

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ

Φίλτρο Μέσης Τιμής Για Την Επεξεργασία Εικόνας
2ο Μέρος

Διδάσκων

Συρακούλης Γεώργιος

Ομάδα 24

Λαζαρίδου Νίνα AM:57260

Μούσλεχ Στυλιανός Γεώργιος AM:57382

Εισαγωγή

Στο πρώτο μέρος της εργασίας υλοποιήσαμε ένα συγκεκριμένο αλγόριθμο επεξεργασίας εικόνας και τον βελτιστοποιήσαμε αυξάνοντας την απόδοση του αλγορίθμου. Θεωρήσαμε άπειρη μνήμη με άμεση πρόσβαση σε αυτή.

Στο δεύτερο μέρος της εργασίας, θα εισάγουμε εμείς τα είδη, τα μεγέθη και την ταχύτητα της μνήμης, δημιουργώντας δικές μας ιεραρχίες. Θα αναλύσουμε διαφορετικές περιπτώσεις και θα τις συγκρίνουμε ως προς το μέγεθος και τα συνολικά cycles του επεξεργαστή.

Αναφορικά με τα μεγέθη των μνημών προσπαθήσαμε να διαλέγουμε μεγέθη που είναι δυνάμεις του 2 καθώς είναι πιο συνηθισμένα μεγέθη για να γίνεται καλύτερη χρήση του address bus.

Η έκδοση του αλγορίθμου που χρησιμοποιούμε σε αυτή την εργασία είναι η βέλτιστη που προέκυψε από το πρώτη εργασία.

Αρχικά στις πρώτες 5 εκδόσεις κάναμε μια ανάλυση της ταχύτητας εισάγοντας μετά την πρώτη έκδοση μια μικρή και γρήγορη SRAM , την οποία μεγαλώναμε ανά έκδοση.

Αφού αναλύσαμε τα αποτελέσματα, στην συνέχεια κάναμε συγκρίσεις αναφορικά με την ταχύτητα της γρήγορης αυτής SRAM καθώς και με το bus size των μνημών μας.

Τέλος , κάναμε μια ανακεφαλαίωση των συμπερασμάτων μας με βάση τα αποτελέσματα μας.

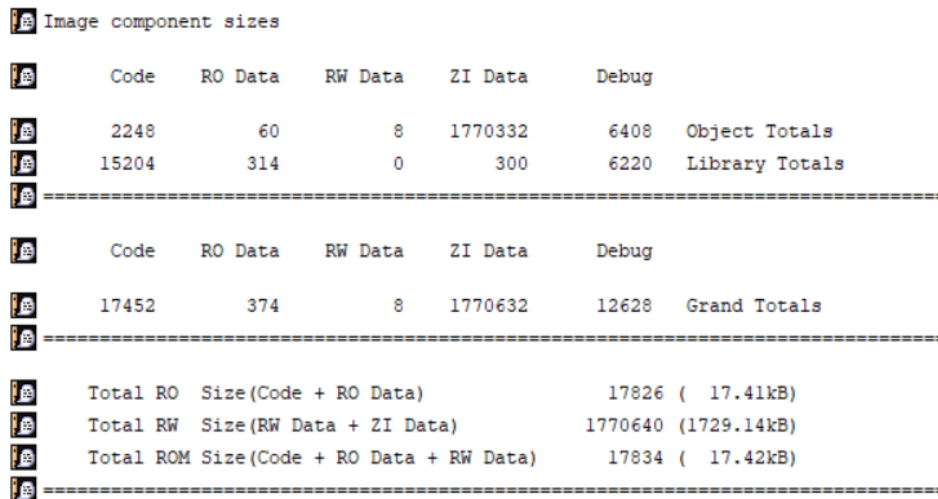
Οι μετρήσεις έγιναν με την εικόνα dog_440x330.yuv και RADIUS=2.

1^η Έκδοση: Αρχική Υλοποίηση

Στην αρχική μας υλοποίηση, θα ακολουθήσουμε την απλή ιεραρχία που εργαστηρίου, βάζοντας μια πολύ γρήγορο ROM και μια πιο αργή DRAM.

(Πολλές φορές η ROM είναι πιο αργή αν όχι το ίδιο γρήγορο με την RAM αλλά για την συγκεκριμένη εργασία θα ακολουθήσουμε τη λογική του εργαστηρίου του μαθήματος).

Αρχικά θα πρέπει να ορίσουμε την μνήμη ROM, η οποία θα περιέχει τον κώδικα, τα RW data και τα RO data. Αυτό το μέγεθος μπορούμε να το δούμε από το make file που παράγει ο Armulator:



	Code	RO Data	RW Data	ZI Data	Debug	
	2248	60	8	1770332	6408	Object Totals
	15204	314	0	300	6220	Library Totals
=====						
	Code	RO Data	RW Data	ZI Data	Debug	
	17452	374	8	1770632	12628	Grand Totals
=====						
Total RO	Size(Code + RO Data)				17826 (17.41kB)	
Total RW	Size(RW Data + ZI Data)				1770640 (1729.14kB)	
Total ROM	Size(Code + RO Data + RW Data)				17834 (17.42kB)	
=====						

Οπότε βλέπουμε ότι χρειαζόμαστε τουλάχιστον 17.42kB.

Αναφορικά με την DRAM, εκεί θα βρίσκονται τα υπόλοιπα data μας, δηλαδή τα ZI Data και οι υπόλοιπες global μεταβλητές καθώς και το stack/heap για local μεταβλητές και για δυναμική δέσμευση μνήμης. Από το παραπάνω make file βλέπουμε ότι το RW size μας είναι 1729.14kB και χρειαζόμαστε επιπλέον αρκετό χώρο για το stack/heap.

Η επιλογή του μεγέθους των μνημών έγινε με βάση τις παραπάνω απαιτήσεις του προγράμματός μας.

Σημειώνεται ότι αντιμετωπίσαμε ένα πρόβλημα σχετικά με την επιλογή του χώρου στην κάθε μνήμη. Πιο συγκεκριμένα, ενώ δίναμε στην μνήμη μέγεθος τουλάχιστον όσο το απαιτούμενο και παραπάνω προέκυπταν errors που συμπεράναμε ότι βασίζονταν στο γεγονός ότι κάποιο τμήμα του stack έβγαινε εκτός μνήμης.

Έπειτα από επικοινωνία με τους υπεύθυνους του εργαστηρίου χρησιμοποιήσαμε αρκετά μεγαλύτερη μνήμη ώστε να αποφύγουμε αυτό το πρόβλημα.

Αυτή η επιλογή εξάλλου δεν επηρεάζει τις συγκριτικές μετρήσεις με τις επόμενες εκδόσεις.

Τέλος βλέποντας τις local μεταβλητές που έχουμε φροντίσαμε η απόσταση stack heap να είναι αρκετή.

Η αρχική μας υλοποίηση λοιπόν :

Memory Map

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 08000000 DRAM 4 RW 250/50 250/50
```

Scatter File

```
ROM 0x0 0x00080000
{
ROM 0x0 0x00080000
{
*.o (+R0)
}
DRAM 0x00080000 0x08000000
{
* (+RW,+ZI)
}

}
```

Stack File

```
#include <rt_misc.h>

__value_in_regs struct __initial_stackheap __user_initial_stackheap(
unsigned R0, unsigned SP, unsigned R2, unsigned SL){

struct __initial_stackheap config;

//config.heap_limit = 0x00724B40;

//config.stack_limit = 0x00004000;
//config.stack_limit = 0x00100000;

/* works */
config.heap_base = 0x00260000;
config.stack_base = 0x00460000;

/* factory defaults */
//config.heap_base = 0x00060000;
//config.stack_base = 0x00080000;
return config;}
```

Τα αποτελέσματα αυτής της εκτέλεσης:

Version	Name	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True idle Cycles
1	nosram	51849073	73243284	54769314	14343004	10377603	0	89815440	169305361	2180260

2^η Έκδοση: Εισαγωγή SRAM για Μεταβλητές

Στην δεύτερη υλοποίηση εισάγουμε μικρή και γρήγορη μνήμη SRAM για να τοποθετήσουμε τις global μεταβλητές που χρησιμοποιούνται συχνά. Πιο συγκεκριμένα τοποθετούμε τις παρακάτω μεταβλητές:

```
#pragma arm section zidata="vars"
int x,y,xx,yy,sum,point;
int tli, tlj, rbi, rbj;
double pixels_inKernel;
#pragma arm section

#pragma arm section rwdatas="othervars"
int imgWidth=M;
int imgHeight=N;
#pragma arm section
```

Έχουμε 12 int μεταβλητές των 4 byte και 1 double μεταβλητή των 8 byte, Άρα χρειαζόμαστε τουλάχιστον 56 byte SRAM. Εισάγουμε λοιπόν 128byte SRAM και κρατάμε τα υπόλοιπα ίδια:

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 08000000 DRAM 4 RW 250/50 250/50
08080000 00000080 SRAM 4 RW 30/15 30/15
```

Τα αποτελέσματα αυτής της εκτέλεσης:

Version	Name	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True idle Cycles
2	varsram	51849100	73243320	54769345	14343007	10377605	0	85785865	165275822	2180260

Βλέπουμε μια σαφή βελτίωση στους κύκλους σε σχέση με την πρώτη έκδοση

Στις επόμενες εκδόσεις επειδή έτσι και αλλιώς αλλάζουμε τάξη μεγέθους στην SRAM, κρατάμε τις μεταβλητές μέσα στην SRAM.

3^η Έκδοση: Αύξηση Μεγέθους SRAM Για Εισαγωγή Ενός Πίνακα Δεδομένων

Στην τρίτη υλοποίηση έγιναν 3 υπό εκδόσεις. Σε αυτήν την υλοποίηση θέλουμε να εισάγουμε έναν από τους πίνακες δεδομένων που έχουμε στον αλγόριθμό μας. Στον τελικό μας αλγόριθμο είχαμε τους παρακάτω 3 πίνακες:

- image_Y : Ο πίνακας της αρχικής εικόνας που διαβάζουμε (padded)
- integrallmg: Ο βοηθητικός πίνακας summed-area table.
- newlmg: Η τελική μας εικόνα μετά την εφαρμογή του φίλτρου

Μπορούμε θεωρητικά να δούμε ποιος πίνακας μας συμφέρει να μπει, βλέποντας σε ποιον πίνακα κάνουμε τις περισσότερες προσπελάσεις. Αυτό το υπολογίσαμε στην προηγούμενη εργασία ως εξής:

	BEST VERSION
Image_Y	$2NM + 6(N+M) + 20$
integrallmg	$8NM + 15(N+M) + M + 61$
newlmg	$2NM$
TOTAL	$12*N*M + 21(N+M) + M + 81$

Οπότε είναι πιο συμφέρον να μπει στην γρήγορη μνήμη ο integrallmg.

Πειραματικά δοκιμάσαμε και τους 3 για να δούμε αν συμπίπτει το θεωρητικό με το πειραματικό αποτέλεσμα.

Οι πίνακες έχουν μέγεθος (για την μεγαλύτερη εικόνα 440x330 και RADIUS=2): 579.28kB , 582.32 kB και 567.18kB αντίστοιχα οπότε εισάγουμε 1MB μνήμης (1024kB). Οπότε το αρχείο memory.map θα γίνει:

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 08000000 DRAM 4 RW 250/50 250/50
08080000 00100000 SRAM 4 RW 30/15 30/15
```

Οπότε έχουμε τις παρακάτω εκδόσεις

- V1_3.1: image_Y
- V1_3.2: newImg
- V1_3.3: integrallmg

Έχουμε τα παρακάτω αποτελέσματα

Version	Name	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True idle Cycles
3.1	image_y	51849100	73243320	54769345	14343007	10377605	0	80858041	160347998	2180260
3.2	newImg	51849100	73243320	54769345	14343007	10377605	0	80994265	160484222	2180260
3.3	integrallmg	51849100	73243320	54769345	14343007	10377605	0	69608297	149098254	2180260

Όπως βλέπουμε γενικά έχουμε καλή μείωση κύκλων σε σχέση με την τις πρώτες 2 εκδόσεις και η καλύτερη είναι όταν βάζουμε τον πίνακα με τις περισσότερες προσπελάσεις όπως υπολογίσαμε θεωρητικά

4^η Έκδοση: Αύξηση Μεγέθους SRAM Για Εισαγωγή Δεύτερου Πίνακα Δεδομένων

Στην τέταρτη υλοποίηση αυξάνουμε το μέγεθος της SRAM ώστε να χωράει 2 πίνακες δεδομένων και βλέπουμε την βελτίωση της απόδοσης που θα έχουμε αν αποθηκεύουμε τους δύο πίνακες με τις περισσότερες προσπελάσεις στην SRAM μνήμη.

Το μέγεθος της SRAM πρέπει να μεγαλώσει και γίνεται 1.5 MB:

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 08000000 DRAM 4 RW 250/50 250/50
08080000 00180000 SRAM 4 RW 30/15 30/15
```

Έχουμε τα παρακάτω αποτελέσματα:

Version	Name	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True idle Cycles
4	2arrays	51849100	73243320	54769345	14343007	10377605	0	64680473	144170430	2180260

Βλέπουμε αναμενόμενη μείωση των total cycles

5^η Έκδοση: Εισαγωγή Στην SRAM Όλων Των Πινάκων Δεδομένων

Στην 5^η υλοποίηση εισάγουμε και τον τρίτο πίνακα δεδομένων στην SRAM. Αυτό μπορούμε να το κάνουμε αυξάνοντας το μέγεθος της SRAM σε 2 MB

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 08000000 DRAM 4 RW 250/50 250/50
08080000 00200000 SRAM 4 RW 30/15 30/15
```

Παρακάτω τα αποτελέσματα:

Version	Name	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True idle Cycles
5	3arrays	51849100	73243320	54769345	14343007	10377605	0	59888873	139378830	2180260

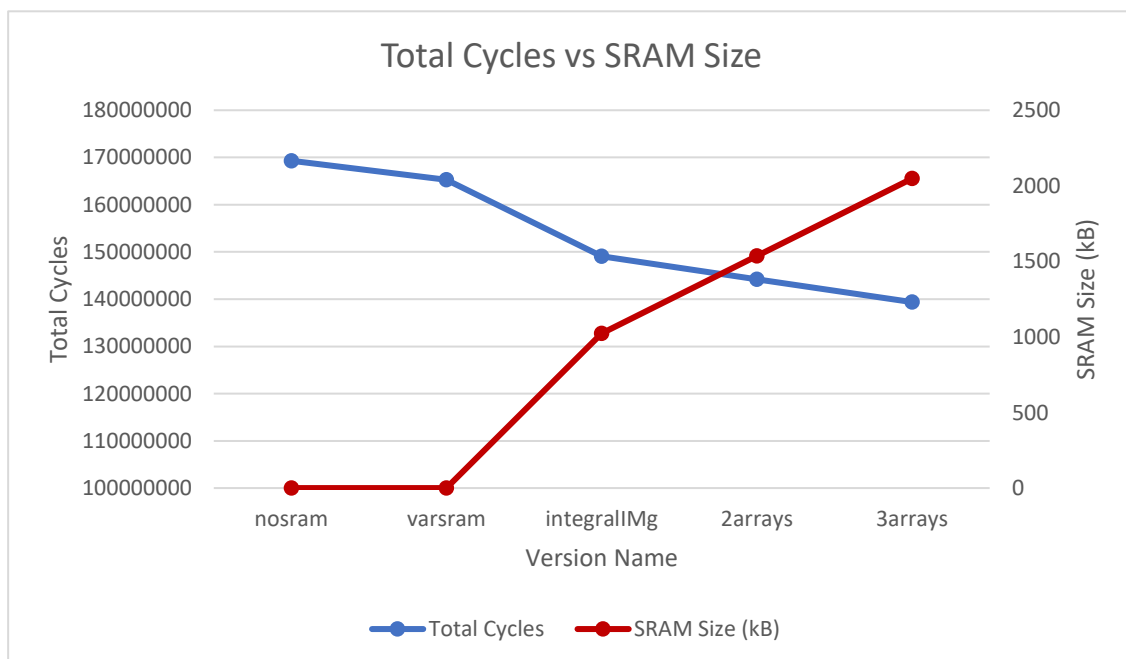
Και πάλι έχουμε μείωση των συνολικών κύκλων

Ανάλυση Αποτελεσμάτων-Συγκρίσεις Μεγέθους SRAM

Έχοντας μεγαλώσει την SRAM σε βαθμό που χωράει τα σημαντικότερα δεδομένα από τον αλγόριθμο μας, αξίζει να δούμε συγκριτικά μερικά αποτελέσματα για να καταλήξουμε σε κάποια γενικότερα συμπεράσματα.

Ο πίνακας με όλους τους κύκλους και το μέγεθος της SRAM ανά έκδοση:

Version	Name	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True idle Cycles	SRAM Size (kB)
1	nosram	51849073	73243284	54769314	14343004	10377603	0	89815440	169305361	2180260	0
2	varsram	51849100	73243320	54769345	14343007	10377605	0	85785865	165275822	2180260	0.125
3.3	integralIMg	51849100	73243320	54769345	14343007	10377605	0	69608297	149098254	2180260	1024
4	2arrays	51849100	73243320	54769345	14343007	10377605	0	64680473	144170430	2180260	1536
5	3arrays	51849100	73243320	54769345	14343007	10377605	0	59888873	139378830	2180260	2048

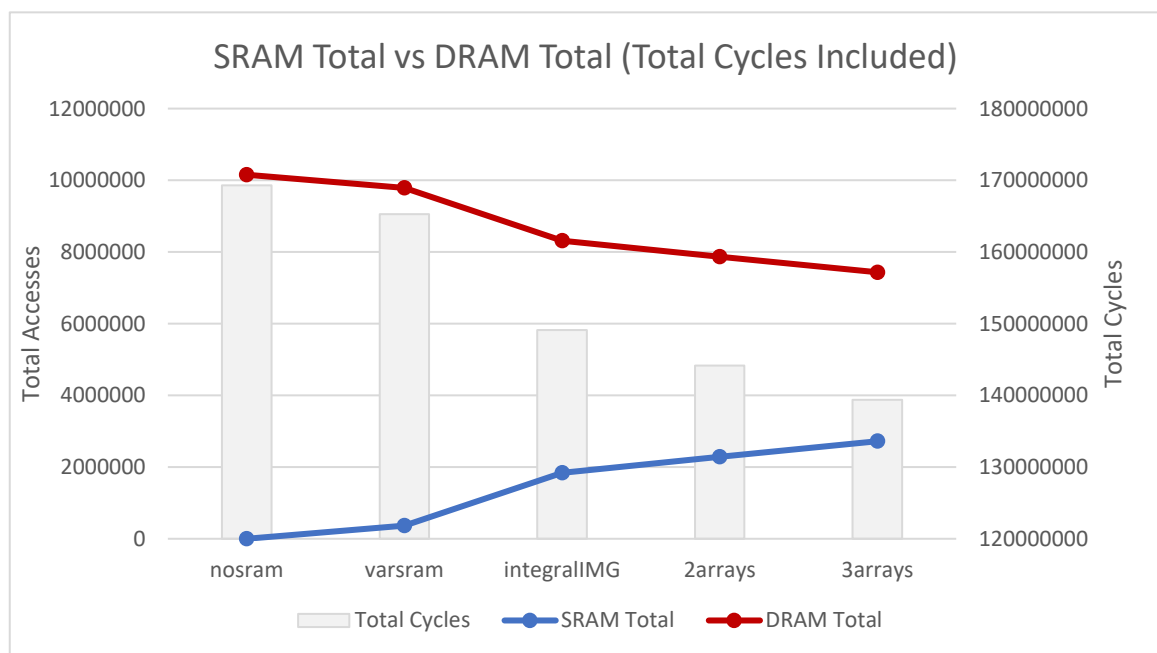


Παρατηρούμε βλέποντας το παραπάνω γράφημα ότι όσο αυξάνουμε το μέγεθος της SRAM και κατά συνέπεια τον αριθμό των δεδομένων που μπορούμε να προσπελάσουμε πιο γρήγορα, πέφτουν οι συνολικοί κύκλοι (και τα wait states) του προγράμματος. Βέβαια η αύξηση του μεγέθους μιας γρήγορης μνήμης κοστίζει όποτε έχουμε ένα trade-off ταχύτητας-μεγέθους που μεταφράζεται σε trade-off ταχύτητας-τιμής.

Επιπλέον για κάθε έκδοση, μπορούμε να δούμε τα reads/writes που κάνουμε σε κάθε μνήμη μέσω του AXD (Debugger Internals -> Internal Variables -> \$memstats):

Version	Name	ROM		SRAM				DRAM			
		N Reads	S Reads	N Reads	S Reads	N Writes	S Writes	N Reads	S Reads	N Writes	S Writes
1	nosram	7392403	51565200	-	-	-	-	3403561	3547040	1310717	1893397
2	varsram	7392406	51565231	38614	0	327711	0	3364947	1310717	3219329	1893397
3.3	integralIMG	7392406	51565231	1063190	0	773823	0	2340371	1310717	2773217	1893397
4	2arrays	7392406	51565231	1214582	0	1070415	0	2188979	1310717	2476625	1893397
5	3arrays	7392406	51565231	1359782	0	1360815	0	2043779	1310717	2186225	1893397

Είναι προφανές ότι θέλουμε να έχουμε περισσότερα reads/writes στην SRAM που είναι πιο γρήγορη απ' ότι στην DRAM . (Τα νούμερα της ROM δεν αλλάζουν καθώς δεν αλλάζουμε το τι αποθηκεύεται εκεί από έκδοση σε έκδοση).



Δοκιμαστικές Εκδόσεις

Έχοντας αναλύσει την επίδραση που έχει η εισαγωγή μιας πιο γρήγορης μνήμης στον κώδικα και συγκεκριμένα το trade-off μεγέθους-ταχύτητας, θα δοκιμάσουμε μερικές εκδόσεις όπου θα συγκρίνουμε πως επηρεάζουν άλλα χαρακτηριστικά της μνήμης την απόδοση του προγράμματος και πιο συγκεκριμένα:

- Σύγκριση μιας ακόμα πιο γρήγορης SRAM που χωράει μόνο 1 πίνακα δεδομένων σε σχέση με τις υπόλοιπες υλοποιήσεις με 1 και παραπάνω πίνακες.
- Σύγκριση της επίδρασης του bus size στο αποτέλεσμα.

6^η Έκδοση-Δοκιμαστική: Χρήση Γρηγορότερης SRAM

Σε αυτήν την δοκιμαστική έκδοση θέλαμε να δούμε την επίδραση της χρήσης μιας ακόμα πιο γρήγορης αλλά μικρότερης SRAM σε σχέση με τις υπόλοιπες που υλοποιήσαμε.

Οπότε θα κάνουμε μια μνήμη 1 MB με ταχύτητες 10/5 10/5 αντί για 30/15 30/15

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 08000000 DRAM 4 RW 250/50 250/50
08080000 00100000 SRAM 4 RW 10/5 10/5
```

Συγκριτικά έχουμε τα εξής αποτελέσματα:

Version	Name	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True idle Cycles	SRAM Size (kB)
1	nosram	51849073	73243284	54769314	14343004	1E+07	0	89815440	169305361	2180260	0
2	varsram	51849100	73243320	54769345	14343007	1E+07	0	85785865	165275822	2180260	0.125
3.1	image_y	51849100	73243320	54769345	14343007	1E+07	0	80858041	160347998	2180260	1024
3.2	newlmg	51849100	73243320	54769345	14343007	1E+07	0	80994265	160484222	2180260	1024
3.3	integralMg	51849100	73243320	54769345	14343007	1E+07	0	69608297	149098254	2180260	1024
4	2arrays	51849100	73243320	54769345	14343007	1E+07	0	64680473	144170430	2180260	1536
5	3arrays	51849100	73243320	54769345	14343007	1E+07	0	59888873	139378830	2180260	2048
6	integralfast	51849100	73243320	54769345	14343007	1E+07	0	6771284	147261241	2180260	1024

Παρατηρούμε, ότι η 6^η έκδοση είναι αρκετά κοντά αλλά λίγο πιο γρήγορη από την 3^η έκδοση (ίδιου μεγέθους αλλά πιο αργή SRAM με τον ίδιο πίνακα μέσα). Παρά ταύτα η 4^η έκδοση που έχει μεγαλύτερο μέγεθος SRAM έχει συνολικά λιγότερους κύκλους από αυτήν την δοκιμαστική.

Παρατηρούμε στη συγκεκριμένη περίπτωση ότι από το να κάνουμε πιο γρήγορη την ήδη πολύ γρήγορη SRAM, είναι προτιμότερο να την μεγαλώσουμε για να φέρουμε σε αυτήν περισσότερα δεδομένα. Αυτό είναι λογικό καθώς η DRAM μας είναι πολύ πιο αργή από την αρχική μας SRAM καθώς το να μεταφέρουμε ένα read/write από χρόνο (ns) 250 ή 50 σε 30 ή 15 είναι προτιμότερο από το να πάμε από 30 ή 15 σε 10 ή 5.

Να σημειώσουμε ότι δεν έχει νόημα να κάνουμε το 10/5 σε 1/1 καθώς με βάση το ρολόι του επεξεργαστή μας ο AXD ήδη θεωρεί 0 cycles αναμονή στην πρώτη περίπτωση.

Memory map:

```
08080000..0817ffff, 32-Bit, wr, wait states: RN=0 \N=0 RS=0 \S=0 RIS=0 \IS=0
00080000..0807ffff, 32-Bit, wr, wait states: RN=12/2 \N=12/2 RS=2 \S=2 RIS=12/2 \IS=12/2
00000000..0007ffff, 32-Bit, -r, wait states: RN=0 \N=Abt RS=0 \S=Abt RIS=0 \IS=0
```

7^η Έκδοση-Δοκιμαστική: Συγκρίσεις Μεγέθους BUS

Το μέγεθος του data bus, καθορίζει τον αριθμό των δεδομένων που μεταφέρονται ανά κάθε πράξη μεταφοράς δεδομένων της μνήμης.

Όσο μεγαλύτερο το data bus, τόσα περισσότερα καλώδια χρησιμοποιούνται σε αυτό και άρα υπάρχουνε μεγαλύτερες απαιτήσεις σε χώρο και κόστος για την μνήμη μας.

Σε αυτές τις δοκιμές αλλάζουμε το μέγεθος του bus ώστε να βγάλουμε συμπεράσματα για την συνεισφορά του στο χρόνο εκτέλεσης. Χρησιμοποιούμε για τις συγκρίσεις την 5η έκδοση των υλοποιήσεων μας δηλαδή την πιο γρήγορη.

Την πρώτη φορά μειώνουμε το bus στα 2 byte για όλες τις μνήμες της 5ης υλοποίησης.

Στην συνέχεια σκεφτήκαμε να δούμε τι συμβαίνει αν υποδιπλασιάσουμε το data bus της γρήγορης SRAM ώστε να κερδίζουμε σε χώρο και σε κόστος μνήμης. Οι άλλες μνήμες παραμένουν ίδιες με της αρχικής 5ης έκδοσης

Τα αποτελέσματα των συγκρίσεων φαίνονται παρακάτω.

Version	Name	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True idle Cycles
5	3arrays(bus 4 byte)	51849100	73243320	54769345	14343007	10377605	0	59888873	139378830	2180260
test_bus1	3arrays (bus 2 byte)	51849100	73243320	54769345	14343007	10377605	0	142853044	222343001	2180260
test_bus2	3arrays(sram2byte)	51849100	73243320	54769345	14343007	10377605	0	62609470	142099427	2180260

Παρατηρούμε λοιπόν ότι άμα υποδιπλασιάσουμε το bus size σε όλες τις μνήμες έχουμε σχεδόν διπλασιασμό των cycles , κάτι που είναι αναμενόμενο.

Όμως βλέπουμε ότι αν κάνουμε μόνο της γρήγορης SRAM το bus size 2 byte αυτό έχει πολύ μικρότερη επίπτωση στα cycles (για τον ίδιο λόγω δεν έχουμε πλήρη υποδιπλασιασμό στην παραπάνω περίπτωση)

Αυτό συμβαίνει καθώς η μνήμη μας είναι πολύ γρήγορη με αποτέλεσμα να έχουμε πάλι τον ίδιο χρόνο για read/writes περίπου σε cycles παρόλο που χρειάζεται ο διπλάσιος καθαρός χρόνος. Όπως βλέπουμε εδώ:

```
Memory map:
00000000..0827ffff, 16-Bit, wr, wait states: RN=1/0 WN=1/0 RS=0 WS=0 RIS=1/0 WIS=1/0
00080000..0807ffff, 32-Bit, wr, wait states: RN=12/2 WN=12/2 RS=2 WS=2 RIS=12/2 WIS=12/2
00000000..0007ffff, 32-Bit, -r, wait states: RN=0 WN=Abt RS=0 WS=Abt RIS=0 WIS=0
```

Καταλήγουμε σε ένα χρήσιμο συμπέρασμα λοιπόν, ότι άμα μπορούμε συγκριτικά με το clock του επεξεργαστή να έχουμε μια πολύ γρήγορη SRAM, μπορούμε να βάλουμε μικρότερο data bus ώστε να γλυτώσουμε κόστος και χώρο χωρίς να έχουμε πολύ χειρότερα αποτελέσματα σε Total Cycles.

Ανακεφαλαίωση Συμπερασμάτων

Μία ιεραρχία μνήμης που συμπεριλαμβάνει 1 επίπεδο μικρότερης και γρηγορότερης μνήμης με τα δεδομένα που κάνουμε συχνότερα access αυξάνει σημαντικά την απόδοση του προγράμματός μας. Όπως προαναφέραμε όσο περισσότερα δεδομένα περιλαμβάνει τόσο μεγαλύτερη βελτίωση έχουμε.

Οστόσο η χωρητικότητά της εξαρτάται από τις ανάγκες του προγράμματός μας αλλά και το κόστος.

Ένας ακόμα παράγοντας που καθορίζει την απόδοση είναι το πόσο γρήγορη είναι μία γρήγορη μνήμη. Εδώ πρέπει να γνωρίζουμε εάν μία SRAM για παράδειγμα χωράει περισσότερα δεδομένα αλλά είναι λίγο πιο αργή ή μία SRAM που χωράει λιγότερα δεδομένα αλλά είναι πιο γρήγορη είναι πιο αποδοτική. Οι δικές μας παρατηρήσεις έδειξαν ότι η πρώτη περίπτωση μας συμφέρει περισσότερο αλλά όπως σχολιάσαμε και πριν αυτό εξαρτάται από τις διαφορές στους χρόνους μεταξύ SRAM και DRAM.

Τέλος, εξετάζουμε την επιρροή του bus size όσον αφορά την ταχύτητα, το μέγεθος της μνήμης και το κόστος. Στο δικό μας παράδειγμα, παρατηρήσαμε ότι μία SRAM με μικρότερο bus size αλλά μεγάλη σχετικά χωρητικότητα θα ήταν ένας καλός συμβιβασμός.

Σημειώνουμε ότι η γρηγορότερη (αλλά και μεγαλύτερη σε μέγεθος) ιεραρχία μνήμης είναι η εξής:

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 08000000 DRAM 4 RW 250/50 250/50
08080000 00200000 SRAM 4 RW 30/15 30/15
```

Όπου στην SRAM έχουμε βάλει και τους 3 πίνακες δεδομένων καθώς και τις global μεταβλητές μας. Είναι δηλαδή η 5^η έκδοση που είδαμε.