TEIS
Teknisk rapport

Författare : Lasse Karagiannis
Uppgift : Task 9, C-programmering

E-postadress: lasse.l.karagiannis@gmail.com
Kontrollerad: Nej
Version: 1.0

Fil

Lasse_Karagiannis_C_task_9.pdf

TEIS

CPU Performance

Task 9 C-programmering

Lasse Karagiannis

16-11-06

Innehåll

1 KRAVSPECIFIKATION

Tabell 1 Kravspecifikation från kund

Krav	Beskrivning	Utfört Ja/nej
	Förstudie	
Krav_001	Gå igenom teorilektionen (Teori_9 och länk*) och skriv en kort sammanfattning vilken ska ingå i rapporten enligt krav_007 (eget kapitel). Beskriv följande begrepp och hur de påverkar prestanda (övergripande): - Data and instruction cash - Pipeline - Beskriv gärna andra CPU-mekanismer - Svara på frågan; Vad är det för skillnad när det gäller prestanda (accestid) på internal RAM och external RAM (utanför chippet)? *https://www.altera.com/products/processors/support.html	JA
Krav_002	Beskriv de olika HW-arkitekturerna (eget kapitel) - 50MHz Economy (HW_CASE_4) - 50 MHz Fast (HW_CASE_4_F)	JA
	Funktionskrav	
Krav_003	Använd testprogrammet i bilaga 1.	
Krav_004	Följande testfall ska gås igenom med de två olika processorerna (economy och fast): ***Testfall 1*** **Mjukvara (all programvara)**: Interna FPGA RAM, enable_reduced_device_drivers, enable_small_c_library. ***Testfall 2*** **Mjukvara (all programvara)**: SRAM minnet/Ingen C-kods optimering ***Testfall 3*** **Mjukvara (all programvara)**: SRAM minnet/ enable_reduced_device_drivers, enable_small_c_library.	JA
Krav_005	Tabell 1 enligt bilaga 2 ska fyllas i med antal klockcykler och tid.	JA
Krav_006	Gör kortfattad analys av siffrorna. (eget kapitel)	
	Dokumentationskrav	
Krav_007	Sammanfoga dokumentationen från krav_001 till krav_006 till en läsbar rapport med separata kapitel. Framsida med titel, en kort sammanfattning, innehålls-förteckning och kapitel. Lägg även till eventuella slutsatser och referenser.	JA
	Leveranskrav	

Krav_008	Leveransen ska ske till plattformen Itslearning. Leveransen ska vara en rapport. Namnet på filen ska vara ”förnamn_efternamn_C_task_9”. Sista leveransdag se kursschema (för VG).	JA

2 Teorilektion_9

Följande begrepp gicks igenom *Data cash, instruction cash, Pipeline.* Vidare nämndes *branch prediction* även om termen som sådan inte introducerades.

Uppgiftskravet avseende detta kapitel är, förutom att återge teori kring ovan nämnda begrepp, också besvara rågan: ”Vad är det för skillnad när det gäller prestanda (accestid) på internal RAM och external RAM (utanför chippet)?”

Avseende prestanda för skrivning och läsning av internt minnet, så görs detta typiskt på 1-2 klock-cykler. För extern SDRAM, så krävs en SDRAM kontroller som fräshar upp minnespositioner, och då är inte minnet tillgängligt. Embedded manualen som finns att ladda ner via länken nedan, anger på sidan 26(497 sidor) en typisk access tid på 17 ns, utan att fördensskull ange klockfrekvensen, men med en typisk klock-frekevens i sammanhanget, nämligen 200MHz, så skulle detta bli ca 3 klock-cykler. Vidare måste man veta om SDRAM-kontrollerna kommunicerar via en tri-state brygga, då har processorn inte ens tillgång till enheten när den vill, utan att först begära tillträde via bryggan.
https://documentation.altera.com/#/00110102-AA$AA00110089

Datacash och instruktionscash ligger på on-chip RAM, och syftet med dessa är att snabba upp åtkomsten av data respektive kod.

Pipelining innebär att dela upp CPU:ns dataväg i stationer av separata arbetsmoment och separera dem emellan med register -arrayer.
Typiskt är *decode* den första station, därefter en station för att ladda data-vägen med operander från antingen register eller minnet, därefter utföra en ALU-operation, och sist ett w*rite-back* steg, då minnet uppdateras.
Dessa steg tar var och en, en klokcykel, och CPU:n hämtar en ny instruktion varje klockcykel, så varje station arbetar enligt löpande -band principen.

Implementeringen att operationskodens olika fält manipulerar olika delar av pipelinen kallas VLIW, annars används OP-koden till att slå upp ett styr koden för datavägen ur ett minne.

Branchprediction handlar om att inte vänta på att status-registret ska uppdateras, förrän deode och hämtning från minnet görs. Man påbörjar exekvering av instruktion men utan att gå så långt att man uppdaterar minnet. Först när t.ex. zero-flaggan visar att att branch ska tas eller inte tas så uppdateras minnet. Om man inte har valt rätt kod-nutt att exekvera i en if-then- else sats så får man tömma resetta register i pipe-linen (flushing) och börjam ed fetch från rätt adress.

3 Nios II/e (economy) och Nios II/f (fast)

Fast-versionen har en instruktions-cash på 4 kB och en data-cash på 2kB, medan economy-varianten inte har någondera, vidare har även fast-versionen en hårdvaru-multiplikator, vilket inte economy har. Typiskt kan man säga att fast-versionen är 10 ggr snabbare. Några benchmarking test, som jämför fast-versionen och economy-versionen, implementerade på några olika FPGA:er, finns redovisade på https://www.altera.com/en_US/pdfs/literature/ds/ds_nios2_perf.pdf

4 Tester enligt kravspecen

Tabell 2: Mätväden från dina testfall

Testfall	Programmets placering	HW Download	
		Economy ingen cashe **HW_CASE_4** (cykler, absolut tid)	Fast Instruktion 4kb och Data 2kb cashe **HW_CASE_4_F** (cykler, absolut tid)
1	Interna FPGA RAM/ enable_reduced_device_drivers, enable_small_c_library	*111898 1.4ms*	4637 58us
2	SRAM minnet/ Ingen C-kods optimering	247943 3ms	5070 63us
3	SRAM minnet/ enable_reduced_device_drivers, enable_small_c_library	247961 3ms	4636 58us

Absolut tid = Antal clock cykler * period tid (system klockan kopplad till timer är 80 MHz).

5 Komentar kring mätningarna

Eftersom economy-versionen saknar cash för både instruktion och data, så är det inte konstigt att tidsåtgången för då koden placerats utanför chip (SDRAM) att tiden är densamma oavsett *small C-library* och *reduced device drivers.* Tidsåtgången varierar från varv till varv, så ett medeltal har angivits för samtliga tre testfall. Att testfall 1 varierar, redovisar här ett litet intervall(111926, 111861, 111903, 111904), trots att all kod ligger on-chip, tyder på att branch-prediction och pipelining används. Offset var här 14.

För de resterande två test-konfigurationerna avseende economy, så är skillnaden inte så stor om *reduced device drivers* och s*mall C-library* används. Skillnaden i tiden att hämta från SDRAM överskuggar skillnaden i antalet instruktioner som exekveras i de olika fallen. Offset var i båda fallen 31

Konfig 2	Konfig 3
247969	247980
247984	247933
247898	247951
247956	247980
247897	247971
247956	247951

Det faktum att testkonfiguration 1 ger så stor skillnad mellan economy och fast, beror förmodligen på att fast använder en mer avancerad arkitektur.

I fallet fast, och on-chip memory så ligger antalet cykler stabilt på 4637, utan några som helst variationer Utan C-kods optimering så får vi cash-missar. Systemet har fått gå och hämta från SDRAM.

4677
4678
7410
4676
4679
4679
4689

Med optimerad kod får vi ingen cashmiss, vi har variationer av av antalet cykler det tar med en cykels skillnad, vidare är offset 3 och inte 4, men detta skulle ju betyda att testet visar att kod skulle kunna exekvera snabbarare om det lades på SDRAM än intent, vilket inte kan vara fallet.

Jag kan nog inte förklarar diskrepansen visavi on-chip ram, för Nios II/f
On-chip RAM: 4637 cykler stabilt över tid, samt 4 cyklers offset stabilt över tid.

Test config 3, optimerad kod på SDRAM Nios II/f :

4635
4636
4636
4635
4636
4635
4635

och offset är 3!