Matrix multiplier

Project Name

Matrix multiplier

MATMUL

Digital High Level Design

Version 1

By:

Yevgeni Yagoodin

Lior Penker

Version 1.1

8 March 2004

Verification Test Objectives

מטרת תוכנית הוריפיציה שלנו הייתה לעבור על מודל המכפל שייצרנו ולבדוק שהכל מתבצע כנדרש.

כדי לעשות זאת אנו כותבים נתוני מטריצה אקראיים לכל אחת מהמטריצות כופלים אותן ומוודאים את התשובה ביחס לגולדן מודל שלנו, הטריצות האקריות נקבעות ע"י הגולדן מודל ונשמרות לקובץ ממנו אנו קוראים אותן ומספקים אותן למודל שלנו.

בבדיקה הבסיסית: בודקים רק כפל בין שני מטריצות בצורה פשוטה וערכים קטנים וחיוביים.

בבדיקת אקסטרימלית:

משלבים את כל הדרישות יחד: חיוביים **ושליליים**, חיבור עם **ביאס**, הפעלת **סטרוב**, ובדיקת **אוור/אנדרפלואו**.

מבצעים כפל ובכל איטרציה שנייה מוסיפים תוצאה של כפל קודם, כך שגם חיבור מטריצות נבדק בפנים.

כל איטרציה שלישית יש בדיקה לאלמנט הסטרוב (למשל צורת שחמט על מטריצות כניסה), כך שנכתבים נתונים חדשים רק לחלקים מהמטריצות ובשאר התאים נשארים הנתונים הקודמים (לא אפסים).

בגלל אופי שני הבדיקות הקודמות בכל 6 איטרציות תיבדק אפשרות של חיבור מטריצות ביחד עם סטרוב.

בנוסף לאלמנטים הללו, כל איטרציה עובדת עם מספרים חייבים ושליליים בגדלים ענקיים עד כמה שאפשר. נבדקים גם הדגלים המעידים על underflow/over בכל איטרציה ומשווים אותם לגולדן מודל.

Test Bench High Level Diagram and Architecture

בתוכנית הוריפיקציה שלנו אנו מריצים קודם כל את הגולדן מודל שמייצר לנו את הנתונים הדרושים לנו כדי להתחיל וגם מבצע את פעולת המודל כדי שנוכל להשוות אליה את התשובות.

לאחר מכן המידע מועבר לTB שלנו, דרך קבצי טקסט, שם הוא מועבר למודל שייצרנו ואנו מחכים לתשובות ממנו.

כאשר התשובות מגיעות אנו משווים אותם לתשובות מהגולדן מודל ובהתאם מראים תקינות.

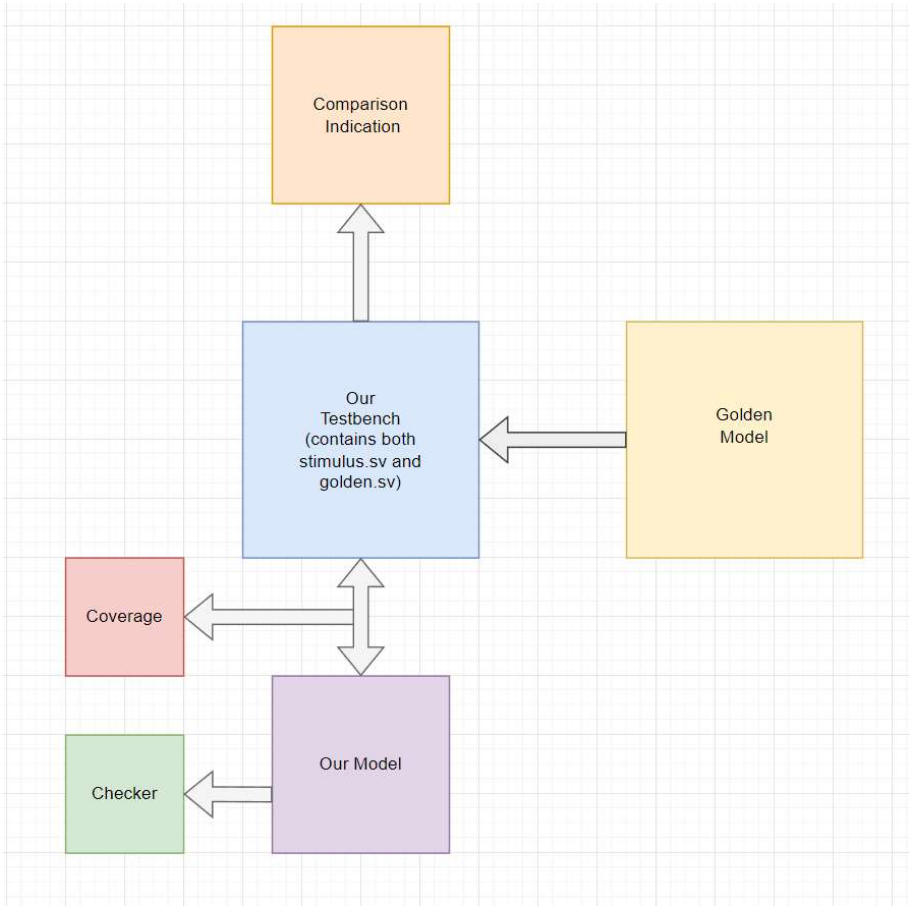
לכל כניסה שנכנסת למודל שלנו יש בדיקת כיסוי כדי לוודא שאנו עוברים על כל אופציה אפשרית.

לכל יציאה גם יש ניתור שהתוצאות בטווח האפשרי למרות שהיציאות במודל זה יכולות להיות בכל הטווח הנתון.

הכתיבה שלנו לזיכרון המודל מתבצעות ברצף ומתארות כתיבה ברצף מהבאס למודל שלנו בעזרת הAPB.

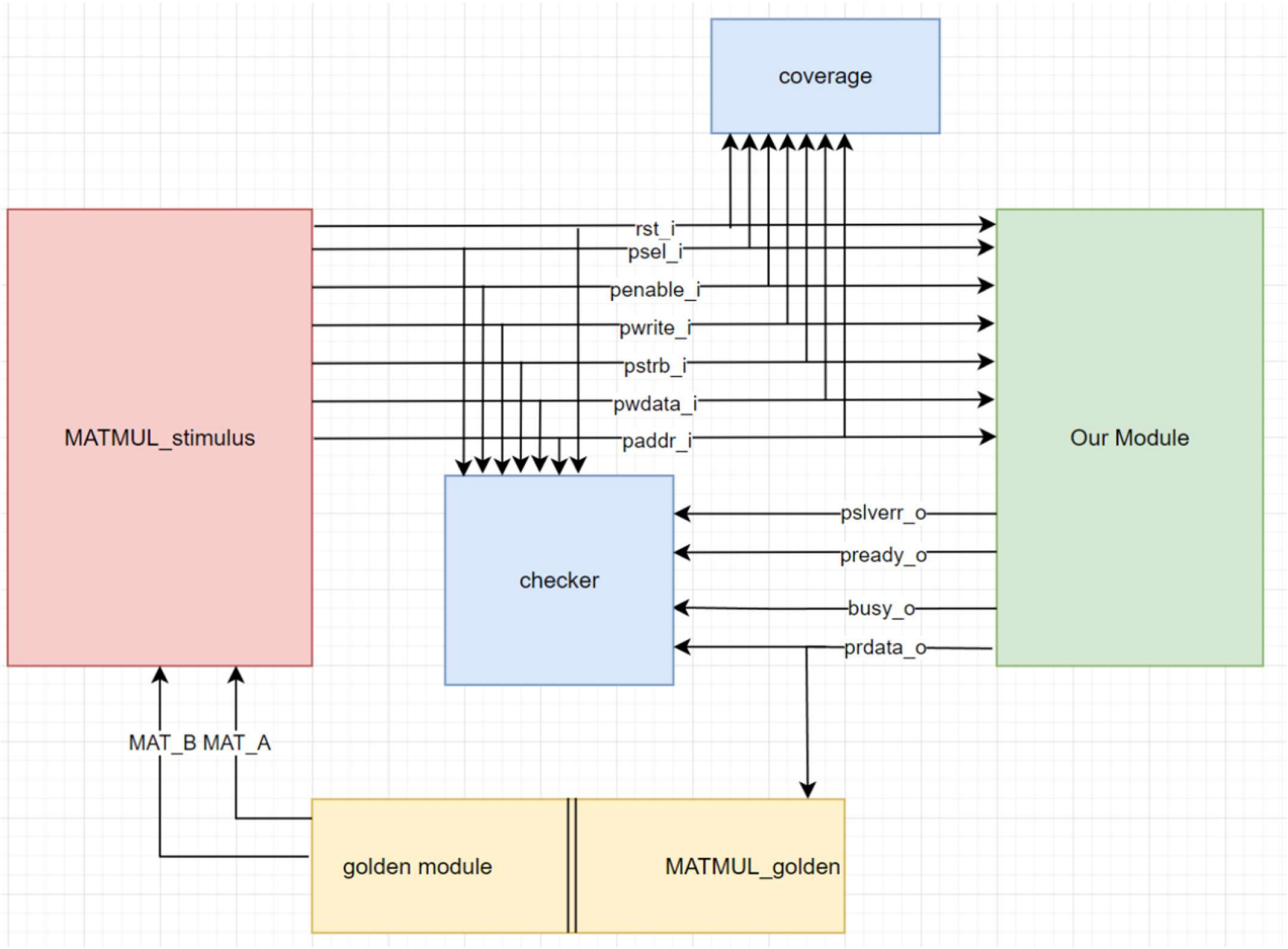
אנו בעצמנו מתחילים את פעולת הכפל ע"י כתיבת 1 לסיבית ההתחלה ברגיסטר הקונטרול.

ניתן לראות בתמונה הבאה את דיאגרמת הבלוקים המתארת את המבנה.



Test Bench Low Level Architecture and Functionality

במודל הבדיקה שלנו יש שני קבצים עיקריים של עבודה stimulus & golden . ב-stimulus אנו מקבלים מהגולדן מודל את שתי מטריצות הכניסה שנוצרים רנדומלית עם איברים בטווחים שתלויים בפרמטרים. ב-golden אנחנו קוראים את שתי מטריצות התוצאה (תוצאה & דגלים) שמחושבים על ידי מודל הזהב בהינתן אותו קלט שנשלח למודל שלנו. מתחילים לעבוד עם המערכת, את ההפעלה נעשה בצורה של הפעלת psel,penale,pwrite ולאחר מכן שליחת ערכים רצויים ב pwdata לכתובת שנקבע ב paddr נוכל גם לקבוע לא לכתוב את כל הערכים בעזרת pstrb אבל אין חובה לכך.. כאשר סיימנו לכתוב דרך קובץ זה נפעיל גם את פעולת הכפל באמצעות startbit ברגיסטר הבקרה. בנוסף במודל זה נוכל לשלוט על אם נרצה לכתוב עם סטרוב או לא בעזרת הכניסה המתאימה, וכמובן גם אם לכפול עם מצב אקומולטור או לא בעזרת רגיסטר הבקרה. בקובץ golden אנו נבצע בעיקר קריאה של ערכים ונשווה, נתחיל מקריאת הערכים מהגולדן מודל המקורי של מטריצת התוצאה ומשם בצורה מתוזמנת יחד עם קובץ stimulus נבצע קריאה מהמערכת שלנו. לאחר הקריאה נשווה את ערכי המטריצה המוכפלת עם הערכים הצפויים ובהתאם לתוצאות נדפיס את אחוזי הדיוק של הכפל. בנוס, לבדיקת מטריצת הכפל נבדקת גם מטריצת הדגלים ומושוות גם היא למטריצת גלים מהגולדן מודל ובאותו עיקרון נוכל לראות אחוזי דיוק. את הקריאה נבצע ע"י שינוי psel,penable ומשם נקרא את prdata ונשווה את הערך שמגיע לתוצאות שכבר יש לנו.



Functional Coverage

בחלק זה נבדקות הכניסות שלנו בהתאם לדרישות העבודה של המכפל, ניתן לראות בטבלה המורטת למטה את הכיסוי של כל כניסה וכניסה ותחום הערכים שנבדקים אצלה

FUNCTION	EVENT	COVERAGE POINT	BINS	scenario
Reset	rst_ni	rst_ni	0,1	Regular / Extreme
Select	Posedge clock	psel_i	0,1	Regular / Extreme
Enable	Posedge clock	penable_i	0,1	Regular / Extreme
Write enable	Posedge clock	pwrite_i	0,1	Regular / Extreme
Write strobe	Posedge clock	pstrb_i[MAX_DIM-1:0]	0,1 (for each bit)	Extreme (using strobe every 3rd input)
Data input	Posedge clock	pwdata_i	[0 : 0.25BW-1] [0.25BW : 0.50BW-1] [0.50BW : 0.75BW-1] [0.75BW : BW-1] (BW = 2**BUS_WIDTH-1)	Regular / Extreme
Address	Posedge clock	Paddr_i	[0 : 2^5-1] main addr [0 : 2^7-1] sub addr i [0 : 2^9-1] sub addr j	Regular / Extreme

Test Bench Functional Checkers

בבדיקה הזאת עליינו היה לבדוק נכונות הפלט תחת דרישות קלט. לא יצא לנו לבצע זאת לעומק, רק כמה שהספקנו כשהייתה גישה לשרתים.

אם היינו מבצעים זאת לעומק היינו דואגים לטווחים של פלט בהינתן קלט.

Condition	Expected Result	Scenario
reset_Active	Prdata_o→0	Regular / Extreme
penable_active	prdata_o→[0:2*BUS_WIDTH-1]	Regular / Extreme
Ready_reset	pready_o→0	Regular / Extreme
reset_data_out	prdata_o →0	Regular / Extreme
Reset_error	Pslverr_o->0	Regular / Extreme

Golden module

הגולדן מודל שלנו נכתב בפייתון והוא עושה גנרציה של 2 מטריצות A וB ושומר אותם מייד לקובץ.

את זה הוא עושה בלולאה באורך כלשהו (מספר אלפים למשל) ובכל איטרציה הוא מייצר 2 מטריצות רנדומליות בטווח הביטים שיש ברשותינו שמחושבים מהפרמטרים שנעבוד איתם בתחילת הקוד. (כמובן צריך להתאים עם DUT כמו תמונה ימנית)

```python
BW, DW = 64, 16
MD = int(BW / DW)
if BW == 16:
    int_type = np.int16
elif BW == 32:
    int_type = np.int32
else:  # BW==64
    int_type = np.int64
max_elm = 2 ** (DW - 1)
min_elm = -2 ** (DW - 1)
max_res = 2 ** (BW - 1)
min_res = -2 ** (BW - 1)
A = B = F = np.zeros( shape: (MD, MD), dtype=np.int64)
C = np.zeros( shape: (MD, MD), dtype=np.int64)
init_txt()
```

```python
for i in range(0, 1000):
    A_c = A.copy()  # save A from prev iter to simulate strobe
    B_c = B.copy()  # save B from prev iter to simulate strobe
    A = np.random.randint(low=min_elm, high=max_elm, size=(MD, MD), dtype=np.int64)
    B = np.random.randint(low=min_elm, high=max_elm, size=(MD, MD), dtype=np.int64)
    with open('matrix_A.txt', 'a') as file_A:  # write A to txt
        file_A.write(',').join(map(str, A.flatten())))
        file_A.write('\n')
    with open('matrix_B.txt', 'a') as file_B:  # write B to txt
        file_B.write(',').join(map(str, B.flatten())))
        file_B.write('\n')
```

על מנתן לסמלץ סטרוב, נשמור את מטריצות הכניסה מאיטרציה הקודמת ונשלב אותם אחד על שני. למשל A קודמת משולבת עם A חדשה בצורת לוח שחמט (חצי נשאר מ-A קודמת וחצי מ-A חדשה). הגדרנו שלבדיקה זה יקרה פעם ב-3 איטרציות. (3%)

```python
if i % 3 == 2:  # every 3 iteration apply strobe simulation
    A = strobe_like(A_c, A)  # simulate strobe to be like
    B = strobe_like(B_c, B)  # chess pattern with prev matrix
```

אחרי שהקלט מוכן (וגם שונה ע"י סטרוב) נתחיל את הכפל. הגדרנו שלבדיקה נעשה אקמולציה עם תוצאה קודמת פעם ב-2 איטרציות, מכאן ה-(2%). לאחר מכן נרוץ על תוצאה סופית ו-"נאסוך" דגלים של אוובר/אנדר פלואו ונשים במטר' F.

לבסוף נחתוך את הביטים שאמורים להמחק אילו היו במכונה ככה שנדמה פעולה של מכפל חומרתי.

```python
C = (i % 2) * C + np.dot(A, B)
F = np.where((C > max_res) | (C < min_res), 1, 0)
C = C.astype(int_type)
```

לסיום נשמור את התוצאות F&C בטקסט באותה דרך כפי שעשינו עם B&A.

```python
with open('matrix_C.txt', 'a') as file_C:
    file_C.write(',').join(map(str, C.flatten())))
    file_C.write('\n')
with open('matrix_F.txt', 'a') as file_f:
    file_f.write(',').join(map(str, F.flatten())))
    file_f.write('\n')
```

כל המטריצות בטקסט נראות ככה:

```
-14,-30961,-16748,18885,-30291,-26266,10531,-25397,-16669,4367,-5423,15879,-25483,-25300,1215,20561
-32222,-32183,26971,-12315,12455,19322,27254,23347,29837,-12781,21465,18880,-1796,-15974,25674,-27738
12765,26613,28539,19738,6175,30549,-22148,-24186,-22381,29283,1110,-22894,-31673,-9188,-26066,-14864
-13086,32084,-609,2385,30160,-1740,-26275,-21335,13740,21460,27650,6325,18425,7978,25820,26473
32693,32301,28794,26580,-98,11279,-6822,-21532,-19950,18602,30283,9509,31313,19122,-1239,-1033
-14657,4352,25824,-27902,8008,8572,-10977,-19358,12831,-28521,-32115,1712,-13374,-4817,-27691,-29095
-31691,18577,13608,-32202,-23432,-2831,-14463,-10575,-23496,-25951,-13623,9782,-26209,-10545,-15749,28588
-10998,-23383,6768,-21001,-30943,-13388,-32022,10455,29293,1530,25273,1484,-20416,-8381,30859,-313
23376,-8192,18855,1137,-25558,21945,-25028,-15532,822,-12304,-32657,10320,20804,-22640,-8516,29296
```

הערה: כאשר רוחב באס הוא 64 ניתקל בבעיה שמקורה בפיתון והיא הגבלה של 64 ביט. כלומר לא נוכל תמיד לתפוס דגל. בעיה זאת לא קיימת עם רוחב פחות מ-64. לכן אם נקבל שגיאה במקרה כזה, נגיד שמקורה בחישוב לא נכון של C על ידי פיתון ולא המודל שלנו.

Verification Results- Golden Model Comparison

הרצנו את הבדיקה על עשרות אלפי מטריצות באקסטרים (מספרים שליליים, עצומים, הפעלת סטרוב ואקומולטור,מימדים שונים).
נדגים כמה תוצאות שונות, ליד כל תמונה נגדיר מה הפרמטרים ומה בדקנו.
הבדיקה נעשית דרך קריאה של קבצים והשוואה מול מודל זהב ומוצגת בצורה אינפורמטיבית, מראה את מספר האיטרציה, השווה איבר מול איבר (רצוי כנגד מצוי) , מציגה את כמות האיברים הנבדקו באיטריציה אחרונה וגם כל הסימולציה יחד ומחשבת את סיכוי הפגיעה באיברים ובדגלים בנפרד. לשם הדגמה הרצנו 1000 איטרציות ככה שנבדקו 17000 או 5000 איברים בסימולציה שלמה.
(17 עבור מטריצה 4x4 , 5 עבור מטריצה 2x2 , הבדיקה הנוספת היא ביטים של דגלים)

BUS, DATA, DIM = 64, 32, 2	BUS, DATA, DIM = 32, 16, 2	BUS, DATA, DIM = 64, 16, 4
כאן רואים מספרים בגדלים עצומים כלל שניתן, כמו כן יש שגיאה בדגלים שמקורה בפיתון שדיברנו עליה, זה רק מראה שיש רגישות לזיהוי שגיאות בדגלים.	כאן בניגוד למקרה מימין הורדנו את רוחב האיבר, ורואים שיש קריאה והשוואה של דגלים שונים אם ישנם. אחרי 1000 הרצות לא נמצאה אף טעות.	כאן רואים מטריצה בגדול 4 עם ערכים גדולים ככל הניתן לקבל בממד 4. אחרי 1000 הרצות לא נמצאה אף טעות.

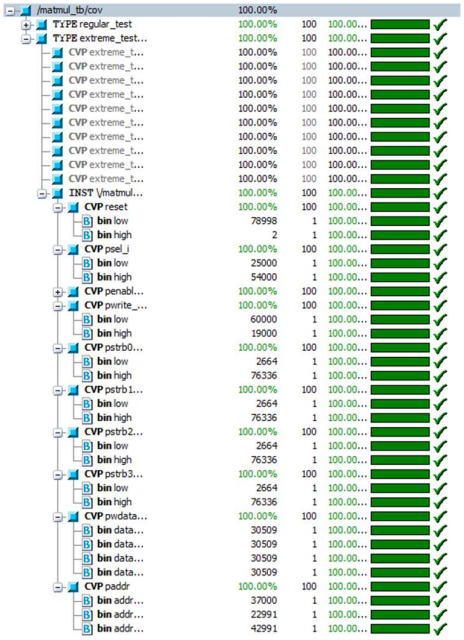
`# ==========>Starting iteration number: 998<==========`
`# HIT!=> expected vs output: [ 2150270123747869283: 2150270123747869283]`
`# HIT!=> expected vs output: [ 3702639756424807702: 3702639756424807702]`
`# HIT!=> expected vs output: [ 1711257025869666961: 1711257025869666961]`
`# HIT!=> expected vs output: [ -8993253641252701 66: -8993253641252701 66]`
`# HIT!=> FLAGS: expected vs output: [ 0: 0]`
`# LAST RUN HIT RATE: 100.00%`
`# WHOLE SIM HIT RATE: 100.00%`
`# TOTAL FLAG HIT RATE: 99.00%`
`# TOTAL VALUES CHECKED: 4990`
`#`
`# ==========>Starting iteration number: 999<==========`
`# HIT!=> expected vs output: [ -6843284648407792 88: -6843284648407792 88]`
`# HIT!=> expected vs output: [ 1012050262862735427: 1012050262862735427]`
`# HIT!=> expected vs output: [ -1144390036763480554: -1144390036763480554]`
`# HIT!=> expected vs output: [ -1898196645062650 67: -1898196645062650 67]`
`# HIT!=> FLAGS: expected vs output: [ 0: 0]`
`# LAST RUN HIT RATE: 100.00%`
`# WHOLE SIM HIT RATE: 100.00%`
`# TOTAL FLAG HIT RATE: 99.00%`
`# TOTAL VALUES CHECKED: 4995`
`#`
`# ==========>Starting iteration number: 1000<==========`
`# HIT!=> expected vs output: [ 1512662891439382 93: 1512662891439382 93]`
`# HIT!=> expected vs output: [ -8194790980507775 60: -8194790980507775 60]`
`# HIT!=> expected vs output: [ -2804613347593436392: -2804613347593436392]`
`# HIT!=> expected vs output: [ -2045169968130495489: -2045169968130495489]`
`# HIT!=> FLAGS: expected vs output: [ 0: 0]`
`# LAST RUN HIT RATE: 100.00%`
`# WHOLE SIM HIT RATE: 100.00%`
`# TOTAL FLAG HIT RATE: 99.00%`
`# TOTAL VALUES CHECKED: 5000`
`# ====> Stopped after 1000 calculations <====`
	`# ==========>Starting iteration number: 994<==========`
	`# HIT!=> expected vs output: [ -2020579052: -2020579052]`
	`# HIT!=> expected vs output: [ 1432021453: 1432021453]`
	`# HIT!=> expected vs output: [ -921223428: -921223428]`
	`# HIT!=> expected vs output: [ 355856643: 355856643]`
	`# HIT!=> FLAGS: expected vs output: [ 1: 1]`
	`# LAST RUN HIT RATE: 100.00%`
	`# WHOLE SIM HIT RATE: 100.00%`
	`# TOTAL FLAG HIT RATE: 100.00%`
	`# TOTAL VALUES CHECKED: 4970`
	`#`
	`# ==========>Starting iteration number: 995<==========`
	`# HIT!=> expected vs output: [ 331422003: 331422003]`
	`# HIT!=> expected vs output: [ -1259835897: -1259835897]`
	`# HIT!=> expected vs output: [ 227980272: 227980272]`
	`# HIT!=> expected vs output: [ -401622351: -401622351]`
	`# HIT!=> FLAGS: expected vs output: [ 0: 0]`
	`# LAST RUN HIT RATE: 100.00%`
	`# WHOLE SIM HIT RATE: 100.00%`
	`# TOTAL FLAG HIT RATE: 100.00%`
	`# TOTAL VALUES CHECKED: 4975`
	`#`
	`# ==========>Starting iteration number: 996<==========`
	`# HIT!=> expected vs output: [ 838723086: 838723086]`
	`# HIT!=> expected vs output: [ 1906422734: 1906422734]`
	`# HIT!=> expected vs output: [ 812124750: 812124750]`
	`# HIT!=> expected vs output: [ -872883136: -872883136]`
	`# HIT!=> FLAGS: expected vs output: [ 2: 2]`
	`# LAST RUN HIT RATE: 100.00%`
	`# WHOLE SIM HIT RATE: 100.00%`
	`# TOTAL FLAG HIT RATE: 100.00%`
	`# TOTAL VALUES CHECKED: 4980`
		`# ==========>Starting iteration number: 999<==========`
		`# HIT!=> expected vs output: [ -142551301: -142551301]`
		`# HIT!=> expected vs output: [ -1235156594: -1235156594]`
		`# HIT!=> expected vs output: [ -421809051: -421809051]`
		`# HIT!=> expected vs output: [ -445631626: -445631626]`
		`# HIT!=> expected vs output: [ -218557383: -218557383]`
		`# HIT!=> expected vs output: [ -132991702: -132991702]`
		`# HIT!=> expected vs output: [ 234634232: 234634232]`
		`# HIT!=> expected vs output: [ 93497390: 93497390]`
		`# HIT!=> expected vs output: [ 240734312: 240734312]`
		`# HIT!=> expected vs output: [ 493401128: 493401128]`
		`# HIT!=> expected vs output: [ 1294142127: 1294142127]`
		`# HIT!=> expected vs output: [ 674079988: 674079988]`
		`# HIT!=> expected vs output: [ -110171823: -110171823]`
		`# HIT!=> expected vs output: [ -1295560687: -1295560687]`
		`# HIT!=> expected vs output: [ -1405512631: -1405512631]`
		`# HIT!=> expected vs output: [ -499012372: -499012372]`
		`# HIT!=> FLAGS: expected vs output: [ 0: 0]`
		`# LAST RUN HIT RATE: 100.00%`
		`# WHOLE SIM HIT RATE: 100.00%`
		`# TOTAL FLAG HIT RATE: 100.00%`
		`# TOTAL VALUES CHECKED: 16983`
		`#`
		`# ==========>Starting iteration number: 1000<==========`
		`# HIT!=> expected vs output: [ -430517572: -430517572]`
		`# HIT!=> expected vs output: [ -1054146392: -1054146392]`
		`# HIT!=> expected vs output: [ -503585316: -503585316]`
		`# HIT!=> expected vs output: [ -945565820: -945565820]`
		`# HIT!=> expected vs output: [ 533691113: 533691113]`
		`# HIT!=> expected vs output: [ 92416199: 92416199]`
		`# HIT!=> expected vs output: [ -364437062: -364437062]`
		`# HIT!=> expected vs output: [ 597109932: 597109932]`
		`# HIT!=> expected vs output: [ -859927235: -859927235]`
		`# HIT!=> expected vs output: [ 276965963: 276965963]`
		`# HIT!=> expected vs output: [ 1463228490: 1463228490]`
		`# HIT!=> expected vs output: [ -111862780: -111862780]`
		`# HIT!=> expected vs output: [ -827412487: -827412487]`
		`# HIT!=> expected vs output: [ -1871766130: -1871766130]`
		`# HIT!=> expected vs output: [ -682068993: -682068993]`
		`# HIT!=> expected vs output: [ -485505048: -485505048]`
		`# HIT!=> FLAGS: expected vs output: [ 0: 0]`
		`# LAST RUN HIT RATE: 100.00%`
		`# WHOLE SIM HIT RATE: 100.00%`
		`# TOTAL FLAG HIT RATE: 100.00%`
		`# TOTAL VALUES CHECKED: 17000`
		`# ====> Stopped after 1000 calculations <====`

לשם הדגמה, נשנה את קובץ טקסט של מטריצת C ונראה שיש גילוי שגיאות:

שגיאה יזומה (expected 111 , 222 , 444)	אותו מקרה אבל ללא שגיאה לשם השווה

`# ==========>Starting iteration number: 1000<==========`
`# ERR!=> expected vs output: [ 111: -469782176]<=========MISS`
`# ERR!=> expected vs output: [ 222: 383554029]<=========MISS`
`# HIT!=> expected vs output: [ -1093215924:-1093215924]`
`# ERR!=> expected vs output: [ 444: 93523274]<=========MISS`
`# HIT!=> FLAGS: expected vs output: [ 0: 0]`
`# LAST RUN HIT RATE: 25.00%`
`# WHOLE SIM HIT RATE: 99.92%`
`# TOTAL FLAG HIT RATE: 100.00%`
`# TOTAL VALUES CHECKED: 5000`
`# ====> Stopped after 1000 calculations <====`
	`# ==========>Starting iteration number: 1000<==========`
	`# HIT!=> expected vs output: [ -469782176: -469782176]`
	`# HIT!=> expected vs output: [ 383554029: 383554029]`
	`# HIT!=> expected vs output: [ -1093215924:-1093215924]`
	`# HIT!=> expected vs output: [ 93523274: 93523274]`
	`# HIT!=> FLAGS: expected vs output: [ 0: 0]`
	`# LAST RUN HIT RATE: 100.00%`
	`# WHOLE SIM HIT RATE: 100.00%`
	`# TOTAL FLAG HIT RATE: 100.00%`
	`# TOTAL VALUES CHECKED: 5000`
	`# ====> Stopped after 1000 calculations <====`

Verification Results- Functional Coverage



/matmul_tb/cov	100.00%		
TYPE regular_test	100.00%	100	100.00...
TYPE extreme_test...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
CVP extreme_t...	100.00%	100	100.00...
INST \/matmul...	100.00%	100	100.00...
CVP reset	100.00%	100	100.00...
bin low	78998	1	100.00...
bin high	2	1	100.00...
CVP psel_i	100.00%	100	100.00...
bin low	25000	1	100.00...
bin high	54000	1	100.00...
CVP penabl...	100.00%	100	100.00...
CVP pwrite_...	100.00%	100	100.00...
bin low	60000	1	100.00...
bin high	19000	1	100.00...
CVP pstrb0...	100.00%	100	100.00...
bin low	2664	1	100.00...
bin high	76336	1	100.00...
CVP pstrb1...	100.00%	100	100.00...
bin low	2664	1	100.00...
bin high	76336	1	100.00...
CVP pstrb2...	100.00%	100	100.00...
bin low	2664	1	100.00...
bin high	76336	1	100.00...
CVP pstrb3...	100.00%	100	100.00...
bin low	2664	1	100.00...
bin high	76336	1	100.00...
CVP pwdata...	100.00%	100	100.00...
bin data...	30509	1	100.00...
bin data...	30509	1	100.00...
bin data...	30509	1	100.00...
bin data...	30509	1	100.00...
CVP paddr	100.00%	100	100.00...
bin addr...	37000	1	100.00...
bin addr...	22991	1	100.00...
bin addr...	42991	1	100.00...

התוצאות של הבדיקה מראות 100% ומראות את כמות השימושים בכל כניסה, למשל אפשר לראות שיש שימוש אקטיבי בסטרוב והוא אחיד כי כפי שציינו מקדום אנחנו עשינו צורה של שחמט כל מטריצות. הערה: במקרה של מטריצה 2x2 נקבל אחוז קטן יותר עקב אי שימוש בשני ביטי סטרוב העליונים. מצורף צילום מסך של הרצה מטריצה 4x4 עם רוחב באס 64 **באקסטרים**. התוצאות של הרצה רגילה היא יותר פשוטה כי לי משתמשים בה בסטרוב והיא תת קבוצה של אקסטרים בכל מקרה, בשני המקרים מקבלים 100%.

Verification Results- Functional Checker

לא עשינו הרבה כאן, רק את הדברים הבסיסיים, ואלו התוצאות. אילו היו אמצעים היינו מנסים לשדרג את הבדיקה.

ניסינו להוציא REPORT אבל מודפס קובץ ריק.

/matmul_tb/check/cover__reset_active	SVA	✓	Off	1	1	Unli...	1	100%	0	0	0 ns	0
/matmul_tb/check/cover__reset_data_out	SVA	✓	Off	1	1	Unli...	1	100%	0	0	0 ns	0
/matmul_tb/check/cover__reset_ready	SVA	✓	Off	1	1	Unli...	1	100%	0	0	0 ns	0
/matmul_tb/check/cover__reset_err	SVA	✓	Off	1	1	Unli...	1	100%	0	0	0 ns	0
/matmul_tb/check/cover__penable_active	SVA	✓	Off	1999	1	Unli...	1	100%	0	0	0 ns	0

Verification Results- Code coverage

בהרצה של כיסוי קוד ראינו כיסוי גדול, הקוד לא נכנס למקומות שמלכתחילה לא אמורים להיכנס אליהם ולכן האחוזים שמתקבלים הם לא 100. צירפנו REPORT לתיקיה של פרויקט שנעלה.

Formal Checker Excluded & unresolved rules

1) File header does not match the defined template – ALL MODULES -**BONUS?**
2) Use a separate line for each HDL statement – ALL MODULES
3) Module has comments density which is below the acceptable minimum ~~(50%)~~ -> (40%)
4) Do not use disallowed character: HT (decimal 9) (TAB)

שלושתם באישור של מייק.

5) Unconnected signals – **only on SYSTOLIC "loose ends"**

בגלל שימוש ביחידות PE, תמיד יישארו חיבורים לא מחוברים ביחדות שנמצאות בצד שמאל או למטה כי לא נצטרך את הפרופוגציה הלאה. אספנו את כל החיבורים לחיבור אחד כדי שבמקום כמה שגיאות נקבל רק בודדת ואותה נשאיר ונתעלם.

6) Nesting at an assignment statement exceeds the maximum of 3/5

```
if (done_i) begin
  for (i = 0; i < DIM; i = i + 1) begin
  for (j = 0; j < DIM; j = j + 1) begin
      case (control[5:1])
```

בגלל שכל הפרויקט עוסק במהותו עם מערכים דו ממדיים, ריבוי קייסים, פרמטריזציה, קשה מאוד ולא פרקטי, שלא להשתמש בקוד שכולל שימושים ב-if/case שנמצאים בתוך לולאה כפולה (דוגמא בצד) לכן ביטלנו את ההזהרה.

7) Avoid using hard coded numeric values such as "[4:0]" for specifying ranges of the multi-bit operand "address" – **only on RegisterFile**

בוטל רק עבור סיגנל של address/subadress . מאחר ומרחב כתובות הוא בהכרח 9 ביטים, כי נתון לפי טבלה שבמקרה הגרוע שבו spn=4 שיושב בכתובת 28, או במקרה שבו spn=1 שיושב בכתובת 16, בכל מקרה נצתרך 5 ביטים לכתובת ראשית , ועבור כתובת-משנה i,j כל אחד log2(MAX_DIM=4) , כלומר 2 ביטים, סה"כ נדרש 9 ביטים ומיקום הביטים הינו קבוע ולא משתנה.

8) Dimension/Range definition "[DIM*i+j +: 1]", for "data_out ", does not comply to descending order convention – **only on RegisterFile**

אין פה באמת ריצה בסדר הפוך כל המידע מועבר כמטריצה שטוחה בסדר רגיל. כנראה הנוטציה ":+" בשילוב אינדקס רץ מבלבל את ה-checker. כנראה כאשר i=j=0 הוא רואה **"[0+: 1]"** וחושב שהסדר של הביטים הפוך.

9) Bit widths differ on left (32) and right (33) of assignment. – **only on loops**

```
63   for (i = 0; i < DIM; i = i + 1) begin
64      for (j = 0; j < DIM; j = j + 1) begin
```

מדובר על אינדקס הלולאה בלבד ולא בסיגנל.

10) FSM states should not be Hardcoded

לא מובן מה הסיבה לשגיאה, ניסינו לשנות את משתנה state לפרמטר/פרמטר לוקאלי ותמיד יש שגיאה אחרת.

11) Net 'paddr_i[15:9]' is unused. -**APV_SLAVE only**
12) Input port 'paddr_i[15:9]' is never used (read from) -**APV_SLAVE only**

הביטים לא בשימוש הפנימי של קובץ הגיסטרים. להבנתינו הם שמורים עבור מכשירים אחרים שמנהל הסלייב, מלבד המכפל.

13) Module undefined.

אם לעשות include השגיאה נעלמת , אבל נאמר לנו שאסור לעשות זאת. ללא include מקבלים את השגיאה הזאת אבל הכל עדיין עובד.