FFIE/726/170

Godkjent
Kjeller 14 january 2000

Torleiv Maseng
Forskningssjef

ADDISJONSKRETS FOR IEEE–754 FLYTTALL OPTIMALISERT FOR GJENNOMSTRØMNING

GUNDERSEN Rune, BLOM Harald

FFI/RAPPORT–2000/00138

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25, 2027 Kjeller, Norge

FORSVARETS FORSKNINGSINSTITUTT (FFI)
Norwegian Defence Research Establishment
P O BOX 25
NO-2027 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT-2000/00138	UNCLASSIFIED	87
1a) PROJECT REFERENCE	2a) DECLASSIFICATION/DOWNGRADING SCHEDULE	
FFIE/726/170		

4) TITLE

ADDISJONSKRETS FOR IEEE-754 FLYTTALL
OPTIMALISERT FOR GJENNOMSTRØMNING

(IEEE-754 FLOATING-POINT ADDER OPTIMIZED FOR THROUGHPUT)

5) NAMES OF AUTHOR(S) IN FULL (surname first)

GUNDERSEN Rune, BLOM Harald

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited
(Offentlig tilgjengelig)

7) INDEXING TERMS IN ENGLISH:	IN NORWEGIAN:
a) Floating-point Addition	a) Flyttallsaddisjon
b) IEEE-754 Floating-point Numbers	b) IEEE-754 flyttall
c) Integrated Circuit Design	c) Kretskonstruksjon
d) High Level Synthesis	d) Høynivåsyntese
e)	e)

THESAURUS REFERENCE:

8) ABSTRACT

This report contains a description of a floating-point adder optimized for throughput rather than latecy. It is confirmed with IEEE-754 standard for binary floating-point arithmetic. The design is fully tested and verified. Synthesis results for the Alcatel Mietec MTC45000 technology are presented.

9) DATE	AUTHORIZED BY This page only	POSITION
14 January 2000	Torleiv Maseng	Director of Research

ISBN 82-464-0396-6

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

INNHOLD

		Side
1	INTRODUKSJON	5
2	BAKGRUNN	6
3	TALLFORMAT	6
3.1	IEEE-754	7
3.2	Våre avvik fra standarden	10
3.3	Diskusjon om behovet for stort dynamikkområde	11
4	ALGORITME FOR ADDISJON AV FLYTTALL	12
4.1	Algoritme for flyttallsaddisjon	13
4.2	Optimaliseringer	15
4.3	Pipelineing av addisjonskretsen	16
5	VERKTØY OG METODE	18
5.1	VHSIC Hardware Description Language (VHDL)	18
5.2	Simulering	19
5.3	Syntese	20
6	SYSTEMBESKRIVELSE FOR ADDISJONSKRETSEN	22
6.1	Kretsens egenskaper	22
6.2	Pinnebeskrivelse	23
6.3	Strukturell beskrivelse	24
7	TESTING OG VERIFIKASJON	26
7.1	Teststrategi	26
7.2	Test- og verifikasjonstimuli	27
7.3	Datagenerator med inngangsparametre	28
7.4	Utførte tester	30
8	SYNTESERESULTATER	31
8.1	Parametre som beskriver egenskapene til kretsens omgivelser	31
8.2	Parametre som beskriver last modeller av nettverket	31
8.3	Parametre som beskriver systemets grensesnitt	32
8.4	Spesifisering av kretsens omgivelsesparametre	32
8.5	Synteseresultater	33
8.6	Synteseresultater for addisjonskreten	34
9	KODEBESKRIVELSE	35
9.1	Egendefinerte VHDL-biblioteker	35

9.2	Strukturell oversikt over VHDL-koden	36
9.3	Spesielle håndgrep for syntese.	37
9.4	Addisjonsenhetens inn- og utgangssignaler	37
9.5	Detaljert gjennomgang av VHDL-koden	38
10	KONKLUSJON	40
	Litteratur	41

APPENDIKS

A	VHDL-KODE FOR FLYTTALLSADDISJONSKRETSEN .	43
A.1	VHDL-filen fpadd.vhd	43
A.2	VHDL-kode for egendefinert bibliotek: types.vhd	53
B	VHDL-KODE FOR TESTBENKEN OG TESTBENKINTERFACE	59
B.1	VHDL-kode for testbenkens C-interface	59
B.2	VHDL-kode for testbenken	59
C	KODE FOR OPPSETT AV SYNOPSYS DESIGN COMPILER	74
C.1	Kode for oppsett av Synopsys: .synopsys_dc.setup.	74
C.2	Kode for oppsett av synopsys: .synopsys_vss.setup.	75
D	C-KODE TIL FASITGENERATOREN	75
D.1	float_add.C	75
D.2	sim_fp.C	77
D.3	Filen basic.h	82
E	PARAMETERSETT FOR TESTBENK	85
	Fordelingsliste	87

ADDISJONSKRETS FOR IEEE-754 FLYTTALL OPTIMALISERT FOR GJENNOMSTRØMNING

1 INTRODUKSJON

For tiden utvikles et større system for å kjøre sanntids Fourier-analyse på data hentet fra Forsvarets måleradar. I systemet inngår egenutviklede brikker for Fourier-transformasjon av signalene. Addisjonskretsen beskrevet i denne rapporten inngår som en viktig byggestein i FFT-prosessoren.

Addisjonskretsen er konstruert ved hjelp av VHSIC Hardware Description Language (VHDL) og høynivåsyntese. Beskrivelsen av kretsen i VHDL er teknologiuavhengig. Den kan enkelt implementeres i ulike teknologier som f eks Field Programmable Gate Array (FPGA) eller CMOS (12)(14). Dette gjør kretsen egnet som byggeblokk for andre regnekrevende applikasjoner hvor høy presisjon er av betydning.

For å gi en mer helhetlig forståelse av arbeidet, inneholder kapittel 2 en del bakgrunnsinformasjon. Her beskrives motivasjonen for at kretsen ble konstruert og hvilke føringer det omliggende systemet gir for kretskonstruksjonen.

Tallformatet som benyttes i kretsen følger IEEE-754 standard floating-point for binary arithmetic (10) med tre unntak. For det første er tallets mantisse trunkert til 15 bit. For det andre benyttes ikke denormaliserte tall. Dessuten er behandlingen av Not a Number (NaN) noe forenklet. Kapittel 3 inneholder en detaljert beskrivelse av tallformatet.

I kapittel 4 beskrives en algoritme for addisjon av flyttall i noen detalj.

Valg av verktøy og metode samt våre erfaringer med disse omtales i kapittel 5.

Kapittel 6 er en systembeskrivelse for addisjonskretsen. Det inneholder en beskrivelse av kretsens egenskaper, grensesnitt til omverdenen og indre struktur.

I kapittel 7 beskrives testing og verifikasjons av addisjonskretsen. Vi beskriver hvilke valg og avveininger som er gjort i forhold til teststrategi og testmønstre.

I kapittel 8 beskriver vi attributtene som brukes for å styre syntesen og syntheseresultatene.

I kapittel 9 i beskrives VHDL-koden. Noen av håndgrepene som er utført for å holde syntheseverktøyet i tøylene kommenteres.

Appendiksene inneholder VHDL-kode for addisjonskretsen, de egendefinerte typebibliotekene og testbenken, C-kode for fasitgeneratoren, oppsettet for Synopsys Design Compiler og parametersett for de ulike testene.

Kretsen inneholder 3293 portekvivalenter, og er simulert på 66 MHz hastighet. Den er omfattende testet og verifisert. Kretsen er også kjørt i omfattende systemsimuleringer hvor den har fungert feilfritt.

2 BAKGRUNN

FFT-kretsen er en vesentlig del av et eksperimentelt system for utvikling av nye metoder og algoritmer for radar og eventuelt andre applikasjoner. Dermed er ikke alle krav som stilles til systemet gitt på forhånd. Det er derfor viktig at systemet ikke lages med egenskaper som sterkt kan begrense muligheten for hva man kan implementere. Viktige parametere her er vektorlengder, minne størrelse og tallformat. Samtidig må man sette visse grenser for disse parameterne for å få et system og en krets som er fysisk realiserbar.

For å oppnå en gitt rekkevidde må vektorlengden i korrelasjonen med utsendt signal tilpasses oppløsningen man har. Med en båndbredde på f eks 500 MHz, har man en avstandsoppløsning på 30 cm. For å få en rekkevidde på f eks 20 km må vektorlengden være 64k. Dette setter krav til at systemet må være i stand til å kjøre med store vektorlengder.

Inputsystemet, vi i første omgang skal benytte, har en A/D-omformer med 10 bit oppløsning. En input med bedre oppløsning må ikke utelukkes.

Et annet moment er at ikke alle beregninger nødvendigvis blir utført av FFT-prosessoren. Det er derfor gunstig å benytte et tallformat som er likt det som brukes i standard regnemaskiner og som følger gitte standarder eller et format som enkelt kan konverteres til/fra slike.

For å håndtere FFT med så store vektorlengder, og flere etterfølgende slike, anser vi det for nødvendig å bruke et skalerbart tallformat. Enkelte andre realisasjoner av FFT brikker har brukt flyttall med felles eksponent for grupper av tall. Det ekstreme er å ha felles eksponent for alle delresultater etter en butterfly. Dette er imidlertid ikke mulig i en pipelinearkitektur fordi man begynner å regne i et trinn (butterfly) før man har alle resultatene fra tidligere trinn. Dermed vet man ikke hvilken felles eksponent disse skulle hatt.

Det eneste som kunne være felles i disse variantene er felles eksponent for et komplekst tall. Dette er imidlertid heller ikke valgt fordi besparelsen ville vært liten. For å få en "ren og enkel" implementasjon har vi derfor valgt flyttallrepresentasjon.

Vi har valgt et 24-bits format som er likt IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754), med enkel presisjon. Dette gir mulighet for et stort dynamikkområde samtidig som det har relativt bra oppløsning.

FFT-prosessoren er nærmere beskrevet i en egen rapport (4).

3 TALLFORMAT

I dette kapitlet beskrives tallformatet vi benytter i addisjonskretsen. Med tre unntak følger vi IEEE-754 (10) (8) (6).

Standaren definerer hvordan flyttall representeres binært i digitale systemer og hvordan aritmetiske operasjoner utføres på slike tall. Den sikrer at data kan flyttes mellom forskjel-

lige datamaskiner og at resultatene av like aritmetiske operasjoner blir de samme. Standaren er utbredt og brukes av de fleste maskinprodusenter. Standaren kan implementeres i maskinvare, i programvare eller i en kombinasjon av disse.

3.1 IEEE-754

IEEE-754 definerer fire flyttalls formater: enkel presisjon, enkel presisjon utvidet, dobbel presisjon og dobbel presisjon utvidet.

Et flyttall (F) er bygd opp som vist i ligningen:

$$F = (-1)^S \times M \times 2^{E+B} \tag{3.1}$$

hvor S er fortegnet, M er mantissen, E er eksponenten og B er en forskyvning (bias).

Mantissen som vanligvis er normalisert (dvs $1.0 \leq M < 2.0$) kan skrives som M = 1.f hvor f (fraction) er brøken som representerer tallets presisjon. Som vi ser er den mest signifikante biten (MSB) i mantissen alltid 1. Den utelates derfor fra tallrepresentasjonen. Vi sier at mantissen har en skjult ledende bit.

Biasen, som er et konstant tillegg, plasserer tallet 1.0 midt i representasjonsområdet, slik at omlag like store områder avsettes for negative og positive eksponenter. Eksponenten har mao ikke noe fortegn. Vi lar E_{max} og E_{min} betegne henholdsvis den største og den minste mulige eksponenten som kan representeres.

Tabell 3.1. viser verdiene for de ulike parametre for de fire formatene.

Presisjon	Enkel	Enkel utvidet	Dobbel	Dobbel utvidet
Antall bit totalt	32	≥ 43	64	≥ 79
Antall bit i f	23	≥ 31	52	≥ 63
Antall bit i E	8	≥ 11	11	≥ 15
E_{max}	127	≥ 1023	1023	≥ 16383
E_{min}	-126	≤ -1022	-1022	≤ -16382
Eksponent bias	127	Uspesifisert	1023	Uspesifisert

Tabell 3.1 Verdier for en del parametre i IEEE-754.

Da vi har et begrenset antall bit i mantissen og eksponenten, er det åpenbart at ikke alle tall kan representeres. Både oppløsningen, ϵ, og tallområdet som kan representeres er begrenset. Standarden inneholder derfor detaljerte regler for hvordan resultatet av en aritmetisk operasjon skal avrundes til et tall som kan representeres, foruten mekanismer for å håndtere overflow og underflow.

For å kunne håndtere unntakstilfeller på en god måte definerer standarden en rekke spesialverdier. Hvilke verdier dette er og kodingen av dem er vist i tabell 3.2. Som vi ser av tabellen, opererer standarden med to versjoner av tallet null: +0 og -0. Ved sammenligning er disse, per definisjon, like. Videre, inneholder standarden både $+\infty$ og $-\infty$.

Ved å tillate denormaliserte tall utvides tallområdet så små resultater avhjelpes ytterligere. Denormaliserte tall er ikke normaliserte, dvs, vi har ingen implisitt ledende ener. Fortolkningen av et denormalisert flyttall med enkel presisjon blir altså:

$$F = (-1)^S \times 0.f \times 2^{-126} \qquad (3.2)$$

Tilfelle	Signifikand	Eksponent	Brøk	Flytallsverdi
Null	S { 0,1 }	$E = E_{min} - 1$	$f = 0$	$(-1)^S 0.0$
Denormalisert tall	S { 0,1 }	$E = E_{min} - 1$	$f \neq 0$	$(-1)^S 2^{Emin}(0.f)$
Uendelig	S { 0,1 }	$E = E_{max} + 1$	$f = 0$	$(-1)^S \infty$
Not a Number	S { 0,1 }	$E = E_{max} + 1$	$f \neq 0$	NaN

Tabell 3.2 Representasjon av spesialverdier i IEEE-754

En del operasjoner, som f eks $0 \times \infty$, har ikke noe entydig resultat. IEEE-754 håndterer dette ved å introdusere spesialverdier, kalt NaN (Not a Number), som skal være resultatet av slike ulovlige operasjoner. Standarden definerer to typer NaN, signalerende NaN og stille NaN, men definerer ikke hvordan man skiller dem fra hverandre.

IEEE-754 definerer en rekke ulovlige operasjoner som skal gi NaN som resultat. Disse operasjonene er listet i tabell 3.3.

Operasjon	Tilfeller som gir NaN
+	$\infty + (-\infty)$
-	$-\infty - (-\infty)$
$\times$	$0 \times \infty$
/	$0 / 0, \infty / \infty$
REM	x REM 0, ∞ REM y
$\sqrt{}$	$\sqrt{}$x (hvis x < 0)

Tabell 3.3 Operasjoner som gir NaN som resultat

IEEE-754 definerer fem klasser av unntakstilfeller som skal signaleres enten ved at et korresponderende flagg settes eller ved at kjøringen avbrytes. De fem klassene er:

Overflow

Signaleres hver gang et resultat er større enn det største tallet som kan representeres i tallformatet. Resultatet av operasjonen er $\pm \infty$ eller $\pm$ det største endelige tallet tallformatet kan representere, avhengig av hvilken avrundingsmodus som er valgt, som vist i tabell 3.4.

Underflow

Signaleres hver gang et resultat, forskjellig fra 0, ligger mellom $\pm 2^{E_{min}}$. Resultatet av operasjonen er enten 0, $\pm 2^{E_{min}}$ eller et denormalisert tall.

Divisjon med null

Signaleres hver gang divisor er 0 og dividenden er et endelig tall forskjellig fra 0. Resultatet av operasjonen er ∞ med korrekt fortegn.

Ulovlig operasjon

Signaleres hvis en operand er ulovlig for operasjonen som skal utføres. Resultatet av operasjonen er NaN.

Unøyaktig resultat

Signaleres hver gang et avrundet resultat ikke er eksakt. Resultatet av operasjonen er et avrundet resultat.

I utgangspunktet skal slike unntak medføre at et eller (i noen tilfeller flere) flagg settes uten at kjøringen avbrytes. Da dette er den tilnærmingen vi har valgt, utelates diskusjonen om hvordan avbrudd skal behandles.

Et endelig antall bit i mantissen gjør det umulig å representere alle mulige tall. De tallene som ikke kan representeres eksakt må avrundes. Ved avrunding søker man å erstatte et tall med et tall som kan representeres i tallformatet på en slik måte at feilen blir minst mulig.

Før en aritmetisk operasjon utføres, innrettes de to mantissene slik at de får samme vekt ved at mantissen til operanden med minst tallverdi forskyves mot høyre. De minst signifikante bitene i mantissen vil forsvinne ut. Tre bit, kalt guard (g), round (r) og sticky (s), holder på tilstrekkelig og nødvendig informasjon til å kunne gjøre korrekt avrunding av resultatet av operasjonen. I guard lagres biten rett til høyre for p_0 (LSB i mantissen). Round lagrer den neste biten til høyre. (Guard og round holder altså de to siste bitene som ble skjøvet ut.) De resterende bitene OR-es sammen. Resultatet lagres i sticky.

IEEE-754 spesifiserer fire forskjellige avrundings modi, nemlig:

Avrunding mot nærmeste like tall

Resultatet skal avrundes til den verdien som er nærmest det nøyaktige svaret. Hvis de to nærmeste verdiene er like nære, skal resultatet som har null i LSB benyttes.

Avrunding mot minus uendelig

Resultatet avrundes til den nærmeste verdien som er mindre enn resultatet.

Avrunding mot pluss uendelig

Resultatet avrundes til den nærmeste verdien som er større enn resultatet.

Avrunding mot null

Resultatet avrundes til den verdien som har absolutt verdi med minst avvik, men som ikke er større enn resultatet.

Kriteriene for avrunding i de fire forskjellige modiene er som vist i figur 3.4.

Avrundings-modus	Resultat ≥ 0	Resultat < 0	Resultat ved positiv overflow	Resultat ved negativ overflow
$-\infty$		+1 hvis $r \vee s$	formatets største endelige tall	$-\infty$
$+\infty$	+1 hvis $r \vee s$		∞	-(formatets største endelige tall)
0			formatets største endelige tall	-(formatets største endelige tall)
Nærmeste like tall	+1 hvis $r \wedge p_0$ eller $r \wedge s$	+1 hvis $r \wedge p_0$ eller $r \wedge s$	∞	$-\infty$

Tabell 3.4 Resultat ved avrunding eller overflow i de forskjellige avrundingsmodi. P_0 er LSB av mantisen, r er round- og s er sticky-biten.

3.2 Våre avvik fra standarden

I en sen fase av konstruksjonsarbeidet ble det funnet at vi hadde betydelige plass-problemer på FFT-brikken. Da addisjonskretsen var den submodulen som gikk igjen flest ganger, var det naturlig å forsøke å redusere størrelsen på denne. Konstruksjonen ble gjennomgått og vi fant å kunne gjøre følgende "forenklinger":

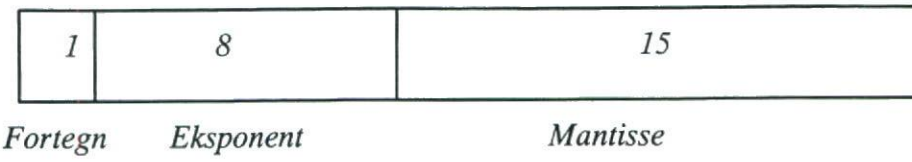
Den utvidelsen av tallområdet denormaliserte tall representeres ble funnet å resultere i uforholdsmessig mye logikk. Derfor supporterer addisjonskretsen ikke denormaliserte tall. Hvis en av operandene er et denormalisert tall settes denne lik null før addisjonen utføres.

Behandlingen av NaN er forenklet. Hvis en av operandene til addisjonskretsen er NaN, slippes denne uforandret igjennom. Hvis resultatet av addisjonen skal være en NaN, setter kretsen ut en NaN som på en HP PA-RISC maskin er en stille NaN. Dette gjør at feil som oppstår i FFT-prosessoren hverken påvirker eller belaster vertsmaskinen.

Valget av tallformat er basert på kravene til totalsystemets dynamikkområde. Basert på systemkravene, ble det funnet at et tallformat med redusert lengde på mantissen ga tilstrekkelig oppløsning. Et slikt tallformat er også svært enkelt å konvertere til IEEE-754 flyttall.

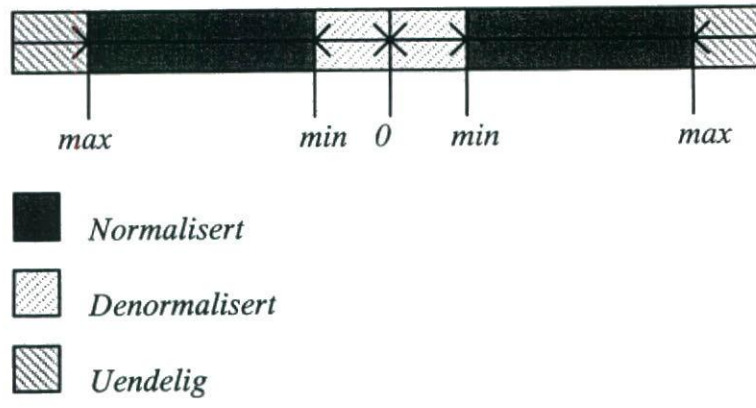
Addisjonskretsen benytter mao et tallformat på 24 bit med en bit fortegn, 8 bit eksponent og 15 bit mantisse, som vist i figur 3.1, med følgende oppbygging:

$$F = (-1)^S \times 0.f \times 2^{-127} \tag{3.3}$$



Figur 3.1 Vårt 24 bits tallformat med 1 bit fortegn , 8 bit eksponent og 15 bit mantisse.

Som vist i figur 3.2, går dynamikkområdet til dette tallformatet fra (desimalt) -3.4028×10^{38} til -1.17549×10^{-39} og fra 1.17549×10^{-39} til 3.4028×10^{38}. Tabell 3.5 viser de eksakte grenseverdiene på heksadesimalt format.



Figur 3.2 Dynamikkområdet til vårt 24 bits tallformat.

Tilfelle		Eksponent	Mantisse	Desimalt
Null		00	8000	Null
Denorm		00	$\neq 0$	Null < denorm < $\text{Normaliset}_{\text{min}}$
Normalisert	min	01	8000	1.175494×10^{-38}
	max	FE	FFFF	3.402823×10^{38}
Uendelig		FF	8000	$\infty > \text{Normalisert}_{\text{max}}$
NaN		FF	$\neq 0$	NaN > ∞

Tabell 3.5 Dynamikkområdet til vårt 24 bit tallformat, eksponent og mantisse er vist på heksadesimalt format.

3.3 Diskusjon om behovet for stort dynamikkområde

Da det har betydning for valg av tallformat, inkluderer vi noen betraktninger om FFT-beregninger på lange vektorer.

La $u(x)$ være en stykkevis glatt, 2π-periodisk funksjon som er $m-1$ ganger kontinuerlig deriverbar på $[0, 2\pi]$. Disse antakelsene setter klare begrensninger på funksjonen $u(x)$ sin regularitet. $u(x)$ har da et endelig antall singulære punkter, diskontinuiteter eller knekkpunkter på $[0, 2\pi]$. Dette betyr at $u(x)$ kan på $[0, 2\pi]$ deles opp i et endelig antall M underintervaller, (γ_j, γ_{j+1}), $j = 0, 1, 2, ..., M-1$, hvor de ensidige grensene fra det indre av hvert underintervall vil eksistere.

De diskrete Fourier-koeffisientene, $\tilde{u}_k$, til funksjonen $u(x)$ er da definert ved

$$\tilde{u}_k \stackrel{def}{=} \frac{1}{N}\sum_{j=0}^{N-1} u(x_j)e^{-ikx_j}, \text{ for } k = -\frac{N}{2}, -\frac{N}{2}+1, ..., \frac{N}{2}-1,$$

hvor de N diskrete funksjonsverdiene $u(x_j)$, $j = 0, 1, 2, ..., N-1$, er gitt i punktene $x_j = \frac{2\pi j}{N}$.

Under disse forutsetningene kan det vises at de diskrete Fourier-koeffisientene tilfredsstiller for det positive heltallet P

$$\tilde{u}_k = O(N^{-(m+1)}) \text{ for } \frac{N}{2} - P \leq |k| < \frac{N}{2} \text{ når } N \to \infty.$$

Likningen over forteller oss at jo glattere funksjonen $u(x)$ er, jo raskere vil de øverste Fourier-koeffisientene avta ettersom $N \to \infty$. De laveste derimot, trenger *ikke* oppføre seg på denne måten. Den diskrete Fourier-koeffisienten $\tilde{u}_0$ uttrykker middelverdien til funksjonen $u(x)$, en verdi som generelt vil være $O(1)$, også når $N \to \infty$.

Dette betyr at i lange FFT'er, vil det være behov for et stort dynamisk område. Et lite dynamisk område vil medføre muligheter for store feil i de øverste diskrete Fourier-koeffisientene. Disse koeffisientene bærer viktig informasjon om signalets diskontinuiteter i alle deriverte. Radarkoder som inneholder diskontinuiteter kan brukes i bredbåndete radarer. Forsvarets måleradar benytter blant annet denne type koder.

Med et lite dynamisk område kan den absolutte avrundingsfeil i de øverste Fourier-koeffisientene riktignok bli liten i forhold til verdien på de laveste Fourier-koeffisientene. Men den relative kan være svært stor, ikke utenkelig mange størrelsesordener. Manglende dynamikkområde kan altså sees på som et unønsket lavpassfilter som avhenger av data som filtreres, uten at man har kontroll over filtreringen.

I et FFT-system som skal beregne lange FFT'er er det derfor nødvendig med et godt dynamikkområde.

4 ALGORITME FOR ADDISJON AV FLYTTALL

I det følgende beskrives en algoritme for addisjon av flyttall i noen detalj. En grundigere gjennomgang av temaet finnes i (3), (5) og (9).

Addisjonskretsen tar som input to flyttall med n bit presisjon og produserer et n-bits resultat. Resultatet av operasjonen er avrundet da vi har like mange bit i resultatet som i operan-

dene. Da en addisjon av to tall med forskjellig fortegn faktisk er en subtraksjon, er det å utføre addisjonen en relativt kompleks operasjon. Det følger som en konsekvens av dette at addisjonskretsen også kan utføre subtraksjon.

4.1 Algoritme for flyttallsaddisjon

Den implementerte kretsen er basert på algoritmen som vi presenterer i dette avsnittet med unntak av enkelte optimaliseringer. Vi lar a og b være to flyttall som skal adderes. Videre, lar vi s_n være fortegnet, e_n eksponenten og m_n mantissen til et slikt tall. Vi benytter dessuten betegnelsen g for guard, r for round og s for sticky. Da kan flyttallsaddisjon av de to operandene utføres i følgende steg:

Sjekk om operandene har gyldige verdier

Operandene kan ha verdier eller kombinasjoner av verdier som må behandles særskilt. Hvis et forsøk på en ulovlig operasjon detekteres, settes resultatet til NaN og forsøket flagges vha. "ulovlig operasjon" flagget. De ulovlige operasjonene er $(+\infty)+(-\infty)$, $(-\infty)+(+\infty)$, $(+\infty)-(+\infty)$ og $(-\infty)-(-\infty)$. Hvis en av operandene har uendelig tallverdi, settes denne verdien som resultat forutsatt lovlig operasjon. Er en av operandene en NaN, settes resultatet til samme NaN. Denormaliserte tall settes til 0.

Sjekk om operandene skal bytte plass

Hvis $e_1 < e_2$ lar vi operandene bytte plass. Hvis $e_1 \leq e_2$ og $m_1 < m_2$ bytter vi også om på operandene. Hvis $e_1 > e_2$ og $m_1 > m_2$ byttes ikke operandene. På denne måten sikrer vi at differansen mellom eksponentene overholder $d = e_1 - e_2 \geq 0$ så addisjonen eller subtraksjonen ikke vil gi fortegnsskifte.

Finn ut om m_2 skal toerkomplementeres

Hvis fortegnet $s_1 \neq s_2$, erstattes m_2 med sin toerkomplement. Dette for å slippe å utføre operasjonen subtraksjon (som medfører maskinvare for borrow-overføring mm).

Still opp mantissene på linje

M_2 justeres $d = e_1 - e_2$ plasser mot høyre slik at biter som adderes har samme vekt. Det justerte tallet fortegnsutvides. Vi fyller inn 0'er hvis $s_1 = s_2$. Hvis $s_1 \neq s_2$, fyller vi inn 1'ere da m_2 er toerkomplementert i dette tilfellet. Under en eventuell høyrejusteringen av m_2 setter *vi guard* lik den mest signifikante av de utjusterte bitene, *round* lik den nest mest signifikante av de utjusterte bitene og *sticky* lik en logisk *OR* av resten av de utjusterte bitene.

Utfør heltallsaddisjon av mantissene m_1 og m_2

Summen $S = m_1 + m_2$ legges i et $n + 1$ biters register .

Normalisering av resultatet

Hvis de to mest signifikante bitene i S begge er 1, justeres S en posisjon til høyre og eksponenten e_1 økes med en. Hvis de to bitene er lik 01 forblir S uforandret. Hvis begge bitene er

0, søker vi igjennom S til vi finner ett bit satt til 1. Dette bitets posisjon kaller vi x. Deretter justerer vi S x plasser til venstre og subtraherer x fra e_1. Ved den første venstrejusteringen settes verdien av *guard* inn i den minst signifikante posisjonen i S. Deretter settes det inn 0'er. Hvis vi ikke finner noen 1'ere i S, er resultatet et denormalisert tall. I dette tilfellet settes resultatet lik 0.

Justering av round og sticky

Hvis S under normaliseringen ble justert til høyre så setters $s := g\ OR\ r\ OR\ s$. Round settes lik det minst signifikante biten i S. Hvis S ikke ble justert settes $s := r\ OR\ s$ og $r := g$. Hvis vi hadde en venstre justering, forblir s og r uforandret. Ved to eller flere venstre justeringer av S, settes s og r til null.

Avrund det normaliserte resultatet

Hvis $r = 1$ og $s = 1$ eller det minst signifikante biten i S er 1 så setter vi $S := S + 1$. Hvis nødvendig må eksponenten, e_1, justeres.

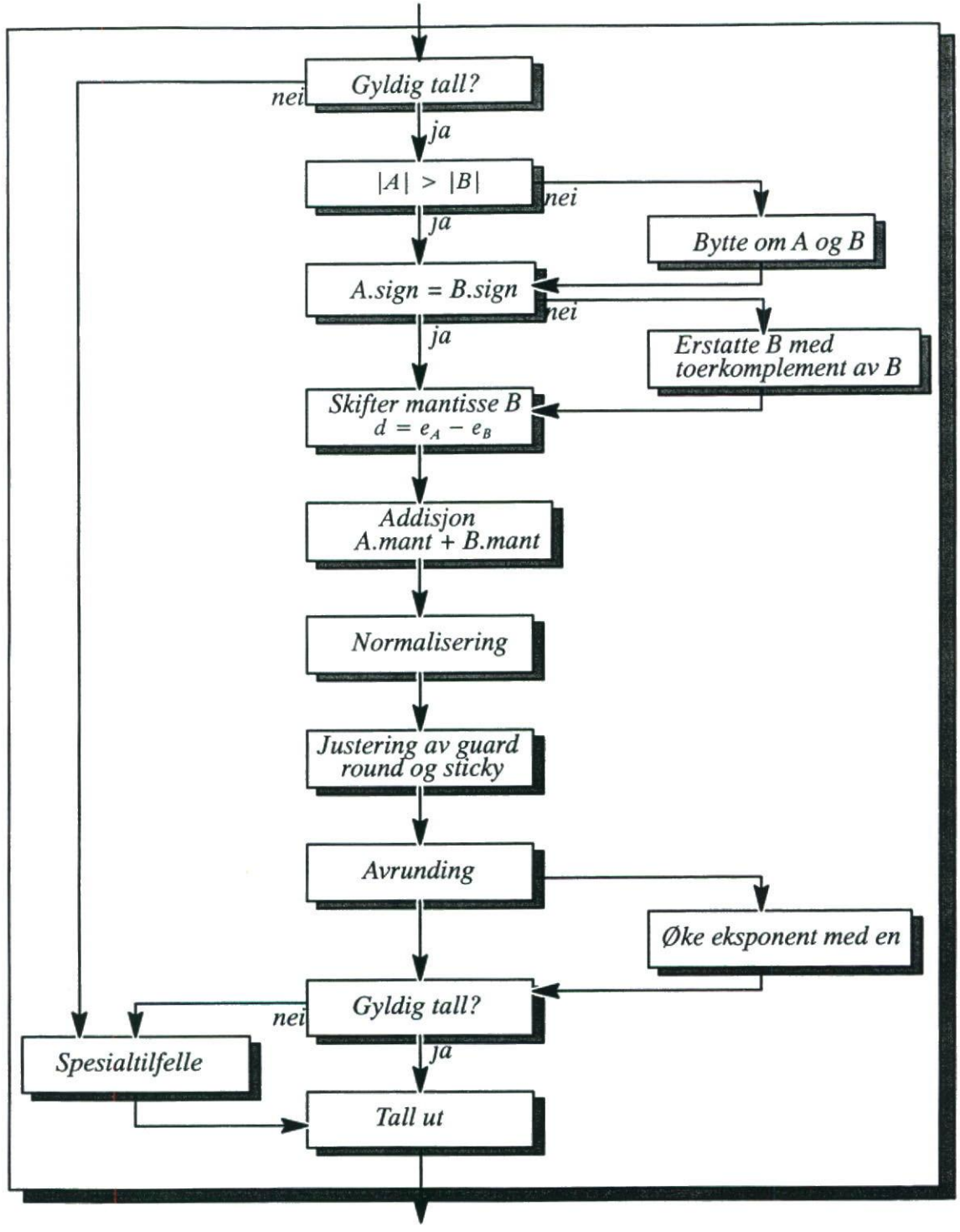
Bestem fortegnet til resultatet

Fortegnet er gitt av det største tallet og er lik s_1.

Sjekk at resultatet er et gyldig tall

Til slutt må vi sjekke at vi har et gyldig resultat. Hvis ikke må vi sette ut spesialverdier og flagge disse i henhold til standarden.

En grafisk fremstilling av algoritmen for flyttallsaddisjon er vist i figur 4.1.



Figur 4.1 Flytskjema for flyttallsaddisjon.

4.2 Optimaliseringer

Det er publisert mange håndgrep som kan brukes for å utføre addisjonen raskere. Vi ønsker å presisere at designkriteriene må være utgangspunktet for eventuelle optimaliseringer. Som tidligere nevnt inngår addisjonskretsen i et system for FFT-beregning på lange strømmer av data. De fleste av disse håndgrepene er derfor lite relevante i vårt tilfelle.

I motsetning til mange publiserte addisjonskretser har vi ikke strenge krav til tidsbruken for å utføre en addisjon (latency). Det viktige for oss er gjennomstrømmingen av data (throughput). Vi ønsker å levere et resultat for hvert klokketikk. Om det tar noen tikk før resultatene begynner å komme, er dette uproblematisk.

Dette gir naturlig en pipelinet arkitektur for addisjonskretsen. Hvor mange pipeline-trinn kretsen skal ha, er allikevel en avveining. Hvor mange porter som går med til å lage et ekstra pipeline-trinn må veies mot antall porter forbundet med å lage to parallelle løp. Ved å duplisere noen av modulene og kjøre en del av beregningene i parallell, kan vi få et kortere kritisk spor og kanskje spare et pipeline-trinn.

Etter en gjennomgang av addisjonskretsen fant vi at følgende optimaliseringer er gunstige:

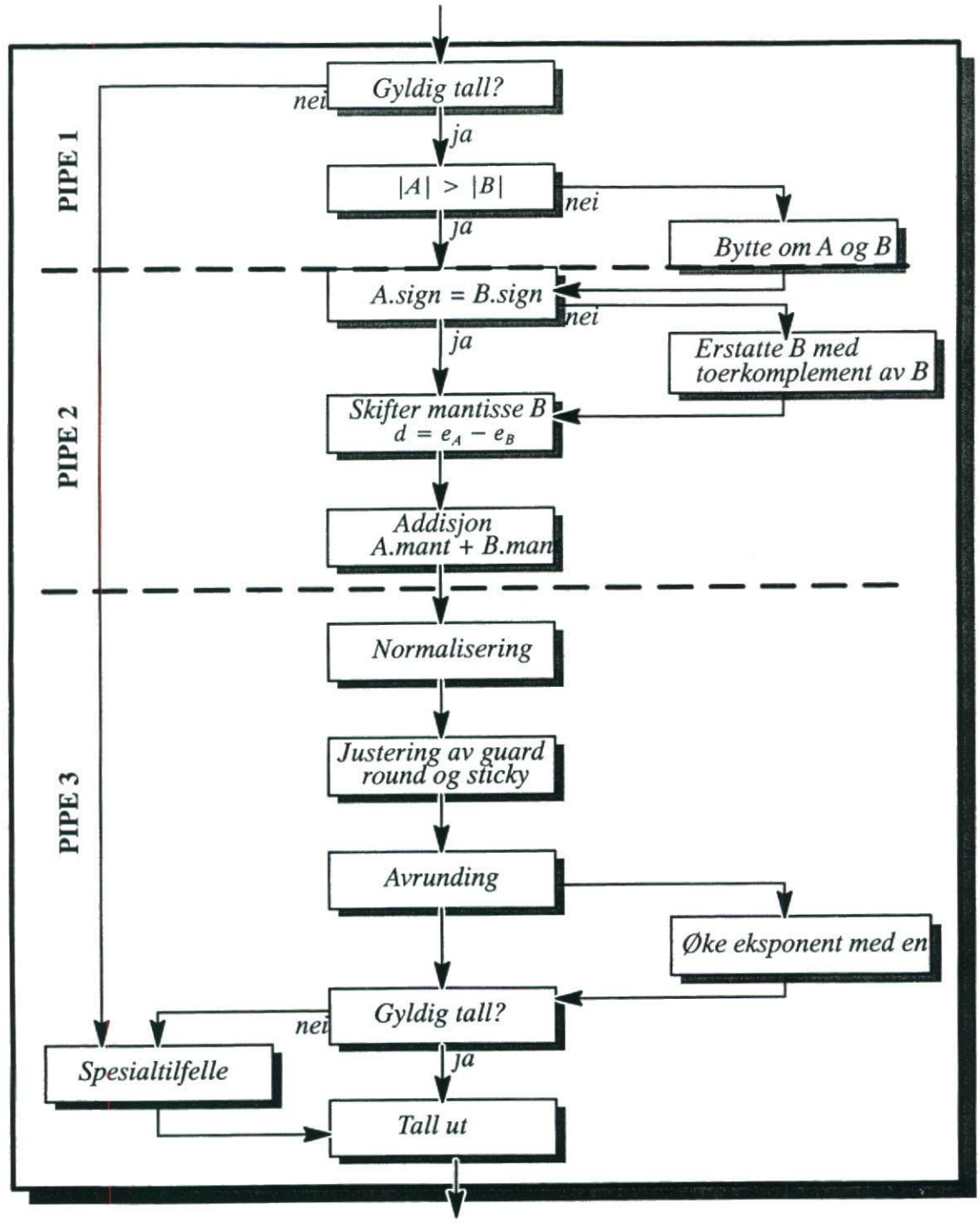
Sjekken på om operandene (*a* og *b*) er gyldige kan gjøres i parallell.

Ved beregning av differansen mellom eksponentene e_A og e_B kan vi bruke to addisjonskretser; en for å beregne $diff_{a-b} = e_A - e_B$ og en for å beregne $diff_{b-a} = e_B - e_A$. Ved å gjøre disse beregningene parallelt med sammenligningen av A og B, for å finne ut hvilket tall som er størst, tas beregningene ut av det kritiske sporet.

Vi kan sammenligne absoluttverdien av operandene og bytte om på dem hvis nødvendig. Da slipper vi å toerkomplimentere et eventuelt negativt resultat. Vi vet at svaret vil ha samme fortegn som det største talles fortegn. Denne eventuelle toerkomplementeringen ville ha kommet i kritisk spor etter selve addisjonen av mantissene og ført til en tregere krets.

4.3 Pipelineing av addisjonskretsen

Det ble etterhvert klart at den automatiske pipelineings opsjonen i Synopsys sin Design Compiler ikke greide å handtere så komplekse design som denne flyttalsaddisjonskretsen. Vi måtte derfor legge inn alle registere og pipeline-trinn for hand, noe som medførte en del forarbeide før syntese. Med utgangspunkt i en del synteseresultater fra addisjonskretsen som ikke var pipelinet, viste det seg mulig å greie systemkravene med tre pipelinetrinn. Første trinn ble lagt mellom ombyttingsenheten og operasjonssjekken. Andre trinn ble plassert mellom addisjonen og normaliseringsenheten. En grafisk fremstilling av addisjonskretsen med tre pipelinetrinn er vist i figur 4.2.



Figur 4.2 Flytskjema for flyttallsaddisjonen med visualisering av de tre innlagte pipeline stegene.

5 VERKTØY OG METODE

Kompleksiteten til digitale elektroniske systemer har vist seg å øke tilnærmet eksponentielt med tiden. Dette faktum, kombinert med kortere produktlevetid og økte krav om pålitelighet, har tvunget konstruktørene til å øke sin produktivitet og heve kvaliteten på sitt arbeide.

Innen programvare har man hatt en overgang fra lavnivå til høynivå programmeringspråk. Tilsvarende teknikker, som her blir benyttet for å håndtere kompleksitet og feildeteksjon, kan anvendes innen utvikling av maskinvare. Ved å heve abstraksjonsnivået kan konstruktøren se bort fra irrelevante detaljer og konsentrere seg om systemets oppførsel.

Syntese av maskinvare er en metode for å oppnå en slik hevning av abstraksjonsnivået. Et syntesepogram tar en beskrivelse av maskinvaren (på et høyere abstraksjonsnivå) og oversetter den til en beskrivelse på et lavere abstraksjonsnivå. Det er utviklet egne språk for beskrivelse av maskinvare. Input til syntesen vil typisk være en kretsbeskrivelse i et slikt språk. Output fra syntesen kan være en beskrivelse i et slikt språk, et kretsskjema el.

Vårt valg av synteseverktøy fulgte automatisk av valg av prosesshus da dette kun støttet bruk av Synopsys Design Compiler. Vi ønsker likevel å framsette en del betraktninger rundt valg av verktøy og metode og våre erfaringer med disse.

5.1 VHSIC Hardware Description Language (VHDL)

Det finnes en rekke språk for beskrivelse av maskinvare. Etter en gjennomgang av de tilgjengelige alternativene, falt vårt valg naturlig på VHDL som er det mest utbredte av disse språkene. Det er det eneste som er standard. Det ble understøttet av verktøyet vi brukte.

VHDL ble opprinnelig utviklet på initiativ fra et program kalt Very High Speed Integrated Circuits (VHSIC) igangsatt av amerikanske myndigheter. Det ble deretter videreutviklet og adoptert som IEEE Standard 1076 (11) i 1987.

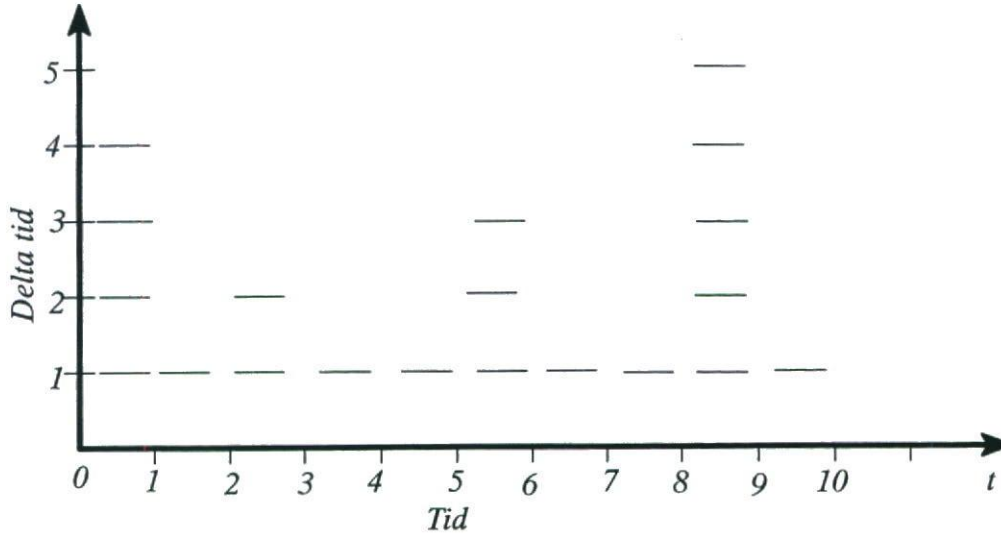
Opprinnelig var VHDL utviklet med dokumentasjon i tankene. Ved hjelp av VHDL kunne man lage entydige (kjørbare) spesifikasjoner. Språket inneholder mekanismer for å modellere tre aspekter ved integrerte kretser; oppførsel, timing og struktur. Dermed gir VHDL oss muligheten til å beskrive kretskonstruksjonen på et langt høyere nivå enn tradisjonelle tekniker som krever portnivå beskrivelse. Ved å skrive i VHDL har man hevet abstraksjonsnivået til et systemnivå der vi beskriver oppførselen til systemet og overlater til syntheseverktøyet å beskrive kretsen på portnivå.

En slik VHDL-beskrivelse kan være teknologiuavhengig. Dette letter gjenbruk av ferdige konstruksjoner. En og samme beskrivelse kan syntetiseres forskjellig for flere teknologier. Da maskinvarebeskrivelsen er skrevet i et standard språk, er den portabel mellom maskiner og verktøy. Ved bruk av VHDL er det også mulig å oppdage feil på et tidligere tidspunkt ved hjelp av simuleringer.

5.2 Simulering

En av grunnene til å bruke et programmeringspråk som VHDL til å beskrive maskinvare er muligheten til å simulere konstruksjonen før vi implementere den. VHDL har innebygde mekanismer for å understøtte simulering. Blant annet har det en innebygd modell for tid, det gir mulighet for å kjøre forskjellige modeller av samme krets sammen, det understøtter filhåndtering og det gir mulighet for å modellere omgivelsene vha en testbenk (12) (14).

VHDL har tid som innebygd type med oppløsning fra femtosekunder til timer. For å kunne simulere parallellitet, har det en todelt tidsmodell: tid og "delta-tid". For hvert tidspunkt i simuleringen avanseres delta-tiden i steg til alle hendelser på tidspunktet har hatt effekt. Som vist i figur 5.1, går stegene i delta-tid ikke langs positiv tidsakse, men innen den minste tidsoppløsningen som VHDL har. Hovedtiden står stille mens man lar logikken som vi simulerer få tid til å stabilisere seg.



Figur 5.1 VHDL's todelte tidsakse.

Tidlig i konstruksjonsarbeidet benyttet vi verktøyet Qhsim (13) (fra Metor Graphics) til å simulere kretsen med. Verktøyet har et grafisk grensesnitt slik at vi kunne studere bølgeformer ol. Senere, spesielt under verifikasjonen av det ferdige designet, var informasjonsmengden så stor og simuleringstiden så lang at dette ikke var hensiktsmessig.

Disse simuleringene ble gjort med bruk av filhåndtering i en testbenk. Det ble fremdeles benyttet samme kompilator og simulator, men uten det grafiske grensesnittet. Testbenken ga mulighet for automatisk å kontrollere to forskjellige modeller mot hverandre. Så kan en feilmelding skrives til fil hvis en feil detekteres.

For å redusere muligheten for systematiske feil, er det ønskelig at modellen som sammenlignes er utviklet så uavhengig av hverandre som mulig. Man kan f eks sammenligne oppførselsmodellen av kretsen med portnivåbeskrivelsen. Da kretsen benytter (et tilnærmet) standard tallformat, var det enkelt (ved hjelp av C-kode) å verifisere kretsen mot regne-

verket i en Hewlett Packard HP9000/889 PA-RISC UNIX maskin. Det viste seg at med denne metoden kunne verifikasjonen gjøres enkelt, elegant og effektivt. Samtidig sikret den oss mot eventuelle systematiske programmeringsfeil i VHDL-koden.

5.3 Syntese

En annen motivasjon for å bruke VHDL er muligheten for automatisk syntese. Vi kan, ifølge reklamen, se på konstruksjonen i abstrakte termer, konsentrere oss om de store linjene og overlate detaljene til syntesepprogrammet som automatisk oversetter vår abstrakte kode til en strukturell beskrivelse på et lavere abstraksjonsnivå (15)(18).

Syntheseverktøyet har et sett kontrollkommandoer og innskrenkningsmekanismer som gjør det mulig å styre oversettelsen av VHDL-kode til en portnivå beskrivelse av konstruksjonen. De parameterne som brukeren kan påvirke utenom selve koden er:

- Ønsket hastighet
- Ønsket størrelse

For å oppnå større hastighet må man ofte bruke større areal. Hvis vi ønsker å bruke så lite plass som mulig blir kretsen ofte tregere. De to parameterne representerer mao motstridende interesser. Den største utfordringen ved bruk av er syntheseverktøy er dog å skrive optimal VHDL-kode (1)(3). Frihetsgradene i programeringsspråket er langt større enn i syntheseverktøyet. Det er ikke noe problem å skrive VHDL-kode som ikke lar seg realisere med et syntheseverktøy. Dette setter langt større krav til kunnskap om syntheseverktøyet hos den som skal spesifisere en konstruksjon enn ønskelig.

Til tross for at en konstruktør kan heve sitt abstraksjonsnivå betraktelig ved å bruke VHDL og syntese, er det helt nødvendig at konstruktøren hele tiden har et realistisk bilde av hva som lar seg realisere i virkeligheten. Dette er også viktig for å være i stand til å vurdere resultatet av syntesen og trekke erfaringene tilbake til kodeskrivingen. Dagens syntheseverktøy fungerer best på sekvensiell logikk og har en tendens til å gå i ball hvis konstruksjonen blir stor. Det er derfor viktig å partisjonere designet og gjøre det så modulært som praktisk mulig.

Ved syntese av addisjonskretsen fikk vi problemer med syntesetiden, som ble svært lang, samt at Synopsys innebygde opsjon for automatisk innlegging av pipeline-trinn ikke fungerte på så store design som addisjonskretsen. Vi ble derfor nødt til å dele opp kretsen og manuelt legge inn pipline-registerne.

Som før nevnt, ble kretsen konstruert med strenge krav til forbruk av areal. Et lavt antall logiske porter var altså sterkt ønsket. Vi har av den grunn høstet en del erfaring vi gjerne deler med andre. Under følger et eksempel på fire programmeringstiler som utfører akkurat den samme operasjonen, men er skrevet litt forskjellig. Syntheseverktøyet er Synopsys Design Compiler. Den eneste betingelsen som er satt er et krav om 10 ns klokkeperiode. (Alcatel Mietec 0.35μ teknologi).

Koden i de følgende eksempler bytter om verdiene i to registere på grunnlag av visse kriterier. Tilsvarende kode er brukt i addisjonskretsen for å garantere at den største addenden alltid blir liggende i register a1.

Eksempel a:

```
if (b.exp > a.exp) then
  tmp := a;
  a1 := b;
  b1 := tmp;
elsif ((b.exp = a.exp) then
  if (b.mant > a.mant) then
    a1 := b;
    b1 := tmp;
  end if;
else
  a1 := a;
  b1 := b;
end if;
res <= a1;
```

Eksempel b:

```
a1 := a; --
b1 := b; --
if (b.exp > a.exp) then
  tmp := a;
  a1 := b;
  b1 := tmp;
elsif (b.exp = a.exp)  then
  if (b.mant > a.mant) then
    a1 := b;
    b1 := tmp;
  end if;
end if;
res <= a1;
```

Eksempel c:

```
a1 := a; --
b1 := b; --
if (b.exp > a.exp) then
  tmp := a;
  a1 := b;
  b1 := tmp;
end if;
if (b.exp = a.exp)  then
  if (b.mant > a.mant) then
    a1 := b;
    b1 := tmp;
  end if;
end if;
res <= a1;
```

Eksempel d:

```
a1 := a; --
b1 := b; --
if (b.exp > a.exp) then
  tmp := a;
  a1 := b;
  b1 := tmp;
end if;
if ((b.exp = a.exp) and (b.mant > a.mant)) then
  a1 := b;
  b1 := tmp;
end if;
res <= a1;
```

Eks	kombinatorikk #porter	ikke kombinatorisk #porter	totalt #porter	hastighet i ns
a	310	328	638	-0.83
b	267	157	425	-0.00
c	153	143	296	-0.73
d	176	142	318	-0.59

Tabell 5.1 *Oppsummering av eksempel 1. Alle eksemplene på kodestil møtte tidskravene, men, med svært forskjellig arealforbruk. Merk at totalt arealforbruk varierer sterkt. Negativ tid betyr at syntheseverktøyet har greid kravet med margin gitt av oppgitte tallverdien.*

Eksempel a benytter en vanlig *if-elsif-else* konstruksjon. I eksempel *b* har vi byttet ut else med å tilordne *variablene a* og *b* verdier før *if*-testen. Dette kan gjøres uten å forandre funksjonalitet fordi en av testene i eksempel a alltid vil slå til. Som vi ser av tabell 5.1 medfører denne forandringen at arealforbruket ble redusert med 33%. I eksempel *c* har vi delt opp *if-elsif*-konstruksjonen i to separate *if*-konstruksjoner. Dette ga en ytligere reduksjon på 30%. I eksempel *d* ble den doble *if*-testen rullet ut og satt som en enkelt *if*-test. Dette resulterte i en liten økning av arealet i forhold til eksempel *c*. Med enkle håndgrep ble altså arealforbruket halvert. Vi ser av dette eksemplet hvor viktig det er å ha kjennskap til hvordan syntheseverktøyet jobber. Metoden har på ingen måte fritatt konstruktøren fra å ha inngående kjennskap til maskinvaren han konstruerer.

6 SYSTEMBESKRIVELSE FOR ADDISJONSKRETSEN

Dette kapitelet er en systembeskrivelse av addisjonskretsen. Det inneholder en funksjonell beskrivelse av kretsen, en beskrivelse av grensesnittet mot omverdenen og en skisse av den indre strukturen til kretsen.

6.1 Kretsens egenskaper

Addisjonskretsen utfører addisjon av to 24-bits flyttall. Den er konform med IEEE Standard-754 med de unntak som er beskrevet i kapitel 3. Arkitekturen er optimalisert mot

lavt portforbruk og høy gjennomstrømming. På grunn av pipelineing er resultatene forskjøvet med tre klokkeperioder. Det følgende er en liste over egenskapene til kretsen:

- Addisjonskrets for 24-bits flyttall.
- Kretsen er pipelinet med tre pipeline-trinn.
- Klokkefrekvensen er avhengig av hvilken teknologi som velges.
- Kretsen utfører både addisjon og subtraksjon.
- Hold-funksjonalitet er implementert. Kretsen kan pauses uten at data går tapt.
- Reset-funksjonalitet er implementert. Kretsen kan nullstilles uavhengig av inngangsdata.

Vi angir størrelsen av to versjoner av addisjonskretsen; en generell og en optimalisert for Alcatel Mietecs $0.35\mu m$ CMOS prosess (2). Begge kretsene er beskrevet i VHDL og optimalisert for syntese med Synopsys Design Compiler (18).

- Størelse uten pipeline: 2195 portekvivalenter
- Størrelse med tre trinn pipeline: 3293 portekvivalenter

6.2 Pinnebeskrivelse

Tabell 6.1 inneholder en pinnebeskrivelse for addisjonskretsen. Alle signaler er representert med navn slik de er presentert i VHDL koden i appendiks A. For hvert signal angis retning, ordbredde, oppdeling og funksjon. Vi angir også hvilken datatype signalet har i VHDL koden.

Alle signaler er synkrone med ett unntak; kretsen benytter asynkron reset. Signalene skifter verdi på stigende klokkeflanke.

Pinne-navn	Ret-ning	Bit bredde	Bit opp-deling	Funksjon	VHDL type	Funksjonsopp-deling
a	inn	24	1	Operand A inn til addi-sjonskretsen	nfloat	fortegn
			8			eksponent
			15			mantisse
b	inn	24	1	Operand B inn til addi-sjonskretsen	nfloat	fortegn
			8			eksponent
			15			mantisse
op	inn	1	-	Addisjon eller subtraksjon	std_logic	0= addisjon 1=subtraksjon
clk	inn	1	-	Klokkeinngang	std_logic	
resetn	inn	1	-	Nullstiller kretsen	std_logic	aktiv lav
hold	inn	1	-	Stopper kretsen	std_logic	aktiv høy
res	ut	24	1	Resultatet av addisjonen	nfloat	fortegn
			8			eksponent
			15			mantisse
ufl	ut	1	-	Flagger underflow	std_logic	
ofl	ut	1	-	Flagger overflow	std_logic	
ivo	ut	1	-	Flagger ugyldig operasjon	std_logic	

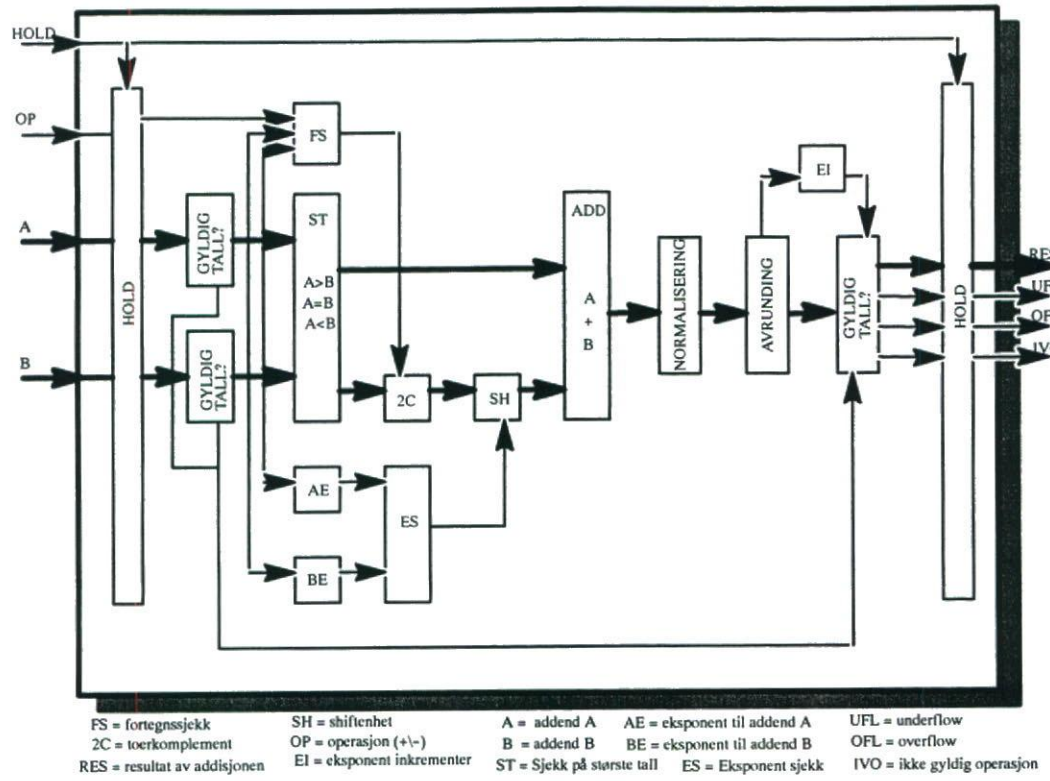
Tabell 6.1 Pinneoversikt for flyttallsadderen. VHDL typen nfloat er egendefinert.

6.3 Strukturell beskrivelse

Den indre strukturen til de to versjonene av addisjonskretsen vises i figur 6.1 og 6.2.

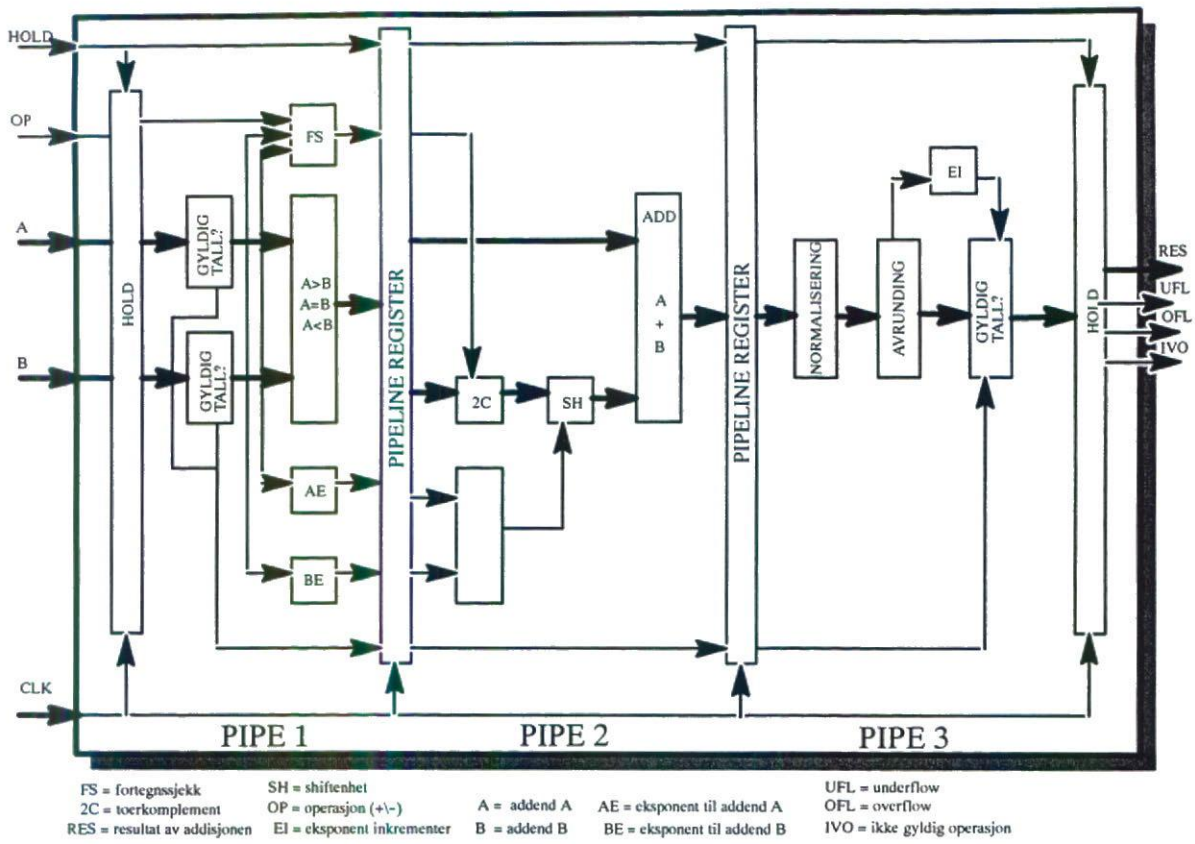
Figur 6.1 viser addisjonskretsen uten pipelineing. Som figuren viser, trenger vi to sett med registere for å implementere hold-funksjonaliteten. To moduler sjekker om operandene er gyldige. Disse modulene består av komparatorer. Fortegnssjekk modulen, FS, utfører fortegnssjekk og bestemmer om operasjonen er en addisjon eller en subtraksjon. Modulen består av en komparator. ST modulen finner addenden med størst absoluttverdi og består av komparatorer. AE og BE modulene regner ut henholdsvis $a_{exp} - b_{exp}$ og $b_{exp} - a_{exp}$ og består av to 8 biters addisjonskretser. Modulen 2C inneholder logikk for å toerkomplimentere addend *b*. ES modulen velger, på grunnlag av kompareringen i ST modulen, hvor mye addend *b* må justeres før addisjon. SH modulen justerer addend *b* i henhold til justeringsverdien fra ES. Modulen er et 8 biters justeringsregister. I ADD modulen adderes de to addendene. Modulen er en 17 biters heltallsaddisjonskrets. Normaliseringenheten består av komparatorer og justeringsregister. Avrundingsmodulen runder av svaret etter IEEE-754 standarden. Den består av en komparator og en justeringsenhet. Eksponentjusteringsmodulen, EI, øker eksponent verdien med 1 på grunnlag av avrundingen hvis nødvendig.

Den består av en inkrementer. Nest siste modul sjekker om svaret er et gyldig tall. Hvis svaret ikke er gyldig, settes en unntaksverdi ut og et av unntaksflaggene settes. Modulen består av komparatorer. Deretter følger et sett med registere for å implementere hold-funksjonaliteten.



Figur 6.1 Funksjonelt blokkdiagram over den indre strukturen til flyttallsaddisjonskretsen.

Figur 6.2 viser den samme kretsen, men med 3 pipeline-trinn plassert på grunnlag av synteseresultater. Forskjellen fra kretsen beskrevet over er innføringen av to sett med klokkede registere som fungerer som pipeline-registere.



Figur 6.2 Funksjonelt blokkdiagram over den indre strukturen til den pipelinede flyttall-saddisjonskretsen.

7 TESTING OG VERIFIKASJON

I dette kapitlet beskrives testing og verifikasjon av addisjonskretsen. Vi beskriver hvilke valg og avveininger som er gjort i forhold til teststrategi og testmønstre.

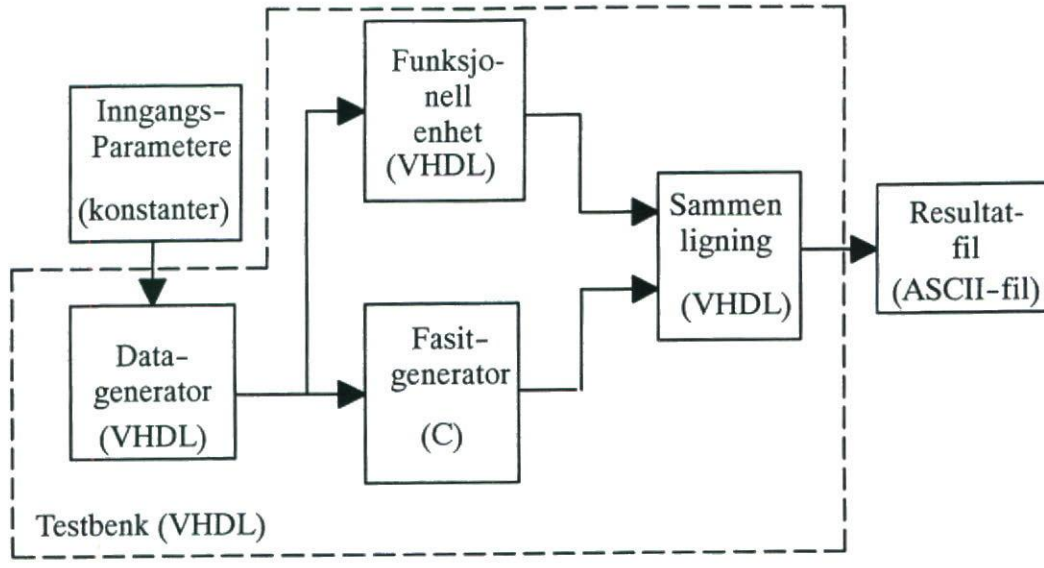
For å kunne utnytte den økende kompleksiteten til digitale elektroniske systemer, har kretskonstruktørene tatt i bruk høynivåbeskrivelse og syntese for å øke sin produktivitet. For testing og verifikasjon av kretsene, har en tilsvarende utvikling ikke funnet sted. Denne delen av arbeidet utgjør derfor en stadig økende del av det totale tidsforbruket. Det er ikke uvanlig at 50 - 60 % av tiden går med til test og verifikasjon. Spesielt kan det være tidkrevende å generere testbenken og finne fram til gode testmønstre (7).

I tillegg krydres det hele av det faktum at jo høyere opp i abstraksjonsnivå en feil er gjort, jo vanskeligere er den å finne.

7.1 Teststrategi

Bruk av manuelt genererte stimuli som sjekkes manuelt er nyttig kun i den helt innledende fasen av test- og verifikasjonsarbeidet. Omfanget av de videre arbeidet krever at det foregår mest mulig automatisk uten inngrep fra konstruktøren. Dette var utgangspunktet for utviklingen av test- og verifikasjonssystemet for addisjonskretsen. For å oppnå dette, gjør

systemet utstrakt bruk av testbenk og filhåndtering. En skisse av systemet finnes i figur 7.1.



Figur 7.1 Skisse av test- og verifikasjonssystemet for addisjonskretsen.

Den funksjonelle enheten (addisjonskretsen) settes inn i en testbenk skrevet i VHDL. Testbenken inneholder en prosess for automatisk generering av inngangsdata, en prosess for automatisk generering av fasitdata og en prosess som automatisk sammenliger simuleringsresultater og fasitdata. Fasitgeneratoren er en C-modell av den aktuelle funksjonelle enheten.

For å redusere muligheten for systematiske feil, er det ønskelig at modellen som benyttes til fasitgenereringen er utviklet uavhengig av kretsen som skal testes. Siden addisjonskretsen benytter et (tilnærmet) standard tallformat, kunne fasitgeneratoren utnytte regneverket i en Hewlett Packard PA-RISC 9000/889 UNIX maskin. Dette ga en fasit med høy grad av troverdighet og med liten mulighet for systematiske feil.

Kildekode for testbenken finnes i appendiks B, mens appendiks D inneholder kildekode for fasitgeneratoren.

7.2 Test- og verifikasjonstimuli

I motsetning til andre komplekse digitale konstruksjoner, har addisjonskretsen begrensede valg/programmerings muligheter. I all sin enkelhet utfører den en aritmetisk operasjon på to operander og produserer ett resultat. Men, selv ikke for et så enkelt system som addisjonskretsen er en fullstendig test mulig. Man må forsøke å oppnå høyest mulig feildekningsgrad ved hjelp av et begrenset antall testmønstre.

En mye brukt metode er å generere inngangsverdier ved å trekke tilfeldige tall. Metoden er svært enkel, men gir ofte ikke høy nok feildekningsgrad alene. Problemet er at den er lite

egnet til å ta feil som er vanskelige å finne. Sannsynligheten for at man får testet alle eksotiske tallkombinasjoner som gir spesialtilfeller er for liten. Derfor bør metoden kompletteres med tester som er skreddersydd for slike spesialtilfeller.

En funksjonalitet som må testes ekstensivt er avrundingsenheten som avrunder svaret på grunnlag av guard (g), round (r) og sticky (s) bitene. Det ble derfor laget en testsuite som tok for seg alle mulige kombinasjoner av mantisseforskyvninger som medfører alle mulige bit kombinasjoner av g, r og s.

En annen enhet som fortjener spesiell oppmerksomhet er normaliseringsmodulen. Det ble også her skrevet en testsuite hvor alle normaliseringsfunksjoner ble testet grundig.

Med utgangspunkt i figur 3.2, som viser dynamikkområdet til tallformatet vårt, kan vi lett identifisere fire tilfeller som bør testes inngående:

1. Operasjoner som involverer ± 0 med alle fortegnskombinasjoner.
2. Operasjoner som involverer tall i tallområdet rundt det minste tallet som kan representeres og denormaliserte tall på både positiv og negativ side.
3. Operasjoner som involverer tall i tallområdet rundt det største tallet som kan representeres på både positiv og negativ side.
4. Operasjoner som involverer $\pm \infty$ med alle fortegnskombinasjoner.

7.3 Datagenerator med inngangsparametre

Vi benytter en VHDL-basert datagenerator. Et sett parametere bestemmer lengden på datasettet og datasettets oppbygning. For å ha flest mulige frihetsgrader ble datageneratoren laget med flere løkker inne i hverandre som følger:

Innerst:	mantisse til operand2
	mantisse til operand1
	eksponent til operand2
	eksponent til operand1
	fortegn til operand2
	fortegn til operand1
	eventuelt operasjonskode
Ytterst:	parametersett

Alle løkkene gjennomløper alle sine verdier en gang mellom hver gang utenforliggende løkke skifter verdi. Dette gir en fleksibel datagenerator. Det følgende er en kortfattet presentasjon av parametrene som styrer testmønstergenereringen.

Mantissene kan lages på fire forskjellige måter:

1. Her vil alle mulige verdier benyttes. En slik fullstendig test vil det med vårt tallformat neppe være realistisk å gjennomføre.

Løkkevariabelen brukes direkte. Gyldige verdier er 0 - 32767

2. Her kjører vi et utsnitt av tallområdet.

   Løkkevariabelen = 0 => mantisse = "000000000000000"
   Løkkevariabelen = 1 => mantisse = "000000000000001"
   Løkkevariabelen = 2 => mantisse = "111111111111111"
   Løkkevariabelen = 3 => mantisse = "111111111111110"
   Gyldige verdier 0 - 3.

3. Her starter vi med alle mantissebitene satt til 1 og for hver runde i testen skifter vi inn en 0 i LSB posisjonen og skifter ut bitet i MSB mao en venstreskift operasjon. Dette kalles "fence of zeroes".

   Løkkevariabelen = 0 => mantisse = "111111111111111"
   Løkkevariabelen = 1 => mantisse = "111111111111110"
   Løkkevariabelen = 2 => mantisse = "111111111111100"
   Løkkevariabelen = 3 => mantisse = "111111111111000"

   Her starter vi med alle mantissebitene satt til 0 og for hver runde i testen skifter vi inn en 1 i LSB posisjonen og skifter ut biten i MSB. Vi utfører mao en venstreskift operasjon. Dette kalles "fence of ones".

   Løkkevariabelen = 13 => mantisse = "100000000000000"
   Løkkevariabelen = 14 => mantisse = "000000000000000"
   Løkkevariabelen = 15 => mantisse = "000000000000001"
   Løkkevariabelen = 16 => mantisse = "000000000000011"

4. Her starter vi med alle mantissebitene satt til 0. For hver runde i testen skifter vi en 1 en posisjon mot venstre. Dette kalles "walking one".

   Løkkevariabelen = 28 => mantisse = "011111111111111"
   Løkkevariabelen = 29 => mantisse = "111111111111111"
   Løkkevariabelen = 30 => mantisse = "000000000000001"
   Løkkevariabelen = 31 => mantisse = "000000000000010"

   Her starter vi med alle mantissebitene satt til 1. For hver runde i testen skifter vi en 0 en posisjon mot venstre. Dette kalles "walking zero".

   Løkkevariabelen = 43 => mantisse = "010000000000000"
   Løkkevariabelen = 44 => mantisse = "100000000000000"
   Løkkevariabelen = 45 => mantisse = "111111111111110"
   Løkkevariabelen = 46 => mantisse = "111111111111101"

   Løkkevariabelen = 58 => mantisse = "101111111111111"
   Løkkevariabelen = 59 => mantisse = "011111111111111"

   Gyldige verdier 0 - 59.

Parametere til mantissegenerering er startverdi, avstand mellom verdier, stoppverdi og hvilket mantissemønster som skal brukes.

Eksponentene kan gå gjennom opptil tre forskjellige tallområder med hver sin startverdi, avstand mellom verdier og stoppverdi. Tallområdene må velges slik at eksponentverdien alltid er stigende.

Fortegnene starter som '0'='+' og skifter til '1'='-'.

Operasjonskode brukes ved testing av adder. '0' = addisjon og '1' = subtraksjon.

Flere parametersett kan brukes for å kjøre flere tester i samme testkjøring.

Parametersetting for de ulike datasettene finnes i appendiks E.

7.4 Utførte tester

Det følgende er en kortfattet gjennomgang av hvilke tester som er kjørt og hvilke feilsituasjoner de ulike testene er myntet på.

Datasett 1

Begge mantissene går gjennom fullt sett med "fence of zeroes", "fence of ones", "walking one" og "walking zero". Eksponentene holdes konstante på verdien 100 (64h).

Totalt 60*60*8=28.800 operasjoner = 432.000ns simulert tid.

Dette gir en god test av mantisseaddisjon/subtraksjon, men lite testing av eksponenthåndtering og mantisseskifting. Datasettet er lite og dermed raskt å kjøre simulering på.

Datasett 2

Mantissene går gjennom fullt sett med "fence of zeroes", "fence of ones", "walking one" og "walking zero". Eksponentene går gjennom sekvensen 0, 1, 2, 127, 128, 253, 254, 255 (heksadesimalt 0, 1, 2, 7F, 80, FD, FE, FF).

Totalt 60*60*8*8*8=1.843.200 operasjoner = 27.648.000ns simulert tid.

Hovedhensikten med dette datasettet er å teste ekstremtilfeller, dvs veldig små og veldig store eksponenter både i inngangsdata og i resultatet. Mange av eksponentkombinasjonene kunne vært testet tilfredsstillende med færre mantisseverdier, men for å forenkle parameteroppsettingen og minske sjansene for noen tilstander ikke ble testet, valgte vi å ta med full mantissegenerering overalt.

Datasett 3

Mantissene har verdiene "0000", "0001", "7FFF", "7FFE".
Eksponentene telles opp fra 105 (69h) til og med 149 (97h).

Totalt 4*4*45*45*8=259.200 operasjoner = 3.888.000ns simulert tid.

Dette datasettet tester mantisseskifting.

Datasett 4

Mantissene har verdiene "0000", "0001".
Eksponentene går sekvensielt gjennom alle verdier "00"-"FF"

Totalt 2*2*256*256*8=2.097.152 operasjoner = 31.457.280ns simulert tid. Tester alle mulige eksponentkombinasjoner.

8 SYNTESERESULTATER

For å øke sannsynligheten for at det ferdige produkt skal fungere tilfredsstillende, er det nødvendig å gi synteseverktøyet en så detaljert beskrivelse som mulig av kretsens operasjonsbetingelser. For å beskrive dette har synteseverktøyet et sett med parametre. I dette kapitlet beskrives disse. Synteseresultatene presenteres også.

I det følgende beskrives de viktigste parameterne som kan benyttes for å beskrive kretsens operasjonsmiljø. Parametrene er gruppert etter hvilke deler av det operative miljøet de beskriver.

8.1 Parametre som beskriver egenskapene til kretsens omgivelser

I de fleste teknologier påvirker svingninger i temperatur, forsyninigsspenning og prosessparametre kretsens ytelse (hastighet). Dette inngår i spesifiseringen av systemets omgivelser.

Temperatur variasjoner.

Med mindre systemet brukes i svært godt kontrollerte omgivelser er temperatur variasjoner uunngåelige. Systemets respons på temperatur variasjoner kan variere, men systemer blir generelt tregere ved stigende temperatur.

Variasjoner i forsyningsspenningen.

Forsyningsspenningen påvirker kretsens hastighet og ved store strømtrekk kan spenningsvariasjonene være betydelige.

Variasjoner i prosessparametrene.

Variasjoner i prosessparametrene er uunngåelig og må derfor inngå som et prosentvis avvik i alle kalkulasjoner.

Ved tidsanalyse av kretsen tar synteseverktøyet hensyn til både beste og verste tilfelle av for variasjoner i prosess-, temperatur- og spenningsvariasjoner.

8.2 Parametre som beskriver last modeller av nettverket

Beregninger av lasten nettverket representerer er gitt av lederens lengde og hvor mange porter som skal drives (fanout). Beregningen skjer på grunnlag av lederens motstandsverdi, kapasitans og areal. Disse faktorene har stor betydning for kretsens hastighet.

En nettverkslastmodell beskriver forholdet mellom lengden på lederen og antall porter som skal drives. ASIC-produsentene oppgir lastmodeller basert på statistisk informasjon for spesifikke prosesser. I mangel av informasjon om konstruksjonen bruker synteseverktøyet denne informasjonen til å beregne lengden på lederne i et design. Synteseverktøyet velger, i prioritert rekkefølge, mellom tre måter å estimere lederlengene på:

- Lengder oppgitt av kretskonstruktøren.
- Automatisk valgt på bakgrunn av kretsens areal.
- I mangel av noe annet brukes standard verdier fra teknologibiblioteket.

Uten denne informasjon kan ikke synteseverktøyet gjøre noen realistisk analyse av kretsen og man ender opp med optimistiske/urealistiske tidsberegninger. I hierarkiske nett må man også bestemme hvilke modeller som skal brukes mellom de forskjellige blokkene. Dette spesifiseres på samme måte som over.

8.3 Parametre som beskriver systemets grensesnitt

De måter man har til å påvirke verdiene på modellens grensesnitt er gitt av:

Definering av inngangs drivere.

Synteseverktøyet bruker informasjon om en ports driveregenskaper til å velge en riktig driver til et nett. I utgangspunktet antar verktøyet at inngangen har null drivermotstand, hvilket vil si at man har uendelig driverstyrke. For å overstyre dette urealistiske valget kan konstruktøren definere porttype og dens driveregenskaper.

Definering av lasten på inn- og utganger.

For at synteseverktøyet skal være i stand til å velge fornuftige drivere til utsignaler, må det spesifiseres hvilken type last som skal drives. Dette gjøres ved å spesifisere kapasitansen til inn- og utgangssignalene til kretsen. Dette bruker verktøyet til å velge driverstyrke på utportene og å beregne transisjonstiden på inngangene.

Definering av lasten som skal drives av utgangene.

Lasten som skal drives av en utgang kan settes eksplisitt og synteseverktøyet vil da søke å tilordne en last mindre eller lik den oppgitte.

8.4 Spesifisering av kretsens omgivelsesparametre

Synteseverktøyet optimaliserer designet på bakgrunn av de oppgitt omgivelsesparametrene. I utgangspunktet er ingen omgivelsesparametre satt. Prosessparametrene er forskjellig fra teknologi til teknologi og må være oppgitt i teknologibiblioteket.

Prosessvariasjoner.

Innbefatter variasjoner i fabrikasjonsprosessen av kretsen og er vanligvis oppgitt som et prosentvis avvik fra antatt verdi.

Spenningsvariasjoner.

Her oppgis spenningen kretsen skal fungere ved som ofte er et avvik fra kretsens ideelle driftsspenning.

Temperaturvariasjoner.

Her oppgis temperaturområdet man vil kretsen skal fungere ved.

Sammenkoblings modeller.

Her defineres hvilken modell som skal brukes for drivere og for nettverkslast. Det er tre modeller: best_case_tree, balanced_tree og worst_case_tree. Ved best_case, er driveren i umiddelbar nærhet av lasten. Ved worst_case, er driveren langt unna lasten og både motstanden og kapasitansen til lederen inngår i beregningene. Ved balanced_tree er avstanden mellom driver og last moderat, i tillegg er bidraget fra kapasitans og motstand like.

For å sikre oss at kretsen fungerer i normal bruk og for å ha litt sikkerhetsmargin er det vanlig å oppgi litt tøffere krav en det kretsen er tiltenkt å brukes ved. I tabell 8.1 oppgir vi verdiene på parametrene som beskriver vårt valg av operasjonsmiljø for vår krets:

Operasjonsbetingelser	Verdier
Prosess avvik	150%
Temperatur	85 grader C
Forsyningsspenning	3.00 V
Sammenkoblings modell	Balanced_tree

Tabell 8.1 *Tabellen viser vårt valg av operasjonsparametre som vi har brukt under syntese. Verdiene er basert på Alcatel Mietec's Worst Case Industrial (WCIND) prosess parametre.*

8.5 Synteseresultater

For å ha en mulighet til å sammenligne og forstå resultatene er det viktig å ha kjennskap til hvilke attributter som er satt og deres verdi. Synteseresultatene kan måles på mange måter, men de mest interessante er:

- Areal
- Hastighet

Til å beskrive areal er det vanlig å bruke port-ekvivalenter. En port-ekvivalent er definert som arealet til en NAND-port med to innganger. I CMOS er overgangen fra portekvivalenter til transistorer gitt av at en realisering av en NAND-port krever fire transistorer. Dette gir bare en pekepinn om det faktiske arealet. For eksempel hvor kompleks ruting kretsen inneholder påvirker også arealet.

Hastigheten til en port avhenger av, i tillegg til prossesspesifikke parametere, geometrisk utforming på transistorene, kapasitansen til lasten som skal drives og tidsforsinkelsen for transportveien til signalet. For å beregne forsinkelsen fra ledningsføringen brukes lastmodeller. Disse lastmodellene kan være en del av teknologibibloteket fra prosesshuset. Dette er statiske modeller som bygger på gjennomsnittlig fan-out for en krets av en gitt størrelse (16). Dette gir grunnlaget for å regne ut forsinkelsesbidraget fra lederne. Lasten forbundet med å drive en inverter kalles en standardlast. Denne standardlasten bruker syntheseverktøyet som måleenheten for last. Kapasitansen til en inverter er gitt i teknologibibloteket. Synteseprogrammet bruker dette til å regne ut forsinkelsen gjennom en krets slik at det på grunnlag av tidskravene kan velge nær optimal driver til hver komponent.

8.6 Synteseresultater for addisjonskreten

Resultatene bygger på syntesekjøringer med Synopsys Design Compiler, alle script og innstillinger er gitt i appendiks C. Syntese resultater for den upipelinede utgave av addisjonskretsen er gitt i tabell 8.2. Total celleareal for kretsen er 2195 portekvivalenter med et tidskrav på 12 MHz. (Dette er det raskeste synteseverktøyet klarte på denne kretsen).

Del av kretsen	Antall
Number of ports:	79
Number of nets:	1117
Number of cells:	944
Number of references:	99
Combinational area:	1681.42 portekviv.
Noncombinational area:	514.00 portekviv.
Total cell area:	2195.42 portekviv.

Tabell 8.2 Synteseresultater for den upipelinede utgave av addisjonskretsen.

Synteseresultatene for kretsen, som ble pipelinet manuelt, med tre pipelinetrinn er gitt i tabell 8.3. Totalt celleareal er 3293 portekvivalenter når tidskravet på 66 MHz er oppfylt. (66MHz er hastigheten på vår systemklokke.)

Del av kretsen	Antall
Number of ports:	79
Number of nets:	1663
Number of cells:	1383
Number of references:	152
Combinational area:	2668.75 portekviv.
Noncombinational area:	624.01 portekviv.
Total cell area:	3292.76 portekviv.

Tabell 8.3 Syntese resultatene for kretsen, som ble manuelt pipelinet, med tre pipelinetrinn.

Uten å innføre annen logikk en pipelineregisterne øker kretsen areal med ca 50%. Med fire pipelinetrinn får man en krets med 3841 portekvivalenter, med fem pipelinetrinn 4390 portekvivalenter, hvilket er en dobling av arealet. Dette viser klart at det er en avveining mellom gjennomstrømmning og hastighet. Man kan ikke pipeline seg bort fra alle problemer. Vi må kjøre en del av beregningene i parallell for å øke hastigheten til kretsen. Med et krav til systemklokke på 66MHz, kom vi til at en krets med tre pipelinetrinn og en del parallell logikk var optimalt for oss.

9 KODEBESKRIVELSE

Vi vil i dette kapitlet i noen grad beskrive VHDL-koden og kommentere noen av håndgrepene som er utført for å holde syntese verktøyet i tøylene. Vi starter med å beskrive de grunnleggende egendefinerte typefilene for deretter å gå igjennom selve addisjons prosessen. Standard VHDL-biblioteker kommenteres ikke.

9.1 Egendefinerte VHDL-biblioteker

Av bekvemlighetshensyn, og for enkelt å ha muligheten til å forandre på bitbredder og sammensetninger av data og kontroll ord, er det praktisk å samle alle deklarasjoner og definisjoner på globale datatyper og konstanter i et felles bibliotek. Et slik bibliotek kalles i VHDL for en *package*. Vi har i filen type.vhd, som er gjengitt i appendiks A.2, definert addisjonskretsens tallformat. Mansiz og expsiz beskriver henholdsvis antall bit i mantissen og eksponenten. I alle underliggende VHDL-moduler har vi benyttet oss av disse definisjonene. Hvis vi skulle ønske å forandre bitbredden på tallformatet ved et senere tidspunkt i konstruksjonsfasen, trenger vi kun å forandre verdiene et sted for hele designet. Denne fleksible måten å definere datatyper og ordbredder på er gjennomført gjennom hele designet. Dette var spesielt viktig da konstruksjonen av kretskomponenter startet før det endelige tallformatet var bestemt.

9.2 Strukturell oversikt over VHDL-koden

Vi forutsetter her en grunnleggende kjennskap til VHDL-koding og VHDL-syntaks. Oppbygningen av koden er i stor grad gitt ut i fra VHDL's syntaks. Ytterst ligger en ramme som kalles en *entity*, denne beskriver komponentens ytre bindinger og signaler. En *entity* må inneholde minst en *process*, for å gjøre noe fornuftig. Denne *processen* beskriver komponentens oppførsel og interne struktur. Vi vil i den videre diskusjonen ta for oss koden fra filen fpadd.vhd som er gjengitt i appendiks A.1.

Den beskrevne addisjonskrets, med tre pipelinetrinn, ble det naturlig å dele opp i tre interne *processer*, én for hvert pipelinetrinn. Disse tre *processene* fungerer uavhengig av hverandre og påvirkes kun av signaler utenifra og signaler *processene* imellom. Fordi addisjon er en seriell prosess, så vil dette i vårt tilfelle medføre at første *process* får innsignaler fra verden utenfor komponenten og den sender utsignaler til andre *process*. Andre *process* får innsignaler fra første *process* og sender sine utsignaler til tredje *process*. Tredje *process* får innsignaler fra andre *process* og sender sine utsignaler ut av komponenten. Det er imidlertid tre unntak: systemklokken, reset og hold. Disse signalene kommer direkte utenfra og inn til alle *processene*.

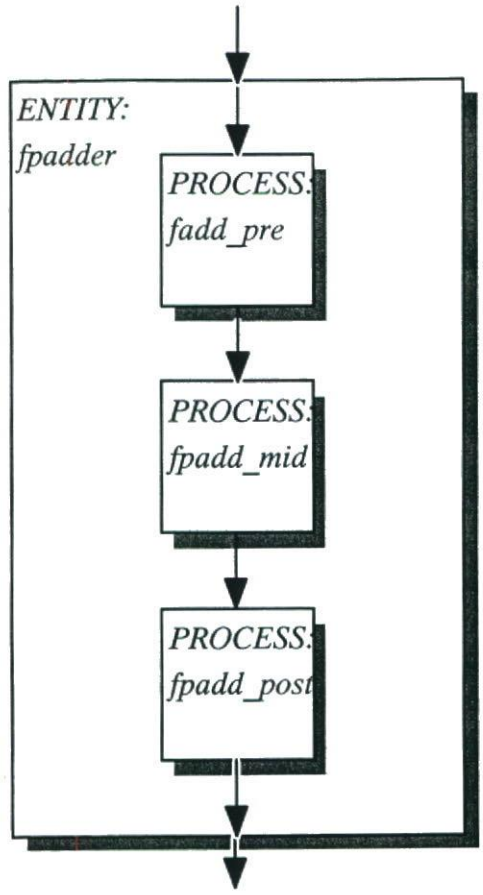
Den strukturelle oppbygningen av koden er visualisert i figur 9.1.

I *entityens* arkitektur defineres alle globale konstanter og signaler og alle inngangssignalene beskrives. Deretter følger deklarasjonen av de tre *processene*.

Første *process* starter med en deklarasjon av *processens* variabler. Deretter følger en signal til variabel tilordning der vi tar signalene fra utenomverdnen inn i *processen*. Dette gjør vi fordi vi ikke ønsker å jobbe med globale signaler inne i *processen*, men heller med lokale kopier av disse. Avslutningsvis tilordnes *processens* utsignaler verdiene til *processens* variabler. Disse settes så ut på utgangene neste klokkeperiode.

I andre *process* tilordnes signaler fra første process til andre *process* sine variabler. Utsignalene sendes til tredje *process*.

Tredje *process* inneholder i likhet med andre process en tilordning av signaler her fra andre process til tredje process sine variabler. Utvariablene tilordnes komponentens utsignaler og sendes ut av komponenten.



Figur 9.1 Visualisering av VHDL-kodens strukturelle oppbygning.

9.3 Spesielle håndgrep for syntese.

Det finnes i VHDL-kommandoer som av VHDL-kompilatoren og simulatoren tolkes som kommentarer, men som tolkes som styringssignaler av synteseverktøyet. En av disse kommandoene kan brukes til å styre synteseverktøyets valg av komponenter. Etter konstantdeklarasjonene i fila fpadd.vhd, settes en del attributter for å benyttes oss av dette. Her tvinger vi syntese verktøyet til å velge en spesiell komponent fra sitt design bibliotek. Denne komponenten brukes i *processen* fadd_pre for å overstyre valg av subtraktor til å regne ut eksponent differansene $a_{exp} - b_{exp}$ og $b_{exp} - a_{exp}$. Synteseteknisk er dette en carry look-ahead heltallssubtraktor hentet fra Synopsys sitt Design Ware bibliotek (17) (DW01_sub).

9.4 Addisjonsenhetens inn- og utgangssignaler

Addisjonskretsen inn- og utgangssignaler er:

Innsignaler:

- a, b, op, clk, resetn og hold

Utsignaler:

- res, ufl, ofl og ivo

Hvis resetn og hold er passive produserer addisjonsenheten ett resultat pr klokkeperiode. Svaret, res, er forsinket med tre klokkeperioder i forhold til inngangsverdiene, a og b, da det er 3 pipelinetrinn i kretsen. Når resetn er aktiv nullstilles alle registere i kretsen. Når hold er aktiv fryses kretsen.

9.5 Detaljert gjennomgang av VHDL-koden

Koden fra filen *fpadd.vhd* er gjengitt i sin helhet i appendiks A.1. Vår implementasjon av flyttallsaddisjons enheten består av følgende enheter:

- Enhet som sjekker om tallene inn til addisjonskretsen er gyldige.
- Enhet som sørger for at vi får det største tallet som addend.
- Enhet som sjekker operasjonstype; addisjon eller subtraksjon.
- Skiftenhet som justerer mantissen til det minste tallet.
- Addisjonsenheten som legger de to mantissene sammen.
- Normaliseringsenhet.
- Avrundingsenhet.
- Enhet som sjekker at vi har et gyldig tall ut.
- Enhet som tilordner signaler og svar på utgangene.

Sjekk på gyldige tall inn

- Innsignaler: a_1 og a_2
- Utsignaler: a_1, a_2, a_uendelig, b_uendelig, a_nan, b_nan, a_null og b_null.

Enheten sjekker at tallene inn ligger innenfor gitte grenser, ref kapitel 3. a/b_uendelig, a/b_nan, a/b_null settes aktive hvis a eller b er henholdsvis uendelig, NaN eller null.

Ombyttings enhet

- Innsignaler: a, b, a_uendelig, b_uendelig, a_nan, b_nan, a_null, b_null.
- Utsignaler: a1_sign, a1_exp, a1_mant, b1_sign, b1_exp, b1_mant, diff, diffb, swp

Enheten finner det største tallet (absolutt verdi) og setter det som addend, *a1*. Dette for å slippe å spesialhåndtere subtraksjon. Fortegnet på svaret vil derfor alltid være lik fortegnet på det største tallet. Vi regner ut både $diff = a1\_exp - b1\_exp$ og $diffb = b1\_exp - a1\_exp$ disse blir senere brukt til å bestemme antall shift som må utføres på det minste tallet før de kan legges sammen. swp settes aktiv hvis *a* og *b* byttes om dvs *b* er større enn *a*.

Sjekk på operasjon; addisjon eller subtraksjon

- Innsignaler: a1_sign, b1_sign, op_inn og b1_mant

- Utsignal: b2
- Funksjon: a1_sign xor b1_sign xor op_inn

Enheten erstatter b1_mant med sin toerkompliment hvis vi skal utføre en subtraksjon. Dette for å slippe å utføre subtraksjonen. Vi adderer a1_mant med en toerkomplementert utgave av b1_mant.

Shifting av mantisse til det minste tallet

- Innsignaler: b2, diff
- Utsignaler: b4, g, r og s

Enheten justerer b2 mantissen på grunnlag av diff slik at de to tallene kan adderes. Guard, round og sticky bitene settes på grunnlag av utskiftede bit fra b2 mantissen. Fortegnsutvider (skifter inn enere) hvis b2 er toerkomplimentert.

Addisjon av de to mantissene

- Innsignaler: a1_mant og b4
- Utsignaler: res1

Enheten legger sammen de to mantissene.

Normalisering av resultatet

- Innsignaler: res1, b4, g, r og s
- Utsignaler: res2, g, r og s

Enheten justerer res1 slik at den ledende eneren befinner seg i posisjon MSB-1. Det kan her forkomme både høyre og venstre shift. Guard, round og sticky justeres også på grunnlag av shiftoperasjonene. Eksponenten til svaret justeres også i henhold til shiftingen av svar mantissen.

Avrunding av resultatet

- Innsignaler: res2, g, r og s
- Utsignaler: res3, res_denorm, add_res

Enheten avrunder resultatet etter regler gitt i kapittel 3 og justerer om nødvendig eksponenten. Det sjekkes også for denormaliserte tall.

Sjekk på gyldig tall ut

- Innsignaler: a_nan, b_nan og add_res

- Utsignaler: res_uendelig og res_null

Enheten sjekker at svaret er innenfor området som kan representeres i vårt tallformat.

Tilordning av utganger

- Innsignaler: add_res, a_nan, b_nan, a_uendelig, b_uendelig, op_diff, res_uendelig, res_null, res_denorm.
- Utsignaler: res, ofl, ufl og ivo

Enheten tilordner verdier til alle utgangene.

10 KONKLUSJON

Vi har i denne rapporten beskrevet en teknologiuavhengig implementasjon av en addisjonskrets som støtter IEEE-754 standard for flyttall. Kretsen er syntetisert for Alcatel Mietec MTC45000 teknologi. Den er fullt uttestet og verifisert. Kretsen er parametriserbar med hensyn på tallformatet. Den vil kunne inngå som en komponent i et teknologi-bibliotek.

Kretsen er konstruert med strenge krav om lavt portforbruk med et hastighetskrav på 66 MHz. Kretsen er optimaliser for gjennomstrømmning.

Addisjonskretsen, som inneholder tre pipelinetrinn, har en total størrelse på 3293 portekvivalenter. Vi har også implementert en upipelinet utgave av kretsen med klokkehastighet på 12 MHz og total størrelse på 2195 porterekvivalenter.

Litteratur

(1) Airiau R, Berge J-M og Olive V (1994): Circuit Synthesis with VHDL, Kluwer Academic Publishers, Dordrecht.

(2) Alcatel Mietec (1998): Standard Cell Design Data Book 0.35μm CMOS, Alcatel Mietec, Brussel.

(3) Ashenden P J (1996): The Designer's Guide to VHDL, Morgan Kaufmann Publishers Inc, San Francisco.

(4) Blom H, Gundersen R: Pipelinet arkitektur for radix-128 FFT, FFI/RAPPORT under utgivelse, Forsvarets forskningsinstitutt.

(5) Cavanagh J J F (1984): Digital Computer Arithmetic, Design and Implementation, McGraw-Hill Company, San Fransisco.

(6) Coonen J T (1980): An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic, *Computer* January 1980, 68 - 79.

(7) Evans A (1998): Functional Verification of Large ASIC's, Proceedings of the 35th DAC.

(8) Goldberg D (1991): What Every Computer Scientist Should Know About Floating-Point Arithmetic, *ACM Computing Surveys, Vol.* **23**, No. **1**, 5-48.

(9) Goldberg D (1990): Computer Arithmetic. In: *Computer Architecture: A Quantitative Approch,* (D A Patterson, J L Hennessy), Appendix A, Morgan Kaufman Publishers, San Francisco.

(10) IEEE (1985): IEEE Standard 754-1985 for Binary Floating-Point Arithmetic, IEEE.

(11) IEEE (1987): IEEE Standard 1076-1987 IEEE Standard VHDL Reference Manual, IEEE.

(12) Lipsett R, Schaefer C, Ussery C (1989): VHDL: Hardware Description and Design, Kluwer Academic Publishers, Dordrecht.

(13) Mentor Graphics Corp(1998): Qhsim User's and Reference Manual, Mentor Graphics Corporation, Wilsonville Oregon.

(14) Navabi Z (1993): VHDL Analysis and Modeling of Digital Systems, McGraw-Hill, New York.

(15) Synopsys Customer Education Services (1997): Advanced Chip Synthesis Workshop Student Guide, Synopsys Inc, Mountain View.

(16) Synopsys Inc(1998): Design Compiler Reference, Version 1998.08, Synopsys Inc, Mountain View.

(17) Synopsys Inc(1998): Design Ware Foundation Quick Reference Guide, Version 1998.08, Synopsys Inc, Mountain View.

(18) Synopsys Inc (1996): Behavioral Synthesis with VHDL, Synopsys Inc, Mountain View.

APPENDIKS

A VHDL-KODE FOR FLYTTALLSADDISJONSKRETSEN

Vi vil i de følgende avsnitt gjengi VHDL-kode for addisjonskretsen, og det egendefinerte typebiblioteket.

Her følger VHDL-koden for addisjonskretsen med tre pipelinetrinn.

A.1 VHDL-filen fpadd.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
LIBRARY common_rgu;
USE common_rgu.types.all;
LIBRARY synopsys;
use synopsys.attributes.all;

ENTITY fpadder IS
  PORT ( a        : IN  nfloat;
         b        : IN  nfloat;
         op       : IN  std_logic;
         clk      : IN  std_logic;
         resetn   : IN  std_logic;
         hold     : IN  std_logic;
         res      : OUT nfloat;
         ufl      : OUT std_logic;
         ofl      : OUT std_logic;
         ivo      : OUT std_logic);
END fpadder;

ARCHITECTURE pipe3 OF fpadder IS

  signal p0_a1m : std_logic_vector(16 downto 0);
  signal p0_a1e : std_logic_vector(7 downto 0);
  signal p0_a1s : std_logic;
  signal p0_b1 : std_logic_vector(16 downto 0);
  signal p0_diff : unsigned(7 downto 0);
  signal p0_b_sign,     p0_op_inn      : std_logic;
  signal p0_a_uendelig, p0_b_uendelig  : std_logic;
  signal p0_a_nan,      p0_b_nan ,p0_swp      : std_logic;
  signal p1_a_uendelig, p1_b_uendelig  : std_logic;
  signal p1_a_nan,      p1_b_nan       : std_logic;
  signal p1_res1 : unsigned(16 downto 0);
  signal p1_sign, p1_g, p1_r, p1_s : std_logic;
  signal p1_exp : std_logic_vector(7 downto 0);
  signal p1_op_diff,p1_op_inn,p1_swp : std_logic;

  constant xpone : signed(4 downto 0) := "00001";
  constant exp_zeros : std_logic_vector(7 downto 0) := "00000000";
  constant e_in_z : unsigned(7 downto 0) := "00000000";
  constant exp_ones : std_logic_vector(7 downto 0) := "11111111";
  constant mant_zeros : std_logic_vector(14 downto 0) :=
```

```
"000000000000000";
  constant m_z_16 : unsigned(15 downto 0) := "0000000000000000";
  constant one : unsigned(14 downto 0) := "000000000000001";
  constant mant_nan : std_logic_vector(14 downto 0) :=
"010000000000000";
  constant won : unsigned(16 downto 0) := "00000000000000001";
  constant m_in_z : std_logic_vector(16 downto 0) :=
"00000000000000000";
  constant zero : std_logic_vector(18 downto 0) :=
"0000000000000000000";
  constant mant_en : unsigned(18 downto 0) := "1111111111111111111";
  constant temp1 : integer range 14 to 14 := 14;
  constant temp2 : integer range 15 to 15 := 15;
  constant r0: resource := 0;
  attribute map_to_module of r0: constant is "DW01_sub";
  attribute implementation of r0: constant is "cla";
  attribute ops of r0: constant is "sub1";

BEGIN
  fadd_pre : PROCESS (clk, resetn)
    variable a_inn, b_inn : nfloat;
    variable a1_sign, b1_sign : std_logic;
    variable a1_exp  : std_logic_vector(7 downto 0);    -- sjo 090699
,  b1_exp
    variable a1_mant, b1_mant : std_logic_vector(16 downto 0);
    variable diff, diffb : unsigned(7 downto 0);
    variable op_inn : std_logic;
    variable a_uendelig,a_nan,a_null : std_logic;
    variable b_uendelig,b_nan,b_null : std_logic;
    variable swp : std_logic;

  BEGIN
    if (resetn  = '0') then
      p0_a1m <= (others => '0');
      p0_a1e <= (others => '0');
      p0_a1s <= '0';
      p0_b1 <= (others => '0');
      p0_diff <= (others => '0');
      p0_b_sign <= '0';
      p0_op_inn <= '0';
      p0_swp <= '0';
      p0_a_uendelig <= '0';
      p0_b_uendelig <= '0';
      p0_a_nan <= '0';
      p0_b_nan <= '0';
    elsif (clk='1' and clk'event) then
--hold
      if (hold = '0') then
     a_inn := a;
     b_inn := b;
     op_inn := op;
      --0.  sjekk på gyldige tall
     a_uendelig := '0';
     a_nan := '0';
```

```
    a_null := '0';
    b_uendelig := '0';
    b_nan := '0';
    b_null := '0';
    swp := '0';

    if (a_inn.exp = exp_ones) then --NaN or +/-oo
      if (a_inn.mant = mant_zeros) then --+/-oo
        a_uendelig := '1';
      else--NaN
        a_nan := '1';
      end if;
    end if;
    if (a_inn.exp = exp_zeros) then --denorm eller 0
      a_null := '1';
    end if;

    if (b_inn.exp = exp_ones) then --NaN or +/-oo
      if (b_inn.mant = mant_zeros) then --+/-oo
        b_uendelig := '1';
      else--NaN
        b_nan := '1';
      end if;
    end if;
    if (b_inn.exp = exp_zeros) then --denorm eller 0
      b_null := '1';
    end if;

     --1.& 1b  sammenligne exp og mant swap minste tall i b
     -- inn  a, b
     -- ut  a1, b1
    diff  := unsigned(a_inn.exp) - unsigned(b_inn.exp);--param
label sub1
    diffb := unsigned(b_inn.exp) - unsigned(a_inn.exp);--param
label sub1
    a1_sign := a_inn.sign;
    a1_exp  := a_inn.exp ;
    a1_mant := "01"&a_inn.mant;
    if (a_null = '1' or b_uendelig = '1') then
        a1_mant := m_in_z;
        diff := e_in_z;
    end if;
    b1_sign := b_inn.sign;
    b1_mant := "01"&b_inn.mant;
    if (b_null = '1' or
        a_uendelig = '1'  ) then
      b1_mant := m_in_z;
      diff := e_in_z;
    end if;
-- b.exp > a.exp
        if (b_inn.exp > a_inn.exp) then
      diff := diffb;
      a1_sign := b_inn.sign;
      a1_exp  := b_inn.exp ;
```

```
        a1_mant := "01"&b_inn.mant;
        if ( b_null = '1' or a_uendelig = '1') then
          a1_mant := m_in_z;
          diff := e_in_z;
        end if;
        b1_sign := a_inn.sign;
        b1_mant := "01"&a_inn.mant;
        if ( a_null = '1' or
             b_uendelig = '1'  ) then
          b1_mant := m_in_z;
          diff := e_in_z;
        end if;
        swp := '1';
      end if;
--a.exp==b.exp b.mant > a.mant
        if (b_inn.exp = a_inn.exp) then
        if (b_inn.mant > a_inn.mant) then
          diff := diffb;
          a1_sign := b_inn.sign;
          a1_exp  := b_inn.exp ;
          a1_mant := "01"&b_inn.mant;
          if (b_null = '1' or a_uendelig = '1') then
            a1_mant := m_in_z;
            diff := e_in_z;
          end if;
          b1_sign := a_inn.sign;
          b1_mant := "01"&a_inn.mant;
          if ( a_null = '1' or
            b_uendelig = '1'  ) then
            b1_mant := m_in_z;
            diff := e_in_z;
          end if;
          swp := '1';
        end if;
         end if;

      if (a_nan = '1' and b_nan = '1') then
        a1_sign := a_inn.sign;
        a1_exp  := a_inn.exp;
        a1_mant := "01"&a_inn.mant;
        b1_sign := a_inn.sign;
        b1_mant := m_in_z;
        diff := e_in_z;
      end if;
      p0_a1s <= a1_sign;
      p0_a1e  <= a1_exp;
      p0_a1m <= a1_mant;
      p0_b1 <= b1_mant;
      p0_diff <= diff;
      p0_b_sign <= b1_sign;
      p0_op_inn <= op_inn;
      p0_swp <= swp;
      p0_a_uendelig <= a_uendelig;
      p0_b_uendelig <= b_uendelig;
```

```
      p0_a_nan <= a_nan;
      p0_b_nan <= b_nan;
       end if; --hold
     end if;    -- clk
  END PROCESS fadd_pre;

--PIPE_1

  fadd_mid : PROCESS(clk, resetn)
    variable a1_s : std_logic;
    variable a1_e : std_logic_vector(7 downto 0);
    variable a1_m : std_logic_vector(16 downto 0);
    variable b1_m : std_logic_vector(16 downto 0);
    variable b2 : unsigned(16 downto 0);
    variable diff : unsigned(7 downto 0);
    variable b4 : std_logic_vector(18 downto 0);
    variable op_inn,g,r,s,swp : std_logic;
    variable res1 : unsigned(16 downto 0);
    variable a_uendelig,a_nan : std_logic;
    variable b_uendelig,b_nan : std_logic;
    variable b_sign, op_diff : std_logic;

  BEGIN
    IF (resetn = '0') THEN
      p1_sign <= '0';
      p1_exp <= (others => '0');
      p1_res1 <= (others => '0');
      p1_op_diff <= '0';
      p1_op_inn <= '0';
      p1_swp <= '0';
      p1_g <= '0';
      p1_r <= '0';
      p1_s <= '0';
      p1_a_uendelig <= '0';
      p1_b_uendelig <= '0';
      p1_a_nan <= '0';
      p1_b_nan <= '0';
    ELSIF (clk='1' and clk'event) THEN
      IF (hold = '0') THEN
     a1_s := p0_a1s;
     a1_e := p0_a1e;
     a1_m := p0_a1m;
     b1_m := p0_b1;
     diff := p0_diff;
     b_sign := p0_b_sign;
     op_inn := p0_op_inn;
     swp := p0_swp;
     a_uendelig := p0_a_uendelig;
     b_uendelig := p0_b_uendelig;
     a_nan := p0_a_nan;
     b_nan := p0_b_nan;
     op_diff := '0';
      --2.  sjekke på fortegn
      --2b. 2erkomplement av b
```

```
      b2 := unsigned(b1_m);
      if((a1_s xor b_sign xor op_inn) = '1') then
        b2 := (unsigned((not b1_m)) + one);
      end if;
       --3.  shifter b + rgs
      op_diff := a1_s xor b_sign xor op_inn;
       -- inn b2, a1.exo og b1.exp
       -- ut  b4, g, r, s
      b4 := zero;
      g:='0';
      r:='0';
      s:='0';
      if (conv_integer(diff) < 18) then
        b4 := std_logic_vector(shr(b2&"00",diff));
        if((a1_s xor b_sign xor op_inn) = '1') then--shift inn enere
          b4 := std_logic_vector(shr(mant_en&b2&"00",diff)(18 downto
0));
        end if;
        g := b4(1);
        r := b4(0);
      end if;
      if (conv_integer(diff) > 2) then
        for i in 0 to manhi loop
          if (conv_integer(diff)-2 > i) then
            s := s or b2(i);
          end if;
        end loop;
      end if;

       --4.  addisjonen av a+b
       -- inn a1.mant og b4
       -- ut res1
      res1 := unsigned(a1_m) + unsigned(b4(18 downto 2));

      p1_sign <= a1_s;
      p1_exp <= a1_e;
      p1_res1 <= res1;
      p1_op_diff <= op_diff;
      p1_op_inn <= op_inn;
      p1_swp <= swp;
      p1_g <= g;
      p1_r <= r;
      p1_s <= s;
      p1_a_uendelig <= a_uendelig;
      p1_b_uendelig <= b_uendelig;
      p1_a_nan <= a_nan;
      p1_b_nan <= b_nan;
       end if;--hold
    end if;--clk
  end process fadd_mid;

--PIPE_2
```

```
  fadd_post : PROCESS(clk, resetn)
    variable add_res: nfloat;
    variable g,r,s,ii : std_logic;
    variable res1 : unsigned(16 downto 0);
    variable res2 : unsigned(17 downto 0);
    variable expadd : signed(7 downto 0);
    variable tmp_exp0, tmp_exp1 : signed(8 downto 0);
    variable res3 : unsigned(16 downto 0);
    variable a_uendelig,a_nan : std_logic;
    variable b_uendelig,b_nan : std_logic;
    variable res_uendelig, res_null, res_denorm : std_logic;
    variable a1_sign : std_logic;   -- sjo 090699 , b1_sign
    variable a1_exp : std_logic_vector(7 downto 0);
    variable lopf, pxpof, expcor0 : std_logic;      -- sjo 090699  ,
expcor1
    variable op_diff, ops, op_inn, swp : std_logic;

  BEGIN
    IF (resetn = '0') THEN
      res.sign <= '0';
      res.exp  <= (OTHERS => '0');
      res.mant <= (OTHERS => '0');
      ofl <= '0';
      ufl <= '0';
      ivo <= '0';
    ELSIF (clk='1' and clk'event) THEN
      IF (hold = '0') THEN
     add_res.sign := '0';
     add_res.exp  := (OTHERS => '0');
     add_res.mant := (OTHERS => '0');
     a1_sign := p1_sign;
     a1_exp := p1_exp;
     res1 := p1_res1;
     op_diff := p1_op_diff;
     op_inn := p1_op_inn;
     swp := p1_swp;
     g := p1_g;
     r := p1_r;
     s := p1_s;
     a_uendelig := p1_a_uendelig;
     b_uendelig := p1_b_uendelig;
     a_nan := p1_a_nan;
     b_nan := p1_b_nan;

      --5.&6.  normalisering og justering av r s
      -- inn res1
      -- ut  res2
     ii := '0';
     lopf := '0';
     pxpof := '0';
     expcor0 := '0';
     expadd := "00000000";
     res2 := res1&"0";
     for i in res1'high downto 0 loop
```

```
        if (res1(i)= '1' and ii = '0') then
          if (i = res1'high) then -- i.e. one right shit
            s := g or r or s;
            r := res1(0);
            expadd := "00000001";
            res2 := shr(res1&"0","1");
            lopf := '1';
            pxpof := '1';
          end if;
          if (i = res1'high-1) then -- i.e. no shit
            s := r or s;
            r := g;
            expadd := "00000000";
            lopf := '1';
          end if;
          if (i = temp1) then -- i.e. single left shift
            expadd := "11111111";
            res2 := shl(res1&conv_unsigned(g,1),"1");
            lopf := '1';
          end if;
          if (i < temp1) then -- i.e. two or more left shits
            s := '0';
            r := '0';
            expadd := conv_signed((i-temp2),8);

res2:=shl(res1&conv_unsigned(g,1),conv_unsigned(temp2-i,5));
            lopf := '1';
          end if;
          ii := '1';
        end if;
        if (ii = '0' and i = 0 and g = '1') then
          res2 := "0100000000000000";
          expadd := "11110000";
          g := '0';
          r := '0';
          s := '0';
          lopf := '1';
        end if;
      end loop;

      tmp_exp0 := signed('0'&a1_exp) + expadd;
      tmp_exp1 := signed('0'&a1_exp) + expadd + xpone;
      expcor0 := tmp_exp0(8);


       --7.  avrunding
      res_denorm := '0';
      res3 := "0"&res2(16 downto 1);
      if (r = '1')  then
        if (s = '1' or res2(1) = '1') then
          res3 := "0"&res2(16 downto 1) + won;
        end if;
      end if;
      add_res.mant := std_logic_vector(res3(14 downto 0));
```

```
      add_res.exp := std_logic_vector(tmp_exp0(7 downto 0));
      if (expcor0 = '1') then
        add_res.exp := exp_zeros;
        res_denorm := '1';
          if (pxpof = '1') then
            add_res.exp := exp_ones;
            res_denorm := '0';
          end if;
      end if;
      if (res3(res3'high) = '1') then
        add_res.exp := std_logic_vector(tmp_exp1(7 downto 0));
        add_res.mant := std_logic_vector(res3(15 downto 1));
        if (expcor0 = '1') then
          res_denorm := '1';
          add_res.exp := exp_zeros;
          if (pxpof = '1') then
            add_res.exp := exp_ones;
            res_denorm := '0';
          end if;
        end if;
      end if;
      if lopf = '0' then
        add_res.exp := exp_zeros;
      end if;

       --7b. exp justering
       --8.  sign
      add_res.sign := a1_sign;
      if (swp = '1' and op_inn = '1') then
--        res.sign <= op_inn xor a1_sign; -- A - B == (-B) + A
          add_res.sign := not a1_sign; -- A - B == (-B) + A
      end if;
       --9.  gyldig tall ut
      res_uendelig := '0';
      res_null     := '0';

      if ((add_res.exp = exp_ones) and
          (a_nan = '0' and b_nan = '0')) then --NaN or +/-oo
          res_uendelig := '1';
      end if;
      if (add_res.exp = exp_zeros) then --denorm eller 0
        if (res3(15 downto 0) = m_z_16 and res_denorm  = '0') then
          res_null := '1';
        else
          res_denorm := '1';
        end if;
      end if;

--tilordning av utganger
      ops := '0';
      res <= add_res;
      ufl <= '0';
      ofl <= '0';
      ivo <= '0';
```

```
      if (a_nan = '1' or b_nan = '1') then
        res.exp  <= exp_ones;
        res.sign <= add_res.sign;
        res.mant <= mant_nan;
        ops := '1';
      end if;
      if (ops = '0' and (a_uendelig = '1' and
                  b_uendelig = '1' and
                  op_diff = '1' )) then
        res.sign <= '0';
        res.exp  <= exp_ones;
        res.mant <= mant_nan;
        ivo <= '1';
        ops := '1';
        end if;
      if (ops = '0' and (a_uendelig = '1' or
                  b_uendelig = '1')) then
        res.exp  <= exp_ones;
        res.sign <= add_res.sign;
        res.mant <= mant_zeros;
        ops := '1';
      end if;
      if (ops = '0' and res_uendelig = '1') then
        res.exp  <= exp_ones;
        res.sign <= add_res.sign;
        res.mant <= mant_zeros;
        ofl <= '1';
        ops := '1';
      end if;
      if (ops = '0' and res_null = '1') then
        res.exp  <= exp_zeros;
        res.sign <= '0';
        res.mant <= mant_zeros;
        ops := '1';
      end if;
      if (ops = '0' and res_denorm = '1') then
        res.exp  <= exp_zeros;
        res.sign <= '0';
        res.mant <= mant_zeros;
        ufl <= '1';
        ops := '1';
      end if;
       end if; --hold
     end if;    -- clk
  END PROCESS fadd_post;
END pipe3;
```

A.2 VHDL-kode for egendefinert bibliotek: types.vhd

Her følger VHDL-koden for det egenedfinerte VHDL-biblioteket.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;
-- USE ieee.math_real.ALL;


PACKAGE types IS

  CONSTANT mansiz  : integer := 16;        -- MSB = hidden bit!
  CONSTANT expsiz  : integer := 8;

  CONSTANT fflsiz  : integer := mansiz + expsiz + 1;
  CONSTANT nflsiz  : integer := mansiz-1 + expsiz + 1;
  CONSTANT fcpxsiz : integer := fflsiz * 2;
  CONSTANT ncpxsiz : integer := nflsiz * 2;
  CONSTANT intsiz  : integer := 24;

  CONSTANT exphi   : integer := expsiz-1;
  CONSTANT manhi   : integer := mansiz-1;

  CONSTANT fflhi   : integer := fflsiz-1;
  CONSTANT nflhi   : integer := nflsiz-1;
  CONSTANT fcpxhi  : integer := fcpxsiz-1;
  CONSTANT ncpxhi  : integer := ncpxsiz-1;
  CONSTANT inthi   : integer := intsiz-1;

  TYPE ffloat IS RECORD
    sign : std_logic;
    exp  : std_logic_vector (exphi downto 0);
    mant : std_logic_vector (manhi downto 0);
  END RECORD;

  TYPE nfloat IS RECORD
    sign : std_logic;
    exp  : std_logic_vector (exphi downto 0);
    mant : std_logic_vector (manhi-1 downto 0);
  END RECORD;

  TYPE fcomplex IS RECORD
    re : ffloat;
    im : ffloat;
  END RECORD;

  TYPE ncomplex IS RECORD
    re : nfloat;
    im : nfloat;
  END RECORD;

  CONSTANT I_SIZE  : integer := 21;       -- 8;
```

```
  SUBTYPE i_type  IS std_logic_vector (I_SIZE downto 0);
  SUBTYPE b8_type IS std_logic_vector (7 downto 0);
  SUBTYPE b4_type IS std_logic_vector (3 downto 0);
  SUBTYPE b3_type IS std_logic_vector (2 downto 0);
  SUBTYPE b2_type IS std_logic_vector (1 downto 0);

  CONSTANT I_ZERO : i_type := (OTHERS => '0');


  -- functions for converting between ffloat and nfloat
  FUNCTION conv_float (val : ffloat) RETURN nfloat;
  FUNCTION conv_float (val : nfloat) RETURN ffloat;

  -- functions for converting between fcomplex and ncomplex
  FUNCTION conv_complex (val : fcomplex) RETURN ncomplex;
  FUNCTION conv_complex (val : ncomplex) RETURN fcomplex;

  -- functions for converting between float formats and
std_logic_vector
  FUNCTION float2lv  (val : ffloat) RETURN std_logic_vector;
  FUNCTION float2lv  (val : nfloat) RETURN std_logic_vector;
  FUNCTION lv2ffloat (lv  : std_logic_vector(fflhi DOWNTO 0)) RETURN
ffloat;
  FUNCTION lv2nfloat (lv  : std_logic_vector(nflhi DOWNTO 0)) RETURN
nfloat;

  -- functions for converting between complex formats and
std_logic_vector
  FUNCTION cpx2lv  (val : fcomplex) RETURN std_logic_vector;
  FUNCTION cpx2lv  (val : ncomplex) RETURN std_logic_vector;
  FUNCTION lv2fcpx (lv  : std_logic_vector(fcpxhi DOWNTO 0)) RETURN
fcomplex;
  FUNCTION lv2ncpx (lv  : std_logic_vector(ncpxhi DOWNTO 0)) RETURN
ncomplex;

--pragma translate_off

  -- functions for converting between float formats and real
  FUNCTION nfloat2real (val : nfloat) RETURN real;
  FUNCTION ffloat2real (val : ffloat) RETURN real;
--  FUNCTION real2nfloat (val : real) RETURN nfloat;
--  FUNCTION real2ffloat (val : real) RETURN ffloat;

--pragma translate_on

END types;


PACKAGE BODY types IS

  -- function for converting from ffloat to nfloat
  FUNCTION conv_float (val : ffloat) RETURN nfloat IS
```

```
  VARIABLE ret_val : nfloat;
BEGIN
  ret_val.sign := val.sign;
  ret_val.exp  := val.exp;
  ret_val.mant := val.mant(val.mant'left-1 DOWNTO 0);

  RETURN ret_val;
END;


-- function for converting from nfloat to ffloat
FUNCTION conv_float (val : nfloat) RETURN ffloat IS
  VARIABLE ret_val : ffloat;
BEGIN
  ret_val.sign := val.sign;
  ret_val.exp  := val.exp;
  ret_val.mant := '1' & val.mant;

  RETURN ret_val;
END;


-- function for converting from fcomplex to ncomplex
FUNCTION conv_complex (val : fcomplex) RETURN ncomplex IS
  VARIABLE ret_val : ncomplex;
BEGIN
  ret_val.re := conv_float(val.re);
  ret_val.im := conv_float(val.im);

  RETURN ret_val;
END;


-- function for converting from ncomplex to fcomplex
FUNCTION conv_complex (val : ncomplex) RETURN fcomplex IS
  VARIABLE ret_val : fcomplex;
BEGIN
  ret_val.re := conv_float(val.re);
  ret_val.im := conv_float(val.im);

  RETURN ret_val;
END;


-- function for converting from ffloat to std_logic_vector
FUNCTION float2lv (val : ffloat) RETURN std_logic_vector IS
BEGIN
  RETURN val.sign & val.exp & val.mant;
END;


-- function for converting from nfloat to std_logic_vector
FUNCTION float2lv (val : nfloat) RETURN std_logic_vector IS
BEGIN
```

```
    RETURN val.sign & val.exp & val.mant;
  END;


  -- function for converting from std_logic_vector to ffloat
  FUNCTION lv2ffloat (lv : std_logic_vector(fflhi DOWNTO 0)) RETURN
ffloat IS

    CONSTANT MANT_LEFT : integer := manhi;
    CONSTANT EXP_RIGHT : integer := MANT_LEFT + 1;
    CONSTANT EXP_LEFT  : integer := MANT_LEFT + expsiz;
    CONSTANT SIGN      : integer := EXP_LEFT + 1;

    VARIABLE ret_val   : ffloat;

  BEGIN

    ret_val.sign := lv(SIGN);
    ret_val.exp  := lv(EXP_LEFT DOWNTO EXP_RIGHT);
    ret_val.mant := lv(MANT_LEFT DOWNTO 0);

    RETURN ret_val;

  END;


  -- function for converting from std_logic_vector to nfloat
  FUNCTION lv2nfloat (lv : std_logic_vector(nflhi DOWNTO 0)) RETURN
nfloat IS

    CONSTANT MANT_LEFT : integer := manhi-1;
    CONSTANT EXP_RIGHT : integer := MANT_LEFT + 1;
    CONSTANT EXP_LEFT  : integer := MANT_LEFT + expsiz;
    CONSTANT SIGN      : integer := EXP_LEFT + 1;

    VARIABLE ret_val   : nfloat;

  BEGIN

    ret_val.sign := lv(SIGN);
    ret_val.exp  := lv(EXP_LEFT DOWNTO EXP_RIGHT);
    ret_val.mant := lv(MANT_LEFT DOWNTO 0);

    RETURN ret_val;

  END;


  -- function for converting from fcomplex to std_logic_vector
  FUNCTION cpx2lv (val : fcomplex) RETURN std_logic_vector IS
  BEGIN
    RETURN float2lv(val.re) & float2lv(val.im);
  END;
```

```
  -- function for converting from ncomplex to std_logic_vector
  FUNCTION cpx2lv (val : ncomplex) RETURN std_logic_vector IS
  BEGIN
    RETURN float2lv(val.re) & float2lv(val.im);
  END;


  -- function for converting from std_logic_vector to fcomplex
  FUNCTION lv2fcpx (lv : std_logic_vector(fcpxhi DOWNTO 0)) RETURN
fcomplex IS

    VARIABLE tmp_re,tmp_im : std_logic_vector(fflhi DOWNTO 0);
    VARIABLE ret_val       : fcomplex;

  BEGIN
    tmp_re := lv(lv'left DOWNTO fflhi+1);
    tmp_im := lv(tmp_im'RANGE);

    ret_val.re := lv2ffloat(tmp_re);
    ret_val.im := lv2ffloat(tmp_im);

    RETURN ret_val;
  END;


  -- function for converting from std_logic_vector to ncomplex
  FUNCTION lv2ncpx (lv : std_logic_vector(ncpxhi DOWNTO 0)) RETURN
ncomplex IS

    VARIABLE tmp_re,tmp_im : std_logic_vector(nflhi DOWNTO 0);
    VARIABLE ret_val       : ncomplex;

  BEGIN
    tmp_re := lv(lv'left DOWNTO nflhi+1);
    tmp_im := lv(tmp_im'RANGE);

    ret_val.re := lv2nfloat(tmp_re);
    ret_val.im := lv2nfloat(tmp_im);

    RETURN ret_val;
  END;

--pragma translate_off

  -- function for converting from nfloat to real
  FUNCTION nfloat2real (val : nfloat) RETURN real IS
    VARIABLE ret_val : real;
    VARIABLE mant    : real;
    VARIABLE exp     : integer;
  BEGIN
    exp  := conv_integer(val.exp) - 127;
    mant := real(conv_integer(val.mant))/(2.0**manhi) + 1.0;
    ret_val := mant * (2.0**exp);
```

```
    IF (val.sign = '1') THEN
      ret_val := -ret_val;
    END IF;

    RETURN ret_val;
  END;


  -- function for converting from ffloat to real
  FUNCTION ffloat2real (val : ffloat) RETURN real IS
  BEGIN
    RETURN nfloat2real(conv_float(val));
  END;


  -- function for converting from real to nfloat
--  FUNCTION real2nfloat (val : real) RETURN nfloat IS
--    VARIABLE ret_val : nfloat;
--    VARIABLE val_tmp : real;
--    VARIABLE exp     : integer;
--    VARIABLE mant    : integer;
--  BEGIN
--
--    ret_val := ('0',(OTHERS => '0'),(OTHERS => '0'));
--    IF (val = 0.0)
--    THEN
--      RETURN ret_val;
--    END IF;
--
--    val_tmp := val;
--    IF (val_tmp < 0.0) THEN
--      ret_val.sign := '1';
--      val_tmp := -val_tmp;
--    END IF;
--
--    exp := 0;
--    IF (val_tmp >= 2.0) THEN
--      exp := integer(floor((log(val_tmp)/math_log_of_2)));
--    ELSIF (val_tmp < 1.0) THEN
--      exp := -integer(ceil((-log(val_tmp)/math_log_of_2)));
--    END IF;
--
--    val_tmp := (val_tmp/2.0**exp)-1.0;
--
--    exp := exp + 127;
--    ret_val.exp  := conv_std_logic_vector(exp,expsiz);
--
--    mant := integer(round(val_tmp * 2.0**manhi));
--    ret_val.mant := conv_std_logic_vector(mant,manhi);
--
--    RETURN ret_val;
--  END;
```

```
  -- function for converting from real to ffloat
--  FUNCTION real2ffloat (val : real) RETURN ffloat IS
--  BEGIN
--    RETURN conv_float(real2nfloat(val));
--  END;

--pragma translate_on


END types;
```

B VHDL-KODE FOR TESTBENKEN OG TESTBENKINTERFACE

Her følger koden for testbenken som ble brukt til å teste og verifisere addisjonskretsen.

B.1 VHDL-kode for testbenkens C-interface

```
library IEEE;
use IEEE.std_logic_1164.all;

entity facitgen is
        port(ina: in std_logic_vector(23 downto 0);
             inb: in std_logic_vector(23 downto 0);
             op : in std_logic;
             res: out std_logic_vector(23 downto 0) );
end facitgen;

architecture c_connect of facitgen is
        ATTRIBUTE foreign: string;
        ATTRIBUTE foreign of c_connect: architecture is
         " fpadd_init__FPvPcP18interface_list_tagT3
../CCinterface/facitgen.sl;
";
begin
end c_connect;
```

B.2 VHDL-kode for testbenken

```
-------------------------------------------------------------------
-------
--FPADD TESTBENCH
-------------------------------------------------------------------
-------
LIBRARY WORK;
USE WORK.ALL;
```

```
LIBRARY STD;
USE STD.TEXTIO.ALL;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;

LIBRARY common_lib;
USE common_lib.types.ALL;

--LIBRARY addlib;
--USE addlib.ALL;
LIBRARY fpadd_lib;
USE fpadd_lib.ALL;

ENTITY testbench IS
END testbench;


ARCHITECTURE testbench OF testbench  IS

constant inp_expsize    : integer := 8;   -- Size of input exponent
constant outp_expsize   : integer := 8;   -- Size of output exponent
constant inp_wordsize   : integer := 24;  -- Size of input_word
constant outp_wordsize  : integer := 24;  -- Size of output_word

constant inp_mantsize   : integer := (inp_wordsize-inp_expsize) - 1;
 -- Size o
f input mantissa
constant outp_mantsize  : integer := (outp_wordsize-outp_expsize) -
1; -- Size o
f output mantissa

constant NAN : std_logic_vector(outp_wordsize-2 downto 0) :=
"111111110100000000
00000";

constant dutpipelen: integer := 3; -- number of pipelinedelays in
DUT.

-------------------------------------------------------------------
--------
--  Her setter vi konstanter for pattern-generatoren
--
-------------------------------------------------------------------
--------
    constant number_of_sets : integer := 4;
    TYPE params IS ARRAY (0 TO (number_of_sets-1)) OF integer;

    constant exp1_start  : params := (   100,     0,   105,     0);
    constant exp1_inc1   : params := (    -1,     1,    -1,    -1);
    constant exp1_stop1  : params := (    -1,     2,    -1,    -1);
    constant exp1_start2 : params := (    -1,   127,    -1,    -1);
```

```
    constant exp1_inc2   : params := (   -1,    1,   -1,   -1);
    constant exp1_stop2  : params := (   -1,  128,   -1,   -1);
    constant exp1_start3 : params := (   -1,  253,   -1,   -1);
    constant exp1_inc3   : params := (    1,    1,    1,    1);
    constant exp1_stop   : params := (  100,  255,  149,  255);

    constant exp2_start  : params := (  100,    0,  105,    0);
    constant exp2_inc1   : params := (   -1,    1,   -1,   -1);
    constant exp2_stop1  : params := (   -1,    2,   -1,   -1);
    constant exp2_start2 : params := (   -1,  127,   -1,   -1);
    constant exp2_inc2   : params := (   -1,    1,   -1,   -1);
    constant exp2_stop2  : params := (   -1,  128,   -1,   -1);
    constant exp2_start3 : params := (   -1,  253,   -1,   -1);
    constant exp2_inc3   : params := (    1,    1,    1,    1);
    constant exp2_stop   : params := (  100,  255,  149,  255);

    constant mant1_start : params := (    0,    0,    0,    0);
    constant mant1_inc   : params := (    1,    1,    1,    1);
    constant mant1_stop  : params := (   60,   60,    4,    2);
    constant mant1_type  : params := (    2,    2,    1,    1);
    constant mant2_start : params := (    0,    0,    0,    2);
    constant mant2_inc   : params := (    1,    1,    1,    1);
    constant mant2_stop  : params := (   60,   60,    4,    4);
    constant mant2_type  : params := (    2,    2,    1,    1);


COMPONENT fpadd
   PORT(
        a        : IN  nfloat;
        b        : IN  nfloat;
        op       : IN  std_logic;
        clk      : IN  std_logic;
        resetn   : IN  std_logic;
        hold     : IN  std_logic;
        res      : OUT nfloat;
        ufl      : OUT std_logic;
        ofl      : OUT std_logic;
        ivo      : OUT std_logic


   );
END COMPONENT;
for all : fpadd use entity fpadd_lib.fpadder(pipe3);

COMPONENT facitgen
   PORT(
        ina : IN  std_logic_vector(inp_wordsize-1 downto 0);    --
argument a
        inb : IN  std_logic_vector(inp_wordsize-1 downto 0);    --
argument b
        op  : IN  std_logic ;
        res : OUT std_logic_vector(outp_wordsize-1 downto 0)    --
final result
       );
```

```
END COMPONENT;
for all : facitgen use entity work.facitgen(c_connect);

signal clk : std_logic;

TYPE slv_vector IS ARRAY (0 TO 5) OF std_logic_vector(inp_wordsize-1
downto 0);
TYPE sl_vector IS ARRAY (0 TO 5) OF std_logic;
signal facina, facinb : slv_vector;
signal facop : sl_vector;
signal facres : std_logic_vector(outp_wordsize-1 downto 0);
signal dutina, dutinb, dutres: nfloat;
signal dut_op :  std_logic;
signal resetn, hold, dut_ufl, dut_ofl, dut_ivo : std_logic;
signal dataerror     : boolean := false;
signal datafinished  : boolean := false;
signal going  : boolean := true;
TYPE bool_vector IS ARRAY (0 TO 5) OF boolean;
signal finishdly  : bool_vector;



function int2sl(inint : integer) return std_logic is
  variable half : integer;
begin
  half := inint/2;
  if (half*2 = inint) then
    return '0';
  else
    return '1';
  end if;
end int2sl;

function int2slv(inint : integer; outbits : integer) return
std_logic_vector is
  variable slv : std_logic_vector(outbits-1 downto 0);
  variable current, last, i : integer;
begin
  current := inint;
  converting:  FOR i IN 0 to outbits-1 LOOP
    slv(i) := int2sl(current);
    current := current/2;
  END LOOP converting;
  return slv;
end int2slv;



BEGIN

  DUT : fpadd
    PORT MAP(
              a       => dutina,
              b       => dutinb,
```

```
            op      => dut_op,
            clk     => clk,
            resetn  => resetn,
            hold    => hold,
            res     => dutres,
            ufl     => dut_ufl,
            ofl     => dut_ofl,
            ivo     => dut_ivo

            );


  FACGEN : facitgen
    PORT MAP(
            ina  => facina(dutpipelen-1),
            inb  => facinb(dutpipelen-1),
            op   => facop(dutpipelen-1),
            res  => facres
            );
```

```
-------------------------------------------------------------------
-------
  --  Process: clock
  --  Purpose: clock generation process with 16, 32  cycle period,
  --           8/16 ns low offset and 8/16 ns high time

-------------------------------------------------------------------
-------


      clock : PROCESS

      BEGIN  --  PROCESS clock

        clk <= '1';
        WAIT for 7 ns;
        clk <= '0';
        WAIT for 8 ns;

      END PROCESS clock;

-------------------------------------------------------------------
-------
--  Process: pM-etrykk
--  Purpose: read operations from file and perform read or write
access
-------------------------------------------------------------------
-------
  pattern_gen : PROCESS
```

```
    variable op                                 : integer; --
operasjon +/-

    variable Sign1, Sign2                       : std_logic;
    variable Exponent1, Exponent2               :
std_logic_vector(inp_expsize-1 d
ownto 0);
    variable Mantisse1, Mantisse2               :
std_logic_vector(inp_mantsize-1
downto 0);

    variable s1, s2, exp1, exp2                 : integer;
    variable mant1, mant2, set_number           : integer;
    variable signbit1                           : boolean;


    function gen_mant(mant_type, i : integer) return
std_logic_vector is

      variable mantisse : std_logic_vector( inp_mantsize-1 downto 0
);

    begin
      case mant_type is
        when 0 =>
          mantisse := int2slv(i, inp_mantsize);
        when 1 =>
          case i is
            when 0 =>
              mantisse := (others => '0');                          --
all zeros
            when 1 =>
              mantisse := (others => '0');                          --
zeros, bit0 o
ne
              mantisse(0) := '1';
            when 2 =>
              mantisse := (others => '1');                          --
all ones
            when others =>
              mantisse := (others => '1');                          --
ones, bit0 ze
ro
              mantisse(0) := '0';
          end case;
        when 2 =>
          if (i < inp_mantsize) then
-- fence of
zeros
            mantisse := (others => '1');
            mantisse(i downto 0) := (others => '0');
          elsif (i < inp_mantsize*2 ) then
-- fence of
ones
```

```
            mantisse := (others => '0');
            mantisse( i-inp_mantsize downto 0) := (others => '1');
          elsif (i < inp_mantsize*3 ) then
-- walking o
ne
            mantisse := (others => '0');
            mantisse( i-(inp_mantsize*2) ) := '1';
          else                                                    --
walking zero
            mantisse := (others => '1');
            mantisse( i-(inp_mantsize*3) ) := '0';
          end if;
        when others =>
      end case;

      return mantisse;

    end gen_mant;

  BEGIN -- pattern_gen

    resetn <= '1';
    hold <= '0';

    set_number := 0;
    set_loop : while set_number < number_of_sets loop
      op := 0;
      Operation_loop : while op < 2 loop
        s1 := 0;
        Signbit1_loop : while s1 < 2 loop
          s2 := 0;
          Signbit2_loop : while s2 < 2 loop
            exp1 := exp1_start(set_number); -- Nederste loop
startverdi
            Exponent1_loop : while exp1 <= exp1_stop(set_number)
loop
              exp2 := exp2_start(set_number);  -- Nederste loop
startverdi
              Exponent2_loop : while exp2 <= exp2_stop(set_number)
loop
                mant1 := mant1_start(set_number);
                Mantisse1_loop : while mant1 <
mant1_stop(set_number) loop
                  mant2 := mant2_start(set_number);
                  Mantisse2_loop : while mant2 <
mant2_stop(set_number) loop

                    wait on clk until (clk = '1' and clk'last_value
= '0');
                    dut_op      <= int2sl(op);
                    dutina.sign <= int2sl(s1);
                    dutina.exp  <= int2slv(exp1, inp_expsize);
                    dutina.mant <= gen_mant(mant1_type(set_number),
mant1);
```

```
                    dutinb.sign <= int2sl(s2);
                    dutinb.exp  <= int2slv(exp2, inp_expsize);
                    dutinb.mant <= gen_mant(mant2_type(set_number),
mant2);
                    mant2 := mant2 + mant2_inc(set_number);
                  end loop Mantisse2_loop;

                  mant1 := mant1 + mant1_inc(set_number);
                end loop Mantisse1_loop;

                if exp2 < exp2_stop1(set_number) then
                  exp2 := exp2 + exp2_inc1(set_number);
                elsif exp2 = exp2_stop1(set_number) then
                  exp2 := exp2_start2(set_number); -- Midtre loop
startverdi
                elsif exp2 < exp2_stop2(set_number) then
                  exp2 := exp2 + exp2_inc2(set_number);
                elsif exp2 = exp2_stop2(set_number) then
                  exp2 := exp2_start3(set_number); -- M-Xverste loop
startverdi

                else
                  exp2 := exp2 + exp2_inc3(set_number);
                end if;
              end loop Exponent2_loop;

              if exp1 < exp1_stop1(set_number) then
                exp1 := exp1 + exp1_inc1(set_number);
              elsif exp1 = exp1_stop1(set_number) then
                exp1 := exp1_start2(set_number); -- Midtre loop
startverdi
              elsif exp1 < exp1_stop2(set_number) then
                exp1 := exp1 + exp1_inc2(set_number);
              elsif exp1 = exp1_stop2(set_number) then
                exp1 := exp1_start3(set_number); -- M-Xverste loop
startverdi
              else
                exp1 := exp1 + exp1_inc3(set_number);
              end if;
            end loop Exponent1_loop;

            s2 := s2 + 1;
          end loop Signbit2_loop;

          s1 := s1 + 1;
        end loop Signbit1_loop;
        op := op + 1;
      end loop Operation_loop;

      set_number := set_number+1;
    end loop set_loop;

    datafinished <= true;
    wait;
```

```
    END PROCESS pattern_gen;


-------------------------------------------------------------------
-------
     --  Process: delayline
     --  Purpose: Delayer data fra FACIT i h h t pipelinedelay i DUT
.
     --

-------------------------------------------------------------------
------
  delayline : process (clk)

  BEGIN
    IF (clk ='1' and clk'event ) THEN
      delayline:  FOR i IN 5 downto 1 LOOP
        facina(i) <= facina(i-1);
        facinb(i) <= facinb(i-1);
        facop(i) <= facop(i-1);
        finishdly(i) <= finishdly(i-1);
      END LOOP delayline;
      facina(0) <= dutina.sign&dutina.exp&dutina.mant;
      facinb(0) <= dutinb.sign&dutinb.exp&dutinb.mant;
      facop(0)  <= dut_op;
      finishdly(0) <= datafinished;
    END IF;

  END PROCESS delayline;




-------------------------------------------------------------------
-------
     --  Process: Check
     --  Purpose: Compare results from DUT and FACGEN.
     --           Store the results in "results.out"

-------------------------------------------------------------------
------
  check : process (clk)
    VARIABLE linebuffer, inline : line;
    FILE out_file    : TEXT IS OUT "results.out";

    variable out_data    : std_logic_vector(outp_wordsize-1 downto
0);
    variable shuttle     : string(1 to 2) := "  ";
    variable i, data_cnt, start_dly  : integer := 0;
    variable write_trig : unsigned(13 downto 0) := "00000000000000";
    variable tb_ofl, tb_ufl, tb_ivo : std_logic;
```

```
  BEGIN
    IF (clk ='1' and clk'event ) THEN

      if start_dly < (dutpipelen+2) then
        start_dly := start_dly + 1;
      else
        tb_ivo := '0';
        tb_ofl := '0';
        tb_ufl := '0';
        if (dutres.exp = "11111111") then
          if (dutres.mant = "00000000000000") then

              -- uendelig
            if (    (facina(dutpipelen-1)(22 downto 0) /=
"11111110000000000000
00")              -- ingen av inngangene uendelig
                and (facinb(dutpipelen-1)(22 downto 0) /=
"11111110000000000000
00")) then        -- ingen av inngangene uendelig
              tb_ofl := '1';
            end if;
          else

              --NaN
            if (    (   (facina(dutpipelen-1)(22 downto 15) /=
"11111111")
                -- ina ikke NaN eller uendelig
                   or (facina(dutpipelen-1)(22 downto 0) =
"11111110000000000
00000"))          -- ina = uendelig.      ie. ina ikke NaN
                and (   (facinb(dutpipelen-1)(22 downto 15) /=
"11111111")
                -- inb ikke NaN eller uendelig
                   or (facinb(dutpipelen-1)(22 downto 0) =
"11111110000000000
00000"))) then    -- inb = uendelig.      ie. inb ikke NaN
              tb_ivo := '1';
            end if;
          end if;
        end if;

        if ((dutres.exp = "00000000")
            and ((dutres.mant /= "00000000000000")

                -- denormalisert
              or ((    (facina(dutpipelen-1)(22 downto 15) /=
"00000000")
                -- eller ina eller inb forskjellig
                    or (facinb(dutpipelen-1)(22 downto 15) /=
"00000000"))
                --              fra 0 eller denorm
               and ((facina(dutpipelen-1)(22 downto 0) /=
facinb(dutpipelen-1)
```

```
(22 downto 0))     --     og ina /= -inb
                         or ((facop(dutpipelen-1) = '0')
                             and (facina(dutpipelen-1)(23) =
facinb(dutpipelen-1)(
23)))
                         or ((facop(dutpipelen) = '1')
                             and (facina(dutpipelen-1)(23) /=
facinb(dutpipelen-1)
(23))))))) then          --
            tb_ufl := '1';
        end if;

        out_data(outp_wordsize-1 downto 0):=
dutres.sign&dutres.exp&dutres.mant;
        if(      ( out_data /= facres
                   and (    out_data(outp_wordsize-2 downto 0) /= NAN
                        or facres(outp_wordsize-2 downto 0) /= NAN
))
           or dut_ofl /= tb_ofl or dut_ufl /= tb_ufl or dut_ivo /=
tb_ivo) then
          dataerror <= true;
          hwrite(linebuffer, "00"&std_logic_vector(write_trig));
          write(linebuffer, string'("  ina: "));
          hwrite(linebuffer, facina(dutpipelen-1));
          write(linebuffer, string'("  inb: "));
          hwrite(linebuffer, facinb(dutpipelen-1));
          write(linebuffer, string'("  actual: "));
          hwrite(linebuffer, out_data);
          write(linebuffer, string'("  facit: "));
          hwrite(linebuffer, facres);
          write(linebuffer, string'("  ofl: "));
          write(linebuffer, dut_ofl, right, 1);
          write(linebuffer, string'(" "));
          write(linebuffer, tb_ofl, right, 1);
          write(linebuffer, string'("  ufl: "));
          write(linebuffer, dut_ufl, right, 1);
          write(linebuffer, string'(" "));
          write(linebuffer, tb_ufl, right, 1);
          write(linebuffer, string'("  ivo: "));
          write(linebuffer, dut_ivo, right, 1);
          write(linebuffer, string'(" "));
          write(linebuffer, tb_ivo, right, 1);
          writeline(out_file, linebuffer );
        end if;
        write_trig := write_trig + 1;
        if (write_trig = 0 and not(finishdly(dutpipelen-1)))  then
          data_cnt := data_cnt + 1;
          write(linebuffer, string'("checked "));
          write(linebuffer, data_cnt*16);
          write(linebuffer, string'("k values "));
          writeline(out_file, linebuffer );
        end if;
      end if;
    END IF;
```

```
  END PROCESS check;


-------------------------------------------------------------------
-------
     --  Process: result_msg
     --  Prosessen trigges av at data er finished. Skriver sM-e ut
en melding
     --  om det er feil i datasjekkingen eller ikke.

-------------------------------------------------------------------
------
  result_msg : process
     VARIABLE message     : LINE;
     FILE messagefile     : TEXT IS OUT "msg";
     BEGIN
      wait until finishdly(dutpipelen-1);
        if dataerror then
          assert false report " Error(s) in data !!!!! " severity
NOTE;
          write(message, string'("  Error(s) in data !!!!!   "));
          writeline(messagefile, message);
        else
          assert false report " OK, no errors. " severity NOTE;
          write(message, string'("  OK, no errors.  Simulation run
for "));
          write(message, now);
          writeline(messagefile, message);
          writeline(messagefile, message);

          write(message, string'("Parameters used : "));
          writeline(messagefile, message);
          writeline(messagefile, message);

          write(message, string'("  Pipelinelength          : "));
          write(message, dutpipelen, right, 4);
          writeline(messagefile, message);
          writeline(messagefile, message);

          write(message, string'("  size of input exponent  : "));
          write(message, inp_expsize, right, 4);
          writeline(messagefile, message);

          write(message, string'("  size of input mantissa  : "));
          write(message, inp_mantsize, right, 4);
          writeline(messagefile, message);

          write(message, string'("  size of input word(s)   : "));
          write(message, inp_wordsize, right, 4);
          writeline(messagefile, message);
          writeline(messagefile, message);

          write(message, string'("  size of output exponent : "));
          write(message, outp_expsize, right, 4);
```

```
          writeline(messagefile, message);

          write(message, string'("  size of output mantissa : "));
          write(message, inp_mantsize, right, 4);
          writeline(messagefile, message);

          write(message, string'("  size of output word     : "));
          write(message, inp_wordsize, right, 4);
          writeline(messagefile, message);
          writeline(messagefile, message);

          write(message, string'("  exp1 start  value : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_start(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp1 inc1 value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_inc1(i), righ
t, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp1 stop1 value  : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_stop1(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp1 start2 value : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_start2(i), ri
ght, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp1 inc2 value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_inc2(i), righ
t, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp1 stop2 value  : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_stop2(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp1 start3 value : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_start3(i), ri
ght, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp1 inc3 value   : "));
```

```
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_inc3(i), righ
t, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp1 stop value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp1_stop(i), righ
t, 8); end loop;
          writeline(messagefile, message);
          writeline(messagefile, message);

          write(message, string'("  exp2 start  value : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_start(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp2 inc1 value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_inc1(i), righ
t, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp2 stop1 value  : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_stop1(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp2 start2 value : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_start2(i), ri
ght, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp2 inc2 value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_inc2(i), righ
t, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp2 stop2 value  : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_stop2(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp2 start3 value : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_start3(i), ri
ght, 8); end loop;
          writeline(messagefile, message);
```

```
          write(message, string'("  exp2 inc3 value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_inc3(i), righ
t, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  exp2 stop value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
exp2_stop(i), righ
t, 8); end loop;
          writeline(messagefile, message);
          writeline(messagefile, message);

          write(message, string'("  mant1 start value : "));
          for  i in 0 to number_of_sets-1 loop write(message,
mant1_start(i), ri
ght, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  mant1 inc value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
mant1_inc(i), righ
t, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  mant1 stop value  : "));
          for  i in 0 to number_of_sets-1 loop write(message,
mant1_stop(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  mant1 type        : "));
          for  i in 0 to number_of_sets-1 loop write(message,
mant1_type(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  mant2 start value : "));
          for  i in 0 to number_of_sets-1 loop write(message,
mant2_start(i), ri
ght, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  mant2 inc value   : "));
          for  i in 0 to number_of_sets-1 loop write(message,
mant2_inc(i), righ
t, 8); end loop;
          writeline(messagefile, message);

          write(message, string'("  mant2 stop value  : "));
          for  i in 0 to number_of_sets-1 loop write(message,
mant2_stop(i), rig
ht, 8); end loop;
          writeline(messagefile, message);
```

```
          write(message, string'("  mant2 type        : "));
          for  i in 0 to number_of_sets-1 loop write(message,
mant2_type(i), rig
ht, 8); end loop;
          writeline(messagefile, message);

       end if;
    END PROCESS result_msg;

END testbench;
```

C KODE FOR OPPSETT AV SYNOPSYS DESIGN COMPILER

Her følger scriptene som ble brukt til å konfigurere Synopsys Design Compiler for syntese.

C.1 Kode for oppsett av Synopsys: .synopsys_dc.setup.

```
designer = "Rune Gundersen";
company = "FFI/E";
plot_command = "lp -dbig -obin 3"

/* Avoid writing log files defined in global .synopsys_dc.setup */
command_log_file = "";
view_command_log_file = "";
filename_log_file = "" ;

/* Library and Search Path variables */

search_path = search_path + { ./db_files }
search_path = search_path + {
/raid2/home/cesar/asic/fft/design/syntese/memif/sa_design/memif/db }
search_path = search_path + {
/raid2/home/cesar/asic/fft/design/released/parts/ADS/generators/synt
hesis }
search_path = search_path + { /raid1/home/rgu/cmos035/v1.6/syn97.8 }

link_library               = "* MTC45000_WL.db MTC45000.db
GENERATORS.db";
/*synthetic_library        = "dw02.sldb dw01.sldb";*/
target_library             = "MTC45000_WL.db MTC45000.db";
symbol_library             = "MTC45000.sdb";
vhdlout_use_packages       = {"IEEE.std_logic_1164" \
                              "MTC45000.components"};

/*synlib_disable_limited_licenses = "false";*/

/* Disabling unwanted cells */
dont_use {MTC45000/FJK*};
/*dont_use {DW01/RPL*};*/

hdl_keep_licenses  = "true"
suppress_errors = {UIO-12 VHDL-2091 SEC-80 SEC-81}
```

```
/*compile_fix_multiple_port_nets = "true";*/
compile_preserve_sync_resets = "true";

/* Edif variables setup (as defined by GPS) */
/* (as dot_synopsys plus edifout_*_net_name must be defined) */

edifout_netlist_only = true
edifout_power_and_ground_representation = net
edifout_power_net_name = VDD
edifout_power_net_property_name = VDD
edifout_power_net_property_value = 1
edifout_ground_net_name = GND
edifout_ground_net_property_name = GND
edifout_ground_net_property_value = 0

edifin_power_net_property_name = VDD
edifin_power_net_property_value = 1
edifin_ground_net_property_name = GND
edifin_ground_net_property_value = 0


/* Net to Port Connection variables */

single_group_per_sheet = "true";
write_name_nets_same_as_ports = "true";


/*********************************/
/***  VHDL netlist parameters  ***/
/*********************************/
vhdlout_architecture_name = "netlist";
vhdlout_single_bit = "VECTOR";
```

C.2 Kode for oppsett av synopsys: .synopsys_vss.setup.

```
------------------
---  Libraries  ---
------------------
WORK                    > WORKLIB
common_rgu              > WORKLIB

WORKLIB         :  ./synlib
```

D C-KODE TIL FASITGENERATOREN

Her følger C-kode for fasitgeneratoren som ble brukt til å teste og verifisere addisjonskretsen.

D.1 float_add.C

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


int read_data(FILE* fd, int n, unsigned int data[])
{
  // Read input data file

  char s[1024];
  int  i_tmp[4];

  if (n <= 0) return 0;

  while (fscanf(fd, "%s", s) != EOF) {

    // Comments

    if (s[0] == '#') {
      while (getc(fd) != '\n');
    }

    else {
      if (sscanf(s, "%x", &data[0]) != 1)
        return 0; // ERROR

      for (int i = 1; i < n; i++) {
        if (fscanf(fd, "%x", &data[i]) != 1)
          return 0; // ERROR
      }
      return 1;
    }
  }
  return 0;  // EOF
}


void err_usage(char* name)
{
  fprintf(stderr, "Usage: %s [input_data_file] \n", name);
  exit(1);
}



int main(int argc, char* argv[])
{
  FILE* fd_in;

  if ((argc < 1) || (argc > 2)) err_usage(argv[0]);
  if (argc == 1) { // Read from stdin
    fd_in = stdin;
  }
  else {
```

```
    fd_in = fopen(argv[1], "r");
    if (fd_in == 0) {
      perror(argv[1]);
      exit(1);
    }
  }

  unsigned int data[2];

  while (read_data(fd_in, 2, data)) {
    data[0] <<= 8;
    data[1] <<= 8;
    float* f = (float*)data;
    float f_res = f[0] + f[1];
    unsigned int* i_res = (unsigned int*)&f_res;
//    printf("%08X \n", *i_res);
    printf("%06X \n", (*i_res) >> 8);
  }
}
```

D.2 sim_fp.C

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <complex.h>
#include <limits.h>


#include "../lib/basic.h"



int read_data(FILE* fd, int n, unsigned int data[])
{
  // Read input data file

  char s[1024];
  int  i_tmp[4];

  if (n <= 0) return 0;

  while (fscanf(fd, "%s", s) != EOF) {

    // Comments

    if (s[0] == '#') {
      while (getc(fd) != '\n');
    }

    else {
      if (sscanf(s, "%x", &data[0]) != 1)
```

```
        return 0; // ERROR

      for (int i = 1; i < n; i++) {
        if (fscanf(fd, "%x", &data[i]) != 1)
          return 0; // ERROR
      }
      return 1;
    }
  }
  return 0;  // EOF
}


void err_usage(char* name)
{
  fprintf(stderr, "Usage: %s operation input_data_file \n", name);
  fprintf(stderr, "operation : cfpmul, fpadd, fpsub, fpmul, sqrt,
fp2intNN, sint2fpNN, uint2fpNN \n");
  exit(1);
}


int main(int argc, char* argv[])
{
  int operation = 0;

  if ((argc < 3) || (argc > 3)) err_usage(argv[0]);

  if (strcmp(argv[1], "cfpmul") == 0)
    operation = 0;
  else if (strcmp(argv[1], "fpadd") == 0)
    operation = 1;
  else if (strcmp(argv[1], "fpsub") == 0)
    operation = 2;
  else if (strcmp(argv[1], "fpmul") == 0)
    operation = 3;
  else if (strcmp(argv[1], "sqrt") == 0)
    operation = 4;
  else if (strncmp(argv[1], "fp2int", 6) == 0)
    operation = 5;
  else if (strncmp(&argv[1][1], "int2fp", 6) == 0)
    operation = 6;
  else
    err_usage(argv[0]);


  FILE* fd_in = fopen(argv[2], "r");
  if (fd_in == 0) {
    perror(argv[2]);
    exit(1);
  }


  unsigned int data[6];
```

```
  complex c0, c1, cr;

  if (operation == 0) { // cfpmul
    complex c0, c1, cr;
    while (read_data(fd_in, 6, data)) {
      // Skip data[0] and data[1]
      c0 = complex(f24_to_f32(data[2] >> 8), f24_to_f32(data[3] >>
8));
      c1 = complex(f24_to_f32(data[4] >> 8), f24_to_f32(data[5] >>
8));
      cr = c0 * c1;
//      printf("%08x %08x \n", f64_to_f24(real(cr)),
f64_to_f24(imag(cr)));
      printf("%02X  %X  %08X %08X  %08X %08X  %08X %08X %05X \n",
              data[0], data[1], data[2], data[3], data[4], data[5],
f64_to_f24(real(cr)), f64_to_f24(imag(cr)), 0);
    }
  }


  else if (operation == 1) { // fpadd
    double d0, d1, dr;
    while (read_data(fd_in, 4, data)) {
      // Skip data[0] and data[1]
      d0 = f24_to_f64(data[2] >> 8);
      d1 = f24_to_f64(data[3] >> 8);
      dr = d0 + d1;
//      printf("%08x \n", f64_to_f24(dr));
      printf("%02X  %X  %08X  %08X  %08X %05X \n",
             data[0], data[1], data[2], data[3], f64_to_f24(dr), 0);
    }
  }



  else if (operation == 2) { // fpsub
    double d0, d1, dr;
    while (read_data(fd_in, 4, data)) {
      // Skip data[0] and data[1]
      d0 = f24_to_f64(data[2] >> 8);
      d1 = f24_to_f64(data[3] >> 8);
      dr = d0 - d1;
//      printf("%08x \n", f64_to_f24(dr));
      printf("%02X  %X  %08X  %08X  %08X %05X \n",
             data[0], data[1], data[2], data[3], f64_to_f24(dr), 0);
    }
  }

  else if (operation == 3) { // fpmul
    double d0, d1, dr;
    while (read_data(fd_in, 4, data)) {
      // Skip data[0] and data[1]
      d0 = f24_to_f64(data[2] >> 8);
      d1 = f24_to_f64(data[3] >> 8);
```

```
      dr = d0 * d1;
//      printf("%08x \n", f64_to_f24(dr));
      printf("%02X  %X  %08X  %08X  %08X %05X \n",
             data[0], data[1], data[2], data[3], f64_to_f24(dr), 0);
    }
  }

  else if (operation == 4) { // sqrt
    double d0, dr;
    while (read_data(fd_in, 4, data)) {
      // Skip data[0] and data[1]
      d0 = f24_to_f64(data[2] >> 8);
      dr = sqrt(d0);
//      printf("%08x \n", f64_to_f24(dr));
      printf("%02X  %X  %08X  %08X  %08X %05X \n",
             data[0], data[1], data[2], data[3], f64_to_f24(dr), 0);
    }
  }

  else if (operation == 5) { // fp2intNN (NN = 8, 16 or 32)
    double d0;
    int    ir;
    int n_bit = 0;

    sscanf(&argv[1][6], "%d", &n_bit);

    while (read_data(fd_in, 4, data)) {
      // Skip data[0] and data[1]
      int overflow = 0;

      d0 = f24_to_f64(data[2] >> 8);
      ir = (int)rint(d0);

      if ((data[2] & 0x7f800000) == 0x7f800000)
        overflow = 1;

      if (n_bit == 32) {
        if (d0 > INT_MAX)
          overflow = 1;
        if (d0 < INT_MIN)
          overflow = 1;
      }
      else if (n_bit == 16) {
        if (ir > SHRT_MAX) {
          ir = SHRT_MAX;
          overflow = 1;
        }
        if (ir < SHRT_MIN) {
          ir = SHRT_MIN;
          overflow = 1;
        }
        ir = (ir << 16) >> 16;
      }
      else if (n_bit == 8) {
```

```
        if (ir > CHAR_MAX) {
          ir = CHAR_MAX;
          overflow = 1;
        }
        if (ir < CHAR_MIN) {
          ir = CHAR_MIN;
          overflow = 1;
        }
        ir = (ir << 24) >> 24;
      }
      else {
        fprintf(stderr, "Ilegal operation %s\n", argv[1]);
        exit(1);
      }

//      printf("%08x \n", ir);
      printf("%02X  %X  %08X  %08X  %08X %04X%X \n",
             data[0], data[1], data[2], data[3], ir, 0, overflow);
      }
      else {
        fprintf(stderr, "Ilegal operation %s\n", argv[1]);
        exit(1);
      }

//      printf("%08x \n", ir);
      printf("%02X  %X  %08X  %08X  %08X %04X%X \n",
             data[0], data[1], data[2], data[3], ir, 0, overflow);
    }
  }

  else if (operation == 6) { // sint2fpNN, uint2fpNN
    int    i0;
    double dr;
    int n_bit = 0;
    int sign = 0;

    if (argv[1][0] == 's')
      sign = 1;
    else if (argv[1][0] == 'u')
      sign = 0;
    else {
      fprintf(stderr, "Ilegal operation %s\n", argv[1]);
      exit(1);
    }

    sscanf(&argv[1][7], "%d", &n_bit);
    if (n_bit > 24) {
      fprintf(stderr, "Ilegal operation %s\n", argv[1]);
      exit(1);
    }


    while (read_data(fd_in, 4, data)) {
      // Skip data[0] and data[1]
```

```
      i0 = data[2];

      if (sign)
        dr = (double)((i0 << (32 - n_bit)) >> (32 - n_bit));
      else
        dr = (double)(i0 & (0xffffff >> (24 - n_bit)));

//      printf("%08x \n", f64_to_f24(dr));
      printf("%02X  %X  %08X  %08X  %08X %05X \n",
            data[0], data[1], data[2], data[3], f64_to_f24(dr), 0);
    }
  }

}
```

D.3 Filen basic.h

```
#ifndef _BASIC_H
#define _BASIC_H

#include <complex.h>

#define max(x, y) ((x) > (y) ? (x) : (y))
#define min(x, y) ((x) < (y) ? (x) : (y))



double fcmp(double a, double b);
int compare(int cnt, complex *res, complex *fac, float limit, int
print);
int pow_int(int m, int e);
int log_2(int v);

int is_2_in_n(
/*
   Check that input is 2**n,
   where n is 1,2,3,...
*/
  int x);     // The number to check

int first_radix(
/*
   Return the the radix of the first butterfly in an fft
   where:

   fft_len = first_radix * pow(max_radix, n)
*/
  int fft_len,     // The actual fft length
  int max_radix); // The highes radix used

int n_stage(
```

```
/*
   Return the the the number of passes (n+1) in an fft
   where:

   fft_len = first_radix * pow(max_radix, n)

*/
  int fft_len,     // The actual fft length
  int max_radix); // The highes radix used

unsigned int mv_lower_bits(unsigned int val, unsigned int n_bit,
unsigned int pos);
void bprint(int val, int n_bit);
unsigned int swap(unsigned int val, int base, int n_bit);
unsigned int swap_u(unsigned int val, int base, int n_bit);


complex to_15_bit_mantissa(complex data);
   where:

   fft_len = first_radix * pow(max_radix, n)

*/
  int fft_len,     // The actual fft length
  int max_radix); // The highes radix used

unsigned int mv_lower_bits(unsigned int val, unsigned int n_bit,
unsigned int pos);
void bprint(int val, int n_bit);
unsigned int swap(unsigned int val, int base, int n_bit);
unsigned int swap_u(unsigned int val, int base, int n_bit);


complex to_15_bit_mantissa(complex data);
double to_15_bit_mantissa(double data);
complex cmult(complex a, complex b);

double f24_to_f64(int data);
float f24_to_f32(int data);
int f64_to_f24(double data);
long double f40_to_f128(unsigned long long data);
long double f24_to_f128(int data);
int f128_to_f24(long double data);

/*
  41 bit format: sign (1 bit) exp (9 bit) mant (31 bit)
*/
long long f64_to_f41(double data);
double f41_to_f64(long long data);
long double f41_to_f128(unsigned long long data);

int fpadd_24bit(int a, int b, int op);
int fpmul_24bit(int a, int b);
int fpsqrt_24bit(int a);
```

```
int i2fpconv_24bit(int signed_input, int n_bit, int a);
long long fpshift_24bit(int cpx, int shr, int n_sh, int re, int im);
long long pack_ll(int re, int im);
int unpack_re(long long a);
int unpack_im(long long a);
long long cadd_packed(long long a, long long b);
long long fpacc_24bit(int reset, int cpx, int n_acc, int re, int
im);
long long cmul_24bit(int a_re, int a_im, int b_re, int b_im);


int fpadd_40bit(unsigned long long a, unsigned long long b);
long long fpmul_41bit(int a, int b);
int fpadd_41bit(unsigned long long a, unsigned long long b, int op);

int post_conv(int reset, int opcode, int re, int im);

#endif
```

E PARAMETERSETT FOR TESTBENK

Parametre til mantissegenerering er startverdi, avstand mellom verdier, stoppverdi og hvilket mantissemønster som skal brukes. Eksponentene kan gå gjennom opptil tre forskjellige tallområder med hver sin startverdi, avstand mellom verdier og stoppverdi. Tallområdene velges slik at eksponentverdien alltid er stigende. Fortegnene starter som 0 og skifter til 1.

parameternavn	**sett1**	**sett2**	**sett3**	**sett4**
exp1 start value :	100	0	105	0
exp1 inc1 value :		1		
exp1 stop1 value :		2		
exp1 start2 value :		127		
exp1 inc2 value :		1		
exp1 stop2 value :		128		
exp1 start3 value :		253		
exp1 inc3 value :	1	1	1	1
exp1 stop value :	100	255	149	255
exp2 start value :	100	0	105	0
exp2 inc1 value :		1		
exp2 stop1 value :		2		
exp2 start2 value :		127		
exp2 inc2 value :		1		
exp2 stop2 value :		128		
exp2 start3 value :		253		
exp2 inc3 value :	1	1	1	1
exp2 stop value :	100	255	149	255
mant1 start value :	0	0	0	0
mant1 inc value :	1	1	1	1
mant1 stop value :	60	60	4	2
mant1 type :	2	2	1	1
mant2 start value :	0	0	0	2
mant2 inc value :	1	1	1	1
mant2 stop value :	60	60	4	4
mant2 type :	2	2	1	1

Alle datasettene kjøres med alle kombinasjoner av operasjonskode og fortegn.
(add++, add+-, add-+, add--, sub++, sub+-, sub-+, sub--)