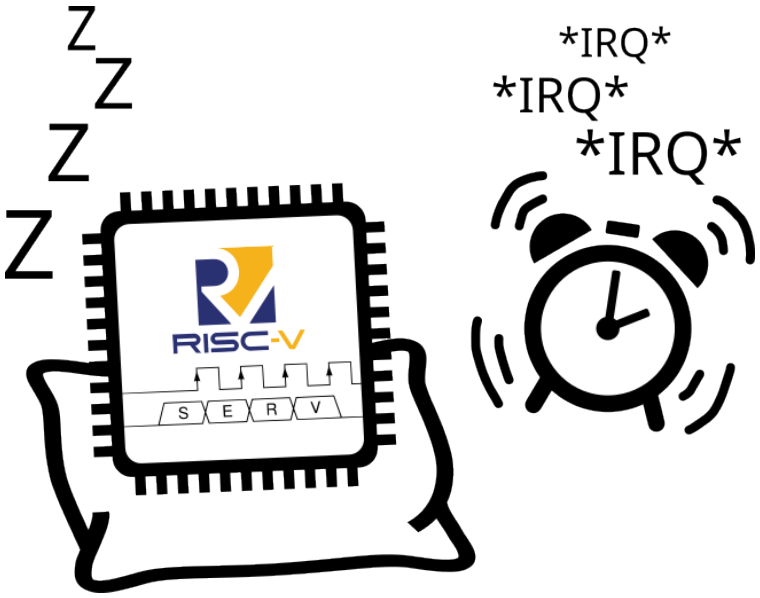
CHALMERS



Implementation av externa avbrott i RISC-V-processorn SERV

Implementation of external interrupts in the RISC-V processor SERV

Externa avbrott, WFI-instruktion och sleep-funktionalitet för RISC-V-processorn SERV

Kanditatarbete i datateknik och elektroteknik

Simon Andersson, Alfred Forsberg,
Holger Johansson Skarin, August Ådahl

INSTITUTIONEN FÖR MIKROTEKNOLOGI OCH NANOVETENSKAP

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2025
www.chalmers.se

Kandidatarbete 2025

Implementation av externa avbrott i RISC-V-processorn SERV

Externa avbrott, WFI-instruktion och sleep-funktionalitet för RISC-V-processorn SERV.

Simon Andersson
Alfred Forsberg
Holger Johansson Skarin
August Ådahl

CHALMERS

Institutionen för mikroteknologi och nanovetenskap
Chalmers tekniska högskola
Göteborg, Sverige 2025

Implementation av externa avbrott i RISC-V-processorn SERV.
Externa avbrott, WFI-instruktion och sleep-funktionalitet för RISC-V-processorn SERV.

Simon Andersson, Alfred Forsberg, Holger Johansson Skarin & August Ådahl

Handledare: Lars Svensson, Institutionen för mikroteknologi och nanovetenskap
Examinator: Per Lundgren, Institutionen för mikroteknologi och nanovetenskap

Kandidatarbete 2025
Institutionen för mikroteknologi och nanovetenskap
Chalmers tekniska högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: En sovande FPGA blir väckt av ett avbrott i form av en väckarklocka.

Skrivet i LaTeX
Göteborg, Sverige 2025

Implementation av externa avbrott i RISC-V-processorn SERV.
Externa avbrott, WFI-instruktion och sleep-funktionalitet för RISC-V-processorn SERV.

Simon Andersson, Alfred Forsberg, Holger Johansson Skarin & August Ådahl

Institutionen för mikroteknologi och nanovetenskap
Chalmers tekniska högskola

Abstract

Different processor designs are optimized to meet criteria such as maximum performance, power efficiency or high core count. The SERV processor was designed around one simple criterion: to be the world's smallest RISC-V core. This has been achieved by using a bit-serial datapath and by waiving all but the most essential features. In order to make SERV a more practically usable processor, we have implemented external interrupts and sleep mode while still adhering to the original goal of a low implementation cost. In addition to being open source, SERV is also very well documented. To adhere to this standard, we have made all of our code open sorce and created block diagrams for every feature that was implemented. We have also tested the implementation via testbenches and on FPGA hardware to verify that everything works as expected. We have evaluated the design in terms of implementation cost and power savings on an Artix-7 FPGA.

Keywords: RISC-V, SERV, RV32IZicsr, FPGA, sleep, WFI, avbrott, interrupt, Artix-7.

Sammandrag

Olika processordesigner är optimerade för att möta kriterier såsom maximal prestanda, strömsnålhet eller högt kärnantal. Processorn SERV är designad kring ett enkelt kriterium: att vara världens minsta RISC-V-processor. Detta har uppnåtts genom att använda en bit-seriell dataväg och genom att göra avkall på all funktionalitet förutom den mest väsentliga. För att göra SERV till en mer praktiskt användbar processor, har vi implementerat externa avbrott och sleep-läge samtidigt som vi hållit oss till det övergripande målet om låg implementationskostnad. Utöver att vara open source, är SERV dessutom mycket väldokumenterad. För att hålla oss till denna standard har vi gjort all vår kod open source och skapat blockdiagram för all funktionalitet som implementerats. Vi har också testat implementationen via testbenches och på FPGA-hårdvara för att verifiera att allt fungerar som väntat. Vi har utvärderat designen enligt implementationskostnad och energibesparingar på en Artix-7 FPGA.

Förord

Arbetet är ett kandidatarbete som omfattar 15 högskolepoäng med studenter från data- och elektroteknik på Chalmers tekniska högskola. Projektet är en vidareutveckling av SERV-processorn som i grunden är utvecklad av Olof Kindgren. Vi är tacksamma och stolta över att få möjligheten att medverka i utvecklingen av SERV. Vi vill även tacka vår handledare, Lars Svensson, för hans stöd under arbetet.

Simon Andersson, Alfred Forsberg, Holger Johansson Skarin & August Ådahl
Göteborg, 2025

Innehåll

Akronymer

Nedan listas akronymer som använts i arbetet i alfabetisk ordning:

CAD	Computer-Aided Design
CSR	Control/Status Registers
FPGA	Field-Programmable Gate Array
GPIO	General-Purpose Intput/Output
I/O	Input/Output
LUT	LookUp Table
SERV	The SErial RISC-V CPU
SoC	System-on-a-Chip
WFI	Wait-For-Interrupt

Figurer

1

Introduktion

Rapporten inleds med en ämneshistoria och grundläggande begrepp som krävs för att förstå arbetet.

1.1 Ämnesbakgrund

År 1945 färdigställdes ENIAC, världens första generellt programmerbara dator. Jämfört med moderna datorer var det en gigantisk maskin, 167 kvadratmeter till ytan och vägde över 30 ton [1]. På den tiden fanns inte transistorer så man använde elektronrör för att konstruera datorns olika beståndsdelar. Elektronrör är stora, dyra och konsumerar enorma mängder energi, flera miljoner gånger mer än en transistor [2]. ENIAC förbrukade 200 kilowatt när den var igång. Datorer från sent 40-tal och tidigt 50-tal tillverkades ofta i enstaka exemplar och användes bara vid universitet och av militären.

När transistorer började ersätta reläer och elektronrör kunde datorer som var mycket mindre än tidigare konstrueras. Den nya generationens datorer kunde serieproduceras till ett pris som gjorde att företag kunde börja använda dem. Under 60-talet började även så kallade minidatorer produceras. Dessa var billigare och ungefär lika stora som ett kylskåp. Runt 60-talet började man även kunna integrera flera transistorer på en och samma kiselplatta [3]. Den integrerade kretsen möjliggjorde ännu billigare produktion eftersom samma funktionalitet krävde färre komponenter. Ytterligare en milstolpe kom 1971 när Intel lyckades producera ett chip som innehöll all funktionalitet som en processor behöver [4]. Mikroprocessorns uppfinnande möjliggjorde persondatorns genomslag under tidigt 80-tal.

Förbättrade produktionskedjor och miniatyrisering har fortsatt driva ner priset samtidigt som datorernas prestanda har ökat exponentiellt. Datorer utgör idag en integrerad och oumbärlig del av det moderna samhället. Deras närvaro påverkar såväl individers vardag som samhällets infrastrukturella, ekonomiska och kommunikativa system. Vanligtvis associeras begreppet dator med skrivbordsdatorer, laptops, smartphones och kanske servrar. Majoriteten av alla datorer är dock de små inbyggda system som sitter i exempelvis fjärrkontroller, mikrovågsugnar, värme-/ventilationssystem och bilar. Priset för en persondator har legat stilla ganska länge samtidigt som prestandan har ökat drastiskt medan priset för inbyggda system har sjunkit dramatiskt [5]. Idag finns det chip med inbyggd processor, minne och periferienheter, så kallade mikrokontrollers, som kostar mindre än en krona styck. Detta kan tyckas vara en försumbar summa men om kraven på prestanda är låga och produktionsvolymen är hög så finns det pengar att spara på att effektivisera sin processordesign.

1.1.1 Open source

Open source, eller öppen källkod som det heter på svenska, är en filosofi inom mjukvaruutveckling och till viss del hårdvaruutveckling [6]. Den som skriver mjukvaran publicerar den öppet och gratis för vem som helst att använda eller vidareutveckla. Vissa open-source-projekt drivs även genom att flera personer kollektivt utvecklar mjukvaran och driver den framåt. Tanken är att mjukvara ska växa organiskt och på så sätt främja säkerhet, flexibilitet och lättilgänglighet. Det mest kända exemplet av open source är förmodligen Linuxkärnan. Linux är en robust och säker operativsystemkärna eftersom vem som helst kan gå in och studera källkoden efter eventuella svagheter. Linus lag, "given enough eyeballs, all bugs are shallow", myntad av Eric S. Raymond [7, s. 50] beskriver fenomenet. Dessutom är Linux gratis att använda och fritt att modifiera efter egna behov. Ett stort antal företag använder sig av Linux och är med och förbättrar Linux, bland annat: Ericsson, Meta, Intel, Oracle, Samsung och Hitachi [8].

1.1.2 Processorns uppbyggnad

En processor består framförallt av transistorer som sammankopplats till logiska grindar. Med många ihopsatta grindar kan man bygga större enheter som kan genomföra aritmetiska operationer eller styra datans förflyttning inne i processorn. Man kan även konstruera vippor som används för att lagra ettor och nollor. Flera vippor i bredd brukar kallas för register och används för att lagra data och processorns interna tillstånd.

1.1.3 Processorarkitekturer

I datorns barndom var ett vanligt problem att mjukvara som var skriven för en datormodell inte enkelt kunde flyttas över och användas på en annan dator. Man var tvungen att skriva om program från grunden för varje datormodell man ville köra det på. En metod som man använder för att komma runt detta problem är genom standardiserade instruktionsuppsättningsarkitekturer, förkortat ISA. En ISA bestämmer vilka hårdvaruinstruktioner som processorn ska kunna köra [9, s. 11-12]. Man kan alltså konstruera två helt olika processorer, med olika egenskaper, som ändå kan köra samma maskinkod. Först med detta var IBM när de 1964 lanserade sin nya datorserie System/360. System/360 nådde enorm framgång eftersom kunder kunde köpa en billigare modell och uppgradera senare [10]. Idag är det framförallt två ISA:er som dominerar, x86 och ARM. Intel och AMD använder x86 i sina processorer. Därmed körs x86 i nästan alla persondatorer och servrar. ARM används i framförallt mobiltelefoner och andra mobila applikationer samt i inbyggda system.

1.1.4 RISC-V

RISC-V är en nyare ISA introducerad 2014 [11] som till skillnad från x86 och ARM har en öppen källkodslicens, vilket betyder att den kan användas fritt utan licenskostnader. Den används idag framförallt inom forskning och utbildning och i inbyggda system [12].

1.1.5 FPGA

En FPGA (Field-Programmable Gate Array) är en integrerad krets som ofta används när man utvecklar hårdvara. En FPGA består till stor del av programmerbara logikenheter:

vippor (även kallat register) och lookup tables (LUTs). Vipporna kan synkroniseras med olika klocksignaler i FPGAn. En LUT är konfigurerbar och används för att realisera olika logiska uttryck. Vipporna och LUTarna sammankopplas i nät för att utföra godtycklig hårdvarufunktionalitet. Detta är inte samma sak som att programmera exempelvis en mikrokontroller eftersom mikrokontrollern i sig är en färdig hårdvarudesign som inte går att ändra.

Under utvecklingsprocessen av hårdvara (exempelvis en ethernet-switch) vill man gärna kunna funktionstesta olika implementationer för att avgöra vilken som är bäst, då kan man använda en FPGA. Uppstartskostnaden för hårdvaruproduktion är skyhög, flera miljoner kronor. Vid nischade uppgifter med låg produktionsvolym är en FPGA därför ett ekonomiskt alternativ till applikationsspecifik hårdvara. En FPGA är dock långsammare och mindre energieffektiv än motsvarande applikationsspecifik hårdvara [13].

1.1.5.1 Artix-7 och Nexys A7

Nexys A7 är det FPGA-utvecklingskort som vi har använt i vårt projekt. Hjärtat i Nexys A7 är en AMD Artix-7 FPGA. Vi valde att utveckla för den plattformen eftersom den är vanligt förekommande i kurser på Chalmers tekniska högskola. Dessutom har den processor (SERV) som vi ska vidareutvecka sedan tidigare stöd för Artix-7 plattformen, vilket underlättar implementationen.

1.1.6 Hårdvarubeskrivande språk

Hårdvarubeskrivande språk används för att beskriva digitala kretsar genom att definiera logikelement, minneselement och sammankopplingar. På så vis kan man i text beskriva komplexa komponenter såsom en mikroprocessor. Två vanliga hårdvarubeskrivande språk är VHDL och Verilog. Hårdvarubeskrivande programmering är inte samma sak som imperativ programmering i exempelvis Python, C eller assembler. Hårdvarubeskrivande språk bör snarare ses som ett textbaserat alternativ till CAD och blockdiagram för kretsdesign. En krets som är beskriven i ett hårdvarubeskrivande språk kan enkelt föras över till en FPGA eller användas vid framtagande av ett nytt kiselchip.

1.1.7 Avbrott och undantag

En utmanande del av processordesign är implementation av avbrotts- och undantagshantering [14, s. 315]. Dessa är händelser som ändrar det normala instruktionsflödet. Händelserna kan vara antingen interna eller externa. Terminologin varierar, men i RISC-V kallas externa händelser för avbrott och interna händelser för undantag. Ett exempel på ett avbrott kan således vara ett knapptryck från ett tangentbord eller en sensor som överskridit ett referensvärde. Ett undantag kan orsakas av exempelvis division med noll, ett försök till minnesåtkomst utanför det giltiga adressutrymmet eller ett försök att exekvera en odefinierad instruktion.

Hårdvarans ansvar kan förenklat beskrivas som att spara orsaken till avbrottet eller undantaget i ett register, spara adressen till instruktionen som kördes när avbrottet inträffade samt hoppa till en avbrottshanterare definierad i operativsystemet. Avbrottshanterarens ansvar är att läsa orsaken och agera därefter. Vid till exempel en I/O-förfrågan (såsom ett knapptryck) kan operativsystemet spara programmets tillstånd, betjäna I/O-förfrågan och därefter återställa programmets tillstånd för att fortsätta exekvering [14, s. 319].

1.1.8 Effektförbrukning

Effektförbrukningen i digitala kretsar brukar delas upp i dynamisk effekt och statisk effekt. Statisk effekt kommer från läckströmmar i transistorer, medan dynamisk effekt kommer från uppladdning och urladdning av kapacitanser i kretsens grindar och ledningar. Den dynamiska effekten beror således på kretsens kapacitanser och spänningar, men i synnerhet är den proportionell mot frekvensen av signalövergångar, dvs. kretsens aktivitet. Ett sätt att minska aktiviteten är att stoppa klockan till oanvända delar av kretsen, så att dess signaler inte slår om i onödan [15, s. 99].

1.1.9 SERV

SERV[1] är en liten processor skriven i hårdvarubeskrivande språket Verilog av Olof Kindgren. Den är i dagsläget världens minsta processor, sett till antalet grindar, som implementerar RISC-V-standarden. SERV implementerar i dagsläget RV32I- och Zifencei-instruktionsuppsättningarna, men har även stöd för instruktionstilläggen C, M och Zicsr. Zicsr är mest relevant för projektet och beskrivs i mer detalj i 2.1.1.

SERV-processorn uppnår sin lilla storlek genom att den är bit-seriell. Alltså, till skillnad från moderna processorer vars databuss är 32 eller 64 bitar bred, hanterar SERV en bit per klockcykel. Det innebär att en beräkning på ett 32-bitars register kräver 32 klockcykler istället för en klockcykel.

1.2 Syfte och frågeställningar

Målet är att implementera funktionalitet för att kunna hantera externa avbrott i SERV-processorn samt att implementera en mekanism för att försätta processorn i ett sleepläge och väcka SERV-processorn vid inkommande avbrott. Implementationerna ska följa gällande specifikationer inom RISC-V och ska överensstämma med SERV-projektets övergripande mål om låg implementationskostnad och god dokumentationsstandard. Syftet är att utöka SERVs funktionalitet med externa avbrott och minska energiförbrukningen med hjälp av sleep-funktionalitet. Implementationen ska testas för att verifiera funktionalitet och utvärderas för att besvara frågeställningarna nedan. Hårdvarutesterna och utvärderingen genomförs främst för hårdvaruplattformen Artix-7.

1.2.1 Frågeställningar

- Vilken implementationskostnad medför en implementation av externa avbrott och sleep för SERV?
- Vilken energibesparing medför en implemenatation av externa avbrott och sleep för SERV?

1.2.2 Avgränsningar

Vi kommer inte undersöka om implementationen är den minsta möjliga. Den ska heller inte avvika från RISC-V standarden. SERV ska inte modifieras mer än nödvändigt.

[1] `https://github.com/olofk/serv.git`

2
Teknisk bakgrund

I detta kapitel beskrivs gällande specifikationer och konventioner i RISC-V vilka ligger till grund för vår implementation av externa avbrott och sleep. Senare i kapitlet beskrivs de delar av SERV som krävs för att förstå vår implementation.

2.1 Avbrott i RISC-V

I manualen till RISC-V definieras ett avbrott som "en extern händelse som inträffar asynkront mot nuvarande RISC-V-tråd". Detta ska skiljas från undantag, som är "ett ovanligt förhållande som inträffar under exekvering associerad med en instruktion i nuvarande RISC-V-tråd". Vi likställer en tråd med en processorkärna i denna rapport (och vi arbetar bara med en sådan). Med "trap" menas mekanismen som överför kontroll till en viss rutin (trap-hanterare) vid ett avbrott eller undantag [16, avsnitt 1.6].

Avbrott hör alltså inte ihop med det normala instruktionsflödet, utan uppkommer av andra orsaker. I RISC-V finns stöd för mjukvarupåkallade avbrott, timeravbrott, externa avbrott och lokala avbrott.

RISC-V-manualen specificerar att "externa avbrott hanteras före interna (timer/mjukvara) avbrott eftersom externa avbrott brukar genereras av enheter som kan kräva korta avbrottshanteringstider" [17, s. 38].

2.1.1 Relevanta register i RV32Ziscr

Ziscr-tillägget i RISC-V definierar ett antal register, CSRs (Control & Status Registers), som underlättar hantering av olika avbrott. Sex register är särskilt intressanta för detta ändamål: `mip`, `mie`, `mstatus`, `mcause`, `mepc` och `mtvec`. Djupgående information om dessa register finns att hämta i RISC-V-manualen [17], och en kort beskrivning ges nedan.

`mip` (machine interrupt pending) – Innehåller information om vilka avbrottsorsaker som inväntar hantering. I detta register finns en bit `meip` (machine external interrupt pending) specifikt för externa avbrott.

`mie` (machine interrupt enable) – Bestämmer vilka avbrott som är aktiverade och alltså ska hanteras. I detta register finns en bit `mie.meie` (machine external interrupt enable) specifikt för externa avbrott.

`mstatus` (machine status) – Innehåller flera olika bitar för att hålla reda på processorns tillstånd, och kan bland annat användas för att tillfälligt avaktivera trap-hantering genom att nollställa biten `mstatus.mie` (machine status machine interrupt enable). Det finns en bit `mstatus.mpie` i registret som används för att spara och återställa värdet av `mstatus.mie`.

`mcause` (machine cause) – Innehåller en bit (bit 31) som berättar om det är ett avbrott (1) eller ett undantag (0) som utlöste trap-mekanismen, samt en kod som indikerar orsaken till avbrottet eller undantaget. Orsakskoden för timeravbrott är 7 och orsakskoden för externa avbrott är 11.

`mepc` (machine exception program counter) – Innehåller programräknarens värde innan senaste avbrottshantering initierades.

`mtvec` (machine trap vector base-address) – Innehåller basadressen för tabellen över avbrottsvektorer, samt information om huruvida processorns avbrottshantering är direkt eller vektoriserad (se nedan).

2.1.2 Avbrottshantering

Det finns en allmänt vedertagen metod för avbrottshantering i RISC-V. Metoden skisseras i SiFives Interrupt Cookbook [18], och genom att studera vad varje register gör kan följande beskrivning av metoden göras:

Ett avbrott utlöses genom att en bit i `mip`-registret aktiveras. Om motsvarande bit i `mie` samt `mstatus.mie`-biten är aktiverade, och så fort processorn är redo att hantera avbrottet, sparas programräknaren i `mepc`-registret. Ett logiskt uttryck tar reda på avbrottsorsaken och lägger in orsakskoden i registret `mcause`. Biten `mstatus.mie` sparas i biten `mstatus.mpie` och nollställs; detta avaktiverar hantering av nya avbrott under innevarande avbrottshanteringen. Slutligen ändras programräknaren till adressen för en avbrottsrutin med hjälp av registrena `mtvec` och `mcause`, och hur detta görs beror på om `mtvec` är konfigurerad i direktläge eller vektoriserat läge.

Om `mtvec` är konfigurerad i direktläge, skrivs adressen som finns i `mtvec` till programräknaren, och en avbrottsrutin (mjukvara) på denna adress läser `mcause`-registret för att avgöra avbrottets orsak. Om `mtvec` istället är konfigurerad i vektoriserat läge, hämtas avbrottsrutinens adress från en minnesadress som beräknas genom att orsakskoden i `mcause` tolkas som en offset som läggs till basadressen i `mtvec`-registret.

I slutet av avbrottsrutinen exekveras MRET-instruktionen. Det medför att `mepc` skrivs tillbaka till programräknaren och `mstatus.mpie` skrivs tillbaka till `mstatus.mie`.

2.2 WFI-instruktion

WFI-instruktionen beskrivs i RISC-V-manualen [17, avsnitt 3.3.3]. Enligt manualen ska instruktionen väsentligen signalera till processorn att exekvering kan avstanna tills ett avbrott signaleras. När ett aktiverat avbrott signaleras, försätts processorn i avbrottshantering som vanligt och exekvering återupptas, dvs. processorn väcks. Manualen klargör att instruktionen inte får påverkas av `mstatus.mie`-biten, men å andra sidan skall inte processorn väckas om orsaken till det signalerade avbrottet har motsvarande bit i `mie`-registret nollställd.

2.3 SERV

På grund av SERVs bit-seriella arkitektur är det behändigt att definiera instruktionssteg. Vidare finns det en modul `serv_csr` som är central i sammanhanget avbrottshantering.

2.3.1 Instruktionssteg

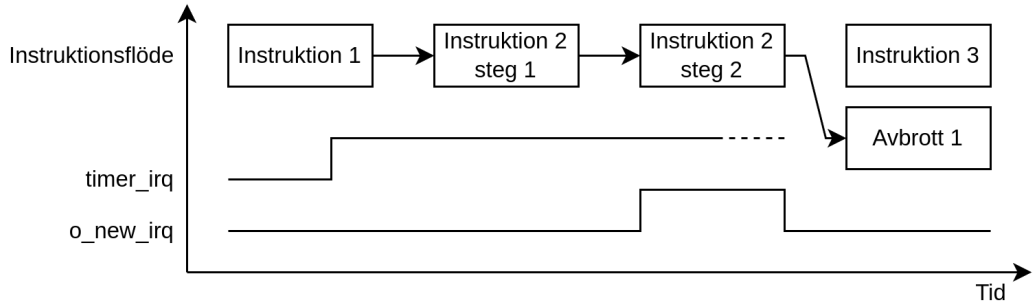
En serie av 32 konsekutiva cykler där kärnan är aktiv kallas ett steg, och de flesta instruktioner är så kallade enstegsoperationer. I SERV finns det även tvåstegsoperationer som istället utförs under 64 klockcykler, eller två steg. För att urskilja första steget från det andra används `init`-signalen som hålls hög under första steget och låg under andra. SERV använder biträknarsignaler, benämnda `cnt0` till `cnt31`, som urskiljer vilken klockcykel i instruktionssteget som exekveras.

2.3.2 Avbrott i SERV

Avbrott i SERV hanteras av modulen `serv_csr`. Modulen implementerar de delar av de fyra CSR-registren `mip`, `mie`, `mstatus` och `mcause`, som krävs för att hantera undantagen ecall, ebreak och misalignment samt timeravbrott [19]. (Ytterligare fyra CSR-register `mepc`, `mtvec`, `mtval` och `mscratch` finns implementerade annanstans i SERV.)

Inkommande avbrott signaleras i modulen `serv_csr` genom att en bit i `mip`-registret blir hög. Modulen kontrollerar om det aktuella avbrottet är aktiverat i registret `mie` samt om processorn är i läge att hantera avbrott enligt biten `mstatus.mie`. I fallet timeravbrott kommer följaktligen den interna signalen `timer_irq` att aktiveras. Signalen `timer_irq` används för att generera utsignalen `o_new_irq`, som signalerar till processorn att avbrottshanteraren ska laddas till programräknaren. Signalen `o_new_irq` aktiveras under hela nästkommande *avslutande* instruktionssteg eftersom det är då programräknaren laddas (se figur 2.1), och enbart om inget pågående avbrott är aktivt i registret `mstatus` [19].

Signalen `o_new_irq` är även en del av logiken som bestämmer värdet i `mcause`-registret; om signalen är hög så sätts bit 31 till värdet 1 (för att indikera avbrott) och koden till värdet 7 (för att indikera timeravbrott).



Figur 2.1: Tidsdiagram över SERVs avbrottshantering. Notera att instruktion 2 är en tvåstegsinstruktion.

2.3.3 Servant

Servant är en simpel System-on-a-Chip (SoC), alltså ett komplett system på ett chip, designad för SERV. Servant implementerar timer, GPIO och minne. För att kunna köra kod på SERV-processorn behövs Servant, eller annan liknande plattform, eftersom SERV själv inte har minne för sina register.

3
Metod

Kapitlet beskriver arbetets tre faser: implementation, test av implemention och utvärdering av implementation.

3.1 Implementation

Implementationen är uppdelad i tre distinkta deluppgifter: implementation av externa avbrott, implementation av WFI-instruktion och implementation av sleep. Implementationen ska skrivas i det hårdvarubeskrivande språket Verilog med byggsystemet FuseSoC [20]. FuseSoC är en pakethanterare och byggsystem för hårdvarubeskrivande språk skrivet av utvecklaren av SERV.

En generell implementationen av externa avbrott och WFI-instruktionen kommer att implementeras i SERV. Det innebär att samma implementation av externa avbrott och WFI kan användas oavsett vilken hårdvara processorn använder. Implementationen av sleep är delvis hårdvaruberoende och kommer delas in i två delar - en generell del i SERV och Servant, och en hårdvaruspecifik del i Servants topplager. Den hårdvaruspecifika implementationen i Servant kommer att utvecklas för hårdvaruplattformen Artix-7 på utvecklingskortet Nexys A7.

Under arbetet kommer testbenches användas för felsökning och funktionsverifikation. Dessa ska skrivas i C++ och länkas med Servant-projektet kompilerat med Verilator [21]. När dessa binärer körs produceras en VCD-fil som kan visas med GTKWave för att se timingdiagram. I dessa timingdiagram kan man se exakt vad processorn gör under exekvering för varje klockcykel.

3.2 Test av implementation

Testerna är en mycket viktig del av arbetet och säkerställer att implementationen fungerar. De genomförs på hela och delar av implementationen, och genom både simulations- och hårdvarutester.

3.2.1 Simulationstest

Det huvudsakliga testet av implementationen sker genom simulationstester. Dessa ger en mycket stor insikt i implementationen och möjliggör, till skillnad från hårdvarutester, en noggrann analys av ett stort antal signaler för varje klockcykel. Simulationstester genomförs med C++ testbenches, som användas för att styra processorn. Exempelprogram exekverade av processorn ska skrivas i RISC-V-assembler. Projektet anses färdigt när alla specifikationer är implementerade och verifierade.

3.2.1.1 Test av instruktionsavkodaren för WFI

För att testa implementationen av WFI-instruktionen användes en testbench skriven i C++ och Verilog. Denna testbench instansierar SERV:s instruktionsavkodningsmodul `serv_decode` som visas i kodstycke 1.

```
module serv_decode (
   input wire        clk,
   // Input
   input wire [31:2] i_wb_rdt,
   input wire        i_wb_en,
   // To state
   output reg        o_sh_right,
   output reg        o_bne_or_bge,
   ...
   output reg        o_wfi,
   ...
);
```

Kodstycke 1: `serv_decode`-modulens in- och utsignaler

Instruktionsavkodaren tar in tre signaler: klockan, `wb_rdt` och `wb_en`. `wb_en` är en enable-signal som endast säger om instruktionsavkodning ska ske eller ej, och `wb_rdt` är en databuss som överför instruktioner till avkodaren. Modulen har 45 utsignaler som skickas vidare till andra delar av processorn, till exempel `o_wfi` som säger åt processorn att stoppa klockan och vänta på ett avbrott. Avkodningsmodulen körs i testbenchen genom att klocka `clk`-signalen och köra `Vdecoder_sim::eval()`. Genom att köra `eval()` uppdateras modulens utgångar baserat på modulens ingångar, och då kan man se vilka utsignaler som sätts på och av.

Testningen görs på två sätt, ett är att köra WFI-instruktionen och verifiera att rätt utsignaler aktiveras och inga andra; här jämförs utsignalerna med EBREAK-instruktionen. Med rätt utsignaler menas `o_wfi`-signalen samt övriga utsignaler som ej påverkar dessa instruktioners funktion men ändå aktiveras av övrig logik i instruktionsavkodaren. Kodstycke 2 visar hur detta utförs. Först läggs instruktionen i `wb_rdt`, `wb_en` sätts på och sedan körs `cycle()` som utför en klockcykel och kör `eval()`.

Andra sättet är att testa alla andra instruktioner i instruktionsuppsättningarna RV32I, Zifencei och Zicsr och se till så att WFI-signalen inte blir ändrad när de körs. Många instruktioner har fält som kan variera i värde, men ett slumpmässigt valt exempel väljs för varje instruktion. För att göra det anropas `test_instruction` som har en `assert`-kontroll för just detta, se kodstycke 2. Testbenchen för avkodningsmodulen kör `test_instruction` för alla instruktioner som tillhör de nämnda instruktionsuppsättningarna.

3.2.1.2 Test av externa avbrott

För att testa att externa avbrott fungerade tidigt i utveckligen användes en testbench som kör ett RISC-V-program skrivet i assembler visat i kodstycke 3. Detta program sätter på avbrott, registrerar en avbrottshanterare, och väntar sedan för evigt i en loop. När ett externt avbrott sker så hoppar programräknaren ur loopen och kör koden i `handler`.

```
void test_instruction(Vdecoder_sim *top,
  const std::string& instr_name, uint32_t instr) {
  top->wb_rdt = instr >> 2;
  top->wb_en = 1;
  cycle(top);
  outfile << "ebreak: " << std::to_string(top->ebreak) << std::endl;
  outfile << "wfi: " << std::to_string(top->wfi) << std::endl;
  outfile << "sh_right: " << std::to_string(top->sh_right) << std::endl;
  ...
  assert(!(top->wfi && instr_name != "wfi"));
```

Kodstycke 2: Funktionen `test_instruction` loggar utsignaler från avkodningsmodulen

```
_start:
    // sätt mtvec till handleradress
    la t0, handler
    csrw mtvec, t0
    // sätt på externa avbrott
    li t0, 0x8
    csrrs x0, mstatus, t0
    li t0, 0x800
    csrrs x0, mie, t0
wait:
    // vänta i en loop
    j wait
handler:
    // hoppa tillbaka till wait-loopen.
    mret
```

Kodstycke 3: RISC-V-kod för att aktivera och vänta på externa avbrott med jump-loop

Testbenchen för att kontrollera att det fungerar är väldigt simpel: den väntar ett tag för programmet att fastna i loopen wait; sedan sätts avbrottssignalen på och då väntar testbenchen på att `mret`-signalen ska bli hög samtidigt som programräknaren skrivs ut på skärmen för att manuellt se att programmet hoppar korrekt.

3.2.1.3 Test av timeravbrott med sleep genom WFI

För att testa att WFI-instruktionen fungerade och att processorn kunde komma ur sleep användes en testbench som sätter på ett timeravbrott med `mstatus.mie`-, `mie.mtie`-, och `mtime`-registren. Processorn försätts sedan i sleep med WFI-instruktionen och vaknar när timeravbrottet inträffar.

Programmet i kodstycke 4 verifieras med en testbench som skriver ut klockan `main_clk`, som går in i Servant, tillsammans med programräknaren. Signalen `main_clk` stängs inte av vid sleep men grindas i toppfilen för Servant med utgång `wb_clk`. Man kan då se att WFI-instruktionen körs vid `main_clk` klockcykel 300, och lite efter `main_clk` klockcykel 3 000 körs koden i `handler`.

```
_start:
        // sätt mtvec till handleradress
        la t0, handler
        csrw mtvec, t0
        // ladda 3000 till mtime
        li t0, 0x80000000
        li t1, 3000
        sw t1, 0(t0)
        // sätt på timeravbrott
        li t0, 0x8
        csrrs x0, mstatus, t0
        li t0, 0x80
        csrrs x0, mie, t0
        // stäng av klocka och vänta på avbrott
        wfi
handler:
        // stäng av timeravbrott
        li t0, 0x8
        csrrc x0, mstatus, t0
        li t0, 0x80
        csrrc x0, mie, t0
        // hoppa tillbaka till wfi och sov tills vidare
        mret
```

Kodstycke 4: RISC-V-kod för att aktivera timeravbrott och vänta med klockan avstängd

3.2.2 Hårdvarutest

Vissa funktioner i implementationen kan inte testas av simulationstester då de interagerar direkt med hårdvara som inte går att simulera. En sådan funktion är topplagret i Servant som interagerar direkt med klockhårdvaran i Artix-7. Hårdvarutesterna kompletterar simulationstesterna genom att testa hårdvaruspecifika funktioner på riktig hårdvara.

Testerna genomförs på utvecklingskortet Nexys A7 med en logikanalysator genom detaljerad analys av specifika signaler när processorn exekverar olika testprogram. Testprogrammen är designade för att testa de olika funktioner som implementerats i projektet och komplettera simulationstesterna.

3.2.2.1 Test av sleep och wakeup

Den främsta funktionaliteten som inte kan testas i simulationstester är sleep och wakeup med en riktig hårdvaruklocka. Testerna utförs på den fullständiga implementationen och verifierar att systemet fungerar i sin helhet.

Testet består av tre delar: en för sleep genom WFI, en för wakeup genom externa avbrott och en för wakeup genom timer-avbrott. Testerna genomförs genom att försätta processorn i sleep och sedan väckas av timer-avbrott och externa avbrott. När processorn försätts i sleep och vaknar, analyseras klocksignaler och andra relevanta signaler (främst `sleep_req` och `wakeup_req`) för att verifiera att implementationen fungerar som förväntat.

3.3 Utvärdering av implementation

För att besvara frågeställningarna måste implementationen utvärderas ur två perspektiv: implementationskostnad och energibesparing.

3.3.1 Implementationskostnad

Implementationskostnaden av hela designen mäts med verktyget `Report Utilization` efter syntes och implementation i Vivado, med en klockperiod på 10 ns. Jämförelsevis mäts implementationskostnaden även för den icke-modifierade designen. Jämförelsemåttet är antal "slice LUTs" och "slice registers" som visas i Vivado, dvs. antal LUTs och vippor.

3.3.2 Energibesparing av sleep

Utvärderingen av energibesparingen av sleep genomförs både genom simulering och genom mätning på hårdvara.

3.3.2.1 Energibesparing i Vivado

Energibesparingen av sleep mäts både i Vivado och genom mätningar på hårdvara. Mätningen i Vivado genomförs med en testbench-simulation. Det går ut på att testbenchen startar processorn med samma kod som i kodstycke 3. Sedan körs testbenchen tills signalen `mret` blir aktiv. Tiden från att processorn startas och ett avbrott uppstår specificerades till 3 000 000 klockcykler, och för jämförelse är initialisering ca 1 000 klockcykler. Det betyder att det huvudsakliga bidraget till den genomsnittliga strömförbrukningen kommer komma från att vänta på avbrott. När denna testbench körs så sparas information om alla signaler i en datafil som kan användas av Vivados verktyg `Report Power` för att beräkna en genomsnittlig strömförbrukning.

3.3.2.2 Energibesparing på Nexys A7

Hårdvarumätningen genomförs med en amperemeter på 5 V-matningsspänningen till Nexys A7. Mätningen genomförs med hjälp av oscilloskop, amperemeter och logikanalysator (se appendix C för bild på mätutrustningen).

Mätningen utförs på två versioner av SERV: en med både sleep och externa avbrott och en med enbart externa avbrott. Versionen med både sleep och externa avbrott testas när den är i sleep och när den kör ett testprogram i form av en testloop (`loop: j loop`). Versionen med enbart externa avbrott testas genom att köra testloopen.

Nexys A7 har en basström som räknas bort från den totala strömförbrukningen för att få strömförbrukning av SERV. Basströmmen mäts genom att programmera Nexys A7 med en helt tom design. Strömbesparingen beräknas genom en jämförelse mellan strömförbrukningarna av SERV med både sleep och externa avbrott och SERV med enbart externa avbrott.

Utöver strömförbrukningen av SERV, kommer också strömförbrukningen av klockgenereringen för de två versionerna att mätas. Mätningarna kommer att genomföras med respektive klockgenereringskrets i en tom design där klockan dras in i en avaktiverad klockbuffer (`BUFGCE`). Klockbuffrarna säkerställer att klocksignalen inte driver några klocknät så att enbart strömförbrukningen från klockgeneratorerna mäts.

3.4 Övriga verktyg

Versionshantering av arbetets kod ska hanteras av versionshanteringssystemet Git. För att underlätta att dela kod ska GitHub användas som plattform. Under arbetet ska gruppen skapa en egen gren av SERV för implementering av externa avbrott tillgänglig öppet på GitHub[1]. Grenen ska senare sammanslås med huvudprojektet.

Verktyget draw.io används för strukturella diagram och verktyget matplotlib används för grafer.

[1] `https://github.com/alfredfo/serv.git`

4
Resultat

Resultatet presenteras i fyra delar: beskrivning, simulationstest, hårdvarutest och utvärdering av implementationen.

4.1 Implementation

Implementationen är uppdelad i tre delar: externa avbrott, WFI-instruktion och sleep.

4.1.1 Externa avbrott

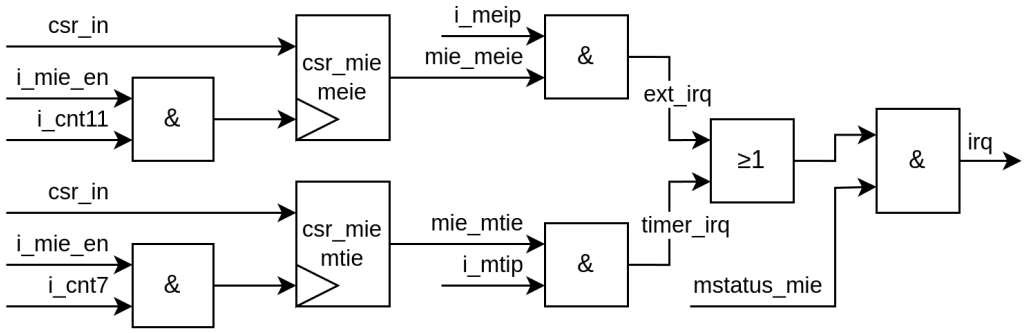
En extern enhet signalerar ett externt avbrott genom att slå om signalen `meip` (machine external interrupt pending) till hög. Det är den externa enhetens ansvar att buffra avbrottssignalen så att `meip` hålls hög tills att SERV har hunnit slå om signalen `o_new_irq` till hög. Signalen `meip`, likt `mtip` (machine timer interrupt pending), går in i modulen `serv_csr` som genererar utsignalen `o_new_irq` och uppdaterar registret `mcause`. I det följande beskrivs de relevanta signaler och kretsar som finns inuti `serv_csr`.

4.1.1.1 Signalen o_new_irq

CSR-registret `mie` är implementerat med biten `mtie` (machine timer interrupt enable) och biten `meie` (machine external interrupt enable). Biten `meie` togs fram genom att duplicera implementationen av `mtie`, med enda modifieringen att `meie` utnyttjar biträknarsignalen `cnt11` istället för `cnt7`, eftersom `meie`-biten finns på bit 11 i registret `mie`. Således är båda dessa bitar är implementerade som vippor som aktiveras vid en skrivning till `mie`-registret och mer specifikt under klockcykeln då SERVs biträknare signalerar att motsvarande bit behandlas. Se figur 4.1.

Externa avbrott utlöses genom att insignalen `meip` aktiveras. Om även `meie` är aktiverad, genereras signalen `ext_irq`. På analogt sätt genereras signalen `timer_irq`. Om någon av signalerna `ext_irq` och `timer_irq` är höga, genereras en gemensam avbrottssignal `irq`. Då `mstatus_mie` är aktiverad, går `irq`-signalen vidare till en krets som genererar utsignalen `o_new_irq`. Kretsen säkerställer att `o_new_irq` aktiveras vid rätt tidpunkt för att ladda avbrottshanteraren till programräknaren (se avsnitt 2.3.2). Kretsen använde signalen `timer_irq` tidigare, men kunde enkelt modifieras så att den använder den gemensamma avbrottssignalen `irq` som beskrivet. Se figur 4.1.

Till skillnad från den ursprungliga implementationen, går inte längre signalen `mstatus_mie` in i grinden som genererar `timer_irq`; denna signal har flyttats till den gemensamma grinden som genererar `irq` för att kunna spara på en onödig grind.



Figur 4.1: En överblick av implementationen av signalen `irq`, som används för att generera signalen `o_new_irq`.

4.1.1.2 Registret mcause

Varje bit i registret `mcause` i `serv_csr` bestäms av en logisk summa. Detta gjorde det enkelt att modifiera enskilda termer för att stödja externa avbrott. I detta avsnitt beskrivs enbart den term i varje sådan summa som relaterar till avbrott.

Bit 31 i `mcause` finns implementerad i `serv_csr` som ett register. Den läser sedan tidigare signalen `o_new_irq` för att skilja mellan avbrott och undantag vid en trap. Ingen ändring behövde göras.

Bit 0-3 i `mcause` finns implementerade som ett fyr-bitars register, och lagrar orsaken till avbrottet eller undantaget. Ett timeravbrott skriver kod 7 (binärt 0111) till registret och ett externt avbrott skriver kod 11 (binärt 1011) till registret enligt nedan.

Bit 0 och 1 sätts båda till den gemensamma avbrottssignalen `o_new_irq` eftersom båda koder har värdet 1 i dessa bitar. Bit 3 sätts till `o_new_irq & !ext_irq` eftersom detta indikerar ett timeravbrott. Bit 4 sätts till `o_new_irq & ext_irq` eftersom detta indikerar ett externt avbrott. Valet att läsa `ext_irq` hellre än `timer_irq` medför att externa avbrott prioriteras över timeravbrott.

4.1.2 WFI-instruktion

Likt övriga instruktioner implementerades WFI-instruktionen som ett utsignalsregister i instruktionsavkodaren, där registrets inmatning är värdet av ett booleskt uttryck. För att unikt identifiera WFI-instruktionen bland de redan implementerade instruktionerna, räcker det att kontrollera att opkod bit 4 och opkod bit 6 är satta till 1, att `funct3`-fältet är tomt, och att bit 22 i instruktionsordet är satt till 1. I SERV-intern kod blir detta uttryck:

```
opcode[4] & opcode[2] & op22 & !(|funct3).
```

(Notera att SERV kapar bort de två minst signifikanta bitarna i opkoden eftersom de har samma värde i samtliga instruktioner som stöds i SERV, så t.ex. `opcode[4]` motsvarar opkod bit 6.)

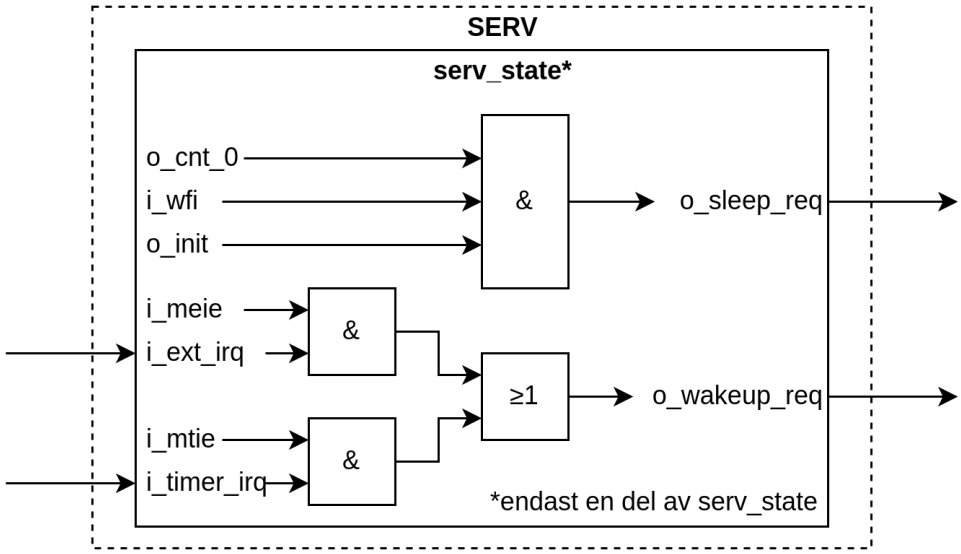
WFI-instruktionen är en tvåstegsinstruktion för att undvika problem med SERVs instruktionshämtning (se avsnitt 5.1.2).

4.1.3 Sleep

Implementationen av sleep är uppdelad mellan SERV och referensplattformen Servant. Implementationen i SERV avgör *när* sleep ska aktiveras medan implementationen i Servant avgör *hur* sleep aktiveras.

4.1.3.1 Sleep i SERV

Sleep aktiveras med hjälp av två nya styrsignaler från SERV, `sleep_req` och `wakeup_req`. Signalerna implementeras i den existerande modulen `serv_state` (se figur 4.2). Signalen `sleep_req` aktiveras i första klockcykeln under en WFI-instruktion och signalerar till Servant att processorn begär sleep. Signalen `sleep_wakeup` aktiveras vid ett inkommande avbrott från en avbrottskälla och signalerar till Servant att processorn bör vakna från sleep. En förutsättning för att en avbrottskälla ska kunna väcka processorn är att avbrottet är aktiverat. Signalen `wakeup_req` aktiveras således enbart om motsvarande bit är aktiv i CSR-registret `mie`.



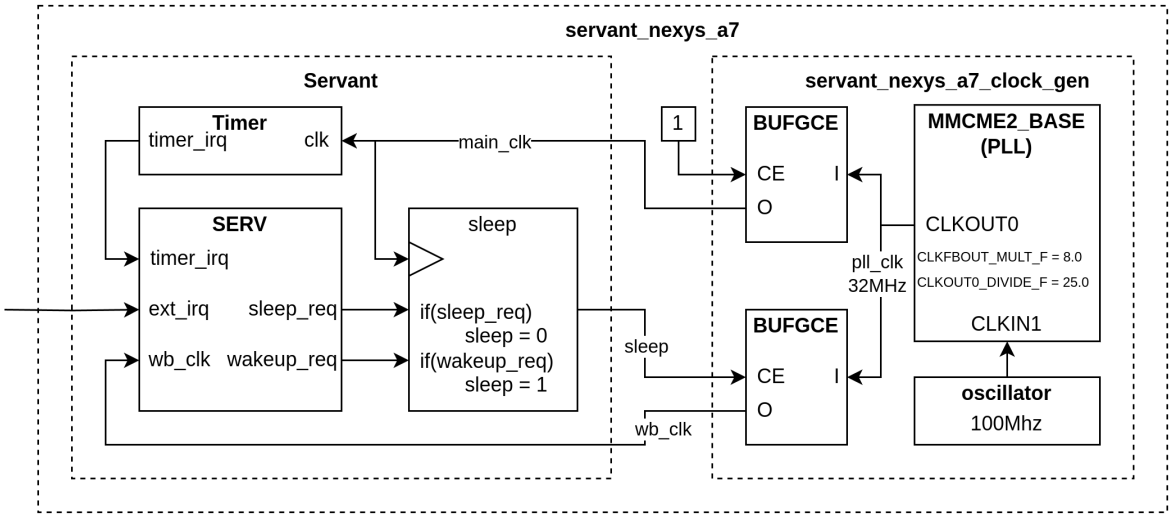
Figur 4.2: En överblick av den del av modulen `serv_state` som implementerar de två styrsignalerna för sleep. Notera att implementationen är helt asynkron.

4.1.3.2 Sleep i Servant

Servant är uppdelad i två nivåer: ett baslager och ett topplager. Baslagret hanterar alla gemensamma grundläggande funktioner i Servant. Topplagret interagerar direkt med hårdvaran och implementeras därför olika beroende på vilken hårdvaruplattform man använder. En överblick av hela Servant finns i figur 4.3.

Den gemensamma implementationen i baslagret innehåller en sleep-vippa och två klocksignaler, `wb_clk` och `clk_main`. Sleep-vippan styr sleep-funktionaliteten och aktiveras vid sleep-begäran och avaktiveras vid wakeup-begäran. Klocksignalen `wb_clk` är kopplad till SERV och används för att klocka processorn. Klocksignalen `clk_main` klockar de funktioner i Servant som alltid behöver vara aktiverade. Dessa innefattar timer-klockan och sleep-vippan, som bidrar till att SERV vaknar från sleep.

Hårdvaruplattformen Artix-7, som används i projektet, har inte stöd för att försätta delar av FPGA:n i strömlöst tillstånd. Sleep implementerades således genom att enbart inaktivera `wb_clk` med hjälp av en klockbuffer (`BUFGCE`). För att undvika synkroniseringsproblem mellan de två klocksignalerna implementerades även en klockbuffer för `main_clk`, som alltid är aktiverad, i enlighet med klockmanualen för Artix-7 [22].



Figur 4.3: En överblick av Servant för utvecklingsplattformen Nexys A7 där sleep är implementerat för FPGA:n Artix-7.

4.2 Simulationstest

Simulationstesterna testar all grundläggande funktionalitet i vår implementation.

4.2.1 Test av WFI-instruktionen och `o_wfi`-signalen

I testerna för `o_wfi` signalen verifierades det att alla instruktioner förutom WFI i instruktionsuppsättningarna RV32I, Zicsr och Zifencei inte satte `o_wfi`-utsignalen aktiv i SERVs instruktionsavkodare. När WFI-instruktionen testades så jämfördes utsignalerna med EBREAK, och enda skillnaden var då att utsignalen `o_wfi` var hög och `o_ebreak` var låg, vilket är det korrekta resultatet.

4.2.2 Externa avbrott

Funktionen av externa avbrott verifierades genom att se så att processorn hoppar till avbrottshanteraren efter en viss tidsperiod som sätts i testbenchen. I detta fall aktiveras `ext_irq` signalen efter 6 000 klockcykler. I utdata 4.1 så visas det hur programräknaren hoppar till avbrottshanteraren vid address 0x60 efter ungefär 6 000 klockcykler, alltså fungerar externa avbrott.

```
967  |  48
1103  |  48
...
6135  |  48
6207  |  60
6279  |  64
6351  |  68
6423  |  72
6495  |  76
6567  |  48
mret detected, exiting
```

Utdata 4.1: Klockcykel och programräknare i testbench för externa avbrott

4.2.3 Sleep

På samma sätt som för externa avbrott verifieras sleep med hjälp av samma testbench och assemblerkod, där skillnaden är att WFI-instruktionen användes istället för en loop. Då körs inga instruktioner mellan WFI och avbrottshanteraren, bara andra steget av WFI, därför syns endast 48 två gånger i utdata 4.2 till skillnad från utdata 4.1. Detta är exakt vad som förväntas eftersom inga instruktioner ska köras medan processorn sover. Liknande test har också gjorts med timeravbrott och givit samma resultat.

```
967  |  48
6163  |  48
6235  |  60
6307  |  64
6379  |  68
6451  |  72
6523  |  76
6595  |  48
mret detected, exiting
```

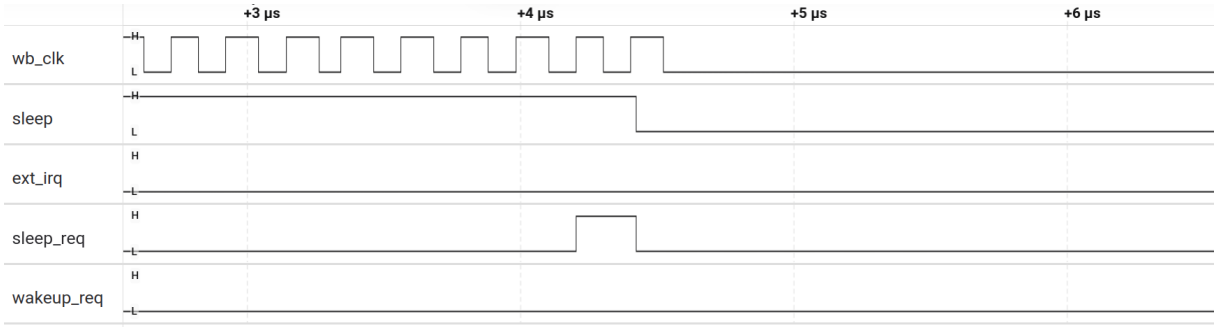
Utdata 4.2: Klockcykel och programräknare i testbench för sleep

4.3 Hårdvarutest

Hårdvarutesterna kompletterar simulationstesterna och visar att implementationen fungerar även på hårdvara. Testerna utförs på en Artix-7 med den fullständiga implementationen, vilket verifierar att systemet fungerar i sin helhet.

4.3.1 SERV försätts i sleep av WFI-instruktionen

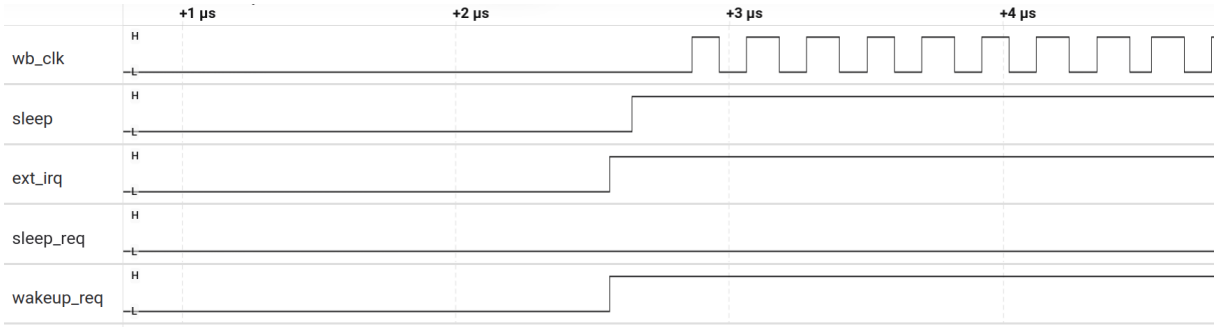
Grafen i figur 4.4 visar att signalen `sleep_req` försätter SERV i sleep efter att processorn exekverat en WFI-instruktion.



Figur 4.4: SERV försätts i sleep genom WFI-instruktionen.

4.3.2 SERV vaknar av externa avbrott

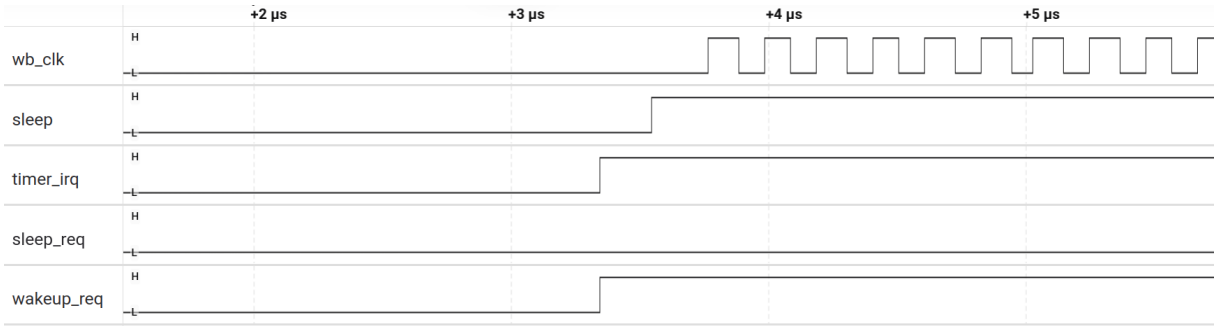
Grafen i figur 4.5 visar att en sovande SERV vaknar vid externa avbrott av att signalen `wakeup_req` aktiveras. I testet exekveras många efterföljande avbrott eftersom det externa avbrottet är aktivt under en längre period. När avbrottet är hanterat försätts SERV åter i sleep med hjälp av en WFI-instruktion. Implementationen fungerar som förväntat eftersom avbrott hanteras så länge det externa avbrottet är aktivt och SERV inte försätts i sleep förrän alla avbrott är hanterade.



Figur 4.5: SERV vaknar av ett externt avbrott.

4.3.3 SERV vaknar av timeravbrott

Grafen i figur 4.6 visar att SERV vaknar av ett inkommande timer-avbrott som aktiverar signalen `wakeup_req`. När avbrottet är hanterat försätts SERV åter i sleep med hjälp av en WFI-instruktion.



Figur 4.6: SERV vaknar av ett timeravbrott.

4.4 Utvärdering

Implementationen utvärderas baserat på totala implementationskostnaden och energibesparingen som medförs av sleep-funktionalitet i SERV.

4.4.1 Implementationskostnad

Syntes och implementation i Vivado visade att vår design använder 255 (slice) LUTs och 236 (slice) register, medan den icke-modifierade designen använder 250 (slice) LUTs och 234 (slice) register. Med andra ord krävde externa avbrott, WFI och sleep sammanlagt, en ökning med 2 % mer hårdvara i form av LUTs och 0,8 % mer hårdvara i form av register (vippor).

4.4.2 Energibesparing av sleep

Energibesparingen av sleep mättes både på hårdvara och genom simulering.

4.4.2.1 Energibesparing i Vivado

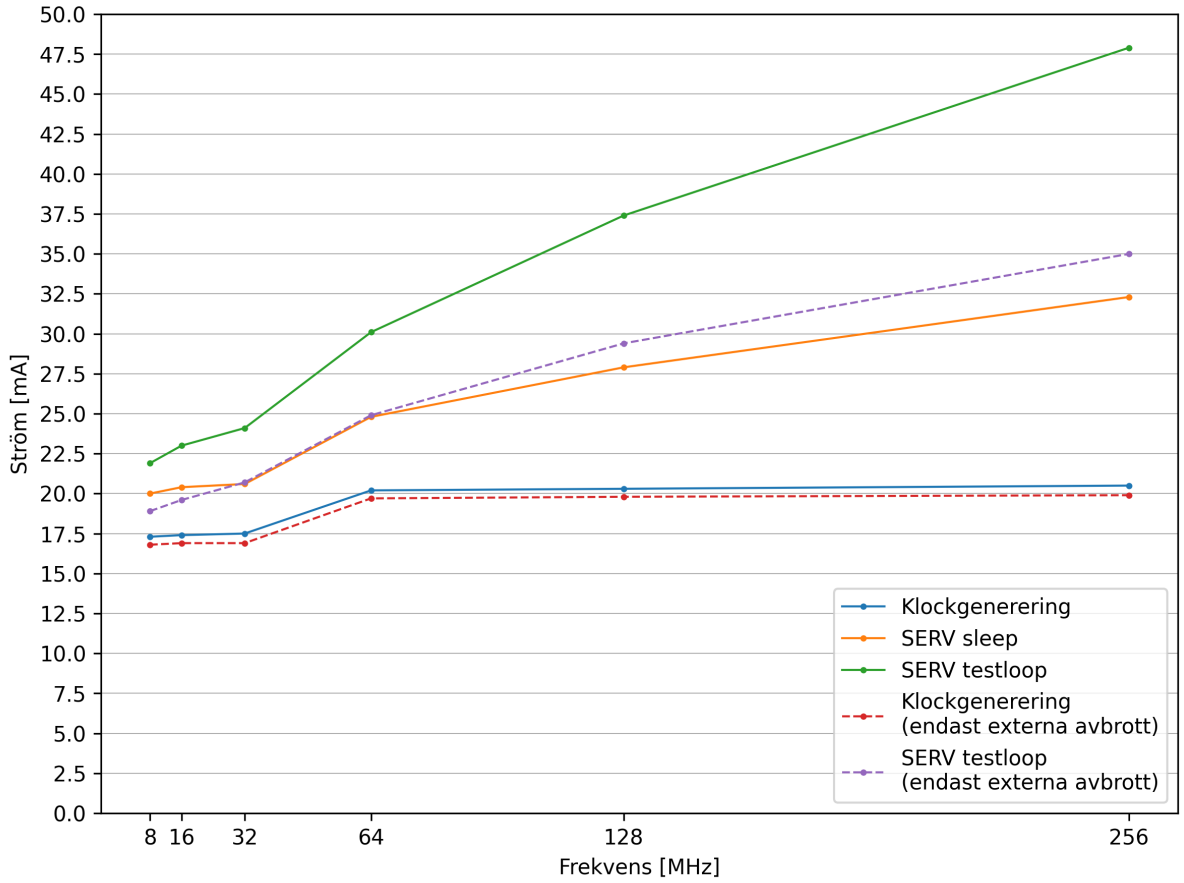
Simulering i Vivado gav inget resultat med mätbar skillnad. Den dynamiska effektförbrukningen rapporterades som 0,098 W både med loop och med WFI. Vivados effektrapportering visas i appendix B.

4.4.2.2 Energibesparing på Nexys A7

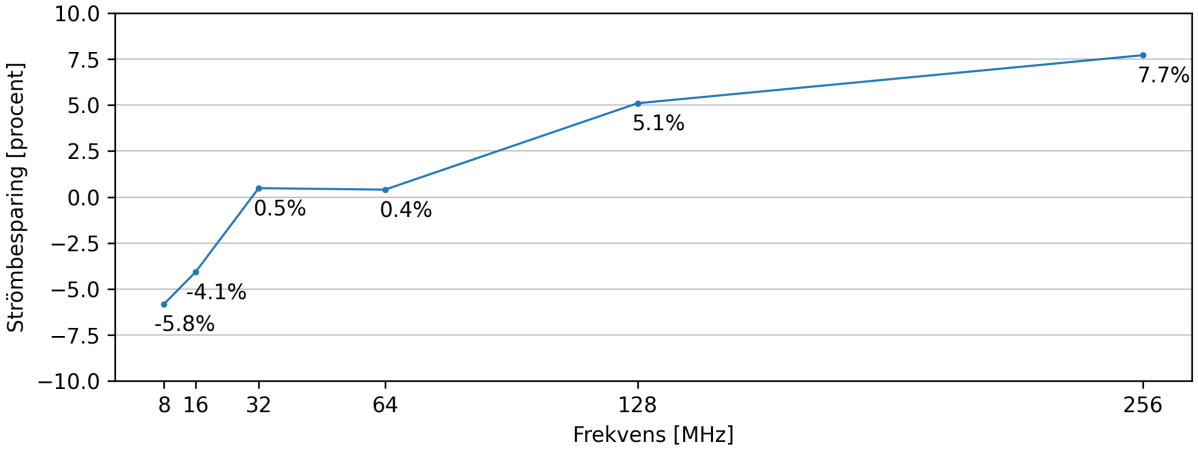
Hårdvarumätningen genomfördes på 5 V-matningen till Nexys A7. Mätningarna genomfördes på två olika versioner av vår implementation - en med både sleep och externa avbrott, och en med enbart externa avbrott. Mätresultaten redovisas i sin helhet i appendix A. Mätvärderna ligger till grund för graferna i figur 4.7 och 4.8. Nexys A7 har en basförbrukning på ungefär 120,7 mA utan SERV och utan klockgenerering. Basförbrukningen har räknats bort från graferna för att tydliggöra SERVs strömförbrukning.

Grafen i figur 4.7 visar strömförbrukningen vid olika klockfrekvenser för de två versionerna av SERV på Nexys A7. Den största delen av strömförbrukningen består av klockgenereringen. Klockgenereringen är mestadels konstant, vilket resulterar i en högre strömbesparing desto snabbare klockfrekvens SERV har eftersom implementationen då förbrukar mer ström.

Grafen i figur 4.8 visar den procentuella strömbesparingen av SERV med både sleep och externa avbrott jämfört med SERV med endast externa avbrott på Nexys A7. Det visar sig att implementationen av SERV med både sleep och externa avbrott endast ger en märkbar energibesparing i frekvensspannet 64 - 256 MHz och att strömförbrukningen faktiskt blivit högre i frekvensspannet 8 - 32 MHz.



Figur 4.7: Strömförbrukning för två versioner av SERV på Nexys A7.



Figur 4.8: Procentuell strömbesparing av SERV med både sleep och externa avbrott jämfört med SERV med enbart externa avbrott på Nexys A7.

5
Diskussion

I detta kapitel diskuteras designprocessen som gav upphov till implementationen och resultaten angående implementationskostnad och strömbesparing.

5.1 Designprocess

Under utvecklingsdelen av projektet uppstod flera designval, utmaningar och problem. Nedan följer en redogörelse för dessa tillsammans med en utvärdering av resultatet.

5.1.1 Implementation av externa avbrott

Från början hade vi flera förslag på hur externa avbrott skulle fungera. Förslagen var bland annat att avbrottet skulle kunna aktiveras på positiv/negativ flank eller att man skulle stödja flera olika avbrottskällor. Vi insåg dock snabbt att dessa förslag skulle leda till en onödigt stor implementation och på grund av designprinciperna i SERV valdes istället den minsta möjliga implementationen som följer RISC-V standarden; alltså att `meip` är en input till SERV. Flanktriggning och stöd för flera olika avbrottskällor bör istället implementeras utanför SERV och Servant vid behov.

5.1.2 Implementation av WFI-instruktion

För att härleda det booleska uttrycket, gjordes en genomgång av opkoder för de instruktioner som stöds i SERV. Det visade sig att opkod bit 6 och opkod bit 4 är samtidigt satta till 1 endast i instruktionerna EBREAK, ECALL, MRET och WFI samt CSR-instruktioner. CSR-instruktionerna har åtminstone någon bit i `funct3`-fältet satt till 1. Ingen av EBREAK, ECALL och MRET har bit 22 i instruktionsordet satt till 1. Således återstår bara WFI och ett unikt uttryck för instruktionen. Härledningen av uttrycket ovan var inte svår i detta fall. Den ger ett mycket litet uttryck som bevisligen bara kan uppfyllas av WFI.

Under utvecklingen av WFI i kombination med sleep upptäckte vi att den nästkommande instruktionen efter WFI exekverades direkt efter att processorn vaknat, innan avbrottet hanterats. Anledningen till detta är att SERV hämtar nästa instruktion under nuvarande instruktions sista steg. Då WFI vid den här tidpunkten var en enstegsinstruktion hade inhämtningen av nästa instruktion redan påbörjats när SERV vaknade av ett avbrott. Problemet löstes genom att omvandla WFI till en tvåstegsinstruktion. Då somnade SERV under första delen av instruktionen och inledde andra delen av instruktionen först efter att avbrottet inkommit, vilket resulterade i att instruktionen för avbrottshanteraren inhämtades istället för den nästkommande instruktionen.

Omvandlingen av WFI från en enstegs- till en tvåstegsinstruktion medför en något längre exekveringstid men resulterar i en mycket låg implementationskostnad på endast en grind. Att istället göra en bättre, men mer omfattande, lösning skulle innebära en betydligt större implementationskostnad, vilket inte är önskvärt i projektet.

5.1.3 Implementationen av sleep

Den initiala planen var att implementera sleep direkt i SERV, men det visade sig tidigt att detta var omöjligt då SERV inte har någon kontroll över specifika hårdvaruplattformar. En mycket bättre lösning var därför att implementera sleep-mekanismen i referensplattformen Servant för varje hårdvaruplattform och låta SERV signalera när den vill sova och vakna.

En utmaning i implementationen av Servant var att både sleep-vippan och Servants timer måste vara klockade för att processorn ska kunna vakna från sleep vid timer eller externa avbrott. Problemet löstes dock genom införandet av en ny klocka, `main_clk`, som alltid är aktiverad. Klockan `main_clk` driver således funktioner som alltid måste vara aktiva i processorn. SERV-processorns ordinarie klocka, `wb_clk`, stängs av för att försätta processorn i sleep.

En annan utmaning var införandet av klockbuffern på `wb_clk` i Servants topplager. Det fanns en risk att klockbuffern inför en viss fördröjning i klockpulserna, vilket kan ge upphov till ett synkroniseringsproblem där de båda klockorna hamnar ur synk. Problemet löstes genom att införa en klockbuffer även på `main_clk`.

5.2 Implementationskostnad

Under implementationen i `serv_state` identifierades en optimering som gick ut på att *inte* kontrollera om avbrottet var aktiverad i CSR-registret `mie` innan signalen `wakeup_req` aktiverades. Optimeringen skulle eventuellt kunna spara två grindar, vilket är önskvärt i projektet. En risk med optimeringen är dock att processorn skulle kunna väckas felaktigt av en oavsiktlig signal. Ett exempel på det är om signalen för externa avbrott alltid är aktiv. Vi valde därför att inte implementera optimeringen.

5.3 Energibesparing av sleep

Här analyseras den uppmätta strömförbrukningen på Nexys A7 och framtida förbättringsmöjligheter.

5.3.1 Mätning av strömförbrukningen

Vi misstänker att mätnoggrannheten i Vivados effektrapportering inte är tillräckligt noggrann för att mäta energibesparingen på SERV. Hela 97 % av effekten förbrukas av FPGA:ns klockgenerering; övriga delar ligger omkring tröskelvärdet 0,001 W. Det är alltså möjligt att en relativt stor besparing rundas av i rapporteringen.

Hårdvarumätningen var dock tydligare; implementationen av sleep kan ge upphov till en energibesparing på Nexys A7 under vissa förhållanden. Det faktiska uppmätta värdena är dock mindre användbara då de uppmättes på 5 V-matningen innan regulatorn på Nexys A7, vilket gör det svårt att uppskatta den faktiska strömförbrukningen på FPGA:n.

5.3.2 Analys av strömbesparingen på Nexys A7

Resultatet visar att strömbesparingen är negativ innan klockhastigheten når 32 MHz och positiv efter 64 MHz. Vi har inte lyckats identifiera någon anledning till varför strömbesparingens tecken beror på klockfrekvensen. Vi misstänker dock att det beror på en förändring i sättet klockan hanteras mellan 32 MHz och 64 MHz baserat på den plötsliga höjningen i den annars konstanta strömförbrukningen av klockgenereringen.

De främsta bidragande orsakerna till strömförbrukningen i sleep på Nexys A7 var klockgenereringen, strömsatta grindar och drivandet av klocknäten. Klockgenereringens strömförbrukning var relativt konstant, men ändå en mycket stor del av den totala strömförbrukningen vid låga frekvenser. De konstant strömsatta grindarna bidrog till (tillsammans med många periferienheter) till den höga basströmmen på Nexys A7. Drivandet av klocknäten visade sig vara förhållandevis oberoende av antalet vippor och grindar i implementationen eftersom strömförbrukningen (utöver klockgenereringen) fördubblas när både `main_clk` och `wb_clk` är aktiva jämfört med enbart `main_clk`. Detta trots att `main_clk` endast driver en mycket liten del av implementationen.

Energibesparingen av sleep hade troligtvis kunnat bli betydligt högre om implementationen hade utförts på hårdvara med effektivare klockhantering och stöd för att försätta delar av implementationen i strömlöst läge.

Strömförbrukningen när SERV inte var i sleep visade sig vara betydligt högre än både SERV i sleep och SERV med endast avbrott. Även denna strömförbrukningen kan förklaras av klockhanteringen på Artix-7 eftersom ett separat klocknät måste drivas. Om SERV är i sleep en stor del av exekveringstiden kommer den högre strömförbrukningen inte spela särksilt stor roll eftersom den bidrar till en väldigt liten procentuell energiförbrukning. Den höga strömförbrukningen kan däremot bidra till en ökad energiförbrukning jämfört med SERV med endast avbrott om SERV är i sleep endast en liten del av exekveringstiden.

5.3.3 SERV i sleep på annan hårdvara

Ett av arbetets främsta syften var att implementera en mekanism för att försätta SERV i sleep. Vår implementation av sleep i Servant och SERV styr endast *när* sleep ska aktiveras och säkerställer att detta sker på ett korrekt sätt. Exakt *hur* sleep implementeras är olika för olika hårdvaruplattformar och avgör helt vilken energibesparing som är möjlig. Strömbesparingen skulle alltså kunna bli betydligt högre på annan hårdvara.

5.3.4 Dynamisk klockfrekvens

Under arbetets gång undersökte vi om en sänkning av klockhastigheten för `main_clk` kunde resultera i en sänkt strömförbrukning vid sleep. Det visade sig dock snabbt att införandet av en ny klockfrekvens resulterade i att klockhårdvaran i Artix-7 förbrukade mer ström än innan, vilket blev mycket märkbart i vår minimala implementation.

Vi upptäckte också att vår implementation inte stödjer olika klockfrekvenser på `wb_clk` och `main_clk` eftersom Servants timer och sleep-vippan korsar de två klockdomänerna. Implementationen skulle kunna modifieras för att stödja olika och dynamiska klockfrekvenser, men det skulle resultera i en större implementation som inte bidrar till en lägre energiförbrukning.

6
Slutsats

Arbetet anses delvis lyckat eftersom externa avbrott och sleep-funktionalitet har implementerats, testats och utvärderats enligt målet, men att strömförbrukningen i vissa fall blev högre när SERV var i sleep jämfört med en SERV med enbart externa avbrott på Artix-7.

Avsnitt 4.1 i resultatet visar att externa avbrott, WFI-instruktionen och sleep har implementerats enligt RISC-V-standarden. Avsnitt 4.2 visar att implementationen för externa avbrott och WFI-instruktionen har testats och verifierats med hjälp av grundliga simulationstester. Avsnitt 4.3 visar att sleep-funktionaliteten och implementationen som helhet har testats och verifierats av hårdvarutester på hårdvaruplattformen Artix-7. Avsnitt 4.4 visar att implementationen har utvärderats baserat på implementationskostnad och strömförbrukning för Artix-7.

Implementationen av externa avbrott, WFI-instruktion och sleep-funktionalitet förmedlar en mekanism för att hantera externa avbrott, försätta SERV i sleep-läge och att väcka SERV vid inkommande avbrott. Funktionsverifikationen säkerställer att implementationen fungerar. Utvärderingen besvarar frågeställningarna. Dessa uppfyller tillsammans projektets mål.

Frågeställningen ”Vilken implementationskostnad medför en implementation av externa avbrott och sleep för SERV?” besvaras med att implementationen resulterade i en ökning med 5 LUTs och 2 vippor för Artix-7.

Frågeställningen ”Vilken energibesparing medför en implementation av externa avbrott och sleep för SERV?” besvaras med att implementationen resulterade i en tveksam strömbesparing för Nexys A7. Den procentuella strömbesparingen visade sig vara ungefär -6 % vid 8 MHz och runt 8 % vid 256 MHz klockfrekvens. Den främsta orsaken till den låga strömbesparingen är klockhanteringen på Artix-7. Den potentiella energibesparingen skulle dock kunna bli betydligt högre på annan hårdvara med en effektivare klockhantering och stöd för att försätta delar av designen i strömlöst läge.

Referenser

[1] IEEE. ”ENIAC: The World’s First Computer.” (2023), URL: https://life.ieee.org/eniac-the-worlds-first-computer/ (hämtad 2025-05-06).

[2] AT&T Tech Channel. ”The Transistor: a 1953 documentary, anticipating its coming impact on technology.” (2015), URL: https://youtu.be/V9xUQWo4vN0?si=cfbFep9Asn2ySxaU (hämtad 2025-05-06).

[3] Kungl. Vetenskapsakademien. ”Pressmeddelande: Nobelpriset i fysik 2000.” (2000), URL: https://www.nobelprize.org/prizes/physics/2000/9719-pressmeddelande-nobelpriset-i-fysik-2000/ (hämtad 2025-04-29).

[4] IEEE. ”Chip Hall of Fame: Intel 4004 Microprocessor > The first CPU-on-a-chip was a shoestring crash project.” (2024), URL: https://spectrum.ieee.org/chip-hall-of-fame-intel-4004-microprocessor (hämtad 2025-05-06).

[5] Our world in data. ”Moore’s law: The number of transistors per microprocessor.” (2022), URL: https://ourworldindata.org/grapher/transistors-per-microprocessor (hämtad 2025-04-30).

[6] Red Hat. ”What is open source?” (2019), URL: https://www.redhat.com/en/topics/open-source/what-is-open-source (hämtad 2025-05-06).

[7] E. S. Raymond, *The Cathedral and the Bazaar*. O’Reilly Media, 1999.

[8] The Linux Foundation, ”Annual Report 2024 - Accelerating Industry Innovation,” 2024, Nedladdas via https://www.linuxfoundation.org/resources/publications/linux-foundation-annual-report-2024.

[9] J. Hennessy och D. Patterson, *Computer Architecture: A Quantitative Approach (Fifth ed.)* Morgan Kaufmann Publishers, 2012.

[10] IBM. ”The IBM System/360. The 5-billion-dollar gamble that changed the trajectory of IBM.” (u.å.), URL: https://www.ibm.com/history/system-360 (hämtad 2025-05-08).

[11] K. Asanović och D. A. Patterson, ”Instruction Sets Should Be Free: The Case For RISC-V,” University of California at Berkeley, tekn. rapport UCB/EECS-2014-146, 2014, Available at http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html.

[12] B. Tovar, C. Yee, R. Ng, H. Al-Asaad och S. Patil, ”2025 IEEE 15th Annual Computing and Communication Workshop and Conference, CCWC 2025,” i *Survey of RISC-V Pipelines and Testers*, Cited by: 0, 2025, s. 885–895. DOI: 10.1109/CCWC62904.2025.10903846. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-105001143762&doi=10.1109%2fCCWC62904.2025.10903846&partnerID=40&md5=c79a1ee3d56436d4e483fd363d6e7bd9.

[13] I. Kuon, R. Tessier och J. Rose, ”FPGA Architecture: Survey and Challenges,” The Edward S. Rogers Sr. Department of Electrical m. fl., 2007, Available at `https://www.eecg.toronto.edu/~jayar/pubs/kuon/foundtrend08.pdf`.

[14] D. A. Patterson och J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface (RISC-V Edition)*. Elsevier, 2017.

[15] W. J. Dally, R. C. Harting och T. M. Aamodt, *Digital Design Using VHDL - A Systems Approach*. Cambridge University Press, 2016.

[16] RISC-V International, *The RISC-V Instruction Set Manual, Volume I: Unprivileged Architecture*. 2024, Version 20240411.

[17] RISC-V International, *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture*. 2024, Version 20240411.

[18] SiFive, Inc., *SiFive Interrupt Cookbook*. 2020, Version 1.2.

[19] O. Kindgren m.fl. ”Datasheet.” (2020), URL: `https://serv.readthedocs.io/en/latest/datasheet.html` (hämtad 2025-02-05).

[20] O. Kindgren m.fl. ”Understanding FuseSoC.” (u.å.), URL: `https://fusesoc.readthedocs.io/en/stable/user/overview.html` (hämtad 2025-02-05).

[21] Veripool. ”Verilator.” (2024), URL: `https://www.veripool.org/verilator/` (hämtad 2025-02-03).

[22] AMD Xilinx. ”7 Series FPGAs Clocking Resources.” (2018), URL: `https://docs.amd.com/v/u/en-US/ug472_7Series_Clocking` (hämtad 2025-04-08).

A

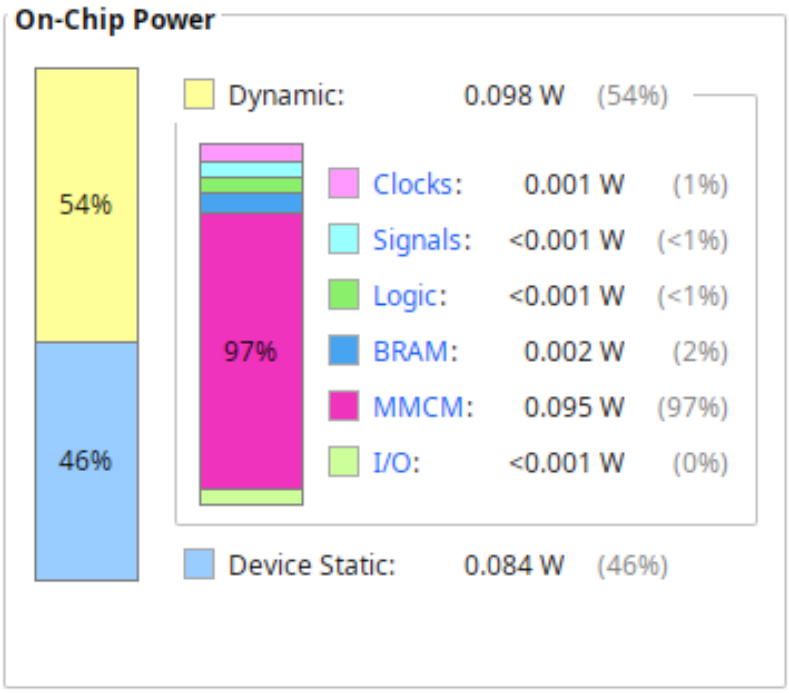
Appendix 1 - Mätresultat strömmätning Nexys A7

Frekvens [MHz]	8	16	32	64	128	256
Nexys A7 basförbrukning [mA]	120,7					
Klockgenerering [mA]	138,0	138,1	138,2	140,9	141,0	141,2
SERV sleep [mA]	140,7	141,1	141,3	145,5	148,6	153,0
SERV testloop [mA]	142,6	143,7	144,8	150,8	158,1	168,6
Klockgenerering endast externa avbrott [mA]	137,5	137,6	137,6	140,4	140,5	140,6
SERV testloop endast externa avbrott [mA]	139,6	140,3	141,4	145,6	150,1	155,7

Tabell A.1: SERV strömförbrukning på 5 V-matningsspänning till Nexys A7. Notera att alla mätvärden är rådata och att strömmarna inkluderar basströmmen hos Nexys A7.

B

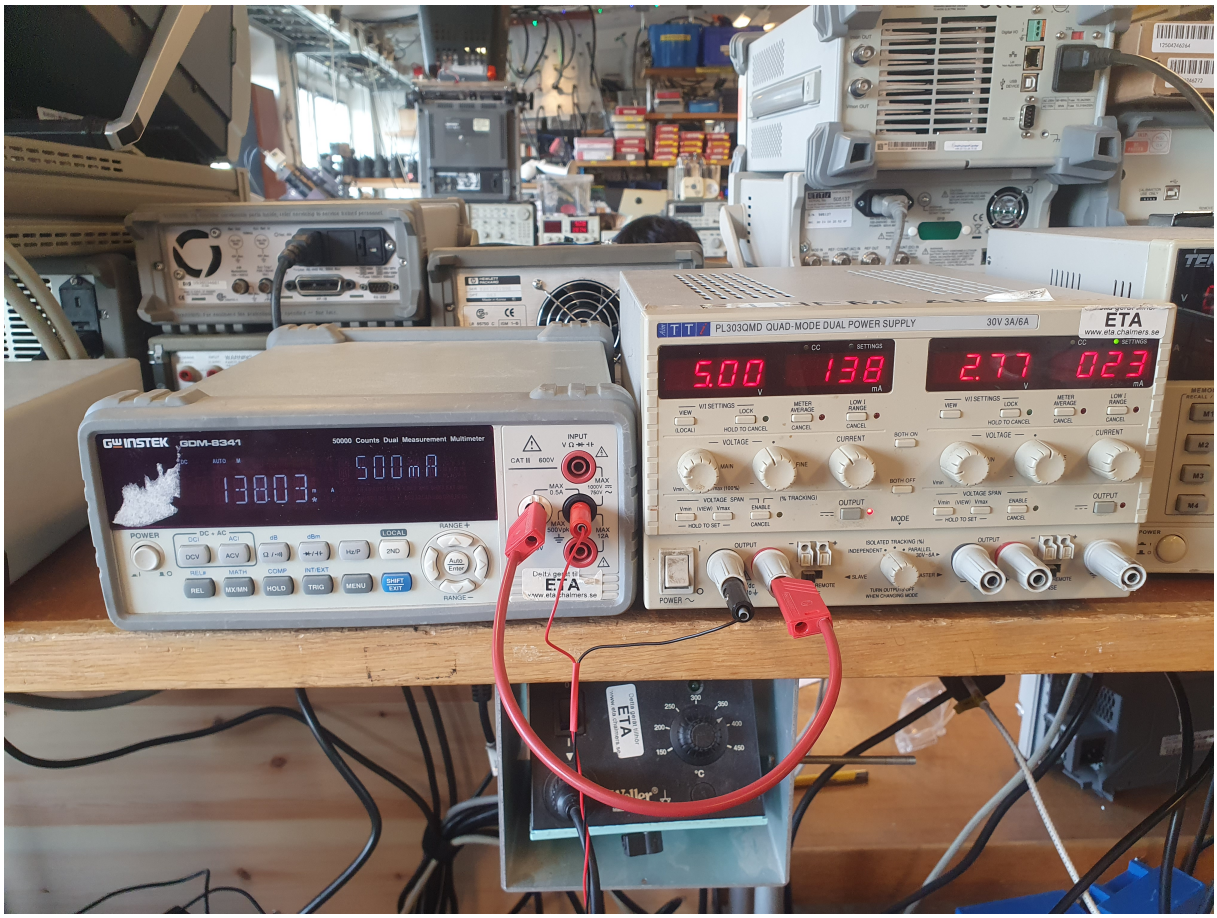
Appendix 2 - Resultat av effektrapportering i Vivado



Figur B.1: Servants effektförbrukning vid simulering med verktyget `Report Power` i Vivado.

C

Appendix 3 - Mätutrustning för strömmätning på Nexys A7



Figur C.1: Nätaggregat och amperemeter som används för att mäta strömförbrukningen på Nexys A7.

Institutionen för mikroteknologi och nanovetenskap
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS