

***ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΠΡΟΗΓΜΕΝΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ***

Assignment 1

Συγγραφείς: Σακελλαρίου Βασίλης , Φίλης Χάρης

Contents

1	Εργαστήριο 1	3
1.1	Ανάλυση του αρχείου : starter_se.py	3
1.2	2	4
1.3	Λόγια για CPU models του gem5 και Προσομοιώσεις προγράμματος σε C στον gem5	5
1.3.1	Προσομοίωση στον gem5 / Εκτέλεση προγράμματος σε C σε arm επξεργαστή	6

Chapter 1

Εργαστήριο 1

1.1 Ανάλυση του αρχείου : `starter_se.py`

Απο το αρχείο `starter_se.py` αναγνωρίζουμε τα εξής στοιχεία:

- Ορίζεται το dictionary `cpu_types` που αντιστοιχίζει το `-cpu` σε έναν από τους παρακάτω τύπους(δομή δεδομένων):
 1. AtomicSimpleCPU : "atomic" -> Η προεπιλογή τύπου επεξεργαστή που δεν έχει caches.
 2. MinorCPU : "minor" -> Στην περίπτωση αυτή χρησιμοποιείται η βιβλιοθήκη `devices.py` που ορίζονται οι δομικές μονάδες του minor(δηλ. *L1 caches(dcachelcache), L2 cache*) με όλα τα χαρακτηριστικά τους.
 3. HP1 : "hpi" -> Κατα αντιστοιχία με το παραπάνω μέσω της βιβλιοθήκης `HPI.py` ορίζονται οι απαραίτητες μνήμες cache για hpi επεξεργαστή ή cluster.
- Ορίζεται στην κλάση `SimpleSeSystem` Clock speed: **1GHz** στο πεδίο
`self.clk_domain = SrcClockDomain(clock="1GHz",
voltage_domain=self.voltage_domain)`
- Στην main function ορίζονται από τον parser τα εξής:
 - Ο τύπος του επεξεργαστή που στο δικό μας παράδειγμα επιλέγεται ο MinorCPU.
 - Caches: με την επιλογή minor cpu έχουμε και L1 cache και L2 cache
 - Μνήμη: με την εντολή που τρέχουμε δεν δίνουμε την παράμετρο `-mem-size` άρα παίρνει την default τιμή που είναι 2Gb ούτε και το `-mem-type` οπότε by default πάει στην τεχνολογία **DDR3_1600_8x8**.
 - Components voltage: default->3.3V.
 - CPU cluster voltage: default->1.2V.

- CPU freq: default->4Ghz.
- Number of cores: default->1.

1.2 2

2 a)

Ανοίγοντας το config.ini

[system.clk_domain]

clock = 1000 (επαληθεύεται)

[system.cpu_cluster.cpus.dcache] -> size = 32768 (που επαληθεύεται απο το αρχείο devices.py οπου η data cache l1 του minor είναι 32KB)

[system.cpu_cluster.cpus.icache] -> size = 49152 (που επαληθεύεται απο το αρχείο devices.py οπου η instruction cache l1 του minor είναι 48KB)

[system.cpu_cluster.l2] -> size = 1048576 (που επαληθεύεται απο το αρχείο devices.py οπου η cache l2 του minor είναι 1MB)

[system.voltage_domain] voltage=3.3(που επαληθεύεται)

[system.cpu_cluster.voltage_domain] -> voltage = 1.2(που επαληθεύεται)

[system] -> *mem_ranges* = 0 : 2147483648 (που επαληθεύει τα 2GB by default ram)

2 b)

Ανοίγοντας το αρχείο stats.txt Committed intructions 5028 από:

line14 :system.cpu_cluster.cpus.committedInsts 5028 // Number of instructions committed

2c) Το συνολικό πλήθος των προσβάσεων στην L2 cache είναι 479, από τα οποία 332 είναι απο miss της icache και 147 απο miss της dcache.

system.cpu_cluster.l2.demand_accesses::total 479 // number of demand (read+write) accesses

system.cpu_cluster.l2.overall_accesses::cpu_cluster.cpus.inst 332 // number of overall (read+write) accesses

system.cpu_cluster.l2.overall_accesses::cpu_cluster.cpus.data 147 // number of overall (read+write) accesses.

1.3 Λόγια για CPU models του gem5 και Προσομοιώσεις προγράμματος σε C στον gem5

- AtomicSimpleCPU
Ο AtomicSimpleCpu είναι μια εκδοχή SimpleCPU που παρέχει ο gem5 και χρησιμοποιεί atomic memory accesses. Σύμφωνα με το documentation του gem5 **atomic accesses** είναι προσβάσεις στην μνήμη γρηγορότερες από τις λεπτομερείς (*detailed accesses*) και χρησιμοποιούνται για fast forwarding και warming up caches. Ο AtomicSimpleCPU εκτελεί όλες τις λειτουργίες για μια εντολή σε κάθε CPU tick() όπως φαίνεται από το σχεδιάγραμμα λειτουργίας.
- TimingSimpleCPU
Ο TimingSimpleCPU είναι μια εκδοχή του SimpleCPU που παρέχει ο gem5 και χρησιμοποιεί timing memory accesses. Επειδή το μοντέλο αυτό δεν έχει μνήμη cache σταματάει (*stall*) στα cache accesses και περιμένει από την κύρια μνήμη να ανταποκριθεί πριν συνεχίσει την εκτέλεση εντολής. Είναι επίσης ένα fast-to-run μοντέλο, δεν υποστηρίζει pipelining, οπότε εκτελεί μόνο μία εντολή κάθε στιγμή. Atomic και Timing memory accesses δεν μπορούν να συνυπάρχουν σε ένα σύστημα μνήμης.¹
- MinorCPU
Ο Minor είναι ένας in-order επεξεργαστής με τέσσερα επίπεδα pipeline και παραμετροποιήσιμες δομές δεδομένων και συμπεριφορά εκτέλεσης ώστε να μπορεί να εξομοιώνει όσο πιο κοντά γίνεται ένα πραγματικό επεξεργαστή. Τα 4 στάδια pipeline του minor είναι τα εξής:
 1. Fetch1 :: Fetch1 is responsible for fetching cache lines or partial cache lines from the I-cache and passing them on to Fetch2
 2. Fetch2 :: decomposes cache lines into instructions
 3. Decode :: takes a vector of instructions from Fetch2 (via its input buffer) and decomposes those instructions into micro-ops (if necessary) and packs them into its output instruction vector
 4. Execute :: execution procedure
- High-Performance In-order (HPI) CPU
Το μοντέλο αυτό είναι βασισμένο στην αρχιτεκτονική Arm και το ονομάζουμε HPI. Το HPI CPU timing model ρυθμίζεται για να αντιπροσωπεύει μια μοντέρνα in-order Armv8-A εφαρμογή. Το pipeline του HPI CPU χρησιμοποιεί το ίδιο four-stage μοντέλο όπως ο MinorCPU που αναφέραμε προηγουμένως.

¹Timing accesses are the most detailed access. They reflect our best effort for realistic timing and include the modeling of queuing delay and resource contention. Once a timing request is successfully sent at some point in the future the device that sent the request will either get the response or a NACK if the request could not be completed (more below). Timing and Atomic accesses can not coexist in the memory system.

1.3.1 Προσομοίωση στον gem5 / Εκτέλεση προγράμματος σε C σε arm επξεργαστή

Ο κώδικας που γράψαμε σε c είναι ο εξής:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    double x = rand()%10;
    if ( x > 5){
        printf("Random number is greater than 5\n");
    } else
    {
        printf("Random number is less than 5\n");
    }
    printf("\n");
    printf("%f\n",x);

    return 0;
}
```

Επειδή το cpu_clock είναι στα 500ticks by default το ορίζουμε σε όλες τις προσομοιώσεις στο 1000ticks. Όταν αλλάζουμε συχνότητα επεξεργαστή μας ενδιαφέρει το sys_clk.

a)

Για την πρώτη προσομοίωση τρέχουμε το εξής για minor:

```
./build/ARM/gem5.opt -d ~/Desktop/MinorCPUdefault configs/example/se.py --cpu-type
--caches -c ~/Desktop/gem5_test
```

απο το stats.txt:

MinorCPU

Line 12: sim_seconds 0.000050 // Number of seconds simulated

Αντίστοιχα για τον TimingSimpleCPU:

```
./build/ARM/gem5.opt -d ~/Desktop/MinorCPUdefault configs/example/se.py --cpu-type
--caches -c ~/Desktop/gem5_test
```

TimingSimpleCPU

Line 12: sim_seconds 0.000066 // Number of seconds simulated

b)

Από τις προσομοιώσεις ο MinorCPU είναι πιο γρήγορος από τον TimingSimpleCPU. Αυτό ήταν αναμενόμενο καθώς ο MinorCPU υποστηρίζει 4 επίπεδα pipeline και τεχνολογία μνημών cache. Επίσης όσον αφορά τους κύκλους που χρειάστηκαν για την ολοκλήρωση του προγράμματος στον TimingSimpleCPU είναι 65607 σε αντίθεση με τους 50071 κύκλους του MinorCPU. Αυτή η διαφορά οφείλεται στο

οτι ο TimingSimpleCPU κάνει απευθείας προσπέλαση
c)

1. Προσομοίωση σε συχνότητα επεξεργαστή 500Mhz-> -sys-clock = 500000000,
-cpu-clock=1000000000
απο το stats.txt:
MinorCPU
Line 12: sim_seconds 0.000057 // Number of seconds simulated
TimingSimpleCPU
Line 12: sim_seconds 0.000073 // Number of seconds simulated
Σχόλια : "Βλέπουμε μια αύξηση του χρόνου εκτέλεσης που είναι απολύτως
λογικό καθώς υποδιπλασιάζουμε την συχνότητα του ρολογιού οπότε ο χρόνος
κύκλου διπλασιάζεται."
2. Προσομοίωση με αλλαγή τεχνολογίας μνήμης DDR4_2400_8x8 απο την
default DDR3_1600_8x8 -> -mem-type=DDR4_2400_8x8.
απο το stats.txt:
MinorCPU
Line 12: sim_seconds 0.000049 // Number of seconds simulated
TimingSimpleCPU
Line 12: sim_seconds 0.000065 // Number of seconds simulated
Σχόλια : "Παρατηρούμε μείωση του χρόνου εκτέλεσης γιατί τώρα έχουμε
μνήμες με μεγαλύτερο bandwidth και καλύτερης τεχνολογίας Double Data
Rate".