2ο μέρος εργασίας του μαθήματος «Σχεδιασμός Ενσωματωμένων Συστημάτων»

Ομάδα 10

Φοιτητές:

Ηλίας Παπαδέας 56989

Χριστόφορος Σπάρταλης 56785

Table of Contents

Εισαγωγή

Αρχικά υπάρχει μία διόρθωση πάνω στο πρώτο μέρος της εργασίας. Έπειτα ακολουθεί μία σύντομη θεωρητική προσέγγιση των ζητημάτων των οποίων άπτεται το δεύτερο μέρος της εργασίας. Στη συνέχεια, γίνεται αναφορά στον βελτιστοποιημένο κώδικα που αναδείχθηκε από το πρώτο μέρος της εργασίας. Ακολούθως υπάρχει το κυρίως κομμάτι της εργασίας, δηλαδή η διερεύνηση πάνω στα ζητήματα που θέτονται. Είναι σημαντικό να ακολουθηθεί από την αρχή και σειριακά η εξελικτική πορεία της σκέψης και των συμπερασμάτων, καθώς μόνο έτσι μπορεί να καταστεί ξεκάθαρος ο τρόπος εργασίας μας, οι εκτιμήσεις και τα συμπεράσματά μας. Τέλος απαντώνται τα δύο ερωτήματα που θέτει το δεύτερο μέρος της εργασίας.

Διόρθωση

Στο πρώτο μέρος της εργασίας θέτουμε ως παράδειγμα RO data τα εξής:

#define N 288 / frame dimension for QCIF format */*
#define M 352 / frame dimension for QCIF format */*
#define filename "*akiyoy.yuv*"
#define finalfilename "*akiyo_high_y.yuv*"

Παρόλα αυτά η αντίληψη ότι τα παραπάνω αποτελούν σταθερές είναι λανθασμένη. Τα παραπάνω δεν αποτελούν σταθερές αλλά είναι μέρος του κώδικα. Όταν το πρόγραμμα «κατεβαίνει» στον επεξεργαστή τα διάφορα βοηθητικά ονόματα που δίνουμε στις εκάστοτε τιμές αντικαθίστανται από αυτές τις τιμές. Για παράδειγμα δεν υπάρχει πουθενά Μ και στη θέση του υπάρχει το 288.

Θεωρητικό Σκέλος

ROM (Read Only Memory)

Οι μνήμες μόνο για ανάγνωση (Read-only memories – ROM) είναι προγραμματισμένες από πριν με σταθερά δεδομένα. Είναι ιδιαίτερα χρήσιμες στα ενσωματωμένα συστήματα δεδομένου ότι σημαντικό μέρος του κώδικα και ενδεχομένως ορισμένα δεδομένα δεν αλλάζουν στο χρόνο.

Η μνήμη ROM είναι μη πτητική, δηλαδή δε χάνει τα δεδομένα της με τη διακοπή τροφοδοσίας του ρεύματος. Γι’ αυτό σ’ αυτήν φορτώνουμε όλα τα δεδομένα τα οποία είναι απαραίτητα έτσι ώστε να πρόγραμμα να εκκινεί πάντα σωστά (π.χ. μετά από κάποιο reset).

Η μνήμη ROM είναι πολύ πιο φθηνή από τη RAM, αλλά συνήθως είναι πιο αργή και επίσης δεν μπορούν όλα τα δεδομένα της να προσπελαστούν απευθείας από την ΚΜΕ.

Οι μνήμες ROM είναι επίσης λιγότερο ευαίσθητες στα σφάλματα που προκαλούνται λόγω ακτινοβολίας. Υπάρχουν διαθέσιμες διάφορες ποικιλίες μνημών ROM.

RAM (Random Access Memory)

Οι μνήμες τυχαίας προσπέλασης μπορούν τόσο να διαβαστούν όσο και να γραφούν. Καλούνται τυχαίας προσπέλασης επειδή, αντίθετα από τους μαγνητικούς δίσκους, οι διευθύνσεις μπορούν να διαβαστούν με οποιαδήποτε σειρά.

Υπάρχουν δύο βασικές κατηγορίες μνήμης τυχαίας προσπέλασης: η στατική RAM (static RAM – SRAM) κατασκευασμένη από transistor και latches και η δυναμική RAM (dynamic RAM – DRAM) κατασκευασμένη από transistor και πυκνωτές. Αυτοί οι δυο τύποι έχουν σημαντικές διαφορές στα χαρακτηριστικά τους.

- Η SRAM είναι γρηγορότερη από τη DRAM.
- Η SRAM καταναλώνει περισσότερη ενέργεια από τη DRAM.
- Μπορούμε να τοποθετήσουμε περισσότερη DRAM σε ένα απλό chip.
- Οι τιμές των δεδομένων στις DRAM πρέπει να ανανεώνονται (refreshed) περιοδικά.

Περιγραφή περιοχών μνήμης

Στη ROM φορτώνουμε τόσο τα RO data και τον κώδικα όσο και τα RW data παρόλο που κατά τη διάρκεια της εκτέλεσης τα RW data θα αντιγραφούν στη μνήμη RAM για προφανείς λόγους (η ROM είναι προορισμένη μόνο για ανάγνωση). Κατά τη διάρκεια της εκτέλεσης στη ROM βρίσκονται οι σταθερές (π.χ. *const char c=123;*) καθώς και οι τιμές στις οποίες αρχικοποιούμε global μεταβλητές (π.χ. το 31 όταν αρχικοποιούμε τη global μεταβλητή *int d=31;*). Επίσης, όπως προείπαμε, βρίσκεται ήδη φορτωμένα τα αρχεία του αντικειμενικού κώδικα και οι βιβλιοθήκες καθώς και κάποια περιεχόμενα του αρχικού κώδικα (π.χ. τιμές τις οποίες εκχωρούμε απευθείας σε μεταβλητές και τιμές οι οποίες ορίζονται με το define)

Στη RAM κατά τη διάρκεια της εκτέλεσης αποθηκεύονται τα zero-initialized data, δηλαδή global μεταβλητές οι οποίες όταν “κατεβαίνουν” στον επεξεργαστή αρχικοποιούνται στο μηδέν, αντίστοιχα εκεί αποθηκεύονται και οι global μεταβλητές οι οποίες αρχικοποιούνται από τον προγραμματιστή (όπως προείπαμε στη RAM αποθηκεύεται μόνο η global μεταβλητή, ενώ η τιμή στην οποία αρχικοποιείται αποθηκεύεται στη ROM). Επίσης στη RAM βρίσκονται και δύο στοίβες η stack στην οποία αποθηκεύονται οι local μεταβλητές και αυξάνεται (γίνεται pop) προς τα κάτω και η heap η οποία

χρησιμοποιείται για τη δυναμική δεύσμευση μνήμης και η οποία αυξάνεται προς τα πάνω.

Memory Map

Λαμβάνοντας υπ’ όψη την κατάταξη των μνημών -όπως παρουσιάστηκε στο θεωρητικό σκέλος- όσον αφορά την ταχύτητα και ανατρέχοντας σε ορισμένα datasheets, καταλήξαμε στους παρακάτω χρόνους ανάγνωσης/ εγγραφής:

- Για την ROM 250 ns χρόνος ανάγνωσης.
- Για την DRAM 150 ns χρόνος ανάγνωσης και 256 ns χρόνος εγγραφής.
- Για την SRAM 55 ns χρόνος ανάγνωσης και χρόνος εγγραφής.

Σημείωση1: Στα memory maps όταν δηλώνουμε τα χαρακτηριστικά και οριοθετούμε το μέγεθος της ROM βάλαμε ως χρόνο εγγραφής σειριακών και μη-σειριακών δεδομένων ίσο με 1. Αυτός ο χρόνος είναι προφανώς φαινομενικός καθώς η ROM δεν επιτρέπει την εγγραφή (δηλώνεται και ως R και όχι RW).

Σημείωση2: Ως εύρος διάυλου ορίσαμε τα 4 Bytes για όλες τις μνήμες.

Βελτιστοποιημένος κώδικας

Για τη διεξαγωγή του δεύτερου μέρους της εργασίας χρησιμοποιείται ο βελτιστοποιημένος κώδικας, με την τεχνική του loop unrolling, για την εφαρμογή υψηπερατού φίλτρου στην επεξεργασία εικόνας. Η τεχνική του loop unrolling αποδείχθηκε η πιο αποδοτική από τις 7τενικές που εφαρμόσαμε στον κώδικα τον οποίον αναπτύξαμε. Παραθέτουμε τα αποτελέσματα του armulator, καθώς είναι χρήσιμα και για το δεύτερο μέρος της εργασίας για εποπτικούς λόγους.

Image Component Sizes:

```
==============================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1228         60          0    1226792       4836   Object Totals
     10196        314          0        300       4548   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11424        374          0    1227092       9384   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11798 (  11.52kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11798 (  11.52kB)
==============================================================================
```

Debugger Internal Statistics:

Debugger Internals

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	9600875	18348737	8882914	7402475	2290557	0	18575946

Διερεύνηση

1. Μέγεθος μνημών ROM και RAM

Το πρώτο πράγμα το οποίο διερευνήσαμε είναι αν το μέγεθος των μνημών επηρεάζει τους συνολικούς κύκλους και με ποιον τρόπο.

Σε όλες τις μετρήσεις που πραγματοποιήσαμε, επικεντρώνοντας σε αυτό το ζήτημα, λάβαμε τα ίδια αποτελέσματα όσον αφορά τα image component sizes

Image Component Sizes:

```
==============================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1260         40          0    1226792       6064   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11388        354          0    1227092      10536   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11742 (  11.47kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11742 (  11.47kB)
==============================================================================
```

1. ROM 0x80000 // RAM 0x8000000

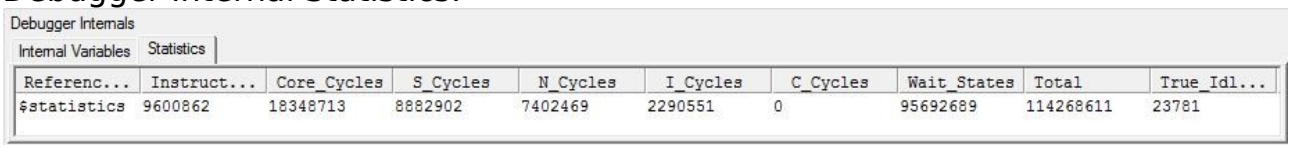
Memory Map:

```
00000000 00080000 ROM 4 R 250/1 250/1
00080000 08000000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x08000000
{
  ROM 0x0 0x80000
  {
    *.o  (+RO)
  }
  RAM 0x80000 0x8000000
  {
    * (+ZI, +RW)
  }
}
```

Debugger Internal Statistics:


Debugger Internals

Internal Variables | Statistics

Referenc...	Instruct...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idl...
$statistics	9600862	18348713	8882902	7402469	2290551	0	95692689	114268611	23781

2. ROM 0x40000 // RAM 0x4000000

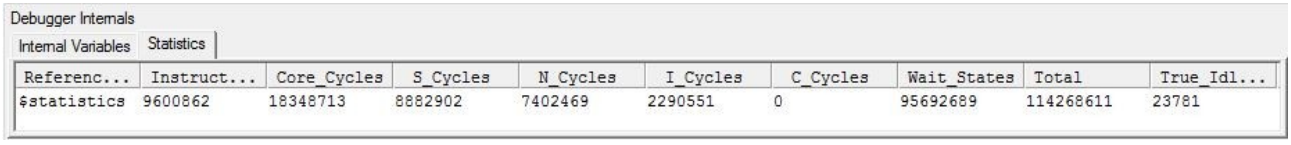
Memory Map:

```
00000000 00040000 ROM 4 R 250/1 250/1
00040000 04000000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x04000000
{
  ROM 0x0 0x40000
  {
    *.o  (+RO)
  }
  RAM 0x40000 0x4000000
  {
    * (+ZI, +RW)
  }
}
```

Debugger Internal Statistics:


Debugger Internals

Internal Variables | Statistics

Referenc...	Instruct...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idl...
$statistics	9600862	18348713	8882902	7402469	2290551	0	95692689	114268611	23781

3. ROM 0x8000 // RAM 0x800000

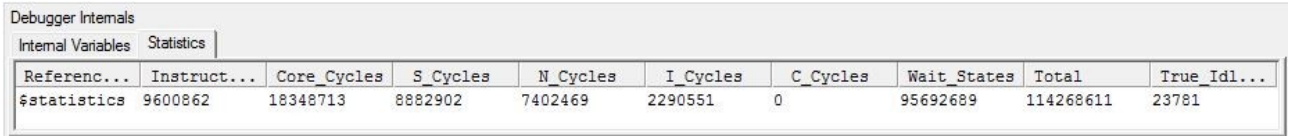
Memory Map:

```
00000000 00008000 ROM 4 R 250/1 250/1
00008000 00800000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x0800000
{
  ROM 0x0 0x8000
  {
    *.o  (+RO)
  }
  RAM 0x8000 0x800000
  {
    * (+ZI, +RW)
  }
}
```

Debugger Internal Statistics:



4. ROM 0x4000 // RAM 0x40000

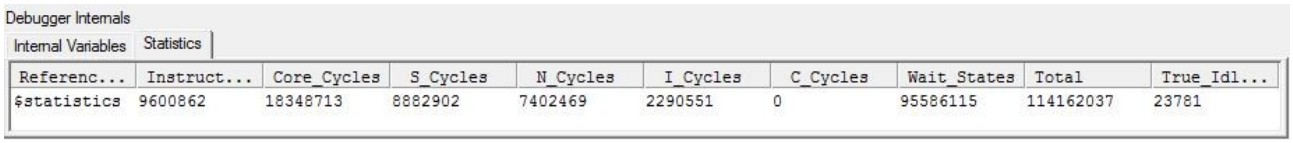
Memory Map:

```
00000000 00004000 ROM 4 R 250/1 250/1
00004000 00400000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x0400000
{
  ROM 0x0 0x4000
  {
    *.o  (+RO)
  }
  RAM 0x4000 0x400000
  {
    * (+ZI, +RW)
  }
}
```

Debugger Internal Statistics:


Debugger Internals

Internal Variables | Statistics

Referenc...	Instruct...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idl...
$statistics	9600862	18348713	8882902	7402469	2290551	0	95586115	114162037	23781

5. ROM 0x3000 // RAM 0x30000

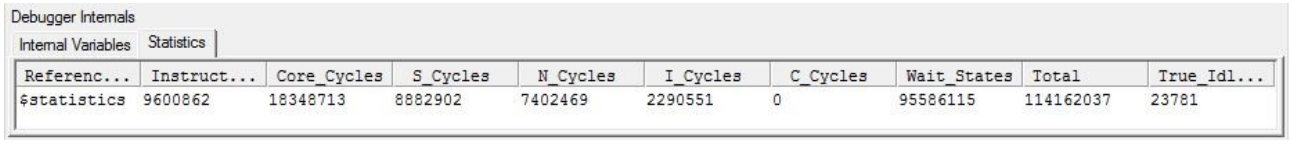
Memory Map:

```
00000000 00003000 ROM 4 R 250/1 250/1
00003000 00300000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x0300000
{
  ROM 0x0 0x3000
  {
    *.o  (+RO)
  }
  RAM 0x3000 0x300000
  {
    * (+ZI, +RW)
  }
}
```

Debugger Internal Statistics:


Debugger Internals

Internal Variables | Statistics

Referenc...	Instruct...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idl...
$statistics	9600862	18348713	8882902	7402469	2290551	0	95586115	114162037	23781

6. ROM_0x2DE0 // RAM_0x12B954

$(2DE0)_{16} = (11744)_{10}$ και $(12B954)_{16} = (1227092)_{10}$

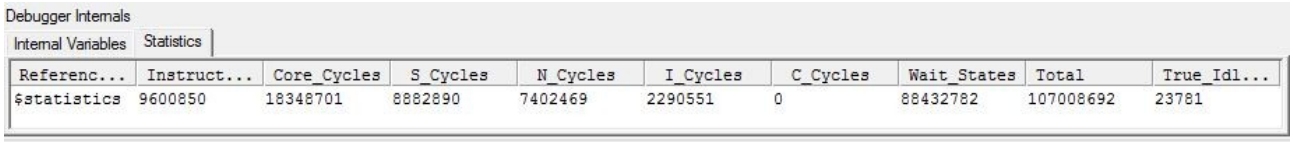
Memory Map:

```
00000000 00002DE0 ROM 4 R 250/1 250/1
00002DE0 0012B954 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x012B954
{
  ROM 0x0 0x2DE0
  {
    *.o  (+RO)
  }
  RAM 0x2DE0 0x12B954
  {
    * (+ZI, +RW)
  }
}
```

Debugger Internal Statistics:


Debugger Internals

Internal Variables | Statistics

Referenc...	Instruct...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idl...
$statistics	9600850	18348701	8882890	7402469	2290551	0	88432782	107008692	23781

Παρατηρήσεις

Αρχικά, όσον αφορά το μέγεθος των δεδομένων, παρατηρούμε ότι τα RO data έχουν μειωθεί σε σχέση με τις αρχικές μετρήσεις (δηλαδή όταν δεν υπήρχε διαχείριση μνημών).

Έπειτα, όσον αφορά τους κύκλους, παρατηρούμε ότι τα wait states και τα total cycles παραμένουν σταθερά στις πρώτες τρεις περιπτώσεις. Στη συνέχεια μειώνονται στην τέταρτη και πέμπτη περίπτωση (όπου αμφότερες έχουν δίνουν τις ίδιες μετρήσεις). Και τέλος μειώνονται περαιτέρω στην έκτη περίπτωση, όπου οι μνήμες έχουν τέτοιο μέγεθος ώστε τα δεδομένα τα οποία προορίζονται να αποθηκευτούν σε αυτές να «χωράνε» ακριβώς. Μία ακόμη παρατήρηση, η οποία πρέπει να γίνει, είναι ότι ενώ τα Code+RO data είναι 11742 Bytes η ROM έπρεπε να έχει ελάχιστη χωρητικότητα 11744 Bytes.

Συμπεράσματα

Όσο μειώνεται η χωρητικότητα των μνημών, τα wait states και τα total cycles μειώνονται ή παραμένουν σταθερά. Φυσικά, αυτό γίνεται μέχρι τα δεδομένα τα οποία προορίζονται για την εκάστοτε μνήμη να «χωράνε» ακριβώς, οπότε έχουμε και το βέλτιστο αποτέλεσμα. Αυτό συμβαίνει διότι εμείς μπορεί να έχουμε την εντύπωση ότι τα δεδομένα αποθηκεύονται σειριακά, αλλά αυτό δεν ισχύει. Έτσι όταν η μνήμη είναι μεγαλύτερη, έχουμε και μεγαλύτερα πήγαινε-έλα μέσα στην μνήμη για την προσπέλαση των δεδομένων. Αξίζει να τονιστεί ότι κατά αυτό τον τρόπο κάνουμε σωστή χρήση της μνήμης.

2. Μέγεθος μνημών ROM, DRAM και SRAM

Σε αυτό το σημείο εξειδικεύσαμε περισσότερο την έννοια της RAM και το ρόλο τον οποίο επιτελούσε τον χωρίσαμε στα δύο. Έτσι βασιζόμενοι και στο θεωρητικό μέρος θεωρήσαμε πως πρέπει να έχουμε μία DRAM στην οποία θα αποθηκεύονται γενικά τα Zero-Initialized και Read/Write δεδομένα (όπως χρησιμοποιούσαμε προηγουμένως τη RAM) και μία SRAM στην οποία θα χρησιμοποιούμε ορισμένα δεδομένα τα οποία χρησιμοποιούμε συχνότερα και αξίζει να έχουμε ταχύτερη πρόσβαση σε αυτά.

Με βάση το προηγούμενο συμπέρασμα οριοθετήσαμε το μέγεθος των μνημών τόσο όσο απαιτείται έτσι ώστε τα δεδομένα που προορίζονται να αποθηκευτούν σε αυτές να «χωράνε» ακριβώς.

1. ROM 0x2DF4 // DRAM 0x134 // SRAM 0x12B820

Pragma Arm Section:

```
/* code for armulator*/
#pragma arm section zidata="sram"
int current_y[N][M];
int temparray[N+2][M+2];
int newtemparray[N+2][M+2];
#pragma arm section
```

Όπως φαίνεται και παραπάνω, για την SRAM προορίζεται ο πίνακας δεδομένων μας (current_y) και δύο προσωρινοί πίνακες, οι οποίοι χρησιμοποιούνται στη διαδικασία του φιλτραρίσματος.

Θυμίζουμε πως N=288 και M=352, οπότε το μέγεθος της SRAM προκύπτει ως εξής:

(288*352 + 290*354 + 290*354) * 4Bytes = 1226784 = $(12B820)_{16}$ Bytes

Το μέγεθος της DRAM προκύπτει ως εξής:

Total RW Size – SRAM Size = 1227092 – 1226784 = 308 = $(134)_{16}$ Bytes

Το μέγεθος της ROM προκύπτει όπως και προηγουμένως:

Total RO Size + 2 Bytes = 11764 = $(2DF4)_{16}$ Bytes

Image Component Sizes:

```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1260         60          0    1226792       6064   Object Totals
     10128        314          0        300       4472   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11388        374          0    1227092      10536   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                11762 (  11.49kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11762 (  11.49kB)
================================================================================
```

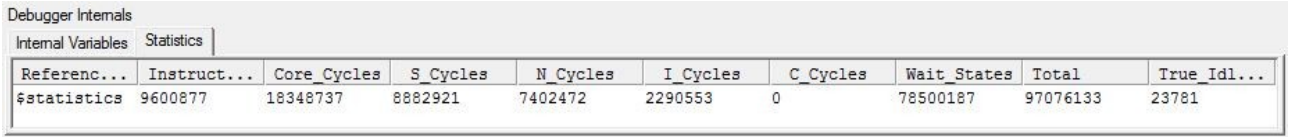
Memory Map:

```
00000000 00002DF4 ROM 4 R 250/1 250/1
00002DF4 00000134 DRAM 4 RW 150/250 150/250
00002F28 0012B820 SRAM 4 RW 55/55 55/55
```

Scatter File:

```
ROM 0x0 0x012B954
{
  ROM 0x0 0x2DF4
  {
    *.o  (+RO)
  }
  DRAM 0x2DF4 0x134
  {
    * (+ZI, +RW)
  }
  SRAM 0x2F28 0x12B820
  {
    *(sram)
  }
}
```

Debugger Internal Statistics:



Referenc...	Instruct...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idl...
$statistics	9600877	18348737	8882921	7402472	2290553	0	78500187	97076133	23781

2. ROM 0x2DF4 // DRAM 0x C8954 // SRAM 0x63000

Pragma Arm Section:

```
/* code for armulator*/
#pragma arm section zidata="sram"
int current_y[N][M];
#pragma arm section
```

Όπως φαίνεται και παραπάνω, για την SRAM προορίζεται μόνο ο πίνακας δεδομένων μας (current_y).

Θυμίζουμε πως N=288 και M=352, οπότε το μέγεθος της SRAM προκύπτει ως εξής:

(288*352) * 4Bytes = 405504 = $(63000)_{16}$ Bytes

Το μέγεθος της DRAM προκύπτει ως εξής:

Total RW Size – SRAM Size = 1227092 – 405504 = 3104524 = $(C8954)_{16}$ Bytes

Το μέγεθος της ROM προκύπτει όπως και προηγουμένως:

Total RO Size + 2 Bytes = 11764 = $(2DF4)_{16}$ Bytes

Memory Map:

```
00000000 00002DF4 ROM 4 R 250/1 250/1
00002DF4 000C8954 DRAM 4 RW 150/250 150/250
000CB748 00063000 SRAM 4 RW 55/55 55/55
```

Scatter File:

```
ROM 0x0 0x012B954
{
  ROM 0x0 0x2DF4
  {
    *.o  (+RO)
  }
  DRAM 0x2DF4 0xC8954
  {
    * (+ZI, +RW)
  }
  SRAM 0xCB748 0x63000
  {
    *(sram)
  }
}
```

Debugger Internal Statistics:

Debugger Internals

Internal Variables | Statistics

Referenc...	Instruct...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idl...
$statistics	9600877	18348737	8882921	7402472	2290553	0	85372887	103948833	23781

Παρατηρήσεις

Αρχικά, όσον αφορά το μέγεθος των δεδομένων, παρατηρούμε ότι τα RO data έχουν αυξηθεί σε σχέση με τις μετρήσεις που κάναμε όταν χρησιμοποιούσαμε μόνο ROM και RAM (αντί για ROM, DRAM και SRAM)

Έπειτα, όσον αφορά τους κύκλους, παρατηρούμε ότι τα wait states και τα total cycles μειώνονται όσο αυξάνουμε το πλήθος των δεδομένων που αποθηκεύονται στην SRAM κάτι το οποίο είναι αναμενόμενο. Ακόμη παρατηρούμε πώς οι χωρητικότητες των SRAM και DRAM πρέπει να αθροίζονται στο Total RW Size (RW Data + ZI Data). Και αυτό για να έχουμε τους λιγότερους δυνατούς κύκλους, όπως παρουσιάστηκε παραπάνω.

Συμπεράσματα

Προφανώς, έχουμε λιγότερα wait states και total cycles αν αποθήκευσουμε όλα τα RW και ZI data στην SRAM, αφού ο χρόνος ανάγνωσης και γραφής είναι μικρότερος. Παρόλα αυτά, όπως γνωρίζουμε, υπάρχει ένα trade-off μεταξύ κόστους και ταχύτητας ανάγνωσης/ γραφής της μνήμης. Γι’ αυτό συνηθίζεται η DRAM να είναι μεγαλύτερη από την SRAM. Έτσι ενώ η πρώτη ιεράρχιση μνήμης πετυχαίνει καλύτερα αποτελέσματα, εντέλει η δεύτερη προσεγγίζει περισσότερο τα πραγματικά δεδομένα.

3. Χρήση πινάκων προσωρινής αποθήκευσης τμήματος δεδομένων από μεγάλου μεγέθους πίνακα δεδομένων που εμφανίζουν μεγάλο αριθμό προσπελάσεων

Σε αυτό το σημείο χρησιμοποιήσαμε πίνακες δεδομένων προσωρινής αποθήκευσης τμήματος δεδομένων από μεγάλου μεγέθους πίνακα δεδομένων που εμφανίζουν μεγάλο αριθμό προσπελάσεων. Πιο συγκεκριμένα, δημιουργήσαμε τρεις μονοδιάστατους πίνακες, οι οποία αποθηκεύουν τμήματα δεδομένων του πίνακα temparray και χρησιμοποιούνται στη διαδικασία του φιλτραρίσματος.

1. ROM 0x2EC0 // DRAM 0x12B958 // SRAM 0x1098

Image Component Sizes:

```
==============================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1464         60          0    1231044       6360   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11592        374          0    1231344      10832   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11966 (  11.69kB)
    Total RW  Size(RW Data + ZI Data)           1231344 (1202.48kB)
    Total ROM Size(Code + RO Data + RW Data)      11966 (  11.69kB)
==============================================================================
```

Θυμίζουμε πως M+2=354, οπότε το μέγεθος της SRAM προκύπτει ως εξής:

3 πίνακες * 354 * 4Bytes = 4248 = $(1098)_{16}$ Bytes

Το μέγεθος της DRAM προκύπτει ως εξής:

Total RW Size – SRAM Size = 1231344 – 4248 = 1227096 = $(12B958)_{16}$ Bytes

Το μέγεθος της ROM προκύπτει όπως και προηγουμένως:

Total RO Size + 2 Bytes = 11968 = $(2EC0)_{16}$ Bytes

Pragma Arm Section:

```
#pragma arm section zidata="sram"
int buffer1[M+2];
int buffer2[M+2];
int buffer3[M+2];
#pragma arm section
```

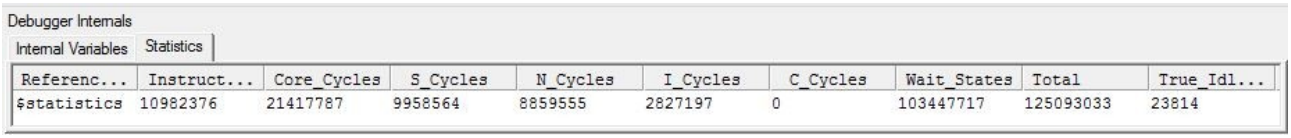
Memory Map:

```
00000000 00002EC0 ROM 4 R 250/1 250/1
00002EC0 0012B958 DRAM 4 RW 150/250 150/250
0012E818 00001098 SRAM 4 RW 55/55 55/55
```

Scatter File:

```
ROM 0x0 0x012B954
{
  ROM 0x0 0x2EC0
  {
    *.o  (+RO)
  }
  DRAM 0x2EC0 0x12B958
  {
    * (+ZI, +RW)
  }
  SRAM 0x12E818 0x1098
  {
    *(sram)
  }
}
```

Debugger System Internals:



2. ROM 0x2EC0 // DRAM 0xC8958 // SRAM 0x64098

Θυμίζουμε πως M+2=354, οπότε το μέγεθος της SRAM προκύπτει ως εξής:

(3 πίνακες * 354 + 288*352) * 4Bytes = 409752 = $(64098)_{16}$ Bytes

Το μέγεθος της DRAM προκύπτει ως εξής:

Total RW Size – SRAM Size = 1231344 – 409752 = 1227096 = $(821592)_{16}$ Bytes

Το μέγεθος της ROM προκύπτει όπως και προηγουμένως:

Total RO Size + 2 Bytes = 11968 = $(2EC0)_{16}$ Bytes

Pragma Arm Section:

```
#pragma arm section zidata="sram"
int current_y[N][M];
int buffer1[M+2];
int buffer2[M+2];
int buffer3[M+2];
#pragma arm section
```

Memory Map:

```
00000000 00002EC0 ROM 4 R 250/1 250/1
00002EC0 000C8958 DRAM 4 RW 150/250 150/250
000CB818 00064098 SRAM 4 RW 55/55 55/55
```

Scatter File:

```
ROM 0x0 0x012B954
{
  ROM 0x0 0x2EC0
  {
    *.o  (+RO)
  }
  DRAM 0x2EC0 0xC8958
  {
    * (+ZI, +RW)
  }
  SRAM 0xCB818 0x64098
  {
    *(sram)
  }
}
```

Debugger System Internals:

Debugger Internals

Internal Variables | Statistics

Referenc...	Instruct...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idl...
$statistics	10982376	21417787	9958564	8859555	2827197	0	100387762	122033078	23814

Παρατηρήσεις

Αρχικά, όσον αφορά το μέγεθος των δεδομένων, παρατηρούμε ότι τα ZI data έχουν αυξηθεί, αφού οι 3 καινούριοι μονοδιάστατοι πίνακες θα αρχικοποιηθούν με μηδενικά όταν «κατέβουν» στον επεξεργαστή. Και για προφανείς λόγους αυξήθηκε και ο κώδικας.

Έπειτα, όσον αφορά τους κύκλους, παρατηρούμε ότι τα wait states και τα total cycles αυξήθηκαν αφού εισάγαμε τους πίνακες δεδομένων προσωρινής αποθήκευσης τμήματος δεδομένων από τον newtemparray. Επίσης παρατηρούμε ξανά πως όσα περισσότερα δεδομένα αποθηκεύσω στην SRAM τόσο λιγότεροι είναι οι συνολικοί κύκλοι. Μία ακόμη παρατήρηση, η οποία πρέπει να γίνει, είναι ότι ενώ τα Code+RO data είναι 11966 Bytes η ROM έπρεπε να έχει ελάχιστη χωρητικότητα 11968 Bytes.

Συμπεράσματα

Οι πίνακες δεδομένων προσωρινής αποθήκευσης τμήματος δεδομένων από μεγάλου μεγέθους πίνακα δεδομένων που εμφανίζουν μεγάλο αριθμό προσπελάσεων, όπως κατανοήθηκαν και υλοποιήθηκαν από μέρους μας δεν κατάφεραν να μειώσουν τα wait states και τα total cycles, αντιθέτως τα αύξησαν. Αυτό οφείλεται στο παραπάνω κόστος που που εισάγουν οι πίνακες προσωρινής αποθήκευσης τμήματος δεδομένων (buffers) που χρησιμοποιούμε.

Παρόλα αυτά, η συγκγκριμένη τεχνική έχει νόημα. Διότι ενδέχεται να υπάρχει τέτοιος περιορισμός στη χωρητικότητα της μνήμης SRAM, που να μη μας επιτρέπει να αποθηκεύσουμε έναν ολόκληρο πίνακα ακεραίων με διαστάσεις 288*352 ή 290*354, οπότε αναγκαστικά αν θέλουμε να χτησιμοποιούμε τη γρηγορότερη μνήμη SRAM, θα πρέπει να αποθηκεύουμε τμήματα των συχνά προσπελάσιμων πινάκων και όχι ολόκληρους τους πίνακες.

Απάντηση στο πρώτο ερώτημα

Η απάντηση στο πρώτο ερώτημα εντοπίζεται στην τρίτη υποενότητα της διερεύνησης. Παρόλα αυτά έχει νόημα και είναι απαραίτητο να ακολουθηθεί από την αρχή και σειριακά η εξελικτική πορεία της διερεύνησης καθώς έτσι θα κατανοηθεί και η εξελικτική πορεία των συμπερασμάτων.

Απάντηση στο δεύτερο ερώτημα

Όπως αναφέρθηκε και προηγουμένως, είναι απαραίτητο να ακολουθηθεί από την αρχή και σειριακά η εξελικτική πορεία των συμπερασμάτων. Πιο συγκεκριμένα η απάντηση εντοπίζεται στα τρια συμπεράσματα της ενότητας της διερεύνησης.

Αξίζει να αναφερθεί ότι τα «image component sizes» της εκάστοτε υποενότητας ήταν ίδια και γι’ αυτό το λόγο εμφανίζονται μόνο μία φορά σε κάθε υποενότητα.

Βιβλιογραφία

Διαφάνειες εργαστηρίου από το μάθημα «Σχεδιασμός Ενσωματωμένων Συστημάτων»

Σημειώσει από το μάθημα «Σχεδιασμός Ενσωματωμένων Συστημάτων»

Οι υπολογιστές ως συστατικά στοιχεία, Wayne Wolf

Οργάνωση & Αρχιτεκτονική Υπολογιστών, William Stallings