטים שיש בשבב כלומר המסכה מכילה פרטים בגודל כלל העיצוב וכדי להשיג זאת האור העובר דרך המסכה עובר גם דרך עדשה מרכזת כך שהמסכה יכולה להיות גדולה בהרבה מה"דמות" הסופית שנרצה להקריין על הפרוסה. כמובן שכלל שהמסכה גדולה יותר קל לייצר אותה. למרות השימוש באופטיקה עדיין ייצור המסכה מכיל פרטים קטנים שגם אותם קשה לייצר במדויק. לכן משתמשים בייצור המסכה במספר שלבים של "הקטנה בעזרת האור" עד שמגיעים ממסכה שאפשר לייצר בעזרת כלים מכניים עדינים למסכה מספיק קטנה לשימוש

בסטפר. כדי לחסוך זמן יקר בביצוע תהליך הייצור משתמשים בעדשות המייצרות מספר עותקים עבור התרשים וכך בעצם נוצרת הקרנה של משפר שבבים בצילום אחד.

1.1.3 שלבים כימיים וטכניקות בתהליך ייצור השבב

ככלל ניתן להגיד שבתהליך ייצור השבבים יש נושא מרכזי החוזר ומופיע בווריאציות שונות שוב ושוב עד להשלמת התהליך והוא יצירת השבב. ככלל כל וריאציה של התהליכים מבוססת על השלבים הבאים:

- השקעת חומר המטרה (deposition)
 - ליטוש לסילוק החומר המיותר מחוץ ל"תעלות" (CMP).
 - מריחת החומר הרגיש לאור (photo resist spin-on).
 - ליתוגרפיה (lithography).
 - קיבוע בחומר האמיד לאור (process photo resist)
 - החריטה (Etching) – תהליך האיכול ע"י החומצה.
- השקעת חומר המטרה כמו גם מריחת החומר הרגיש לאור, נעשית בטמפרטורה כזו שהחומר המיועד לפיזור הוא נוזלי כך שניתן לשפוך אותו במרכז הפרוסה ועל ידי סיבובה הוא מתפזר לשוליים בדומה לצנטריפוגה. במהלך הפיזור מתקרר החומר ובסיומו הוא כבר חוזר למצב מוצק וניתן להוציא את הפרוסה. יש לוודא שהפרוסה לא תסתובב מהר או לאט מדי, כי אז הפיזור לא יהיה אחיד ונקבל עודף חומר במרכז (בגלל סיבוב איטי מדי) או בשוליים (בגלל סיבוב מהיר מדי). למרבה המזל ברוב שלבי התהליך מידה קטנה של עודף או חוסר לא תפגום באיכות הסופית של המוצר.
- תהליך הקיבוע דומה לפיתוח פילם (השריית הפרוסה בחומר הכימי המתאים). בתהליך החריטה טובלים את הפרוסה בחומצה בדיוק לזמן המתאים ושטיפים היטב. כאן הדיוק חשוב מאוד כי אם לא ניתן לחומצה מספיק זמן לעבוד, התעלות לא תיחפרנה עד לעומק הנדרש (דבר שעלול לגרום לנתק בקו) מצד שני אם נשאיר את הפרוסה בחומצה זמן רב מדי, התעלות תיחפרנה יותר מדי וזה יכול להתבטא בקצר חשמלי כתוצאה ממגע בין שני קווים שאמורים היו להיות מבודדים זה מזה.
- שלב הליטוש דומה לליטוש של משטחים רגילים בעולם התעשייתי. מברשות הכוללות ראשי ליטוש שגודלם בערך כגודל הפרוסה (20-30 ס"מ) יורדים עליה, מסתובבים במהירות ומסלקים את עודף החומר. כדי שהתהליך יתבצע ביעילות מוזרם על הפרוסה חומר כימי מתאים המסייע בתהליך ולכן פעולה זו נקראת "ליטוש מכני-כימי" (CMP-chemical mechanical polishing).
- סביבת העבודה של תהליך ייצור השבבים קרויה "חדר נקי" או "אזור נקי" ומתאפיינת ברמת ניקיון גבוהה. אסור שיימצא בה ולו חלקיק אבק אחד או שערה מראשו של אחד העובדים כי כל חלקיק כזה בנפילתו על הפרוסה גורם לשבב להיות פגום. בנוסף ליצירת המוליכים כל שבב בנוי ביחידה הנקראת טרנזיסטור שהוא מעין מתג אלקטרוני המהווה בסיס למימוש הפעולות הלוגיות המהווים אבני בניין לכל רכיב ספרתי. תהליך ייצור הטרנזיסטורים בשכבה מסוימת הוא וריאציה על תהליך ייצור המוליכים שתואר לעיל אבל נעשה בתת-תהליך נפרד כלומר מייצרים את המוליכים בשכבה מסוימת ולאחר מכן את הטרנזיסטורים המחברים בשכבה למוליכים אלו. לאחר מכן מכסים את השכבה הנ"ל בחומר מבודד למעט נקודות החיבור לשכבה הבאה וחוזר חלילה. פירוט על תהליך ייצור הטרנזיסטור ניתן למצוא בנספח 1.

1.2 תהליך בקרת האיכות בייצור השבבים

תהליך ייצור השבבים כפי שתואר בפסקה הקודמת מורכב ומסובך וחייב להיות מדויק מאוד. חבילת הפרוסות שיוצאות ממכונה אחת עוברות מיד למכונה הבאה בתהליך וחבילת פרוסות חדשה מובאת מיד למכונה הראשונה (בתהליך אוטומטי). לכל הפסקה בתהליך הייצור או פגיעה בו ישנם השלכות כלכליות נרחבות וברור שבתהליך זה ישנם תקלות שחייבות להיות מאותרות מוקדם ככל האפשר כדי למנוע השלכה של רכיבים רבים שיתגלו כתקולים בסוף התהליך.

כתוצאה מכך בדיקת שכבות הפרוסה השונות במהלך שלבי הייצור הינה קריטית על מנת להגיע לניצול ותפוקה גבוהים של תהליך הייצור. באמצעות בדיקת שכבות הפרוסה במהלך הייצור ניתן להשיג את הדברים הבאים:

- זיהוי תקלות במכונות הייצור השונות. במידה שמבצעים בדיקה שוטפת של השכבות לאחר כל שלב נוכל לזהות סוג מסוים של מכונת ייצור או לחלופין מכונה מסוימת שלא תפקדה כהלכה ולא לגרור את התקלה הזאת על פני פס הייצור אלא רק במקבץ התקול. לדוגמא נוכל לקחת את המכונה שמבצעת איכול ולראות אם התוצר ממנה תקין ובמידה שלא להרים "דגל אדום" שאין להמשיך עם מכונת הייצור התקולה שכן כל פס הייצור ממנה יהיה תקול.
 - שיפור התהליך – בעזרת בקרת האיכות על שכבות הפרוסה השונות ניתן להבין את התהליכים הכימיים שמתרחשים ולבדוק את מידת הצלחתם.
- למזלנו מרבית התקלות אינן קורות לכל השבבים על הפרוסה ביחד אלא מדי פעם לשבב בודד בפרוסה. אם למשל תקלה במערכת הסינון גורמת לכך שמדי פעם נופל חלקיק אבק על הפרוסה, אם כמות החלקיקים שהסתננו לחדר הנקי קטנה, ישתבשו כמה שבבים אבל מרביתם עדיין תקינים. או אם נשארה הפרוסה בחומצה מעט יותר מן הדרוש – פה ושם יתהוו קצרים אבל רק בכמה שבבים ולא בכולם בבת אחת. עובדה זו עוזרת לנו בכך שאפשר לסמוך על כך שגם בפרוסה תקולה יהיו הרבה שבבים תקינים שישמשו "דוגמא" ל"איך נראה שבב תקין" מכיוון שאין אפשרות לקבל מידע על איך נראה במערכת האופטית שבב תקין. הפיקוח על התהליך נעשה ע"י מספר מכונות בצורה איטרטיבית כדי לגלות פגמים שבב כאשר בעבודה זו נתמקד במכונות מסוג Wafer Inspection.

1.2.1 מכונת Wafer Inspection

כאמור מלאכת זיהוי הפגמים היא הפעולה הראשונה שצריך לבצע כדי לבדוק שתהליך ייצור פרוסות הסיליקון על שכבותיהן השונות תקין. מכונת הפיקוח (inspection) סורקת את כל הפרוסה (wafer) במהירות מרבית ואמורות להצביע על מיקומם של הפגמים השונים. רוב המכונות הקיימות בשוק היום מבוססות על מיקרוסקופ אופטי. מכונות אלו אמורות להיות אמינות, יחסית זולות ומהירות ועם זאת לספק דרגת רגישות (sensitivity) גבוהה ואמינות גבוהה כלומר יחס דיווח פגמים אמיתיים לעומת שיקרים גבוה. חסרונן של מכונות אלו הוא בכך שהן פחות מדויקות, כלומר הן מגלות שיש פגמים נותנות תיאור כללי שלו (שריטה, חלקיק וכו') אבל לא את מקור התקלה וללא סיווג עדין יותר של מהות הפגם.

1.2.2 מכונת SEM Defect Review

מכונה זו הנקראת מכונת אבחון (review) משתמשת לרוב במיקרוסקופ אלקטרוני המכונה SEM ולא נועדה לסקירה כללית. מהנדס התהליך בוחר חלק מייצג מהפגמים שהתגלו ע"י מכונת ה-inspection המהירה ומביטה עליהם בהגדלה גבוהה ובדיוק רב (עד 0.5nm לפיקסל).

1.2.3 מכונת Mask Inspection

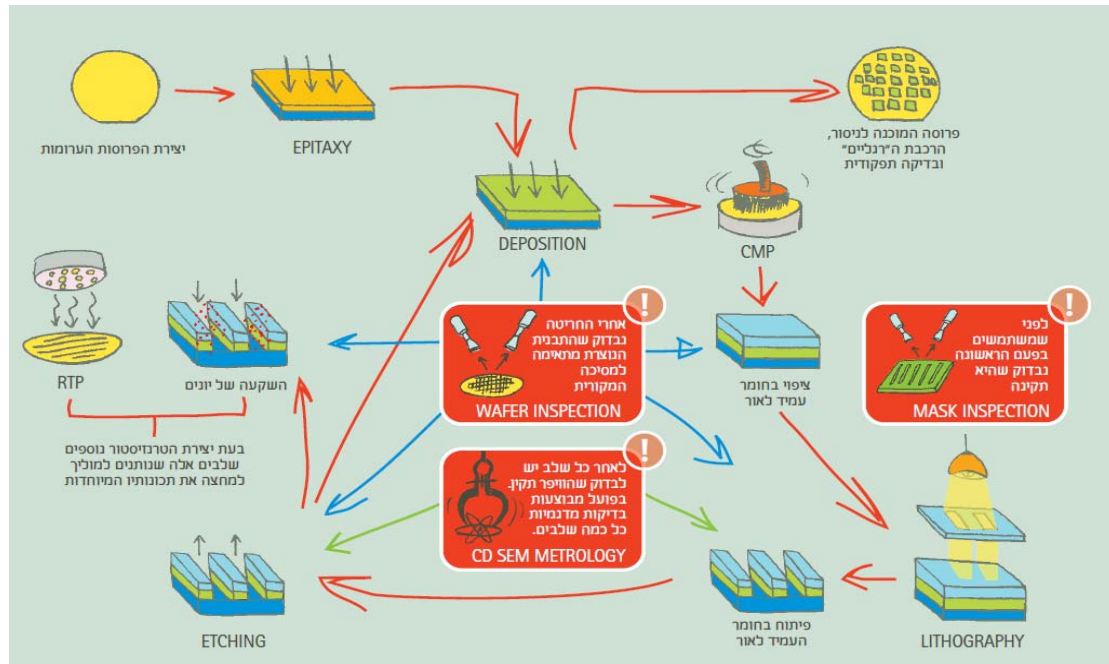
מכונה זו משמשת את ה-FAB לבדיקת תקינות של המסכה שתוארה בתת-פרקים הקודמים.

1.2.4 מכונת CD

כדי לתכנן ייצור מעגלים משולבים בעלי ביצועים גבוהים נדרש לשלוט בזהירות על הפרמטרים בתהליך הייצור. תכונות העובי של השכבות והמתכות חייבים להיות מדויקים, אחידים ומבוקרים. רוחבי הקווים חייבים להיות בגבולות דקים והמכשיר או הרכיב צריך להיות נטול פגמים המשפיעים על התפוקה. כדי לשמור על התהליך האיכות של השכבות הדקות נמדדת בקביעות ע"י מכונות הנקראות Critical (Dimension). למעשה מכונה זו היא וריאציה של מכונת ה-SEM Review.

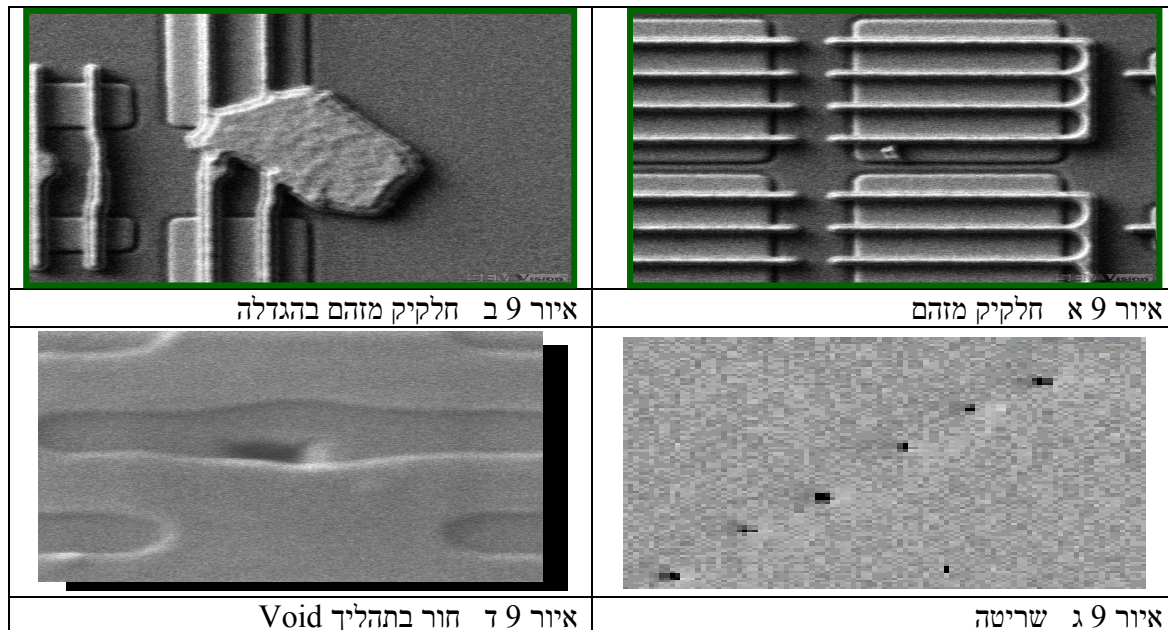
1.2.5 תרשים בקרת התהליך ודוגמאות לפגמים

ניתן לתאר את התהליך הכללי ב-FAB שבו נעשה שימוש במכונות לבקרת התהליך בעזרת התרשים הבא:



איור 8 תרשים גרפי של תהליך בקרת האיכות כתלות בתהליך הייצור

כל שלב בתהליכים ובתת-תהליכים מבוקר ונדגם אוטומטית בעזרת המערכת הממוחשבת ב-FAB. מכונות ה- Review, Inspection גם לשרתים מרכזיים המבצעים ניתוח וניהול התפוקות ב-FAB. האיור הבא מציג מספר תמונות של פגמים אופייניים שהתגלו על פרוסות במהלך הייצור ונדגמו ע"י מכונת SEM.



איור 9 א-ד דוגמאות לתמונות מפרוסות סיליקון עם פגמים

1.3 הפן הכלכלי

ההשקעה בפיתוח מכונות לצידוד בדיקה בכלל ופתרונות מחשוב לעיבוד התמונה בפרט היא עצומה. העלות של מכונות WI לדוגמא נעה בתחום של 3-6 מיליון דולר למכונה. בנוסף לכך גם עלותן של מכונות SEM DR ו-CD היא בסדר גודל של 1-4 מיליון דולר. עלותה של מכונת Mask Inspection אף גבוהה יותר ומגיעה לתחום של 10-20 מיליון דולר בגלל העלות הגבוהה של המכלולים האופטיים שלה. מצד שני שוק יצרני השבבים הינו שוק תחרותי מאוד ומשתתפות בו חברות ענקיות

כגון: Intel, IBM, Samsung, AMD, TSMC ועוד עשרות יצרניות משנה קטנות יותר. ההערכות הכלכליות מדברות על שוק שבין 20-40 מיליארד דולר לשנה עם עליות וירידות מחזוריות לאורך השנים. ההשקעה בציוד ובדיקה לשוק זה היא בסדר גודל של 1.7 מיליארד דולר בשנה בתחום ה-WI, קרוב ל-1 מיליארד דולר בשנה בתחום ה-SEM-DR ועוד קרוב לחצי מיליארד בשאר הנישיות שהוזכרו. כאמור זיהוי תקלות בשלבי הייצור השונים הוא קריטי לקבלת שבב תקין, מצד שני החברות משקיעות הון עתק בהקמה ותפעול מפעלי הייצור (סדר גודל של מיליארדים בודדים למפעל) ולכן לא קשה להבין את הדרישות המחמירות שיש ממכונות ה-WI שצריכות לגלות פגמים שונים שלעיתים הם קטנים וחמקמים אולם עלולים לגרום לכישלון של תהליך ייצור שבב מסוים. כשלון תהליך כזה (הנמשך חודשים בודדים בייצור מסיבי) עלול לגרום להפסדים אדירים למפעלי השבבים שכן הם מצפים להחזר השקעה מידי במכירת השבבים. בנוסף לכך שהמכונות צריכות לגלות פגמים קטנים שלעיתים נמצאים בגבולות היכולות האופטיות הקיימות כיום ובנוסף הם צריכות לעשות זאת במהירות המרבית כדי לשמש כחלק מחוג הבקרה בייצור השבבים ולא לעקב אותם.

שתי הדרישות הללו (רגישות לגילוי פגמים וגילוי מהיר שלהם כלומר היכולת לסרוק כמה שיותר wafers בשעה) סותרות אחת את השנייה אבל מציבות אתגרים רבים בפני המהנדסים המתכננים את המכונות אבל בנוסף לכך עליהם לדאוג לכך שמכונות אלו יהיו כלכליות עבור יצרני ציוד הבדיקה. מכל האמור לעיל ברור שמשימת בנית מכונות אלו הינה קשה ומסובכת כוללת אינטגרציה קשה בין מכלולים אופטיים, מכניים, אלקטרוניים וכן תכנון של מערכות מחשב מתקדמות ויעילות אולם לנוכח נתוני המכירות של שוק השבבים ברור שההשקעה העצומה במחקר ופיתוח של מכונות הנותנות פיתרונות מתאימים ליצרני שוק השבבים הינה כדאית כלכלית.

סיפור מעניין מובא בקישור הבא הקשור לכישלון בייצור מעבדים של חברת NVIDIA שגרם להפסדים

אדירים. [/http://hwzone.co.il/news/1217803368/72973](http://hwzone.co.il/news/1217803368/72973)

[/http://hwzone.co.il/news/1215115722/55414](http://hwzone.co.il/news/1215115722/55414)

2 מערכות לגילוי פגמים

פרק זה יתאר את העקרונות הפיזיקליים, הכימיים והמכניים של מכונות WI. הבנת עקרונות והאילוצים שמכונות אלו דורשות מהמתכננים חשובה לצורך הבנת צורת מימוש האלגוריתמים לזיהוי פגמים ותכנון ובנייה של מערכות עיבוד התמונה המתאימות למכונות אלו. חלקו הארי של המידע בפרק זה מבוסס על המאמרים [1],[2],[4],[5],[24],[26],[28],[31],[32] שבמקורות.

2.1 מבוא

מערכות WI עוזרות ליצרניות המוליכים למחצה להגדיל ולשמר את התפוקה של ייצור הרכיבים. המטרה העיקרית של מערכות WI היא לפקח על התהליך לבדוק שהוא תחת שליטה ואם יצא משליטה להצביע על מקור הבעיה. התכונות והמאפיינים החשובים ביותר של מכונות אלו הם רגישות לגילוי הפגמים (sensitivity) וקצב הסריקה המקסימלי (תפוקה או הספק מקסימלי) ($\text{throughput} = \text{TPT}$) הנקבע במספר wafer הנסרקים בשעה ($\text{wafer per hour} = \text{WPH}$). בעצם שני פרמטרים אלה קשורים אחד בשני באופן הבא כך שרגישות גדולה גורמת בדרך כלל ל-TPT נמוך. הסיבה לכך היא שבדרך כלל כדי לקבל רגישות גבוהה דרושים אלגוריתמים מתוחכמים הדורשים יותר פעולות חישוב, דבר שגורם להאטת קצב סריקת המכונה מכיוון שקצב כניסת המידע הוא גבוה מאוד ולכן תוספת קטנה של פעולות גורמות לנפח אדיר של תוספת חישוב. הפרקים הבאים יתמקדו בדרכי מימוש מיוחדות להפחתת זמן החישוב. בנוסף ישנם גם סיבות פיזיקליות וכלכליות לדואליות זו. החלק היחסי של רגישות ו-TPT תלוי גם בפונקציה של מערכת הפיקוח. ישנם 3 דרישות כלליות למערכות אלה:

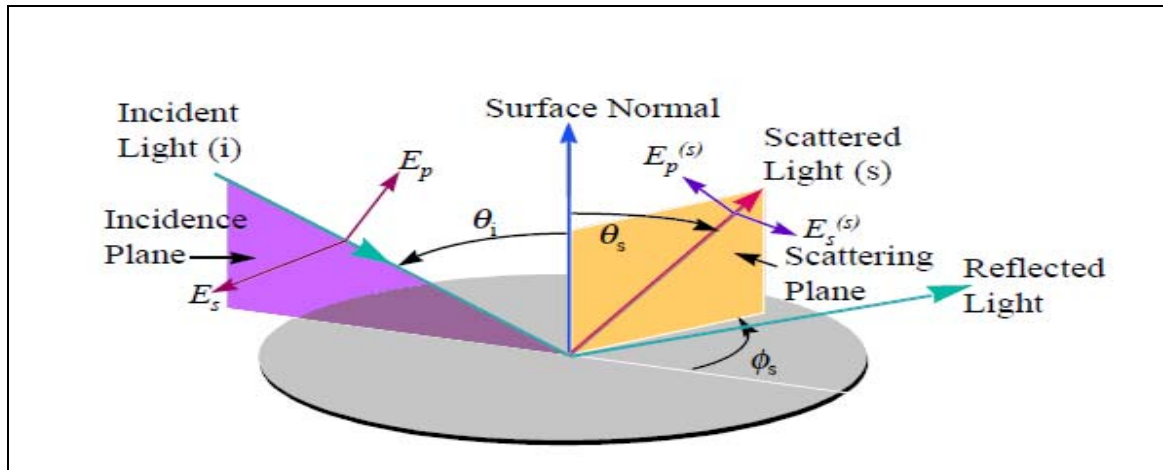
הראשונה הינה גילוי וסיווג פגמים בפיתוח התהליך, השנייה תהיה בהשגחה ופיקוח על קו הייצור של התהליך והאחרונה כתחנת מעקב. בפיתוח התהליך ניתן לקבל TPT נמוך כדי לתפוס ולגלות פגמים קטנים, ותחום רחב יותר של סוגי פגמים, אולם במעקב אחר קו ייצור או תחנה מחיר ההשקעה של יצרני המוליכים למחצה חייב לתת פריור ולא לעקב את התהליך הייצור. במקרה זה כמוכן הרגישות חייבת להיות מותאמת לכדי פגמים המגבילים את הייצור. היצרנים המובילים בעולם בפיתוח מכונות WI הם חברת AMAT וחברת KLA ו-Hitachi המציגות טכנולוגיות פיזיקליות הגורמות מחד ליתרונות אבל גם לחסרונות של כל מערכת בהשוואה לאחרת.

באופן כללי מערכת WI חייבת להיות מסוגלת לגלות נוכחות של פגמים ב-wafers ולאחר את מיקומם המרחבי. ניתן לתאר את בעיית מציאת הפגמים ב-wafer לבעיה שקולה שבה אנו מצלמים אצטדיון כדורגל מלמעלה ועלינו לאתר את המיקומים של כל הנמלות החיות על משטח הדשא באצטדיון. גילוי פגמים דורש מנגנון של הבחנת ניגודים (contrast mechanism) בין הפגם לסביבתו. מנגנונים אלה מתחברים לנושא של יצירת התמונה האופטית, פיזור האור איסוף אופטי שלו ועיבוד התמונה. כדי לגלות ולפקח על פרוסת הסיליקון, אנו משתמשים בהדמיה אופטית בכך שאנו מתבוננים על המערכת בעזרת גילוי הפוטונים שמפוזרים ע"י העצם שאותו אנו רוצים לבקר. הפוטון המתפזר הוא פונקציה של המיקום (תמונה) ואנו מקווים שהוא מכיל אינפורמציה הדרושה כדי לקבוע האם קיים פגם.

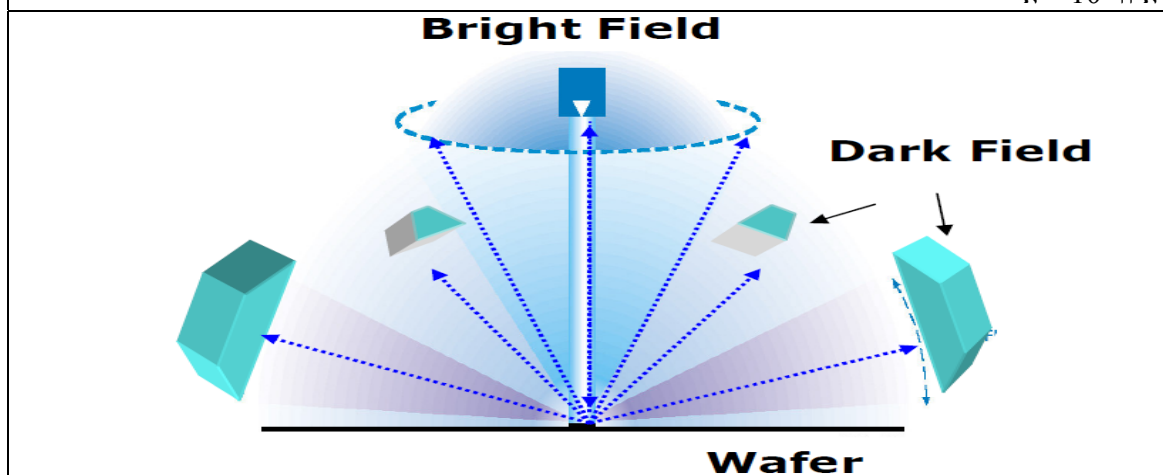
באופן בסיסי תהליך גילוי הפגמים כולל את השלבים הבאים: ראשית קבלה או רכישה של התמונה, שנית עיבוד התמונה ושלישית החלטה האם התמונה מכילה פגם או שלא. האתגר הגדול של מערכות ה-WI הוא לגלות פגמים בעזרת מערכות אופטיות הנחותות לעומת התהליך שבו הם נוצרו (תהליכי הליתוגרפיה נעשים בעזרת אופטיקה יקרה ומורכבת בהרבה) בהתייחס לרגישות ולגודל הפגם.

באופן בסיסי וגם ניתן לחלק את מכונות ה-WI למכונות DF (dark field) ומכונות BF (bright field) כאשר ההבדל בניהם מבוסס על זוויות הפגיעה ואיסוף של האור הפוגע במשטח (Wafer) והנאסף בהתאמה. כדי להבין את המבנה של התמונה האופטית המתקבלת נתאר את ה-wafer כמשטח המואר בזווית כלשהי בצורה אחידה. סוגי ההארה האפשריים הם הארה מהצד (יכול להיות ממקור או אחד או ממספר מקורות אור) או הארה מלמעלה.

תווי הפנים והפגמים במשטח מפזרים את האור לפי התכונות החומריות והטופוגרפיה שלהן. סדרה של עדשות לוכדות את האור החוזר (שבו זווית החזרה שווה לזווית הפגיעה) המסווג כאור BF (bright field), או את האור המתפזר המסווג כאור DF (dark field).



איור 10 – א



איור 10 – ב

איור 10 א-ב תיאור קואורדינטות והגדרות DF, BF

באיור 10-א זווית הפגיעה הינה מהצד ולכן זווית ההחזרה יחסית לאנך הנורמלי למשטח גם היא מהצד ואור זה נקרא BF לעומת איור 10-ב שבו ההארה היא מעל למשטח ולכן אור ה-BF הוא האור החוזר למעלה גם הוא. לעומת זאת אור ה-DF יכול להיות זה המתפזר בזוויות חדות. האור המוחזר או המתפזר מדמה את השינוי המרחבי של משטח ה-wafer לתוך חיישן שטח כגון מצלמות CCD, חיישן TDI או PMT. האינפורמציה המועברת ע"י התמונה נובעת מההבדלים בדרך שבה הפגמים והאלמנטים התבניתיים בפרוסה מפזרים או מחזירים את האור. הרזולוציה של מערכת התמונה האופטית נקבעת לפי גודל הפיקסל של חיישן השטח, הספקטרום של מקור האור, עוצמת הפוטון של המשטח המואר והניגודיות האופטית בין הפגם לסביבתו. יש לציין שבדרך כלל במערכות BF האור מוקרן מלמעלה בעוד שבמערכות DF הוא יכול להיות מלמעלה או מהצד או משניהם גם יחד.

2.1.1 אופן רכישת התמונה ועיבודה

רכישת התמונה היא כאמור השלב הראשון בתהליך ה-inspection. הוא כולל הארה של ה-wafer עם מקור אור (לייזר או מנורה), יצירת התמונה או איסוף האור המתפזר וגילוי אור זה ע"י גלאי TDI, PMT או CCD. על מקור האור להיות מספיק בהיר כדי לגרום לעוצמה מספיקה של פוטו-אלקטרון לעזוב את הגלאי כדי לייצר אות בעל יחס אות לרעש הניתן לעיבוד סביר (S/N signal to noise ratio). לצורך עבודה זו המתמקדת על מכונות מבוססות BF נדגיש כי קיימות שתי גישות מובילות אצל שתי החברות המובילות שתוארו. האחת טוענת שאור רחב פס ממנורת arc מתאים יותר לגילוי עבור מערכות BF והשנייה טוענת שדווקא אור לייזר קוהרנטי עדיף (בסעיפים הבאים אתן על כך את הדעת).

באופן עקרוני לכל מערכות ה-BF (גם אלו המאירות במנורת arc וגם אלו המאירות בלייזר) ישנה תופעה פיזיקלית המפריעה לגילוי ומשבשת אותו ומהווה אתגר גדול למתכנני האלגוריתמים לגילוי פגמים במכונה זו. תופעה זו נקראת Color Variation ותוצאתה היא שינויי צבע הרקע לאורך ה dies ב-wafer. המקור של הבדלים אלו הוא שישנם חוזרים של קרניים משכבות תחתונות לשכבה הנסרכת ובנוסף העובי של הפרוסות אינו זהה, דבר הגורם להבדלי צבע בין die ל-die שכן תופעה זו מקשה מאוד על גילוי הפגמים מכיוון שהיא יכולה אובייקטיבית ליצור הבדלים הנראים כפגם למרות שאינם כאלו (לדיון מעמיק בנושא זה ובבעיות נוספות במערכות BF [ראה נספח 2](#)).

לאחר רכישת התמונה, מערכת עיבוד התמונה צריכה לקבוע את נוכחותו של פגם בתמונה. מציאת הפגם וגילוי צרכים להיעשות כמעט במהירות הסריקה של ה-wafer מכיוון שבדרך כלל אין במערכת עיבוד התמונה מספיק זיכרון לאחסון כל תמונות סריקה ה-slice וגם אם ישנו מספיק מקום נרצה שקצב העיבוד יהיה מהיר כדי לא לעקב את קצב הסריקה. במערכות ללא תבנית (un-patterned) האלגוריתמים יתבססו על קביעת ספים לתמונות הפרשים. אולם במערכות עם תבנית (patterned wafer) נדרש אלגוריתם מבוסס השוואת die-to-die או cell-to-cell (עבור רכיבי SRAM, DRAM). המהירות והמחיר של מערכות עיבוד התמונה הם קריטיים לצורך מימוש האלגוריתמים הדרושים לעיבוד התמונה בזמן הסריקה הדרוש. למרות השיפור הקבוע ביכולת המעבדים (יותר MIPS) משימת תכנון ובניית מערכת עיבוד התמונה הינה מורכבת ומסובכת הן מכיוון שצריך למצוא דרך יעילה להזין את מערכת עיבוד התמונה במידע המתקבל מהגלאים ובנוסף משימת התכנות ומימוש האלגוריתמים הופכת לקשה ומסובכת יותר במערכת מרובת מעבדים שבה צריך להפיק מכל מעבד את המקסימום האפשרי כדי לעמוד בקצבי הסריקה של מערכת ה-WI. כאמור מערכות ה-BF הנוכחיות משמשות למציאת פגמים קטנים ומבוססים כיום על אורכי גל מסוג DUV מסוגלות לזהות פגמים בגודל מינימלי של 30 ננומטר ומשמשות ככלי הפיקוח המתקדם ביותר עבור היצרנים.

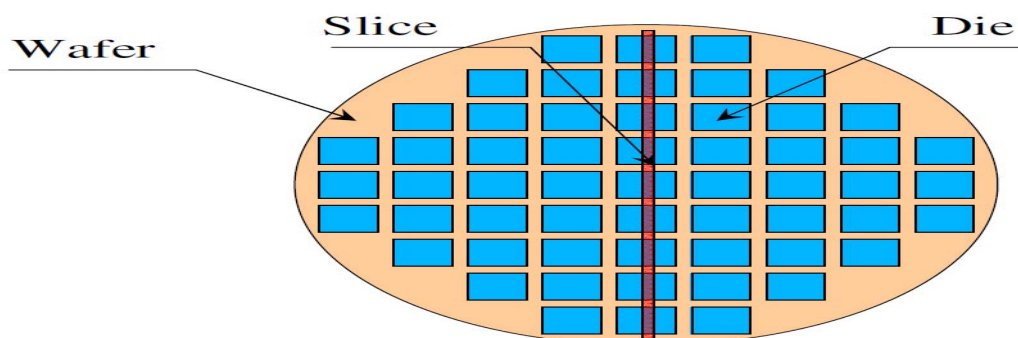
3 מודל כללי עבור מערכת עיבוד תמונה במערכות inspection

בפרק נסקור את התפתחות מערכות עיבוד תמונה למערכות WI, נציג את הדרישות ההכרחיות למערכות אלה בהתייחס לכוח מחשוב ויחס מחיר לכוח מחשוב (cost/performance ratio), ארכיטקטורות מחשוב רלוונטיות, ובחירה של טכנולוגיה מתאימה או שילוב של יותר מטכנולוגיה אחת ליצירת מודל מתאים למערכת עיבוד תמונה למכונות WI. בנוסף נתאר את אופן הכללי של מימושי האלגוריתם בליבה המהווה את הבסיס ליחידות החישוב. למבוא וסקירה היסטורית של מערכות עיבוד התמונה והאבולוציה שלהם וטכנולוגיית חישוב אפשריות כבסיס למערכת עיבוד התמונה ראה [נספח 3](#). הדיון בפרק זה מבוסס על המאמרים [1], [2], [3], [4], [5], [13], [14], [15], [22], [27], [28] שבמקורות שבביבליוגרפיה.

3.1 הדרישות ממערכת עיבוד תמונה עבור WI

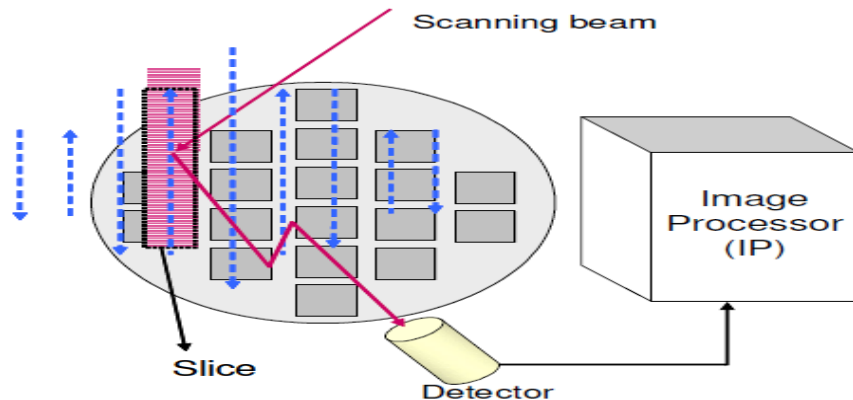
כאמור עבודה זו תתמקד במכונות inspection מסוג BF. הדרישות ממערכת עיבוד התמונה הם כדלקמן:

- מערכת עיבוד התמונה צריכה להיות בעלת כוח עיבוד מספיק לביצוע האלגוריתמים הדרושים לצורך גילוי הפגמים בלי לפגוע או לעקב את קצב הסריקה של המכונה מכיוון שאם לא כן היא תפגע ב-TPT המקסימאלי האפשרי.
- קצבי סריקה התנועתי של מערכות אופטיות/מכאניות עבור גדלי פיקסל שונים עבור מערכות WI מסוג BF הם מחצי WPH לפיקסל הקטנים ביותר (עשרות בודדות של ננומטרים) ועד 8 WPH לפיקסל הגדולים ביותר (מאות בודדות של ננומטרים).
- מערכת עיבוד התמונה צריכה להיות דטרמיניסטית כך שקצב הסריקה כמעט שאינו מושפע מכמות הפגמים. כלומר עד סף מסוים של פגמים מקסימאלי מערכת עיבוד התמונה לא צריכה להאט את סריקת המכונה ולא להיות מושפעת מכך.
- תהליך רכישת התמונה (data grabbing) הינו חלק אינטגרלי ממערכת עיבוד התמונה.
- המחיר של מערכת עיבוד התמונה לא יעלה על סדר גודל של \$100K. יש לציין שמחיר זה לא כולל רק את העלות הפיזית של מערכת עיבוד התמונה אלא גם את נושא אחזקתה ובעיקר את עלות הקירור שלה.
- עיבוד המידע הבסיסי יתבצע על מידע גולמי של פיקסלים בגודל בית כלומר 8 סיביות. עיבוד המידע יהיה ביחידות fixed point ואין צורך לעבד מידע מסוג floating point או double.
- מצד שני רוב עיבוד המידע נעשה על כל פיקסל ופיקסל המגיע למערכת כלומר אין כמעט data reduction למעט השלב האחרון שבו שולחים את רשימות הפגמים למחשב המרכזי. כמות הפגמים היא מסדר גודל של פרומיל מכמות המידע הנכנסת למערכת.
- קצבי הנתונים הנכנסים למערכת הם סדר גודל של עד 20Gbyte/sec ומערכת עיבוד התמונה כוללת את המכלולים המאפשרים לקלוט נתונים ממערכות איסוף נתונים בפורמט ידוע (כגון Camera Link או דומה לו). קצב ההוצאה של הנתונים ממערכת עיבוד התמונה (רשימות ותמונות פגמים) הוא בסדר גודל של פרומיל מגודל המידע הנכנס למערכת.
- תהליך סריקת פרוסות הסיליקון הוא כדלקמן: המערכת מצלמת כל פעם חלק מתוך ה die כאשר חלק זה נקרא slice.



איור 11 תיאור גרפי של המושגים slice, die ו-wafer.

במהלך סריקת המכונה קרן הלייזר סורקת את כל שטח ה-wafer ועושה זאת slice אחר slice. גלאי ה-PMT (או TDI) או חיישני CCD אוספים את האור המוחזר וממירים אותו לאות חשמלי שמומר לאות ספרתי בתחום של 0-255 (פיקסל).



איור 12 תיאור גרפי של סריקת ה-slices ב-wafer.

- מערכת עיבוד התמונה צריכה להיות ניתנת לשינוי במידה כזו שהוספת יחידות חישוב מעלה את כוח העיבוד ומאפשרת הגדלת ה-TPT עבור מכונות קיימות ושמירת TPT עבור מכונות חדשות.
- אריכות חיים של מערכת עיבוד התמונה (longevity) היא 3-5 שנים.
- מערכת עיבוד התמונה צריכה לתפוס מקום מינימלי כדי להקטין את הנפח אותו תופסת מכונת ה-WI.
- מכיוון שהמערכת תהיה מורכבת ממספר גדול של יחידות חישוב עיקרון מנחה בתכנון המערכת יבוסס על כך שכל יחידת חישוב מעבדת חלק מהתמונה והתוכנה הרצה בכל יחידת חישוב זהה ברמה העקרונית בין כל יחידות החישוב.

3.2 סקירת הדרישות מארכיטקטורות חישוב למערכות עיבוד תמונה

מימוש מערכת עיבוד תמונה למערכות WI דורש מערכת מחשוב בעלת כוח עיבוד המסוגלת לעבד סדר גודל של עשרות ג'יגה בית לשנייה. כמובן שעוצמת המערכת הנדרשת תלויה במורכבות האלגוריתמים. בהנחה שמימוש האלגוריתמים כולל מספר שלבים שיתוארו בהמשך, ההנחה הבסיסית היא שכל שלב כולל מעבר על תמונת מקור ותמונות ביניים שונות בגודל תמונת המקור ושכל שלב צריך לבצע מעבר על כל הפיקסלים בתמונה. הפעולות המבוצעות על כל פיקסל מתמונת המקור ותמונות הביניים כוללות מכפלות, השוואות מציאת מינימום, מקסימום ועוד. פעולות אלו מאפיינות אלגוריתמים לעיבוד תמונה. כדי לממש פעולות אלה צריך יחידות חישוב או מעבדים שמסוגלים לבצע פעולות אלו בצורה יעילה. קיימות מספר טכנולוגיות המאפשרות לבצע פעולות חישוב מסוג זה בצורה יעילה. הרעיון הבסיסי המאפשר לבצע פעולות אלה בצורה יעילה הוא ריבוי מעבדים או יחידות חישוב מאותו סוג. בנוסף לכך כל יחידת עיבוד מאפשרת לבצע פעולות וקטוריות על מספר פיקסלים בזמנית ובכך להאיץ את מהירות החישוב. בבחירת פלטפורמת החישוב עלינו להביא בחשבון את יכולת החישוב הנדרשת וכן את היחס עלות לביצועים של המערכת. כלומר אנו צריכים לבחור מערכת כזו שתהיה יעילה מבחינת אופן המימוש של האלגוריתמים וכן שהעלות שלה לא תעלה על אחוז מסוים מעלות ייצור המכונה, כלומר שמערכת עיבוד התמונה תהיה בעלות שהיא בסדר גודל של \$100K-\$120K. יש לציין שתכנון ובחירת ארכיטקטורה מתאימה למכונת WI מבוסס על בחירת טכנולוגיה בעלת ביצועים גבוהים (high performance computing) המתאימה ביותר להשגת הביצועים המתאימים (עבור סוגי האלגוריתמים הממומשים ומתאימים ל-WI) ומצד שני שתיתן יחס cost/performance מתאים כדי שהמערכת תהיה כלכלית וניתן יהיה לייצרה. יש לקחת בחשבון שמכיוון שבחירת הטכנולוגיה נעשית במתודולוגיה של רכיבה אחר גלים גלובליים של טכנולוגיות מחשוב (שלא תוכננו במיוחד עבור מערכות WI) ונדרשות התאמות של טכנולוגיות off the shelf למערכות WI. במקרים מסוימים התאמות אלו (כגון הכנסת המידע למערכת) עלולות לייקר את המערכת בצורה משמעותית.

קיימות טכנולוגיות וארכיטקטורות שונות כדוגמת FPGA, GPUs, IBM Cell Intel Multi-Core המהוות בסיס אפשרי לפלטפורמות מחשוב למערכות עיבוד תמונה. לסקירה על כל אחת מהאפשרויות ראה את הפרקים המתאימים [\(בנספח-3\)](#) [\(נספח Intel\)](#), [\(נספח IBM\)](#), [\(נספח FPGA\)](#), [\(נספח GPU\)](#).

3.2.1 ארכיטקטורת מערכת עיבוד תמונה מבוססת DSP

מעבדי DSP הם מעבדים המתוכננים לביצועים אופטימאליים עבור אפליקציות עיבוד תמונה מורכבות. בדרך כלל תדר העבודה שלהם יהיה קטן בסדר גודל של פעמיים עד שלוש ממעבד כללי כדוגמת אלה של Intel או AMD. היתרון היחסי של מעבדים אלה בהתייחס למימוש מערכת עיבוד תמונה מתבטא בעובדות הבאות:

- צריכת הספק נמוכה יחסית לתדר העבודה והביצועים שניתן להפיק ממעבד זה.
- יעילות ואחוזי ניצול גבוהים של חומרת המעבד להשגת ביצועים מרביים. דבר זה נובע מכך שיש שליטה כמעט מקסימלית על תצורת פעולת הזיכרון של המעבד בנוסף לזיכרון cache ברמות L1, L2 ישנם גם זיכרונות SRAM פנימיים וכן בקר DMA המאפשר שינוע של נתונים (data) מהזיכרון החיצוני אל הזיכרונות הפנימיים ולהיפך בצורה היעילה ביותר.
- מבוסס על טכנולוגיית VLIW שמצד אחד מעניקה אפשרויות לעיבוד מקבילי מחד ואם זאת אין היא מסבכת את חומרת המעבד (כדוגמת Intel).
- בד"כ קיימים ממשקי IO המאפשרים את חיבור ה-DSP לממשקי הזנה עבור קלט או פלט. ממשקים אלה יכולים להיות מנוצלים כדי להטמיע את פעולת רכישת התמונה כחלק אינטגרלי ממערכת עיבוד התמונה.
- רוב ה-DSP תומכים בחיבור Ethernet Mac או ב-PCI. היתרון בכך שניתן לחבר מספר רב של מעבדים למחשב מרכזי בצורה נוחה וזולה יחסית.
- קיומו של ממשק חיבור עבור רכיבי DDR2. נתון זה בשילוב בקרי DMA שהם חלק אינטגרלי ממבנה ה-DSP מאפשר לממש פונקציות עיבוד תמונה מאוזנות כך שלא יוגבלו מבחינת פעולות IO לזיכרון וממנו או מבחינת עיבוד. נתונים אלה מאפשרים לבצע במקביל פעולות עיבוד ופעולות IO ובכך להשיג יעילות מרבית.
- קיומם של פקודות אסמבלר המותאמות במיוחד לביצוע פונקציות עיבוד תמונה.

בגלל מקדם היעילות הגבוה של ה-DSP במימוש אלגוריתמים לעיבוד תמונה נניח שכל השלבים במימוש

האלגוריתמים דורשים $60 \frac{\text{cycle}}{\text{pixel}}$ בממוצע (לעומת $100 \frac{\text{cycle}}{\text{pixel}}$ במעבדי Intel). נניח שתדר העבודה

של מעבד DSP הוא 800Mhz ונקבל שכל מעבד כזה מסוגל לעבד קצב של

$$\frac{800\text{Mhz}}{60\text{cyc}} \approx 13.33 \frac{\text{Mbyte}}{\text{Sec}}$$

בהתחשב בקצב המידע הנכנס למערכת נקבל שמספר הליבות (cores) הדרוש הוא

$$\frac{20\text{Gbyte/sec}}{13.33\text{Mbyte/sec}} \approx 1500\text{cores}$$

אין ספק שבהשוואה למספר מעבדי DSP הדרושים לעומת מערכות מבוססות [מעבדי Intel](#) נקבל שיש לנו מעט יותר מפי שניים מעבדים. אולם יש לקחת בחשבון שראשית אלו מעבדים זולים למדי עם צריכת הספק נמוכה לעין שיעור לעומת מעבדי Intel והם מתאימים ויעילים במיוחד לפעולות עיבוד תמונה המבוססות 8bit או 16bit.

נגדיר את מעבד ה-DSP בתוספת רכיבי ה-DDR2 המחוברים אליו כיחידת עיבוד בשם node. הספק של node כזה בממוצע יהיה 3-4 watt כך שניתן ללא קושי לפזר על blade board עשרות בודדות של רכיבים כאלה. בהנחה שניתן לבנות board שעליו עד 30 יחידות חישוב כאלה נקבל סדר גודל של 50 כרטיסים והעלות שלהם בתחום שהוגדר.

היתרון של מערכת זו המבוססת על מימוש מערכת עיבוד תמונה במודל של מערכת מבוססת Multi DSP blade server הוא בכך שהיא מאפשרת לרתום טכנולוגיה לצורך התאמתה למערכות WI. יחס ה-

cost/performance המתקבל הוא הגבוה ביותר ומאפשר עמידה בעלויות הייצור הכלכליות הנוקשות שהוגדרו. מצד שני רוב מעבדי ה-DSP העובדים בתדר שהצגנו אינם מאפשרים פעולות floating point ומסתפקים בפעולות fixed point בלבד עם דגש על פעולות ווקטוריות יעילות על בית או 16 סיביות. מצד שני, על מנת לממש את היעילות המרבית (המאפשרת להעריך את כל שלבי האלגוריתמים ב-

$60 \frac{\text{cycle}}{\text{pixel}}$) עלינו לכתוב קוד יעיל כך שינצל את הפקודות המיוחדות בטכנולוגיית VLIW וכן את

גמישות תצורות הזיכרון ופעולות בקר ה-DMA שמציע ה-DSP.

לסיכום העבודה מציגה מימוש על פלטפורמת ה-DSP לפי המאפיינים הבאים:

- ה-core הנבחר יהיה אחד מאלה הנמצאים במעבדי ה-DSP הנפוצים בשוק. לאחר סקירת שוק core הנבחר יהיה C64+ של TI כאשר המעבד יהיה TMS320DM6437 שהוא מעבד Low-Cost במחיר שבין \$20-\$30 אבל מצד שני הוא מאפשר להפיק ביצועים (המתאימים לפעולות המתבצעות ב-WI) המתקרבים ביעילותם לאלה של High או Medium Performance DSP.
 - חלוקת המידע בין ה-DSP תעשה מערכתית ע"י כל שכל DSP מקבל region of interest של אותו חלק מה-dies עבור כל ה-dies עם חפיפות מתאימות עבור פעולות הדורשות שוליים. כך יושג מקבול מערכתי ברמת חלוקת המידע (data) בין ה-DSP.
 - לפי האמור לעיל בסעיף הקודם כל DSP עובד וניגש רק לזיכרונות המקומיים שלו כלומר תצורת המערכת היא מסוג NUMA.
 - כל DSP מריץ באופן עקרוני את אותם שלבים באלגוריתמים ועובד רק על חלק אחד של ה-dies וכך מובטח load-balancing אוטומטי של מערכת עיבוד התמונה כלומר כל הליבות עסוקים במידה שווה.
 - העברת התוצאות למחשב המרכזי היא דרך Ethernet או PCI או ערוץ תקשורת אחר שעליו ניתן להעביר מידע מה-DSP. קצבי ההעברה למחשב המרכזי הם פרומיל מסדר גודל של המידע המעובד.
 - השאפה המרכזית בכל מימוש של פונקציות עיבוד תמונה במודל זה הוא לנצל את הזיכרונות הפנימיים הנמצאים בתוך ה-core (on chip memory) בכך שהמעבד יבצע את פעולות העיבוד על מידע הנמצא בזיכרונות אלה בלבד. מכיוון שהקיבולת של זיכרונות אלה הוא קטנה יעשה שימוש מרכזי בטכניקת של עבודה על מספר שורות בודדות מהתמונה ושימוש ב-DMA להבאה שורות עתידיות (שורות הבאות מהתמונה) ברקע. טכניקה זו הידועה כ-double buffering implementation משתמשת בעובדה שבעוד המעבד מעבד חלק מהמידע הוא יכול להעתיק ברקע תוך שימוש בבקר ה-DMA את השורות הבאות לעיבוד כך שבסיום העיבוד של השורות הנוכחיות השורות הבאות כבר מוכנות לעיבוד.
 - עיבוד המידע יעשה ע"י שימוש בפקודות המיוחדות המנצלות את יכולות ה-DSP ויכתבו באסמבלר או intrinsic.
- דוגמא לזרימת המידע בפונקציית עיבוד תמונה כלשהי המעבדת בלוק כלשהו תהיה כדלקמן: עיבוד על הבלוק יעשה ע"י חלוקתו לתת בלוקים המתאימים לעיבוד בתוך הזיכרון הפנימי של ה-DSP. כדי להימנע ממצב שהמעבד ימתין למידע שיגיע לזיכרון הפנימי נעזרים ב-DMA כדי להעתיק ברקע את השורות הבאות (מהתת הבלוק הבא שצריך לעבד) לעיבוד בזמן שמעבדים שורות נוכחיות (מהתת הבלוק הנוכחי). הפסידו קוד הבא מתאר את התהליך הנ"ל.

- Allocate Input Buffers A, B in DSP internal Memory
- Allocate Output Buffers A,B in DSP internal Memory
- Configure EDMA copy
- Start EDMA input copy
- Wait for EDMA input copy completion
- For all sub Blocks
- {
- Start EDMA input copy (next)
- process sub Block (current)
- Wait for EDMA output copy completion (previous)
- Start EDMA output copy (current)
- Wait for EDMA input copy completion (next)
- }
- Wait for EDMA output copy completion (last)

איור 13 תיאור פסידו קוד של double buffering implementation
עיבוד התת-בלוקים המתוארים יעשה תוף שימוש בפקודות המיוחדות של ה-DSP התומכות בפעולות
נפוצות המשמשות כאבני בניין באלגוריתמים לעיבוד תמונה ויודגמו באופן מפורט בפרק הבא.

3.2.1.1 שיקולים לבחירת ה- DSP

מלכתחילה בחרתי מעבדי DSP של חברת TI מבוססי על ליבה מסוג C64+ המתקדם של TI. הסיבה
לבחירת ליבה זו היא שהוא משיג ביצועים גבוהים ביותר כאשר פרמטר ההשוואה הוא מספר מגה פעולות
בשנייה. הוא כולל 8 יחידות פונקציונאליות שכל אחת יכולה לעבוד על 4 פיקסלים בגודל של בית כך
שמעשית מקבלים עבודה על וקטור באורך 32 בתיים. ישנו קו של מעבדים המוגדרים ב-low-cost
שמחיריהם נעים בין \$9-\$20 כך שהרעיון המרכזי המאפיין אותם הוא שהם מכילים כמות זיכרון L2
קטנה בגודל של 128Kb ומצד שני זיכרון L1 בגודל של 80Kb שניתן לעבוד עם רובו כ-L1-cache
או לחלק אותו כרצוננו ל-SRAM ו-cache. מכיוון שאנו רוצים לבחור מעבד שיתאים ככל האפשר
לדרישות שהוגדרו האופציה הכדאית ביותר היא הינה מעבד low-cost אבל שיפיק ביצועים לישומי WI
בסדר גודל של המעבדים היקרים יותר. אין כל יתרון בשימוש ברכיבי פריפריה שבתוך השבב אם איננו
משתמשים בהם. הבחירה הייתה על משפחת TMS320DM6437 המכילה את התכונות העיקריות
הבאות:

- מבוססת על TI C64+.
- תדר עבודה 600Mhz, 700Mhz.
- מחיר בסדר גודל של \$20.
- הספק עבודה מקסימאלי 3 וואט.
- כוללת ממשק לחיבור Video שיכול גם לקלוט raw-data (ממשק אפשרי לרכישת התמונה)
- זיכרון L2 בגודל 128Kb.
- זיכרון L1 בגודל 80Kb. גודל זיכרון זה מהווה יתרון משמעותי במימוש פונקציות עיבוד תמונה
ובעצם משדרג את יכולות ה-DSP למרות מחירו הנמוך יחסית.
- זיכרון L1P בגודל 32Kbyte.
- כולל יחידת עיבוד וידאו VPSS המסוגלת לבצע פעולות עיבוד שונות על מידע המגיע דרך ה-
video port אבל ניתן גם לשנע אליה מידע דרך EDMA.
- כניסת ויציאת VPBE ו-VPBE
- חיבור לזיכרון DDR2 בגודל של עד 256Mbyte עם קצב של
 $166Mhz * 8Byte = 1.3Gbyte/sec$
- קישוריות לערוצי PCI-33Mhz או Ethernet 100/10 Mbit/sec.
- ממשק Emif2.1 א-סינכרוני לחיבור רכיבי זיכרון ברוחב של 8bits לחיבור רכיבי זיכרון
איטיים.
- ממשקים לרכיבים טוריים I2C, SPI, Uart, MSBSP.

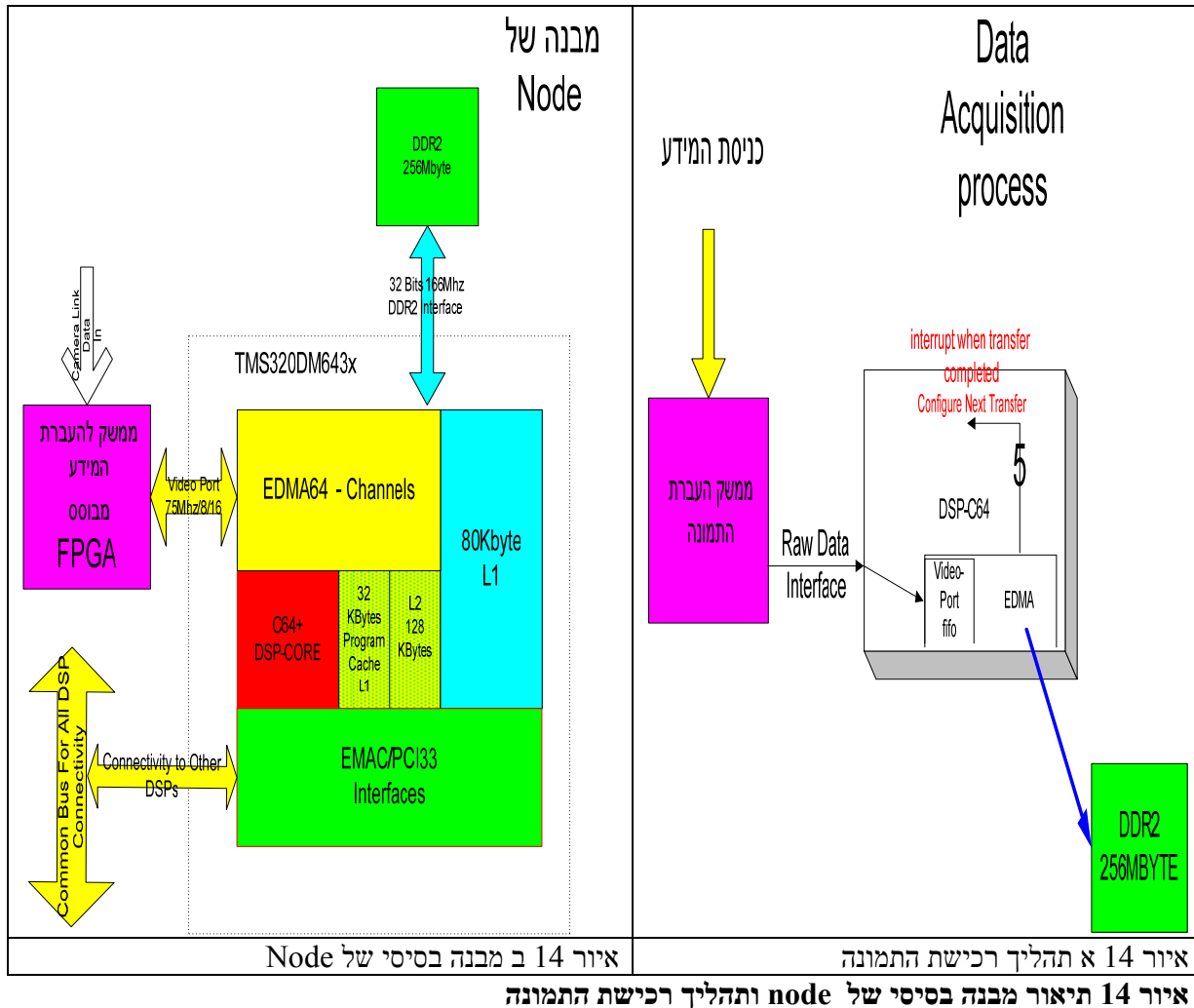
3.2.1.2 תיאור כללי של תצורת מערכת עיבוד התמונה מבוססת DSP

כאמור עבודה זו מציגה את המימושים של האלגוריתמים על מעבד DSP מסדרה TMS320DM6437. נגדיר את המושג node כיחידת עיבוד בסיסית הכוללת את ה-DSP, זיכרון, ערוץ להכנסת הנתונים למעבד כחלק מתהליך רכישת התמונה וזיכרון חיצוני DDR2 עם קיבולת של 256Mbyte. הערוץ שלתוכו ניתן לדחוף נתונים ל-DSP יכול להיות ערוץ ה-video-port שיעבוד בתצורה של raw data. ערוץ ה-video-port עובד בתדר של 75Mhz ומסוגל לקבל נתונים ברוחב 8bits או 16bits כך שסה"כ ניתן להזרים דרכו עד 150Mbyte/sec. בחישובים שנעשו בסעיף הקודם ראינו שכל DSP מתוכנן לעבד סדר גודל של 13-15 Mbyte/sec אבל למרות שאלו הנתונים המעובדים בפועל בכל DSP צריך לקחת בחשבון שכל DSP צריך לבצע עיבוד על מידע נוסף מכיוון שהאלגוריתמים דורשים שוליים שהולכים וקטנים במהלך שלבי ביצוע האלגוריתמים.

כלומר צריך להכניס לכל DSP יותר מידע כדי שבפועל הוא יוכל לעבד חלק מה-die. ההנחה שבמקרה המקסימאלי לכל DSP ידחפו עד פי שלושה מידע מסדר הגודל שהוא צריך לעבד לכן שימוש ב-video-port יספק רוחב פס מתאים כי לכל היותר נצטרך להזין כל מעבד ב- 50Mbyte/sec.

כאמור מערכת עיבוד התמונה תכיל סדר גודל של 1500 nodes כאשר כל מעבד DSP יחובר דרך ערוץ ה-video-port לממשק רכישת התמונה. בנוסף כל node יחובר דרך ערוץ ה-PCI או Ethernet (הכלולים בתוך המעבד) לערוץ משותף שבו כל ה-nodes מחוברים בצורת עץ למחשב מרכזי.

כאמור יש לכל DSP בקר DMA או בשמו המסחרי EDMA המסוגל לשנע מידע מתוך זיכרון ה-DDR2 ואליו וכן להתקני פריפריה כדוגמת ה-video-port. היתרון הגדול בבחירת ה-DSP הוא שתהליך הכנסת המידע ל-DSP נעשה ברקע ללא התערבות הליבה. ביסודו של דבר זהו תהליך המבוסס על אירועים (Events) (Event) הוא אות המגיע למעבד וגורם להפעלת העתקת EDMA ללא התערבות ה-core) המפעילים העתקות EDMA ברקע מה-video-port לתוך זיכרון ה-DDR2 של ה-node. כל אירוע (Event) יכול לסמן על שורה אחת או יותר שצריך להעתיק מהתור (FIFO) הפנימי של ה-video-port. לאחר מספר events כאלה הבלוק הגיע בשלמותו לזיכרון ה-DDR2 של ה-DSP ומתרחשת פסיקה (interrupt) שאותה יוזם ה-EDMA כדי להודיע שבלוק הגיע ונמצא בזיכרון ומוכן לתחילת העיבוד.



לסיכום יחידת העיבוד הבסיסית שעליה מושתתת מערכת עיבוד התמונה משלבת את טכנולוגיית ה-FPGA וה-DSP ליצירת ארכיטקטורה מתאימה עבור מערכות עיבוד תמונה ל-wafer inspection. למרות שמערכת עיבוד התמונה כוללת מספר רב של מעבדים ולכאורה כוח החישוב אמור להספיק למימוש האלגוריתם עדיין כתיבת קוד סדרתי רגיל עלול לגרום לביצועים נמוכים ולא יאפשר את מימוש המערכת. כדי ליעל את זמן ריצת האלגוריתמים בארבעה סדרי גודל (פי 10000 יחסית לקוד רגיל) מממשים את האלגוריתמים תוך שימוש בטכניקות עיבוד מקביליות הממומשות בשלוש רמות שונות כדלקמן:

- הרמה הראשונה מבצעת שימוש בבקר DMA כדי להעתיק פיסות מידע מהזיכרון החיצוני (שהינו זיכרון איטי מאוד) לזיכרון הפנימי במקביל לעיבוד שנעשה ע"י ה-DSP על פיסות מידע שהובאו בצעד הקודם. כך משיגים מיקבול בין עיבוד אלגוריתמי המתבצע בזיכרון מהיר אך קטן בקיבולתו לבין הבאת המידע מהזיכרון החיצוני.
 - הרמה השנייה מתבססת על העובדה שה-DSP מכיל 8 יחידות עצמאיות שכל אחת יכולה לבצע פעולה ללא תלות באחרת. כדי לנצל זאת משתמשים בטכניקת כתיבת קוד ב-pipe-line המאפשרת עבודה מקבילית בו זמנית בכל אחת מהיחידות.
 - הרמה השלישית של המיקבול נעשית ברמת הפיקסל. מכיוון שכל אחת משמונה היחידות של ה-DSP יכולה לעבוד על יותר מפיקסל אחד ניתן לכתוב קוד ולממש אלגוריתמים כך שכל שימוש ביחידת עיבוד של ה-DSP יעשה בצורה וקטורית כלומר כל פעולה תתבצע על מספר פיקסלים כגודל הווקטור של כל יחידת עיבוד.
- לתיאור מפורט של שיטות אלו ודוגמאות למימושים מקבלים של פעולות לעיבוד תמונה ראה [נספח 4](#).

4 אלגוריתמי השוואה כפולה מבוססים die2die למציאת פגמים

האלגוריתמים הבאים יתארו אפשרויות שונות לגילוי פגמים. להבנת אופן פעולת האלגוריתמים נדרש רקע בעקרונות עיבוד תמונה כגון דגימה, שחזור עבודה במרחב התמונה ובמרחב התדר, שיווי היסטוגרמות ועוד. לתיאור נושאים רלוונטיים בעיבוד תמונה הדרושים להבנת האלגוריתמים המתוארים [ראה נספח 5](#). המידע המתואר בפרק זה מבוסס על המאמרים

[1],[2],[3],[4],[24],[32],[31],[27],[13],[15],[22] שבביליוגרפיה.

חלוקת המידע בין המעבדים נעשית מראש כך שכל DSP מקבל יחידות מידע הממוקמים במיקומים זהים ב-dies הנמצאים ב-wafer. למרות שעבודה זו מתמקדת בגילוי פגמים בתמונות מחזוריות לכאורה ניתן היה להתבסס על העובדה שהמידע מחזורי ולבצע השוואה בתוך בלוק נתון מ-die נתון בלי להתייחס ל-die אחר. הסיבה לכך שלמרות שהמידע מחזורי לא ניתן להשוות אותו בתוך אותו die וחייבים להשתמש בהשוואה בין die ל-die היא קיומם לעתים של עיוותים בציר y ו x כתוצאה מגלאים לא סימטריים, דבר שגורם לכך שההשוואה בין אזורים דומים בתוך אותו die איננה אפשרית או קשה ומסובכת כי תיקון העיוותים עלול להיות מסובך עד כדי בלתי אפשרי. בנוסף לכך בפרוסות רבות קיימים אזורים מחזוריים לצד אזורים שאינם מחזוריים ולכן אזורים אלה (שאינם מחזוריים) חייבים להיבדק ע"פ עיקרון השוואה של die2die. הרעיון הבסיסי באלגוריתם המוצא פגמים בפרוסות סיליקון הוא כאמור למצוא עבור כל פיסת סיליקון הנסרכת אזור ייחוס (reference area) שאותו ניתן יהיה להשוות לאזור הנבדק. לכאורה איזור הייחוס מתאר משטח ללא פגמים.

השיטה הטובה ביותר היא להשתמש ב-die ייחוס נטול פגמים ולהשוותו ל-die הפגום. אבל אליה וקוץ בה איך ניתן לקבל die נקי מפגמים. התשובה לכך שאי אפשר לקבל כקלט למכונה die כזה והיא צריכה לנסות לחלץ אותו מהמידע המגיע מצילום ה-die.

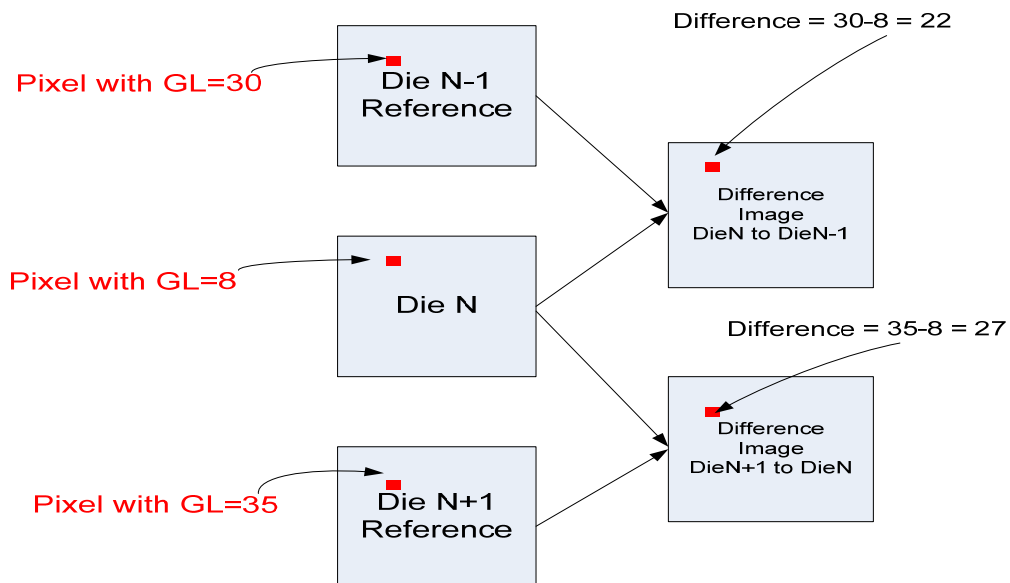
הסיבות לכך שלא ניתן לדעת איך נראה die תקין הם כדלקמן:

- התמונה המתקבלת מ-dies שונים עלולה להראות שונה בגלל סיבות שונות שחלקן תוארו לעיל כגון בעיית color variation שמשנה את ערכי הפיקסלים בתמונה או בגלל בעיית הזזת ההיסט (הזזת רגיסטרציה).
- יכולים להיות כמה סוגי גלאים המוזכרים בפרק 2. קיומם של גלאים שונים עלול ליצור מצב שהם רואים רק חלק מהפרטים ב-die כלומר אין הם רואים כל הפרטים ב-die אלא רק אלמנטים שטוחים או מרחביים או שילוב חלקי שלהם.
- גם אם באופן תיאורטי בלבד הבעיות שהוזכרו לעיל לא היו קיימות עדיין לא ניתן לדעת מצילום של die האם הוא נקי מפגמים.

כלומר המטרה של כל האלגוריתמים הוא לבנות ייחוס נקי ככל האפשר מפגמים ולהשוות את הפגום אליו. כלומר בעצם הבנייה או השחזור של ה-die התקין לכאורה מבוסס על חקירת dies סמוכים וניסיון לבנות ייחוס הקרוב ככל האפשר למצב בו מתאר die ללא פגמים.

ההנחה הבסיסית מאחורי רעיון זה הוא שאם פגם מתרחש באזור מסוים ב-die אחד אין זה סביר שהוא יתרחש באותו מיקום ב-die אחר. השוואת אזורים ממיקומים זהים מתבססת על חישוב תמונת ההפרשים ממיקומים דומים ולאחר מכן נרמול של ההפרש בתלות בערך של זוג הפיקסלים המשתתפים בקביעת ההפרש. במידה וההפרש המנורמל גדול מסף מסוים ניתן להניח שיש סיכוי לקיומו של פגם. ישנם שתי שיטות לקביעת הספים. השיטה הראשונה נעשית בתהליך כיוול מקדים בו אוספים היסטוגרמות משניים או יותר dies שקטים. בעזרת היסטוגרמות אלו המציינות התפלגות נורמאלית קובעים את הספים כך שרק אחוז מסוים מההפרשים יעברו את הסף. בהנחה שיש לנו התפלגות נורמאלית הספים הללו תקפים לגבי כל ה-wafer. בשיטה השנייה מוסיפים גם מדידת רעש נוספת בין כל שני dies ומעדכנים את הספים שנקבעו בתהליך הסטטי של הכיוול המקדים עבור כל זוג dies. בעבודה זו נניח ספים סטטיים כמתואר בשיטה הראשונה מכיוון שמדידת רעש ועדכון ספים אדפטיבי הוא נושא מורכב כשלעצמו.

כדי לוודא את מיקומו של הפגם (הרי ההפרש בין ערכי הבהירות יכול לנבוע מה die הנבדק אבל גם מ-die הייחוס) נבצע השוואה נוספת בין ה die הנוכחי ל-die הבא. השוואה כפולה זו תאפשר לנו ראשית לוודא שההפרש אכן אמיתי ולא נובע מרעש מכיוון שאם יש הפרש בין הנקודה מסוימת בין die n ל-die n-1 וגם בין die n ל-die n+1 ושני ההפרשים (המנורמלים) עולים מעל סף מסוים אזי ההפרש מראה על פגם כפי שמודגם באיור הבא:



איור 15 תאור גרפי עקרוני של אלגוריתם D2D עם השוואה כפולה

בנוסף לכך כאשר פגם מופיע בהשוואה כפולה סביר להניח שזהו פגם אמיתי וכך נחסוך דיווח על פגמים שיקרים. ישנם שתי בעיות עיקריות בבואנו לממש אלגוריתם זה. הבעיה הראשונה קשורה לעובדה שבזמן סריקת ה-wafer ישנם רעידות קטנות מאוד של ה-stage שעליו מונח ה-wafer (כמו שמכונת כביסה רועדת כאשר היא עובדת) שביחד עם העובדה שגודל הפיקסל הוא בסדר גודל של עשרות עד מאות ננומטרים נוצר מצב שהבלוק המתקבל מ-die N איננו מיושר יחסית לבלוק המתקבל מ-die N-1. עובדה זו גורמת לכך שאיננו יכולים לחשב את תמונת ההפרשים מכיוון שפיקסל ממיקום מסוים מושווה לפיקסל ממיקום אחר. הבעיה אף מסובכת יותר מכיוון שמספר הפיקסלים שבהם מוזזת תמונה אחת יחסית לשנייה יכולה להיות מורכבת ממספר שלם של פיקסלים (למשל פיקסל אחד או שניים) ובנוסף תזוזה לא שלמה של פיקסל (sub-shift).

כדי לפתור את הבעיה של תזוזות התמונות אחת ביחד לשנייה צריך לבצע תהליך הנקרא יישור תמונה אחת יחסית לשנייה (image registration). תהליך זה כולל כאמור יישור ההיסט בין התמונות והוא כולל שני שלבים כדלקמן:

- השלב הראשון כולל את מדידת ההיסט בין התמונות. בשלב זה כולל גם מדידה פיקסל שלם וגם מדידה תת-פיקסלית של התזוזה כפי שיוסבר בהמשך.
- בשלב השני מבצעים תיקון ההיסט של תמונת הנוכחית (current image) יחסית לתמונת הייחוס (reference image).

בעיה נוספת שקיימת (עוצמתה חזקה יותר במערכות BF כפי שהוסבר בפרק 2) היא בעיית ה-color variation הגורמת להשתנות של צבע רקע כתוצאה מעובי שונה של פרוסת הסיליקון. בעיה זו עלולה לגרום לשוני ערכו של הפיקסל בין die ל-die אבל אין היא מצביעה על פגם. יצירת תמונת הפרשים (כמובן לאחר יישור ההיסט) עלולה ליצור פגמי שווא (false alarm).

כדי לענות על בעיית יישור ההיסט נתעלם לעת עתה מתופעת ה-color variation (האלגוריתם להתמודד איתה יוצג בהמשך). בתהליך שיוסבר בתת סעיפים הבאים בפרק זה נבצע מדידה של ההיסט בין כל שני בלוקים עוקבים. תוצאות המדידה של ההיסט בין הבלוקים (displacement vectors) ישמשו כקלט לפונקציה שתבצע תיקון פיקסל שלם וכן של תת-פיקסל של בלוק מ-die n יחסית לבלוק מ-die n-1 ורק לאחר שתיקון ההיסט הושלם ניתן לבצע השוואה בין בלוק n לבלוק n-1. ההשוואה בין שני הבלוקים עם ההיסט המתוקן של אחד ביחס לשני תבצע כאמור ע"י ביצוע הפרש מנורמל כפי שהודגם באלגוריתם הבסיסי ומימושו [בנספח-4 \(מימוש פונקצית השוואה\)](#) אותו תהליך יתבצע גם בין זוג הבלוקים מ-die n ו-die n+1. תוצאות תמונות ההפרשים המנורמלות משתי תמונות ההפרשים יעברו למודול שיבחר רק את המיקומים של ההפרשים ששניהם עברו סף מסוים. השוואה כפולה זו תבטיח רמת גילוי נאותה ותסנן מספר רב של פגמי שווא. קיימים אלגוריתמים ומחקרים רבים

למידת ההיסט בין שתי תמונות ושיטות שונות לתקן תזוזה של תמונה יחסית לשנייה ובסעיפים הבאים תוצג אחת מהאפשרויות בהתבסס על יכולת מימוש יעילה של אפשרות זו והמתבססת על המאמרים [17], [18], [21], [27] שבמקורות.

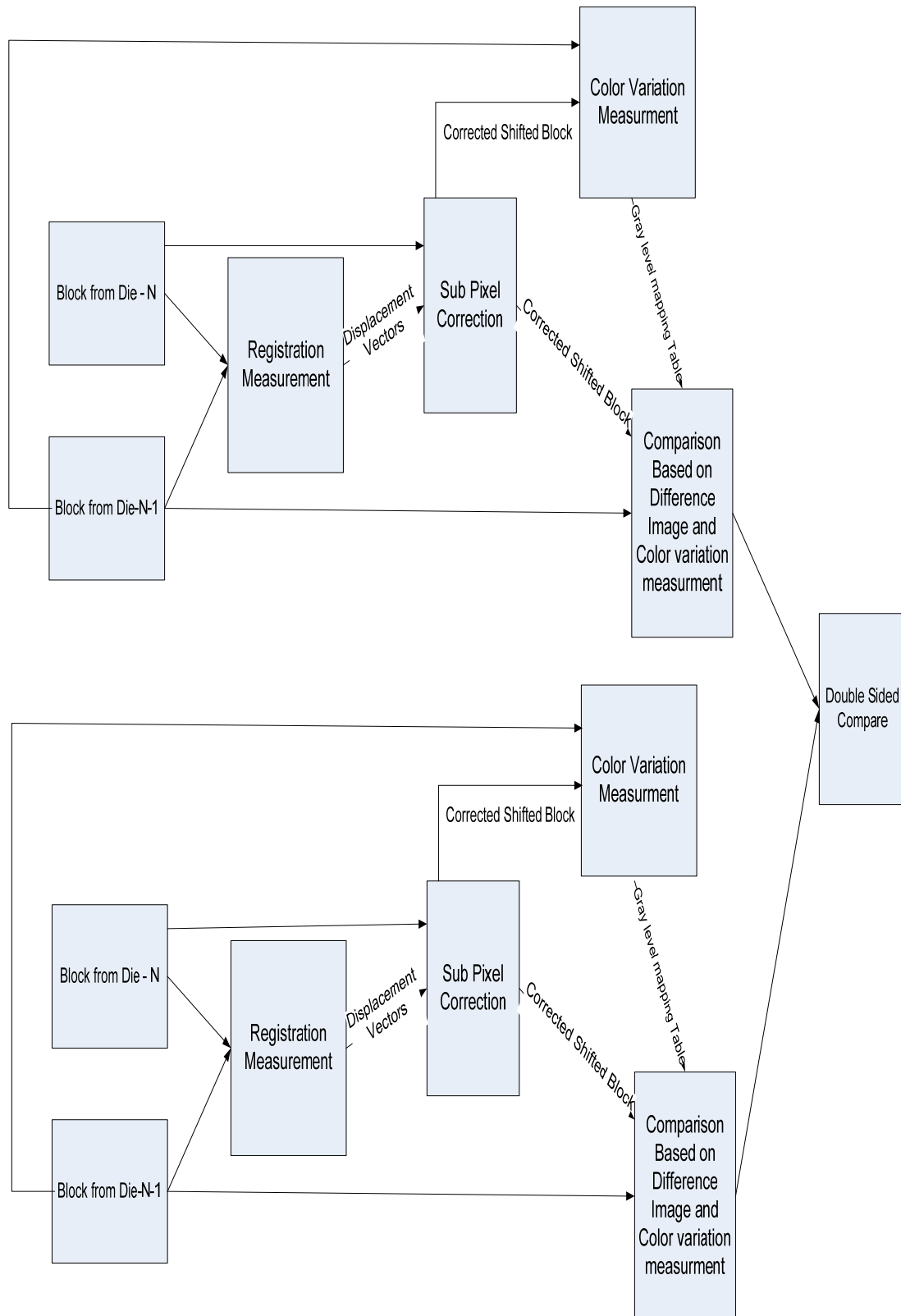
הבעיה השנייה שתוארה הינה בעיית ה- color variation. במכונות בהם תופעה זו חזקה נקבל מספר רב של פגמי שווא (false alarm) במידה ולא נטפל בה. הגישה שבה ניתן לנקוט כדי לטפל בתופעה זו היא למדוד את עוצמתה עבור כל אחד מהפיקסלים או קבוצות של פיקסלים או אזורים מוגדרים מראש ב- die ובהתאם לתוצאות המדידה לבצע תיקון של תמונת ההפרשים, או ההפרשים עצמם או לתקן את הספים או את ערכי הפיקסלים המשתתפים ביצירת ההפרש (כמובן שאלו פעולות שקולות) בהתבסס על תוצאות המדידה. הדיון והמימוש בנושא זה יתבסס על המאמרים [19], [20], [31] שבמקורות. השאלה שנשאלת היא האם ניתן לבצע תיקון ההיסט בין התמונות כאשר תופעת ה- color variation קיימת. התשובה לכך חיובית וזאת מכיוון שתהליך תיקון ההיסט שתואר מקודם מתבסס על טכניקת pattern matching ומסוגל למדוד את ההיסט בין שני ה- dies גם אם צבע הרקע שונה (בגלל תופעת ה- color variation) לכן נוכל להניח שתהליך תיקון ה- color variation יכול להתבצע אחרי תהליך תיקון ההיסט בין התמונות ואיננו מפריע למדידת התזוזה ותיקונה.

כלומר לפני ביצוע ההשוואה בין שני ה- dies נמדוד את תופעת ה- color variation כפי שיוסבר בהמשך, לאחר מדידת התופעה נבצע תיקון של רמות האפור של בלוק אחד יחסית לשני בדרך שתוסבר בהמשך וכך נוכל לגלות פגמים גם בתמונות רוויות בתופעת ה- color variation.

לסיכום אלגוריתם d2d עם ביצוע תיקון color variation מתואר באיור הבא והוא כולל כמו שהוסבר שני מרכיבים עיקריים כדלקמן: הראשון הינו מדידת ההיסט בין שני בלוקים עוקבים (משני dies עוקבים) ותיקונם כך שבלוק אחד מיושר יחסית לשני.

השני כולל את מדידת תופעת ה- color variation ותיקון עוצמות רמות האפור של הפיקסלים בזמן חישוב תמונות ההפרשים המנורמלות.

האיור הבא מתאר גרפית את שלבי האלגוריתם שתוארו לעיל.



איור 16 תאור גרפי של אלגוריתם למציאת פגמים מבוסס d2d עם תיקון תופעת ה- color variation

4.1.1 מדידת ההיסט בין תמונות (Registration Measurement)

המידע בסעיף זה נלקח ברובו ממאמרים [17] [18] [21] [27] בנוסף לחומר כללי הנלמד בקורסים לעיבוד תמונה. בסעיף הקודם תוארה בעיית ההיסט בין תמונות עוקבות ככזו המפריעה לביצוע ההשוואה בין שני בלוקים עוקבים המתארים תמונות שנלקחו ממיקומים זהים משני dies עוקבים. הרעיון הבסיסי במדידת ההיסט הוא להשתמש באלגוריתם לזיהוי צורות ע"י התאמת תבניות (pattern matching algorithm). נניח לרגע שאנו יודעים על מיקומים כלשהם בבלוק שבהם ישנם תבניות צורה חזקות בכיוון X או בכיוון Y (תבניות אלו ניתנות לאיתור יחסית בקלות פעם אחת בסריקה והם נכונות לגבי כל ה-dies). ניקח לדוגמא חלונות עם גודל של 32×32 פיקסלים. הסיבה לבחירת חלונות בגודל 32×32 קשורה לגודל התבניות שבעזרתם נרצה לאתר את ההיסט והמחזוריות של התבניות. אם המחזוריות קטנה מ-32 פיקסלים הדבר ישבש את זיהוי התבנית מכיוון שהיא לא תהיה ייחודית בחלון החיפוש. נחפש התאמה מקסימאלית ברדיוס חיפוש של 2 פיקסלים מכל כיוון. לפי תורת ההסתברות והסטטיסטיקה מקדם המתאם ההדדי דהיינו מקדם הקורלציה בין שני משתנים בלתי תלויים נתון לפי הנוסחה הבאה:

תמונה נוכחית שעליו מתבצע גילוי פגמים C
תמונת הייחוס M

$C = \text{Current Image}$ $M = \text{Memory Reference Image}$	$\rho = \frac{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \{ (C_{i,j} - \bar{C}) (M_{i,j} - \bar{M}) \}}{\sqrt{(\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (C_{i,j} - \bar{C})^2) (\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (M_{i,j} - \bar{M})^2)}}$
---	---

כדי להתאים את נוסחת חישוב מקדם הקורלציה לביצוע חישוב תוך מעבר אחד על המידע נבצע את

הפעולות הבאות כדי לקבל ביטוי שאיננו כולל את $\bar{C}$ ו- $\bar{M}$ ובכך לאפשר חישוב יעיל של מקדם הקורלציה תוך מעבר בודד על המידע. כדי להתאים את הנוסחה כאמור נפתח את הביטוי הבא: נתבונן

$$\begin{aligned} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \{ (C_{i,j} - \bar{C}) (M_{i,j} - \bar{M}) \} &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (C_{i,j} M_{i,j} - C_{i,j} \bar{M} - M_{i,j} \bar{C} + \bar{C} \bar{M}) = \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} M_{i,j} - \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} \bar{M} - \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{i,j} \bar{C} + \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \bar{C} \bar{M} = \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} M_{i,j} - \bar{M} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} - \bar{C} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{i,j} + \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \bar{C} \bar{M} = \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} M_{i,j} - mn \bar{M} \bar{C} - mn \bar{C} \bar{M} + mn \bar{C} \bar{M} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} M_{i,j} - \frac{1}{mn} \left(\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{i,j} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} \right) \end{aligned}$$

כעת נטפל במכנה בצורה הבאה:

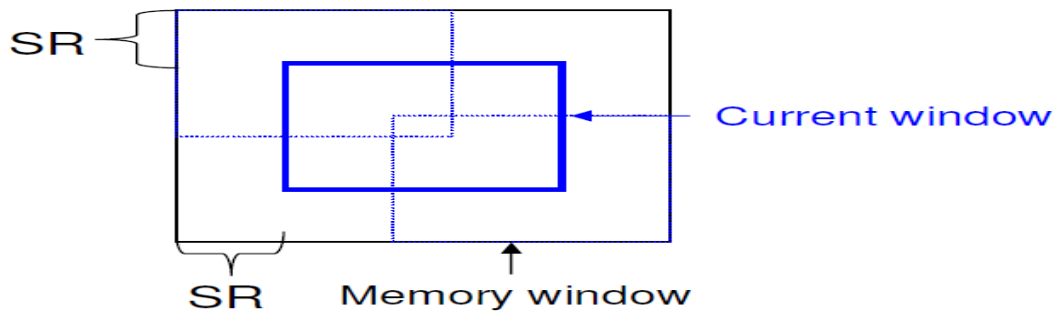
$$\begin{aligned} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (C_{i,j} - \bar{C})^2 &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (C_{i,j}^2 - 2C_{i,j} \bar{C} + (\bar{C})^2) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j}^2 - 2\bar{C} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} + mn \bar{C}^2 = \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j}^2 - 2\bar{C} mn \bar{C} + mn \bar{C}^2 = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j}^2 - mn \bar{C}^2 = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j}^2 - \left(\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} \right)^2 \frac{1}{mn} \end{aligned}$$

נבצע את אותו חישוב עבור M ונכפול מונה ומכנה ב mn נקבל את הנוסחה הבאה:

$$\rho(C, M) = \frac{\left(mn \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} M_{i,j} - \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{i,j} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} \right)}{\sqrt{\left(mn \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j}^2 - \left[\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{i,j} \right]^2 \right) \left(mn \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{i,j}^2 - \left[\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{i,j} \right]^2 \right)}}$$

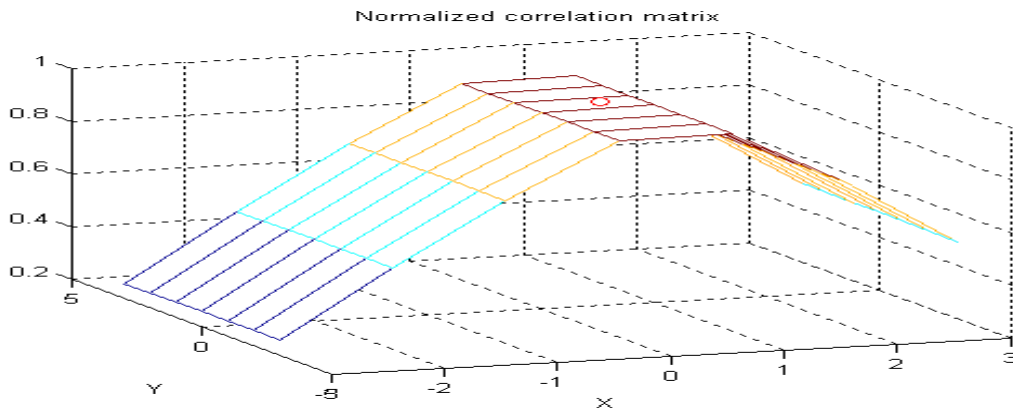
המאפשרת לחשב את מקדם הקורלציה במעבר אחד על חלונות הקורלציה.

כדי למצוא את ההיסט נבצע חישוב של מקדמי ההתאמה ההדדית (מקדמי הקורלציה) בין חלון מהבלוק הנוכחי (C) לבין הבלוק הקודם (M). את הפעולה זו נעשה בתזוזה לפי מאפייני המכונה. אם המאפיינים הטכניים של המכונה מראים על תזוזה של מקסימום 2 פיקסלים לכל כיוון נבצע את חישובי מקדמי הקורלציה כמתואר באיור הבא:



איור 17 תאור גרפי של תזוזת חלון C על M.

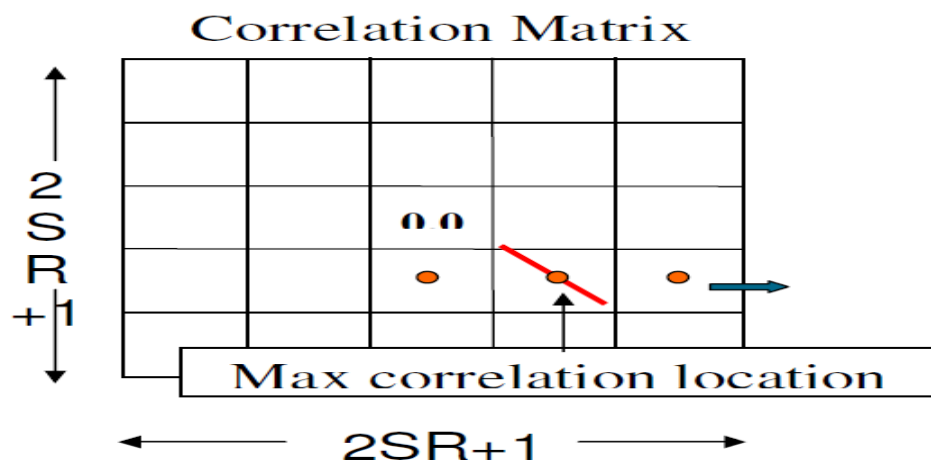
במקרה שלנו נקבל מטריצת קורלציה בגודל 5×5 בה כל ערך מציין את ציון הקורלציה בתזוזה המתאימה. הקורלציה של חלון C עם חלון M במרכז מסומנת ע"י $\rho_{0,0}$ אחת ימנה בכיוון x מסומנת ע"י $\rho_{0,1}$ ותזוזה שמאלה ע"י $\rho_{0,-1}$ ואותו כנ"ל לגבי ציר y. כאמור סה"כ מטריצת הקורלציה מכילה 25 מקדמים המתארים משטח דו ממדי בגודל 5×5 . ניתן להסתכל על מטריצת הקורלציה האידיאלית כמבנה המתאר פרבולואיד כמודגם באיור הבא:



איור 18 תאור גרפי של מטריצת הקורלציה במדידת תזוזת חלון C על M.

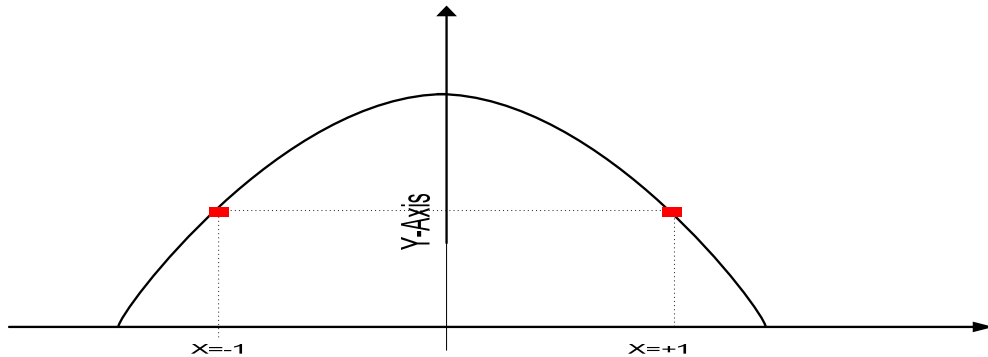
המקום שבו מקדם הקורלציה הוא מקסימאלי הוא המקום בו ההזזה היא בפיסקל שלם. ההנחה הבסיסית שמטריצת הקורלציה מתארת פרבולואיד כך שהמקסימום מתאר את הקורלציה המקסימאלית. ניתן לבצע מדידת תזוזה בנפרד ל- x ו- y ולכן ניתן לעבור מגוף דו-ממדי כפרבולואיד להיטל החד ממדי שהוא פרבולה. כמו שתואר בתחילת הסעיף מכיוון שהדגימה איננה שלמה קיימת גם תזוזה תת-פיקסלית. כדי לחשב אותה נניח שבציר החד-ממדי נקודת המקסימום ושתי הנקודות שמצדדיה מתארות פרבולה.

Full Pixel – max correlation

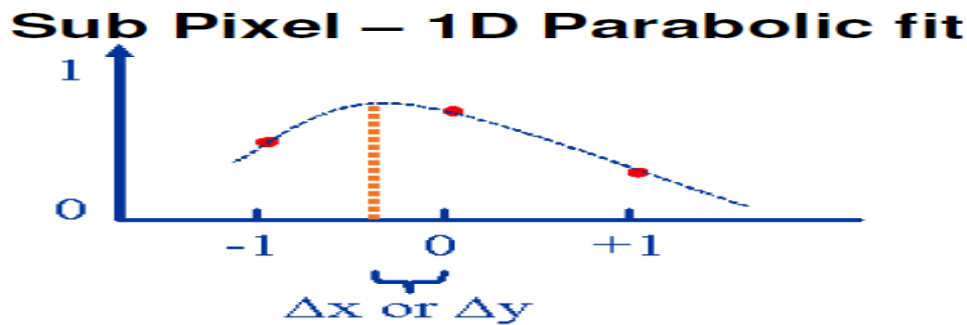


איור 19 תאור גרפי של תזוזת מטריצת הקורלציה בגודל 5×5

ע"י ביצוע התאמה לפרבולה שבציר החד-ממדי נקודת המקסימום ושתי הנקודות שמצדדיה מתארות פרבולה. במידה וערכי הנקודות שמימין ומשמאל למקסימום שוות (כמו באיור הבא), אז התזוזה התת-פיקסלית היא אפס.



איור 20 תאור גרפי של פרבולה סימטרית מסביב למקסימום
כאמור ברוב הפעמים אין סימטריה סביב נקודת המקסימום ולכן נמצא את המקסימום ע"י קירוב לפרבולה כמודגם באיור הבא:



איור 21 תאור גרפי של שלוש נקודות מדידה, הקירוב לפרבולה וההיסט Δx או Δy
כאמור ברוב המקרים אין סימטריה של הנקודות סביב המרכז ולכן נבצע את החישוב הבא כדי למצוא את התזוזה התת-פיקסלית.

$$x = -1 : \rho_1$$

$$x = 0 : \rho_2$$

$$x = +1 : \rho_3$$

נרשום את משוואת הפרבולה בצורה הבאה: $y = Px^2 + Qx + R$
נציב את הנקודות 0, -1, +1 ונקבל

$$P - Q + R = \rho_1$$

$$R = \rho_2$$

$$P + Q + R = \rho_3$$

$$Q = \frac{\rho_3 - \rho_1}{2}$$

$$P = \frac{\rho_1 + \rho_3 - 2\rho_2}{2}$$

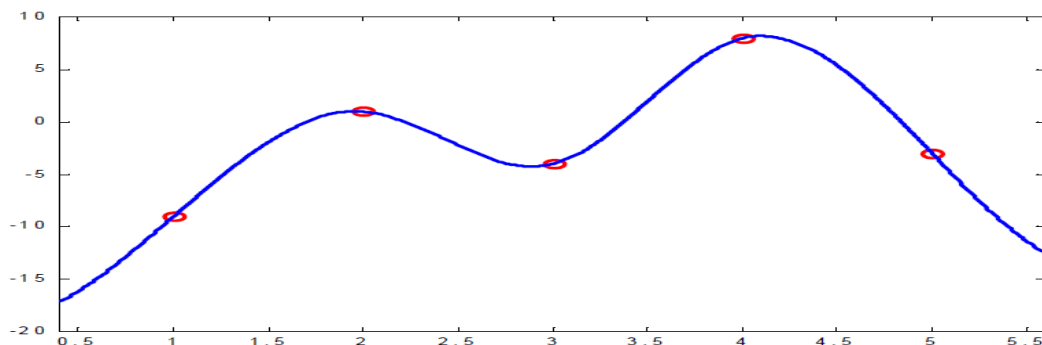
$$x_{MAX} : f'_x = 2Px + Q$$

$$x_{MAX} = -\frac{Q}{2P} = \frac{\rho_1 - \rho_3}{2(\rho_1 + \rho_3 - 2\rho_2)}$$

המרחק של x_{MAX} מ-0 זוהי ההזזה התת-פיקסלית.
מכיוון שהתזוזה של תמונה אחת יחסית לשנייה איננה בהכרח זהות בכל נקודה בבלוק המעובד יתכן ונזדקק לבצע את אלגוריתם החיפוש ומדידת התזוזה על רצועות שונות ברחבי הבלוק. בנוסף בכל רצועה כפי שהוזכר יש לחפש שתי תזוזות, האחת בציר X והשנייה בציר Y. כפי שנראה בהמשך, מדידות התזוזה דורשות פעולות רבות אך למזלנו אלו הם פעולות (מכפלות וסכומים) שבהם כמובן המעבד DSP מצטיין ולכן ניתן לממש אותן ביעילות מרבית. כאמור ביצוע מדידות ההיסט מתבצע על מספר רצועות בבלוק שאורכם 128-64 פיקסלים ורוחבם כגודל הבלוק. הסיבה לכך היא שהתזוזה של ההיסט איננה זהה בכל הבלוק ולכן נדרשות מספר מדידות היסט.

4.1.2 תיקון ההיסט בין תמונות עוקבות (Registration Correction)

המידע בסעיף זה נלקח ברובו ממאמרים [17] [18] [21] בנוסף לחומר כללי הנלמד בקורסים לעיבוד תמונה.
לאחר השלב של מדידת ההיסט בין שתי תמונות עוקבות מגיע כאמור שלב התיקון של תמונה אחת יחסית לשנייה כדי שנוכל להשוות בניהן כלומר ליצור תמונת הפרשים מנורמלת. תיקון של אזור אחד בבלוק יחסית לשני בפיקסל שלם הינו קל יחסית. נניח לדוגמא שההיסט השלם בציר Y ו-X הוא 1. כדי לתקן את החלק השלם נזיז את המצביע על בלוק התמונה נוכחית (current) בשורה אחת קדימה ובעוד פיקסל אחד ימינה. כלומר פעולת תיקון החלק השלם היא בעצם הזזת מצביעים ותו לא. תיקון החלק התת-פיקסלי מסובך מעט יותר. ניתן להגדיר את התזוזה התת-פיקסלית כסוג של עיוות או מריחה של האות. כדי לתקן את האות נהוג לבצע סוג של אינטרפולציה בין פיקסלים שכנים כדי להסיק מהם על ערכו של הפיקסל שהיה אמור להיות בהיסט המתאים. קיימים סוגים שונים של אינטרפולציות המתאימים לתופעות פיזיקאליות ומתמטיות שונות. מכיוון שהמערכת האופטית שקיימת במכונת WI מבוססת לייזר משתמשת ב-spot (מפתח העדשה היוצרת את הפיקסלים) שההתנהגות שלו בתחום התדר היא בדומה לפונקציה המתמית $\text{Sinc } c = \frac{\text{Sinx}}{x}$. האפשרות הטובה ביותר לשחזר את האות הוא לבצע אינטרפולציה המבוססת על משחזר אידיאלי שהוא פונקציית sinc. משחזר אידיאלי כמובן איננו מעשי ומכיוון שהתזוזה מוגבלת ל-2 פיקסלים משתמשים spline כדי לבצע את האינטרפולציה ההכרחית לשחזור האות בתזוזה התת-פיקסלית. דוגמא לעקומת אינטרפולציה בעזרת spline מתוארת באיור הבא:

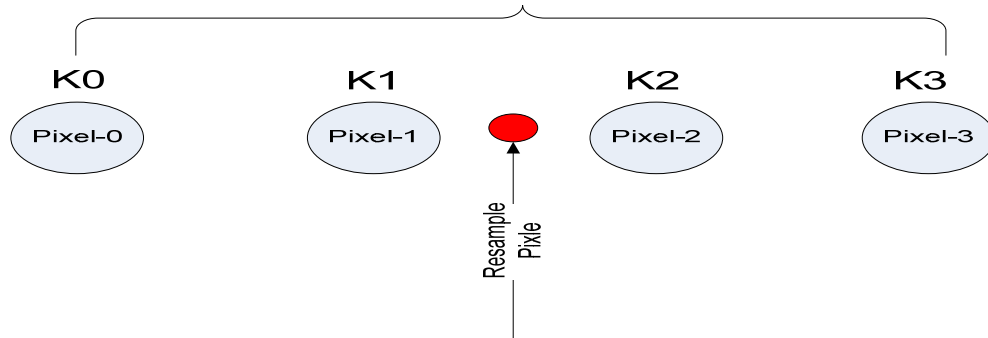


איור 22 תאור גרפי של אינטרפולציה בעזרת spline במימד אחד

הבעיה עם spline שהיא דורשת חישובים רבים כדי להסיק את מקדמי הנרמול והדגימה. לצורך כך במקרים בהם האות צריך להידגם ולהיבנות מחדש נשתמש בשיטה הנקראת PCC (parametric cubic convolution) ראה מאמר [21] בביבליוגרפיה) המאפשרת לייצר מקדמים המאפשרים לדגום מחדש את האות בהיסט בו אין לנו דגימה והיא מהווה קירוב טוב למשחזר sinc. אנו בונים לכל ציר את מקדמי הדגימה שלו (מקדמים אלו מוכפלים בערכי הפיקסלים השכנים עבור כל נקודה כדי לקבל את הנקודות החסרות) בצורה מופרדת ולאחר בונים מקדמים בגודל 4x4 משני וקטורים של 1x4 ו-4x1. הנוסחה עבור פונקציית האינטרפולציה הקובית בעזרת פרמטר היא

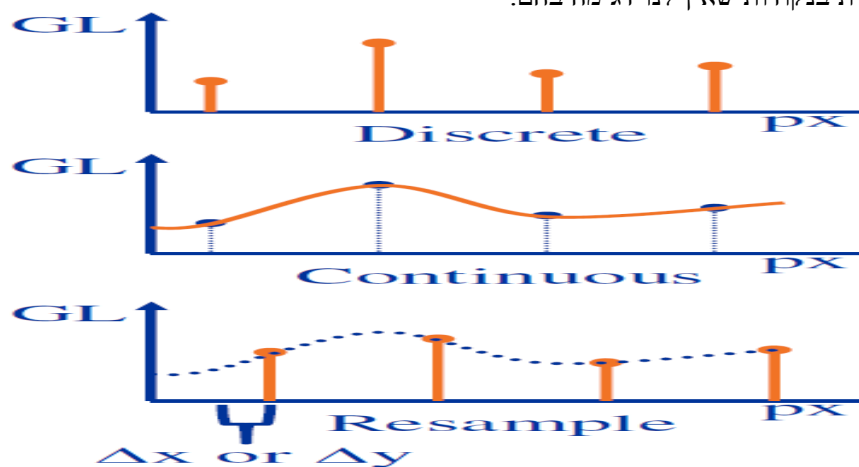
$$h(x) = \begin{cases} (\alpha + 2) |x|^3 - (\alpha + 3) |x|^2 + 1 & \longrightarrow \text{for } 0 \leq |x| < 1 \\ \alpha |x|^3 - 5\alpha |x|^2 + 8\alpha |x| - 4\alpha & \longrightarrow \text{for } 1 \leq |x| < 2 \\ 0 & \longrightarrow \text{otherwise} \end{cases}$$

כדי לחשב נקודה חסרה משתמשים ב 4 פיקסלים שכנים בצורה הבאה:
Kernel Coefficients



איור 23 תאור גרפי של שחזור הפיקסל בהיסט תת-פיקסלי בתלות במקדמים k0-k3 וארבע פיקסלים שכנים

התהליך המתואר באיור שלעיל מציג את חישוב הערך החדש של הפיקסל (בצבע אדום) שדגימתו חסרה ע"י שקלול ערכי הפיקסלים השכנים ומכפלת המקדמים k0-k3. כדי להבין את התהליך נתבונן באיור הבא המתאר את בדיד שהוא תוצאת דגימה של אות רציף ואנו מעונינים לשחזר אות בנקודות שאין לנו דגימה בהם.



איור 24 תאור גרפי של אינטרפולציה בעזרת cubic spline בחד מימד

כאמור אנו רוצים לבצע דגימה מחדש של כל הפיקסלים. את המקדמים עבור תזוזה מסוימת ניתן לחשב מראש עבור כל היסט אפשרי ולהשתמש בהם לפי הצורך כפי שיודגם בפרק הבא. חישוב המקדמים יעשה בצורה הבאה: נחלק את המרחב להזזה מינימאלית של 1/128. בהתבסס על פונקצית האינטרפולציה הקובית הפרמטרית חישוב מקדמי התיקון יהיה כדלקמן:

$$K0 = \alpha(2 - \text{abs_dx})^3 - 5\alpha(2 - \text{abs_dx})^2 + 8\alpha(2 - \text{abs_dx}) - 4\alpha$$

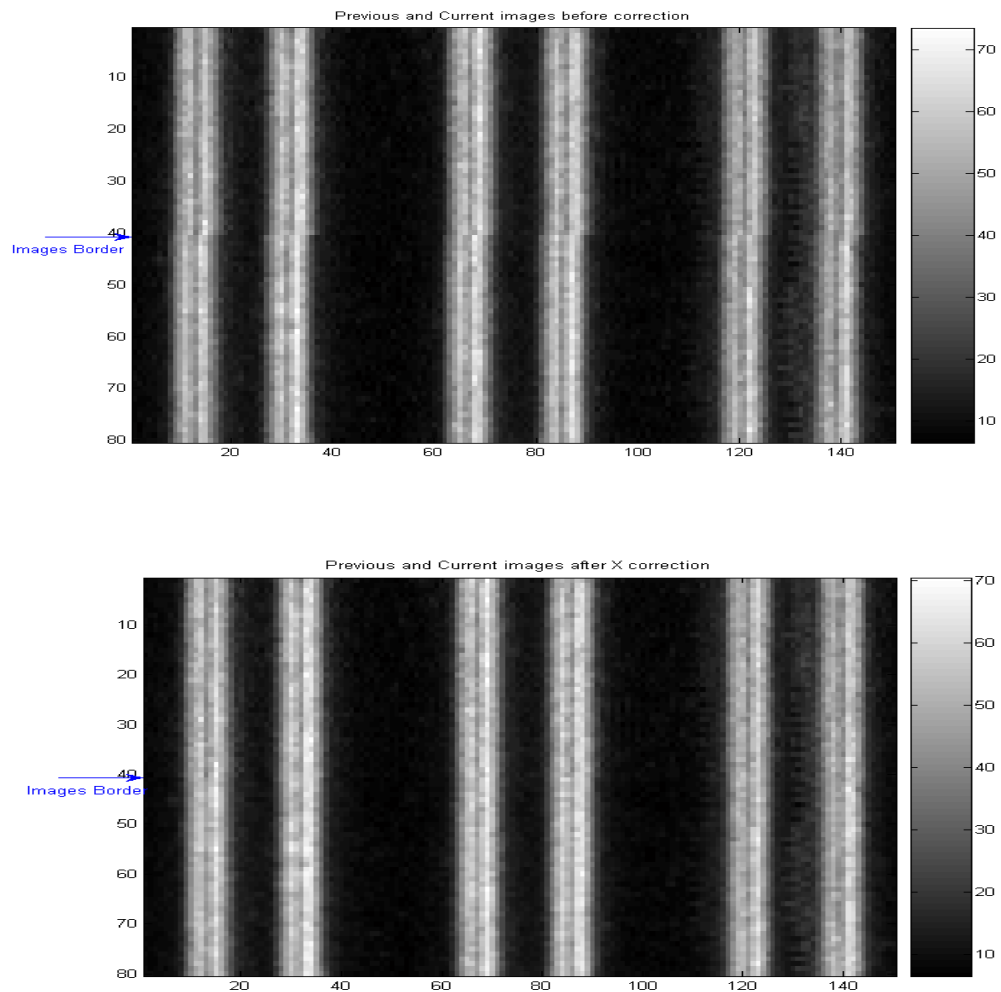
$$K1 = (\alpha + 2)(1 - \text{abs_dx})^3 - (\alpha + 3)(1 - \text{abs_dx})^2 + 1$$

$$K2 = (\alpha + 2)\text{abs_dx}^3 - (\alpha + 3)\text{abs_dx}^2 + 1$$

$$K3 = \alpha(1 + \text{abs_dx})^3 - 5\alpha(1 + \text{abs_dx})^2 + 8\alpha(1 + \text{abs_dx}) - 4\alpha$$

הפרמטר α מתאר משמעות פיזיקאלית שמשמעותה עקמומיות ה-sinc. הערך שנבחר במקרה זה הוא 0.65. כדי ליצור גרעין בגודל 4x4 ניתן להשתמש בעובדה שהאינטרפולציה 4x4 בנויה מהזזה של

1x4 בציר ה-X והזזה של 4x1 בציר ה-Y וכך לבצע מכפלת מטריצות 4x1 ב-1x4 ולקבל מטריצה 4x4 מתאימה.
האיור הבא מתאר שני תת בלוקים לפני תהליך יישור ההיסט ואחריו:

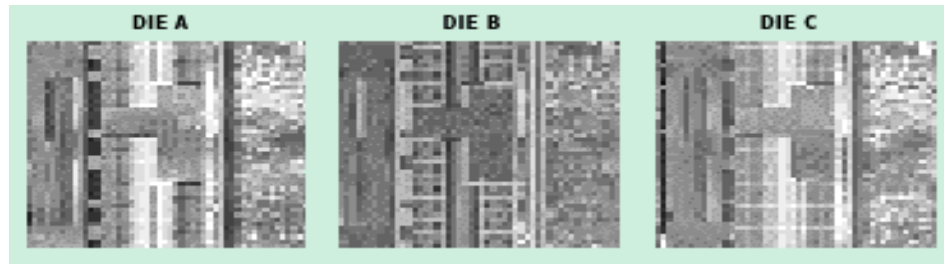


איור 25 תאור בלוק לפני תיקון ואחרי תיקון ההיסט במימד X
יש לציין שאחוזי ההצלחה או הכישלון של תהליך image registration תלויים בפרמטרים רבים כמו סוגי התבניות שעליהם מתביתים, המחזוריות שלהם ועוד. בתבניות שיש על ה-wafer שנסקרו היו מספר כישלונות מועט של כ-1% ואז בוצע מיסוך של האזור (במקרה וציון הקורלציה המקסימאלי נמוך מידי). אציין רק שנושא image registration נחקר רבות וקיימות שיטות רבות ושונות לשיפור התהליך (שכמובן עלולות לפגוע בזמן הריצה) אך לא נרחיב בעבודה זו על כך.

4.1.3 מדידה ותיקון של תופעת ה-Color Variation

המידע בסעיף זה נלקח ברובו ממאמרים [19] [20] [31] [33] בנוסף לחומר כללי הנלמד בקורסים לעיבוד תמונה.
בעיית ה-color variation הינה קריטית ליכולת ההשוואה ומציאת הפגמים של אלגוריתם מבוסס השוואה die2die. כדי להבין את עומק הבעיה נתבונן בתמונות משני dies עוקבים.
בדוגמא שלפנינו רואים שינוי מהותי בצבע של משטחים שונים הנובעים כמובן (כמו שהוסבר בפרק 2) מהעובדה שקיים אורך גל קצר ומקור אור מונו-כרומטי בעל אורך גל יחיד ביחד עם שכבה ברוחב משתנה. ברור שביצוע השוואה כפי שתוארה **בנספח-4 (מימוש פונקציית השוואה)** יניב דיווח על פגמי שווא שאינם באמת קיימים מכיוון שההפרשים בין זוגות של פיקסלים משני dies עוקבים הינם גדולים בגלל תופעת ה-color variation. מצב זה מקשה על קביעת ספים מכיוון שאם נוריד את הרגישות ונקבע

ספים גבוהים אנו עלולים לא לגלות פגמים הרסניים מצד אחד ומצד שני אם נוריד את הספים נגרום ליצירת וגילוי של פגמי שווא.



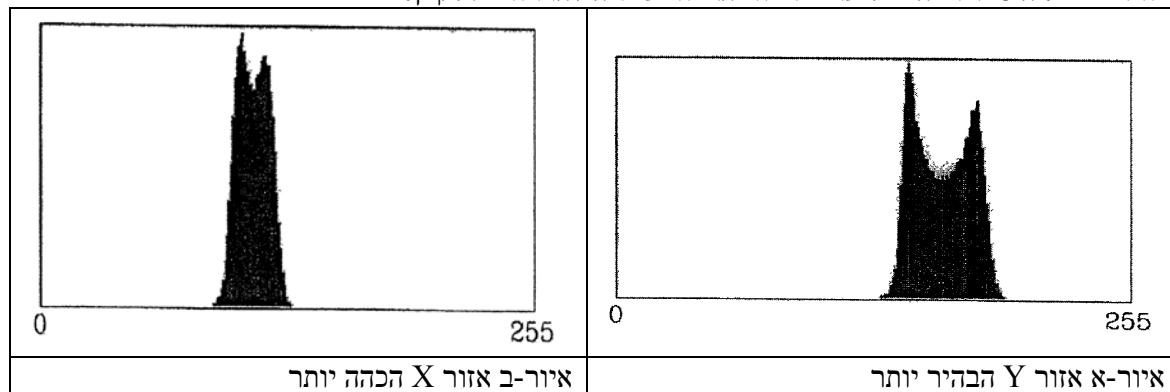
איור 26 תאור דוגמא של CV בין שלושה dies עוקבים

ניתן לחשוב שאם נכיל את הספים בכל השוואה בין שני בלוקים מחדש, נוכל להמעיט בדיווח פגמי השווא, אולם ישנם שתי בעיות בגישה זו. ראשית פעולת כיוול הספים עבור כל זוג בלוקים עוקבים מחדש היא פעולה יקרה במונחי זמן חישוב. שנית גם אם נכיל את הספים ונעלה אותם במקרה הצורך עדיין נפחית את הדיווח על פגמי השווא אבל מצד שני נפגע בגילוי של פגמים אמיתיים מכיוון שנהיה פחות רגישים (העלאת הספים גורמת לחוסר רגישות גדול יותר). דרושה אם כן דרך ללמוד על שינויי הצבע, למפות אותם ולתקן תופעה זו כדי שנוכל להשוות בלוקים עוקבים.

הרעיון הבסיסי המוצג במספר פטנטים ומאמרים העוסקים בבעיה זו (ראה [19], [20], [31] במאמרים) מבוסס על מדידת המיפוי של רמות אפור או קבוצות של רמות אפור מבלוק אחד לשני ולהיפך (כלומר משני dies עוקבים או יותר). לאחר מדידת התופעה ניתן לתקן את ההפרשים בין current ל-previous כדי לקבל הסתברות נכונה לפגם. מדידת התופעה מתבססת על העובדה שהשינוי לאורך ה-wafer גורם לשינוי רמות אפור (gray level) בתדירות נמוכה בין שני dies סמוכים.

תחת השפעת color variation ה-predictor כלומר אות הייחוס $GL(current) = GL(previous)$ כבר לא תקף וניתן כאמור לשפר אותו ע"י סטטיסטיקות גלובליות. כאמור הרעיון הבסיסי שהוצג במאמרים [19], [20], [31] מבוסס על בנייה של טבלת תיקון עבור כל GL או קבוצת GL. האלגוריתם צריך גם לזהות מיקרים של איכות מדידה נמוכה ולמסך אזורים בעייתיים כאלה.

הרעיון הבסיסי כאמור בכל השיטות הוא לנסות לתקן את רמות האפור של האזורים המושוים או לתקן את הספים בהתאם לשינוי הרקע כתוצאה מ-color variation. הרעיון המוצג ב-[19] על ידי Chung מניח שמבין שני האזורים המושוים משני ה-dies העוקבים ישנו אזור כהה יותר ואזור בהיר יותר מכיוון שההנחה שהתבניות המתקבלות זהות ואילו צבע הרקע שונה. נגדיר את האזור X כאזור הכהה ואת הבהיר יותר כ-Y. היסטוגרמה של ערכי רמות האפור עשויה להראות כדלקמן:



איור 27 תאור האזור הכהה והבהיר משני dies סמוכים עם CV

כדי למפות ולתקן את ערכי הבהירות מהאזור הכהה לבהיר מחשבים ממוצע (mean) וסטית תקן (std) של כל אחד מהאזורים. ערכי רמות האפור של הפיקסלים (gray level) מאזור X מתוקנים כך שהממוצע וסטיית התקן של אזור X שווים לאלה של אזור Y.

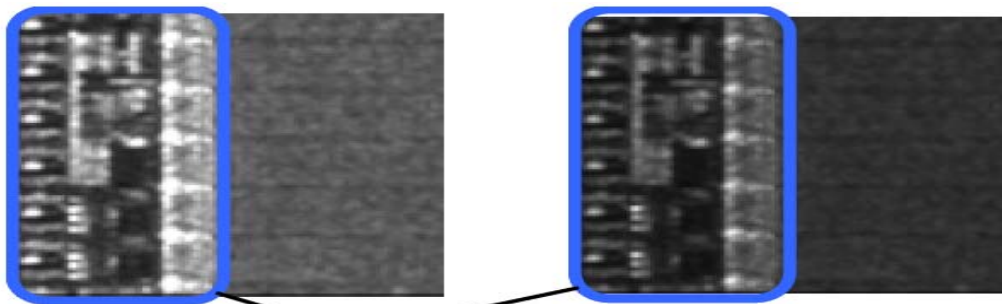
הדבר מתבצע בעזרת הנוסחה הבאה:

$$X'_i = \mu_Y + (X_i - \mu_X) \frac{\sigma_Y}{\sigma_X}$$

כאשר μ, σ הם הממוצע וסטיית התקן בהתאמה. ניתן לפשט את תיאור התיקון המוצע בכך שבעצם אנו מבצעים תיקון על ידי שיווי היסטוגרמות בין שני הבלוקים הנבדקים (histogram equalization). החיסרון באלגוריתם זה הוא שלא תמיד ההיסטוגרמה היא הומוגנית ומרוכזת באזורים צפופים אלא פזורה על תחום רחב של רמות האפור וקשה לאפיין ולבצע שיווי היסטוגרמות כנדרש אולם במקרים שבהם ההיסטוגרמות הם מהצורה המוצגת השיטה יעילה ומהירה.

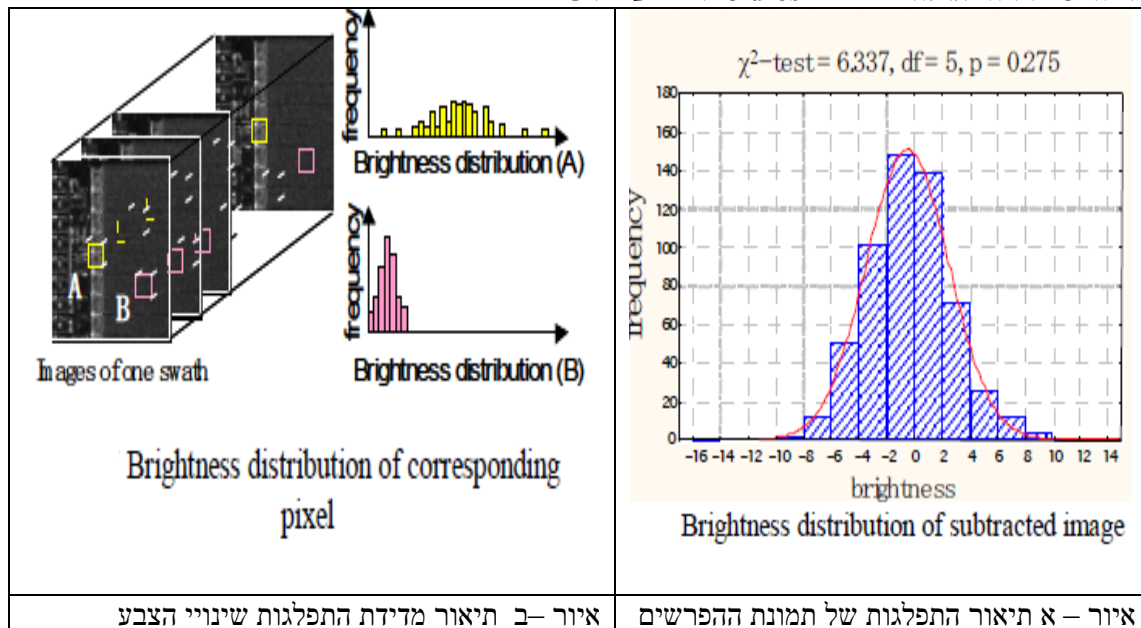
חיסרון נוסף באלגוריתם זה הוא שבנוסף להנחה על אופי התפלגות ערכי רמות האפור (התפלגות מרוכזת בתחום מסוים שאיננה פזורה על כל התחום הדינאמי) יש מיפוי ומדידה (וכמובן תיקון) רק בכיוון אחד ללא בדיקה של המיפוי בכיוון השני מ-Y ל-X שעשוי להיות מתאים יותר. בנוסף לכך ברור שבמקרים של הכיווניות של תופעת CV שונה עבור ערכי רמות אפור שונות נקבל מצב שבו קצה אחד בהיסטוגרמה יכול להיות ממופה לקצה מנוגד ולהיפך. כלומר שיטה זו תהיה יעילה במידה ואכן צבע הרקע משתנה לכל ערכי האפור במידה שווה.

השיטה המתוארת ב-[31] על ידי Akira מרחיבה את מדידות התפלגות הבהירות עבור אזורים שונים כאשר האזור המינימאלי יכול להיות גם זוג פיקסלים במיקום כלשהו על ה-wafer או בצורה מעודנת יותר קבוצת פיקסלים קרובים או אזורים שלמים. בשיטה הנ"ל בודקים את השתנות צבע הרקע לאורך מספר רב של dies ומאפיינים את התפלגות הבהירות לפי מודל כלשהו (התפלגות נורמאלית, פולינומיאלית או כל התפלגות המתאימה למודל של התופעה) מוצאים את הממוצע וסטיית התקן של כל איזור ובהתאם לכך מעדכנים את הספים כך שיגרמו לגילוי של אחוז אחד מהפרשים. האיור הבא מדגים זאת:



Brightness unevenness

איור 28 תאור האזורים משני dies סמוכים עם CV



איור 29 תאור התפלגות תמונת ההפרשים מאזור מסוים שבו נמדדה תופעת ה-CV.

ניתן לראות שהרעיון באלגוריתם זה מבוסס על איסוף סטטיסטיקות ממספר רב של dies ותיקון הספים בהתאם. הבעיה העיקרית באלגוריתם זה וגם בקודם הוא שאין התייחסות למדידת המיפוי בשני הכיוונים של ה-dies המשווים, לעתים דווקא לקיחת סטטיסטיקות מיותר משני dies עלולה לפגוע בהשוואה של שני dies מסוימים. יש לציין שאיסוף סטטיסטיקות גלובלי בכל ה-wafer כנדרש כאן לא קל למימוש כי הוא תלוי חומרה (תקשורת רבה בין כל יחידות החישוב ועוד). מצד שני כאשר אזורי ה-die2die מעטים יש יתרון לאיסוף סטטיסטי לאורך מספר dies.

בנוסף לכך יש יתרון לאפשרות לאסוף סטטיסטיקות גלובליות לאורך מספר dies בכך שניתן לעשות מעקב אחר השתנות הצבע לאורך מספר dies ולהסיק מכך האם שינוי הוא פגם או color variation. הרעיון האחרון המוצג ב-[20] הוא בעצם פשרה של שני הרעיונות המוצגים [19] ו-[31]. באלגוריתם זה מתבצעת בדיקת מיפוי של קבוצות של רמות אפור כאשר ברמה העקרונית כל ערך של רמת אפור הוא קבוצה כלומר יש לנו 256 מיפויים. עבור כל זוג פיקסלים משני dies סמוכים (current ו-previous) נבנה טבלת מיפוי של ערך הפיקסל של ה-current שהוא הממוצע של ערכי ה-previous המתאימים אליו, וכנ"ל להיפך טבלת מיפוי של ערך הפיקסל של ה-previous שהוא הממוצע של ערכי הפיקסלים של ערכי ה-current המתאימים אליו. בנוסף לבדיקת הממוצע של המיפויים בשני הכיוונים נחשב את סטיית התקן של כל מיפוי.

היתרון של בדיקת המיפוי בשני הכוונים הוא שניתן לבחור את המתאים ביותר ולא לקבוע שרירותית כיוון אחד כמו באלגוריתמים הקודמים. מצד שני השוני כאן שהמאפיין למדידה הסטטיסטית בכל כיוון הוא ערך הפיקסל ב-die המנוגד לזה שבו רוצים למדוד את התופעה. הרעיון הבסיסי באלגוריתם שאותם ממשתי והמבוסס על [20] יהיה כדלקמן עבור כל זוג בלוקים עוקבים נבנה טבלת מיפויים בשני כיוונים:

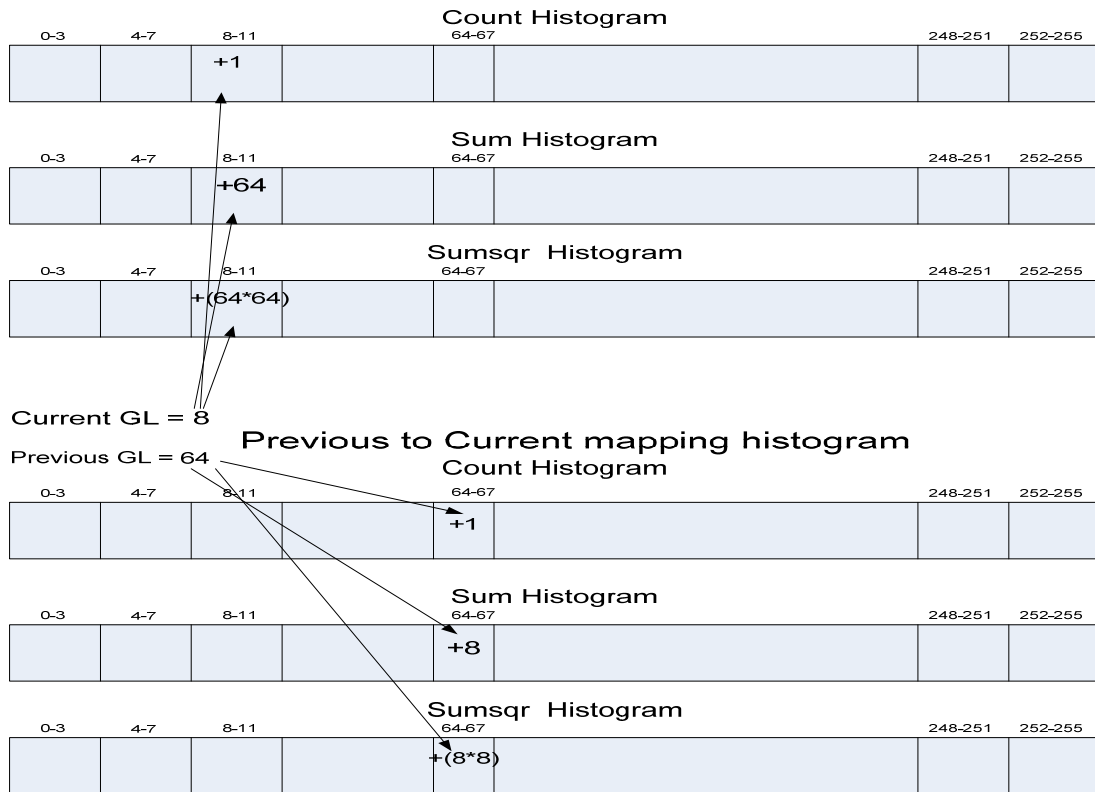
- מבלוק נוכחי (current) לבלוק קודם (previous) נסמן את מיפוי זה ע"י C2P (current to previous).
- מבלוק (previous) לבלוק נוכחי (current) נסמן את מיפוי זה ע"י P2C (previous to current).

$$GL_{Current} \Rightarrow Mean(GL_{Previous}); STD_{Current-to-Previous}$$

$$GL_{Previous} \Rightarrow Mean(GL_{Current}); STD_{Previous-to-Current}$$

כדי לעמוד בביצועים סבירים (כפי שהוסבר בפרקים הקודמים זהו תנאי בל יעבור) חילקתי את תחום ה-gray level ל 64 חלקים כך שכל חלק כולל 4 רמות אפור וזאת כדי להגיע לקוד עם ביצועים יעילים (שכמובן עלול לפגוע ברגישות). כדי לבנות את מיפויי רמות האפור מבלוק נוכחי לבלוק קודם ולהיפך נבנה היסטוגרמות שהמבנה שלהם כדלקמן: עבור כל זוג פיקסלים הנמצאים במיקומים זהים משני בלוקים עוקבים נסכם עבור כל כיוון את הסכום, הסכום בריבוע והכמות. האיור הבא מתאר את מבנה ההיסטוגרמות בשני הכיוונים (מבלוק current ל-previous ולהיפך).

Current to previous mapping histogram



איור 30 תאור דוגמא למיפוי זוג previous current, להיסטוגרמות מיפויים.

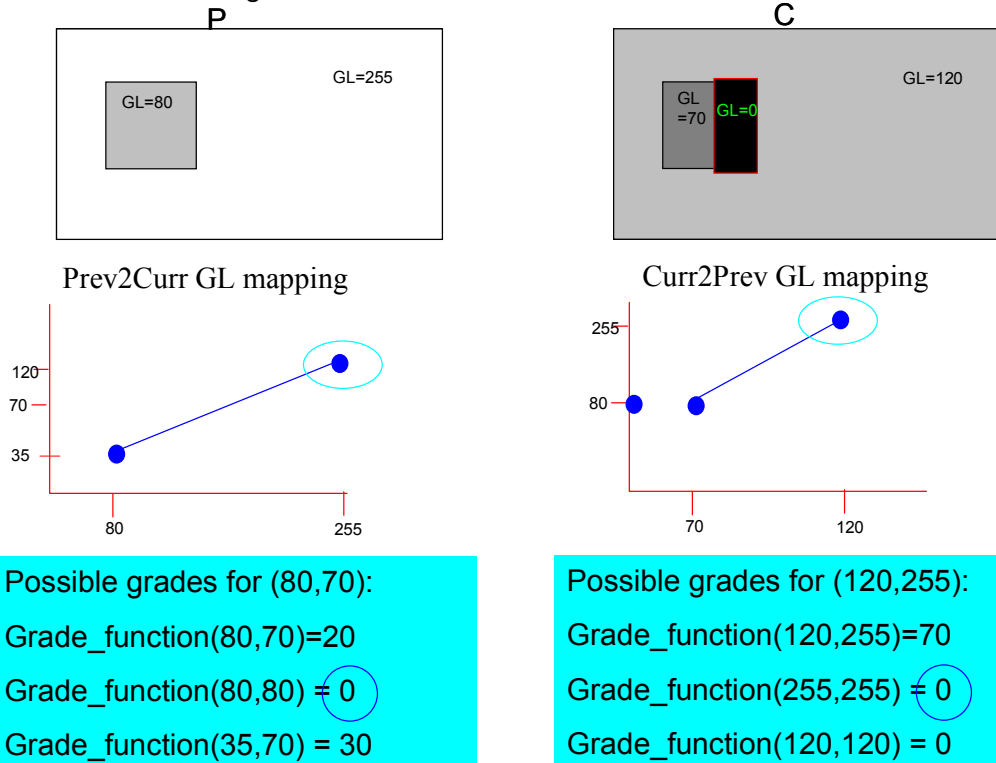
כמובן שבהינתן עבור כל bin בהיסטוגרמה ה- sum , count , sum^2 שלו ניתן לחשב בקלות את הממוצע (Mean) ואת סטיית התקן (STD) של כל אחד. לפי המוצע באלגוריתם המיפוי יהיה זה שסטיית התקן שלו היא מינימאלית או מתחת לסף מסוים ואם אף אחד מהמיפויים לא מקיים תנאי זה ניתן להחליט שעבור רמת האפור הללו אין גילוי (ובמקרה זה נבצע מיסוך כלומר ניתן לו את הציון אפס) או לקחת את ההשוואה המקורית של זוג הפיקסלים המקורי במידה וההפרש שלו המינימאלי ביותר מזה של שני המיפויים. אפשרות אחרת שהיא די שקולה תהיה לקחת את ההפרש המינימאלי מבין שני המיפויים ומבין הזוג המקורי.

עבור כל זוג C, P של GL $P = GL_{\text{Previous}}$; $C = GL_{\text{Current}}$ נבצע את הפעולה הבאה:
נחשב 3 הפרשים בצורה הבאה:

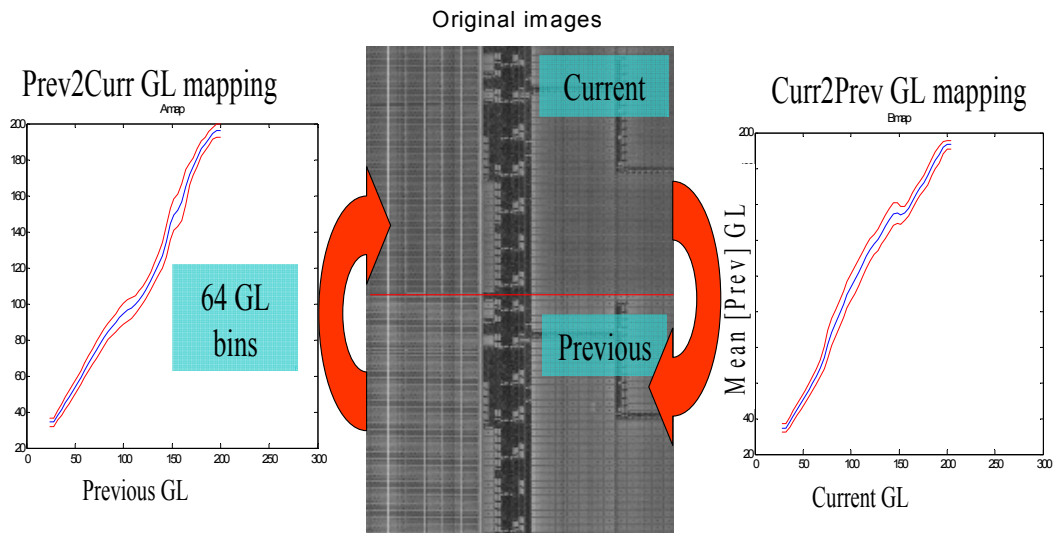
$$\begin{aligned} &(C, P) \\ &(\text{Mean}(P), P) \\ &(C, \text{Mean}(C)) \end{aligned}$$

משלושת ההפרשים נבחר את המינימאלי ביותר. המטרה של מדידת ה- STD היא לקבל מדד על אופי המיפוי. אם ה- STD גדול מדי כנראה שמדידת המיפוי לא אמינה ולכן כדי לוותר על התוצאות של מיפוי זה ולהשתמש רק בהשוואה המקורית ו/או המיפוי השני. דוגמא סינטטית המדגימה את הפעולה מתוארת באיור הבא:

- 2 blocks after registration:

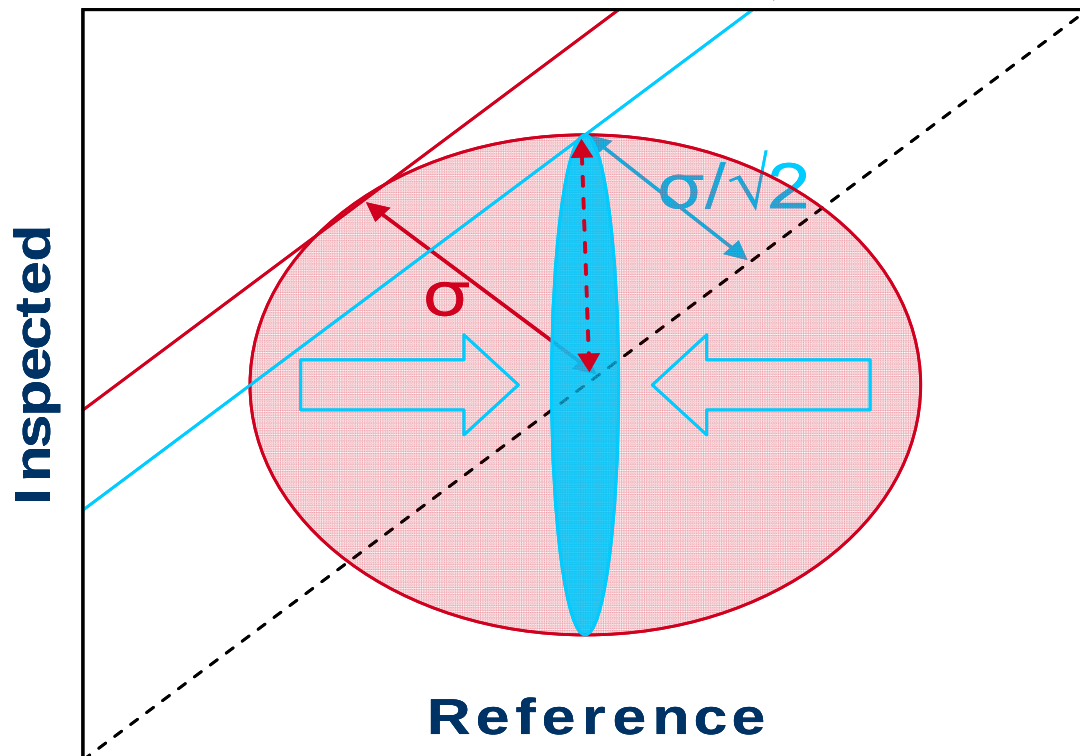


איור 31 תאור דוגמא לתוצאות מיפוי בהתייחס לפונקציית ההשוואה.
 בדוגמא שבאיור ניתן לראות שמתוך שלושה זוגות הפיקסלים המשווים (המקורי, C2P, P2C) אנו מקבלים השוואה של 80 עם 80, 255 עם 255 וכמובן שזוהי ההשוואה הנכונה כי אין כאן בעצם פגם אלא רק תופעת color variation.
 בגרפים הבאים ניתן לראות את תוצאות המיפויים עבור שני בלוקים מ-die מסוג logic.



איור 32 תאור המיפויים עבור 2 בלוקים עוקבים
 ניתן להסתכל על אופן מימוש האלגוריתם הכולל את מדידת תופעת ה-color variation כפי שהוצגה לעיל ותיקון תוך שימוש בהפרשים התלויים במיפויים כהשוואה שבעצם מייחסת לכל ערך gray level השוואה עם מועמדים בתוך ה-die שלו עצמו כלומר אנו מממשים השוואה של כל פיקסלים עם אוכלוסיית

המועמדים בתוך ה- die עצמו. המטרה של הפיקסלים מה- die השכן הוא בעצם לשמש כמאפיין של הפיקסלים ב- die הנוכחי. כלומר בעצם המיפויים עזרו לנו לבצע השוואות המבוססות על מידע הנמצא בתוך ה- die עצמו. כלומר בעזרת המיפויים אנו מבצעים למעשה הטלה מ- die אחד לשני ולהיפך והפיקסלים מה- die השכן עוזרים לנו בבחירת המיצועים המתאימים עבור ערכי רמות אפור מתוך ה- die הנוכחי. החיסרון בשיטה זו שבמידה ואין מספיק סטטיסטיקה עבור כל ערכי רמות אפור (או בצורה יותר מעודנת לכל קבוצה קטנה של ערכי רמות אפור) המיפויים עלולים שלא לשקף נכונה את התופעה מכיוון שפעמים רבות האזורים הללו קטנים יחסית לכל ה- die. בהתייחס לשיפור הגילוי ניתן להסיק שבמידה וכן יש מדגם סטטיסטי אמין מההשוואות בין dies עוקבים ניתן לקבל ייחוס השוואה (predictor pixel) נקי מרעש ככל האפשר כתוצאה מכך ה- SNR של תמונת ההפרשים גדל ולכן באופן טבעי ה- false alarm rate קטן ויכולת הגילוי גדלה. לפי תורת הסטטיסטיקה ככל שאנו ממצעים יותר פיקסלים כך רמת הרעש יורדת. לצורך המחשה נתבונן בבלוקים הבאים המציינים מקומות משני dies עוקבים. בהנחה שהרעש מתפלג גאוסיאני כלומר הוא רעש אקראי בעל ממוצע אפס. אם נבדוק את התפלגות הרעש בתמונת ההפרשים שבה בכל תמונה הרעש הוא בגודל σ נקבל שהרעש בתמונת ההפרשים הוא $\sqrt{2}\sigma$. האיור הבא מתאר מרחק ההשוואה עבור שתי תמונות כאלה.



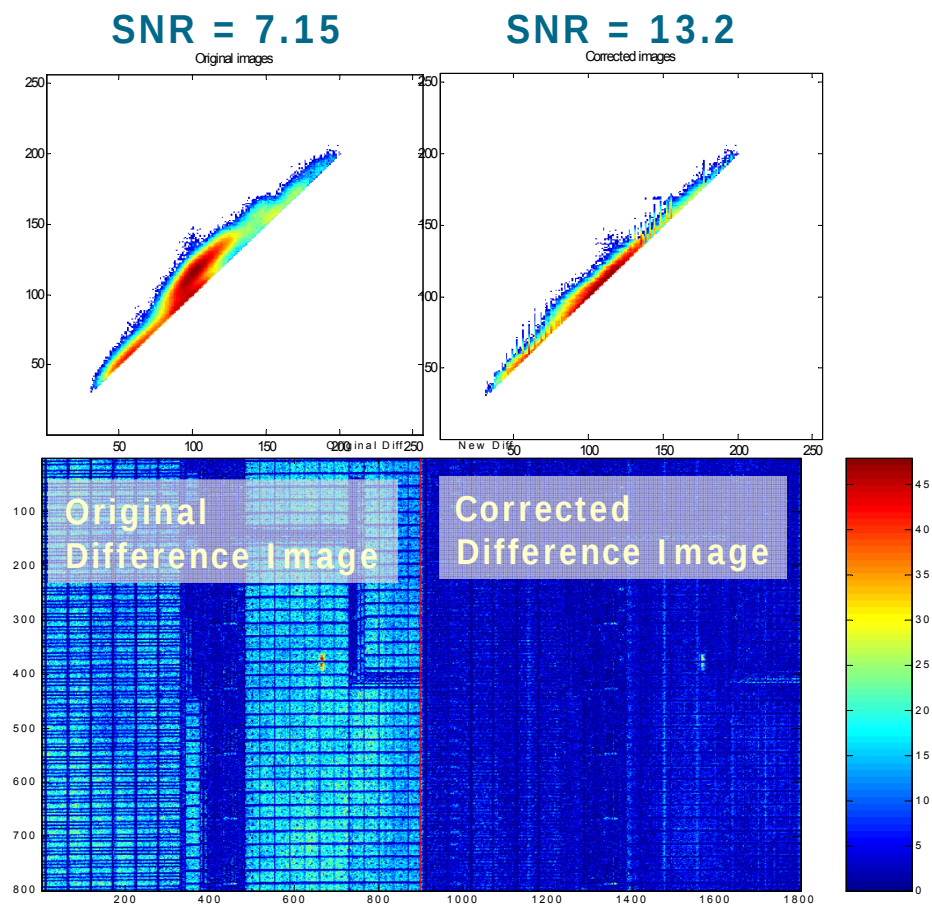
איור 33 דוגמא לתוצאות האלגוריתם הנ"ל על שני בלוקים עוקבים.
הפגם הוא בעצם המרחק מהקו המקווקו (המסומן בשחור). כאשר מבצעים הפרש רגיל בין שתי תמונות רועשות הרעש הוא בגודל $\sqrt{2}\sigma$. ככל ש- N גדל הגאוסיות (העיגול) נהייה צר יותר (אליפסה בתכלת) עד שכאשר $N \rightarrow \infty$ הוא הופך לקו כלומר תמונת הייחוס נקייה מרעש ומייצגת את המקור. במצב זה מקבלים (קו מקווקו באדום) שמרחק ההשוואה האפשרי הוא בגודל σ כי נשארו עם הרעש רק בתמונת הנבדקת (inspected) כאשר תמונת הייחוס נקייה מרעש. מכיוון שרמת הרעש מופחתת מקבלים שלתמונת ההפרשים לאחר הפעלת האלגוריתם יש יחס אות לרעש המאפשר גילוי (שלא כמו בהשוואה שאין בה תיקון של color variation). הניתוח המתמטי הבא מחזק את הקביעות שהועלו כאן:
יהי $X_i, Y_i \sim (\mu, \sigma^2)$ משתנים בלתי תלויים בעלי התפלגות אחידה ומציינים את הפגם ואת תמונת הייחוס.

יהי N מספר הנקודות שעברו מיצוע (averaging) בתמונת הייחוס.

$$STD(Y_i - \frac{1}{N} \sum_{i=1}^N X)_i = \sqrt{\sigma^2 + \frac{\sigma^2}{N}} = \sqrt{\frac{N+1}{N}} \sigma \rightarrow N \rightarrow \infty \approx \sigma$$
 סטיית התקן תהיה σ
 כלומר עבור $N=2$ נקבל שאם הרעש בשתי התמונות הוא σ אז הרעש של הפרש התמונות הוא $\sqrt{2}\sigma$.
 אם ניקח N מספיק גדול נקבל:

$$\frac{\text{Reduced Noise}}{\text{Noise STD}} = \frac{\sqrt{\frac{N+1}{N}} \sigma}{\sqrt{\frac{3}{2}} \cdot \sigma} \xrightarrow{N \rightarrow \infty} \sqrt{\frac{2}{3}} \approx 0.816$$

כלומר נוכל לצפות לשיפור של כמעט 19% בשיפור ברמת הרעש של תמונת ההפרשים.
 דוגמא להרצת אלגוריתם על דוגמת בלוק עם color variation. ניתן לראות שהמיפוי עזר להשיג SNR גבוה יותר של תמונת ההפרשים כאשר יכולת הגילוי השתפרה בלא תוספת של גילוי פגמי שווא.



איור 34 דוגמא לתוצאות האלגוריתם הנ"ל על שני בלוקים עוקבים.

4.1.4 בחינת ביצועי אלגוריתמי השוואה מבוסס Die2Die

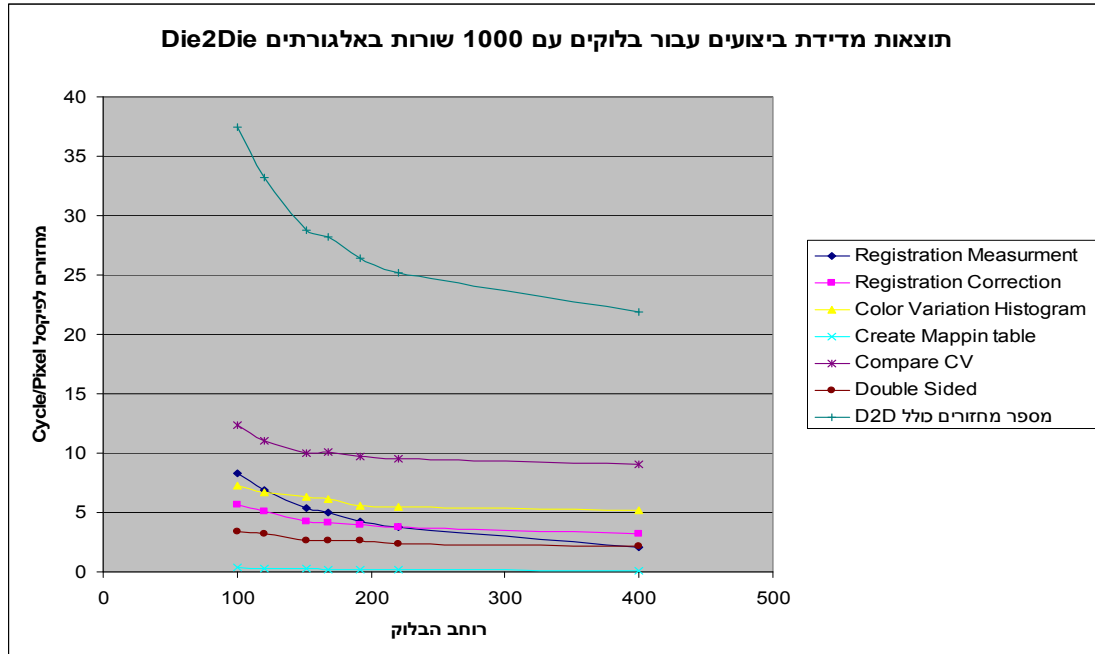
- בחינת ביצועי האלגוריתמים הנוכחי (וגם הבאים) תעשה ע"פ מספר פרמטרים כדלקמן:
- אחוז דיווח על פגמים שיקרים, (false alarm rate) כלומר כמה אחוזים מהפגמים המדווחים הם פגמי שווא.

- יחס אות לרעש (signal to noise ration=SNR) של תמונת הבלוק הנוכחי מול תמונת הייחוס. במקרה של האלגוריתם למציאת פגמים נבחן את יחס אות לרעש של תמונת ההפרשים הסופית.
 - עלות ביצוע האלגוריתם בהתייחס לזמן הריצה. כמו שהוגדר בפרקים הבאים התקציב לכל אלגוריתמים הוא בסדר גודל של 30-40 מחזורי מעבד בממוצע עבור כל פיקסל.
 - מספר או אחוז הפגמים הידועים שלא התגלו.
- נקודת ההתייחסות הבסיסית היא לביצוע השוואה die2die ללא הפעלת יישור ההיסט וללא תיקון תופעת ה-color variation אבל לא ננתח את ביצועי האלגוריתם die2die הבסיסי מכיוון שהוא תיאורטי מדי (בגלל אי טיפול בבעיית ההיסט וה color variation) ולכן אין משמעות במימושו אלא רק לצרכי ייחוס בהשוואות ביצועים. תוצאות ריצה אלו מתוארות [בנספח 4 \(תוצאת ריצת אלגוריתם d2d בסיסי\)](#). כאמור אלגוריתם השוואה מבוסס die2die (עם מדידת ההיסט ותיקונו ומדידת תופעת ה-color variation ותיקונה) מורכב כמתואר לעיל מהשלבים הבאים:
- מדידת ההיסט בין שני בלוקים עוקבים (registration measurement). כל בלוק חולק כל באורך 64 פיקסלים. בכל רצועה כזו התבצעה מדידת ההיסט בעזרת שתי מטריצות קורלציה האחת עבור מדידת ההיסט בציר ה-X והשני בציר ה-Y. לדוגמא עבור בלוק באורך 1024 שורות ישנם 16 חלונות מדידה עבור כל אחד מהצירים.
 - תיקון התזוזה (registration correction) ע"י העברת kernel 4x4 שהוא תוצאה של גרעינים (kernels) חד ממדיים מופרדים בגודל 1x4, 4x1. עבור כל רצועה (stripe) מופעלים גרעין המתאים לו.
 - חישוב היסטוגרמות המיפוי שתוארו לעיל (color variation histogram)
 - מתוך היסטוגרמות המיפוי חשוב הממוצע (Mean) וסטיית התקן (STD) עבור כניסה (bin) בהיסטוגרמה (create mapping table).
 - חישוב פונקצית ההשוואה המנורמלת (compare CV) הבוחרת את הציון המינימאלי מבין שלושת הזוגות (c,p) (mean(c),c) (mean(p),p) (double sided comparison).
 - חישוב פונקצית ההשוואה הכפולה (double sided comparison).
- התוצאה של הפונקציה האחרונה היא רשימת התראות (alarms). לכל alarm מתואר מיקומו בבלוק (ציר Y וציר X) וציון המציין את ההסתברות שלו להיות פגם. בנוסף לכך לכל alarm ישנו תיאור של ערכי הפיקסלים משלושה dies עוקבים במיקומים של ה-alarm. הטבלה הבאה מתארת את זמן הריצה של כל אחד מהשלבים באלגוריתם וכן את הסיכום הכולל. תוצאות הריצה הינם בהתייחס לזמן ריצה ביחידות של cycle/pixel.

מספר מחזורי כולל לפיקסל D2D	Double Sided	Compare CV	Create Mapping Table	Color Variation Histogram	Registration Correction	Registration Measurement	רוחב בלוק
37.48	3.4	12.4	0.38	7.3	5.7	8.3	100
33.20	3.2	11	0.32	6.7	5.1	6.88	120
28.75	2.6	10	0.25	6.3	4.2	5.4	152
28.24	2.66	10.1	0.23	6.1	4.15	5	168
26.38	2.6	9.7	0.20	5.6	4	4.28	192
25.15	2.38	9.55	0.17	5.5	3.75	3.8	220
21.85	2.2	9.1	0.10	5.2	3.2	2.05	400

טבלה 1 טבלת זמני ריצה של die2die עבור בלוקים בעלי 1000 שורות

בצורה גרפית:



איור 35 תוצאות זמני ריצה של die2die עבור בלוקים עם 1000 שורות
בנוסף להלן תוצאות הריצה של אותם ריצות עבור בלוקים עם 500 ו- 2000 שורות.

מספר מחזורים כולל D2D	Double Sided	Compare CV	Create Mapping table	Color Variation Histogram	Registration Correction	Registration Measurement	רוחב בלוק
37.89	3.5	12.5	0.76	7.4	5.75	8.36	100
33.62	3.3	11.1	0.64	6.8	5.2	6.9	120
29.75	2.8	10.2	0.5	6.5	4.5	5.5	152
28.53	2.7	10.2	0.46	6.2	4.2	5	168
26.90	2.65	10	0.4	5.7	4.05	4.3	192
25.27	2.4	9.6	0.34	5.5	3.8	3.8	220
22.20	2.2	9.2	0.2	5.3	3.3	2.1	400

טבלה 2 טבלת זמני ריצה של die2die עבור בלוקים עם 500 שורות

מספר מחזורים כולל D2D	Double Sided	Compare CV	Create Mapping table	Color Variation Histogram	Registration Correction	Registration Measurement	רוחב בלוק
36.81	3.3	12.4	0.19	7.24	5.67	8.2	100
32.65	3.2	11	0.16	6.7	5.05	6.7	120
28.10	2.6	9.9	0.125	6.1	4.1	5.4	152
27.33	2.63	9.7	0.115	6	4.1	4.9	168
26.20	2.6	9.75	0.1	5.6	4	4.25	192
24.75	2.35	9.55	0.085	5.4	3.7	3.75	220
21.45	2.1	9.1	0.05	5.1	3.15	2	400

טבלה 3 טבלת זמני ריצה של die2die עבור בלוקים עם 2000 שורות

הערות לגבי תוצאות הריצה:

- במדידת ההיסט בין שני בלוקים עוקבים (registration measurement) משתמשים במדידת קורלציה אחת עבור תבניות בציר X ואחת עבור ציר Y עבור רצועת בלוק באורך של 64 שורות. מכיוון שמספר הקורלציות (חלונות הרגיסטרציה) לא משתנה בתלות ברוחב הבלוק ברור שהגדלתו תורמת לביצועים גבוהים יותר, כלומר פחות מחזורים לפיקסל אולם עלולים

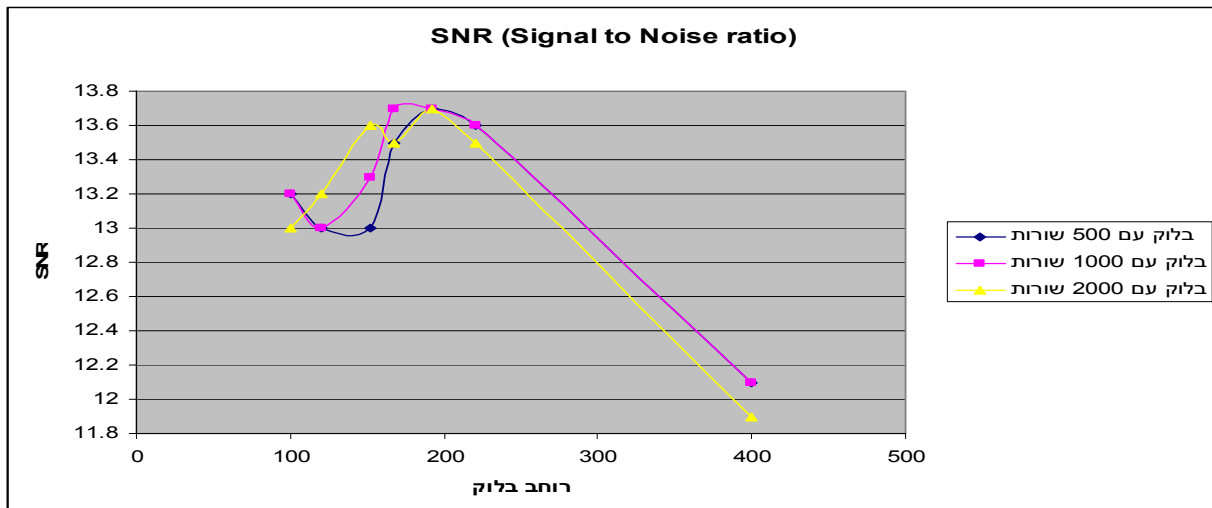
להשפיע על איכות התוצאה מכיוון שכעת שטחים גדולים מיושרים (registration correction) על בסיס מדידות שיכולות להיות מרוחקות יותר מהמקום בו נמצאים הפיקסלים הרלוונטיים.

- תוצאות הריצה של פונקצית יצירת הטבלאות פועלת על ההיסטוגרמות אבל אנו מתייחסים במדידת הביצועים ל- cycle/pixel ולכן נקבל שככל שהבלוק יותר גדול כך הביצועים מהירים יותר ואין השפעה של רוחב הבלוק (ואורכו) על הביצועים אלא רק למספר הפיקסלים בבלוק.
- ניתוח זמני הריצה של אלגוריתם מראה בבירור שהגדלת גודל הבלוק בציר ב X תורמת לכך שהאלגוריתם רץ מהר יותר כלומר אנו צורכים פחות cycle/pixel כדי לממשו. ההסבר לכך נובע מהעובדה שבמימוש הפונקציות הלולאה היעילה (tight loop כפי שהוסבר [בנספח 4](#)) רצה על שורות בציר ה- X וכמובן שככל שהיא ארוכה יותר כך אנו נמצאים פחות בחלק ה- prolog ו- epilog ולכן מגיעים לניצול מלא של טכנולוגיית ה- pipeline. הסיבה לכך שבחלק מהפונקציות הגדלת מספר השורות בבלוק תורמת לשיפור מועט בביצועים נובעת מהעובדה שהעברות ה- EDMA יעילות יותר מכיוון שככל שהבלוק ארוך יותר כל החלק ההתחלתי של ההעברה הראשונה והאחרונה (שאינם מתבצעות במקביל לעיבוד) נהיה קטן יותר. כלומר טכנית גם אם נגדיל את כמות המידע לעיבוד ונחלק אותו ליחידות (בלוקים) גדולות יותר המערכת מעבדת ביעילות יותר ולכן קצב הסריקה לא יפגע (כמובן אם ניתן לכל DSP בלוק רחב יותר). בבלוקים עם רוחב של סדר גודל של 400 הניצול של ה- pipeline הוא כמעט מלא והגדלת הבלוק תוך כדי השארת מספר הליבות כמו שהוא יגרום כבר לירידה בביצועים. בחינת ביצועי הגילוי של האלגוריתם תעשה ע"י בדיקת ה- SNR של תמונת ההפרשים של הזוג (מבין השלושה) שנתן את הציון המינימאלי וכן נבדוק את אחוז גילוי פגמי השווא. להלן תוצאות ה- SNR.

רוחב בלוק	בלוק עם 500 שורות	בלוק עם 1000 שורות	בלוק עם 2000 שורות
100	13.2	13.2	13
120	13	13	13.2
152	13	13.3	13.6
168	13.5	13.7	13.5
192	13.7	13.7	13.7
220	13.6	13.6	13.5
400	12.1	12.1	11.9

טבלה 4 טבלת SNR עבור בלוקים בגדלים שונים

בצורה גרפית:



איור 36 סיכום מדד ה- SNR עבור בלוקים בגדלים שונים

בחינת תוצאות מדד ה- SNR מראה שעבור רוב גדלי הבלוקים השונים התוצאות די דומות אם כי ישנה רגישות כלשהי הנובעת מכך ששינוי במספר הסטטיסטיקות בבלוק יכול לגרום להפחתת רעש שונה. בנוסף לכך חלוקת הבלוק לגדלים שונים גורמת לשינויי מדידת ההיסט במיוחד התופעה בולטת בבלוק

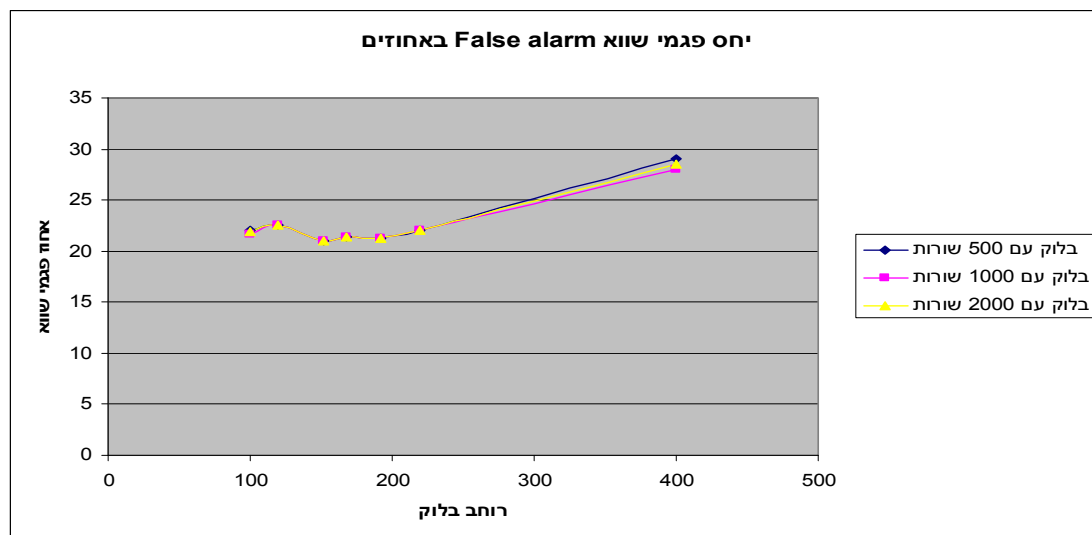
רחב (400 פיקסלים) שבו מדידת ההיסט עם מספר חלונות זהה לכל רצועה גורמת לכך שישנם שטחים המתוקנים לפי מדידות שאינן מדויקות מספיק ולכן ההזזה הלא מדויקת תורמת להעלאת הרעש. כמובן שניתן לחשב עבור בלוקים רחבים יותר מספר חלונות נוסף לפיקסלים מ-200 עד 400 אבל כמובן שנשלם על כך בתוספת cycles לעיבוד.

בחנית פרמטר אחוז פגמי השקר false alarm rate נעשתה על wafer עם פגמים ידועים מראש כדי לאמת פרמטר זה. באופן עקרוני ככל שה- SNR גבוה יותר יש סיכוי שאחוז הפגמים השקרים כתוצאה מזיהוי של רעש (מדידת ההיסט, או מ- color variation) כפגם יפחת. עדיין יכולים להיות פגמי שווא בגלל סיבות אחרות. להלן התוצאות:

רוחב בלוק	בלוק עם 500 שורות	בלוק עם 1000 שורות	בלוק עם 2000 שורות
100	22	21.7	21.9
120	22.5	22.5	22.5
152	21	21	21
168	21.4	21.4	21.4
192	21.3	21.3	21.3
220	22	22	22
400	29	28	28.5

טבלה 5 טבלת FA באחוזים עבור בלוקים בגדלים שונים

בצורה גרפית:



איור 37 סיכום מדד ה- FA עבור בלוקים בגדלים שונים

מעיון בתוצאות רואים שיש קורלציה הפוכה בין מדד ה- SNR ל- FA והדבר ברור מכיוון שככל שניטב לבצע הפרדה טובה של האות מהרעש ונגדיל את ה- SNR נפחית את אחוז ה- FA. מסיכום התוצאות רואים שהאלגוריתם רגיש באופן חלוקת המידע בין המעבדים מכיוון שאז תיקון ההיסט משתנה וכן המדידות המקומיות של ה- color variation.

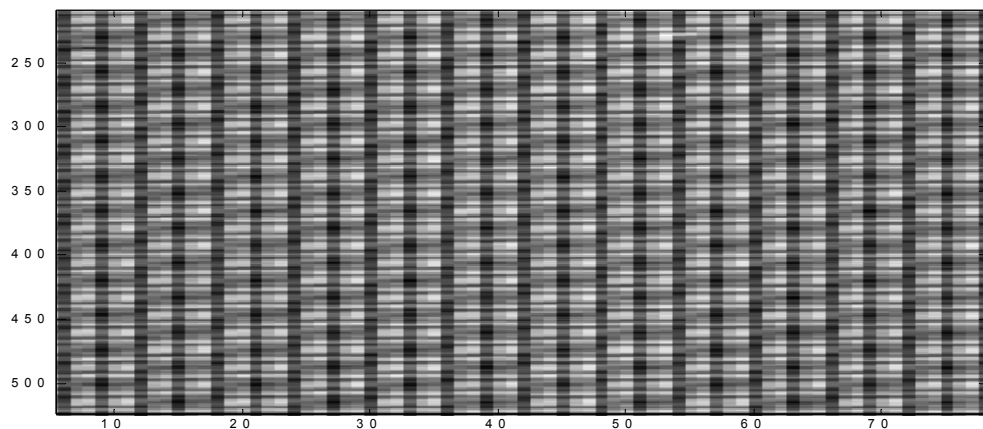
ניתן כמובן לשפר אלגוריתם זה ע"י שיפור תהליך מדידת ההיסט וכן לשפר את יחס ה- FA עבור פגמים שעדיין עוברים. אפשרות אחת היא לטפל בפגמים שעברו (שהם פרומיל מסך המידע) ע"י אלגוריתם מורכב יותר כך שיעבוד רק על תמונת הסביבה של הפגם וינסה לקבוע אם הוא פגם או דיווח שקר. בכך נוכל לשפר את יחס ה- FA.

בנוסף כדי לשפר את ה- SNR ניתן לבצע קיטוע (segmentation) של הבלוק לאזורים רועשים יותר ורועשים פחות לפי מאפיינים שונים ולכלל פיקסל לתת מאפיין רעש. פעולה זו תאפשר לתת ספים שונים לאזורי רעש שונים. כמובן שפעולה זו תצריך יצירת תמונה נוספת שתהווה קלט נוסף לכל הפונקציות שבתהליך וכמובן תגדיל את ה- cycle/pixel. הלכה למעשה אנו עובדים בבלוקים שרוחבם הוא בסדר גודל של 200 פיקסלים ואורכם הוא בסדר גודל של 1000 שורות.

5 אלגוריתמי השוואה Cell to Cell למציאת פגמים

המידע בסעיף זה נלקח ברובו ממאמרים [23] [9] [11] [6] [7] [8] [25] [29] בנוסף לחומר כללי הנלמד בקורסים לעיבוד תמונה.

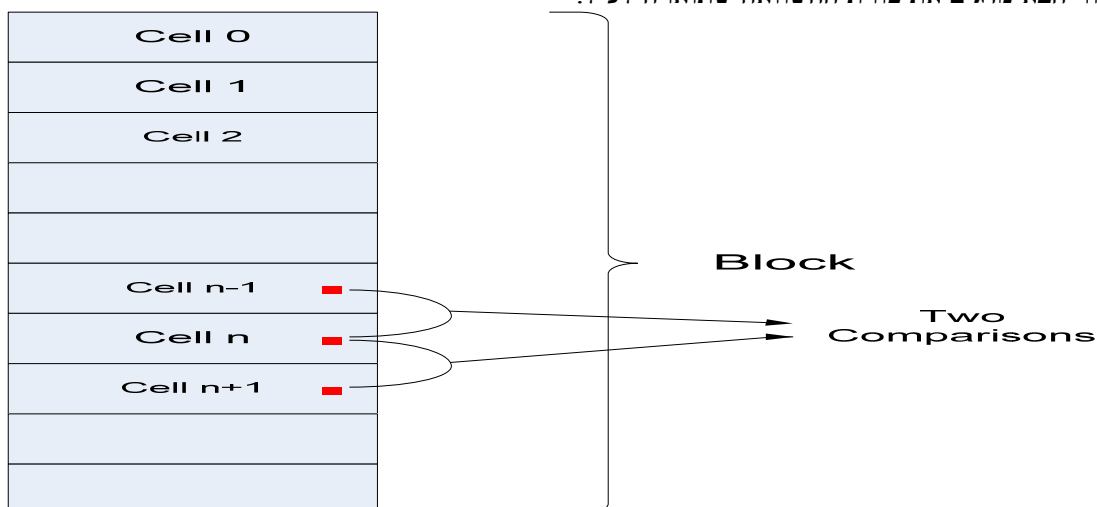
כאמור בשכבות ואפליקציות רבות המשטחים (בעיקר ברכיבי DRAM, SRAM) בנויים מיחידות מחזוריות הנקראות תאים (cells). מכיוון שהשכבות עליהם פועלת מכונת ה-WI ידועות לפני תהליך הסריקה ניתן לבצע אלגוריתם מורכב ככל שיהיה לפני תחילת הסריקה ולמצוא בו את המחזוריות המתאימה ביותר לביצוע האלגוריתם. קיומם של מחזורים מאפשר לנו להשתמש בהשוואות בתוך אותו die. היתרון בביצוע השוואות בתוך אותו die הוא שבעיית יישור ההיסט כבר לא קיימת (אין צורך ליישר שני dies סמוכים כי ההשוואה נעשית בתוך אותו die ומרחק ההשוואה קטן) ובנוסף בעיית ה-color variation נחלשת מכיוון שמרחקי ההשוואה קטנים יותר ולכן גם שינויי הצבע מעטים יותר (למרות שעדיין הם יכולים להשפיע בהשוואות מרובות הדורשות מרחקי השוואה ארוכים). לצורך העניין התא (Cell) הטוב ביותר איננו חייב להיות המינימאלי אולם לא נעסוק בעבודה זו באלגוריתם למציאה ולהחלטה על גודלו של התא.



איור 38 דוגמא לאזור מחזורי

5.1 תיאור אלגוריתם השוואה מבוסס Cell2Cell

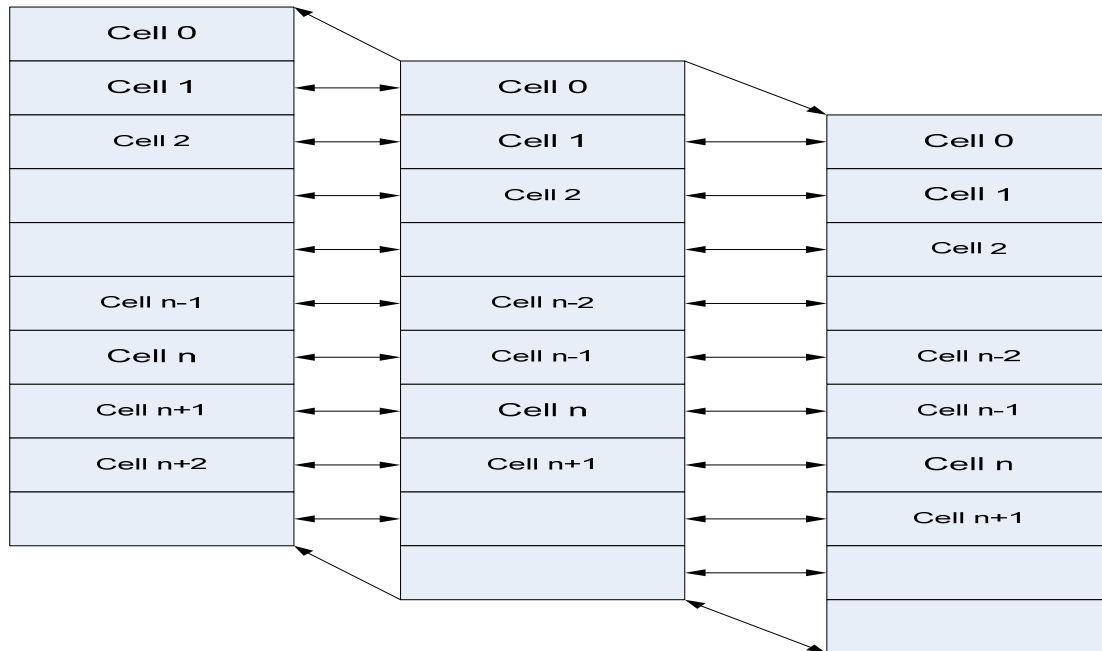
עקרונית אלגוריתם השוואה מבוסס cell2cell מתבסס על השוואה כפולה בדומה לזו שהוצגה ב-die2die אלא שכאן יצירת אזורי ההשוואה $n-1$ ו- $n+1$ נעשית ע"י הזזה של הבלוק הנוכחי ב cell אחד קדימה ו- cell אחד אחורה בהתאמה יחסית ל cell הנוכחי שעליו רוצים לגלות את הפגם. ע"י ההזזות מקבלים בלוקים וריטואלים שאיתם ניתן להשוות את הבלוק המקורי. האיור הבא מדגים את צורת ההשוואה שתוארה לעיל:



איור 39 תיאור ההשוואות באלגוריתם cell2cell

מימוש ההשוואות נוספות יכול להתבצע גם בציר X אך מכיוון שלעתים קרובות יש עיוותים בציר X אין אפשרות מעשית לממש זאת לכן ההשוואות יהיו בציר Y בלבד.

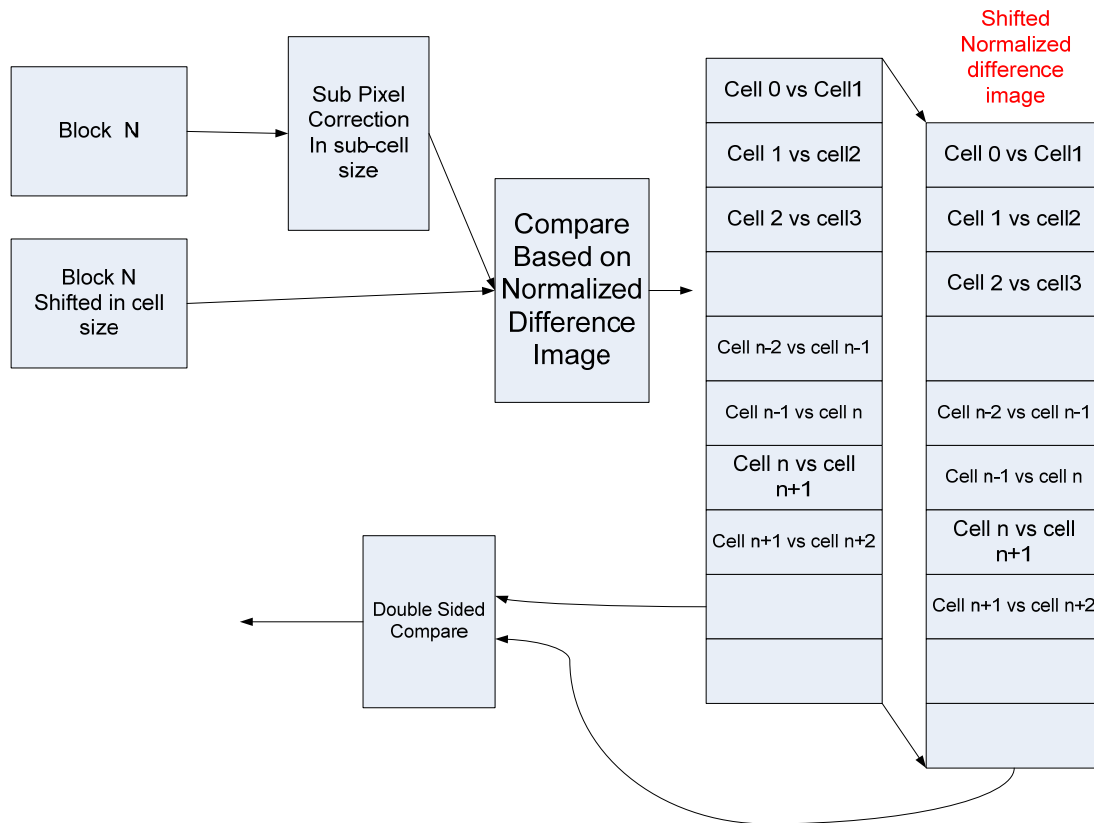
כדי לממש את האמור לעיל עלינו ליצור תמונות ייחוס שאותם נוכל להשוות בעזרת פונקצית השוואה בדומה לזו שביצענו ב- die2die (פונקצית השוואה בסיסית). נשתמש כאן באותם מינוחים שהשתמשנו באלגוריתם die2die. ניקח את בלוק המקור ונזיז אותו הזזה בפיסקל שלם כלומר הזזת מצביעים בלבד. אם נזיז את בלוק המקור קדימה נקבל את הייחוס מול המקור, ובאותו אופן אם נזיז אחורה את הבלוק (כל ההזזות הם בעצם הזזות של מצביעים) נקבל ייחוס של בלוק מקור מול הייחוס הקודם. לאחר הזזת בלוק המקור בגודל cell אחורה ניתן לבצע את ההשוואה של בלוק n מול בלוק $n-1$ וכמובן לאחר הזזת בלוק המקור קדימה ב- cell ניתן להשוות את בלוק המקור מול $n+1$ כמודגם באיור הבא:



איור 40 מימוש ההשוואות n מול $n-1, n+1$ ע"י הזזת הבלוק ב cell-cell+

הבעיה העיקרית שעלינו לטפל בבואנו לממש אלגוריתם זה הוא שגודל התא איננו כפולה שלמה של פיקסל. כלומר לגודל של תא יש חלק שלם של פיקסל וחלק תת-פיקסלי שאותו מקבלים בזמן מדידת התא. במידה שנזיז רק את החלק השלם (הזזת מצביעים) נקבל השוואה רועשת כי בעצם התעלמנו מהחלק הלא שלם בפיסקל של התא. פתרון הבעיה יהיה כדלקמן: לאחר שהזזנו את הבלוק בכפולה שלמה של פיקסל נשתמש במקדמים שחישבנו עבור תהליך תיקון ההיסט בסעיף הקודם ונבצע הזזה תת-פיקסלית של בלוק הייחוס המוזז לעומת בלוק המקור. לאחר ההזזה התת-פיקסלית (כאשר מקדמי התיקון הם מסוג 4×1 בלבד כי מתקנים רק בכיוון Y) נוכל לבצע את ההשוואה.

למעשה האלגוריתם של cell2cell דומה באופן עקרוני לזה של die2die בכך שגם כאן אנו רוצים לבצע השוואה כפולה עבור כל cell ולאחר מכן לבצע את פעולת ה- double sided בדיוק כמו שמופיע באיור 16. מצד שני ב- die2die עבור כל בלוק מבצעים רק השוואה אחת עם הבלוק הקודם ולאחר מכן תוצאת שתי ההשוואות עוברת להשוואת double sided. אם במקרה הנוכחי נבצע שתי פונקציות השוואה על כל בלוק נפסיד יעילות ונשלם ב- cycles מיותרים שכאמור אנו רוצים לחסוך. כדי לטפל בסוגיה זו (כלומר לבצע רק פונקצית השוואה אחת compare על כל בלוק) ניתן להשתמש בעובדה שכאשר נייצר תמונות הפרשים מנורמלת בעצם בגלל ההזזה שעשינו עבור כל cell יש גם ההשוואה הנוספת כי התא השכן נמצא מעליו ולמעשה אין צורך לבצע השוואה פעם נוספת לתמונות ההפרשים המנורמלות (אחת מוזזת ביחס לשנייה נכנסות לביצוע double sided). הפעולה מתוארת באיור הבא:



איור 41 תיאור זרימה עבור אלגוריתם cell2cell.

5.2 בחינת ביצועי אלגוריתמי השוואה מבוססים Cell2Cell

בחינת ביצועי האלגוריתמים cell2cell המתואר בסעיף הקודם (וגם אלה המתוארים בסעיפים הבאים) תעשה ע"פ אותם פרמטרים שבהם בחנו את אלגוריתם die2die בסעיף הקודם. כאמור שלבי ביצוע האלגוריתם כפי שתוארו לעיל הם:

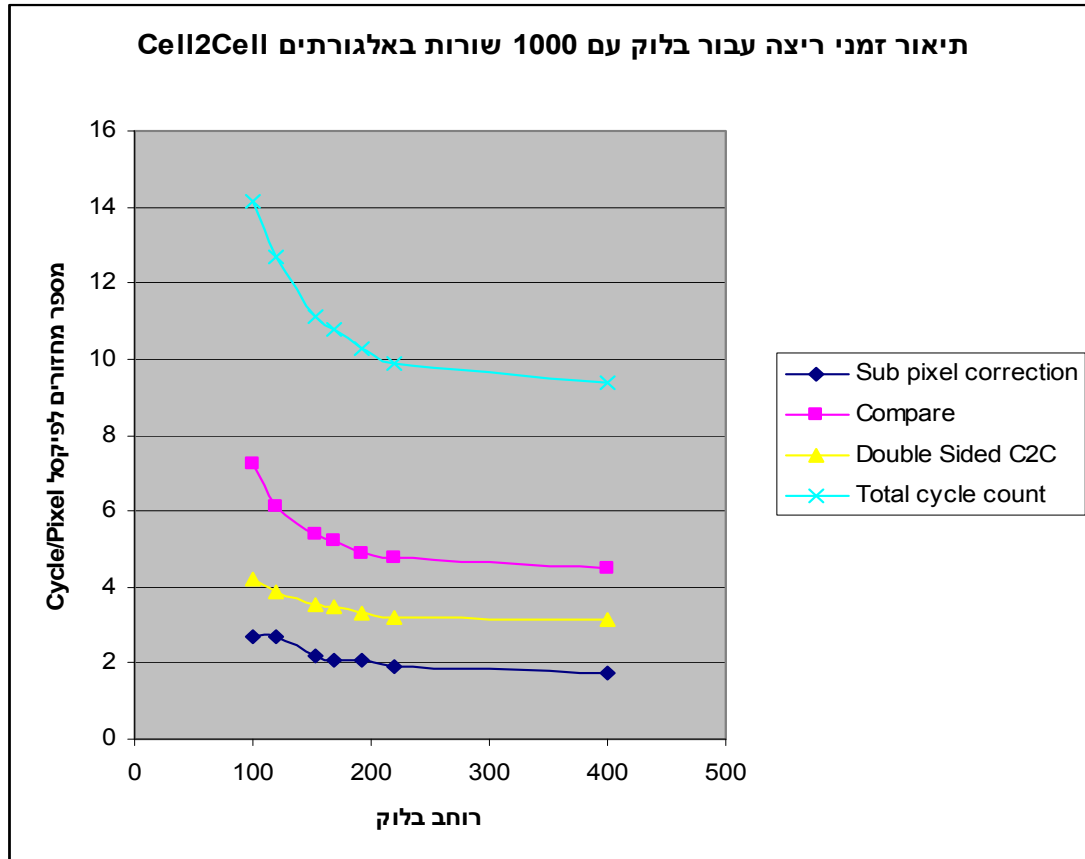
- תיקון תת פיקסלי של הבלוק כדי לתמוך בגודל תא שאיננו כפולה שלמה של פיקסל (sub pixel correction).
- ביצוע פונקציית השוואה לפי העיקרון שהודגם [בנספח 4](#).
- חישוב פונקציית ההשוואה הכפולה (double sided C2C) לפי אותו עיקרון שהוסבר ב-die2die.

התוצאה של הפונקציית האחרונה היא רשימת מועמדים לפגמים (alarms) שבו לכל כזה יש מיקום בבלוק וציון המציין את ההסתברות שלו להיות פגם וכן תמונת alarms המציינת ערכים אלו לכל פיקסל. הטבלה והגרף הבא מסכמים את זמני הריצה עבור בלוקים בעלי 1000 שורות.

רוחב בלוק	Sub pixel correction	Compare	Double Sided C2C	Total cycle count
100	2.7	7.27	4.2	14.17
120	2.7	6.1	3.9	12.7
152	2.17	5.4	3.55	11.12
168	2.1	5.2	3.5	10.8
192	2.05	4.9	3.3	10.25
220	1.9	4.8	3.2	9.9
400	1.75	4.5	3.15	9.4

טבלה 6 טבלת זמני ריצה של cell2cell עבור בלוקים עם 1000 שורות

בצורה גרפית:



איור 42 תיאור זמני ריצה של cell2cell עבור בלוקים עם 1000 שורות

תוצאות הריצה עבור בלוקים עם 500 ו- 2000 שורות מסוכמות בטבלאות הבאות:

רוחב בלוק	Sub pixel correction	Compare	Double Sided C2C	Total cycle count
100	2.8	7.3	4.3	14.4
120	2.4	6.2	4	12.6
152	2.2	5.5	3.6	11.3
168	2.15	5.4	3.5	11.05
192	2.1	5.2	3.3	10.6
220	2	5	3.2	10.2
400	1.9	4.6	3.15	9.65

טבלה 7 טבלת זמני ריצה של cell2cell עבור בלוקים עם 500 שורות

רוחב בלוק	Sub pixel correction	Compare	Double Sided C2C	Total cycle count
100	2.7	7.12	4	13.82
120	2.68	6.1	3.9	12.68
152	2.17	5.2	3.5	10.87
168	2.1	5.1	3.44	10.64
192	2	4.7	3.25	9.95
220	1.88	4.7	3.2	9.78
400	1.7	4.5	3.15	9.35

טבלה 8 טבלת זמני ריצה של cell2cell עבור בלוקים עם 2000 שורות

פונקציית ההשוואה (compare) יוצרת כמו שהוסבר תמונה עם ציונים המבוססים על הפרשים מנורמלים (במידה שאכן ההפרש עבר את הסף המתאים) והיא דומה מאוד לפונקציית ההשוואה הבסיסית של

die2die מכיוון שזוהי כמעט אותה פונקציה למעט טיפול שונה במיסוכים הקשורים למיקומם של התאים בקצה הבלוק. ניתוח זמני הריצה של אלגוריתם cell2cell מראה תוצאות דומות לאלו של die2die. גם כאן הגדלת רוחב הבלוק בציר X תורמת לכך שהאלגוריתם רץ מהר יותר מאותם סיבות שהוזכרו לגבי die2die. מכיוון שמרחק ההשוואה קצר אין כמעט השפעה של גודל הבלוק על מדד ה-SNR ועל מדד ה-false alarm. ה-SNR הנמדד בשני wafers שונים היה 14.5, 13.8 ויחס פגמי השקר בהתאמה היה 15% ו-19%. לסיכום ניתן לומר שהיתרון של אלגוריתם cell2cell הוא שהוא יעיל מבחינת זמן ריצה שכן אין הוא צריך לטפל בבעיות רגיסטרציה (יישור ההיסט) ובבעיית color variation. מצד שני ישנם wafers (במיוחד בגודל פיקסל קטן) שבהם אנו סורקים בתנאים שבהם הגילוי הוא על סף הרעש כלומר קשה להפריד את אות הפגם מאות הרעש ולכן במידה והספים נמוכים אחוז פגמי השקר עולה ובמידה והם גבוהים מפסידים גילוי. אחת מהאפשרויות לשיפור אלגוריתם זה הוא לבצע מדידה מקומיות של הרעש ולעדכן דינאמית את הספים בהתאם אך לא נטפל בכך בעבודה זו.

6 אלגוריתמים להשוואת תא מחזורי למציאת פגמים

מכיוון שבלוק עיבוד מכיל מספר תאים נרצה לנצל את העובדה שהתאים הינם מחזוריים כדי לקבל ייחוס נקי ככל האפשר מרעש עבור כל פיקסל. הבעיה העיקרית באלגוריתם הקודם המבוסס על שתי השוואות בין תמונת המקור לתמונה מוזזת הוא שמכיוון ששתי התמונות רועשות (גם המקורית וגם המוזזת) תמונת ההפרשים רועשת יותר משניהם מכיוון שה-STD של הרעש גדל בגורם $\sqrt{2}$ (בדומה להסבר שניתן ב-die2die). המטרה באלגוריתם זה הוא לשפר את ה-SNR של תמונת הייחוס ובכך לשפר את הגילוי של פגמים אמיתיים ולהקטין גילוי של פגמי שקר. המטרה במימוש אלגוריתם המבוסס על מספר רב של השוואות (ולא רק שתי השוואות כמו ב-cell2cell) הוא לשפר את הגילוי ע"י מציאה ושיחזור של התא המקורי מאוסף התאים הנמצאים בבלוק ויחד עם זאת עדיין לקבל אלגוריתם שמבחינת זמן ריצה יהיה חסכוני באותו סדר גודל של אלגוריתם cell2cell. קיימות שתי גישות עיקריות לבניית האלגוריתם כפי שמוצגים במאמרים [6],[7],[8],[9],[10],[11].

- בשיטה הראשונה מתקיים ניתוח של התמונה במישור התדר על בסיס הידיעה שבתמונה קיימים אלמנטים מחזוריים שאנליזה שלהם במישור התדר אפשרית.
- בשיטה השנייה מתקיים כאמור ניתוח של התמונה במישור מרחב התמונה ע"י פעולות לינאריות או אלינאריות או שילוב שלהם.

6.1 ניתוח ועיבוד התמונה המחזורית במישור התדר

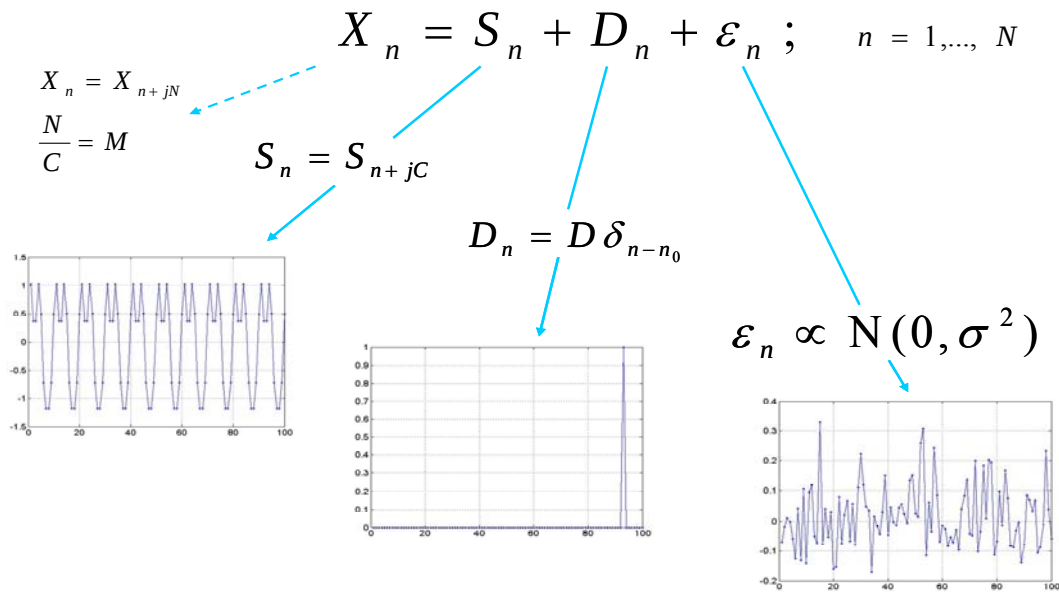
הרעיון המרכזי מתבסס על העובדה שתמונה ללא פגם וללא רעש מכילה רכיבים מחזוריים הניתנים לאיתור מתוך תמונת התמרת ה-FFT (במקור [9], [11] ישנם רעיונות לניתוח משטחים מחזוריים וחלק מהמידע בסעיף זה מבוסס על מאמרים אלו). לדוגמא אם ניקח תמונה של קווים ישרים אופקיים ואנכיים נוכל להבחין שיש להם רכיבים מובהקים במישור התדר. רכיבים אלה מיצגים כמובן את האות המחזורי של התא. לצורך פשטות הניתוח המתמטי נתייחס כאל התמונה כאות חד ממדי. נגדיר את האותות הבאים:

- X_n - התמונה המקורית.
- S_n - האות המחזורי הכולל את התאים (cells).
- D_n - הפגם (תקלה).
- ε_n - הרעש באות.

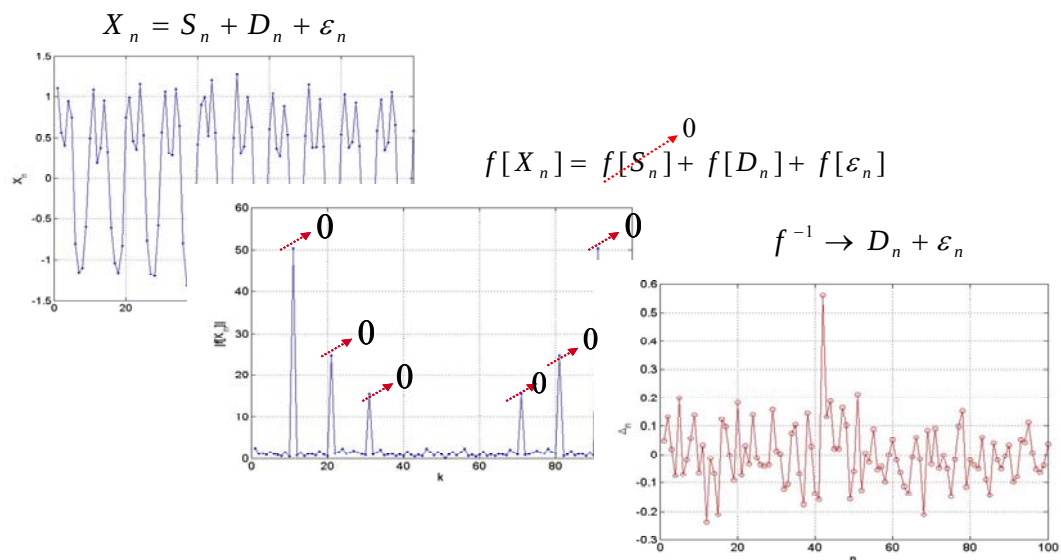
ניתן לבטא את האות הכולל בצורה הבאה:

$$X_n = S_n + D_n + \varepsilon_n ; \quad n = 1, \dots, N$$

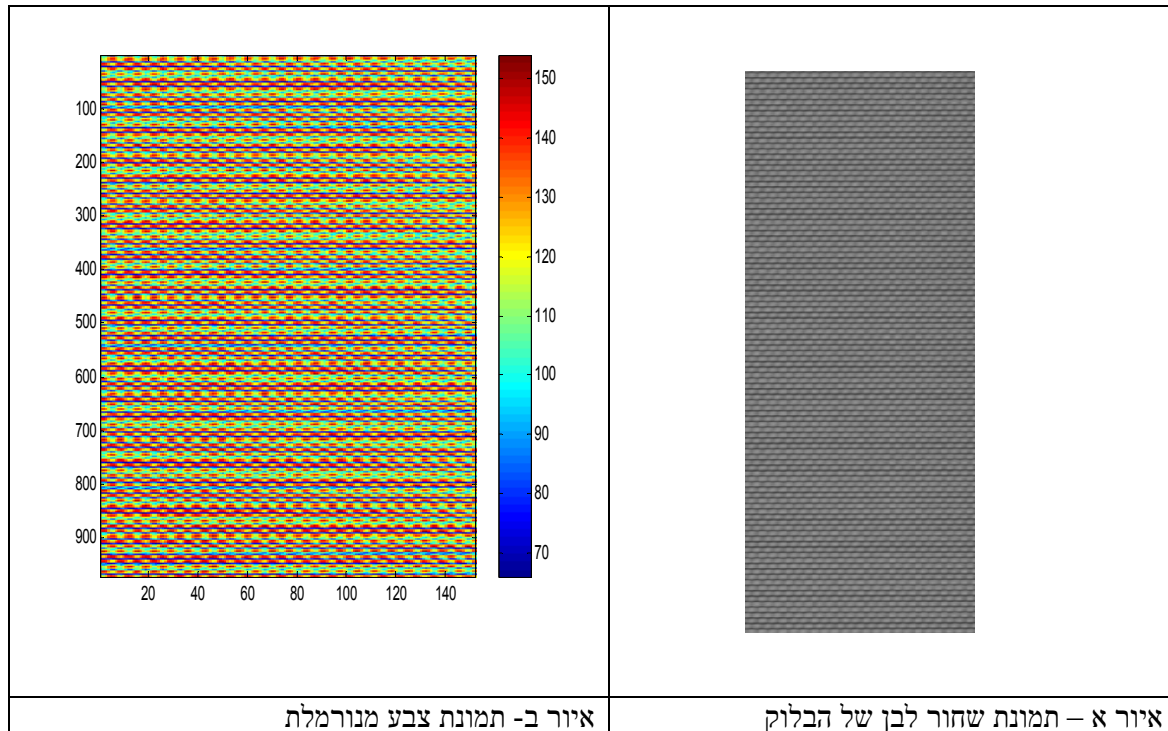
האיור הבא ממחיש את ההצגה המתמטית שלעיל



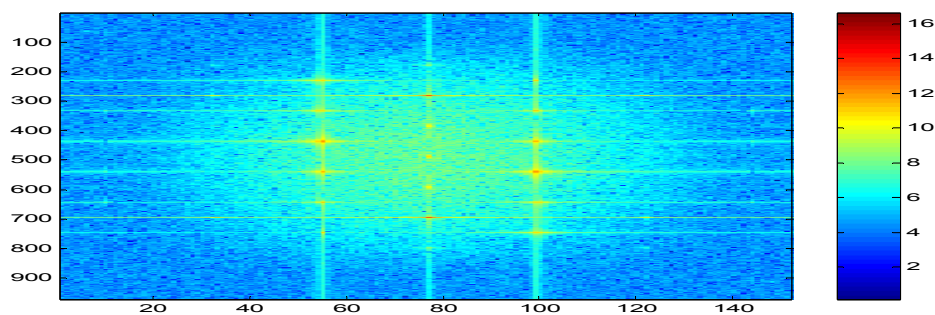
איור 43 תיאור האות כהרכבה של התאים, הפגם והרעש הגאוסיאני. גילוי הפגמים יכול תיאורטית להתבצע כדלקמן: נבצע התמרת FFT על האות. מכיוון שהאות מכיל רכיבים מחזוריים עם אמפליטודה גבוהה נוכל לזהותם ולאפס אותם באות ה-FFT. כך נקבל מצב שאות ה-FFT יכיל עתה את התדרים המייצגים את הרעש והפגם. כעת נבצע התמרה הפוכה למישור האות בציר הזמן ונקבל את הרעש והפגם. בהנחה שניתן להעביר סף (מכיוון שהרעש בעוצמה נמוכה מהפגם) נוכל לקבל את הפגם. האיור הבא מדגים את התהליך שתואר כאן:



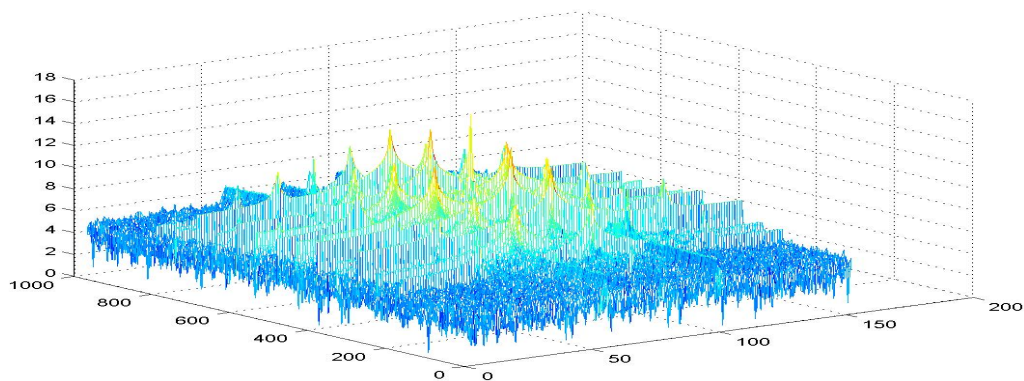
איור 44 תיאור מעבר למישור התדר סילוק ההרמוניות המחזוריות של התא וחזרה למישור האות אם נשליך את האמור לעיל למישור התמונה ולהתמרה FFT הדו-ממדית נסיק שאפשר לבצע את אותו תהליך במישור התמונה. לדוגמא: אם ניקח את הבלוק הבא המכיל בתוכו תאים:



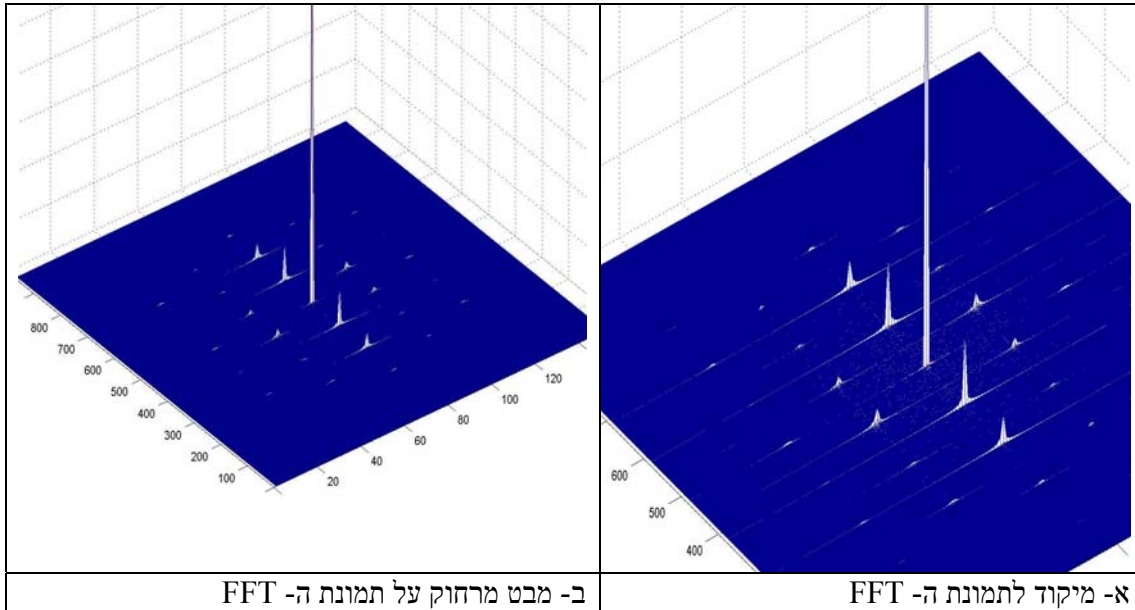
איור 45 תיאור תמונת בלוק המכילה תאים (cells) מחזוריים
אם נבחן את התמרת ה-FFT2 נראה שלכאורה היא מכילה הרמוניות חזקות בתדר של התא ובמחזוריים שלו.



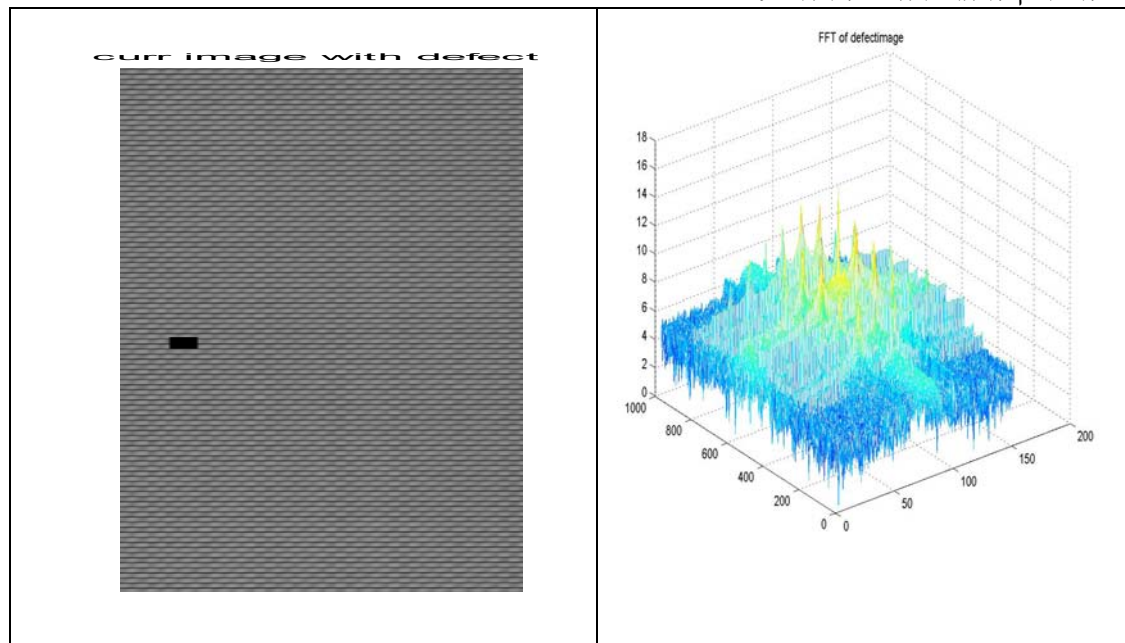
איור 46 תמונת log של התמרת FFT2 של התמונה.
מבט תלת ממדי בהתמרה מחזק את הנראה בתמונה הקודמת.

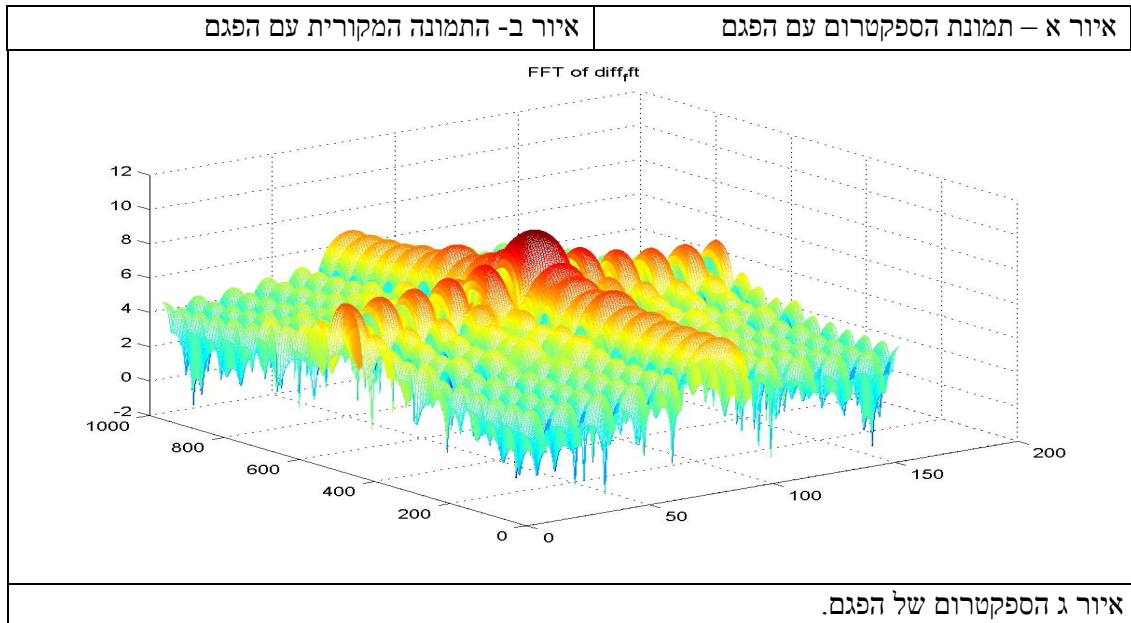


איור 47 תמונת log תלת ממדית של התמרת FFT2 של התמונה.



איור 48 תמונת תלת ממדית של התמרת FFT2 של התמונה
הבעיה העיקרית בשיטה זו שצריך לאפס את התדר של התא וגם את כל ההרמוניות שהם כפולה של תדרי התא. במקרה שבאיורים גודל התא הוא 18.84375 ולכן ייצוג ההרמוניות לא יהיה מוחלט (כלומר הרמוניה מסוימת מתפרסת על יותר מנקודה אחת בתמונת ה-FFT) דבר שמחריף את יכולת ההחלטה הנכונה על אפוס התדרים הנכונים. יותר מכך גם אם גודל התא הוא כפולה שלמה של פיקסל (וניתן לדאוג לכך בפעולה מקדימה ע"י ביצוע rescaling לתמונה) עדיין קשה למצוא מסנן כזה שיסנן את תדרי התא ואת כל ההרמוניות שלו מכיוון שגם הפגם וגם הרעש מופצים לכל התדרים ובאפוס תדרי התא אנו פוגעים גם בתדרי הפגם ולכן התמונה שנקבל לאחר ההתמרה ההפוכה תהיה עם SNR נמוך ונתקשה לגלות את הפגם מתוך הרעש. במקרה של התמונה במצב של color variation (יש שינויי צבע איטיים לאורך הבלוק) הוא יתבטא כאוסף של תדרים שאותם ללא ספק יהיה קשה לסנן. סיבה נוספת היא העיוות בציר ה-X שיש במכונה דבר שתורם ליצירת הרמוניות גם בציר ה-Y והופך את שיטת העבודה הנ"ל ללא מעשית במקרה שלפנינו.
כדי להדגים את האמור לעיל נתבונן בספקטרום של פגם פשוט בתמונת הבלוק ונראה כי אכן הספקטרום שלו מופץ לתחום רחב של תדרים.





איור 49 תיאור התפלגות ספקטרום של הפגם

6.2 ניתוח התמונה המחזורית במרחב מישור התמונה

כאמור מפאת כל הסיבות שתוארו לעיל עבודה במישור התדר איננה מעשית במקרה שלנו הן בגלל אילוץ TPT (מספר ה- cycles/pixel למימוש העבודה בתחום התדר הוא גבוה מידי) והן מבחינת הביצועים האלגוריתמים. בחלק זה נציג את מימוש האלגוריתם Cell2ManyCell במרחב התמונה בהתבסס על הרעיונות מן המקורות [6] [7] [8] [25] עם שינויים והשלמות כנדרש. הרעיון הבסיסי שמוצג במאמרים שלעיל מבוסס על שימוש במידע ממספר רב של תאים כדי ליצור את תא הזהב, כלומר תא ייחוס הנקי מרעש והקרוב ככל האפשר לתא המקורי הנמצא על ה- wafer. הדרך הפשוטה ביותר (בצורה דומה לזו שהוצגה באלגוריתם die2die) היא למצע מספר רב של פיקסלים מתאימים השייכים לאותה נקודה בתא כלומר נבנה את הייחוס ממספר רב ככל האפשר של תאים שכנים ובכך נפחית את הרעש ונגדיל את ה- SNR בהשוואה לאלגוריתם cell2cell שהוצג בתת-סעיף הקודם. הניתוח המתמטי הבא מחזק טענות אלה:

ניתוח זה יעשה רק בממד Y מכיוון שהוא זה שיהיה רלוונטי בהמשך אבל הנכונות בשני הממדים היא טריוויאלית. נגדיר:

- השונות של הרעש בתמונת ההפרשים - $V[\Delta_n] \sim V[X_n - X_{n+C}] \sim 2\sigma^2, V \equiv Var$
 - כמו שהוסבר בפרק של die2die לתמונת הייחוס המורכבת ממוצע של הרבה תאים (M) יש רמת רעש היורדת לאפס ככל ש M גדל. $V[\Delta_n] \sim V[X_n - \bar{X}] \sim \sigma^2$
 - נוסיף את ההגדרות מסעיף 6.1.
 - M – מספר התאים.
 - מספר נקודות הדגימה בתמונה בממד אחד N.
 - האמפליטודה של הפגם D.
 - ה- STD של הרעש σ .
- אם נבנה את הייחוס כממוצע מכל התאים נקבל
- $$\hat{X}_n = \frac{1}{M} \sum_{j=0}^{M-1} X_{n+jC}; \quad M = \frac{N}{C}$$
- $$\Delta_n = X_n - \hat{X}_n =$$

$$\begin{aligned} &= S_n + D_n + \varepsilon_n - \hat{S}_n - \hat{D}_n - \hat{\varepsilon}_n \\ &= D_n - \hat{D}_n + \varepsilon_n - \hat{\varepsilon}_n \end{aligned}$$

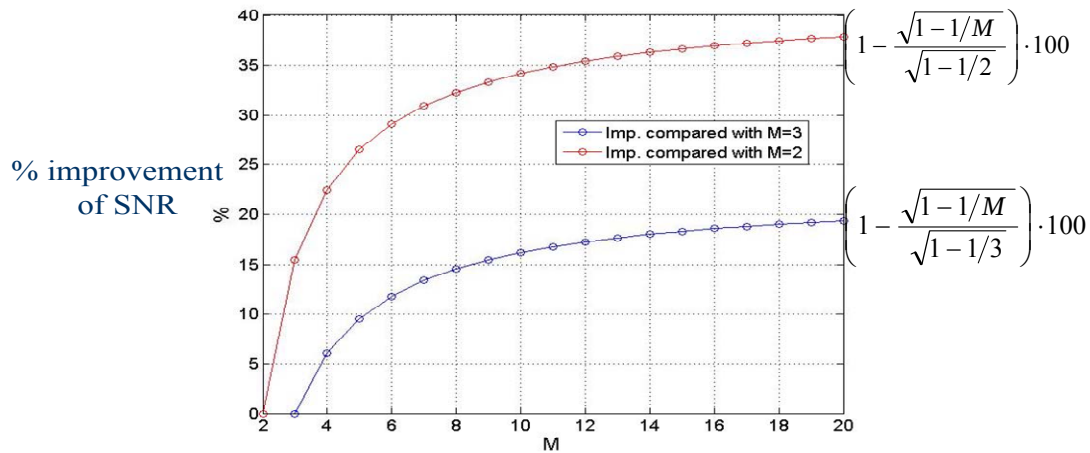
$$\overline{SNR} = \frac{\bar{\Delta}_{n_0}}{\sqrt{V[\Delta_n]}} = \frac{D_{n_0} - \hat{D}_{n_0}}{\sqrt{V[D_n - \hat{D}_n] + V[\varepsilon_n - \hat{\varepsilon}_n]}}$$

⋮

$$\overline{SNR} = \frac{D \left(1 - \frac{1}{M}\right)}{\sqrt{\frac{D}{N} \left(\frac{1}{M} - \frac{1}{N}\right) + \sigma^2 \left(1 - \frac{1}{M}\right)}}$$

$$\overline{SNR} = \frac{D \left(1 - \frac{1}{M}\right)}{\sqrt{\frac{D}{N} \left(\frac{1}{M} - \frac{1}{N}\right) + \sigma^2 \left(1 - \frac{1}{M}\right)}} \xrightarrow{\frac{D}{N} \ll \sigma^2} \frac{D}{\sigma} \sqrt{1 - \frac{1}{M}}; \quad M_{\max} = \frac{N}{C}$$

הניתוח המתמטי מראה שאם נבדוק מהו השיפור ב-SNR נקבל את השיפור כפונקציה למספר התאים שאנו ממצעים. ההשוואה מתייחסת למצב שבו משווים רק שני תאים למצב שבו משווים שלושה תאים.



איור 50 השיפור ב-SNR של מספר רב של השוואות לעומת 2 או 3 השוואות.

ניתן לראות שהחל ממספר תאים שהוא בערך 14 השיפור ב-SNR הוא איטי. נקודה מעניינת זו יכולה להיות נקודת החלטה במימוש האלגוריתם כפי שנראה בהמשך.

הראשון שטיפל בנושא היה Khalaj [25] שהיציע טכניקה לחלץ אבן בנייה בסיסית של התבנית המחזורית שעליה רוצים לגלות פגמים. בשלב הראשון הופעל האלגוריתם ESPRIT שמטרתו למצוא ולהעריך את המחזור של התא. בשלב השני בונים את אבן הבנייה (building block) בעזרת המשוואה הבאה:

Eq. 1:

$$\begin{aligned} B(k, l) = & \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} [(1-r_i)(1-s_j)F(k_i+k, l_j+l) + r_i(1-s_j)F(k_i+k+1, l_j+l) \\ & + (1-r_i)s_jF(k_i+k, l_j+l+1) + r_is_jF(k_i+k+1, l_j+l+1)] \end{aligned}$$

שבה $F(k,l)$ הוא הערך של תמונה $N \times N$ במיקום (k,l) ו- $B(k,l)$ הוא הערך של אבן הבנייה $T_x * T_y$ במיקום (k,l) כאשר T_x ו- T_y הם המחזוריים האופקיים והאנכיים בהתאמה.

$$1 \leq k \leq \text{int}(T_x) + 1, \quad 1 \leq l \leq \text{int}(T_y) + 1$$

$$n_1 = \text{int}(N / T_x), \quad n_2 = \text{int}(N / T_y)$$

$$k_i = \text{int}(T_x * i), \quad l_j = \text{int}(T_y * j)$$

$$r_i = T_x * i - k_i, \quad s_j = T_y * j - l_j$$

ע"י מיצוע מקבלים קירוב טוב של אבן הבנייה. השלב הבא הוא להשוות כל נקודה בתמונה המקורית עם המתאימה לה באבן הבנייה. אם ההפרש גדול מסף (הנקבע בתהליך מקדים) אז הנקודה עלולה להיות פגם אפשרי. בגלל האפקט של הכימות בקצוות התמונה כל נקודה מושווית עם שמונת השכנים המתאימים שלה באבן הבנייה. מבין כל תשעת הפרשי הערכים המוחלטים הקטן ביותר נשמר כדי לתת מדד על ההסתברות של הנקודה להיות פגם. ניתן לתאר את האמור לעיל ע"י המשוואה הבאה:

Eq. 2:

$$\min_{-1 \leq i, j \leq 1} |F(m,n) - B(k+i, l+j)|$$

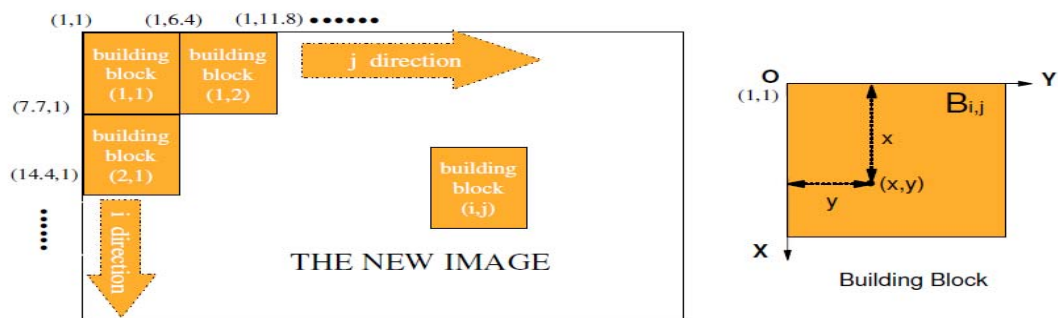
$$\text{where} \quad k = m - \text{int}(m/T_x) * T_x, \quad l = n - \text{int}(n/T_y) * T_y$$

במאמר [8] המחברים הציעו שיטה לשפר את בניית אבן הבנייה של התא. מכיוון שהמחזור של התא הוא לא גודל שלם של פיקסל ברוב המקרים נובע מכך שהגודל של אבן הבנייה (building block) שהוגדר איננו הגודל האמיתי של התבנית המחזורית.

לכן הוגדרה אבן בנייה מדומה (simulated building block) שהגודל שלה הוא בדיוק $T_x * T_y$. הערכים של $B(k,l)$ כאשר

$$1 \leq k \leq \text{int}(T_x) + 1 \text{ and } 1 \leq l \leq \text{int}(T_y) + 1$$

מתקבלים על ידי אינטרפולציה לינארית. בעזרת משוואה 1 מחשבים את הערכים של הנקודות השלמות באבן הבנייה. הבנייה של התמונה הנקייה מפגמים המתחשבת בגודל הפיקסל הלא שלם מתבצעת כדלקמן: נניח שיש לנו מחזור $T_x = 6.7; T_y = 5.4$ אז האיור הבא מראה את דרך בנייתה של תמונה נקייה מפגמים F' .



איור 51 המבנה של תמונה נקייה מפגמים F' עם $T_x = 6.7; T_y = 5.4$ מניחים כאן שקואורדינטות של ה-BB הם יחסית ל $(1,1)$ והערך של כל נקודה (x,y) בתוך (i,j) בתוך ה-BB מצוינת על ידי $B_{i,j}(x,y)$ המתאימה (k,l) ב- F' עבור $B_{i,j}(x,y)$ כלשהו ניתנת לחישוב בעזרת המשוואה הבאה:

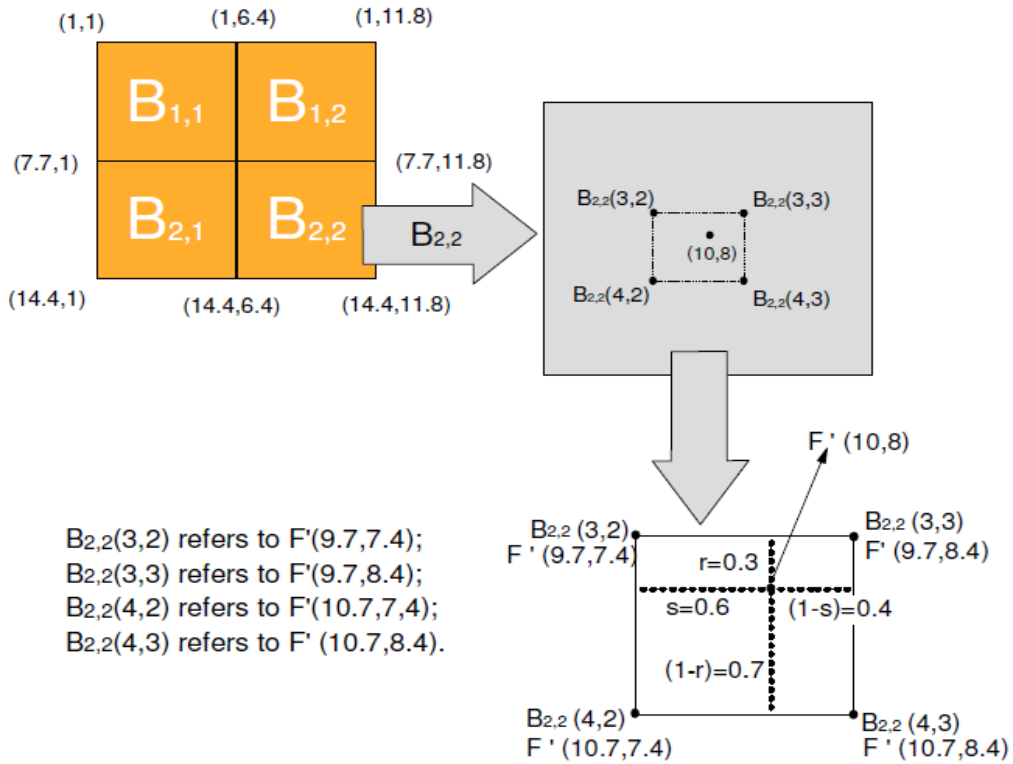
Eq. 3:

$$k = (i-1) * T_x + x, \quad l = (j-1) * T_y + y$$

למשל ניקח כדוגמא חישוב של ערך הפיקסל ב- $(10,8)$ בתמונה הנקייה מפגמים. $(10,8)$ ממוקם ב- $B_{2,2}$. הוא מרוחק אופקית 0.6 פיקסלים מ- $B_{2,2}(i,2)(i=3,4)$ ו- 0.4 פיקסלים מ- $B_{2,2}(i,3)(i=3,4)$ ו- 0.3 פיקסלים אנכית מ- $B_{2,2}(3,j)(j=2,3)$ ו- 0.7 פיקסלים מ-

$B_{2,2}(4, j)(j = 2, 3)$. הערך של רמת האפור שלו תיקבע ע"י

$B_{2,2}(3, 2), B_{2,2}(3, 3), B_{2,2}(4, 2), B_{2,2}(4, 3)$ על ידי אינטרפולציה לינארית כמודגם באיור הבא:



איור 52 בניית תמונה נקייה מפגמים מ- $M \times N$ BB עבור תמונה עם $T_x = 6.7; T_y = 5.4$ לסיכום ערך רמת האפור של הפיקסל בתמונה F' מחושב לפי הנוסחה הבאה:

Eq. 4:

$$F'(k, l) = (1-r)(1-s)B_{i_1, j_1}(x_1, y_1) + r(1-s)B_{i_2, j_1}(x_2, y_1) + s(1-r)B_{i_1, j_2}(x_1, y_2) + rsB_{i_2, j_2}(x_2, y_2)$$

כאשר $F'(k, l)$ הוא הערך של התמונה החדשה בגודל $N \times M$ במיקום (k, l) . כאשר כל ארבעת האינטרפולציות של $F'(k, l)$ יושבות באותו BB נקבל

$$i_1 = i_2 = \text{int}(k / T_x) + 1, \quad j_1 = j_2 = \text{int}(l / T_y) + 1$$

$$x_1 = \text{int}(k - \text{int}(k / T_x) * T_x), \quad x_2 = x_1 + 1$$

$$y_1 = \text{int}(l - \text{int}(l / T_y) * T_y), \quad y_2 = y_1 + 1$$

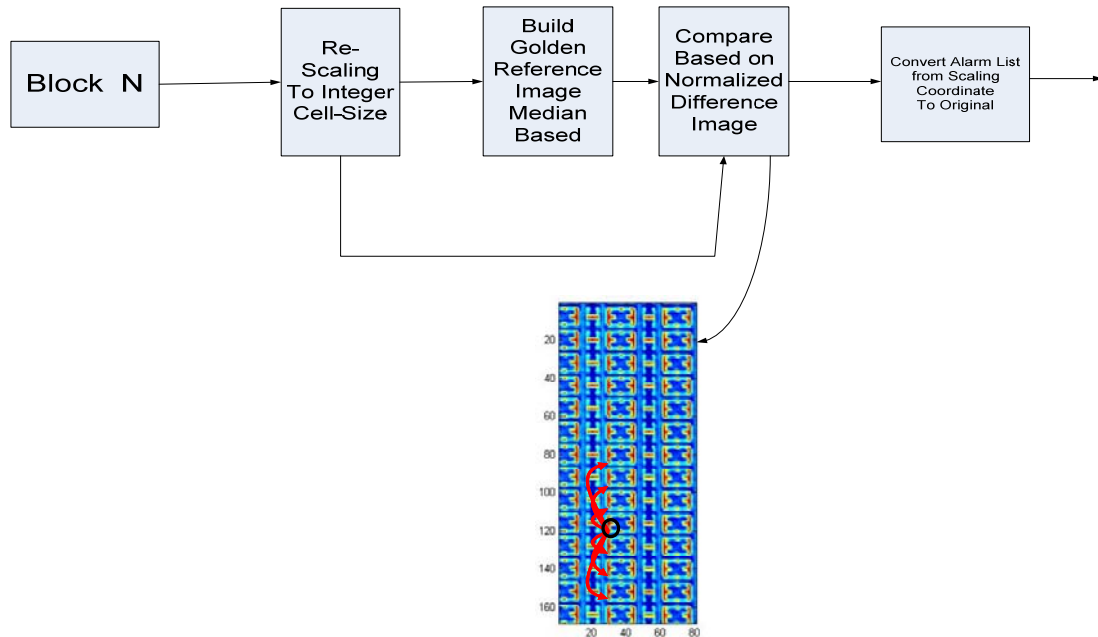
$$r = \frac{k - k_1}{k_2 - k_1} = k - \text{int}(k / T_x) * T_x - x_1, \quad s = \frac{l - l_1}{l_2 - l_1} = l - \text{int}(l / T_y) * T_y - y_1$$

k_1, k_2, l_1 and l_2 are calculated from x_1, x_2, y_1 , and y_2 using Eq. 3.

6.3 התאמות למערכות WI וניתוח התוצאות

כדי לממש את האלגוריתם שהוצג בסעיף הקודם נדרשות מספר התאמות ושינויים כדי להתאימו לריצה במערכת WI מהסיבות הבאות:

- ביצוע אינטרפולציה לינארית אינו מתאים למבנה ה-spot במערכת WI. כמו שהוסבר בסעיף לתיאור אלגוריתם die2die ו-cell2cell אנו זקוקים לביצוע של אינטרפולציה קובית (Cubic) כפי שהוסבר בסעיפים אלה.
 - מכיוון שישנם עיוותים רבים (הדבר נפוץ באופטיקה הסורקת לאורך ציר X) לאורך ציר ה-X נסתפק בבניית Golden Cell או אבן הבנייה רק לאורך ציר Y.
 - במידה ונמצע את כל הפיקסלים המתאימים לתא לכל אורך ההבלוק נהיה חשופים להשפעותיה של תופעת ה-color variation. כזכור לאורך הבלוק (שהוא בין 1000 ל-3000 שורות) ישנם שינויי צבע איטיים שישפיעו על מיצוע ארוך.
 - כדי להתגבר על השפעות ה-color variation נעדיף למצע מספר בודד של תאים בין 5 ל-9 וכך אפקט ה-color variation יהיה זניח.
 - מצד שני מכיוון כשאנו מצמצמים את מספר ה-cells שאותם ממצעים עלול להיווצר מצב שבו פגמים ובמיוחד כאלה בעלי שטח גדול יחסית ישפיעו על תוצאת המיצוע ויפגמו באיכות תא הזהב (golden cell) שניצור. כדי לתקוף את הבעיה אבצע שינוי כך שבמקום למצע אבחר את הפיקסל המתאים ב-Golden Cell להיות החציון (median) מבין אלה שאיתם בונים אותו. שימוש בפעולת median יגרום לכך שבוודאות לא נבחר פיקסלים כאלה שיש בהם פגם מכיוון שהפגם ישנה את ערך הבהירות לגבוה או נמוך מהממוצע. החיסרון בפעולת החציון לעומת מיצוע שקשה לממש ביעילות חציון על מספר רב של פיקסלים אולם במקרה שלנו מכיוון שאין צורך לממש חציון על יותר מ-9 פיקסלים אנכיים ניתן לבצע זאת ביעילות חישוב על 8 פיקסלים במקביל תוך שימוש בפעולות max ו-min הווקטוריות של ה-DSP. בנוסף לכך גם היחס אות לרעש ישתפר בצורה ניכרת ונבטל כמעט לחלוטין את השפעת הפגם על בניית הייחוס.
 - באלגוריתם המוצע בסעיף הקודם מתבצעת אינטרפולציה לכל פיקסל בנפרד ולאחר מכן הוא מושווה מול סף. במקרה שלנו הציון בתמונת הפגמים (alarms) מחושב לפי הפרש מנורמל וביצוע של אינטרפולציה על כל פיקסל בצורה נפרדת לא יאפשר מימוש יעיל של לולאת ה-DSP. לכן הרעיון שלי להתמודד עם הבעיה יהיה בכך שראשית נעשה תיקון לכל התמונה על ידי כך שנמתח אותה (rescaling) כך שכל פיקסל יישב ברשת של פיקסלים שלמים, כלומר נתקן את התאים מחלק שלם ותת-שלם (sub-pixel) לחלק שלם בלבד. פעולה זו תוכל להתבצע ללא פגיעה משמעותית בתקציב ה-cycle/pixel מכיוון שהיא בעצם דורשת החלפת מקדמי הגרעין (kernel) עבור כל שורה וזכור הלולאה היעילה (tight loop) מתבצעת על השורה. לאחר שמיקמנו ותיקנו את כל התמונה והאם שבתוכה כך שישבו על פיקסלים שלמים בלבד נוכל לבצע ביעילות את בניית תמונת הייחוס המכילה עבור כל פיקסל את ייחוס הזהב (golden reference) שהוא החציון מתשעה פיקסלים. לאחר בניית תמונת ייחוס הזהב (golden reference) נוכל לבצע פונקצית השוואה בסיסית שתוארה בפרקים הקודמים בין תמונת ייחוס הזהב לתמונת המקור ולקבל תמונה ורשימת פגמים (alarms).
 - השלב האחרון ידרוש המרת הקואורדינטות של הפגמים חזרה למישור של התמונה המקורית לפני שעברה scaling.
- לסיכום האלגוריתם הממומש ומותאם למערכות WI יהיה כדלקמן:
- בהינתן בלוק ראשית נבצע עליו re-scaling כדי ליישב אותו על רשת של תאים עם גודל פיקסל שלם.
 - בניית בלוק ייחוס הזהב ע"י ביצוע חציון ל 7,5 או 9 עמודות לכל פיקסל עם פיקסלים מתאים שכנים מלמעלה ומלמטה בהתאמה.
 - ביצוע פונקצית השוואה בסיסית (compare) בין תמונת ה-golden לתמונת המקור שעברה scaling ויצירת תמונת הפרשים מנורמלת (עבור אלו שעבור את הסף אחרת הערך בפיקסל יהיה אפס) ורשימת alarms.
 - תרגום רשימת ה-alarms מקואורדינטות בתמונה שעברה scaling בחזרה לקואורדינטות מתמונת המקור.
- האיור הבא מתאר את זרימת האלגוריתם השוואת תא מחזורי:



איור 53 תיאור מלבני של אלגוריתם השוואת תא מחזורי

פעולת ה- rescaling תבצע כדלקמן: נניח שגודל תא הוא C והוא איננו מספר שלם של פיקסלים. נגדיר $\alpha = C/[C]$ כאשר $[C]$ הוא השלם המעוגל הקרוב ביותר. אם $Y(n)$ הוא התמונה כאשר $n = 1, \dots, N$

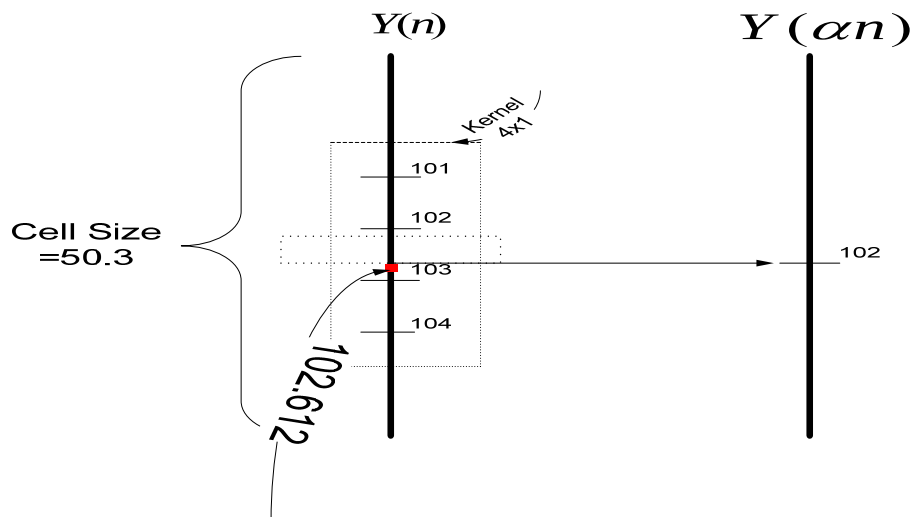
נקבל שכדי לבנות את התמונה המתוחה לחלק שלם של התא נרשום $\bar{Y}(n) = Y(\alpha n)$ כאשר $Y(\alpha n)$ מתקבל ע"י אינטרפולציה B-cubic.

האיור הבא מביא דוגמא והסבר לבחירת מקדם התיקון עבור שורה כלשהי:

$$\text{Cell Size} = 50.3$$

$$[C] = 50$$

$$\alpha = C/[C] = 50.3/50 = 1.006$$



איור 54 תיאור דוגמא לבחירת מקדם התיקון עבור שורה כלשהי

נניח שאנו רוצים לקבל את שורה 102 אז מכיוון ש- $\frac{102 * 50.3}{50} = 102.612$. מכיוון שאנו עובדים

כאמור עם תיקון ברזולוציה של $1/128$ נקבל $0.612 * 128 = 78.336$ כלומר התיקון התת-פיקסלי יהיה

78/128. בחינת ביצועי האלגוריתמים תעשה ע"פ אותם פרמטרים שבהם בחנו את אלגוריתם die2die ו-cell2cell בסעיף הקודם.

כאמור שלבי ביצוע האלגוריתם כפי שתוארו לעיל הם:

- ביצוע rescaling של התמונה.
- בניית ייחוס הזהב (golden reference) ע"י ביצוע של median באורך 9,7,5 על התמונה שעברה rescaling.
- ביצוע פונקצית השוואה לפי העיקרון שהודגם [בנספח 4 \(מימוש פונקצית השוואה\)](#) בין התמונה לאחר rescaling ל-golden reference.
- חישוב/תיקון של מיקומי ה-alarms לפי התמונה המקורית.

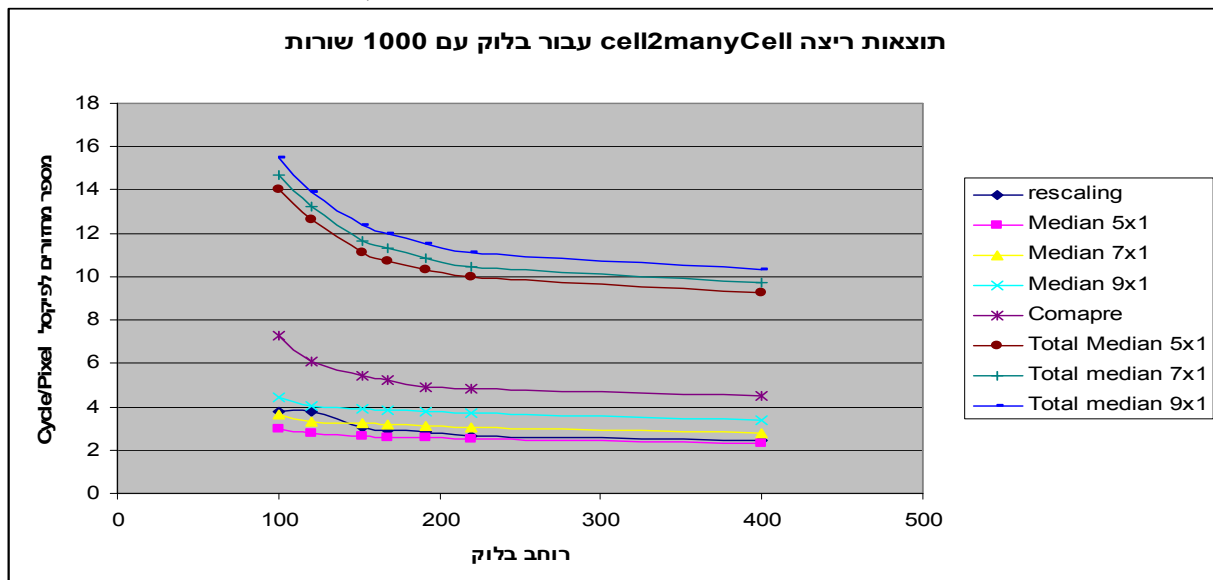
התוצאה של הפונקציה האחרונה היא רשימת alarms שעבור כל alarm יש מיקום בבלוק וציון המצוין את ההסתברות שלו להיות פגם וכן תמונת alarms המציינת את הערכים הללו לכל פיקסל.

יש לציין שמימוש יעיל של פעולת ה- median התגבר ע"י הצורך לבצע 9,7,5 העתקות של בלוקים (הזזות פוינטרים) ע"י העתקות דו ממדיות של המידע שעליו עובדים בלבד לתוך הזיכרון הפנימי בעזרת ה-EDMA.

תוצאות סיכום ריצת כל אבני הבניין של האלגוריתם היא כדלקמן עבור בלוק עם 1000 שורות:

רוחב בלוק	rescaling	Median 5x1	Median 7x1	Median 9x1	Compare	Total Median 5x1	Total median 7x1	Total median 9x1
100	3.78	3	3.63	4.41	7.27	14.05	14.68	15.46
120	3.78	2.75	3.3275	4.0425	6.1	12.63	13.2075	13.9225
152	3.038	2.66	3.2186	3.9102	5.4	11.098	11.6566	12.3482
168	2.94	2.6	3.146	3.822	5.2	10.74	11.286	11.962
192	2.87	2.55	3.0855	3.7485	4.9	10.32	10.8555	11.5185
220	2.66	2.5	3.025	3.675	4.8	9.96	10.485	11.135
400	2.45	2.3	2.783	3.381	4.5	9.25	9.733	10.331

טבלה 9 טבלת זמני ריצה כוללת של אלגוריתם השוואת תא מחזורי עבור בלוק עם 1000 שורות



איור 55 תיאור זמני ריצה כוללת של אלגוריתם השוואת תא מחזורי בבלוק עם 1000 שורות
תוצאות הריצה עבור בלוקים עם 500 ו-2000 שורות מסוכמות בטבלאות הבאות:

רוחב בלוק	Rescaling	Median 5x1	Median 7x1	Median 9x1	Compare	Total Median 5x1	Total median 7x1	Total median 9x1
100	3.92	3	3.63	4.41	7.3	14.22	14.85	15.63
120	3.36	2.8	3.388	4.116	6.2	12.36	12.948	13.676

152	3.08	2.7	3.267	3.969	5.5	11.28	11.847	12.549
168	3.01	2.6	3.146	3.822	5.4	11.01	11.556	12.232
192	2.94	2.6	3.146	3.822	5.2	10.74	11.286	11.962
220	2.8	2.5	3.025	3.675	5	10.3	10.825	11.475
400	2.66	2.35	2.8435	3.4545	4.6	9.61	10.1035	10.7145

טבלה 10 טבלת זמני ריצה כוללת של אלגוריתם השוואת תא מחזורי עבור בלוק עם 500 שורות

רוחב בלוק	rescaling	Median 5x1	Median 7x1	Median 9x1	Compare	Total Median 5x1	Total median 7x1	Total median 9x1
100	3.78	2.9	3.509	4.41	7.12	13.8	14.409	15.31
120	3.752	2.75	3.3275	4.0425	6.1	12.602	13.1795	13.8945
152	3.038	2.6	3.146	3.9102	5.2	10.838	11.384	12.1482
168	2.94	2.58	3.1218	3.822	5.1	10.62	11.1618	11.862
192	2.8	2.55	3.0855	3.7485	4.7	10.05	10.5855	11.2485
220	2.632	2.5	3.025	3.675	4.7	9.832	10.357	11.007
400	2.38	2.3	2.783	3.381	4.5	9.18	9.663	10.261

טבלה 11 טבלת זמני ריצה כוללת של אלגוריתם השוואת תא מחזורי עבור בלוק עם 2000 שורות

ניתוח זמני הריצה של אלגוריתם השוואת תא מחזורי מראה תוצאות דומות לאלו של die2die ו-cell2cell. גם כאן הגדלת רוחב הבלוק בציר X תורמת לכך שהאלגוריתם רץ מהר יותר מאותם סיבות שהוזכרו לגבי die2die. האלגוריתם הורץ על אותם wafers שהוא רץ ב-cell2cell ורואים שהשיפור ב-SNR עבור ריצה עם חציון 5x1 היה 8.9% ו-9.3% יחסית לריצה ב-cell2cell ויחס פגמי השקר השתפר כמעט בהתאמה וירד ל-12% ו-13%.

עבור ריצה עם חציון 7x1 השיפור ב-SNR היה 11% ו-11.7% יחסית לריצה ב-cell2cell ויחס פגמי השקר השתפר כמעט בהתאמה וירד ל-9% ו-11%.

עבור ריצה עם חציון 9x1 השיפור ב-SNR היה 14% ו-14.8% יחסית לריצה ב-cell2cell ויחס פגמי השקר השתפר כמעט בהתאמה וירד ל-8% ו-7%.

השוואה זו לאלגוריתם cell2cell מראה על שיפור ניכר ב-SNR וביחס גילוי פגמי שווא התואם את הניתוח התיאורטי שהוצג בתחילת פרק 6.

בנוסף לכך הגידול בתקציב ה-cycles לא היה דרמטי יחסית לאלגוריתם cell2cell והוא נע בסדר גודל של 10-15 אחוז וזאת מכיוון שלמרות שתהליך ה-rescaling יקר במעט מתהליך התיקון (correction) ולמרות שהתווספה אבן הבנייה של ה-golden reference לא הזדקקנו יותר לפעולת ה-double sided וכך התקדמו ה-cycles בהתאמה.

יש לציין שפעולת המרת הקואורדינטות של ה-alarms איננה נלקחת בספירת המחזורים מכיוון שהיא זניחה (עובדים יחסית על מספר קטן של alarms).

לסיכום, אלגוריתם זה המבוסס על המאמרים שהוצגו משתמש ברעיונות ששם לניצול המחזוריות אבל מטפל בכל הבעיות המעשיות של מימוש יעיל ההכרחי למערכות זמן אמת בצורה יצירתית על ידי ביצוע rescaling תחילה לרשת המכילה תאים בגודל שלם.

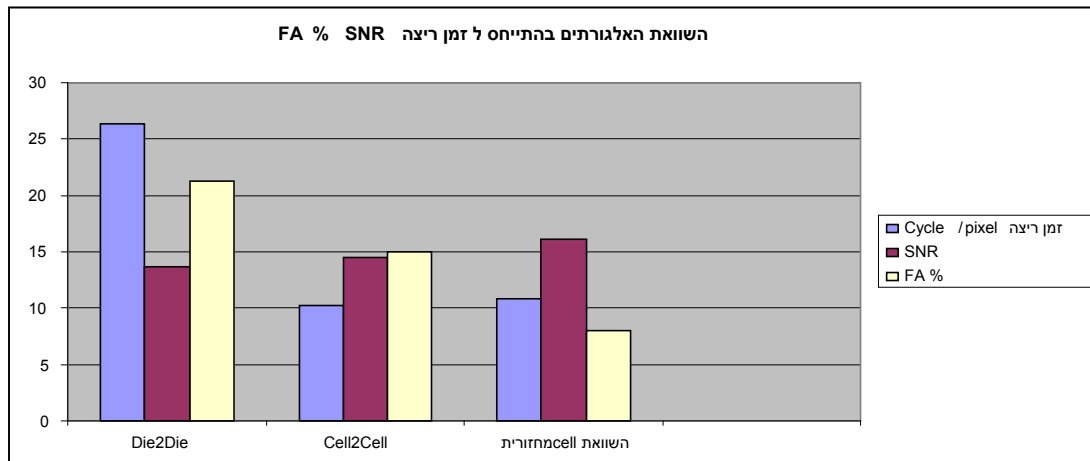
בנוסף האלגוריתם שהצגתי מטפל בהשפעה של פגם גדול על תהליך בניית הייחוס בעזרת שימוש בפעולת החציון בניגוד לפעולת מיצוע ובכך מנטרל את השפעת הפגם בעוד שהאלגוריתם המוצג במאמרים לא מטפל בכך. אומנם ביצוע פעולת החציון דורש התפשרות מסוימת בשיפור ה-SNR לעומת מיצוע של כל הבלוק אולם מיצוע כל הבלוק (או חציון על כולו) היה מכניס לתוך הבנייה את השפעת ה-color variation ממנה רוצים להימנע. בנוסף ההפסד בשיפור בין לקיחת 9 פיקסלים למספר אין סופי הוא 4% והפשרה שביצענו מתקבלת.

7 סיכום ומבט לעתיד

בעבודה זו הדגמתי מימוש של אלגוריתם למציאת פגמים בפרוסות סיליקון על מעבדי DSP מסוג TMS320DM6437. כאשר קוראים מאמר המתאר אלגוריתם כלשהו הדבר הינו מעניין אך לא פחות מעניין לבוא ולנסות לתרגם זאת לפסים מעשיים. הפסים המעשיים תלויים רבות במגבלות הפיזיקאליות של המכונה ובפלטפורמת החישוב עליה היא מתבססת. הפרמטרים הפיזיקאליים גורמים לכך שאלגוריתמים תיאורטיים כמעט ואינם מועילים במימוש ישיר (ראה die2die בסיסי) ונדרשים שיפורים התאמות ושינויים על מנת שנוכל לגלות פגמים בעזרת אלגוריתמים אלה ובנוסף צריך לגרום לכך שמספר המחזורים לפיקסל שהאלגוריתם לוקח לא יעלה על התקציב שקיים. כאשר אני מסתכל על העבודה בפרספקטיבה לאחר הסקתי מספר מסקנות כדלהלן:

1. האלגוריתם die2die מציג בעצם רעיון חדש של מעבר מהשוואת die2die להשוואת בתוך die ומתמודד די בהצלחה עם בעיית ה- color variation במקומות בהם חייבים לסרוק ב-die2die. בנוסף לכך לאלגוריתם זה במימושו הכללי יש יכולת טובה להתמודדות עם תהליכים עתידיים שבו גודל כלל העיצוב יקטן מכיוון שאז התמונה המתקבלת תבחין פחות בפרטים (אורך הגל נשאר קבוע) ולכן יש לו יכולות השוואה של פיקסל מול סביבתו הקרובה באזורים שאינם ברורים מספיק. השאלה העיקרית שנשאלת היא מהם השכנים המתאימים ביותר להשוואה עבור כל פיקסל, כלומר בעיית מציאה ואיתור של השכנים המתאימים היא בעצם ליבו של האלגוריתם.

2. ביצוע ומימוש האלגוריתם **השוואת תא מחזורי** השיג ביצועים מרשימים תוך שיפור של ה-SNR והימנעות מהשפעת הפגם על הגילוי בדרך יצירתית ובנוסף הוא חסכוני במספר מחזורים לפיקסל והוא יכול לשמש כבסיס למערכות inspection בתחומים שונים ולא דווקא עבור בדיקת wafers. הגרף הבא מתאר השוואה בין האלגוריתם השונים בהתייחס לזמן הריצה ואיכותה:



איור 56 השוואה בין האלגוריתמים השונים

הטבלה הבאה מתארת את היתרונות והחסרונות של כל אחד מהאלגוריתמים שתוארו:

אלגוריתם השוואת תא מחזורי	Cell2Cell	Die2Die	תכונה
נמוך	נמוך	גבוה	מחזורים לפיקסל
Array	Array	גם logic וגם Array	אזורים נסרקים
איננו מושפע מבעיות רגיסטרציה ו- CV SNR גבוה.	איננו מושפע כמעט מבעיות רגיסטרציה ו- CV	מתאים גם לתמונות עתידיות עם איכות תמונה נמוכה שמקורם בגלל מגבלות האופטיקה.	יתרונות
בעייתי במכונות עם jitter גבוהה בציר ה Y	SNR יחסית נמוך להשוואות מקומיות	רגישות לתהליך Image Registration	חסרונות

מכיוון שאז תהליך ה-Rescaling מסובך. בנוסף לכך פעולת החציון מתאימה לרעשים אקראיים אבל לא תמיד הרעשים הם כאלה. במיקרים אלו אנו עלולים לקבל ירידה ברגישות הגילוי ואף עלייה בגילוי שקר.	מתאים רק ל- Array	מספר Cycle/Pixel גבוה. כדי להתמודד עם בעיית יציבות צריך לשפרו ולמדוד את רמת הרעש הלוקלית. בעיית אם אין מספיק סטטיסטיקה לכל GL.	
---	-------------------	--	--

3. ביצוע עבודה זו תרם לי להבנה של נושאים נוספים מחוץ לאלגוריתמים ובעיקר נושאים הקשורים להנדסת מערכת והקשר בין הנדסת המערכת לפתרונות האלגוריתמים הדרושים.

כשמתבוננים קדימה במבט לעתיד ברור שהמעבר לטכנולוגיות מזעור עתידיות של 22 ננומטר ומטה יגרום לכך שכוח המחשוב הדרוש יגדל בגורם 4 לפחות וידרוש מערכות מחשוב מתקדמות יותר במחיר דומה. הנקודה הנוכחית בזמן מהווה צומת דרכים לתעשיות רבות בתחום תהליכי ייצור הדורשות כוח מחשוב הולך וגדל למכונות בקרת הייצור. אין ספק שהמעבר לריבוי ליבות בכל אחת מטכנולוגיות המחשוב אמור לעזור מצד אחד אבל מצד שני מציב אתגרים לא קטנים בפני מתכנני המערכות שהעיקרי שבניהם איך לנצל את כוח העיבוד להשגת יעילות מרבית ותפוקה מקסימלית. אין ספק שהתכנות של מערכות מרובי ליבות יהווה אתגר ולדעתי חייבת להימצא דרך לנצל את כוח העיבוד בלי להידרש למומחיות רבה בהבנת הטכנולוגיה של פלטפורמת החישוב שאם לא כן מלאכת התכנות וזמן ההגעה לשוק (time to market) עלולים להתעכב רבות.

לסיכום נהניתי מאוד בעבודה זו בה שילבתי את הידע שלי כמהנדס אלקטרוניקה וכן העשרתי את הידע גם בנושא הנדסת מערכת והקשר שלה להנדסת תוכנה ואופן מימוש ופיתוח האלגוריתמים המקביליים. אני רוצה להודות לד"ר עמי מרובקה על העזרה התמיכה והליווי לכל אורך הדרך.

נספחים

נספח 1 - תהליך ייצור הטרנזיסטור

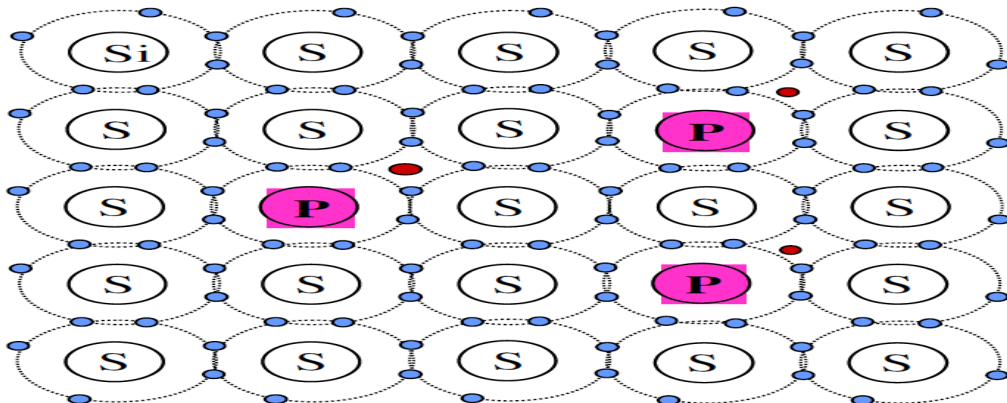
העבודה העיקרית בתוך השבב נעשית ע"י יחידה הנקראת טרנזיסטור שהוא בעצם היחידה הבסיסית והחשובה ביותר של כל רכיב מוליך למחצה. שבב מודרני מכיל בין מאות-אלפים למיליוני טרנזיסטורים שהם ה"רכיבים הפעילים" שלו (בניגוד למוליכים שהם רכיבים פאסיביים). ניתן לתאר את השימוש העיקרי של הטרנזיסטור כמתג אלקטרוני המחבר ומנתק מעגל חשמלי אחד כתלות במתח הקיים במעגל חשמלי אחר. מטרנזיסטור אחד או יותר מרכיבים את שערי ה- NOT, AND, OR שהם אבני הבניין של כל רכיב אלקטרוני ספרתי. למעשה השבב קיים בשביל הטרנזיסטורים שעליו ויתר חלקיו הם מערכת סבוכה של חוטים המחברים בין הטרנזיסטורים.

מבנה הטרנזיסטור

- ראשית נבחן מהו המוליך למחצה. ניתן לסווג חומרים בהתאם להתנהגותם הסגולית לשלוש קבוצות:
- חומרים מתכתיים הינם חומרים שהתנגדותם הסגולית נמוכה.
 - חומרים מבודדים הינם חומרים שהתנגדותם הסגולית גבוהה.
 - חומרים מוליכים למחצה (מ"מ) הינם חומרים שהתנגדותם הסגולית נמוכה משל מבודדים וגבוהה משל מתכות. המבנה האטומי של מוליך למחצה הינו כזה שבקליפה החיצונית של האטום ישנם ארבעה אלקטרונים בקליפה החיצונית. לדוגמא מבנהו האטומי של מוליך למחצה מסוג Si שהוא בעל מספר אטומי 14 (זאת אומרת בעל 14 אלקטרונים ו- 14 פרוטונים) ישנם 4 אלקטרונים בקליפה החיצונית.

המבנה הגבישי של מוליך למחצה

אטומי המוליך למחצה מקושרים בניהם בעזרת קשר קוולנטי, קשר קוולנטי הינו קשר שבו האטומים המשתתפים בקשר תורמים כל אחד אלקטרונים אשר נעים במסלול משותף בין האטומים המקושרים. בעזרת תהליך כימי מזהמים מוליך טהור Si עם כמות קטנה של אטומים מסוג ארסן או זרחן (P) שיש להם עודף של אלקטרון אחד בקליפה ומקבלים **חומר מוליך למחצה מסוג N** המאפשר תנועת אלקטרונים ושהמבנה הגבישי שלו מתואר באיור הבא:



איור 57 מבנה אטומי של מוליך למחצה מסוג N. נקודות אדומות מציינות אלקטרון עודף. בדומה ניתן לזהם בחומר בעל חוסר של אלקטרון בקליפה ולקבל **חומר מסוג P** המאופיין בחוסר של אלקטרון ומאפשר תנועה של חוסר של אלקטרונים או חורים.

צומת PN וטכנולוגית CMOS

צומת PN או דיודה הינו צירוף של מוליך למחצה מסוג N עם מוליך למחצה מסוג P. כתוצאה מהצירוף של מוליך למחצה מסוג N עם מוליך למחצה מסוג P, יזרמו אלקטרונים חופשיים מהחומר מסוג N לחומר מסוג P וימלאו את החורים שבחומר מסוג P.

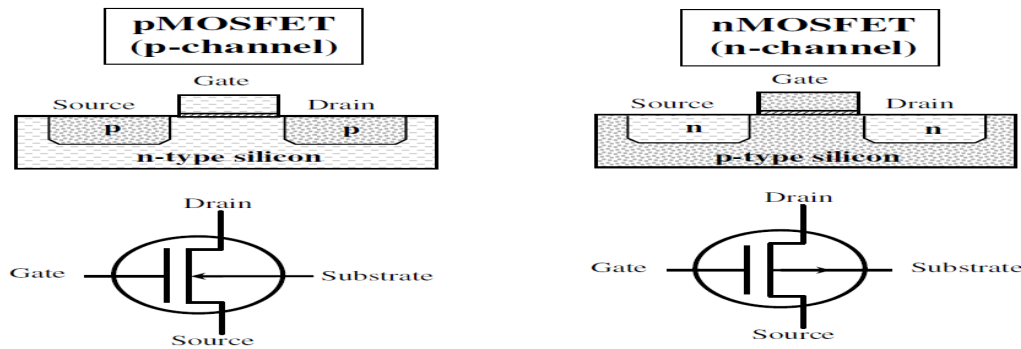
כל אטום בחומר מסוג N אשר האלקטרון החופשי שלו מבחינת ההקשר עבר לחומר מסוג P יהיה בעל מטען חיובי היות שמספר האלקטרונים שלו קטן ב-1 ממספר הפרוטונים, אולם הוא יהיה מאוזן מבחינת הקשר. זאת אומרת, 4 אלקטרונים המקושרים עם כל אחד מהשכנים. כמו כן, כל אטום בחומר מסוג P אשר קיבל אלקטרון מהחומר מסוג N יהיה בעל מטען שלילי היות שמספר האלקטרונים שלו גדל ב-1 ממספר הפרוטונים, אולם הוא יהיה מאוזן מבחינת הקשר.



איור 58 צימת PN

החשיבות של הבנת עיקרון פעולתם של צומת PN הוא בכך שכל סוגי הטרנזיסטורים מבוססים על הצמדת חומר PNP או NPN כדי לקבל התקן שבמעגלים המשולבים מתפקד כמעין מפסק אלקטרוני המבוקר ע"י מתח כניסה. ישנם סוגים שונים של טרנזיסטורים אבל העיקרון של כולם מבוסס על ווריאציות של צמתי PN.

הטרנזיסטור הבא J-FET הוא הבסיס לכל המעגלים. עקרון פעולתו הוא כדלקמן:
ההתקן הבסיסי של טכנולוגית המעגלים המשולבים הוא טרנזיסטור J-FET בעוד שהטכנולוגיה נקראת MOSFET או CMOS אם גם ה-NFET וגם ה-PFET נמצאים באותו שבב.



איור 59 תרשים חשמלי ומבנה CMOS.

כאמור ברוב המעגלים הבסיסיים NFET ו-PFET מיוצרים בזוגות. לדוגמא המעגל הבא מהווה בסיס למספר רב של מעגלים לוגיים. המעגל הבא משמש כמהפך (שער NOT). כאשר בכניסה יש מתח חיובי הטרנזיסטור התחתון (nMOSFET) מוליך ומהווה קצר ולכן במוצא יש מתח שלילי ($-V_{ss}$). במקרה זה הטרנזיסטור העליון בנתק ולכן איננו מוליך. כאשר אות הכניסה שלילי הטרנזיסטור התחתון בנתק והעליון (pMOSFET) מוליך ולכן ליציאה עובר מתח ($+V_{dd}$). יש להעיר שמוליך פירושו קצר. הפעולה הממומשת היא שער NOT.

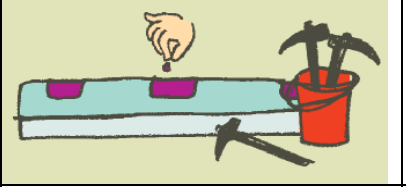
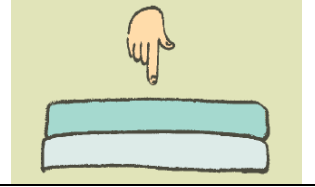
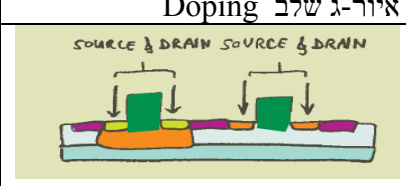
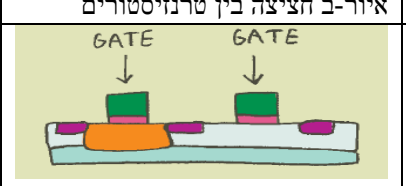
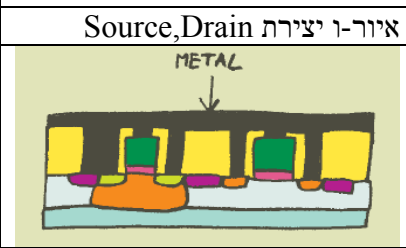
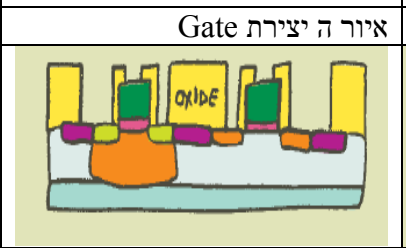
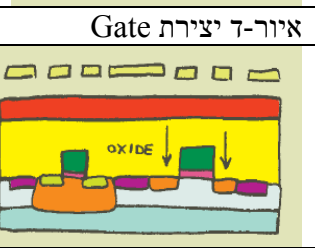
צילום ננו-מטרי של חלק משבב מבוסס CMOS	מבנה שער NOT בסיסי בטכנולוגית CMOS

איור 60 א-ב מבנה שער NOT בסיסי בטכנולוגית CMOS וצילום ננומטרי שלו

איך מייצרים טרנזיסטור

בחלק זה נתאר כאמור איך מייצרים טרנזיסטור מסוג MOSFET שמבנהו הפיזיקאלי והחשמלי תואר בפרק הקודם. בעצם ניתן להגיד שתהליך יצירת הטרנזיסטור הוא בעצם ואריאציה של התהליכים הכימיים שתוארו בפרק 1. בשלב הראשון מצפים את פרוסת הסיליקון בשכבת סיליקון באיכות גבוהה. שלב זה מכונה epitaxy ושכבה זו תהווה את המצע או הבסיס של הטרנזיסטור (איור 61-א).

בשלב השני שותלים בתוך הסיליקון חומר מבודד שיחצוץ בין הטרנזיסטורים ויבטיח שהאחד לא יפריע למשנהו (איור 61-ב). עושים זאת ע"י חפירת בורות ננומטרים ושיקוע של חומר מתאים בתוכם. בשלב הבא עלינו להחליט אם אנו רוצים לייצר טרנזיסטור מסוג N-MOS או מסוג P-MOS. ההבדל בניהם הוא בסוג המטען שיזרום: חיובי P או שלילי N. אם כי האמת היא שבטכנולוגיה של היום המכונה CMOS מייצרים את הטרנזיסטורים בזוגות – אחד מסוג P ואחד מסוג N (ראה איור 61-ג). מפציצים את הסיליקון באלקטרונים (בסוג P יהיו אלה יונים חיוביים). שלב זה נקרא doping, אבל לפני שעושים זאת יש לכסות את כל חלקי השבב שאיננו רוצים להפציץ בשכבת מגן מבודדת שהיא בעצם ווריאציה של התהליכים שתוארו בפרקים הקודמים (איור 61-ג האזור הכתום הוא זה שיופצץ ביונים). כעת מסלקים את שכבת המגן ועוברים לשלב הבא והוא יצירת השער (gate) של הטרנזיסטור. כאן לא חופרים בור, אלא בונים גבעה – ע"י ציפוי הפרוסה בשכבת אוקסיד ומעליה פולי-סיליקון ואחר כך מסלקים את החומר מכל השבב מאזורי השער שעליהם מגנים (איור 61-ד, איור 61-ה). בשלב הבא בונים את ה"מקור" וה"בור" (source, drain) של הטרנזיסטור. שוב פעולה זו מתבצעת ע"י חפירת בורות ננו-מטריים, מילויים בסיליקון באיכות גבוהה והפצצת הסיליקון, הפעם ביונים חיוביים כי המטען על המקור והבור הפוך מהמטען של המצע שמתחתיהם (איור 61-ו). סיימנו את בנייתו של טרנזיסטור ה-N כעת מגיע תור טרנזיסטור ה-P. התהליך בטרנזיסטור P דומה פרט לכך שהמטענים מוחלפים. כעת נותר לאפשר את חיבור הטרנזיסטור לשאר העולם. ראשית מבודדים אותם ע"י שכבה של אוקסיד (איור 61-ז) אחר כך חופרים באוקסיד בורות במקומות שרוצים לחבר את החוטים (למקור, לבור, ולשער) (איור 61-ח). לבסוף ממלאים את הבורות במתכת (שוב וואריאציה על הנושאים הקודמים). ובזה מסתיים תהליך יצירת הטרנזיסטור. לבסוף ישנו ליטוש אחרון (איור 61-י).

		
איור-ג שלב Doping	איור-ב חציצה בין טרנזיסטורים	איור א שלב Epitaxy
		
איור-ו יצירת Source, Drain	איור ה יצירת Gate	איור-ד יצירת Gate
		
איור-ט חיבור טר' לעולם החיצון	איור-ח חיבור טר' לעולם החיצון	איור-ז חיבור טר' לעולם החיצון
		

איור 61 א-י תיאור גרפי של שלבי ייצור הטרנזיסטור.

נספח 2 - מערכות BF מבוססות הארת לייזר לעומת הארת מנורה

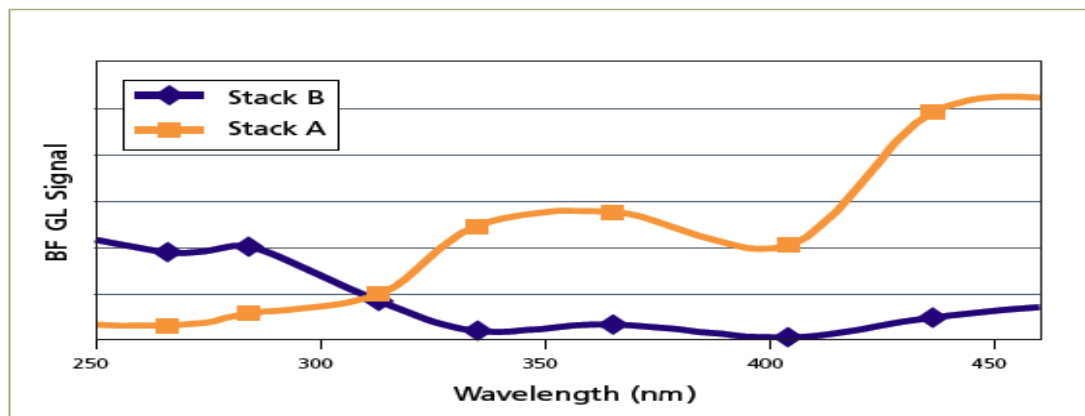
הדיון בנספח זה מבוסס על המאמרים [1],[2],[4],[5],[24],[26],[28],[32] שבמקורות. מכיוון שעבודה זו מתרכזת באלגוריתמים מקביליים למערכות BF-inspection המבוססות על אורכי גל של DUV נציג כאן שתי גישות להארת ה-wafer במכונות BF. גישות אלו מייצגות שתי גישות המרכזיות בעולם ה-wafer inspection ולא בכדי הם מפורסמות ע"י שתי יצרניות הציוד המובילות בעולם שהם מתחרות (AMAT ו-KLA). סקירה זו מבוססת על המקורות שלעיל.

הבעיות העיקריות במערכות BF נובעות מכך שכדי לגלות פגמים קטנים, גודל הפיקסל קטן. בנוסף לכך משתדלים לעבוד עם אופטיקה המבוססת על מקורות אור באורכי גל נמוכים מכיוון שלפי חוקי האופטיקה הגאומטרית, ככל שנקטין את אורך הגל נוכל לגלות פגמים קטנים יותר. הבעיה העיקרית בפיקסלים קטנים היא שנדרש דיוק רב ביישור כל זוג תמונות המשוות. תהליך זה נקרא image registration. ככל שהפיקסלים קטנים יותר תהליך זה הופך לקשה ומסובך הן מהבחינה הפיזיקאלית והאלגוריתמית. בנוסף לכך מכיוון שהאור נאסף מעל למשטח קיימות הפרעות קשות בקבלת התמונה הנובעות מכך שישנם החזרים של קרניים משכבות תחתונות לשכבה הנסרקת ובנוסף העובי של הפרוסות אינו זהה, דבר הגורם להבדלי צבע בין die ל-die שכן. תופעה זו מכונה color variation כלומר שינוי צבע הרקע לאורך ה-dies ב-wafer. כל הגורמים שתוארו לעיל גורמים לירידה ביחס אות לרעש וגורמים בכך לפגיעה בגילוי הפגמים.

השוואת מערכות BF מבוססות הארת מנורה לעומת הארת לייזר

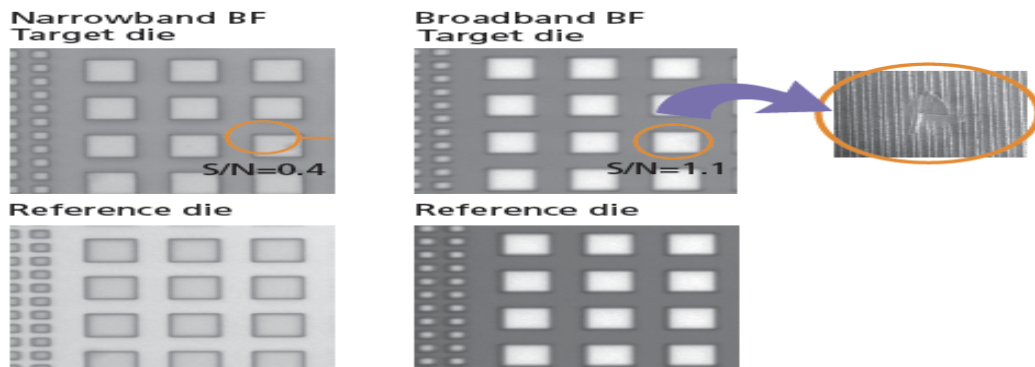
הטענה העיקרית המצדדת במערכות BF מבוססות הארת מנורה היא שמערכת שמוארת באור מנורה מכילה תחום רחב של אורכי גל הניתנים לכיוון. הטענה שככל שיצרני הרכיבים ממשיכים להתפתח ולהשתמש בטכנולוגיות מתקדמות הכוללות חומרים חדשים, הם נתקלים בסוגים שונים של פגמים שגורמים גם לרעשים המכבידים ומעכבים את תהליך הגילוי. להארת BF רחבת פס שניתנת לכיוון יש כמה יתרונות על הגישה של הארה באורך גל אחד. מודלים מחקריים וגם ניסויי מעבדה הראו שסוגי פגמים ושכבות שונות דורשות אורכי גל שונים עבור גילוי פגמים אמין. מקור אור המשתרע על אורכי גל מ-DUV עד האור הנראה המכוון לתחום אורכי הגל המרביים כדי ליצור ניהודיות מקסימלית (maximum contrast) בהתייחס לתכונות האופטיות של שכבה נתונה או פגם מסוים. השימוש בספקטרום רחב יכול להפחית את הפרעות ה-color variation הנובעות כאמור משכבות תחתונות ובגלל עובי שונה של השכבות.

יתרון נוסף כאמור למערכות אלו הוא שיפור יחס אות לרעש של המערכת (S/N ratio). מכיוון שיחס אות לרעש הוא פרמטר יסודי ביכולת הגילוי הבסיסית של מערכות BF הוא תלוי הרבה בניגודיות בין הפגם לסביבתו. במערכות BF הניהודיות הזו תלויה בתכונות האופטיות והתכונות האופטיות הללו הם פונקציה של אורך הגל של הקרן הפוגעת. מערכת BF המבוססת על ספקטרום מלא מספקת יכולת גבוהה ביותר לגילוי פגמים מכיוון שהקרן הפוגעת ניתנת לכיוון כדי ליעל ולהגביר את הניהודיות בין סוגי הפגמים המעניינים וסביבתם.



איור 62 תיאור התלות של האות בתלות האורך הגל

האיור שלעיל מדגיש את התלות באורכי גל כאשר Stack-A מתאר SION בעובי 27nm, שכבת פוטו-רזיסט שהגילוי בה מוצלח באזור 450 ננומטר לעומת Stack-B המתארת שכבת SION בעובי 45nm שהגילוי בה מוצלח באזור 230 ננומטר. לכן כיוון אורך הגל הוא קריטי בלכידה וגילוי של הפגם כאשר שינויים בתהליך הינן הכרחיים בחדשנות פיתוח הרכיב. המצדדים במערכות BF רחבות פס טוענים שהטענה המסורתית שאורך גל קצר יותר מאפשר גילוי של פגמים קטנים יותר פשוט איננו מדויק בהרבה מיקרים ובמקום זה תחום ספקטראלי רחב של אורכי גל מאפשר להפיק את המרב עבור פגם ושכבה מסוימת והוא מפתח חשוב לגילוי תחום רחב של סוגי פגמים שמתרחשים ב-FAB. באופן בסיסי אלגוריתמים של מערכות WI משווים die אחד לאחר כדי לגלות נוכחותו של פגם. בכמה מיקרים dies שכנים עלולים להיות עם הבדלים קלים בעובי השכבה הדיאלקטית שנראה כהבדל ברמות האפור למערכות BF המבוססות על אורך גל אחד. מכיוון שעובי שכבה שונה מעט של השכבות או מקדם שבירה שונה לא משפיעה על ביצועי הרכיב אז פגמים מסוג color variation מסוג זה נחשבים לפגמים שלא מעניינים את היצרנים (nuisance defects). כלומר תופעת ה-color variation קשה יותר במערכות המבוססות על אורך גל אחד (לייזר) לעומת אור מנורה רחב פס ולכן במערכות כאלה יידרש אלגוריתם גילוי הפגמים לטפל ולנטרל תופעה זו. האיור הבא מדגים את תופעת ה-color variation במערכות צרות פס לעומת רחבו פס.



איור 63 תופעת color variation במערכות BF מבוססות מקור אור רחב פס לעומת צר פס.
החלק השמאלי של האיור מתאר תמונת dies שכנים עבור תמונות המתקבלות ממקור צר פס כאשר יש פגם. מצד שני רמות ההארה של מנורת BF במערכות inspection היא נמוכה מטבעם כתוצאה מכך רגישות המכשיר לחומרים שרגישים לנזקי האור או לחוסר הרפלקטיביות יכולה להיזדרז בצורה משמעותית. אומנם האור רחב הפס מקזז את השפעת ה-color variation וגורם ליחס אות לרעש גדול יותר אבל הרזולוציה של התמונה והפרטים בה מסווגים וכך נפגעת רגישות הגילוי ובנוסף לכך שעוצמת ההארה במנורה יחסית חלשה בהשוואה ללייזר מונו-כרומטי שבו ניתן להגיע להספקים גבוהים. מכונת ה-BF inspection המבוססת על לייזר DUV מכילה בנוסף לערוץ ה-BF גם ערוץ GF שהוא ערוץ 3D image ומאפשרת גילוי ופיקוח בתהליכי ייצור מתקדמים עם TPT של כמה WPH.

יתרונות וחסרונות בין הארת מנורה להארת לייזר

כאמור הארת מנורה היא השיטה המסורתית והיתרונות שלה שהאות נקי יותר מרעשים ובעיקר מכיל פחות רעשי color variation. מצד שני העוצמה המופקת ממנורה אינה גבוהה דבר שגורם לאות עם עוצמה נמוכה ולכן בעל איכות ירודה במקרים רבים. מצד שני הארת לייזר גורמת להגברת תופעת color variation המסבכת את הגילוי אבל מצד שני ההארה יכולה להיות בהספקים גבוהים ולכן האות מכיל פרטים ברזולוציה גבוהה למרות שסף הרעש שלו גבוה יותר. שימוש ב-NA גדול האפשרי בקלות רבה יותר בגלאי PMT מפחית את הבעיה אך לא מעלימה כליל. בנוסף קיומם של שני ערוצי איסוף BF ו-3D בו זמניים מאפשרת לגלות פגמים מסוגים שונים באותה סריקה מכיוון שערוץ ה-3D משלים את ערוץ ה-BF. החיסרון בכך שקצב העיבוד כפול (שני ערוצי איסוף) לעומת הארת מנורה ולכן צריך מערכת עיבוד תמונה מהירה וחזקה כדי לממש את האלגוריתמים המסתגלים לרעש הדרושים כדי להפיק אות נקי מרעשים. הפרקים הבאים יטפלו בתכנון מערכת עיבוד תמונה כזאת ודרכי מימוש אלגוריתמים המערכת עיבוד תמונה זו.

נספח 3- מבוא וסקירה היסטורית לאבולוציית מערכות עיבוד תמונה WI

תכנון מערכת עיבוד תמונה למכונות WI מציב בפני המתכננים אתגרים טכנולוגיים הולכים וגדלים ככל שהיצרנים ממשיכים לעקוב אחר חוק מור וממשיכים בהקטנת המימד הקריטי ובהגדלת הרזולוציה. עד שנות ה-2000 משימת תכנון מערכות עיבוד תמונה האלה הייתה מורכבת אבל לכאורה די סבירה. גידול במספר הטרנזיסטורים ב-die תרם ליציאה של מעבדים חזקים יותר וגם יחס ה-cost/performance הלך והשתפר. גודל הפיקסלים היה בסדר גודל של מיקרונים (בעיקר במערכות DF) ולכן קצב עיבוד מידע הדרוש היה בסדר גודל של 200Mbyte/sec. בנוסף פיתוח האופטיקה של מכונות ה-WI עקב אחר אורכי הגל של האופטיקה הליתוגרפית (כמו שהוסבר בפרקים הקודמים) ולכן התמונות שהתקבלו היו ברובם עם יחס signal/noise גדול דבר שהקל על מימוש אלגוריתמים לגילוי פגמים. המבנה של מערכת עיבוד תמונה כזה כולל מערכת לרכישת התמונה ולאחריה שלבים אלגוריתמים די פשוטים לפי התבנית הבאה:

- רכישת התמונה.
- ביצוע החלקת התמונה או סינון מקדים אחר.
- השוואת התמונה המתקבלת מ-die אחד ל-die הקודם.

מכיוון שהתמונות היו עם signal/noise גבוהה השוואת התמונות הייתה מבוססת בעיקרה על תמונת הפרשים בין שני ה-dies ולאחריה מעבר של סף סטטי על תמונת הפרשים. תמונת הפרשים המתקבלת ציינה את ההסתברות של פיקסל זה להיות פגם. הסף על תמונת הפרשים נקבע בתהליך כיוול פשוט למדי ע"י איסוף היסטוגרמות המתארות את התפלגות הפרשים בין שני dies סמוכים והתבצע לפני תחילת הסריקה עבור פרוסות סיליקון מתהליך מסוים. לכאורה קל מאוד היה להמשיך ולתמוך בתעשיית המוליכים למחצה ולייצר מערכות עיבוד תמונה למכונות WI שגם יהיו במחיר סביר וגם שיהיו מסוגלות לממש את האלגוריתמים לצורך זיהוי הפגמים.

אורך הגל של תהליכי הליתוגרפיה המשיך לקטון ואיתו גודל הפיקסלים דבר שגרם לכך שמורכבות האלגוריתמים עלתה מכיוון שכאשר גודל הפיקסל קטן הרעידות של המכונה גורמות לכך שהתמונות המתקבלות משני dies סמוכים אינם מיושרות אחת לשנייה ולכן לפני ביצוע תמונת הפרשים צריך היה למדוד את התזוזה ולתקן את התמונות כך שיהיו מיושרות אחת לשנייה. תהליך זה נקרא registration או יישור ההיסט בין התמונות והוא מבוסס על מדידות קורלציה על חלונות נבחרים בין שני ה-dies. פעולה זו צורכת משאבי מימוש לא קטנים אבל עדיין ברזולוציה הנ"ל עדיין הייתה בת מימוש במשאבי המערכת (פעולה זו תוסבר בפרקים הבאים).

אולם שני מהלכים גרמו לכך שמשימת בניית מערכות עיבוד התמונה הפסיקו להיות טריוויאליות: המהלך הראשון נבע מכך שהממד הקריטי (design rule) של תהליכי הייצור המשיך לקטון למרות שאורך הגל של אופטיקת הליתוגרפיה נעצר באזור 191nm. כדי להמשיך ולרדת בממד הקריטי למרות שאורך הגל נשאר קבוע יצרני הרכיבים משתמשים בשתי שיטות עיקריות להגדלת הרזולוציה שהראשונה עושה שימוש במים בתהליך הליתוגרפיה להורדת אורך הגל והשנייה משתמשת בשיטת double patterning. שבה מגיעים למימד קריטי קטן יותר מזה האפשרי באורך גל נתון ע"י חלוקה לשתי הדפסות נפרדות עם תזוזה מתאימה ביניהם לקבלת צפיפות כפולה. כמובן שבבניית מכונות inspection לא ניתן להשתמש בשיטות שתוארו לעיל. התוצאה של כל האמור לעיל הוא שככל שהצפיפות גדלה התמונות המתקבלות באופטיקת ה-inspection הנוכחיות (ראה פרק 2) הינם בעלי signal/noise נמוך יותר ולכן מלאכת הגילוי נהיית קשה ומסובכת יותר.

הסיבות לכך שכעת משתמשים בסגמנטציה של התמונה לאזורי רעש שונים שיכולים להיות סטטיים ודינמיים כאחד. בנוסף תהליך registration הופך להיות קריטי ומסובך מכיוון שגודל הפיקסל קטן ולכן הרעידות של המכונה מזיזות תמונה אחת יחסית לשנייה בצורה ניכרת נוסף לכך שמכיוון שהתמונות רועשות תהליך הקורלציה הופך לקשה יותר ויותר. בנוסף לכך השוואה בין זוג dies סמוכים כבר איננה מספיקה ולכן משתמשים בהשוואת כל שלושה dies סמוכים או יותר. מכיוון שהתמונות רועשות יותר (ראה דוגמא מרכזית לכך בפרק 2 על בעיית ה-color variation) אז קביעת הספים בכל שלב באלגוריתמים איננה יכולה להתבסס רק על תהליך כיוול מקדים אלא על שילוב בינו לבין תהליך עדכון ספים אדפטיבי דינמי המבוסס על מדידת רעש אדפטיבית ועידון הספים בהתאם לרמת הרעש.

המהלך השני שהתרחש בתעשיית המוליכים למחצה שתהליך הגידול בתדר העבודה (GHz) של המעבדים מפסיק להתקדם. תדר העבודה של המעבדים הגיע לרווית גידול ונמצא באזור של 1-3 GHz אך איננו מתוכנן לגדול מעבר לכך. במקום זאת היצרנים עברו למכונות מבוססות ריבוי ליבות שיושבות על die אחד.

כלומר המגמה זהה והיא הפסקת הגדלת תדר העבודה של המעבדים והגדלת מספרם על שבב אחד. דבר זה יוצר בעיה בתכנון מערכות מכיוון שהמערכת פחות ניתנת להרחבה. כאשר עוברים למערכות מרובות ליבות המידע מתפצל ליותר ויותר פרטים והחלוקות המרובות של המידע בין הליבות השונות גורמות לסיבוכיות רבה יותר של האלגוריתמים. כאמור מערכת BF צריכה לעמוד בקצב עיבוד של 25Gbyte/second כאשר האלגוריתמים הדרושים לצורך מציאת הפגמים הופכים למורכבים ומסובכים. לאור כך יש חשיבות רבה ליעילות האלגוריתם ולדרך מימושו מכיוון שגם אם הוא יעיל במציאת פגמים יתכן בהחלט שאיננו ישים לאור זמן הריצה הארוך שלו.

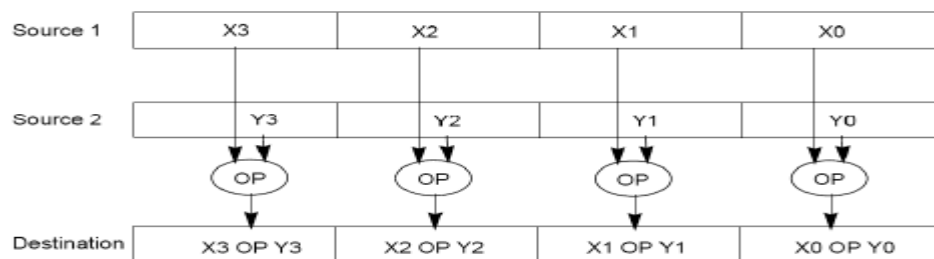
ארכיטקטורת Intel Multi-Core

למרות שלכאורה מוגדרים מעבדי Intel כמעבדים כלליים (General Purpose CPU) הם כוללים יחידות חישוב חזקות המאפשרות ביצוע פעולות אופייניות לעיבוד תמונה בצורה יעילה. ארכיטקטורת Intel הנוכחית המתקדמת כוללת 4 ליבות חישוב בטופולוגיה של UMA לעומתה הדור החדש הנקרא Nehalem וכולל 4 ליבות חישוב עם טופולוגיית NUMA. בנוסף מערכות אלה כוללות זיכרון L1-cache ו-L2-cache עבור כל ליבה וזיכרון L3-Cache המשותף לכל הליבות. הניתוח כאן יתבצע לגבי טכנולוגיית UMA או בשמה המסחרי Intel Pentium Dual-Core. שבה יש 12MB של זיכרון L2-Cache משותף לכל הליבות וזיכרון L1-cache בגודל 32Kb עבור נתונים ו-32Kb עבור התכנית.

הרעיון הבסיסי המאפשר להאיץ פעולות עיבוד תמונה מתבסס על העובדה שבכל פקודה ניתן לעבוד על יותר מפיקסל אחד בו זמנית כי אין תלות בין הפעולה שצריך לבצע על פיקסל אחד לזה שצריך לבצע על שכן. במעבדי Intel קיימות 2 יחידות המאפשרות עבודה במקביל על מספר פיקסלים (עבודה וקטורית):

- יחידת MMX שהיסטורית הייתה קיימת החל ממעבדי 386 ומאפשרת עבודה על 8 פיקסלים בו זמנית.

- יחידת SSE שהתפתחה מאוחר יותר ומאפשרת עבודה על 16 פיקסלים בו זמנית. עיקרון הפעולה של פקודות SSE/MMX מודגם באיור הבא:



איור 64 תיאור גרפי של פקודת SIMD.

הרעיון הוא שישנה פקודת אסמבלר בודדת המבצעת את אותה פעולה על מספר אלמנטים (Single Instruction Multiple Data=SIMD). כאמור קיימות שתי יחידות והם MMX ו-SSE המאפשרות הפעלות פקודות וקטוריות על 8 או 16 בתים.

ישנם 8 אוגרים עבור יחידת SSE והם XMM0-XMM7 ו-8 אוגרים ביחידת MMX ובם MM0-MM7. הרעיון הבסיסי בכתיבת קוד יעיל לפונקציות עיבוד תמונה הוא להימנע ככל האפשר מפקודות הסתעפות ולנצל בצורה מקסימלית את היחידות SSE, MMX.

בנוסף למעבד קיימת חומרה מתוחכמת המנסה לחזות פקודות עתידיות וכן מפענחת פקודות שאין קשר בניהם מכיוון שאותם היא רוצה לבצע אם יש משאבים פנויים. תהליך זה נקרא ביצוע פקודות שלא לפי הסדר (out of order execution). מכיוון שהזיכרון החיצוני בנוי מרכיבי DDR2 גישה אליו הינה יקרה וכוללת סדר גודל של מאות מחזורי CPU. לכן משתמשים במספר רמות של זיכרון מסוג cache כמו שתואר לעיל. אם האלגוריתם עובד על בלוקים קטנים יחסית אז תוצרי השלבים עשויים להישאר ב-

cache וכך לשפר מאוד את זמן הריצה אולם במקרים שבהם הבלוקים שאותם מעבדים גדולים מסדר הגודל של ה-cache נקבל ירידה בביצועים בגלל cache misses. כדי לתת מענה חלקי לבעיה זו המעבד מנסה לחזות את הכתובות הבאות שאליהם התוכנית תיגש ולבצע הבאה מוקדמת שלהם ל-cache. בנוסף לכך קיימות קבוצות פקודות לעזרת המתכנת המאפשרות בצורה יזומה להביא בהבאה מוקדמת שורות של מידע ל-cache. תהליך זה נקרא pre-fetching.

קיימים יצרנים המספקים blade server עם מעבדי Intel. כדי להעריך עד כמה טכנולוגיה זו מתאימה למכונת WI מהסוג שתואר לעיל נניח שאנו רוצים לעבד 20GByte/sec ושהאלגוריתם כולל מספר שלבים כך שסכומם דורש 100cycle/pix בממוצע עבור כל פיקסל שנכנס למערכת (במימוש היעיל ביותר האפשרי). נניח שתדר העבודה של כל ליבה הוא 2.8GHz נקבל בחישוב פשוט שכל ליבה

מסוגלת לעבד קצב של $\frac{2800Mhz}{100cyc} \approx 28 \frac{Mbyte}{Sec}$. בהתחשב בקצב המידע הנכנס למערכת נקבל שמספר

הליבות הדרוש הוא $\frac{20Gbyte/sec}{28Mbyte/sec} \approx 715cores$. יצרני blade server מספקים מערכות עם 8

ליבות על כרטיס (board) וחשוב גם מראה שנזדקק $\frac{715cores}{8cores} \approx 90boards$ ובהנחה שיש

כלובים (cages) עם 14 כרטיסים נצטרך ס"ד של 7 כלובים (cages).

אחת הבעיות העיקריות בשימוש בטכנולוגיית Intel בדרך שתוארה לעיל היא ההספקים הגבוהים שמערכת המתוארת לעיל תצרך. דבר זה יחייב קירור מים של מערכת עיבוד התמונה (אלמנט שמייקר את המערכת בצורה ניכרת). בנוסף עלינו להיות מסוגלים לרכוש את התמונה ולהזין (feeding) את מערכת המחשוב האדירה שתוארה במידע המצולם מפרוסות הסיליקון בזמן אמת (שהרי אם לא אין לנו צורך במערכת עיבוד התמונה). לצורך כך מקובל להשתמש בכרטיסים מיוחדים הנקראים frame grabber שמקבלים את המידע, כלומר את הפיקסלים בפורמט כדוגמת camera-link או דומה לו ואמורים לשלוח אותו על גבי פס טורי מהיר (כדוגמת PCI-Express) לכל אחד ליבות השונות. פיזור המידע לליבות השונות יעשה ע"י כך שכל ליבה תקבל את החלק אותו היא צריכה לעבד וכך מושגת מקביליות ראשונה כבר ברמת תכנון המערכת. החיסרון הגדול של מערכות מבוססות טכנולוגיות Intel הוא המחיר או בעצם יחס העלות לביצועים שלהם. עלות מערכת מסוג זה הכוללת 7 cages בנוסף למערכת קירור מים וכרטיסי frame grabber מתאימים היא בסדר גודל של 300K\$-400K\$. אין ספק שלמרות יכולות המחשוב האדירות המיצוב הכלכלי שלה די בעייתי ביחס למחיר המכונה ועלול לסתום את הגולל על בחירה בטכנולוגיה זו. יש לציין שרבים מהפעולות שטכנולוגיית Intel מציעה כמו עבודה על נקודה צפה אינם בשימוש כאן והדבר משמש לה לרועץ.

מבחינת האפשרות להשיג שיפור ביצועים בפונקציות עיבוד התמונה העובדות על כל פיקסל טכנולוגיות SSE לגרסאותיה השונות (SSE2, SSE3, SSE4) מעניקות שיפור ביצועים בסדר גודל של 10-20. הבעיה העיקרית אם כן במערכות כדוגמת זו שתוארה בפרק זה היא שיחס ה-cost/performance שלה אינו תואם לדרישות שתוארו לעיל.

ארכיטקטורת מערכת עיבוד תמונה מבוססת IBM-Cell

מעבד Cell-Processor הוא תוצאה של שיתוף פעולה בין שלוש החברות IBM, Sony, Toshiba כאשר מטרתו העיקרית לשמש כפלטפורמת החישוב עבור שוק המשחקים (gaming market). הטכנולוגיה מבוססת על 8 מעבדי Power-Pc הרצים בתדר של 3GHz כל אחד כאשר קיים בניהם Link מהיר והם מנוהלים ע"י מעבד Power-Pc נוסף כלומר סה"כ 8 מעבדי PPC (Power PC) עבור עיבוד הנתונים ועוד מעבד אחד עבור ניהול ובקרה. יחידות העיבוד נקראות SPE ויחידת הבקרה PPE. כל יחידת SPE היא בעצם ליבה של מעבד PPC התומך בפעולות SIMD ברוחב של 16 בתים ולכן ניתן לעבד עד 16 Bytes בכל פקודה בדומה למעבדי Intel.

יתרונות של מערכות מבוססות CELL דומות למדי לאלו של מעבדי Intel. כל מעבד SPE עובד בתדר של 3.2 GHz ובין יחידות החישוב קיים ממשק מהיר להעברת נתונים ביניהם (עד 16Gbyte/sec). בנוסף לכל יחידת חישוב יש כאמור קבוצת פקודות מסוג SIMD הדומות למדי לפקודות SSE (בעיקר בהתייחס לפקודות הפועלות על בית או 2 בתים). חיסרון עיקרי של ארכיטקטורה זו הוא כמות הזיכרון

הקטנה יחסית שיש בממוצע לכל יחידת חישוב (סה"כ $512\text{Mbyte}/8 = 64\text{Mbyte}$). נתון זה מגביל מעט את האפשרות שכל יחידת חישוב תעבד כמות מספיקה של נתונים ולכן נאלץ לתכנן את המערכת שיחידות חישוב שונות מבצעות פעולות שונות. דבר זה יאלץ לבצע את פונקציות עיבוד התמונה ב-pipeline כלומר יחידת עיבוד אחת מבצעת פעולות על תוצרים של יחידת עיבוד אחרת. החיסרון במימוש תוכנה המבוסס על pipeline הוא שקשה לגרום לכך שכל יחידות העיבוד יהיו עסוקות תמיד וכך קשה להגיע לניצולת גבוהה של מערכת עיבוד התמונה כלומר ה-load-balancing של המערכת יהיה מורכב וכל הוספה של אלגוריתם חדש למערכת או פונקציה עיבוד חדשה יצריך תכנון מחדש של חלוקת העיבוד בין יחידות החישוב. למרות מורכבות של ארכיטקטורת התוכנה יתכן והמאמצים לניהול מסובך יהיו כדאיים אם יחס ה-cost/performance של המערכת יתאים לדרישות המינימאליות שהוגדרו או מעבר לכך. בחינת עלותה של מערכת עיבוד תמונה מבוססת CELL דומה לזו המבוססת על טכנולוגיית Intel למרות שתדר העיבוד של כל יחידת SPE גדול מתדר העבודה של כל ליבת Intel. גם כאן נזדקק לאותו סדר גודל של יחידות חישוב ומחירו של כרטיס עם שני CELL הוא בסדר גודל של כרטיס עם 2 מעבדי אינטל. מצד שני יש בכל כרטיס פי שניים יותר יחידות חישוב (16 SPE) וכן לכאורה העלות של מערכת מבוססת CELL תהיה קטנה פי שניים מזו של אינטל אבל בגלל מגבלת הזיכרון והאילוץ לעיבוד ב-pipeline ניתן להניח שהיעילות תיפגע בסדר גודל של 40% ולכן בשקלול כולל נגיע למערכת בסדר גודל של $250K\$-200K\$$ שלמרות שהיא יותר נמוכה מזו של אינטל עדיין אינה מגיעה ליעדים שהוגדרו בדרישות ממערכת עיבוד התמונה. בנוסף לכך עלות פיתוח ותחזוק של תוכנה מבוססת pipeline עולה לעין שיעור על מערכת שבה כל יחידת חישוב במערכת מבצעים בדיוק את אותם פעולות ופונקציות.

ארכיטקטורת מערכת עיבוד תמונה מבוססת GPU

מעבדים גרפיים (GPU (Graphic Processing Unit הם טכנולוגיה מעניינת המבוססת על מעבדים גרפיים המכילים בתוכם עשרות (וכיום אפילו מאות בודדות) של יחידות חישוב או מעבדים העובדים בצורה בודדת על חלק מהתמונה העוברת עיבוד. יחידות אלו עובדות בתדר של 500Mhz-600Mhz וישנם בין 64-128 יחידות כאלה. את כל היחידות האלה מנהלת יחידת עיבוד מרכזית השולטת על חלוקת העבודה בין המעבדים ועובדת בתדר גבוה יותר מסדר גודל של 1.3Ghz. באופן עקרוני כל מעבד בסיסי כזה מעבד יחידות בגודל 32 סיביות. הרעיון הבסיסי של מעבד GPU הוא שיחידת חומרה ראשית מנהלת את הגישה לזיכרון החיצוני ומנסה לדאוג ע"י שימוש ב-hyper threading שבמקרה שמחכים למידע מהזיכרון ואחד מהמעבדים במצב stall מעבד אחר יעבוד. למעשה זוהי טכנולוגיית stream processing שבה נעשה שימוש מרובה ב-multithreading בחומרה כדרך להתמודד עם תופעת ה-memory latency במקום לעשות שימוש מורחב בארכיטקטורת cache היררכית (למרות שעדיין ישנם זיכרונות L1 המשותפים למספר ליבות). החיסרון העיקרי של ה-GPU שהוא איננו מעבד שבו ניתן להריץ תוכניות ניהול ולכן הוא צריך לצידו מעבד general purpose כדי להעביר לו את המידע ולשאוב את התוצאות. ישנם מספר בעיות שמונעות בחירה בטכנולוגיה זו כבסיס למערכת עיבוד תמונה:

- מכיוון שצריך לנהל כל GPU ע"י מעבדי עזר עלות המערכת מייקרת לעין שיעור מכיוון שלכל board עם ארבע GPUs דרוש board המנהל אותו.
- התלות בין מיקום המידע וגודלו משפיעה ישירות על מספר ה-threads שניתן ליצור ולכן גורר אילוצים קשים על חלוקת הנתונים בין המעבדים (ניצולת המעבדים תלויה ביחס בין כמות המידע בניהם)
- אין כמעט אפשרויות debug מכיוון שה-GPU הוא יחידה סגורה שהקוד שבה מנוהל ע"י חומרה מרכזית.
- טכנולוגיה זו עדיין איננה בשלה מבחינת נקודת המבט של הזנת ה-GPU במידע לעיבוד. ערוץ התקשורת בינו לבין המעבד המנהל שלו הוא PCI-Express.

ארכיטקטורת מערכת עיבוד תמונה מבוססת FPGA

טכנולוגיית FPGA כוללת רכיבי חומרה הניתנים לתכנות המכילים מספר רב של שערים לוגיים, יחידות זיכרון בסיסיות כגון Jkff. בעזרת לוגיקה זו ניתן לממש אלגוריתם לעיבוד תמונה כלשהו. היתרון הגדול של רכיבי FPGA הוא שניתן לבצע בעזרתם גם את תהליך רכישת התמונה (grabbing) וגם את עיבוד המידע. כל FPGA יכול לעבד את האותות המגיעים בפרוטוקול המשמש את תהליך

רכישת התמונה (למשל camera link שהוזכר בסעיפים הקודמים) וכך לבצע רכישה של חלק מהתמונה עליו הוא אמור לעבוד וכן לבצע את האלגוריתמים הדרושים לצורך מציאת הפגמים. ניתן באופן כללי למקבל פעולות בצורה יוצאת דופן אולם אז יהיה ניצול מקסימלי של האלמנטים ברכיב ויכול להיות מצב שלא יישארו אלמנטים למימוש מרכיבים אחרים באלגוריתמים.

מימוש האלגוריתמים במערכת מבוססת FPGA חייב להיות on-the fly כלומר המידע שנרכש עובר מידית לעיבוד. במערכות בהם האלגוריתמים פשוטים יחסית וכוללים מספר שלבים מועטים מימושם ב-FPGA יהיה יחסית יעיל. לעומת זאת ניהול פעולת האלגוריתמים נהיה מסובך ככל שהאלגוריתמים מסובכים יותר וכוללים פרמטרים רבים יותר מכיוון שקשה ליצור אלגוריתמים מבוסס חומרה שכולל מספר רב של פרמטרים. בעיה נוספת שקיימת במערכת מבוססת FPGA היא שבמידה ומימוש האלגוריתמים דורש מספר בלוקים מ – dies קודמים קשה מאוד לממש אותם כי צריך יחידות זיכרון גדולות על מנת לגשת למספר בלוקים אחורה (יחידות זיכרון אלו נקראות FIFOs) דבר שגוזל משאבים יקרים מהאלמנטים החומריים שבתוך ה-FPGA ומקשה מאוד על המימוש.

בנוסף לכך הרכיבים המתוחכמים בשוק עולים בסדר גודל של מאות דולרים (ואנו זקוקים לרכיבים מתוחכמים כדי לממש אלגוריתמים מסובכים) ולכן עלות בניית board מבוסס רכיבי FPGA איננה זולה. בנוסף חוסר הגמישות בפיתוח והוספה של אלגוריתמים חדשים למערכת מבוססת FPGA נגרם מהעובדה ששינוי ב-FLOW של האלגוריתמים דורש תכנון מחדש של מימשי החומרה ולכן זמני פיתוח ותחזוקה הם ארוכים.

לסיכום אופציית מימוש מערכת עיבוד התמונה על טכנולוגיית FPGA נדחית על הסף אבל כמו שניווכח בסעיפים הבאים במידה ויש מעבדים המסוגלים לקלוט מידע ע"י ממשקים נוחים ניתן לשלב אותם יחד עם FPGA יחסית פשוטים כך שתפקידם של ה-FPGA הוא לבצע את תהליך ה-grabbing ולקרוא את החלק המעניין (ROI=region of interest) עבור המעבד המתאים ושאר העיבוד יתבצע ע"י המעבד הנבחר.

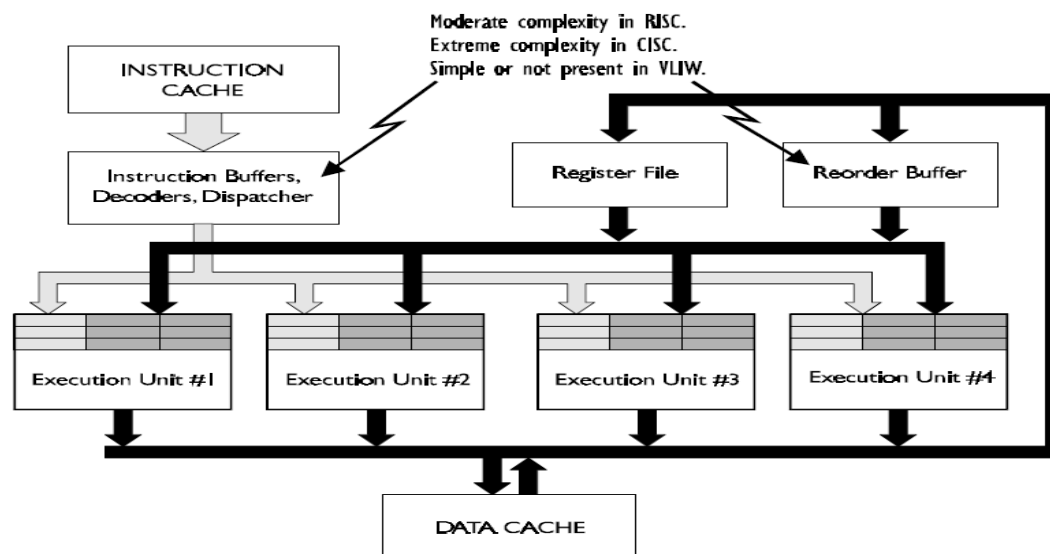
נספח 4 - תיאור יחידת עיבוד Processing Node

בהתאם למודול שהוגדר בפרק 3 בחרנו ב-DSP ממשפחת TMS320DM643x כבסיס ליחידת העיבוד המרכזית במערכת עיבוד התמונה שהוגדרה. בפרק זה נתאר את המכלול של יחידת העיבוד, מרכיביה העיקריים והשיטות והדרכים כדי להגיע לביצועים מקסימאליים. כאמור ה-Core הנבחר (c64+ של TI) עובד בטכנולוגיה של VLIW וכדי להגיע לביצועים מקסימאליים צריך להבין טכנולוגיה זו ולדעת איך לנצל. המעבד מכיל בנוסף ל-Core המבצע את החישוב גם רכיבי פריפריה שונים, זיכרונות ברמות L1, L2, בקר פסיקות ובקר DMA או לפי ההגדרה של EDMA TI (Enhanced Direct Memory Access). כל ההתקנים שתוארו לעיל בנוסף ל-Core C64+ מוגדרים כ-Mega Module והם שונים במעט בין DSP אחד למשנהו. בפרק זה נסרוק ונבין את השיקולים המנחים את תפעול ה-Mega Module להשגת ביצועים מקסימאליים. בהמשך הפרק נבחן מספר פונקציות עיבוד תמונה המשמשות כאבני בניין לאלגוריתמים במערכות עיבוד תמונה עבור WI ונראה איך תכונות ה-Core וה-Mega Module משמשות לקבלת ביצועים מקסימאליים.

תיאור הליבה הנבחרת C64+

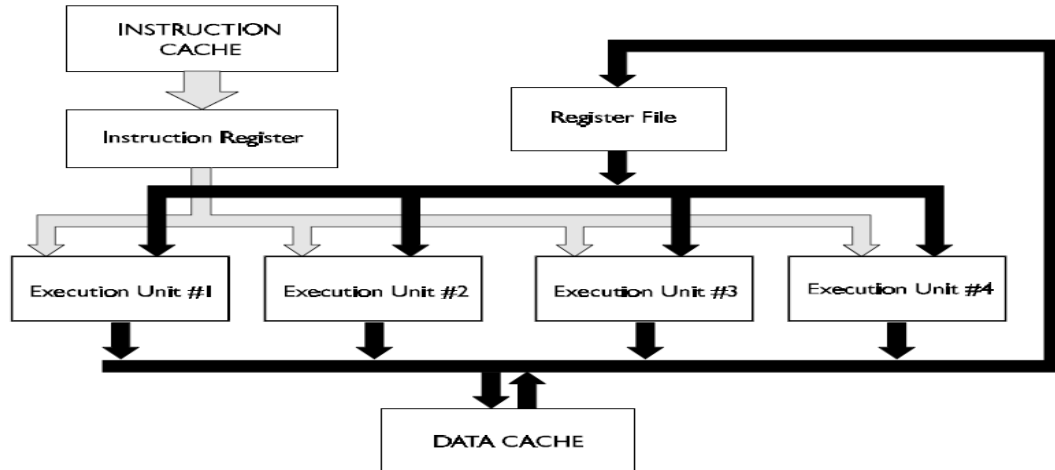
ליבה בארכיטקטורת VLIW

בפרק 3 הוצג המודל של מערכת עיבוד התמונה המתבסס על TMS320DM743x שהליבה שלו הינה C64+. ליבה זו מבוססת על טכנולוגיה הנקראת VLIW. המינוח של מושג זה הוא very Long instruction word (מילת הוראה ארוכה מאוד). מעבדי super-scalar כדוגמת מעבדי INTEL (שתכונותיהם הוצגו בפרק 3) מבצעים עיבוד בעזרת יחידות SSE/MMX ובנוסף כוללים מערכת חומרה מורכבת המנסה לזהות פקודות שאינן תלויות אחת בשנייה ולבצע אותם במקביל. לדוגמא ביצוע פקודה ביחידת SSE כמו פעולת מינימום וקטורית ובנוסף ביצוע load/store ועוד. פעולת זיהוי המרכיבים המקביליים בתוכנית שניתן לבצע במקביל נעשית ע"י יחידת "ביצוע שלא לפי הסדר" (out of order execution) שהיא יחידה הכוללת לוגיקה מורכבת למדי. בנוסף קיימת יחידת ה-branch prediction – שמנסה לזהות אם ומתי תתרחש פקודת ההסתעפות. יחידת ה-branch prediction משתמשת בזיכרון נוסף להקלטת הערכים הזמניים של האוגרים כדי להסיק האם תתרחש ההסתעפות או שלא. כמובן שמימוש לוגיקות אלו מייקר את המעבד, מעלה את ההספק הנומינלי שלו ומקטין את המשאבים שנדרשים על מנת ליישם זיכרונות cache בתוך המעבד ואת מספר האוגרים שלו. לרוב ההוראות במעבדים אלו יהיו בגודל של 32bits.



איור 65 תיאור מבנה עקרוני של מעבד Super-Scalar כדוגמת מעבדי Intel לעומת זאת טכנולוגיה VLIW כאמור כוללת הוראה ארוכה מאוד המתארת חבילה של פקודות המתבצעות במקביל ללא תלות אחת בשנייה. הרעיון המרכזי במעבדים המבוססים על VLIW הוא לקחת

את הפעולות המורכבות המתבצעות בחומרה (branch prediction, out of order execution) ובמקום זה ליישם בתוכנה בעזרת מהדר מתוחכם וכך לפשט מאוד את מימוש החומרה של המעבד דבר שמשליך מידית על מחירו, הספקו ועם זאת מאפשר להשיג ביצועים דומים ואף טובים יותר לזה של מעבדי super-scalar. כמובן שהחיסכון במשאבי חומרה מנוצל להגדלת ה-cache הוספת יחידות עיבוד וכדומה אבל ישנה תלות מרכזית במהדר מכיוון שכל התכנון של מעבד מבוסס VLIW מניח קומפיילר אינטליגנטי המסוגל לנהל את תזמון ההוראות ולמקבלן במידת האפשר המרבית. בנוסף לכך כל פקודה מתוך ההוראה הגדולה הכוללת פקודות כמספר יחידות העיבוד השונות יכולה לעבוד גם היא בצורה וקטורית. האיור הבא מציג ארכיטקטורה כללית למעבדים מבוססי VLIW וניתן לראות שאין היא כוללת יחידות האחראיות על branch prediction ו-out of order execution.



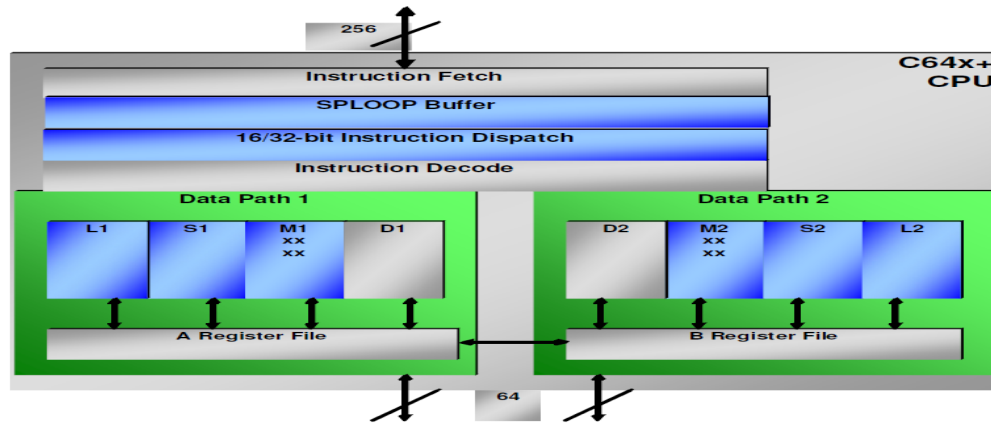
איור 66 תיאור מבנה עקרוני של מעבד מבוסס VLIW

ליבה C64+

הליבה C64+ של TI מבוססת על ארכיטקטורה VLIW וכוללת 8 יחידות ביצוע עצמאיות (functional unit). הליבה כוללת 2 יחידות הנקראות A ו-B שכל אחת כוללת 32 אוגרים בגודל 32bits. כל יחידה כוללת 4 יחידות ביצוע הנקראות L, M, D, S. כל יחידה ביצוע כזאת יכולה לבצע פעולות על 4 נתונים בגודל בית או על 2 נתונים בגודל שני בתים או על נתון אחד בגודל 32 סיביות. תפקיד כל יחידה ביצוע הוא כדלקמן:

- יחידה M מבצעת בעיקר פעולות כפל.
 - יחידה D מבצעת בעיקר פעולות load ו-store.
 - יחידה S מבצעת בעיקר פעולות הזזה, הסתעפות, והשוואה.
 - יחידה L מבצעת בעיקר פעולות לוגיות וחשבוניות.
- בנוסף ישנם יחידות X המאפשרות מעבר בין יחידות ואוגרים מקבוצה A ל-B ולהיפך. יחידה T כגישה לזיכרון. ישנם פקודות כלליות שמתבצעות ע"י כל היחידות (למשל פקודות לוגיות)

Program fetches are 256-bit aligned



Dual 64-Bit Load/Store Paths

איור 67 תיאור מבנה עקרוני של ליבת C64+.

הרעיון הבסיסי במעבד זה הוא שכל הוראה כוללת עד 8 פקודות שכל אחת יכולה להתבצע ללא תלות בשנייה. כדי להגיע לאי-תלות כזאת משתמש המעבד בטכניקת pipeline. כדי להבין את המושג pipeline נתבונן בלולאה המכילה את הפקודות הבאות:

LDH||LDH
MPY
ADD

הסבר הפקודות: הפקודה LDH טוענת מהזיכרון נתון בגודל 16 סיביות. מכיוון שישנם שתי יחידות D ניתן לטעון שני נתונים כאלה. הסימן || מציין שפקודות אלו מתבצעות במקביל. הפקודה MPY מכפילה בין שני הנתונים שנטענו בפקודה הקודמת והפקודה ADD מסכמת אותם. פעולה זו הינה אופיינית לעיבוד תמונה כך שבעצם הכפלנו ערכים מתאימים של 2 וקטורים וסיכמנו אותם. נניח כרגע שכל פקודה מתבצעת במחזור אחד (machine cycle) של המעבד ואנו מבצעים את הלולאה 5 פעמים נקבל סה"כ 15 מחזורים. הטבלה הבאה מדגימה את סדר ביצוע הפקודות ללא שימוש ב-pipeline.

Cycle	.D1	.D2	.M1	.M2	.L1	.L2	.S1	.S2
1	ldh	ldh	.	.	.	.	.	.
2	.	.	mpy	.	.	.	.	.
3	.	.	.	.	Add	.	.	.
4	ldh	ldh	.	.	.	.	.	.
5	.	.	mpy	.	.	.	.	.
6	.	.	.	.	Add	.	.	.
7	ldh	ldh	.	.	.	.	.	.
8	.	.	mpy	.	.	.	.	.
9	.	.	.	.	Add	.	.	.

טבלה 12 סדר ביצוע הפקודות בקוד שאיננו pipe-line

התבוננות בטבלה מראה שבכל שורה בלולאה פעילה יחידה אחת או מקסימום שניים (יחידות D) אבל לא מעבר לכך. נוכל להאיץ את החישוב ע"י שימוש בטכניקת pipeline בבניית הלולאה. דבר זה יעשה ע"י כך שנעשה שימוש בכל יחידות הביצוע בכל שורה בלולאה בצורה שכל יחידת ביצוע מעבדת נתון של איטרציה שונה. הטבלה הבאה מראה את סדר ביצוע הפקודות בקוד המשתמש ב-pipeline.

Cycle	.D1	.D2	.M1	.M2	.L1	.L2	.S1	.S2
-------	-----	-----	-----	-----	-----	-----	-----	-----

1	ldh	ldh	.	.	.	.	.	.
2	ldh	ldh	mpy	.	.	.	.	.
3	ldh	ldh	mpy	.	Add	.	.	.
4	.	.	mpy	.	Add	.	.	.
5	.	.	.	.	Add	.	.	.

טבלה 13 סדר ביצוע הפקודות בקוד מבוסס pipeline

מהתבוננות בטבלה אנו רואים (כל צבע מסמן איטרציה שונה) שפקודות ה-ldh במחזור הראשון (צבע ירוק) מתבצעות ולאחר מכן באיטרציה השנייה מתבצעות פקודות ldh של האיטרציה הבאה ופקודת mpy של הנוכחית. בהמשך מתבצעות פקודות ldh של שתי איטרציות קדימה, פקודת mpy של האיטרציה הבאה ופקודות סיכום של הנוכחית. בשלב זה ניתן להגיד שה-pipeline מלא אבל כמוכן שלקח לנו שתי איטרציות למלא אותו. החלק הגורם לטעינת ה-pipeline בלולאה נקרא prolog. החלק בלולאה שבו מירב היחידות עובדות נקרא kernel והחלק האחרון שבו מתבצעות האיטרציות האחרונות נקרא epilog. בהנחה שהלולאה ארוכה אז רוב האיטרציות שבה מתבצעות בפקודות האסמבלר מהצורה הבאה:

LDH||LDH||MPY||ADD

וכך בעצם קיבלנו שכל איטרציה בלולאה מתבצעת במחזור אחד וסה"כ נקבל חיסכון בגורם 3 בזמן ריצת הלולאה. באופן כללי נוכל לתאר את טכניקת ה-pipeline בצורה הבאה:
בהינתן הלולאה הבאה:

```
loop:
    load
    operate
    store
    bdec loop, count          ; decrement count, branch if count != 0
                                (n מציין איטרציה/חזרה בלולאה המקורית)

    load(n)                   ; loop prolog
    load(n+1) || operate(n)
    load(n+2) || operate(n+1) || store(n)

loop:
    load(n+3) || operate(n+2) || store(n+1) || bdec loop, count(n) ; loop kernel
    operate(n+3) || store(n+2)
    store(n+3)                ; loop epilog
```

בצורה כזאת הרבה יותר עבודה נעשית בכל חזרה בלולאה והלולאה רצה $count+2$ פעמים ומהר בערך פי 4 מהלולאה המקורית.

מבנה ה-pipeline בליבה C64+

בסעיף הקודם תוארה פעולת ה-pipeline באופן עקרוני. אולם כשמגיעים למימוש pipeline במעבד מעשי העניינים מסובכים מעט יותר. הסיבות לכך מקורן בעובדה שהמעבד צריך לבצע הבאה של ההוראה (המכילה כאמור 8 פקודות) לפענח לשגר אותן לשמונת יחידות הביצוע בנוסף לכך שההשהיה או זמן הביצוע של כל פקודה משתנה בתלות בסוג הפקודה (כלומר זמן הביצוע שונה מפקודה לפקודה). תהליך ה-pipeline כולל את השלבים הבאים המתבצעים עבור כל הוראה שהמעבד מבצע:

- הבאת ההוראה - fetch.
- פענוח ההוראה וקידודה - decode.
- ביצוע ההוראה - execute.

כל ההוראות ב-C64+ עוברות את שלושת השלבים שתוארו לעיל. שלב ה-fetch כולל ארבע פאזות לכל ההוראות, שלב ה-decode כולל שני פאזות עבור כל ההוראות. כאמור שלב הביצוע דורש מספר שונה של פאזות התלויות בסוג הפקודה ולכן מסבך כפי שנראה בהמשך בניית pipeline יעיל. האיור הבא מתאר את כל שלבי ה-pipeline



איור 68 תיאור שלבי ה pipeline עבור ליבת C64+.

באופן עקרוני ארבע הפאזות של שלב ה- fetch ב- pipeline הם כדלקמן:

- שלב ייצור בכתובת PG (program address generate)
- שלב שליחת הכתובת PS (Program address send)
- שלב הגישה והמתנה להוראה PW (program access ready wait)
- שלב קבלת ההוראה PR (program fetch packet received)

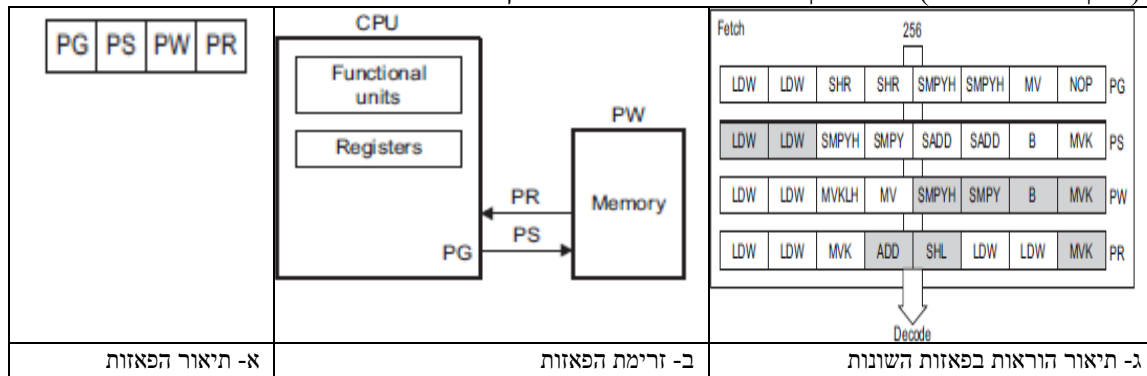
המעבד משתמש כאמור ב- fetch packet של שמונה פקודות שכולם נקראות ומעובדות בתהליך ה- fetch ביחד.

הרעיון המרכזי להשגת יעילות מרבית של ה- pipeline הוא לגרום למעבד לבצע הוראות שבה כל אחת משמונה הפקודות יכולה להתבצע ביחידה אחרת. חלוקה כזאת היא תגרום לכל שלבי ה-

pipeline להיות יעילים ביותר כפי שנראה בהמשך. דוגמא להוראה מסוג זה היא פקודה מהצורה הבאה:

ADD2	.L2X	B24, A1, B10
ADD2	.D1X	A22, B19, A17
ADD2	.S1	A7, A19, A4
ADD2	.L1	A18, A4, A7
SHRMB	.S2	B23, B22, B27
MPYU4	.M1	A17, A28, A5: A4
LDNDW	.D2T2	* - B3(6), B19: B18
MPYU4	.M2	B4, B30, B23: B22

בפקודה זו מתבצעות שתי הכפלות (אחת ב M1 והשנייה ב M2) של 4 בתים בארבע אחרים (הפקודה MPYU4) בנוסף מתבצעת טעינה של 8 בתים מכתובת שאיננה מיושרת (LDNDW) 4 פעולות חיבור של 2 מספרים בגודל 16 ביט (הפקודות ADD2) ופעולת הזזה בין שני אוגרים בגודל 32 סיביות (הפקודה SHRMB). תיאור עקרוני של שלבי ה- fetch נתון באיורים הבאים:



איור 69 א-ג תיאור מפורט של שלב ה- fetch.

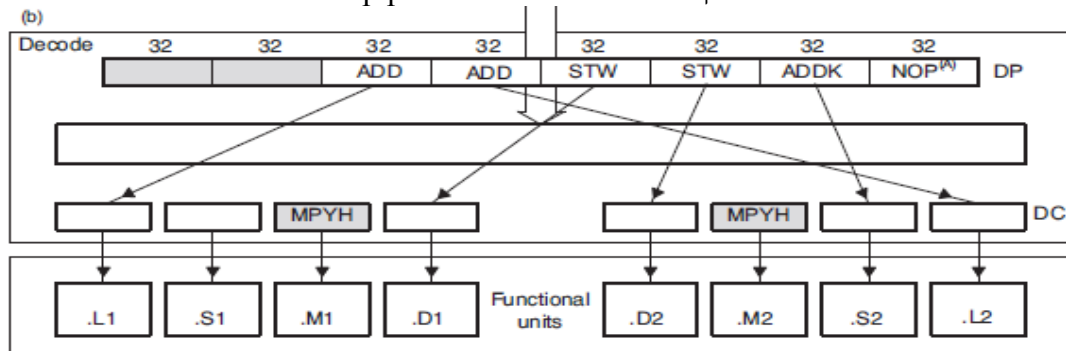
איור א מתאר את השלבים בצורה טורית. באיור ב אנו רואים ששלב ה PG מתרחש בתוך המעבד, שלב ה- PS כולל את שליחת הכתובת לזיכרון, בשלב ה- PW המעבד ממתיך להוראה שתגיע ובשלב ה- PR ההוראה מתקבלת. שלב ה- fetch כולל תמיד הבאה של 8 פקודות כהוראה אחת גם אם לא ניתן לבצע אותם על כל שמונה היחידות (במידה שיש שתי הוראות על אותה יחידה). באיור ג ניתן לראות שהפקודה שנמצאת בשלב הקידוד ב- pipeline כולל 2 פאזות כדלקמן:

- שלב השיגור של הפקודה ליחידה מתאימה - DP (instruction dispatch).
- שלב הקידוד של הפקודה - DC (instruction Decode).

בפאזת ה- DP ה- fetch packets מופצלות לחבילות ביצוע (execution Packet). כל חבילת ביצוע כוללת פקודה אחת או מ 2 עד 8 פקודות שמתבצעות במקביל.

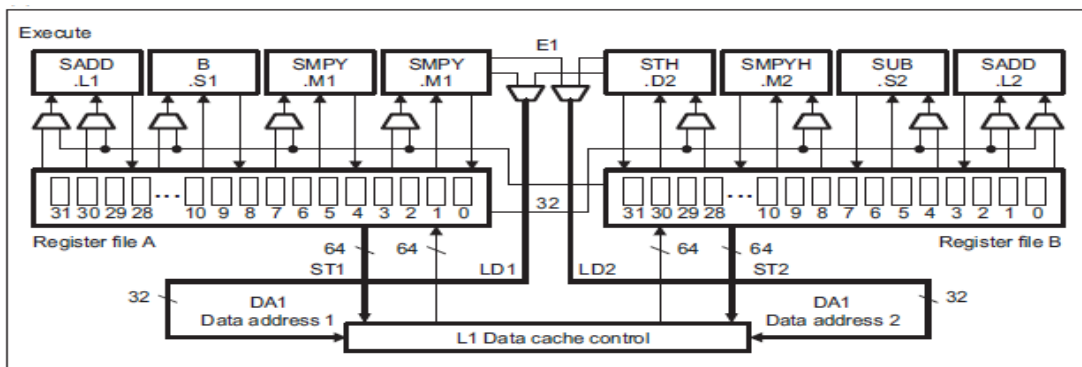
באיור 51-ג ניתן לראות שההוראה שנמצאת בשלב ה- PR בנויה מארבע חבילות ביצוע, ההוראה שנמצאת בשלב ה- PW ו- PS כוללת שתי חבילות ביצוע, והפקודה שנמצאת בשלב ה- PG כוללת חבילת ביצוע אחת. בשלב ה- DC מקשר בין האוגרים שיעזרו בביצוע הפקודה ובין יחידת הביצוע.

האיור הבא מתאר דוגמא לפאזות בתוך שלב ה- decode ב- pipeline.



איור 70 תיאור מפורט של שלב ה- decode.

ניתן לראות fetch packet שכולל 2 יחידות ביצוע שהם מעובדות בפאזות של שלב ה- decode. שלב ה- DC מראה שתי פקודות MPYH המתבצעות במקביל ובשלב הבא מחכות שש פקודות המתבצעות במקביל. שלב הביצוע כולל 5 פאזות והוא משתנה בין פקודה לפקודה. שלב זה ב- pipeline משחק תפקיד חשוב בהבנת יכולות הביצוע של המעבד והוא יוסבר בפירוט בהמשך. האיור הבא מראה דוגמא של שלב הביצוע.



איור 71 תיאור שלב ה- execution.

תפקיד השלבים השונים הוא כדלקמן:

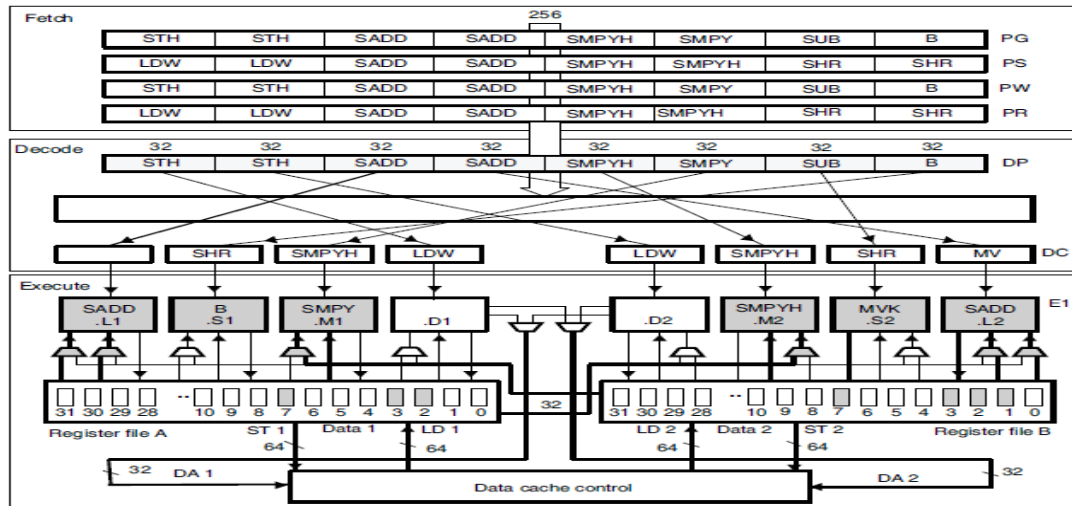
- E1 – כל פקודה מתבצעת לפחות בשלב זה. פקודות כגון ADDU4, CMPGTU4 ועוד דורשות זמן ביצוע של E1 בלבד.
- E2 – פקודות MPY דורשות 2 מחזורים (E1, E2) לביצוע.
- E4 – פקודת LOAD לוקחת 4 מחזורים.
- E5 – פקודת BRANCH לוקחת 5 מחזורים.

כאמור העילות הינה מקסימאלית אם יש חבילת ביצוע אחת עבור כל fetch packet כמתואר באיור הבא:

Fetch packet	1	2	3	4	5	6	7	8	9	10	11	12	13
n	PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5		
n+1		PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	
n+2			PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5
n+3				PG	PS	PW	PR	DP	DC	E1	E2	E3	E4
n+4					PG	PS	PW	PR	DP	DC	E1	E2	E3
n+5						PG	PS	PW	PR	DP	DC	E1	E2
n+6							PG	PS	PW	PR	DP	DC	E1
n+7								PG	PS	PW	PR	DP	DC
n+8									PG	PS	PW	PR	DP
n+9										PG	PS	PW	PR
n+10											PG	PS	PW

איור 72 תיאור פעולת ה- pipeline עבור רצף הוראות עם חבילת ביצוע אחת.

האיור הבא מתאר צילום מצב של קטע קוד ומצב ה- pipeline שלו. בשלבי ה- fetch נמצאות הוראות מלאות עם 8 פקודות כל אחת. ביחידת הביצוע ישנו packet עם 6 פקודות ובשלב ה- DC מוכנות לביצוע הוראה עם 7 פקודות כשמעליה בשלב ה- DP מוכנה הוראה עם 8 פקודות וכך הלאה בשלב ה- fetch.



איור 73 דיאגרמת מלבנים עבור הפאזות ב- pipeline בתוכנית הדוגמא שבאיור הבא

```

SADD      .L1      A2,A7,A2      ; E1 Phase
          SADD      .L2      B2,B7,B2
          SMPYH      .M2X      B3,A3,B2
          SMPY      .M1X      B3,A3,A2
          B          .S1      LOOP1
          MVK        .S2      117,B1

LDW       .D2      *B4++,B3      ; DC Phase
LDW       .D1      *A4++,A3
MV        .L2X      A1,B0
SMPYH     .M1      A2,A2,A0
SMPYH     .M2      B2,B2,B10
SHR       .S1      A2,16,A5
SHR       .S2      B2,16,B5

LOOP1:
          STH       .D1      A5,*A8++[2]      ; DP, PW, and PG Phases
          STH       .D2      B5,*B8++[2]
          SADD      .L1      A2,A7,A2
          SADD      .L2      B2,B7,B2
          SMPYH     .M2X      B3,A3,B2
          SMPY      .M1X      B3,A3,A2
[B1] B     .S1      LOOP1
[B1] SUB    .S2      B1,1,B1

LDW       .D2      *B4++,B3      : PR and PS Phases
LDW       .D1      *A4++,A3
SADD      .L1      A0,A1,A1
SADD      .L2      B10,B0,B0
SMPYH     .M1      A2,A2,A0
SMPYH     .M2      B2,B2,B10
SHR       .S1      A2,16,A5
SHR       .S2      B2,16,B5
    
```

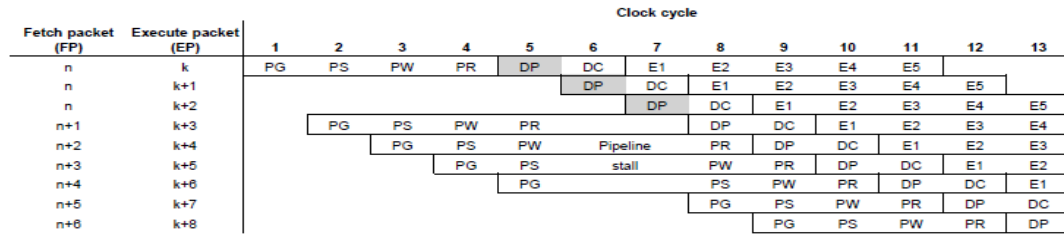
איור 74 דוגמא לקטע קוד עבור שלבי ה- pipeline שבאיור הקודם

כאשר כל fetch packet כולל כמה חבילות ביצוע כדוגמת פסיאודו הקוד הבא :

```

|| instruction A ;      EP k      FP      n
|| instruction B ;
||
|| instruction C ;      EP k + 1    FP      n
|| instruction D
|| instruction E
||
|| instruction F ;      EP k + 2    FP      n
|| instruction G
|| instruction H
||
|| instruction I ;      EP k + 3    FP      n + 1
|| instruction J
|| instruction K
|| instruction L
|| instruction M
|| instruction N
|| instruction O
|| instruction P
    
```

שממשיך לאחר מכן עם execution packet ה- k+4 עד ה- k+8 שיש להם 8 פקודות במקביל ה pipeline יראה כך:



איור 75 תיאור פעולת ה-pipeline עבור רצף הוראות עם מספר חבילות ביצוע.

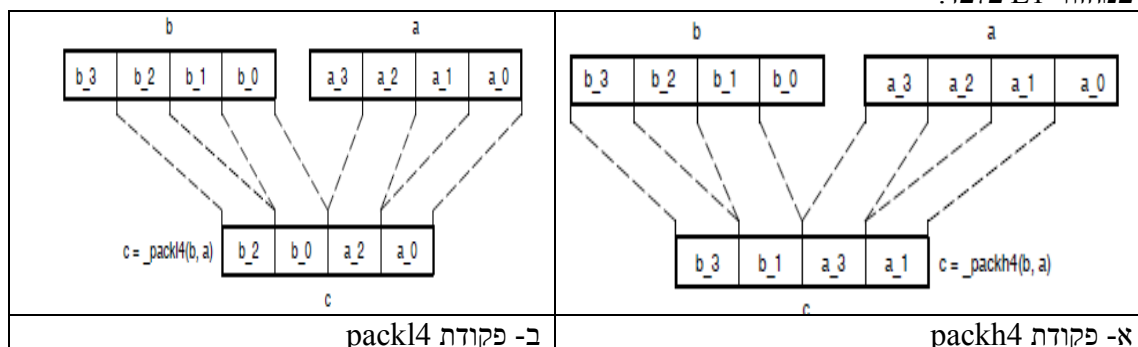
רואים שאיור ש- fetch packet מספר n כולל 3 חבילות ביצוע ואחריו שש FP עם שמונה פקודות לביצוע. כאשר FP מספר n עובר פאזות ה- fetch במחזורים 1-4 מתחיל תהליך ה- packet fetch עבור כל ה- FP שבהמשך. במחזור 5 בשלב ה- DP המעבד מזהה שיש שלוש חבילות ביצוע (k עד k+2) ב- FP מספר n. דבר זה גורם ל- pipeline להיתקע (stall) כדי לאפשר לפאזת ה- DP להתחיל את הביצוע של חבילת ביצוע k+1 ו- k+2 במהלך מחזורים 6 ו-7. כאשר חבילת הביצוע k+2 מוכנה לתזוזה לשלב הבא עצירת ה- pipeline משתחררת. כתוצאה מכך ה- FP שמספרם n+1 עד n+4 כדי שלמעבד יהיה זמן כדי לבצע את פאזת ה- DP עבור כל אחת מחבילת הביצוע k עד k+2 ב- FP שמספרו n. בנוסף FP שמספרו n+5 גם נעצר במחזורים 6 ו-7 מכיוון שהוא לא הורשה להיכנס לפאזת ה- PG עד שעצירת ה- pipeline השתחררה. ה- pipeline ממשיך לעבוד מלא עד שתגיע הוראה חדשה המכילה מספר חבילות ביצוע. לסיכום הרעיון הבסיסי הוא ליצור הוראות עם 8 פקודות כדי להשיג ניצול מקסימלי של ה- pipeline ולא לעצור אותו ולכן כדי לקבל יעילות מקסימאלית רצוי למלא פקודות חסרות ב NOP (או בעצם לסגור הוראות עם פחות מ 8 פקודות בהוראה אחת) כדי להבטיח ניצול נכון יותר של ה- pipeline למרות שפקודת NOP איננה מבצעת דבר וגם לא משוגרת לאף יחידה.

תיאור הפקודות הווקטוריות הנפוצות בעיבוד תמונה ב- C64+

בסעיף הקודם תואר מבנה ה- pipeline של ה- DSP אבל כדי להגדיל את הביצועים עלינו לנצל את העובדה שבנוסף לטכניקת ה- pipeline המאפשרת להפעיל עד 8 יחידות במקביל כל יחידת ביצוע כזאת מסוגלת לעבד מספר נתונים. באופן עקרוני כל יחידה יכולה לעבוד על 4 יחידות בגודל בית או 2 יחידות בגודל 16 סיביות או על נתון אחד בגודל 32 סיביות. מתכנני ה- C64+ כללו פקודות מיוחדות המאפשרות עבודה וקטורית ומקלות על ביצוע אופטימיזציה והעלאת הביצועים במימוש אלגוריתמים. הרעיון המרכזי הוא שעבודה על נתון אחד לא קשורה לנתון השני. להלן מספר פקודות נבחרות שידגימו את האמור לעיל.

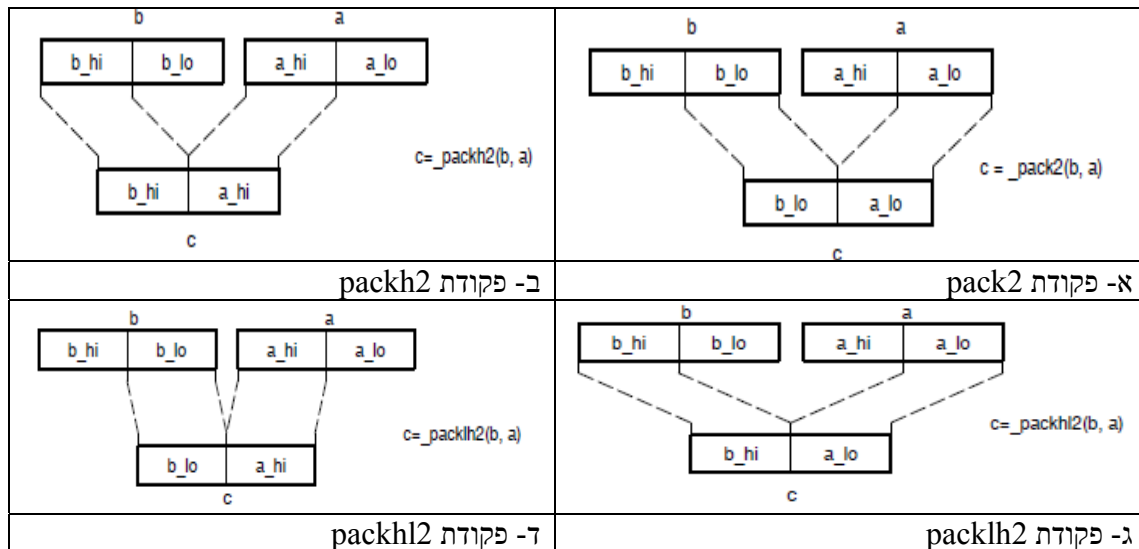
תיאור פקודות pack

פקודות pack לסוגיהם השונים מטרתם לבנות מידע באוגר חדש מתוך חלקי מידע הנמצאים ברגיסטרים אחרים. הפקודות packl4, packh4 בורות 4 בתים מתוך שני אוגרים אחרים. פקודה זו מתבצעת במחזור E1 בלבד.

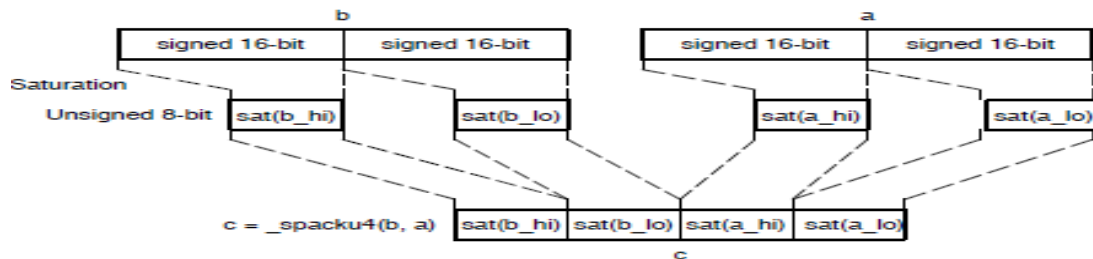


איור 76 א-ב תיאור גרפי של פקודות packl4, packh4

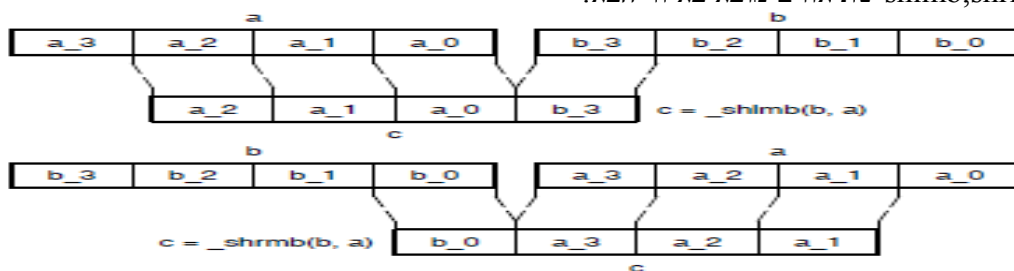
כאמור אלו פקודות הפועלות על 4 בתים ונעשות ביחידה L. בנוסף קיימות הפקודות packl2, packh2, packl4, packh4 שמבצעות פעולות דומות על נתונים בגודל 16 סיביות ותיאורם מובא באיור הבא:



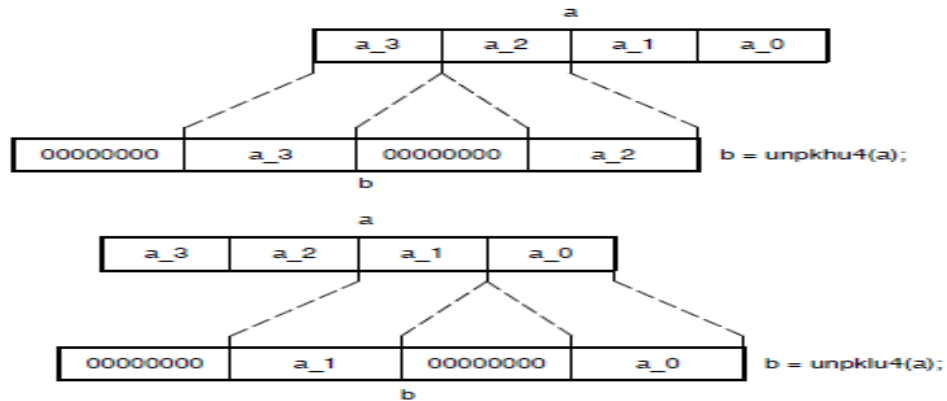
איור 77 א-ד תיאור גרפי של פקודות pack2, packh2, packlh2, packhl2
פקודת spacku4 מבצעת המרה מגדלים של 16 סיביות המייצגים מספר מסומן לבית תוך כדי קיצוץ במידה של רוויה, תיאורה מובא באיור הבא:



איור 78 תיאור גרפי של פקודות spacku4
פקודות נוספת שמקנה שליטה במעברי מידע בין 2 אוגרים היכולים לייצג וקטור של 8 בתים הם shlmb, shrmb שתיאורם מובא באיור הבא:



איור 79 תיאור גרפי של פקודות shlmb, shrmb
המטרה של פקודות אלו לבצע פעולת הזזה בין שני אוגרים.
פקודות העושות את הפעולה ההפוכה לפקודת packl4 היא unpackl4, unpackh4 ושתיאורם מובא באיור הבא:

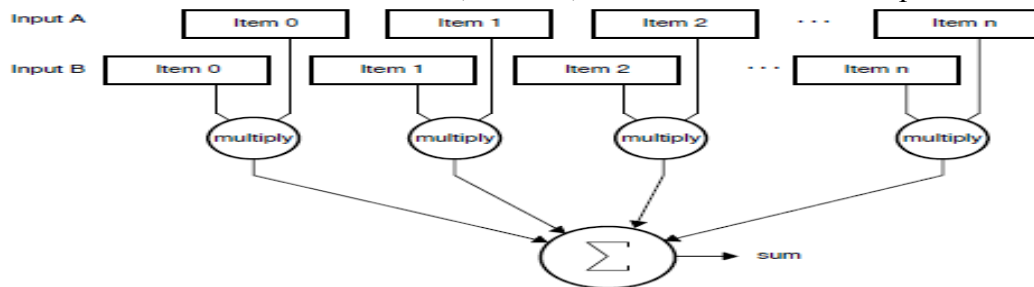


איור 80 תיאור גרפי של פקודות unpackhu4, unpacklu4

פקודות נוספות כגון: spack2, shr2, shr, shur, extu, ext, swap, rotl. [במדריך](#).
כל הפקודות שתוארו כאן מהוות אבני בניין חשובות בכיצוע פעולות על וקטורים כדי להשיג ביצועים מקסימאליים ודוגמאות רבות לשימושם יוצגו בהמשך.

פקודות multiply ו-Dot-Product

אחת הפעולות החשובות בעיבוד תמונה היא מכפלה ומכפלה וסיכום. פקודות אלו מתבצעות בשני הפאזות E1, E2. פקודות המעבד העיקריות התומכות בכך הם כדלקמן:
פקודות dot-product שמאפשרת הכפלה של וקטור עם וקטור וסכימתו כמובא באיור הבא:



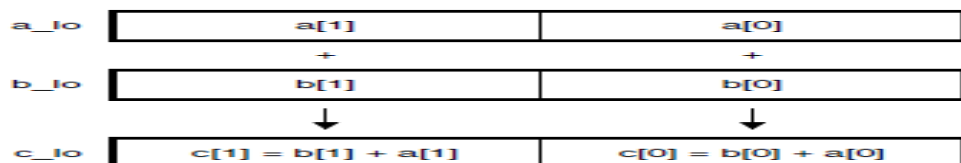
איור 81 תיאור גרפי עקרוני של פקודות dot-product

פקודה זו עובדת על גדלים שונים כמו, 16 סיביות, בית ועוד. במידה ואנו מעוניינים רק לכפול נשתמש באוסף הפקודות מסוג mpyx שתיאורם העקרוני דומה לזה שבאיור 56 למעט העובדה שאין סיכום של תוצאות ההכפלה והם נשארות באוגר היעד.

פקודת חיבור חיסור וקטור מקבילי

פקודת חיבור/חיסור מקבילי שימושיות כאשר נתון לנו וקטור עם נתון בגודל בית או 16 סיביות ומעוניינים לבצע פעולת חיבור/חיסור. הפקודות sub4, subabs4, add2, add4, saddu4, sadd2, saddu2, איור גרפי של הפקודה add2 מובא באיור הבא:

`c_lo = _add2(b_lo, a_lo);`

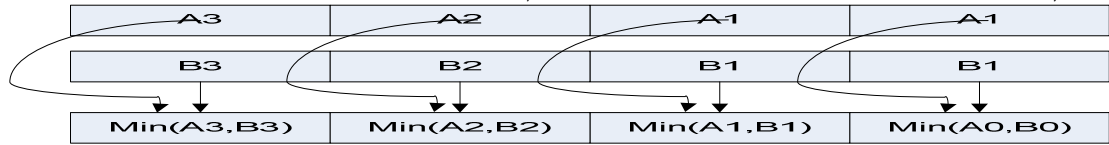


איור 82 תיאור גרפי עקרוני של פקודות add2

הרעיון בשאר הפקודות דומה למדי כאשר הפקודה מסתיימת בספרה 4 משמעותה שהיא פועלת על 4 נתונים בגודל של בית וכאשר היא מסתיימת בספרה 2 משמעותה שהיא פועלת על שני נתונים בגודל 16 סיביות (עיקרון זה נכון לגבי כל הפקודות ב-C64+).

פקודות minimum, maximum ו- avg

סוג נוסף של פקודות המאפשרות לבצע חישובים על וקטור הם פקודות min2, max2, avg2, min4, max4. הפקודות הפועלות על 4 בתיים מתייחסות לנתונים שם כמספרים לא מסומנים (בין 0-255) ואילו הפקודות שעובדות על 16 סיביות מתייחסות למספרים כמספר מסומן במשלים ל-2. לדוגמה האיור הבא מתאר את הפקודה min4:



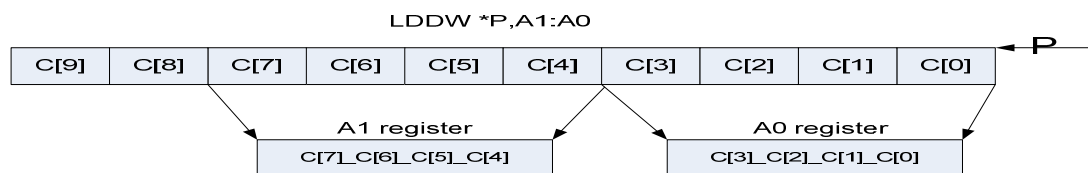
איור 83 תיאור גרפי עקרוני של פקודת min4

פקודת טעינת ואחסון וקטור

כדי להיות מסוגלים לעבד וקטור יש צורך בפקודות שבעזרתם ניתן לטעון וקטור מהזיכרון כדי שנוכל לעבדו וכן לאחסן וקטור המכיל תוצאות כלשהם. הליבה C64+ תומכת בטעינה/אחסון של נתונים בגודל 64 סיביות, 32 סיביות, 16 סיביות ובית אחד. כל טעינה/אחסון כזו יכולה להיות מיושרת או שלא מיושרת יחסית לגודל הנתון. הפקודות התומכות באמור לעיל

הם: ldb, ldh, ldw, lddw, ldndw, stb, sth, stw, stdw, stndw.

לדוגמה הפקודה LDDW מאפשרת לטעון וקטור בגודל 8 בתיים שהכתובת שלו מיושרת לגודל הנתון ומתוארת באיור הבא:



איור 84 תיאור גרפי עקרוני של פקודת LDDW

לולאה יעילה אופיינית למימוש פונקציות עיבוד תמונה תתחיל בסדרה של טעינות של מידע מהזיכרון ובדרך כלל תהיה זו טעינה בגודל מקסימלי האפשרי כמו באיור שלעיל. לאחר שהמידע נטען לאוגרים נוכל לעבד אותן בעזרת הפקודות השונות שחלקן תוארו בסעיפים הקודמים. המעבד יכול לבצע שתי טעינות מיושרות במחזור אחד או אחת שאיננה מיושרת.

פקודות השוואה ובניית מסכה

פקודה הכרחית בכל סוג של קוד או אלגוריתמים היא פקודת השוואה והסתעפות. מכיוון שפקודת הסתעפות לוקחת 6 מחזורים והיא איננה באה בחשבון במימוש פקודות השוואה עבור קוד שחייב להיות יעיל מבחינת זמן הריצה. כדי להימנע משימוש בפקודת הסתעפות (branch command) משתמשים באחת משתי הטכניקות הבאות לפי הצורך:

- ביצוע פקודה בתלות בערכו של אוגר תנאי.
 - שימוש במסכות עבור וקטור כדי לבצע פעולות או למנוע אותן.
- נדגים את הטכניקה הראשונה ע"י פסודו קוד הבא:

```
If A>B
    A=A+2;
Else
    A=A-4;
```

מימוש נאיבי של קוד זה יכול להיות בצורה הבאה:

```
CMPGT A0,B0,A1
[A1] BR place1
    Sub A0,4,A0
    BR Next
Place1:
    Add A0,2,A0
```

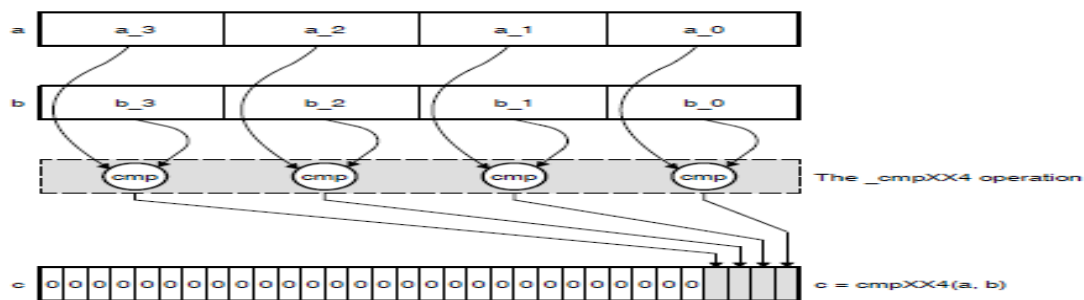
Next:

שימוש בשתי פקודות branch יגרום למעבד לממש קוד זה עם pipeline מאוד לא יעיל כי זמן ביצוע של כל branch אורך שישה מחזורי מכונה ולכן נאלץ לרפד קוד זה עם מספר גדול של פקודות NOP. כדי למנוע מצב שבו המעבד יסתעף למקום אחד שבו הוא יבצע הוספה של 2 ל-A או למקום אחר שבו הוא יחסיר מ-A 4 ניתן לבצע כל הוראה בתלות בערכו של אוגר תנאי. תרגום של קוד מהסוג הנ"ל יתבצע בצורה הבאה:

```
CMPGT A0,B0,A1 ; A1 used as conditional Register
[A1] Add A0,2,A0 ;
|| [!A1] Sub A0,4,A0
```

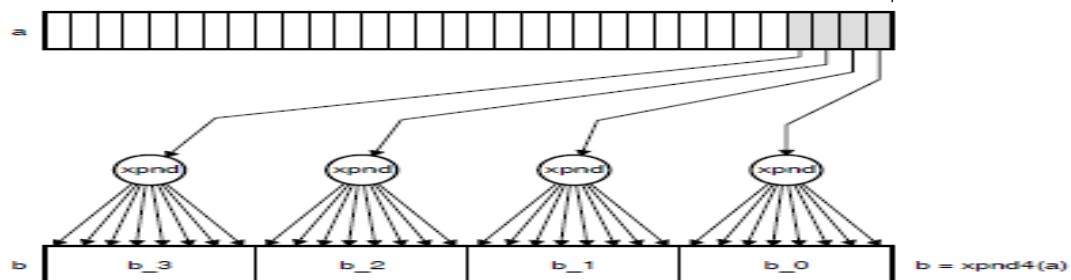
בקוד האסמבלר שלעיל האוגר A1 משמש כאוגר תנאי. אם $A0 > B0$ ערכו יהיה 1 אחרת 0. פקודת האסמבלר הבאה כוללת שתי פקודות המתבצעות במקביל אבל למעשה רק אחת מהן תשפיע על ערכו של הרגיסטר A0. אם $A1 == 1$ תתבצע הפקודה `ADD A0,2,A0` שמוסיפה ערך 2 לאוגר A0. אם $A1 == 0$ תתבצע הפקודה `Sub A0,4,A0` שמחסירה מ-A0 את הערך 4. התנאי [A1] משמעותו לבצע את הפקודה אם $A1 = 1$ והתנאי [!A1] משמעותו לבצע את הפקודה אם $A1 = 0$.

בצורה הזאת ניתן לכתוב קוד תוך כדי הימנעות מקפיצות והסתעפויות מיותרות, כמובן שאם הלולאה מאוד מסובכת וכוללת הרבה תנאים יתכן שחוסר באוגרים יאלץ אותנו להשתמש בפקודות הסתעפות כדי לממש קוד זה. דרך שנייה לטפל בקוד הדורש ביצוע פעולות שונות בתלות בתנאי הוא להשתמש בפקודות השוואה הפעולות על וקטור וליצור מהם מסכה בעזרת פקודות הרחבה מתאימות. מסיכה זו יכולה לעזור בביצוע הקוד. לדוגמא: הפקודה `CMPGTU4` מאפשרת השוואה של 4 נתונים בגודל Byte עם 4 נתונים אחרים בגודל של בית כמתואר באיור הבא:



איור 85 תיאור גרפי עקרוני של פקודות CMPGTU4

התוצאה המתקבלת באוגר היעד כוללת סיבית שתכיל "1" במקרה שההשוואה המתאימה נכונה (כלומר נתון אחד גדול מהשני) אחרת "0". לאחר שהתקבלה התוצאה ניתן להרחיב אותה בעזרת פקודת XPND4 כדי לקבל מסכה כפי שמודגם באיור הבא:



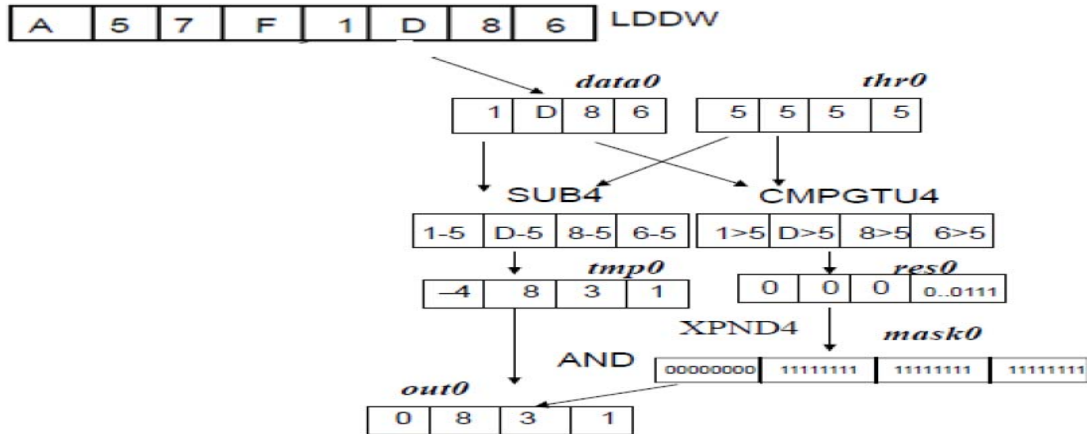
איור 86 תיאור גרפי עקרוני של פקודות XPND4

סדרת הפעולות המורכבת מפקודת `CMPGTU4` ולאחריה פקודת `XPND4` שימושית מאוד למימוש קטעי קוד הדורשים בדיקת תנאי וביצוע פעולות בהתאם לתוצאת התנאי. לדוגמא נניח שאנו רוצים לבצע את קטע הקוד הבא :

```
For each pixel
  If pixel < Threshold
    Then
      Output_pixel = 0 ;
    Else
```

$$\text{Output_pixel} = \text{pixel} - \text{Threshold}$$

כלומר אנו מעוניינים ליצור תמונה חדשה מתמונה נוכחית שבה הערך של פיקסל יהיה "0" אם קטן מסף מסוים אחרת יקבל ערך חדש השווה לערך הפיקסל פחות הסף.
האיור הבא מתאר את סדרת הפקודות בהתייחס לארבעת הפיקסלים הראשונים:



איור 87 תיאור גרפי של קטע הקוד לבדיקת סף עבור כל פיקסל

ראשית מבצעים טעינה של 8 פיקסלים מהזיכרון בעזרת פקודת LDDW. נניח שערך הסף (שהוא 5 בדוגמא) כבר טעון באוגר thr0. הפקודה CMPGTU4 ולאחריה פקודת XPND4 יוצרות מסיכה שבה אם הפיקסל גדול מהסף ערך בית המתאים במסכה יהיה כולו "1" (0xff=11111111) אחרת כולו אפסים. בנוסף משתמשים בפקודת SUB4 להחסיר את הסף מהפיקסל. ברור שאם ערך הפיקסל קטן מהסף נקבל תוצאה שלילית שאותה נרצה לאפס אחרת זוהי תוצאת הפרש נכונה. ע"י ביצוע פעולת AND בין תוצאת ההפרש למסכה נקבל שהמסכה אפס אם ערך הפיקסל קטן מהסף ולכן פעולת AND עם אפסים תגרום לאיפוס הערך, אחרת פעולת AND עם 0xff תשאיר את תוצאת ההפרש ללא שינוי. קטע אופייני באסמבלר למימוש פעולה זו (קטע זה פועל גם על רביעיית הפיקסלים הנמוכה וגם הגבוהה (LSB,MSB).

טען 8 פיקסלים לצמד אוגרים ; LDDW *input_ptr++, data0:data1
השווה 4 פיקסלים לסף ; CMPGTU4 data0,thr0,res0
הרחב את תוצאת הסיביות ליצירת מסכה ; XPND4 res0,mask0
בצע את פעולת החיסור ; SUB4 data0,thr0,tmp0
צור תוצאה סופית ; AND tmp0,mask0,out0
השווה 4 פיקסלים לסף ; CMPGTU4 data1,thr0,res1
הרחב את תוצאת הסיביות ליצירת מסכה ; XPND4 res1,mask1
בצע את פעולת החיסור ; SUB4 data1,thr0,tmp1
צור תוצאה סופית ; AND tmp1,mask1,out1
אחסן תוצאה סופית בזיכרון ; STDW out0:out1, *output_ptr++

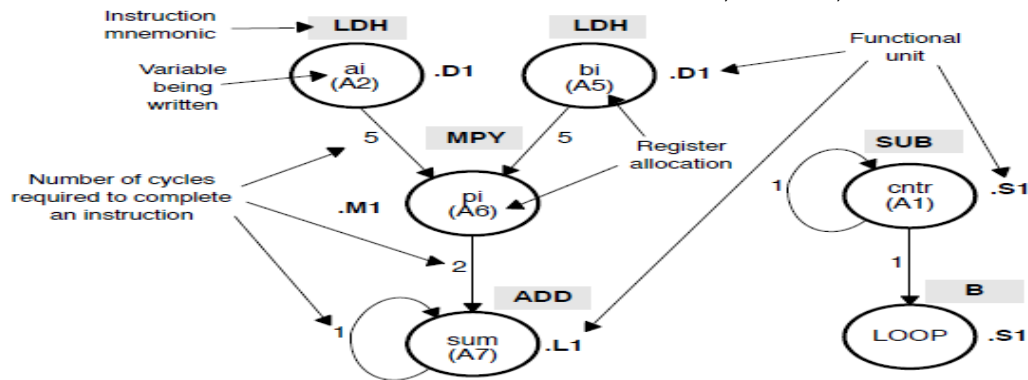
כמובן שהקוד המובא כאן הוא סדרתי וכמו שכבר הוזכר הפעולות המוצגות כאן עבור שמונה פיקסלים יכולות להתבצע במקביל כפי שנראה בסעיפים הבאים אולם כבר עתה ניתן לקבל את הרושם שקוד DSP הכתוב בצורה יעילה אינו מכיל קפיצות או הסתעפויות (למעט הקפיצה למימוש הלולאה) והינו קוד שטוח לחלוטין שמימוש התנאים נעשה בעזרת העקרונות שתוארו בסעיף זה. במבט ראשון בקוד נראה כאילו יש כאן 10 פקודות. מכיוון שהתוצאה תהיה על 8 פיקסלים אז גם אם נניח שכל פקודה מתבצעת במחזור אחד נקבל שהזמן הממוצע במחזורי מעבד עבור כל פיקסל הוא $10/8=1.25$ מחזורים לכל פיקסל. בסעיפים הקודמים ראינו שניתן לבצע פקודות במקביל ע"י שימוש בטכניקת pipeline. שימוש בטכניקה זו ועוד מספר טריקים יכול להביא אותנו לשיפור ביצועים כך שסה"כ נקבל 0.1875 מחזורי מעבד עבור כל פיקסל. למרות שמספר זה נשמע דמיוני ואפילו לא הגיוני נכונותו תתברר בסעיפים הבאים.

בניית לולאת PIPELINE יעילה

בסעיפים הקודמים התמקדנו בהבנת טכנולוגית ה- pipeline ונוכחנו לדעת שהניצולת תהיה מרבית אם כל הוראה (fetch packet) מכילה חבילת ביצוע אחת (כלומר יש 8 פקודות המיועדות ל 8 יחידות ביצוע שונות). פרמטר חשוב שמקשה על בניה יעילה של לולאה המנצלת באופן מלא את ה- pipeline הוא העובדה שלפקודות שונות יש זמן ביצוע שונה (או delay שונה) ובנוסף שלא תמיד צרכי הלולאה והמשאבים שהיא דורשת ניתנים לחלוקה טריוויאלית בין יחידות הביצוע השונות של ה- C64+. בסעיף זה ננסה להבין את הקשיים בבניית לולאה כזו ואיך ניתן להיעזר במהדר כדי לבנות לולאות יעילות בקלות רבה יותר. נתבונן בדוגמת הלולאה הבאה המבצעת הכפלה של איברים מתאימים בשני ווקטורים ומסכמת מכפלות אלו.

```
int dotp(short a[], short b[])
{
    int sum, i;
    sum = 0;
    for(i=0; i<100; i++)
        sum += a[i] * b[i];
    return(sum);
}
```

איור 88 תיאור הלולאה המסכמת מכפלות של איברים מתאימים של שני וקטורים
כאמור הווקטורים הם איברים בגודל 16 סיביות. כדי לכתוב לולאה יעילה עלינו להבין את התלויות בין הפקודות השונות. לשם כך נצייר גרף המתאר את התלויות הללו.



איור 89 תיאור גרף התלויות של לולאת dotp.

שתי פקודות ה- LDH שטוענות את המידע מהווקטורים הם המקור למידע שעליו עובדת הפקודה MPY המכפילה שני איברים אלה. לפקודה LDH לוקח 5 מחזורים להסתיים ולכן אם הפקודה LDH מתוזמנת למחזור ה- i אז הפקודה MPY לא יכולה להתבצע לפני המחזור ה- i+5. הפקודה ADD מוסיפה את pi לסכום הכולל והיא משמשת כקלט למחזור הבא ולכן יש לנו מסלול הנקרא Loop carry path באורך 7. מסלול זה נוצר כאשר מחזור אחד בלולאה כותב ערך שחייב להיקרא ע"י המחזור הבא. גרף התלויות כולל שני חלקים נפרדים מכיוון שהחסרת מונה הלולאה (cnt) והקפיצה להתחלת הלולאה לא קשורות לקריאה ועדכון של וקטורי הקלט והתוצאה. אם נרצה לכתוב קוד אסמבלר לגרף זה הוא יראה בדומה לקוד הבא:

	MVK	.S1	100, A1	; set up loop counter
	ZERO	.L1	A7	; zero out accumulator
LOOP:	LDH	.D1	*A4++, A2	; load ai from memory
	LDH	.D1	*A3++, A5	; load bi from memory
	NOP	4		; delay slots for LDH
	MPY	.M1	A2, A5, A6	; ai * bi
	NOP			; delay slot for MPY
	ADD	.L1	A6, A7, A7	; sum += (ai * bi)
	SUB	.S1	A1, 1, A1	; decrement loop counter
	[A1] B	.S2	LOOP	; branch to loop
	NOP	5		; delay slots for branch
	; Branch occurs here			

איור 90 תיאור קוד אסמבלר של לולאת dotp.

אם נסכם את כל מחזורי הביצוע נקבל שסה"כ הלולאה תהיה באורך של 16 מחזורים. ניתן לראות שבגרף התלויות יחידת הביצוע הטוענת את המידע D1 משמשת גם לטעינת ai וגם לטעינת bi. כמובן שמיותר

להעמיס על יחידה אחת את שתי הטעינות מכיוון שניתן להשתמש ביחידה D2 לצורך כך וכך לקבל לולאה יעילה יותר.

<pre> MVK .S1 100, A1 ; set up loop counter ZERO .L1 A7 ; zero out accumulator LOOP: LDH .D1 *A4++, A2 ; load ai from memory LDH .D2 *B4++, B2 ; load bi from memory SUB .S1 A1, 1, A1 ; decrement loop counter [A1] B .S2 LOOP ; branch to loop NOP 2 ; delay slots for LDH MPY .M1X A2, B2, A6 ; ai * bi NOP ; delay slots for MPY ADD .L1 A6, A7, A7 ; sum += (ai * bi) ; Branch occurs here </pre>	
איור ב	איור א

איור 91 א-ב גרף תלויות מעודכן וקוד אסמבלר מתאים

מכיוון שהטעינה של ai ו-bi לא תלויות אחת בשניה שניהם יכולות להתבצע במקביל כל עוד הן אינן חולקים אותה יחידת ביצוע. כתוצאה מכך אנו צריכים להקצות את ai לצד A של המעבד ואת bi לצד B (D2 ו-B4). כעת מכיוון שלפקודת MPY יש מקור אחד מצד A ומקור אחד מצד B היא משתמשת ב cross path 1x. בנוסף לכך סדר ההוראות מאורגן אחרת דבר שתורם לשיפור ביצועים מכיוון שפקודת ה-SUB יכולה להתבצע במקום אחד ה-NOP ששימשו ל- delay עבור פקודת LDH ובנוסף העברת פקודת ההסתעפות (פקודת B) לאחר ה-SUB מורידה את הצורך לבצע 5 פקודות NOP בסוף הלולאה. כעת הקפיצה מתרחשת מיד לאחר פקודת ה-ADD. לסיכום הלולאה הנוכחית כוללת רק 8 מחזורים לעומת 16 בלולאה הקודמת כלומר שיפור בגורם 2. שיפור נוסף שניתן לבצע מתבסס על העובדה שניתן לשפר את הקריאה של a[i], a[i+1] בעזרת פקודה לקריאה מילה (LDW (Word שתבצע קריאה של שני נתונים בגודל 16 סיביות. ניתן כמובן גם להרחיב ולקרוא 4 יחידות מידע כאלה בעזרת LDDW. ביצוע פעולה כזו נקרא loop unrolling ויש לו אפקט דומה לזה המתבצע בקוד הבא שבו עובדים בלולאה על 2 יחידות מידע.

```

int dotp(short a[], short b[])
{
    int sum0, sum1, sum, i;

    sum0 = 0;
    sum1 = 0;
    for(i=0; i<100; i+=2){
        sum0 += a[i] * b[i];
        sum1 += a[i + 1] * b[i + 1];
    }
    sum = sum0 + sum1;
    return(sum);
}

```

איור 92 דוגמא לקוד המממש את dotp בעזרת loop unrolling.

כדי להדגים את השיפור בביצועים נתבונן בגרף התלויות החדש ובקוד האסמבלר המממש אותו.

<pre> MVK .S1 50,A1 ; set up loop counter ZERO .L1 A7 ; zero out sum0 accumulator ZERO .L2 B7 ; zero out sum1 accumulator LOOP: LDW .D1 *A4++,A2 ; load ai & ai+1 from memory LDW .D2 *B4++,B2 ; load bi & bi+1 from memory SUB .S1 A1,1,A1 ; decrement loop counter [A1] B .S1 LOOP ; branch to loop NOP 2 MPY .M1X A2,B2,A6 ; ai * bi MPYH .M2X A2,B2,B6 ; ai+1 * bi+1 NOP ADD .L1 A6,A7,A7 ; sum0+= (ai * bi) ADD .L2 B6,B7,B7 ; sum1+= (ai+1 * bi+1) ; Branch occurs here ADD .L1X A7,B7,A4 ; sum = sum0 + sum1 </pre>	
איור 93-ב קוד אסמבלר	איור 93 – א גרף תלויות

איור 93 גרף התלויות והקוד עבור מימוש בעזרת loop unrolling.

אם נתבונן בלולאה נראה שהיא כוללת 8 מחזורים אבל פועלת על 2 יחידות מידע ולכן נקבל באופן ממוצע שעל כל נתון יש 4 מחזורים, כלומר שיפור של גורם 2 בהשוואה ללולאה הקודמת. למרות כל השיפורים שנעשו ללולאה עדיין הגורם המשמעותי יוכל להתקבל תוך שימוש ב- software pipeline שעקרונותיו תוארו בסעיפים הקודמים. בעזרת טכניקה זו מתזמנים את הוראת הלולאה כך שמחזורים רבים יתבצעו במקביל. מכיוון של- C64 יש יחידות ביצוע בלתי תלויות ניתן לאתחל מחזור חדש בלולאה לפני שהמחזור הקודם הסתיים. המטרה ב- software pipeline היא להתחיל מחזור לולאה חדש מוקדם ככל האפשר. הלולאה האחרונה שיצרנו דורשת כאמור 8 מחזורים, 5 לפקודת LDW, שתי מחזורים ל- MPY ואחד ל- ADD. הדרך לבנות לולאה אסמבלר יעילה הבנויה מפקודות אסמבלר המתבצעות במקביל היא לבנות טבלה שמראה איך ה- pipeline מתבצע ולעקוב אחר יחידות ביצוע פנויות מחזור אחר מחזור כדי להבטיח שלא משתמשים באותו משאב (יחידת ביצוע) פעמיים במחזור נתון כלשהו. לדוגמא אם נתבונן בלולאה dotp שבנינו נוכל לסכם את הנקודות הבאות:

- פקודות LDW מתבצעות ביחידות D במחזורים 0, 8, 16, 24 וכ"ו.
 - הפקודות MPY ו- MPYH מתבצעות ביחידות M ומתוזמנות למחזורים 5,13,21,29 וכ"ו.
 - פקודות ADD מתבצעות ביחידות L ומתוזמנות למחזורים 7,15,23,31 וכ"ו.
 - פקודת SUB מתבצעת ב- S1 ומתוזמנת למחזורים 1,9,17,25 וכ"ו.
 - פקודת B מתבצעת ב- S2 ומתוזמנת למחזורים 2,10,18,24 וכ"ו.
- הטבלה הבאה מתארת את פריסת הפעולות ללא ביצוע במקביל של הפקודות.

Unit / Cycle	0, 8, ...	1, 9, ...	2, 10, ...	3, 11, ...	4, 12, ...	5, 13, ...	6, 14, ...	7, 15, ...
.D1	LDW							
.D2	LDW							
.M1					MPY			
.M2					MPYH			
.L1							ADD	
.L2							ADD	
.S1		SUB						
.S2			B					

טבלה 14 תיאור פריסת הפקודות ללא ביצוע במקביל של פעולות

ביצוע של software pipeline יכול לשפר ביצועים במידה רבה אולם לפני ביצוע אנו צריכים לדעת מה המספר המינימלי של מחזורים שהלולאה כוללת והאם הם מספיקים למלא את ה-pipeline. בהנחה שיש מספיק מחזורים נוכל לתזמן את הלולאה בצורה הבאה:

Unit / Cycle	Loop Prolog							
	0	1	2	3	4	5	6	7, 8, 9...
.D1	LDW	*	**	***	****	*****	*****	*****
.D2	LDW	LDW	LDW	LDW	LDW	LDW	LDW	LDW
.M1						MPY	MPY	MPY
.M2						MPYH	MPYH	MPYH
.L1								ADD
.L2								ADD
.S1		SUB	SUB	SUB	SUB	SUB	SUB	SUB
.S2			B	B	B	B	B	B

טבלה 15 תיאור פריסת הפקודות תוך שימוש בביצוע מקבילי של פעולות - * מציינ מחזור

העמודה הימנית מראה שתזמון ה-pipeline הוא כדלקמן:

- הפקודה ADD מתבצעת עבור מחזור n.
- הפקודות MPY מתבצעות עבור מחזור n+2 (מסומן **).
- פקודות LDW מתבצעות עבור מחזור n+7 (מסומן *****).
- פקודות SUB מתבצעות עבור מחזור n+6 (מסומן *****).
- הפקודה B מתבצעת עבור מחזור n+5 (מסומן *****).
- צריכים לדעת מה המספר המינימלי של מחזורי מכונה.

בדוגמא הזו מספר מחזורים מתבצעים במקביל כלולאה, ממחזור n עד n+7. תיאור קוד האסמבלר יהיה כדלקמן:

```

||      LDW      .D1      *A4++,A2      ; load ai & ai+1 from memory
||      LDW      .D2      *B4++,B2      ; load bi & bi+1 from memory
||      MVK      .S1      50,A1         ; set up loop counter
||      ZERO     .L1      A7           ; zero out sum0 accumulator
||      ZERO     .L2      B7           ; zero out sum1 accumulator
||
||      [A1] SUB   .S1      A1,1,A1      ; decrement loop counter
||      LDW      .D1      *A4++,A2      ; * load ai & ai+1 from memory
||      LDW      .D2      *B4++,B2      ; * load bi & bi+1 from memory
||
||      [A1] SUB   .S1      A1,1,A1      ; * decrement loop counter
||      B        .S2      LOOP          ; branch to loop
||      LDW      .D1      *A4++,A2      ; ** load ai & ai+1 from memory
||      LDW      .D2      *B4++,B2      ; ** load bi & bi+1 from memory
||
||      [A1] SUB   .S1      A1,1,A1      ; ** decrement loop counter
||      B        .S2      LOOP          ; * branch to loop
||      LDW      .D1      *A4++,A2      ; *** load ai & ai+1 from memory
||      LDW      .D2      *B4++,B2      ; *** load bi & bi+1 from memory
||
||      [A1] SUB   .S1      A1,1,A1      ; *** decrement loop counter
||      B        .S2      LOOP          ; *** branch to loop
||      LDW      .D1      *A4++,A2      ; **** load ai & ai+1 from memory
||      LDW      .D2      *B4++,B2      ; **** load bi & bi+1 from memory
||
||      MPYH     .M1X     A2,B2,A6      ; ai * bi
||      MPYH     .M2X     A2,B2,B6      ; ai+1 * bi+1
||      [A1] SUB   .S1      A1,1,A1      ; **** decrement loop counter
||      [A1] B     .S2      LOOP          ; *** branch to loop
||      LDW      .D1      *A4++,A2      ; ***** ld ai & ai+1 from memory
||      LDW      .D2      *B4++,B2      ; ***** ld bi & bi+1 from memory
||
||      MPYH     .M1X     A2,B2,A6      ; * ai * bi
||      MPYH     .M2X     A2,B2,B6      ; * ai+1 * bi+1
||      [A1] SUB   .S1      A1,1,A1      ; ***** decrement loop counter
||      [A1] B     .S2      LOOP          ; ***** branch to loop
||      LDW      .D1      *A4++,A2      ; ***** ld ai & ai+1 fm memory
||      LDW      .D2      *B4++,B2      ; ***** ld bi & bi+1 fm memory
||
||
||      LOOP:    ADD     .L1      A6,A7,A7      ; sum0 += (ai * bi)
||      ADD     .L2      B6,B7,B7      ; sum1 += (ai+1 * bi+1)
||      MPYH     .M1X     A2,B2,A6      ; ** ai * bi
||      MPYH     .M2X     A2,B2,B6      ; ** ai+1 * bi+1
||      [A1] SUB   .S1      A1,1,A1      ; ***** decrement loop counter
||      [A1] B     .S2      LOOP          ; ***** branch to loop
||      LDW      .D1      *A4++,A2      ; ***** ld ai & ai+1 fm memory
||      LDW      .D2      *B4++,B2      ; ***** ld bi & bi+1 fm memory
||
||      ; Branch occurs here
||      ADD     .L1X     A7,B7,A4      ; sum = sum0 + sum1

```

איור 94 תיאור קוד האסמבלר היעיל ביותר עבור dotp

כאמור לולאת ה-pipeline כוללת מחזור אחד ופועל על שני נתונים כלומר סה"כ קיבלנו שכל נתון מעובד בממוצע בחצי מחזור שזה שיפור בפקטור 32 יחסית ללולאה הראשונה שבה התחלנו וכללה 16

מחזורים. בנוסף לכך כאשר נכנסים ללולאה כל הוראה כוללת 8 פקודות כלומר חבילת ביצוע אחת שתורמת לביצוע מקסימלי של ה-pipeline. בלולאה dotp הודגמו אפשרויות ייעול הקוד אולם ראינו שעבודה לא מעטה הושקעה כדי להפיק יעילות מרבית. למזלנו ניתן להגיע לאותה יעילות (או קרוב ליעילות המקסימלית האפשרית) ללא צורך לכתוב את קוד האסמבלר בצורה ידנית דבר שחוסך עבודה רבה בביצוע שיפורי ריצה. השיטות העיקריות לכך הם:

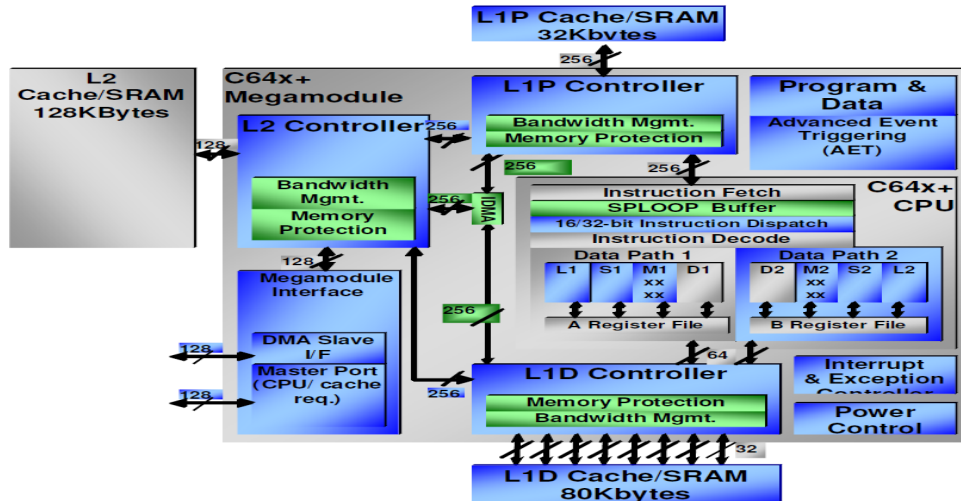
- כתיבת הקוד שימוש בפקודות intrinsic הממפות כל הוראה לפקודת אסמבלר (בשיטה זו לא ניתן למפות פקודות לצד מסוים במעבד).
- שימוש בשפת אסמבלר ליניארית (linear assembler) הכוללת את כל האפשרויות לגמישות הקיימות באסמבלר אבל ללא כתיבת הפעולות במקביל. בשתי השיטות תזמון הפקודות המקבילות יעשה ע"י המהדר כפי שנראה בהמשך.

תיאור Mega-Module והפעלת ה-EDMA

בסעיף הקודם נידונו השיטות ליעול הלולאות המבצעות את עיבוד המידע. המטרה כמובן היא לממש בהצלחה את האלגוריתמים הדרושים לגילוי הפגמים ולעמוד באילוצי הזמן שהוגדרו. באופן ממוצע כל פונקציה עיבוד תמונה שהיא חלק משלבי האלגוריתמים צריכה להתבצע בזמן של מספר מחזורים בודדים בממוצע עבור כל פיקסל (ראינו שפונקציית dotp מהסעיף הקודם התבצעה ב-0.5 cycle עבור כל פיקסל) שאם לא כן לא נעמוד באילוצי זמן האמת של המערכת. כדי להבטיח שלולאת קוד שמתבצעת במשך מספר מחזורים בודדים עבור כל פיקסל אכן תתבצע בזמן זה עלינו לדאוג לכך שצוואר הבקבוק בין הזיכרון ל- C64+ לא יפריע. בהנחה שהבלוקים אותם מעבד כל DSP מגיעים לזיכרון ה-DDR2 (בתהליך רכישת התמונה שתואר בסעיפים הקודמים) הפעלת הלולאות ישירות על הזיכרון החיצוני תגרום לירידה משמעותית בביצועים מכיוון שגישה לזיכרון החיצוני מתבצעת בעשרות עד מאה או מאתיים מחזורים של המעבד. בזמן זה ה-C64+ נמצא במצב stall ומחכה לנתונים שיגיעו מהזיכרון החיצוני. שיטה אפשרית להתמודדות עם בעיה זו היא לאפשר לאזורי הזיכרון הרלוונטיים להיות ממופים לזיכרון L2 ו-L1 של המעבד כלומר לאפשר לזיכרונות אלו להיות בני cache או cacheable. כאמור ל-MegaModule ישנם זיכרונות פנימיים שניתן לעבוד איתם כ-static Ram או כ-cache memory. זמן גישה לזיכרון L1 (cache או SRAM) הוא מחזור מעבד אחד לעומת זאת זמן גישה לזיכרון L2 הוא 10 מחזורים. אפשרות ראשונה כאמור להקצות את זיכרון L1 כך שיכיל את גודל ה-cache המקסימאלי האפשרי (32Kbytes מתוך 80Kbytes הקיימים) ואת כל הזיכרון L2 להקצות כזיכרון cache ולהניח לבקרי ה-Cache controller של ה-Mega Module לטפל בהכנסת המידע מהזיכרון החיצוני לזיכרונות הפנימיים. שיטה זו תשפר לאין ערוך את הביצועים אבל מכיוון שעבור כל מידע הנמצא בשורת cache שלא ניגשנו אליו נשלם מחיר יקר של גישה לזיכרון החיצוני, בהבאת המידע לרמת L2 ולאחריה ל-L1. בהנחה שזמן גישה עבור מידע שלא נמצא בזיכרון החיצוני הוא 100 מחזורים וזמן גישה ל-L2 הוא 10 מחזורים נקבל שעבור כל פיקסל בממוצע נצטרך להמתין

$$\frac{100}{L2cacheLineSize} + \frac{10}{L1CacheLineSize} = \frac{100}{32} + \frac{10}{64} = 3.285 \text{ cycle/pixel}$$

לשם השוואה אם נבדוק את הלולאה הקודמת נגלה שלמרות שזמן העיבוד לוקח 0.5 מחזורים לכל פיקסל במידה ונעבוד בשיטה זו נהיה מוגבלים בזמן הגישה לזיכרון החיצוני עם מנגנון ה-cache שהוא 3.285 cycle/pixel. יש לציין שההנחה שזמן גישה לזיכרון החיצוני לוקחת 100 מחזורים משתנה בין סוגי הזיכרון וגם בעדיפויות שאותם ניתן לקבוע ב-Mega-Module הכולל גם מתגים לקביעת העדיפויות בין חלקיו השונים. כלומר המחיר של עבודה למול זיכרון חיצוני בניהול ה-cache עלול להיות יקר יותר. מכיוון שאילוצי זמן האמת הינם חמורים לא ניתן לכתוב פונקציות שבהם העלות של הכנסת הפיקסלים לזיכרון הפנימי של המעבד היא בסדר גודל של 3 cycle/pixel עבור כל קלט. מצב זה יגרום לכך שנהיה מוגבלי קלט/פלט (IO bounded) ופונקציות שלהן יש מספר תמונת קלט ותמונת פלט אחת יתבצעו במשך 10-20 מחזורים לפיקסל במצב שהם IO bounded.



איור 95 איור עקרוני של ה-Mega-Module.

כדי להתגבר על המחסום שתואר ולעמוד בדרישות זמן האמת של המערכת ניתן להשתמש בעובדות הבאות:

- זיכרונות ה-cache (גם L1 וגם L2) ניתנים לחלוקה כך שחלק מהגודל של כל אחד מהם יישמש כזיכרון Cache וחלק אחר כזיכרון static-ram.
- ה-Mega-Module כולל ממשק לבקר EDMA שאחד בתפקידיו הוא לטפל ברכישת התמונה (כפי שתואר במודל למערכת עיבוד התמונה שהוגדר בפרק 3) וניתן לנצל להעתיק ברקע את המידע הרלוונטי לתוך הזיכרון L1 של המעבד בצורה יעילה בהרבה מזו שאותה מנהל ה-cache controller.

הרעיון העיקרי יהיה לנצל את העובדה שכאשר מעתיקים מספר בתים בסדר גודל של קילו בתים מהזיכרון החיצוני לזיכרון הפנימי זוהי ההעתקה היעילה ביותר שניתן לבצע מזיכרון ה-DDR2. כאשר ה-EDMA מעתיק כמות גדולה של בתים הממשק בין ה-EDMA ל-DDR2 נקרא burst transfer ולמעט הבתים הראשונים כל הבתים האחרים מועתקים בקצב שהוא:

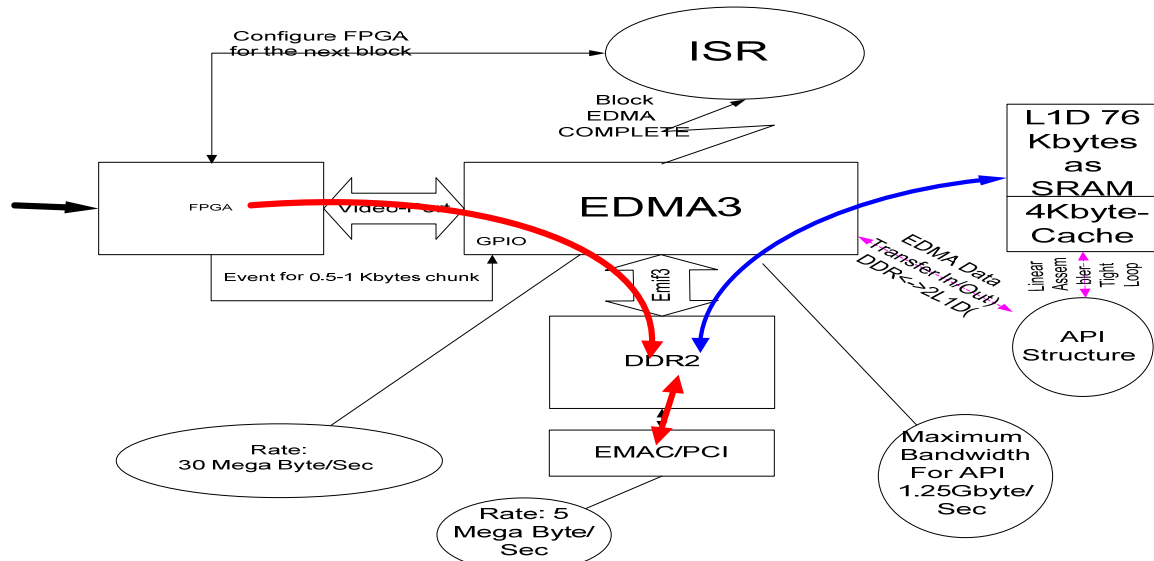
$$DDR2_Clock * 2 * Bus_Width = 166Mhz * 2 * 3 = 1328MegaByte/second$$

אם נגרמל את הקצב המקסימלי שניתן להגיע בעזרת ה-EDMA לפי החישוב הבא:

$$\frac{DSP_Clock_Rate}{DDR2ToDSPBandWidth} = \frac{600Mhz}{1328Mhz} = 0.45 \frac{Cycle}{Pixel}$$

לעיבוד ל DSP (או לבצע את הפעולה ההפוכה של שליחתו מתוך ה-DSP לזיכרון החיצוני). כמובן שקצב IO (input/output) של 0.45 cycle/pixel הינו טוב לעין שיעור לעומת קצב של 3.28 cycle/pixel אבל היתרון המשמעותי ביותר הוא היכולת של ה-EDMA לעשות את ההעברות של המידע ל DSP וממנו במקביל לפעולת העיבוד. בצורה זו ניתן לחפוף בין ההעברות לבין העיבוד בצורה שתוארה בפסידו קוד [שבאיור 13](#). בצורה זו בזמן שמעבדים חלק אחד של המידע מביאים ברקע את החלק הבא. מכיוון שרוב העבודה בפונקציות עיבוד תמונה הינו מקומי נוכל להשתמש בחלק הארי של זיכרון L1 כזיכרון סטטי (נקצה לכך 76Kbyte מתוך ה-80Kbyte הקיימים) ואילו וממנו לבצע העתקות EDMA. כך לדוגמא אם זמן העיבוד של פונקציה שיש לה תמונת קלט ותמונת פלט הוא 1.5 cy/px אז קצב ה-IO יהיה 0.9cy/px ולכן זמן העיבוד הכולל יהיה 1.5 cy/px כי זוהי פונקציה המוגבלת בזמן העיבוד (processing bound). במקרה זה העלות של העברת הפיקסלים היא אפס ומקבלים ביצועים מקסימליים. עלינו לזכור שבמודל שהגדרנו כולל טיפול גם ברכישת התמונה וגם בהעברות של פגמים על PCI או Ethernet למחשב מרכזי.

האיור הבא מתאר את קשרי הגומלין ורחבי הפס הדרושים שהוגדרו במודל.



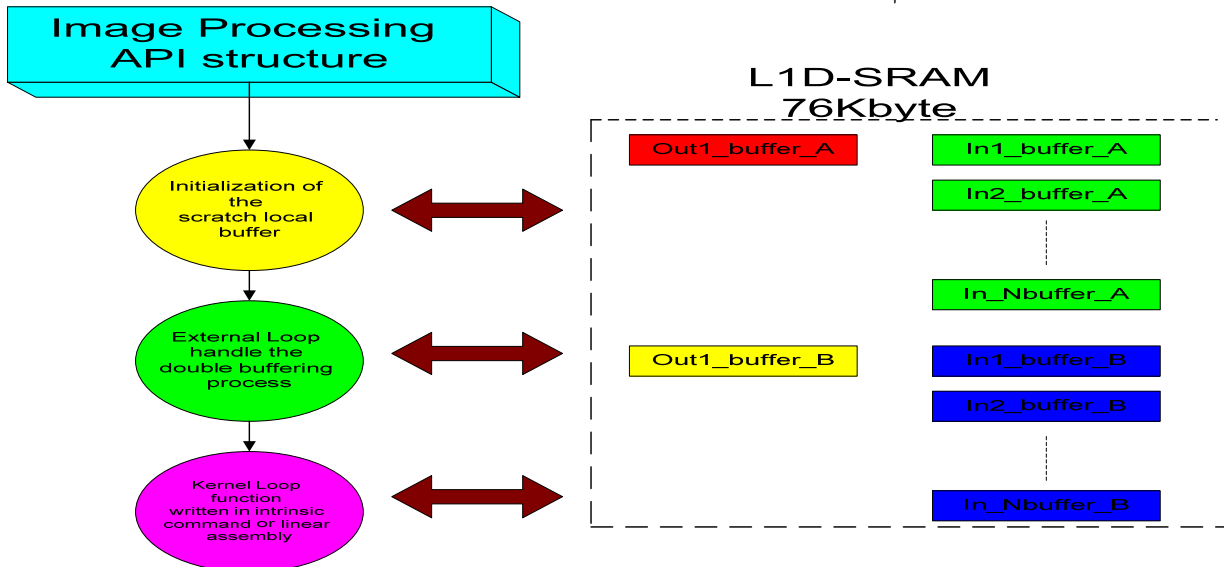
איור 96 תיאור קשרי הגומלין ורחבי הפס לפי המודל שהוגדר.

כאמור כדי לבצע את רכישת התמונה נצטרך רוחב פס של סדר גודל של 30Mbyte/sec. בנוסף ישנם העברות PCI/Ethernet בקצב יחסית איטי של 5MByte/sec. קצבים אלה ינגסו מהקצב המרבי שניתן להעביר מה- DDR2 ואילו וסה"כ נקבל קצב של סדר גודל של 1.25 Gigabyte/sec שהוא מעט פחות מזה שהוצג לעיל ולכן הקצב הנומינאלי עבור כל API (פונקציות עיבוד תמונה) לביצוע העברות הוא 0.48 cycle/pixel. האפשרות להגיע לקצבים אלה תלויה רבות באפשרויות לתת את העדיפות המתאימה לפונקציות עיבוד התמונה ועם זאת לא לפגוע בתהליך רכישת התמונה שלמרות שהוא בקצב נמוך הוא בעל עדיפות גבוהה. כדי להשיג מטרה זו נשתמש בעובדה שבתוך ה- EDMA ניתן לנתב העתקות לתורים עם עדיפות שונה עם FIFOs בעלי גודל שונה ולהתאימם לפי הצורך. כמתואר באיור הבא. מכיוון שלמעט ה- EDMA וה- PCI/Ethernet שבהם קצב ההעברה נמוך מאוד לא צפויה בעיה בחלוקת רוחב הפס עדיף להשאיר את ה- Mega Module בהגדרות ברירות המחדל שלו. כאמור ל- Mega Module יש גם זיכרון L1P המשמש כ- program cache בצורת העבודה שלנו כל הזיכרון L2 משמש כזיכרון cache שיתמוך גם בקוד וגם בנתונים. כדי שנתונים ומשתנים שאליהם ניגשות הפונקציות (בנתונים אין הכוונה לפיקסלים שעוברים עיבוד אלא משתנים כלליים המשמשים במהלך ניהול הקוד לדוגמא בהפעלת ה- EDMA ב- double buffering). כדי שנתונים אלו (הקוד יכנס ל- L1P) ימצאו גם הם ב- L1 נקצה 4Kbyte מתוך ה- 80Kbyte של L1 כדי שישמשו כזיכרון L1D cache.

מימוש פונקציות עיבוד תמונה אופייניות

בסעיפים הקודמים תוארו צורת העבודה והאופן העקרוני של מימוש פונקציות עיבוד תמונה המשמשות כאבני בניין לאלגוריתמים המוצאים פגמים בפרוסות השבבים. כאמור מימוש יעיל של לולאת עיבוד המידע (הנקראת גם tight loop) נעשה ע"י כתיבת קוד אסמבלר אך בדרך כלל נעדיף קוד המבוסס על intrinsic או linear assembler. לאחר שנכתבה הלולאה נבחן את קוד האסמבלר ואת המשוב שמספק המהדר על יעילות הביצוע ונסה לשפר אותו במידת הצורך. יהיו מקרים שהפונקציה היא IO-bounded לדוגמא אם יש לנו פונקציה שמקבלת 4 תמונות קלט ומוציאה תמונת פלט אחת ברור לפי החישובים שהוצגו בסעיף הקודם ההעברות פנימה והחוצה יצרכו $0.48 * 5 = 2.4$ cy/px ולכן גם אם נממש לולאה שתתקח פחות מכך לא נרוויח בהרבה. בנוסף לכך אם למשל יש לנו תמונת קלט אבל אנו צריכים לעבוד על 2 חלקים שלה למשל על החצי הראשון למול החצי השני. מכיוון שכל מידע חייב להיות מועתק לתוך הזיכרונות הפנימיים נאלץ להתייחס לתמונה זו כאל שתי תמונות ולהעתיק שורות מכל אזור כאילו היו מתמונות שונות על מנת שבזיכרון הפנימי של ה- DSP ימצא המידע המתאים לעיבוד. באופן כללי פונקציות עיבוד תמונה תכיל מודולים כדלקמן:

- חלק המבצע אתחול משתנים ומגדיר את שטחי הזיכרון הדרושים עבור כל תמונה קלט או פלט בזיכרון L1D-SRAM. עבור כל קלט או פלט יוקצה מקום בגודל מספר השורות שמעבדים (בד"כ בין 20 ל-30 שורות כפול רוחב התמונה).
- חלק שני מתפעל את ה-EDMA ומבצע העתקות בטכניקת double buffering וקורא ללולאת העיבוד המתאימה על מידע הנמצא בזיכרון הפנימי של המעבד (L1D).
- חלק שלישי הוא הלולאה האופטימלית הכתובה ב- intrinsic או linear assembler. האיור הבא מתאר את מבנה פונקציית עיבוד התמונה כפי שתואר לעיל.



איור 97 תיאור המרכיבים העיקריים של פונקציית עיבוד תמונה

כדי לממש את הלולאות האופטימליות שהם המרכיב העיקרי בדרך להשגת ביצועים מקסימליים דרוש לכתוב קוד בצורה אחרת מהצורה האלגוריתמית שבה הוא מוגדר. רצוי שהקוד יהיה שטוח כמה שאפשר ללא פקודות if רגילות (אלא בטכניקות שהודגמו בסעיפים הקודמים) ופעמים רבות דרוש להיות יצירתי מספיק כדי לנצל את הפקודות המיוחדות של ה-DSP בצורה היעילה ביותר המתאימה למימוש התוכן. בנוסף רצוי שהלולאה לא תהיה ארוכה מדי שכן אז יהיה קשה לממש pipeline בגלל חוסר באוגרים וביחידות ביצוע.

דוגמא תיאור הקוד המנהל את העתקות ה-EDMA

נניח שאנו מעונינים לממש קוד המנהל העתקת EDMA עבור פונקציה שיש לה קלט אחד ופלט אחד. לדוגמא פונקציה העוברת עם מסנן 3x3 על תמונה.

קטע הקוד הבא מממש את האמור לעיל ומשמש כמימוש לפסידו קוד שתואר באיור 37.

```
// pointers to buffer in internal L1D SRAM
extern unsigned char *pSmoothInInternal_A,
                    *pSmoothOutInternal_A,
                    *pSmoothInInternal_B,
                    *pSmoothOutInternal_B;

// Convolution coefficients
extern unsigned char GlobSmoothingCoef[9];
void DoEdmaTransferForSmooth(// Variables for input DMA,
    unsigned char *InputImagePtr,
    short InputImagePitch,
    // Variables for Output DMA
    unsigned char *OutputImagePtr ,
    short OutputImagePitch,
    // window that is processed
    short AlignedInputImageWidth,
    short InputImageLength,
```

```
// Offset for smooth operator
short InputImageOffset_X,
int NumRowsToProcess
)
{
// pointer for double buffer manipulation
unsigned char *pInputImagePtr, *pOutputImagePtr;
unsigned char *pSmoothInputInternal[2];
unsigned char *pSmoothOutputInternal[2];
unsigned short isEvenSet = 0;
unsigned short NextNumRowsToProcess;
unsigned short CurNumRowsToProcess;
int RowCounter;
// structure for EDMA handling
EDMA_Config EDMA0_Config, EDMA1_Config;
// Initialize pointers for external access
pInputImagePtr = InputImagePtr ;
pOutputImagePtr = OutputImagePtr ;
// Initialize pointers for internal access
pSmoothInputInternal[0] = pSmoothInInternal_A;
pSmoothInputInternal[1] = pSmoothInInternal_B;
pSmoothOutputInternal[0] = pSmoothOutInternal_A;
pSmoothOutputInternal[1] = pSmoothOutInternal_B;
NextNumRowsToProcess = NumRowsToProcess;
// Get first GL data
// Init Option as One dimensional Copy
EDMA0_Config.opt = (ONE_DIM_COPY | H_GENERAL_EDMA_TCINT_TCCM_TCC0) ;
// The num of rows to copy will be +2 for 3x3 calculation
EDMA0_Config.cnt = (NumRowsToProcess + 2)*InputImagePitch;
// input pointer
EDMA0_Config.src = pInputImagePtr;
// pointer in local memory (L1D SRAM)
EDMA0_Config.dst = pSmoothInputInternal[isEvenSet];
// load EDMA channel with it's parameter
EDMA_config(h_general_edma, &EDMA0_Config);
// Start EDMA transfer for input
EDMA_setChannel(h_general_edma);
// Advanced pointer for next portion of rows
pInputImagePtr += NumRowsToProcess*InputImagePitch;
CurNumRowsToProcess = NextNumRowsToProcess;
// wait for first EDMA to be finish
// wait for transfer completion
while(!EDMA_intTest(H_GENERAL_EDMA_TCC0));
// clear EDMA polling flag
EDMA_intClear(H_GENERAL_EDMA_TCC0);
// Start the loop for double buffering implementation
for (RowCounter=0; RowCounter<InputImageLength; RowCounter+=CurNumRowsToProcess)
{
isEvenSet = 1 - isEvenSet;
CurNumRowsToProcess = NextNumRowsToProcess;
if(RowCounter+CurNumRowsToProcess+NextNumRowsToProcess > InputImageLength)
{
NextNumRowsToProcess -=
RowCounter+CurNumRowsToProcess+NextNumRowsToProcess-InputImageLength);
}
// Do not transfer in raw data on the last iteration
if ((NextNumRowsToProcess>0) && (RowCounter+CurNumRowsToProcess <
InputImageLength))
{
// Start transfer first lines from external memory to internal using EDMA

```

אתחול העתקה ראשונה
מעתיקים עוד 2 שורות
כי מבצעים 3x3

התחל העתקה

חכה לסיום העתקה
ראשונה

החלף buffers

```
EDMA0_Config.opt= (ONE_DIM_COPY) | H_GENERAL_EDMA_TCINT_TCCM_TCC0) ;
EDMA0_Config.cnt= (NextNumRowsToProcess + 2)*InputImagePitch ;
EDMA0_Config.src= pInputImagePtr;
EDMA0_Config.dst= pSmoothInputInternal[isEvenSet];
EDMA_config(h_general_edma, &EDMA0_Config);
EDMA_setChannel(h_general_edma);
pInputImagePtr += NextNumRowsToProcess*InputImagePitch;
}
// process kernel function
conv3x3_2D((char *) (pSmoothInputInternal[1-isEvenSet]+InputImageOffset_X),
            (char *) (pSmoothOutputInternal[1-isEvenSet]),
            AlignedInputImageWidth,
            InputImagePitch,
            GlobSmoothingCoef,
            CurNumRowsToProcess,
            OutputImagePitch);
// wait for EDMA input EDMA to be finish
if( NextNumRowsToProcess > 0 )
{
    while(!EDMA_intTest(H_GENERAL_EDMA_TCC0));
    EDMA_intClear(H_GENERAL_EDMA_TCC0);
}
// wait for previous Output EDMA to be finish (skip the first one)
if( RowCounter > 0)
{
    while(!EDMA_intTest(H_GENERAL_EDMA_TCC1));
    EDMA_intClear(H_GENERAL_EDMA_TCC1);
}
// get out the current output lines
EDMA1_Config.opt=(ONE_DIM_COPY|H_GENERAL_EDMA_TCINT_TCCM_TCC1),
EDMA1_Config.cnt=(CurNumRowsToProcess*OutputImagePitch);
EDMA1_Config.src=pSmoothOutputInternal[1-isEvenSet];
EDMA1_Config.dst=pOutputImagePtr;
EDMA_config(h_general_edma1, &EDMA1_Config);
EDMA_setChannel(h_general_edma1);
pOutputImagePtr += CurNumRowsToProcess*OutputImagePitch;
}
// wait for last out
while(!EDMA_intTest(H_GENERAL_EDMA_TCC1));
EDMA_intClear(H_GENERAL_EDMA_TCC1);
}
```

בצע העתקה הבאה

בצע עיבוד בעזרת
הלולאה האופטימאלית

חכה לסיום ההעתקה
של המחזור הבא
שנעשתה ברקע לעיבוד
שהסתיים

חכה לסיום ההעתקה
של הפלט מהאיטרציה
הקודמת

בצע העתקה החוצה של
הפלט הנוכחי
החוצה

מימוש מסנן 3x3

כאמור נניח שנרצה להעביר מסנן ספרתי בגודל 3x3 על תמונה. מקדמי המסנן בגודל בית אחד ומייצגים מספר חיובי. חישוב תיאורטי על הביצועים המקסימליים יראה כי לכל פיקסל דרושים 9 מכפלות ומכיוון שניתן לבצע 8 מכפלות במחזור מכונה אחד נובע שלכל היותר נוכל לבנות לולאה שבה נקבל 9/8 cycle/pixel. חישוב זה מתבסס על ההנחה שהיחידה שמגבילה אותנו היא מספר המכפלות שניתן לבצע. הקוד הנאיבי הבא בשפת C מתאר את הדרוש לביצוע הלולאה:

```
void conv3x3(const unsigned char *restrict inptr,
             unsigned char *restrict outptr,
             int x_dim,
             const char *restrict mask,
             int shift)
{
    const unsigned char *IN1, *IN2, *IN3;
    unsigned char *OUT;
    short pix10, pix20, pix30;
    short mask10, mask20, mask30;
    int sum, sum00, sum11;
```

```

int      i;
int      sum22, j;
IN1      =  inptr;
IN2      =  IN1 + x_dim;
IN3      =  IN2 + x_dim;
OUT      =  outptr;
for (j = 0; j < x_dim ; j++)
{
    sum = 0;
    for (i = 0; i < 3; i++)
    {
        pix10 =  IN1[i];
        pix20 =  IN2[i];
        pix30 =  IN3[i];
        mask10 =  mask[i];
        mask20 =  mask[i + 3];
        mask30 =  mask[i + 6];
        sum00 =  pix10 * mask10;
        sum11 =  pix20 * mask20;
        sum22 =  pix30 * mask30;
        sum +=  sum00 + sum11+ sum22;
    }
    IN1++;
    IN2++;
    IN3++;
    sum = (sum >> 8);
    if ( sum < 0 )      sum = 0;
    if ( sum > 255 )    sum = 255;
    *OUT++ =          sum;
}
}

```

כדי לכתוב קוד זה ביעילות המקסימלית נבצע את הפעולות הבאות:
נטען 3 שמיניות של פיקסלים כך שהראשונה מכילה את הפיקסלים 0-7 השנייה את 8-1 והשלישית את 9-2 בצורה הבאה:

0	1	2	3	4	5	6	7		Line-0
1	2	3	4	5	6	7	8		Line-0 shift by 1
2	3	4	5	6	7	8	9		Line-0 shift by 2

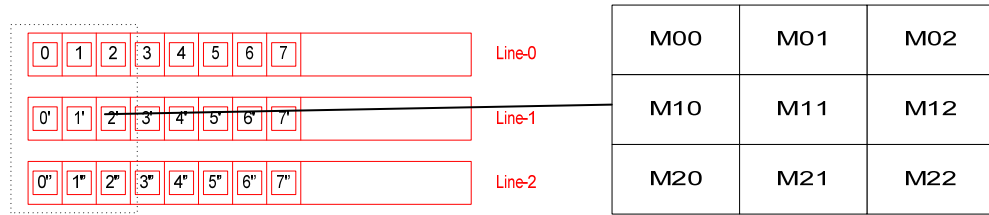
כך נעשה עבור השורה השנייה:

0'	1'	2'	3'	4'	5'	6'	7'		Line-1
1'	2'	3'	4'	5'	6'	7'	8'		Line-1 shift by 1
2'	3'	4'	5'	6'	7'	8'	9'		Line-1 shift by 2

ועבור השורה השלישית:

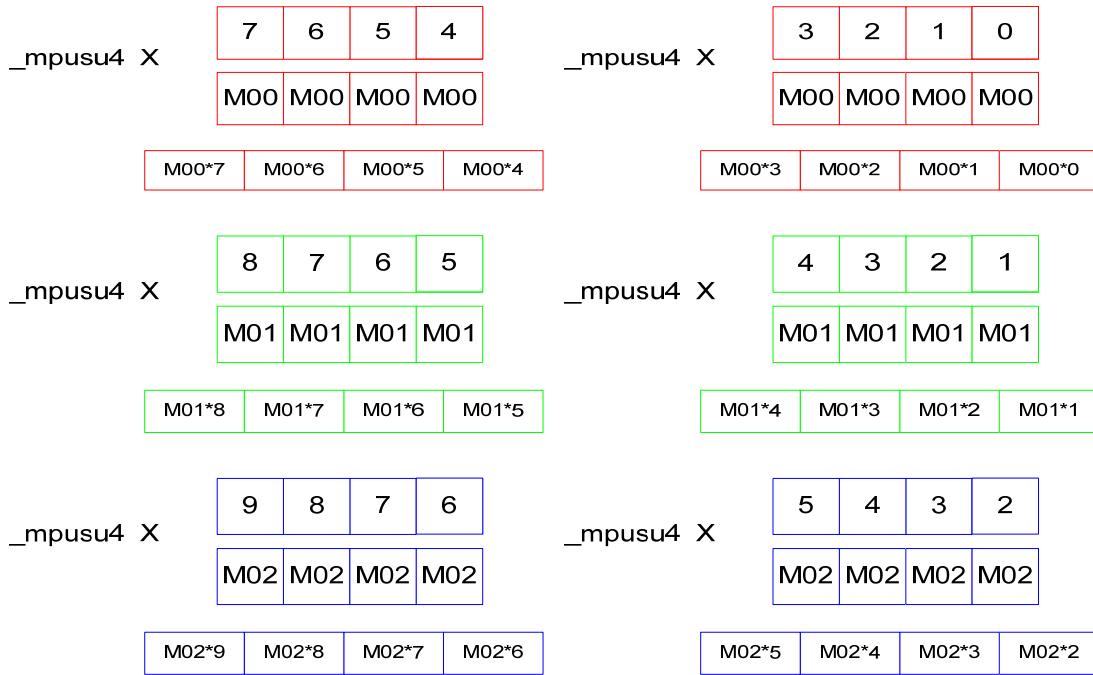
0''	1''	2''	3''	4''	5''	6''	7''		Line-2
1''	2''	3''	4''	5''	6''	7''	8''		Line-2 shift by 1
2''	3''	4''	5''	6''	7''	8''	9''		Line-2 shift by 2

עבור כל פיקסל בתמונה עלינו להעביר את המסנן הבא:

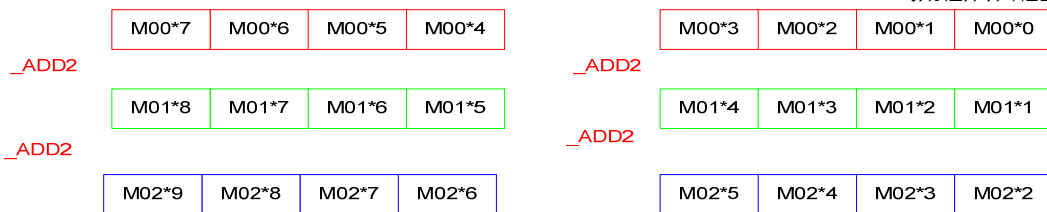


על שורות התמונה הבאות:

נתבונן בתהליך עבור שורה – 0 בלבד. לאחר שטענו כל שמיניה ואת ההיטסים שלה לתוך אוגרים נבצע עתה עבור כל רבעיה של פיקסלים נבצע את הפעולות הבאות:



התוצאות שקיבלנו בשלב זה הם בגודל של 16 סיביות. כעת נשתמש בפקודה ADD2 לסכימת המכפלות בצורה הבאה:



$M00*3 + M01*4 + M02*5$	$M00*2 + M01*3 + M02*4$	$M00*1 + M01*2 + M02*3$	$M00*0 + M01*1 + M02*2$
-------------------------	-------------------------	-------------------------	-------------------------

$M00*7 + M01*8 + M02*5$	$M00*6 + M01*7 + M02*8$	$M00*5 + M01*6 + M02*7$	$M00*4 + M01*5 + M02*6$
-------------------------	-------------------------	-------------------------	-------------------------

התוצאה שקיבלנו נכונה עבור העברת מסנן של 1x3 המתואר למטה.

M00	M01	M02
-----	-----	-----

כדי לקבל את התוצאה של מסנן 3×3 נבצע את כל האמור לעיל עבור שורה שנייה עם המקדמים M_{10}, M_{11}, M_{12} ועבור שורה שלישית עם המקדמים M_{20}, M_{21}, M_{22} .
עבור שורה שנייה נקבל:

$M_{10} \cdot 3' + M_{11} \cdot 4' + M_{12} \cdot 5'$	$M_{10} \cdot 2' + M_{11} \cdot 3' + M_{12} \cdot 4'$	$M_{10} \cdot 1' + M_{11} \cdot 2' + M_{12} \cdot 3'$	$M_{10} \cdot 0' + M_{11} \cdot 1' + M_{12} \cdot 2'$
---	---	---	---

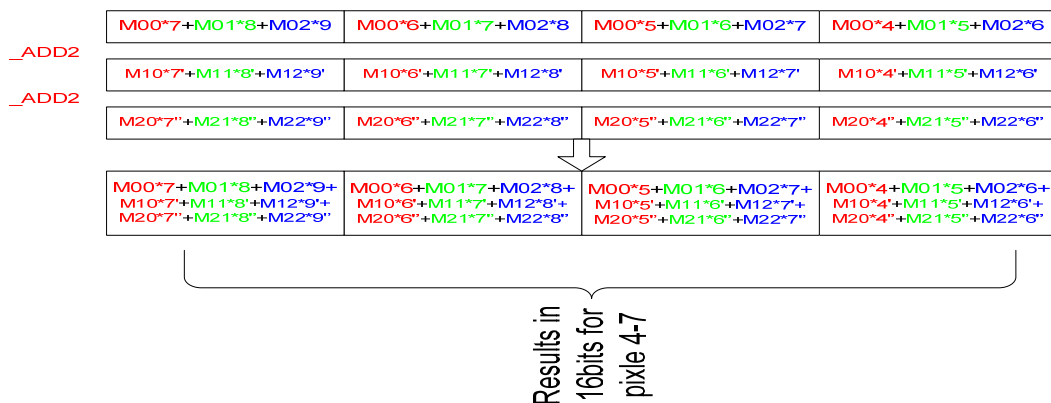
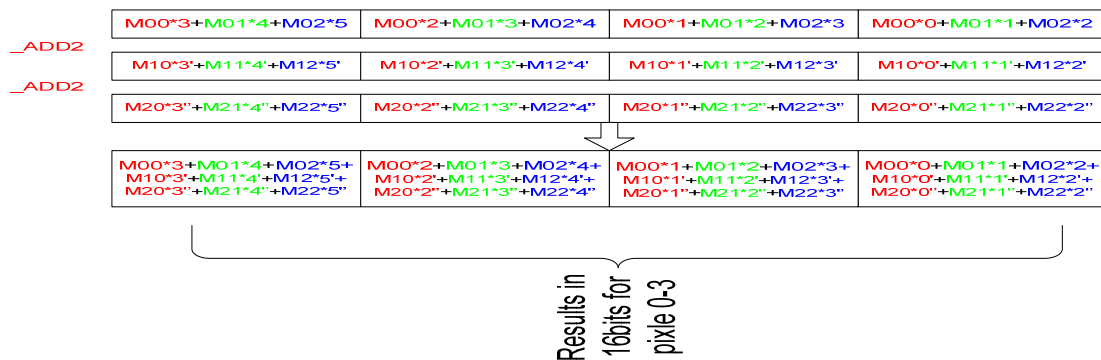
$M_{10} \cdot 7' + M_{11} \cdot 8' + M_{12} \cdot 9'$	$M_{10} \cdot 6' + M_{11} \cdot 7' + M_{12} \cdot 8'$	$M_{10} \cdot 5' + M_{11} \cdot 6' + M_{12} \cdot 7'$	$M_{10} \cdot 4' + M_{11} \cdot 5' + M_{12} \cdot 6'$
---	---	---	---

ועבור שורה שלישית נקבל:

$M_{20} \cdot 3'' + M_{21} \cdot 4'' + M_{22} \cdot 5''$	$M_{20} \cdot 2'' + M_{21} \cdot 3'' + M_{22} \cdot 4''$	$M_{20} \cdot 1'' + M_{21} \cdot 2'' + M_{22} \cdot 3''$	$M_{20} \cdot 0'' + M_{21} \cdot 1'' + M_{22} \cdot 2''$
--	--	--	--

$M_{20} \cdot 7'' + M_{21} \cdot 8'' + M_{22} \cdot 9''$	$M_{20} \cdot 6'' + M_{21} \cdot 7'' + M_{22} \cdot 8''$	$M_{20} \cdot 5'' + M_{21} \cdot 6'' + M_{22} \cdot 7''$	$M_{20} \cdot 4'' + M_{21} \cdot 5'' + M_{22} \cdot 6''$
--	--	--	--

עתה כל מה שנותר לנו הוא לסכום רבעיות של נתונים בגודל 16 סיביות (את אלו השייכים ל 0-3 ואת אלו השייכים ל 4-7) בצורה הבאה:



כעת ניתן לבצע ע"י $SHR2$ נרמול במידה והדבר דרוש ולקחת בעזרת פקודות $PACKL4$ את הבית הגבוה או הנמוך מכל תוצאה של 16 סיביות כדי לקבל תוצאה סופית של 8 בתים.
להלן הפונקציה הנ"ל (כמובן רק לולאת העיבוד) הכתובה ב- `intrinsic`.

```
void Conv3x3_kernel(
    unsigned char * restrict InputImage,
    unsigned int  InputImagePitch,
    unsigned int  InputImageLength,
    unsigned int  InputImageWidth,
    unsigned char * restrict OutputImage,
    unsigned int  OutputImagePitch,
```

```
        unsigned char *coff
    )
{
    Variable devleration
    .

for(rows=0;rows<InputImageLength ;rows++)
{
    p2_in_0 = InputImage + rows*InputImagePitch;
    p2_in_1 = p2_in_0 + InputImagePitch;
    p2_in_2 = p2_in_1 + InputImagePitch;
    p2_out = OutputImage + rows*OutputImagePitch;
#pragma MUST_ITERATE (8,192);
for(col=0;col<InputImageWidth;col+=8)
{
    // First read out of the loop
    // get better result
    // ----- line 0 >>>>
    line00line01 = _memd8(p2_in_0++);
    line02line03 = _memd8(p2_in_0++);
    line04line05 = _memd8(p2_in_0);
    p2_in_0 +=6;
    // ----- line 1 >>>>
    line10line11 = _memd8(p2_in_1++);
    line12line13 = _memd8(p2_in_1++); // because of shifts
    line14line15 = _memd8(p2_in_1);
    p2_in_1 +=6;
    // ----- line 2 >>>>
    line20line21 = _memd8(p2_in_2++);
    line22line23 = _memd8(p2_in_2++);
    line24line25 = _memd8(p2_in_2);
    p2_in_2 +=6;
    line00 = _lo(line00line01);
    line01 = _hi(line00line01);
    line02 = _lo(line02line03);
    line03 = _hi(line02line03);
    line04 = _lo(line04line05);
    line05 = _hi(line04line05);
    line10 = _lo(line10line11);
    line11 = _hi(line10line11);
    line12 = _lo(line12line13);
    line13 = _hi(line12line13);
    line14 = _lo(line14line15);
    line15 = _hi(line14line15);
    line20 = _lo(line20line21);
    line21 = _hi(line20line21);
    line22 = _lo(line22line23);
    line23 = _hi(line22line23);
    line24 = _lo(line24line25);
    line25 = _hi(line24line25);
    prodA00prodA01 = _mpysu4 (m00, line00);
    prodA02prodA03 = _mpysu4 (m00, line01);
    prodA04prodA05 = _mpysu4 (m01, line02);
    prodA06prodA07 = _mpysu4 (m01, line03);
    prodA08prodA09 = _mpysu4 (m02, line04);
    prodA10prodA11 = _mpysu4 (m02, line05);
    prodB00prodB01 = _mpysu4 (m10, line10);
    prodB02prodB03 = _mpysu4 (m10, line11);
    prodB04prodB05 = _mpysu4 (m11, line12);
    prodB06prodB07 = _mpysu4 (m11, line13);
    prodB08prodB09 = _mpysu4 (m12, line14);
    prodB10prodB11 = _mpysu4 (m12, line15);
    prodC00prodC01 = _mpysu4 (m20, line20);
    prodC02prodC03 = _mpysu4 (m20, line21);
    prodC04prodC05 = _mpysu4 (m21, line22);
    prodC06prodC07 = _mpysu4 (m21, line23);
    prodC08prodC09 = _mpysu4 (m22, line24);
    prodC10prodC11 = _mpysu4 (m22, line25);
    pix0_1_sum0 = _add2 (_lo(prodA00prodA01), _lo(prodA04prodA05));
```

```

pix0_1_sum1 = _add2 ( _add2 ( _lo(prodA08prodA09),
                             _lo(prodB00prodB01)), _lo(prodB04prodB05));
pix0_1_sum2 = _add2 ( _lo(prodB08prodB09), _lo(prodB00prodC01));
pix0_1_sum3 = _add2 ( _lo(prodB04prodC05), _lo(prodB08prodC09));
pixel0001 = _add2(_add2 (pix0_1_sum0, pix0_1_sum1), _add2
                  (pix0_1_sum2, pix0_1_sum3));
pix2_3_sum0 = _add2 ( _hi(prodA00prodA01), _hi(prodA04prodA05));
pix2_3_sum1 = _add2 ( _add2 ( _hi(prodA08prodA09),
                             _hi(prodB00prodB01)), _hi(prodB04prodB05));
pix2_3_sum2 = _add2 ( _hi(prodB08prodB09), _hi(prodB00prodC01));
pix2_3_sum3 = _add2 ( _hi(prodB04prodC05), _hi(prodB08prodC09));
pixel0203 = _add2(_add2 (pix2_3_sum0, pix2_3_sum1), _add2
                  (pix2_3_sum2, pix2_3_sum3));
pix4_5_sum0 = _add2 ( _lo(prodA02prodA03), _lo(prodA06prodA07));
pix4_5_sum1 = _add2 ( _add2 ( _lo(prodA10prodA11),
                             _lo(prodB02prodB03)), _lo(prodB06prodB07));
pix4_5_sum2 = _add2 ( _lo(prodB10prodB11), _lo(prodB02prodC03));
pix4_5_sum3 = _add2 ( _lo(prodB06prodC07), _lo(prodB10prodC11));
pixel0405 = _add2(_add2 (pix4_5_sum0, pix4_5_sum1), _add2
                  (pix4_5_sum2, pix4_5_sum3));
pix6_7_sum0 = _add2 ( _hi(prodA02prodA03), _hi(prodA06prodA07));
pix6_7_sum1 = _add2 ( _add2 ( _hi(prodA10prodA11),
                             _hi(prodB02prodB03)), _hi(prodB06prodB07));
pix6_7_sum2 = _add2 ( _hi(prodB10prodB11), _hi(prodB02prodC03));
pix6_7_sum3 = _add2 ( _hi(prodB06prodC07), _hi(prodB10prodC11));
pixel0607 = _add2(_add2 (pix6_7_sum0, pix6_7_sum1), _add2
                  (pix6_7_sum2, pix6_7_sum3));

pixel0001 = _shr2(pixel0001,8);
pixel0203 = _shr2(pixel0203,8);
pixel0405 = _shr2(pixel0405,8);
pixel0607 = _shr2(pixel0607,8);
out_pixel00010203 = _pack14(pixel0203,pixel0001);
out_pixel04050607 = _pack14(pixel0607,pixel0405);
_memd8(p2_out) = _itod(out_pixel04050607,out_pixel00010203);
p2_out +=8;
}
}

```

המשוב שהתקבל מהקומפיילר נראה כדלקמן:

```

*-----*
* SOFTWARE PIPELINE INFORMATION
*-----*
* Loop source line      : 128
* Loop opening brace source line : 129
* Loop closing brace source line : 228
* Known Minimum Trip Count : 8
* Known Maximum Trip Count : 192
* Known Max Trip Count Factor : 1
* Loop Carried Dependency Bound(*) : 1
* Unpartitioned Resource Bound : 9
* Partitioned Resource Bound(*) : 10
* Resource Partition:
*
* A-side B-side
* .L units      1      1
* .S units      3      2
* .D units      4      6
* .M units      9      9
* .X cross paths 6      4
* .T address paths 10*   10*
* Long read paths 0      0
* Long write paths 0      0
* Logical ops (.LS) 0      0      (.L or .S unit)
* Addition ops (.LSD) 17   20      (.L or .S or .D unit)
* Bound(.L .S .LS) 2      2
* Bound(.L .S .D .LS .LSD) 9   10*
*
* Searching for software pipeline schedule at ...
* ii = 10 Register is live too long
* ii = 10 Schedule found with 2 iterations in parallel
*

```

ניתן לראות שסה"כ הלולאה מתבצעת ב- 10 מחזורים המספקים תוצאה עבור 8 פיקסלים. כלומר סה"כ עבור כל פיקסל מתבצעים $10/8 = 1.25 \text{ cycle/pixel}$ שהם רחוקים רק $1/8$ מחזורים מהחישוב התיאורטי שהגדרנו שהוא $9/8$. יתכן שכתובת קוד ב- linear assembler הייתה מביאה אותנו לביצועים של $9/8$ מחזורים לפיקסל אולם לעת עתה נסתפק בכך. לולאת האסמבלר (kernel loop) מתוארת להלן:

```

$CSL4:  .FILED LOCAL KERNEL
$CSDW$Ls_Conv3x3_kernel$6$B
.dwpn file Conv3x3_kernel.c, line 129, column 0, is_stat
    .L2X
    MPYSU4  M1  A8.A10.A5:A4  <210/> <0.9>  ^
    LDNDW  D1T1 A23.B9  A31:A30  <210/> <0.9>  ^
    ADD2    S1  A6.A4.A4  <210/> <0.9>  ^
    ADD2    D2  B24.B9.B5  <210/> <0.9>  ^
    .L2X
    MPYSU4  M1X A3.B10.A7:A6  <210/> <0.10> ^
    LDNDW  D1  B5.B25.B8  <210/> <0.10> ^
    ADD2    S1  A23.A5.A23  <210/> <0.10> ^
    ADD2    D2  B22.B27.B9:B8  <210/> <0.10> ^
    .L2X
    MPYSU4  M2  A23.A7.A23  <210/> <0.11> ^
    LDNDW  D1  A61.B24.B31  <210/> <0.11> ^
    ADD2    S1  B8.B5.B5  <210/> <0.11> ^
    MPYSU4  M1X B19.B5.B25:B24  <210/> <0.11> ^
    LDNDW  D1T1 *A26+{(8),A29:A28  <219/> <1.1>  ^
    .L2X
    ADD2    S2  B31.B4.B31  <210/> <0.12> ^
    LDNDW  D1T2 *B30(6)B11:B10  <210/> <0.12> ^
    ADD2    S1  A23.A5.A23  <210/> <0.12> ^
    MPYSU4  M1  A4.A6.A6  <210/> <0.12> ^
    MPYSU4  M2  B20.B25.A23:A22  <210/> <1.2>  ^
    LDNDW  D2T2 *B23+{(8),B27:B26  <214/> <1.3>  ^
    .L2X
    MV      L2X  A31.B4  <221/> <0.14>  ^ Define a twin register
    ADD2    S1  A23.A5.A5  <210/> <0.14>  ^
    ADD2    D1  A6.A4.A4  <210/> <0.14>  ^
    SHR2    S2  B5.B8.B5  <210/> <0.14>  ^
    MPYSU4  M2X B20.B2.B9:B8  <210/> <1.3>  ^
    MPYSU4  M1X A19.B24.A7:A6  <210/> <1.4>  ^
    LDNDW  D2T1 *B23(7).A23:A22  <215/> <1.4>  ^
    .L2X
    ADD2    S1  A5.A7.A5  <210/> <0.15>  ^
    SHR2    S2  A4.B.A4  <210/> <0.15>  ^
    MPYSU4  M1  B19.B29.B5:B4  <210/> <0.15>  ^
    MPYSU4  M2  A18.A28.A5:A4  <210/> <1.5>  ^
    LDNDW  D2T2 B17.B4.B9:B8  <210/> <1.5>  ^
    ADD2    S2  *B30+{(8),B1:B0  <210/> <1.5>  ^
    .L2X
    MV      D2X  A11.B3  <225/> <0.16>  ^ Define a twin register
    SHR2    S1  A5.B.A5  <210/> <0.16>  ^
    MPYSU4  M2  B16.B1.B25:B24  <210/> <1.6>  ^
    LDNDW  D1T2 A17.B28.A7:A6  <210/> <1.6>  ^
    ADD2    S2  *A26(7).B29:B28  <220/> <1.6>  ^
    ADD2    D2  B8.B24.B5  <210/> <1.6>  ^
    ADD2    S2  B9.B5.B24  <210/> <1.6>  ^
    .L2X
    PACKL4  L1  A5.A4.A4  <210/> <0.17>  ^ Define a twin register
    MPYSU4  M1X B31.A5  <210/> <1.7>  ^
    MPYSU4  M2  A16.A30.A5:A4  <210/> <1.7>  ^
    LDNDW  D2T2 B6.B3.B5:B4  <210/> <1.7>  ^
    ADD2    S1  *B23(5).B25:B24  <210/> <1.7>  ^
    ADD2    D2  A24.A22.A22  <210/> <1.7>  ^
    ADD2    S2  B5.B4.B9  <210/> <1.7>  ^
    ADD2    D2  B24.B9.B24  <210/> <1.7>  ^
    .L2X
    STNDW  D1T1 A5.A4.*A27+{(8)  <210/> <0.18> ^
    MPYSU4  M1X A9.B0.A7:A6  <210/> <1.8>  ^
    MPYSU4  M2  B7.B11.B9:B8  <210/> <1.8>  ^
    MV      L2X  A29.B2  <219/> <1.8>  ^ Define a twin register
    ADD2    S1  A25.A23.A24  <210/> <1.8>  ^
    ADD2    D2  A22.A6.A6  <210/> <1.8>  ^
    ADD2    S2  A22.B9.B5  <210/> <1.8>  ^
    ADD2    D2  B24.B5.B24  <210/> <1.8>  ^
    
```

מדידות שנעשו לפונקציה זו (כולל הפעלת ה- EDMA) הראו זמנים של 1.8 cycle/pixel על בלוק ברוחב 192 בתים המכיל 1000 שורות ותוצאת ריצה של 1.5 cycle/pixel על בלוק עם 1000 שורות ברוחב של 400 פיקסלים. כל העתקת EDMA הכילה 30 שורות. התוצאות די מובנות מכיוון שבדור שכל שרוב השורה גדול יותר כך נמצא זמן רב יותר בתוך ה pipeline kernel ופחות בחלקי ה- prolog וה- epilog של הלולאה.

מימוש פונקצית השוואה

הבסיס למציאת פגמים מבוסס על העברת ספים על תמונות הפרשים. המקור לתמונות הפרשים יהיה בדרך כלל מתמונות (שעברו עיבוד כלשהו) ממקומות זהים משני dies סמוכים או ממקומות דומים מתוך אותה die. בד"כ הסף של תמונת הפרשים תלוי ביחס כלשהו או בערך הפיקסל כלומר לפיקסלים עם ערכים שונים יהיו ספים שונים. בנוסף לכך בדרך כלל נרצה לנרמל את תוצאת ההפרש, כלומר להכפילה ולחלקה בגורמים כלשהם. מכיוון שאין פקודה ב DSP המאפשרת לחלק נשתמש בטריק המאפשר לחלק

ע"י פעולת כפל. נניח שאנו רוצים לחשב את הביטוי $Grade = (Diff - Threshold) * \frac{K1}{K2}$ ללא

שימוש בפעולת חילוק נבנה במקום את $\frac{K1}{K2}$ את $\frac{K1 * 256}{K2}$. מספר זה יהיה בגודל של יותר מ-8 סיביות כלומר יתפוס 16 סיביות. ע"י הכפלה ב- 256 אנו יכולים לחשב את הביטוי הבא:

$Grade = (Diff - Threshold) * \frac{K1}{K2} * 256$ רק ע"י מכפלות מכיוון שהביטוי $\frac{K1}{K2} * 256$ גדול מ-1.

נתבונן בפסידו קוד הבא שאותו נרצה כדוגמא לממש בלולאה יעילה והמבוסס על כך שעבור כל ערך של פיקסל יש סף ומקדם נרמול שונה. לשם כך נבנה טבלה LUT (look up table) שבה יש עבור כל ערך של פיקסל הסף המתאים לו שהוא בגודל בית ומקדם נרמול לפי הטכניקה שהוסברה לעיל שהוא בגודל של 16 סיביות. הפונקציה מקבלת 2 תמונות: תמונה נוכחית Image current ותמונה קודמת previous

image. תוצאת הפונקציה היא תמונת ציונים grades המתארת את ההסתברות של כל פיקסל להיות פגם. פסיאודו קוד עבור האלגוריתם הוא כדלקמן:

```
For each Pixel do
    Minimal = minimum(current,previous)
    Threshold = LUT(Minimal)
    K1,K2= LUT(minimal)
    If abs(current-previous)>Threshold
    {
        Grade = diff(current,previous) * K1/K2
        If Grade>255 Grade=255
    }
    Else
    {
        Grade=0
    }
```

כאמור כדי לממש את הלולאה נבנה LUT שעבור כל פיקסל תהיה כניסה שבה ימצאו הסף עבור הפיקסל וכן מקדם הנרמול מוכפל ב-256. כך נוכל לקבל תוצאה בגודל של 16 סיביות ולקחת את ה-8 סיביות הגבוהים שלה (פעולה השקולה לחלוקה ב-256) ובצעם לעקוף את מימוש החלוקה ע"י מכפלה בלבד. להלן המימוש של לולאת העיבוד:

```
void CalculatePreAlarm_Kernel(
    unsigned char *currBlockImagePtr,
    unsigned char *prevBlockImagePtr,
    unsigned char *lutPtr,
    unsigned short inputCurrBlockPitch,
    unsigned short inputPrevBlockPitch,
    unsigned short numOutCols,
    unsigned short numRows,
    unsigned char *PAImagePtr,
    unsigned short outputPitch)
{
    Variable declaration ....
    for(rows = 0; rows < numRows; rows++)
    {
        curr_image_ptr = currBlockImagePtr + rows*inputCurrBlockPitch;
        prev_image_ptr = prevBlockImagePtr + rows*inputPrevBlockPitch;
        npa_image_ptr = PAImagePtr + rows*outputPitch;
#pragma MUST_ITERATE (8,256);
        for(col = 0; col < numOutCols /*+ 16*/; col+=8)
        {
            // load images
            curr_image_l = _lo(_memd8(curr_image_ptr));
            curr_image_h = _hi(_memd8(curr_image_ptr));
            curr_image_ptr +=8;
            prev_image_l = _lo(_memd8(prev_image_ptr));
            prev_image_h = _hi(_memd8(prev_image_ptr));
            prev_image_ptr +=8;
            // Subtract the previous image from the current image
            // in order to get smoothed difference
            diff_image_l = _subabs4(curr_image_l, prev_image_l);
            diff_image_h = _subabs4(curr_image_h, prev_image_h);
            min_image_l = _minu4(curr_image_l, prev_image_l);
            min_image_h = _minu4(curr_image_h, prev_image_h);
            // extract minimum pixel
            curr_entry0=_extu(min_image_l,24,24);
            curr_entry1=_extu(min_image_l,16,24);
            curr_entry2=_extu(min_image_l,8,24);
            curr_entry3=_extu(min_image_l,0,24);
            curr_entry4=_extu(min_image_h,24,24);
            curr_entry5=_extu(min_image_h,16,24);
            curr_entry6=_extu(min_image_h,8,24);
            curr_entry7=_extu(min_image_h,0,24);
            //LUT access
```

קריאה של תמונות
current,previous

חישוב
Abs(diff),minimal

בניית כניסות ל-
LUT

```
curr_thresh_coef0 = _mem4(lutPtr+curr_entry0*3);
curr_thresh_coef1 = _mem4(lutPtr+curr_entry1*3);
curr_thresh_coef2 = _mem4(lutPtr+curr_entry2*3);
curr_thresh_coef3 = _mem4(lutPtr+curr_entry3*3);
curr_thresh_coef4 = _mem4(lutPtr+curr_entry4*3);
curr_thresh_coef5 = _mem4(lutPtr+curr_entry5*3);
curr_thresh_coef6 = _mem4(lutPtr+curr_entry6*3);
curr_thresh_coef7 = _mem4(lutPtr+curr_entry7*3);
// convert th back to vector
threshold_l = _packh4(_packl4(curr_thresh_coef3, curr_thresh_coef2),
                      _packl4(curr_thresh_coef1, curr_thresh_coef0));

threshold_h = _packh4(_packl4(curr_thresh_coef7, curr_thresh_coef6),
                      _packl4(curr_thresh_coef5, curr_thresh_coef4));
// build norm*2^(8-k)
norm_coef_ll = _pack2(curr_thresh_coef1, curr_thresh_coef0);
norm_coef_lh = _pack2(curr_thresh_coef3, curr_thresh_coef2);
norm_coef_hl = _pack2(curr_thresh_coef5, curr_thresh_coef4);
norm_coef_hh = _pack2(curr_thresh_coef7, curr_thresh_coef6);
// Subtract the threshold from the smoothed difference and
// compare with zero (if [SmoothDiff - Thresh] < 0, than 0 is
// the maximum between the two).
npa22_l = _sub4(diff_image_l, threshold_l);
npa22_h = _sub4(diff_image_h, threshold_h);
npa_mask22_l = _xpnd4(_cmpgtu4(diff_image_l, threshold_l));
npa_mask22_h = _xpnd4(_cmpgtu4(diff_image_h, threshold_h));
npa22_l &= npa_mask22_l;
npa22_h &= npa_mask22_h;
npa_h_16bit = _mpyu4(npa22_h, 0x01010101);
npa_l_16bit = _mpyu4(npa22_l, 0x01010101);
double_npa_ll = _mpy2(_lo(npa_l_16bit), norm_coef_ll);
double_npa_lh = _mpy2(_hi(npa_l_16bit), norm_coef_lh);
double_npa_hl = _mpy2(_lo(npa_h_16bit), norm_coef_hl);
double_npa_hh = _mpy2(_hi(npa_h_16bit), norm_coef_hh);
npa_hhh = (_hi(double_npa_hh));
npa_hhl = (_lo(double_npa_hh));
npa_hlh = (_hi(double_npa_hl));
npa_hll = (_mvd(_lo(double_npa_hl)));
npa_lhh = (_hi(double_npa_lh));
npa_lhl = (_lo(double_npa_lh));
npa_llh = (_hi(double_npa_ll));
npa_lll = (_mvd(_lo(double_npa_ll)));
// saturate checking
if (_hi(double_npa_hh) > 0xffff) npa_hhh = (0xffff);
if (_lo(double_npa_hh) > 0xffff) npa_hhl = (0xffff);
if (_hi(double_npa_hl) > 0xffff) npa_hlh = (0xffff);
if (_lo(double_npa_hl) > 0xffff) npa_hll = (_mvd(0xffff));
if (_hi(double_npa_lh) > 0xffff) npa_lhh = (_mvd(0xffff));
if (_lo(double_npa_lh) > 0xffff) npa_lhl = (0xffff);
if (_hi(double_npa_ll) > 0xffff) npa_llh = (0xffff);
if (_lo(double_npa_ll) > 0xffff) npa_lll = (0xffff);
final_npa_h = _packh4(_pack2(npa_hhh, npa_hhl),
                      _pack2(npa_hlh, npa_hll));
final_npa_l = _packh4(_pack2(npa_lhh, npa_lhl),
                      _pack2(npa_llh, npa_lll));

// store output image
_memd8(npa_image_ptr) = _itod(final_npa_h, final_npa_l);
npa_image_ptr+=8;
}
}
}
```

בניית ספים ומקדמי
נרמול בצורה וקטורית

חישוב הפרש ובניית
מסיכה למיסוך
הקטנים מהספים

מיסוך הקטנים
מהספים.

בדיקת רוויה

חזרה ל- 8 סיביות ע"י
שימוש בפקודות
Packl4, packh4

אחסון התוצאה

כאמור לולאת זו שהיא בסיס ללולאות עבור אלגוריתמים ל- WI מתבצעת ב-16 מחזורי מכונה ועובדת על 8 פיקסלים כלומר לוקחת 2cycle/pixel כמו שניתן לראות מהמשוב שהמהדר מספק.

```

*-----*
*  SOFTWARE PIPELINE INFORMATION
*
*  Loop source line           : 44
*  Loop opening brace source line : 45
*  Loop closing brace source line : 137
*  Known Minimum Trip Count     : 8
*  Known Maximum Trip Count     : 256
*  Known Max Trip Count Factor   : 1
*  Loop Carried Dependency Bound(^) : 6
*  Unpartitioned Resource Bound  : 14
*  Partitioned Resource Bound(*) : 16
*  Resource Partition:
*
*      A-side    B-side
*  .I units      11      11
*  .S units       6       5
*  .D units      10       9
*  .N units       6       6
*  .X cross paths  9      13
*  .T address paths 11     11
*  Long read paths  0       0
*  Long write paths 0       0
*  Logical ops (.LS)      4      (.I cr .S unit)
*  Addition ops (.LSD)    16     (.I cr .S cr .D unit)
*  Bound(.I .S .LS)      11     10
*  Bound(.I .S .D .LS .LSD) 16*   15
*
*  Searching for software pipeline schedule at ...
*  ii = 16 Schedule found with 4 iterations in parallel
*

```

16 מחזורים

תוצאות ריצה אלגוריתם D2D בסיסי

מדידת מספר המחזורים עבור גדלי בלוק שונים עבור אלגוריתם השוואה ללא מדידת ותיקון ההיסט וטיפול בבעיית ה- color variation הוא כמובן היעיל ביותר מבחינת זמן ריצה. להלן התוצאות:

מספר מחזורים בבלוק עם 2000 שורות	מספר מחזורים בבלוק עם 1000 שורות	מספר מחזורים בבלוק עם 500 שורות	רוחב בלוק
7.1	7.3	7.5	100
6.7	6.8	7	120
5.2	5.4	5.5	152
5.1	5.2	5.4	168
4.5	4.9	5.2	192
4.15	4.3	5	220
3.55	3.7	4.5	400

טבלה 16 טבלת זמני ריצה עבור בלוקים בגדלים שונים

מכיוון שיחס אות לרעש באלגוריתם המבוסס על השוואה בלבד הוא נמוך מאוד הן בגלל בעיית רגיסטרציה והן בגלל בעיית ה- color variation נקבל יחס false alarm גבוה מכיוון שקשה להפריד את אות הפגם מאות הרעש. אלגוריתם זה יישמש אך ורק כבסיס להשוואת ביצועים של האלגוריתמים שתוארו בהתייחס למספר המחזורים עבור פיקסל.

מספר ה- cycles שנמדדו גבוה מזה שתואר בפונקציית [ההשוואה הבסיסית](#) מהסיבות הבאות:

- כתשתית לאלגוריתמים נבנתה לולאה שבה בנוסף לתמונת ה- current וה- previous ישנם עוד שתי תמונות המשמשות כמאפיינות את רמת הרעש של הפיקסל. ישנה תמונה אחת עבור ה- current ותמונה אחת עבור ה- previous. כל בית בתמונה מכילה מספר בין 0-15 המציין סיווג לרמת רעש. רמת הרעש נבחרת בצורה כלשהי (נניח פעולת maximum מבין הסיווג של ה- current וה- previous). בנוסף הוגדרה רמת רעש שבה ממסכים אזור כלומר הציונים שלו יהיו 0. הסיבה להגדרת רמת רעש כזו היא הרצון למסך אזורים שלא רוצים לסרוק ב- die2die אלא ב- cell2cell או שמסיבות כלשהם לא רוצים לסרוק בכלל. הוספת הטיפול בתמונות הללו הגדילה את מספר המחזורים בלולאה.
- הגבול התחתון של מספר שהפונקציה תארך הוא כמובן מספר המחזורים בתוך הלולאה ההדוקה. אולם יש לזכור שעדיין צריך לגלגל את ה- data מהזיכרון החיצוני לפנימי ולהפך פעולה שמפעילה קוד ניהול שגם לו יש תקורה ובנוסף אנו עובדים על שורות באורך סופי ולכן אין ניצול מלא של ה- pipeline.

סיכום מגבלות במימוש אלגוריתמים למכונת WI

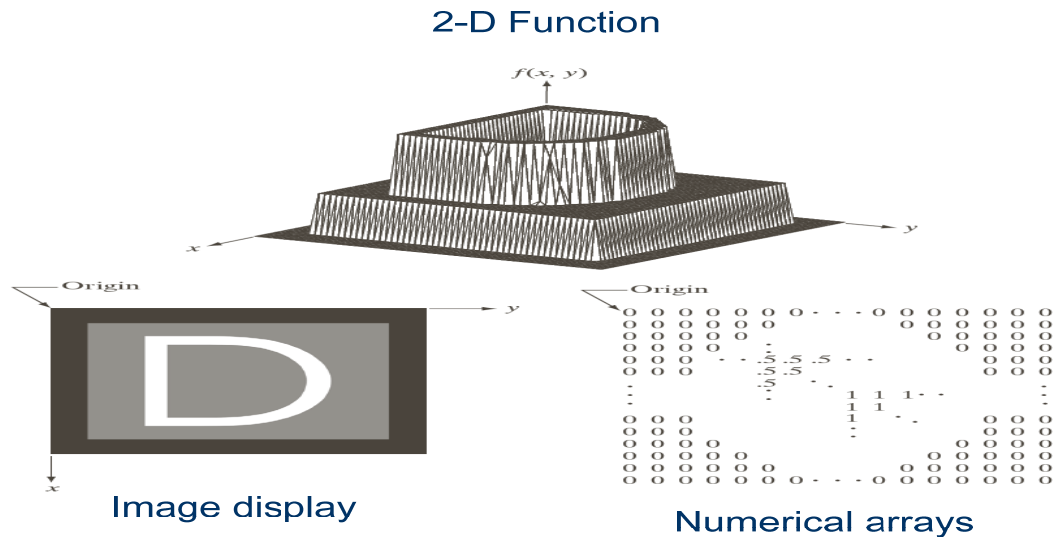
כאמור בסעיפים הקודמים תוארה מערכת עיבוד התמונה שהיחידה הבסיסית שלה היא DSP שמבצעים עיבוד מקבילי של המידע הנסרק כבר ברמת הסריקה כאשר שטח ה-die הנסרק מחולק בין כל ה-DSPs. כל DSP מעבד את חלק ה-die השייך לו וכמובן שרוכש רק את חלקי התמונה השייכים לאזור זה. בנוסף כל DSP מבצע עיבוד מקבילי בעזרת הטכניקות הבאות שתוארו בפירוט הסעיפים קודמים:

- ביצוע לולאת עיבוד בטכניקת pipeline.
- עיבוד וקטורי של המידע.
- עיבוד מידע במקביל להבאה ברקע של מידע חדש לעיבוד בעזרת העתקות EDMA הנעשות במקביל לעיבוד.

כמובן שטכניקות אלו הכרחיות כדי להאיץ את הביצועים ולהתקרב לאפשרות שנוכל לממש את האלגוריתמים בהתאם לדרישות זמן האמת שהוגדרו כלומר לעבד סדר גודל של מספר גיגה בית לשנייה. בחישובים שנעשו בפרק 3 המסקנה המתבקשת הייתה שלכל DSP יש סדר גודל של 50-60 מחזורי עיבוד בממוצע לכל פיקסל. כפי שנראה בהמשך רוב האלגוריתמים כוללים מספר שלבים ובנוסף לכך ישנם מספר אלגוריתמים שונים כפי שנראה בהמשך (C2C, Die2Die ועוד) הרצים על אותו בלוק. יוצא מכך שסדר גודל של מחזורי מעבד עבור פונקציית עיבוד תמונה המשמשת כחלק מאבני הבניין במימוש האלגוריתמים לא יעלה על מספר מחזורים בודדים לפיקסל. מגבלה זו בנוסף לעובדה שצריך להשתמש בטכניקות שתוארו לעיל דורשת יצירתיות רבה במימוש האלגוריתמים כדי לאפשר עבודה יעילה ולוקאליות מתאימה בארגון המידע ובעבודה עליו.

נספח 5 - עקרונות תיאורטיים בעיבוד תמונה

קיימות שתי גישות עיקריות לעיבוד תמונה המבוססות על עיבוד אותות באופן כללי. בשיטה הראשונה עובדים במרחב התמונה ומבצעים פעולות על כל פיקסל. ניתן להסתכל על התמונה כפונקציה $f(x, y)$ שבה לכל זוג x, y ישנו ערך המתאר את רמת הבהירות של הפיקסל. ניתן גם להסתכל על התמונה כפונקציה דו ממדית או כמערך נומרי דו ממדי כמתואר באיור הבא:



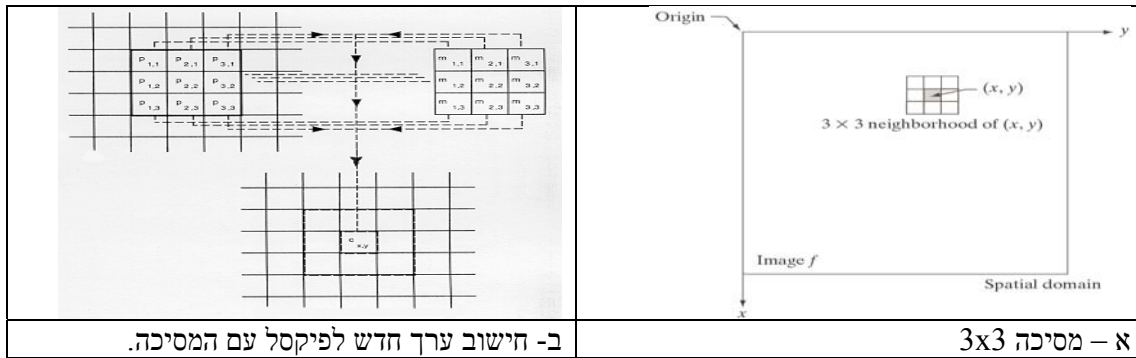
איור 98 תיאור אפשרויות ההצגה השונות של מרחב התמונה

ישנם פעולות רבות שניתן לבצע על הפיקסלים בתמונה למשל ניתן לשנות את ערכו של כל פיקסל בתמונה במידה שהוא מקיים תנאים מסוימים. לדוגמא ניתן לבצע בינרזציה על כל פיקסל בתמונה בכך שכל פיקסל שערכו גדול מסף מסוים יקבל ערך 255 אחרת ערך 0. בנוסף לכך ניתן לבצע פעולות אריתמטיות בין תמונות כגון חיסור/חיבור של תמונות שהכוונה לפעולות בין פיקסלים מתאימים (עם אותן מיקומי x, y). הפעולות הללו מאפשרות שינוי ערכו של הפיקסל בתמונה אך אין התייחסות לסביבתו של הפיקסל. כאשר נדרש לשנות את מאפייני התמונה המרחביים כמו הדגשה או החלקה של שינויי בהירות יש להשתמש בפעולות מרחביות הקרויות גם מסנן מרחבי. פעולות אלו מתבססות על פיקסלים הקרובים של כל פיקסל, כדי לחשב את ערכו החדש – הערך של כל פיקסל מתקבל משקלול ערכי הפיקסלים בסביבתו. סינון מרחבי של תמונה מתבצע בעזרת פעולת קונבולוציה (convolution) עם המסנן. אם המסנן הוא בגודל $M \times N$ אז פעולת הקונבולוציה עם מסנן בנקודה משתמשת בערכי M דגימות סמוכות בממד האופקי ו- N דגימות סמוכות בממד האנכי לחישוב הערך החדש בנקודה הנידונה. הביטוי לקונבולוציה דו ממדית

$$\text{הוא: } g(m, n) = f(m, n) * h(m, n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f(k, l) h(m-k, n-l)$$

המסנן $h(m-k, n-l)$ מתקבל לאחר שהמסנן עבר קיפול בשני הממדים והזזה בשיעור m בכיוון האופקי ו- n בכיוון האנכי. בדרך כלל ניתן כאמור להסתפק בשקלול של הנקודות הקרובות לנקודה הנידונה ופעולה זו נקראת מסכת סינון או חלון סינון. גדלים מקובלים הם 3×3 , 5×5 , 7×7 , 9×9 ועוד. מסכה זו מוזזת על פני התמונה מנקודה לנקודה ובכל נקודה (m, n) מתבצע שקלול של ערך הנקודה וסביבותיה.

האיור הבא מציג את דרך הפעולה של האמור לעיל עם מסכה בגודל 3×3 .



איור 99 תיאור גרפי של פעולת מסנן מסכה בגודל 3x3

כאמור **בנספח 4 (מימוש מסנן 3x3)** מתואר מימוש של מסנן מסוג דומה תוך ניצול יכולות ה-DSP כדי להשיג ביצועים מקסימליים. קיימים סוגים שונים של מסננים מרחביים. לדוגמא מסנן מעביר נמוכים מעביר בלא שינוי מרכיבי תמונה בעלי תדר מרחבי נמוך. מרכיבי תמונה בעלי תדר מרחבי גבוה עוברים ניחות ולמעשה הם חסרים בתמונת המוצא. דוגמא למסנני מעביר נמוכים ניתן לראות באיור הבא:

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

איור 100 מסנני מעביר נמוכים

כדי להיווכח שזהו מסנן מעביר נמוכים נסתכל על משטח אחיד שכולו בעל אותו ערך בהירות לכל הפיקסלים. ניתן לתאר משטח זה בתמונה המכילה רק תדר 0 או DC. העברת המסננים המתוארים לעיל לא תשנה את ערכי הבהירות והיא נותרה כפי שהייתה לפני הסינון. באותו אופן לא יושפעו תדרים מרחביים נמוכים אחרים שכן השתנות הבהירות בקטעי תמונה כאלה נמשכת על פני מספר פיקסלים גדול מרוחב המסנן והמיצוע לא ישנה הרבה. לעומת זאת תדר מרחבי גבוה מונחת או מתבטל ע"י מסנן זה. בפועל כאמור כדי להימנע מחלוקות הכפלנו את מקדמי המסנן כך שסה"כ התוצאה מחולקת ב 256 כדי לקחת את החלק הגבוה של תוצאת ה-16 סיביות בעזרת פקודת PACKL ובכך להימנע מחלוקה. מסנן מעביר גבוהים הוא בעל השפעה הפוכה מזו של מסנן מעביר נמוכים. הוא מדגיש תדרים מרחביים גבוהים ומנחת תדרים מרחביים נמוכים. ניתן לחשב את מקדמיו האפשריים לפי החישוב הבא: כדי להדגיש תדרים גבוהים נרצה להעביר או לחזק את נגזרות התמונה. נגזרת חלקית בציר x מוגדרת כדלקמן:

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

התמונה. הנגזרת השנייה מחושבת

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) - f(x) - (f(x) - f(x-1)) = f(x+1) - 2f(x) + f(x-1)$$

כדלקמן:

כאן תיארונו חישוב במימד אחד אך התמונה כאמור מורכבת משני ממדים לכן נבצע חישובי נגזרות חלקיות בשני ממדי התמונה באופן הבא:

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

הנגזרת החלקית בשני הכיוונים נקראת לפלאסיאן ושווה לביטוי הבא:

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

אם נרצה לתאר שינויים אלכסוניים נקבל באותה צורה את הביטוי הבא:

$$\nabla^2 f(x, y) = f(x-1, y-1) + f(x, y-1) + f(x+1, y-1) + f(x-1, y) + f(x+1, y) + f(x-1, y+1) + f(x, y+1) + f(x+1, y+1) - 8f(x, y)$$

להלן תיאור המסננים שתוארו באיור הבא:

1	1	1	0	1	0
1	-8	1	1	-4	1
1	1	1	0	1	0
ב- לפלאסיאן עם תוספת אלכסונית			א- לפלאסיאן בכיוון x,y		

איור 101 מסנני מעביר גבוהים

מסננים אלו כאמור הינם מעבירי גבוהים והם בעלי השפעה הפוכה לאלו של מעבירי נמוכים. מסננים אלו מדגישים תדרים מרחביים גבוהים ומנחיתים תדרים מרחביים נמוכים. כאמור סכום המקדמים הוא אפס ופרוש הדבר שאם בקטע תמונה רמת הבהירות אחידה אזי מוצא המסנן יהיה אפס. התוצאה של פעולת המסנן תהיה נמוכה (קרובה לאפס) גם אם בסביבתו המידית של הנקודה הנידונה רמת הבהירות של פיקסלים קרובות לזו של הנקודה. מאידך המסנן מתוכנן כך שמקדמים בעלי ערך נמוך מקיפים מקדם חיובי גדול במרכז המסנן. פירוש הדבר שלנקודה המרכזית יש יותר משקל מאשר כל אחת משכנותיה בנפרד בשקלול הערכים ע"י המסנן, אך כל הנקודות השכנות פועלות יחדיו כנגד הנקודה המרכזית. אם לדוגמא רמת הבהירות של הפיקסל המרכזי שונה במידה ניכרת מזו של שכניו המידיים השפעת השכנים תהיה זניחה. ההבדל הניכר בין ערך הפיקסל לשכניו בתמונת המקור מייצג תדר מרחבי גבוה ותדר מרחבי זה אכן מוגבר (או נותר קבוע) ע"י פעולת המסנן מעביר הגבוהים. כפי שצוין ניתן לבצע פעולות סינון מרחביות בתמונה ע"י קונוולוציה עם מסנן מרחבי מתאים. מאחר שמדובר בשינוי תכולת התדר המרחבי בתמונה אך טבעי לצפות שאפשר יהיה לבצע פעולות עיבוד על התמונה גם במרחב התדר (או אפילו באופן כללי במרחב כלשהו). קיים קשר בין פעולות סינון מרחביות בתחום התדר לאלו שבמרחב התמונה והוא מנוסח ע"י משפט הקונוולוציה.

התמרת פורייה הבדידה מאפשרת לקשר בין הדגימות הבדידות של הפיקסלים בתמונה לבין ייצוגה של התמונה במישור התדר כלומר התמרת פורייה הבדידה מקשרת בין דגימות אות נתון לבין דגימות ההתמרה שלו בתחום התדר ובעצם מאפשרת עבודה על התמונה במישור התדר. כאמור עבור אות דגום הכולל N דגימות התמרת פורייה הרציפה נתונה ע"י:

$$F(w) = \sum_{n=0}^{N-1} f(n) e^{-jnw}$$

כאשר w הוא התדר הזוויתי. טור פורייה הבדיד $\tilde{F}(k)$ הוא דגימה בתחום

התדר של ההתמרה הרציפה $F(w)$ עם מרווח של $\Delta w = \frac{2\pi}{N}$ בין הדגימות:

$$\tilde{F}(k) = F(w) | w = k\Delta w = \sum_{n=0}^{N-1} f(n) e^{-jnk\Delta w} = \sum_{n=0}^{N-1} f(n) e^{-j\frac{2\pi}{N}nk}$$

עבור k כלשהו

בטור פורייה הבדיד אין סוף נקודות אולם ניתן להראות שהטור הוא מחזורי עם מחזור באורך N . התמרת פורייה הבדידה ההפוכה מקשרת בין דגימות ההתמרה בתחום התדר לבין דגימות האות המקוריות:

$$F(n) = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{F}(k) e^{j\frac{2\pi}{N}nk}$$

התמרת פורייה הבדידה שקשורה להתמרת פורייה הרציפה מעתיקה אוסף

של דגימות אות לאוסף סופי (בעל אותו אורך) של דגימות בתחום התדר. דגימות אלו בתחום התדר מהוות קירוב של דגימות ההתמרה הרציפה של האות הרציף לכן התמרת פורייה הבדידה כמו הרציפה מצביעה על מרכיבי התדר של האות.

את הפיתוח של התמרת פורייה הבדידה ניתן להרחיב בקלות לתמונה שהיא כאמור אות דו-ממדי בצורה הבאה:

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$$

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$$

מורכבות החישוב של התמרת פורייה היא $O(N*N)$ עבור התמרה חד ממדית ניתנת לפתרון ע"י שימוש ב-FFT שלא נרחיב כאן עליה. משפט הקונולוציה שהוזכר בהתחלת הסעיף אומרת שהתמרת פורייה הבדידה של הקונולוציה בין תמונה ומסנן שווה למכפלת התמרות פורייה הבדידות של התמונה והמסנן. ניתן להגדיר זאת כדלקמן:

$$f(x, y) * h(x, y) \Leftrightarrow F(u, v) H(u, v)$$

$$g(x, y) = \mathcal{F}^{-1} \{ H(u, v) F(u, v) \}$$

כלומר ניתן לבצע התמרה של האות למישור התדר, לבצע פעולות עיבוד במישור התדר ולחזור למישור התמונה כמו שמודגם באיור הבא:

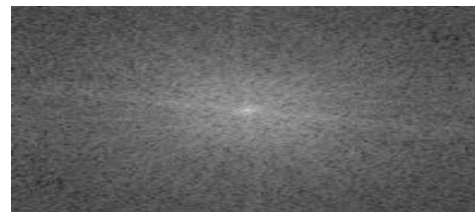


איור 102 פעולה אופיינית ע"י התמרה למישור חדש (תדר) פעולות בו וחזרה למישור המקור (תמונה)

האיור הבא מציג תמונה והתמרת פורייה הבדידה שלה:



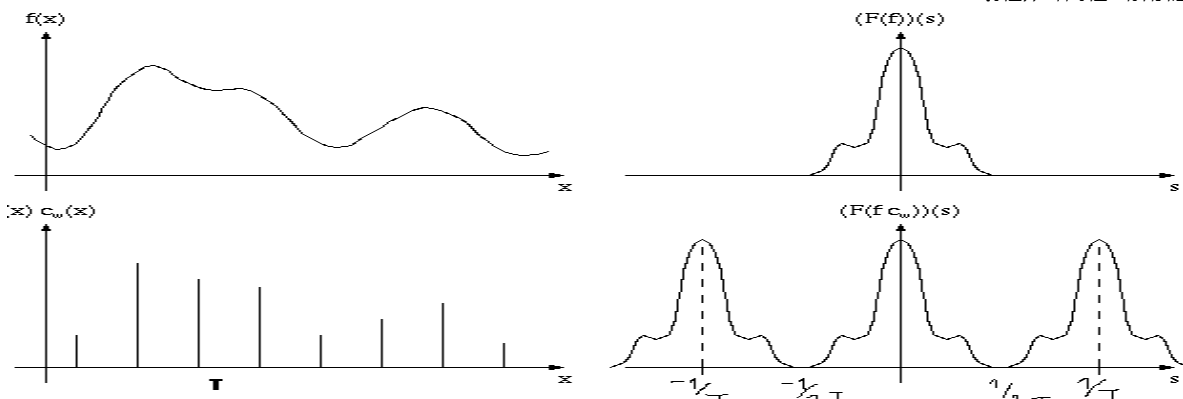
Spatial domain



Frequency domain

איור 103 תמונה (צד שמאל) והתמרת פורייה הבדידה שלה (צד ימין).

הנקודה המרכזית מציינת תדר 0 או DC כאשר שאר הנקודות מתארות את העוצמה של תדרים מרחביים בתמונה. מכיוון שהתמרת פורייה של אות דגום היא מחזורית נוצרות רפליקות (שכפולים) של המחזור כפי שמתואר באיור הבא:



איור 104 תאור התמרת פורייה של אות דגום והשכפולים הנוצרים ממנו.

כדי להבטיח שהשכפולים לא יגרמו לעיוותים דרוש שתדר הדגימה יהיה מספיק גדול כלומר ש- T יהיה מספיק צר. הבחירה בין עבודה בתחום התדר או בתחום התמונה מסתמכת בעיקר על מורכבות החישוב. סינון בתחום התמונה דורש את הזנת המסנן על פני התמונה ותלוי בגודל המסנן. בעבודה בתחום התדר יש לחשב את שתי ההתמרות גם של התמונה וגם של המסנן לעבודה בתחום התדר (נקודה לנקודה) ולחזור

למישור התמונה ע"י ההתמרה ההופכית. ברור שכאשר ממדי המסנן קטנים עדיף לעבוד במישור התמונה אולם עבור מסננים גדולים יתכן וע"י שימוש ב-FFT יהיה כדאי יותר לעבוד במישור התדר. בנוסף למורכבות החישוב לעיתים קל יותר לנתח תמונות מהתבוננות במישור התדר ולא במישור התמונה. לדוגמא (כפי שנראה בהמשך) נניח שיש לתמונה תדרים מחזוריים ידועים והם משובשים מסיבה כלשהי ע"י תדרי רעש, ניתן יהיה בקלות לסלקם מתמונת ההתמרה ולחזור למישור התמונה. שתי השיטות שתוארו לעיל דהינו עבודה במישור התמונה ועבודה במישור התדר נחשבות כשיטות לינאריות. אפשרויות נוספות לפעולות על תמונה יכולות להיות גם לא לינאריות כדוגמת מסנן 3×3 מסוג חציון או הפעלת ספים כלשהם על תמונות כלשהם. פעולות נוספות הנפוצות בעיבוד תמונה ויעשה בהם שימוש נרחב בפרק הבא כוללות איסוף מידע סטטיסטי על התמונה בעזרת היסטוגרמות. היסטוגרמות אלה יכולות להיות של תמונות הפרשים או כל תמונת ביניים המתהווה באלגוריתם והם מאפשרות לאלגוריתמים להסתגל לשינויים כלומר להיות אדפטיבי ויציב יותר.

ביבליוגרפיה

- [1] Steven Lange, Becky Pinto, Jorge Fernandez **"Full-spectrum brightfield inspection uncovers IC defects"**
- [2] Nurit Raccach, Daniel Rogers, Ehud Tzuri, Arik Shavit **"Extending Inspection limits using a DUV-laser based bright-field inspection tool"** MICRO Advanced Process control, Defect reduction and Yield management
- [3] Kenneth W. Tobin. **"Inspection in Semiconductor Manufacturing"** Webster's Encyclopedia of Electrical and Electronic Engineering, vol. 10, pp. 242-262, Wiley & Sons, NY, NY, 1999.
- [4] Elias N. Malams, Euripides G.M. Petrakis, Michalis Zervakis, Laurent Petit. **"A survey on industrial vision systems, applications and tools"**, Image and Vision Computing 21 (2003) 171-188
- [5] Stan Stokowski, Mehdi Vaez-Iravani, **"Wafer Inspection Technology Challenges for ULSI Manufacturing"**, KLA-Tencor.
- [6] Sheng-Uei Guan, Pin Xie, **"A Golden Block Self-Generating Scheme for Continuous Patterned Wafer Inspections"**, IEEE Computer Society ICIAP, p. 436, 10th International Conference on Image Analysis and Processing 1999.
- [7] Sheng-Uei Guan, Pin Xie, **"A golden template self-generating method for patterned wafer inspection"**, <http://bura.brunel.ac.uk/bitstream/2438/1091/1/A+Golden+Template+Self-generating+Method+for+Patterned+Wafer+Inspection.pdf>
- [8] Sheng-Uei Guan, Pin Xie, Hong Li **"A Golden Block Self-refining scheme for repetitive patterned wafer inspections"**, Machine Vision and Applications Volume 13, Issue 5-6 (March 2003) Pages: 314-321.
- [9] J. Vartiainen, A. Sadovnikov, L. Lensu, J.-K. Kamarainen, H. Kälviäinen **"Detecting irregularities in regular patterns"** <http://www2.lut.fi/~jkamarai/publications/downloads/laitosrap93.pdf>
- [10] J. Vartiainen, A. Sadovnikov, L. Lensu, J.-K. Kamarainen, H. Kälviäinen **"Machine vision algorithms for semiconductor wafer inspection: a project with Inspex"**, Electrical and Computer Engineering Dept., University of Massachusetts Dartmouth
- [11] Chi-ho Chan, K. H. Pang, **"Fabric Defect Detection by Fourier Analysis"** IEEE Transactions on industry applications, VOL. 36, NO. 5, september/october 2000
- [12] Wolfram MathWorld, 1 January 2007, <http://mathworld.wolfram.com/>
- [13] TMS320C6000 Programmer's Guide Rev I, <http://focus.ti.com/lit/ug/spru198i/spru198i.pdf>
- [14] Streaming SIMD Extensions, http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions
- [15] Semiconductor manufacturing technology instructor's manual, http://www.smtbook.com/instructor_guide.pdf
- [16] [http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA/\\$file/CB_EA_v1.02_11Oct2007_pub.pdf](http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA/$file/CB_EA_v1.02_11Oct2007_pub.pdf)
- [17] Shaun S. Gleason, Martin A. Hunt, W. Bruce Jatko, **"Subpixel measurement of image features based on paraboloid surface fit"** Oak Ridge National Laboratory*
- [18] Lisa Gottesfeld Brown, **"A Survey of Image Registration Techniques"**

- Department of Computer Science Columbia University New York , NY 10027 , January 1992
- [19] Chung Jun,Sang Chon,Hyoung Kim, **"Wafer Color Variation Correction Method"**
United States Patent April 2006
- [20] Avishai Bartov, **"Color Variation Compensation Algorithm for Bright Field Wafer Inspection"**, July 2003
- [21] STEPEHEN K.Park , **"Image Reconstruction by Parametric Cubic Convolution"**
NASA , Langley Research Center , Hampton Virginia
Computer Vision,Graphics and Image processing 23,258-272
- [22] Byoung-Ho Lee,Dong-Chul Ihm, Jeong-Ho Yeo, Yael Gluk, Doron Meshulach **"Polarization Control Enhanced Defect Detection Advanced Memory Devices"** R&D Center, Memory Business Division, Samsung Electronics Co. Ltd, Korea, Wafer Inspection Division/PDC, Applied Materials, Israel
- [23] Zeev Litichever,Erez Sail,Oren cohen ,**"ADVANCED CELL-TO-CELL INSPECTION"**
United States Patent Application Publication Jan.31 2007
- [24] Steven R. Lange, Becky Pinto, Jorge Fernandez **"Advantages of Broadband Illumination for Critical Defect Capture at 65-nm Node and Below"** KLA-Tencor Corporation
- [25] Babak H. Khalaj, Hamid K. Aghajan, Thomas Kailath **"Automated Direct Patterned Wafer Inspection"** Information Systems Laboratory Department of Electrical Engineering Stanford University Stanford, CA 94305.
- [26] Catherine Perry-Sullivan, Erwan Le Roy, Steven R. Lange, Irfan Malik, Avinash Yerabaka, Adrian Wilson,Mark Shirey, Becky Pinto **"Broadband Brightfield Inspection Enables Advanced Immersion Lithography Defect Detection"** KLA-Tencor Corporation
- [27] Takashi Hiroi, Shunji Maeda, Hitoshi Kubota, Kenji Watanabe*, Yasuo Nakagawa **"Precise Visual Inspection for LSI Wafer Patterns Using Subpixel Image Alignment"** Production Engineering Research Laboratory, Hitachi Ltd. 292 Yoshida-cho, Totsuka-ku, Yokohama, 244, Japan, Semiconductor & Integrated Circuit Division, Hitachi Ltd. 5-20- 1 Jousuihonmachi, Kodaira, 1 87, Japan.
- [28] Kenji Watanabe, Dr. Eng. Shunji Maeda, Dr. Eng. Tomohiro Funakoshi Yoko Miyazaki , **"DUV Optical Wafer Inspection System for 65-nm Technology Node"**
- [29] Mari Nozoe, Masami Ikota **"Recent Technology for Particle Detection on patterned Wafers"** Saitama Works, Hitachi Electronics Engineering Co., Ltd.
- [30] Mark A. Schulze, Martin A. Hunt, Edgar Voelkl, Joel D. Hickson, William Usry, Randall G. Smith, Robert Bryant, C. E. (Tommy) Thomas Jr **"Semiconductor wafer defect detection using digital holography"** nLine Corporation, 4150 Freidrich Lane, Suite A, Austin, Texas 78744
- [31] Akira Hamamatsu, Hisae Shibuya, Yoshimasa Oshima, Shunji Maeda,Hidetoshi Nishiyama, Minoru Noguchi **"STATISTICAL THRESHOLD METHOD FOR SEMICONDUCTOR INSPECTION"** Hitachi, Ltd. Production Engineering Research Laboratory.
- [32] Mark Keefer, Rebecca Pinto, Cheri Dennison,and James Turlo **"The Role Of Metrology And Inspection In Semiconductor Processing"** KLA
- [33] <http://www.micromagazine.com/archive/98/02/dennison.html>
- [34] Michael Quirk, Julian Serda **"Semiconductor Manufacturing Technology"** Prentice Hall

רשימת האיורים:

4	דוגמא לפגם בשבב סיליקון	איור 1
6	גליל סיליקון בצורתו הגולמית המובא ל- FAB	איור 2
6	ייצור השבבים על הפרוסה.	איור 3
7	בדיקה ומיון של כל die.	איור 4
7	חיתוך ואריזה.	איור 5
7	דוגמא ל- Wafers שונים	איור 6
8	תיאור עקרוני של תהליך הליתוגרפיה	איור 7
11	תרשים גרפי של תהליך בקרת האיכות כתלות בתהליך הייצור	איור 8
11	א-ד דוגמאות לתמונות מפרוסות סיליקון עם פגמים	איור 9

14	איור 10	א-ב תיאור קואורדינטות והגדרות DF, BF
16	איור 11	תיאור גרפי של המושגים die, slice ו-wafer
17	איור 12	תיאור גרפי של סריקת ה-slices ב-wafer
20	איור 13	תיאור פסידו קוד של double buffering implementation
22	איור 14	תיאור מבנה בסיסי של node ותהליך רכישת התמונה
23	איור 15	תאור גרפי עקרוני של אלגוריתם D2D עם השוואה כפולה
26	איור 16	תאור גרפי של אלגוריתם למציאת פגמים מבוסס d2d עם תיקון תופעת ה-color variation
28	איור 18	תאור גרפי של תזוזת חלון C על M
28	איור 18	תאור גרפי של מטריצת הקורלציה במדידת תזוזת חלון C על M
28	איור 19	תאור גרפי של תזוזת מטריצת הקורלציה בגודל 5x5
29	איור 20	תאור גרפי של פרבולה סימטרית מסביב למקסימום
29	איור 21	תאור גרפי של שלוש נקודות מדידה, הקירוב לפרבולה וההיסט Δx או Δy
30	איור 22	תאור גרפי של אינטרפולציה בעזרת spline במימד אחד
31	איור 23	תאור גרפי של שחזור הפיקסל בהיסט תת-פיקסלי בתלות במקדמים k0-k3 וארבע פיקסלים שכנים
31	איור 24	תאור גרפי של אינטרפולציה בעזרת cubic spline בחד מימד
32	איור 25	תאור בלוק לפני תיקון ואחרי תיקון ההיסט במימד X
33	איור 26	תאור דוגמא של CV בין שלושה dies עוקבים
33	איור 27	תאור האזור הכהה והבהיר משני dies סמוכים עם CV
34	איור 28	תאור האזורים משני dies סמוכים עם CV
34	איור 29	תאור התפלגות תמונת ההפרשים מאזור מסוים שבו נמדדה תופעת ה-CV
36	איור 30	תאור דוגמא למיפוי זוג previous current, להיסטוגרמות מיפויים
37	איור 31	תאור דוגמא לתוצאות מיפוי בהתייחס לפונקציית ההשוואה
37	איור 32	תאור המיפויים עבור 2 בלוקים עוקבים
38	איור 33	דוגמא לתוצאות האלגוריתם הנ"ל על שני בלוקים עוקבים
39	איור 34	דוגמא לתוצאות האלגוריתם הנ"ל על שני בלוקים עוקבים
41	איור 35	תוצאות זמני ריצה של die2die עבור בלוקים עם 1000 שורות
42	איור 36	סיכום מדד ה-SNR עבור בלוקים בגדלים שונים
43	איור 37	סיכום מדד ה-FA עבור בלוקים בגדלים שונים
44	איור 38	דוגמא לאזור מחזורי
45	איור 39	תיאור ההשוואות באלגוריתם cell2cell
45	איור 40	מימוש ההשוואות n מול n+1, n-1 ע"י הזזה הבלוק ב-cell, +cell
46	איור 41	תיאור זרימה עבור אלגוריתם cell2cell
47	איור 42	תיאור זמני ריצה של cell2cell עבור בלוקים עם 1000 שורות
50	איור 43	תיאור האות כהרכבה של התאים, הפגם והרעש הגאוסיאני
50	איור 44	תיאור מעבר למישור התדר סילוק ההרמוניות המחזוריות של התא וחזרה למישור האות
51	איור 45	תיאור תמונת בלוק המכילה תאים (cells) מחזוריים
51	איור 46	תמונת log של התמרת FFT2 של התמונה
51	איור 47	תמונת log תלת ממדית של התמרת FFT2 של התמונה
52	איור 48	תמונת תלת ממדית של התמרת FFT2 של התמונה
53	איור 49	תיאור התפלגות ספקטרום של הפגם
54	איור 50	השיפור ב-SNR של מספר רב של השוואות לעומת 2 או 3 השוואות
55	איור 51	המבנה של תמונה נקייה מפגמים עם $T_x = 6.7; T_y = 5.4$
56	איור 52	בניית תמונה נקייה מפגמים M*N מ-BB עבור תמונה עם $T_x = 6.7; T_y = 5.4$
58	איור 53	תיאור מלבני של אלגוריתם השוואת תא מחזורי
58	איור 54	תיאור דוגמא לבחירת מקדם התיקון עבור שורה כלשהי
59	איור 55	תיאור זמני ריצה כוללת של אלגוריתם השוואת תא מחזורי בבלוק עם 1000 שורות

איור 56	השוואה בין האלגוריתמים השונים	61
איור 57	מבנה אטומי של מוליך למחצה מסוג N. נקודות אדומות מציינות אלקטרון עודף.	63
איור 58	צומת PN	64
איור 59	תרשים חשמלי ומבנה CMOS.	64
איור 60	א-ב מבנה שער NOT בסיסי בטכנולוגית CMOS וצילום ננומטרי שלו	64
איור 61	א-י תיאור גרפי של שלבי ייצור הטרנזיסטור.	65
איור 62	תיאור התלות של האות בתלות האורך הגל	66
איור 63	תופעת color variation במערכות BF מבוססות מקור אור רחב פס לעומת צר פס.	67
איור 64	תיאור גרפי של פקודת SIMD.	69
איור 65	תיאור מבנה עקרוני של מעבד Super-Scalar כדוגמת מעבדי Intel	73
איור 66	תיאור מבנה עקרוני של מעבד מבוסס VLIW	74
איור 67	תיאור מבנה עקרוני של ליבת C64+	75
איור 68	תיאור שלבי ה pipeline עבור ליבת C64+	77
איור 69	א-ג תיאור מפורט של שלב ה- fetch.	77
איור 70	תיאור מפורט של שלב ה- decode.	78
איור 71	תיאור שלב ה- execution.	78
איור 72	תיאור פעולת ה- pipeline עבור רצף הוראות עם חבילת ביצוע אחת.	78
איור 73	דיאגרמת מלבנים עבור הפאזות ב- pipeline בתוכנית הדוגמא שבאיור הבא	79
איור 74	דוגמא לקטע קוד עבור שלבי ה- pipeline שבאיור הקודם	79
איור 75	תיאור פעולת ה- pipeline עבור רצף הוראות עם מספר חבילות ביצוע.	80
איור 76	א-ב תיאור גרפי של פקודות packl4,packh4	80
איור 77	א-ד תיאור גרפי של פקודות pack2,packh2,packlh2,packhl2	81
איור 78	תיאור גרפי של פקודות spacku4	81
איור 79	תיאור גרפי של פקודות shrmb,shlmb	81
איור 80	תיאור גרפי של פקודות unpacklu4,unpackhu4	82
איור 81	תיאור גרפי עקרוני של פקודות dot-product	82
איור 82	תיאור גרפי עקרוני של פקודות add2	82
איור 83	תיאור גרפי עקרוני של פקודות minu4	83
איור 84	תיאור גרפי עקרוני של פקודות LDDW	83
איור 85	תיאור גרפי עקרוני של פקודות CMPGTU4	84
איור 86	תיאור גרפי עקרוני של פקודות XPND4	84
איור 87	תיאור גרפי של קטע הקוד לבדיקת סף עבור כל פיקסל	85
איור 88	תיאור הלולאה המסכמת מכפלות של איברים מתאימים של שני וקטורים	86
איור 89	תיאור גרף התלויות של לולאת dotp	86
איור 90	תיאור קוד אסמבלר של לולאת dotp	86
איור 91	א-ב גרף תלויות מעודכן וקוד אסמבלר מתאים	87
איור 92	דוגמא לקוד המממש את dotp בעזרת loop unrolling	87
איור 93	גרף התלויות והקוד עבור מימוש בעזרת loop unrolling	88
איור 94	תיאור קוד האסמבלר היעיל ביותר עבור dotp	89
איור 95	איור עקרוני של ה- Mega-Module	91
איור 96	תיאור קשרי הגומלין ורחבי הפס לפי המודל שהוגדר.	92
איור 97	תיאור המרכיבים העיקריים של פונקציית עיבוד תמונה	93
איור 98	תיאור אפשרויות ההצגה השונות של מרחב התמונה	106
איור 99	תיאור גרפי של פעולת מסנן מסכה בגודל 3x3	107
איור 100	מסנני מעביר נמוכים	107
איור 101	מסנני מעביר גבוהים	108
איור 102	פעולה אופיינית ע"י התמרה למישור חדש (תדר) פעולות בו וחזרה למישור המקור	
109	(תמונה)	

- איור 103 תמונה (צד שמאל) והתמרת פורייה הבדידה שלה (צד ימין). 109
איור 104 תאור התמרת פורייה של אות דגום והשכפולים הנוצרים ממנו. 109

רשימת הטבלאות:

- טבלה 1 טבלת זמני ריצה של die2die עבור בלוקים בעלי 1000 שורות 40
טבלה 2 טבלת זמני ריצה של die2die עבור בלוקים עם 500 שורות 41
טבלה 3 טבלת זמני ריצה של die2die עבור בלוקים עם 2000 שורות 41
טבלה 4 טבלת SNR עבור בלוקים בגדלים שונים 42
טבלה 5 טבלת FA באחוזים עבור בלוקים בגדלים שונים 43
טבלה 6 טבלת זמני ריצה של cell2cell עבור בלוקים עם 1000 שורות 46
טבלה 7 טבלת זמני ריצה של cell2cell עבור בלוקים עם 500 שורות 47
טבלה 8 טבלת זמני ריצה של cell2cell עבור בלוקים עם 2000 שורות 47
טבלה 9 טבלת זמני ריצה כוללת של אלגוריתם השוואת תא מחזורי עבור בלוק עם 1000 שורות 59
טבלה 10 טבלת זמני ריצה כוללת של אלגוריתם השוואת תא מחזורי עבור בלוק עם 500 שורות 60
טבלה 11 טבלת זמני ריצה כוללת של אלגוריתם השוואת תא מחזורי עבור בלוק עם 2000 שורות 60
טבלה 12 סדר ביצוע הפקודות בקוד שאיננו pipe-line 75
טבלה 13 סדר ביצוע הפקודות בקוד מבוסס pipeline 76
טבלה 14 תיאור פריסת הפקודות ללא ביצוע במקביל של פעולות 89
טבלה 15 תיאור פריסת הפקודות תוך שימוש בביצוע מקבילי של פעולות - * מציין מחזור 89
טבלה 16 טבלת זמני ריצה עבור בלוקים בגדלים שונים 104