FFIE/726/170

Godkjent
Kjeller 27 januar 2000

Torleiv Maseng
Forskningssjef

MULTIPLIKASJONSKRETS FOR IEEE–754 FLYTTALL OPTIMALISERT FOR GJENNOMSTRØMNING

GUNDERSEN Rune, BLOM Harald

FFI/RAPPORT–2000/00491

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–2000/00491	UNCLASSIFIED	115
1a) PROJECT REFERENCE	2a) DECLASSIFICATION/DOWNGRADING SCHEDULE	
FFIE/726/170		

4) TITLE

MULTIPLIKASJONSKRETS FOR IEEE–754 FLYTTALL
OPTIMALISERT FOR GJENNOMSTRØMNING
(IEEE–754 FLOATING POINT MULTIPLIERER 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 Multiplication	a) Flyttallsmultiplikasjon
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 discription of a floting point adder optimized for throughput, rather than latecy. It is confirm with IEEE–754 standard for binary floting–point atrithmetic. The design is fully tested and verified. Synthetesis results for the Alcatel Mietec MTC45000 technology is presented.

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

ISBN 82-464-0402-4

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

INNHOLD

MULTIPLIKASJONSKRETS 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. Multiplikasjonskretsen beskrevet i denne rapporten inngår som en viktig byggeblokk i FFT–prosessoren.

Multiplikasjonskretsen 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 (13)(16). 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 er IEEE–754 standard floating–point (11) 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 multiplikasjon 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 multiplikasjonskretsen. Det inneholder en beskrivelse av kretsens egenskaper, grensesnitt til omverdenen og indre struktur.

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

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 multiplikasjonskretsen, de egendefinerte typebibliotekene og testbenken, C–kode for fasitgeneratoren, oppsettet for Synopsys Design Compiler og parametersett for de ulike testene.

Kretsen inneholder 4487 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 regne–maskiner og som følger gitte standarder eller et format som enkelt kan konverters 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 pipeline arkitektur 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 flyttall representasjon.

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

FFT–prosessoren er nærmere beskrevet i en egen rapport (5).

3 TALLFORMAT

I dette kapitlet beskrives tallformatet vi benytter i multiplikasjonskretsen. Med tre unntak følger vi IEEE–754 (7) (9) (11).

Standaren definerer hvordan flyttall representeres binært i digitale systemer og hvordan aritmetiske operasjoner utføres på slike tall. Den sikrer at data kan flyttes mellom forskjellige datamaskiner og at svarene på 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 parametere 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 parametere 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, pr 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:

$$F = (-1)^S \times 0.f \times 2^{-127} \tag{3.2}$$

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

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 multiplikasjonskretsen var en av submodulene 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":

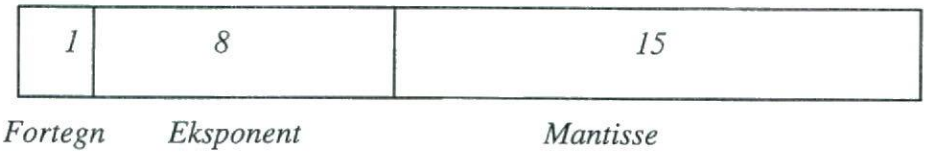
Den utvidelsen av tallområdet denormaliserte tall representeres ble funnet å resultere i uforholdsmessig mye logikk. Derfor supporterer multiplikasjonskretsen ikke denormaliserte tall. Hvis en av operandene er et denormalisert tall settes denne lik null før multiplikasjonen utføres.

Behandlingen av NaN er forenklet. Hvis en av operandene til multiplikasjonskretsen er NaN, slippes denne uforandret igjennom. Hvis resultatet av multiplikasjonen 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.

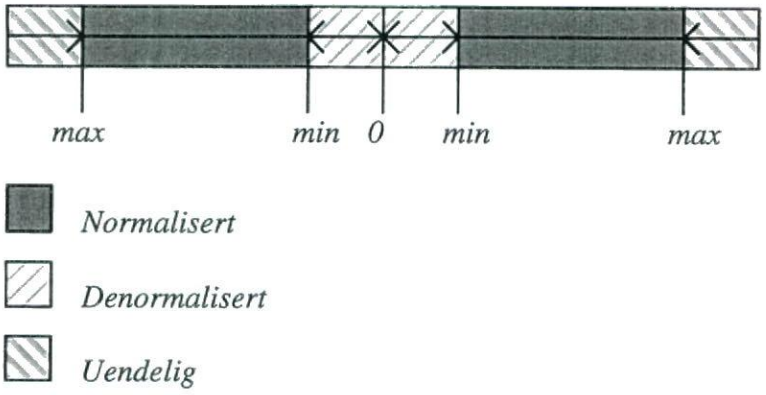
Multiplikasjonskretsen benytter mao et tallformat på 24–bit med en bit til fortegnet, 8 biter til eksponenten og 15 biter til mantissen, som vist i figur 3.1, med følgende oppbygging:

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



Figur 3.1 *Vårt 24–biters tallformat med 1 bit til fortegnet, 8 biter til eksponenten og 15 biter til mantissen.*

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



Figur 3.2 *Dynamikkområdet til vårt 24–biters tallformat.*

		Eksponent	Mantisse	Desimalt
Null		00	8000	Null
Denorm		00	$\neq 0$	Null < denorm < Normaliset_{min}
Normalisert	min	01	8000	1.175494×10^{-38}
	max	FE	7FFF	3.402823×10^{38}
Uendelig		FF	8000	$\infty > \text{Normalisert}_{max}$
NaN		FF	$\neq 0$	NaN > ∞

Tabell 3.5 *Dynamikkområdet til vårt 24–biters 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 under–intervaller, (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

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

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 uønsket 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 MULTIPLIKASJON AV FLYTTALL

I dette kapitelet diskuteres flyttallsmultiplikasjon i noen detalj. Vi viser hvordan multiplikasjonen kan deles opp i deloperasjoner og hvordan de ulike deloperasjonene kan utføres.

Den enkleste måten å gjøre dette på er å benyttet en av de generiske heltallsmultiplikasjonskretsene for multiplikasjon som ligger i Synopsys Design Compiler. Basert på våre synteseresultater, viste det seg at disse multiplikasjonskretsene ikke hold mål. De var for trege til å oppfylle våre systemkrav. For å oppnå akseptable resultater, designet vi derfor en multiplikasjonskrets selv, basert på et grundig literaturstudie. Vi presenterer derfor noe teori rundt design av hel– og flyttalls multiplikasjonskretser og utvelgelseskriterier i dette kapitlet. Vi beskriver også en algoritme for flyttallsmultiplikasjon. En grundigere gjennomgang av temaet finnes i (6)(10).

4.1 Flyttallsmultiplikasjon

En krets for flyttallsmultiplikasjon tar som input to flyttall med n bit presisjon og produserer et n–bits resultat. Resultatet av operasjonen er avrundet fordi vi har like mange bit i resultatet som i operandene.

Vi lar A og B være to flyttall som skal multipliseres. 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. En multiplikasjon av to flyttall, på formen vist i ligning (3.1), kan da utføres som vist i ligning (4.1).

$$\begin{aligned} A \cdot B &= (-1)^{S_A} \cdot M_A \cdot 2^{E_A+b} \cdot (-1)^{S_B} \cdot M_B \cdot 2^{E_B+b} \\ &= ((-1)^{S_A} \cdot (-1)^{S_B}) \cdot (M_A \cdot M_B) \cdot 2^{(E_A+E_B+2b)} \end{aligned} \tag{4.1}$$

Som vi ser av ligningen består en flyttallsmultiplikasjon av:

- En multiplikasjon av mantissene til de to multiplikandene
- En addisjon av multiplikandenes eksponenter
- En XOR–operasjon av multiplikandenes fortegn

En flyttallsmultiplikasjon kan derfor naturlig deles i tre parallelle beregninger. Den mest komplekse av disse er multiplikasjonen av mantissene som er en heltallsmultiplikasjon. I de følgende avsnitt tar vi derfor for oss noen detaljer rundt heltallsmultiplikasjon.

4.2 Heltallsmultiplikasjon

En multiplikasjon av to tall, multiplikand og multiplikator, kan utføres ved at multiplikanden legges sammen multiplikand antall ganger. Vi genererer først partielle produkter, hvor hvert partielle produkt er produktet av en bit i multiplikanden multiplisert med hele multiplikatoren. Hvert partielle produkt venstrejusteres slik at det får riktig vekt. Deretter sum-

meres de partielle produktene. Heltallsmultiplikasjonen kan altså brytes ned i to operasjoner:

- Generere de partielle produktene.
- Summere de partielle produktene.

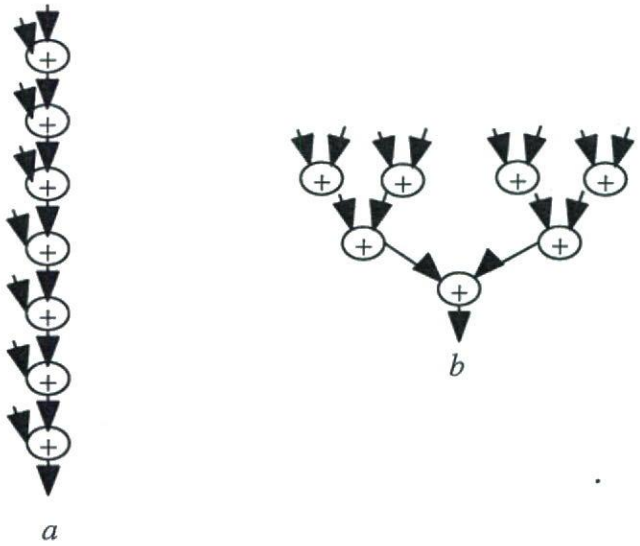
De partielle produktene genereres vha *carry–save–addere* som er bygd opp av generelle byggeklosser som full– og halvaddisjonskretser. Summasjonen av de partielle produktene gjøres ved hjelp av addisjonskretser.

4.3 Parallelle multiplikasjonsalgoritmer (trestrukturer)

Ved å utnytte muligheten til å dele opp en multiplikasjon i flere parallelle utregninger, der partielle produkter med samme vekt genereres samtidig, kan man oppnå en raskere multiplikasjonskrets med liten bruk av ekstra maskinvare. En sammenstilling av halv– og fulladdisjonskretser i en trestruktur kalles et addisjonstre. Addisjonstrær er alminnelig anerkjente for å ha tilnærmet optimal ytelse, med hensyn på hastighet, for addisjon av flere operander ved hjelp av 'carry–save addere' (CSA) (1). Den første som presenterte en slik struktur var Wallace (21).

Figur 4.1. viser en sammenligning mellom en lineær addisjonskrets og et addisjonstre. Som vi ser av figuren er summasjonstiden for en lineær addisjonskrets er av orden O(n), mens den er av orden O(log$n$) for et addisjonstre.

Det største ankepunktet mot slike trær er at de ofte medfører svært kompleks intern ruting, noe som er arealkrevende.



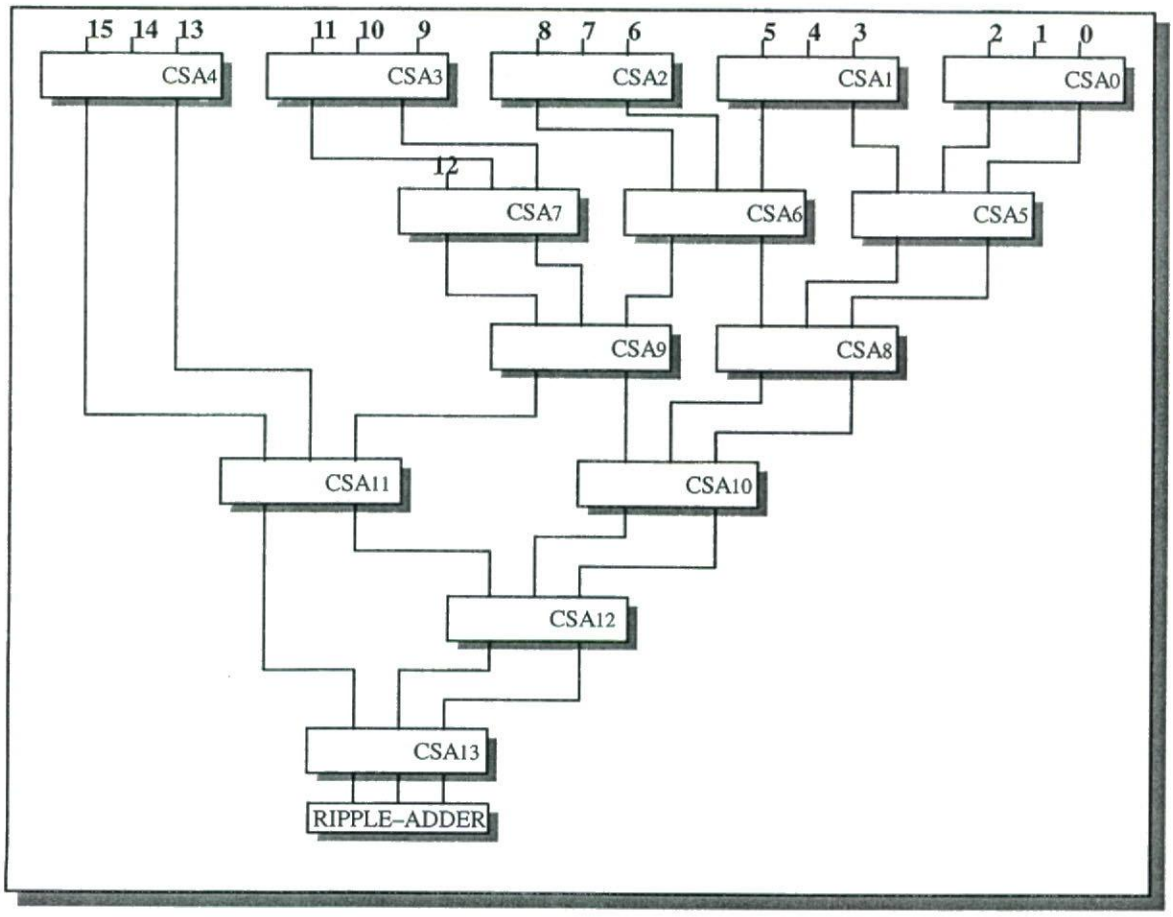
Figur 4.1 Sammenligning av et lineært summasjonsnettverk (a) og en trestruktur (b) for summering av åtte biter med lik vekt. Ved bruk av en trestruktur reduseres summasjonstiden fra O(n) til O(log$n$). (Høyden på summasjonsnettverket i figuren reduseres fra 7 til 3).

4.4 Konstruksjon av heltallsmultiplikasjonstræ

Vi har valgt å implementere en modifisert utgave av et Wallace–tre hvor vi benytter en teknikk som kalles *overturned–stairs* (OS). OS er en teknikk for å optimalisere sammenkoblingen mellom de parallelle grenene i et addisjonstre (17).

Våre konstruksjonskriterier satte høye krav til arealbruk og gjennomstrømning (throughput), men lave krav til hastighet (latency). OS–treet ble derfor modifisert for å passe bedre til konstruksjonskriteriene. Implementasjonen av OS–trær i (15) benyttet komplekse addisjonsblokker optimalisert for hastighet som derfor var plasskrevende. Vi skiftet ut disse addisjonsblokkene med enklere varianter som var mindre arealkrevende.

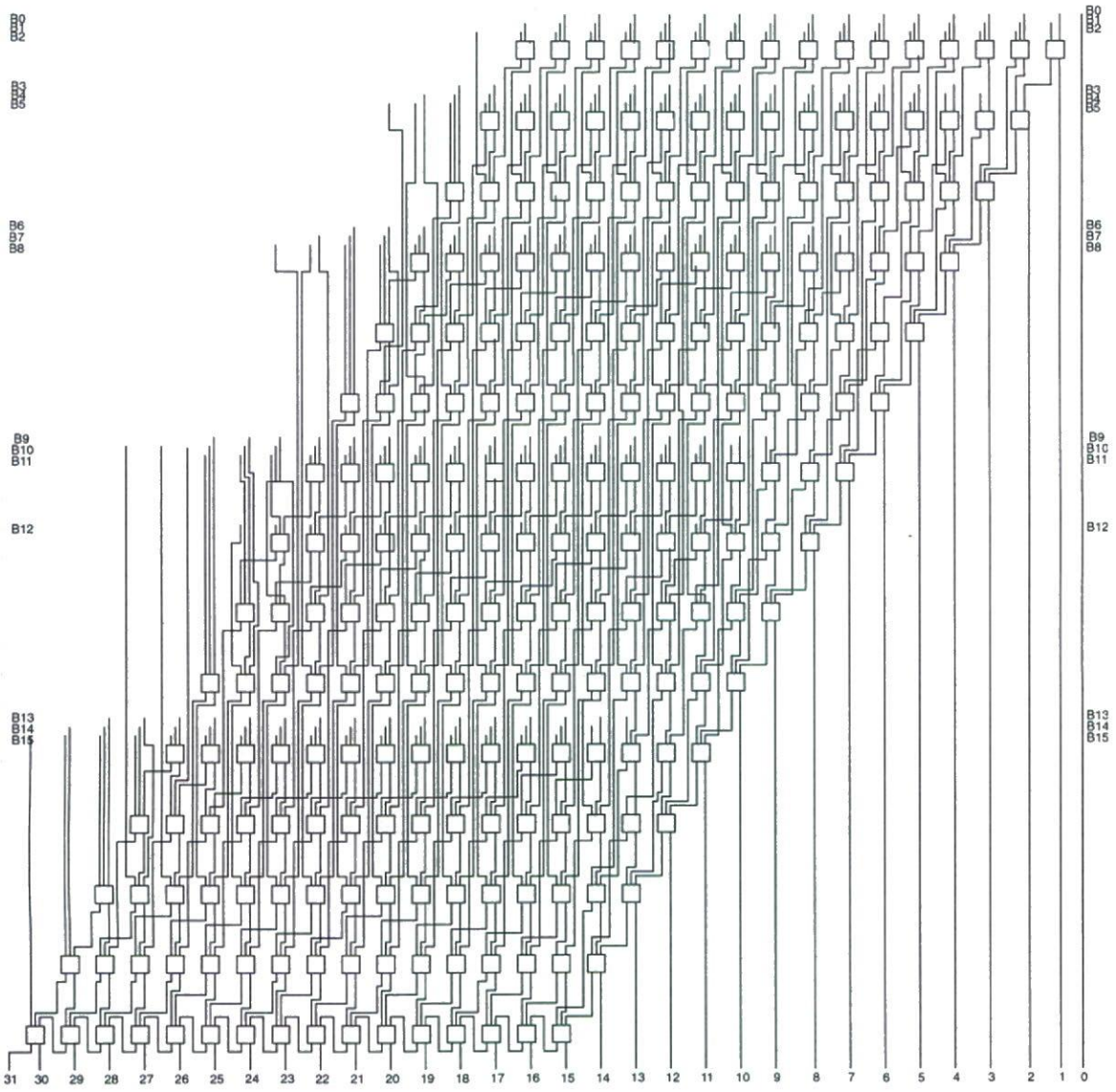
Figur 4.2 viser trestrukturen som vår implementasjon benytter.



Figur 4.2 Trestrukturen i vår heltallsmultiplikator. Blokkene inneholder Carry–Save addere (CSA).

Figur 4.3 inneholder en detaljert skisser av vår implementasjon av heltallsmultiplikatoren.

I figuren er rektanglene med tre innganger og to utganger full–addisjonskretser. De med to innganger og to utganger er halv–addisjonskretser.



Figur 4.3 *Illustrasjon av et* 16 · 16 *binær matrise multiplikasjonstre.*

4.5 Algoritme for flyttallsmultiplikasjon

Som vist i avsnitt 4.1, kan en flyttallsmultiplikasjon naturlig deles i tre parallelle beregninger. Da kan flyttallsmultiplikasjon 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 flag-

ges vha "ulovlig operasjon" flagget. De ulovlige operasjonene er $0 * (+\infty)$ og $0 * (-\infty)$. Hvis en av operandene er null, settes denne verdien som resultat forutsatt lovlig operasjon. Er en av operandene en NaN, settes resultatet til samme NaN. Denormaliserte tall settes til 0.

Fortegnsbehandling

Fortegnet til produktet er en *XOR*–operasjon av multiplikandenes fortegn, $S_Z = S_A \otimes S_B$.

Eksponenthåndtering

Eksponenten til mellomresultatet, før normalisering av mantisseproduktet, er gitt av en addisjon av eksponentene til de to multiplikandene $E_{tmp} = E_A + E_B$.

Multiplikasjon av de to mantissene

Multiplikasjonen utføres i en 16–biters heltalls matrisemultiplikator $P = M_A \cdot M_B$. Resultatet av en 16–biters multiplikasjon gir et 32–biters svar. Dette svaret må avrundes og normaliseres for å kunne representeres. Fordi antall bit skal halveres, tilordnes nedre halvdel av produktet til guard, round og sticky. De nederste 16 bitene brukes altså kun til avrunding av svaret.

Normalisering av resultatet

Siden alle lovlige mantisseverdier inn til multiplikatoren ligger i intervallet $1.0 \leq innverdi < 2.0$ vil produktet av to slike mantisser ligge i intervallet $1.0 \leq produkt < 4.0$. Hvis produktet får verdien to eller større, justeres P en posisjon til høyre og eksponenten E_{tmp} økes med en. Hvis ikke forblir P uforandret.

Justering av round og sticky

Hvis P under normaliseringen ble justert til høyre så setters $s := g \ OR \ r \ OR \ s$. Round settes lik det minst signifikante biten i P. Hvis P ikke ble justert, settes $s := r \ OR \ s$ og $r := g$.

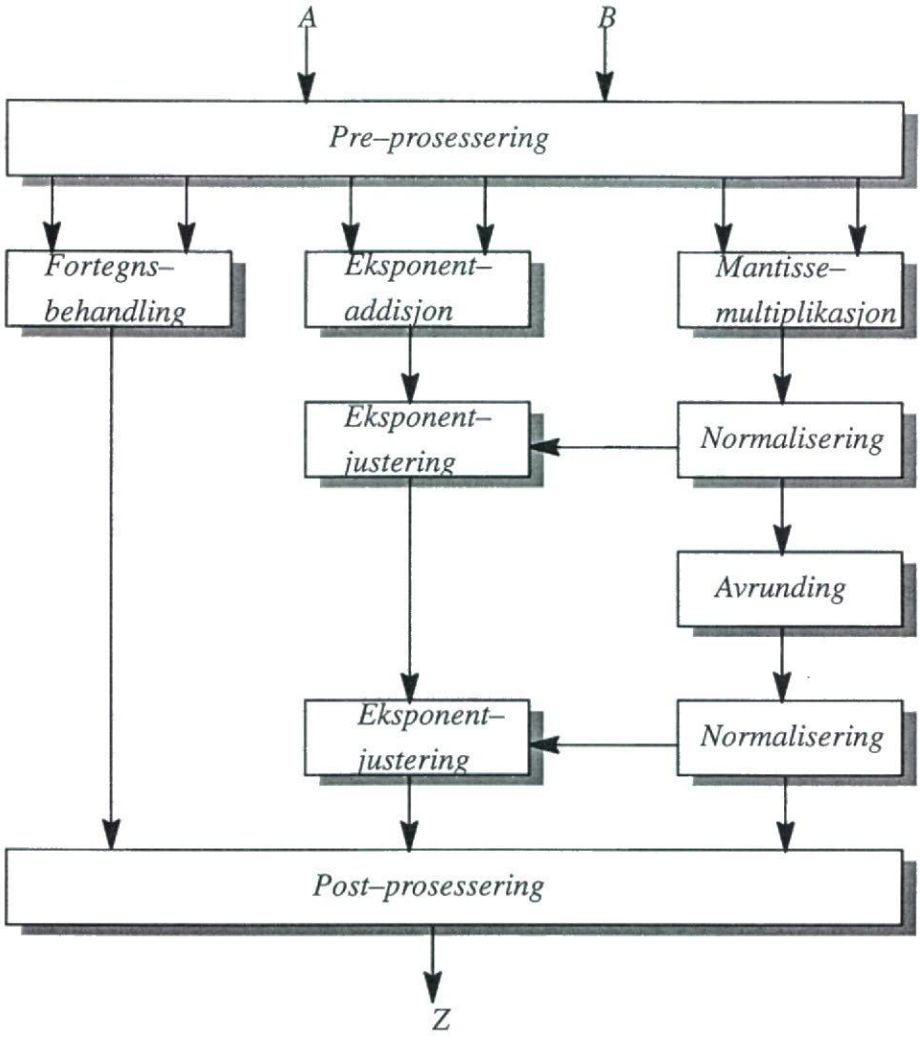
Avrund det normaliserte resultatet

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

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 4.4.



Figur 4.4 Flytskjema for flyttallsmultiplikasjons prosessen.

4.6 Optimaliseringer

Det er publisert mange håndgrep som kan brukes for å utføre multiplikasjonen 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 multiplikasjonskretser, har vi ikke strenge krav til tidsbruken for å utføre en multiplikasjon (latency). Det viktige for oss er gjennomstrømingen av data (throughput). Vi må 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 multiplikasjonskretsen. Hvor mange pipelinetrinn kretsen skal ha, er allikevel en avveining. Hvor mange porter som går med til å lage et ekstra pipelinetrinn 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 multiplikasjonsjonskretsen fant vi at følgende optimalisering var gunstig:

En direkte addisjon, av de to multiplikandenes eksponenter, vil resultere i en sum der vi har tatt med biasen til eksponenten to ganger:

$$E_{tmp} = (E_A + b) + (E_B + b) = E_A + E_B + 2b \tag{4.2}$$

Dette vil medføre et galt svar og vi må trekke i fra én bias. Hvilket medfører en addisjonskrets fulgt av en subtraksjonskrets, dette søker vi å unngå. Derfor utnytter vi en metode som er raskere og krever mindre ressurser (1). Vi utnytter at eksponentens bias er gitt ved $2^{N-1} - 1$, hvor N er antall bit i eksponenten. Eksponenten med bias har da følgende verdi:

$$E_{tmp} = E_A + E_B + 2b = E_A + E_B + 2(2^{N-1} - 1) \tag{4.3}$$

Vi ser da at ved å invertere den mest signifikante biten i eksponenten, subtraherer vi i praksis 2^{N-1} fra E_{tmp}. En addisjon, av de to multiplikandenes eksponenter, hvor dette utnyttes gir da:

$$\begin{aligned} E_{tmp} &= E_A + (2^{N-1} - 1) - (2^{N-1}) + E_B + (2^{N-1} - 1) - (2^{N-1}) \\ &= E_A + E_B - 2 \end{aligned} \tag{4.4}$$

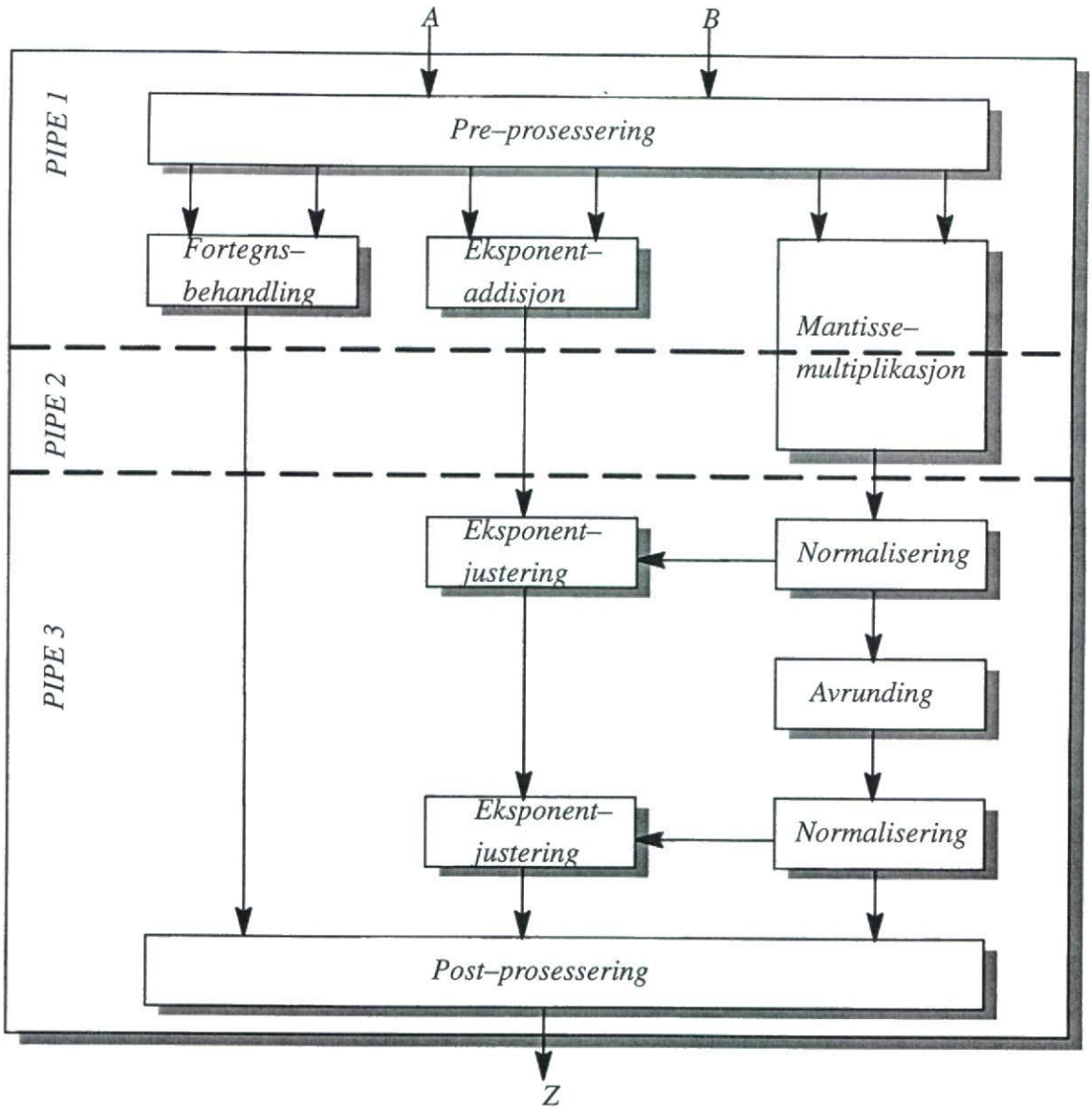
Ved å sette carry–inngangen på addisjonskretsen får vi addert 1 til summen og ved å invertere den mest signifikante biten i E_{tmp} får vi den korrekte summen.

$$E_{tmp} = E_A + E_B - 2 + 1 + 2^{N-1} = E_A + E_B + 2^{N-1} - 1 = E_A + E_B + b \tag{4.5}$$

Ved å invertere de to mest signifikante bitene i E_A og E_B før addisjonen og ved å invertere den mest signifikante biten i E_{tmp} etter addisjonen, har vi spart en subtraksjon av biasen fra eksponentsummen etter addisjonen.

4.7 Pipelining av addisjonskretsen

Det ble etterhvert klart at den opsjonen for automatisk pipelining i Synopsys Design Compiler ikke greide å håndtere så komplekse design som denne flyttalsmultiplikasjonskretsen. Vi måtte derfor legge inn alle registere og pipelinetrinn for hånd, noe som medførte en del forarbeid før syntese. Med utgangspunkt i en del synteseresultater fra multiplikasjonskretsen som ikke var pipelinet, viste det seg mulig å greie systemkravene med tre pipelinetrinn. Første trinn ble lagt omlag 2/3 ute i mantissemultiplikasjonen. Andre trinn ble plassert mellom mantissemultiplikasjonen og normaliseringsenheten. En grafisk fremstilling av multiplikasjonskretsen med tre pipelinetrinn er vist i figur 4.5.



Figur 4.5 Flytskjema for flyttallsmultiplikasjonen med visualisering av de tre innlagte pipeline–trinnene.

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 programeringspråk. 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 synteseprogram 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 (12) 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 synteseverktøyet å beskrive kretsen på portnivå.

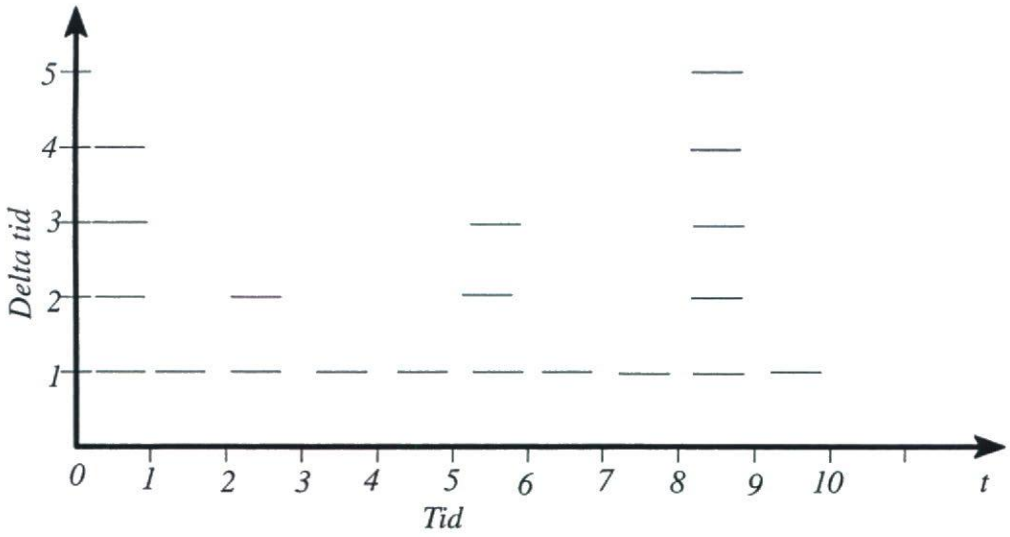
En slik VHDL–beskrivelse kan være teknologiuavhengig. Dette letter gjenbruk av ferdige konstruksjoner. En og samme beskrivelse kan syntetiseres forskjellig for flere teknologier. Da maskinvarebeskrivelsen er skrevet i et standard språk, er den portabel mellom maskiner og verktøy. Ved bruk av VHDL er 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 implementerer 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 (13)(16).

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 (14) (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 modellene 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 regneverket 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 synteseprogrammet som automatisk oversetter vår abstrakte kode til en strukturell beskrivelse på et lavere abstraksjonsnivå (18)(20).

Syntеseverktøyet 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 et syntеseverktøy er dog å skrive optimal VHDL–kode (2)(4). Frihetsgradene i programeringsspråket er langt større enn i syntеseverktøyet. Det er ikke noe problem å skrive VHDL kode som ikke lar seg realisere med et syntеseverktøy. Dette setter langt større krav til kunnskap om syntеseverktøyet 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 syntеseverktøy 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 multiplikasjonskretsen 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 multiplikasjonskretsen. Vi ble derfor nødt til å dele opp kretsen og manuelt legge inn pipelineregisterne.

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. Syntеseverktøy er Synopsys Design Compiler. Eneste betingelse satt er tidskrav 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.

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
    tmp := a;
```

```
      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
      tmp := a;
      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
      tmp := a;
      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
```

```
        tmp := a;
        a1 := b;
        b1 := tmp;
    end if;
    res <= a1;
```

Eks	kombinatorikk #porter	ikke kombinatorisk #porter	totalt #porter	tidsmargin 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 eksempelet. Alle eksemplene på kodestil møtte tidskravene, men med svært forskjellig arealforbruk. Merk at totalt arealforbruk varierer sterkt.*

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 halveres. Vi ser av dette eksemplet hvor viktig det er å ha kjennskap til hvordan synteseverktøyet jobber. Metoden har på ingen måte fritatt konstruktøren fra å ha inngående kjennskap til maskinvaren han konstruerer.

6 SYSTEMBESKRIVELSE FOR MULTIPLIKASJONSKRETSEN

Dette kapitelet er en systembeskrivelse av multiplikasjonskretsen. 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

Multiplikasjonskretsen utfører multiplikasjon 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 gjennomstrøming. På grunn av pipelining er resultatene forskjøvet med tre klokkeperioder. Det følgende er en liste over egenskapene til kretsen:

- Multiplikasjonskrets for 24–bits flyttall.
- Kretsen er pipelinet med tre pipeline–trinn.

- Klokkefrekvensen er avhengig av hvilken teknologi som velges.
- Multiplikasjonen utføres i en array–multiplikator.
- Hold–funksjonalitet er implementert. Kretsen kan pauses uten at data går tapt.
- Reset–funksjonalitet er implementert. Kretsen kan nullstilles uavhengig av inngangs–data.

Vi angir størrelsen på to versjoner av multiplikasjonskretsen; en med to pipelinetrinn og en med tre pipelinetrinn. Begge er optimalisert for Alcatel Mietecs $0.35\mu m$ CMOS prosess (3). Begge kretsene er beskrevet i VHDL og optimalisert for syntese med Synopsys Design Compiler (20).

- Størrelse med to pipeline trinn: 4084 portekvivalenter (optimalisert for 15 ns klokke–periode)
- Størrelse med tre pipeline trinn: 4487 portekvivalenter (optimalisert for 13.5 ns klokke-periode)

6.2 Pinnebeskrivelse

Tabell 6.1 inneholder en pinnebeskrivelse for multiplikasjonskretsen. 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
multipli-cand	inn	24	1	Operand A inn til multiplikasjons-kretsen	nfloat	fortegn
			8			eksponent
			15			mantisse
multiplier	inn	24	1	Operand B inn til multiplikasjons-kretsen	nfloat	fortegn
			8			eksponent
			15			mantisse
clk	inn	1	–	Klokkeinngang	std_logic	
resetn	inn	1	–	Nullstiller kretsen	std_logic	aktiv lav
hold	inn	1	–	Stopper kretsen	std_logic	aktiv høy
product	ut	24	1	Resultatet av mul-tiplikasjonen	nfloat	fortegn
			8			eksponent
			15			mantisse

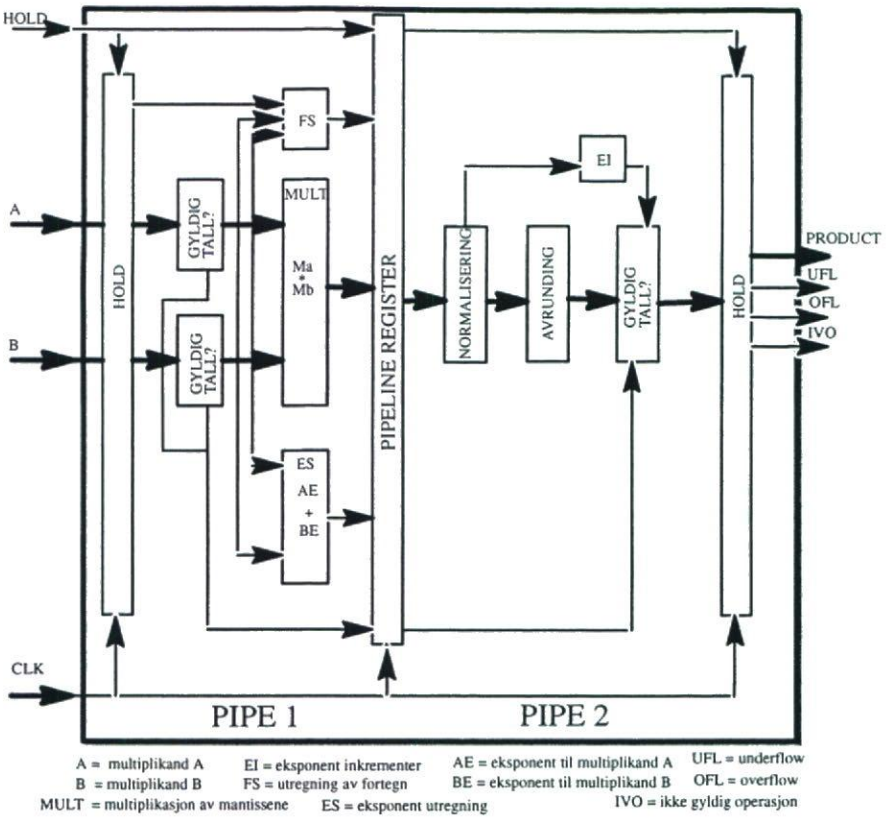
mult_ufl	ut	1	–	Flagger underflow	std_logic	
mult_ofl	ut	1	–	Flagger overflow	std_logic	
mult_ ivo	ut	1	–	Flagger ugyldig operasjon	std_logic	

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

6.3 Strukturell beskrivelse

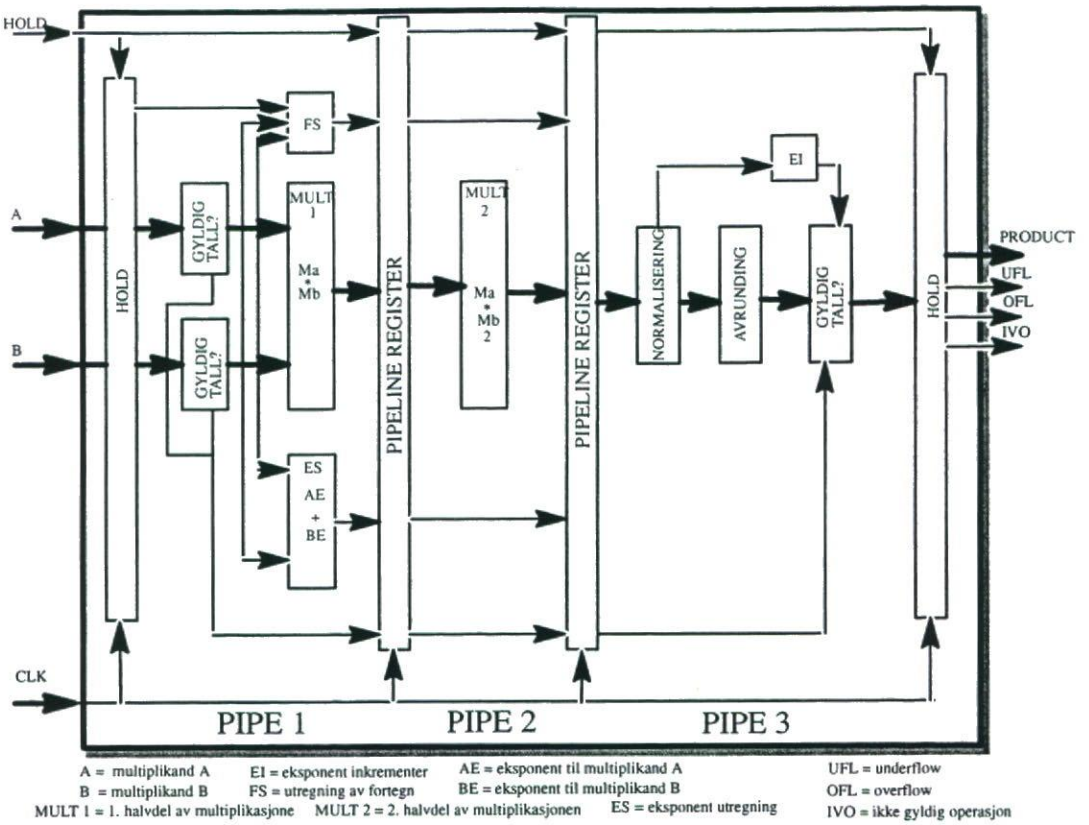
Den indre strukturen til de to versjonene av multiplikasjonskretsen vises i figur 6.1 og 6.2.

Figur 6.1 viser multiplikasjonskretsen med to pipelinetrinn. 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. Modulen for utregning av eksponenten, ES, summerer de to multiplikandenes eksponenter. Modulen består av en 8 biters addisjonskrets. I MULT modulen multipliseres de to multiplikandenes mantisser. Dette utføres i en 16 biters array–multiplier. 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 flyttallsmultiplikasjonskretsen med to pipeline–trinn.

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 og at multiplikasjonen av de to multiplikandenes mantisser er splittet i to. Splitting er lagt midt i multiplikatoren mellom utregningen av delprodukt m_9 og m_{10} referert til figur 4.2.



Figur 6.2 Funksjonelt blokkdiagram over den indre strukturen til flyttallsmultiplikasjonskretsen med tre pipeline–trinn.

7 TESTING OG VERIFIKASJON

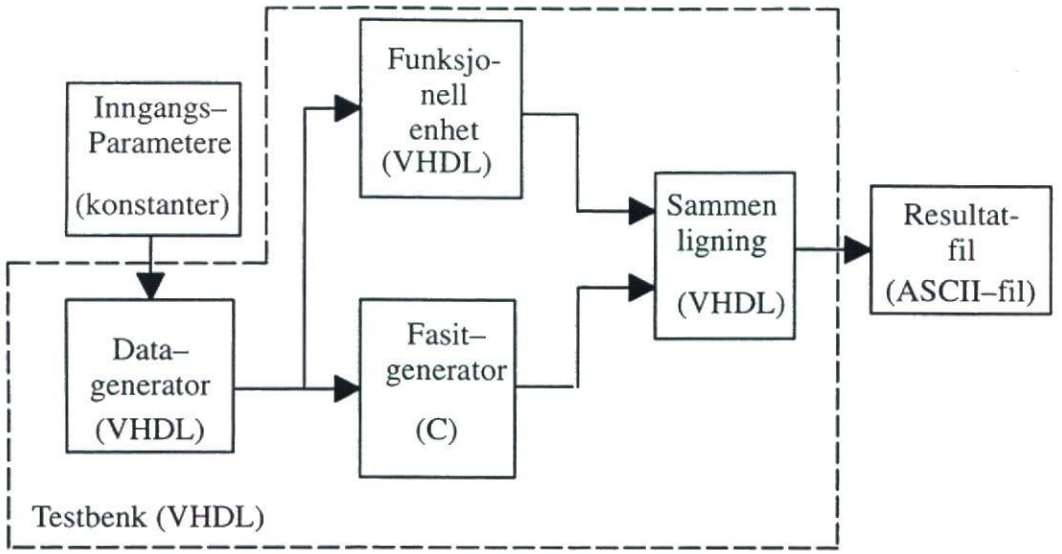
I dette kapitlet beskrives testing og verifikasjon av multiplikasjonskretsen. Vi beskriver hvilke avveininger og valg som er gjort i forhold til teststrategi og testmønstere.

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 (8).

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 arbeidet krever at det foregår mest mulig automatisk uten inngrep fra konstruktøren. Dette var utgangspunktet for utviklingen av test– og verifikasjonssystemet for multiplikasjonskretsen. 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 multiplikasjonskretsen.

Den funksjonelle enheten (multiplikasjonskretsen) 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 multiplikasjonskretsen 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 multiplikasjonskretsen begrensede valg/programmerings muligheter. I all sin enkelhet utfører den én aritmetisk operasjon på to operander og produserer ett resultat. Men, selv ikke for et så enkelt system som multiplikasjonskretsen 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 tallkombinasjoner, som gir spesialtilfeller, er for liten. Derfor bør metoden kompletteres med tester som er skreddersydd for slike spesialtilfeller.

En funksjonalitet som må testes intensivt 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 inngangsparametere

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 parameterene som styrer testmønstergenereringen.

Mantissene kan lages på seks 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"
   ⋮
   Løkkevariabelen = 13 => mantisse = "100000000000000"
   Løkkevariabelen = 14 => mantisse = "000000000000000"

4. Deretter 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 venstre-skift operasjon. Dette kalles "fence of ones".

   Løkkevariabelen = 14 => mantisse = "000000000000000"
   Løkkevariabelen = 15 => mantisse = "000000000000001"
   Løkkevariabelen = 16 => mantisse = "000000000000011"
   Løkkevariabelen = 17 => mantisse = "000000000000111"
   ⋮
   Løkkevariabelen = 28 => mantisse = "011111111111111"
   Løkkevariabelen = 29 => mantisse = "111111111111111"

5. 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 = 29 => mantisse = "111111111111111"
   Løkkevariabelen = 30 => mantisse = "000000000000001"
   Løkkevariabelen = 31 => mantisse = "000000000000010"
   Løkkevariabelen = 32 => mantisse = "000000000000100"
   ⋮
   Løkkevariabelen = 43 => mantisse = "010000000000000"
   Løkkevariabelen = 44 => mantisse = "100000000000000"

6. 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 = 45 => mantisse = "111111111111110"
   Løkkevariabelen = 46 => mantisse = "111111111111101"
   Løkkevariabelen = 46 => mantisse = "111111111111101"
   Løkkevariabelen = 46 => mantisse = "111111111111011"
   ⋮
   Løkkevariabelen = 58 => mantisse = "101111111111111"
   Løkkevariabelen = 59 => mantisse = "011111111111111"

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'='–'.

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 mantissemultiplikasjon, 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 (hexsadesimalt 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 mantisse–generering 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 bruksmiljø. For å beskrive dette har synteseverktøyet et sett med parametere. I dette kapitlet beskrives disse. Synteseresultatene presenteres også.

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

8.1 Parametere som beskriver egenskapene til kretsens omgivelser

I de fleste teknologier påvirker svingninger i temperatur, spenning til strømkilden og prosessparametere 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 prosessparameterene

Variasjoner i prosessparameterene 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 Parametere 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 Parametere 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.

8.4 Spesifisering av kretsens omgivelsesparametere

Synteseverktøyet optimaliserer designet på bakgrunn av de oppgitt omgivelsesparameterene. I utgangspunktet er ingen omgivelsesparametere satt. Prosessparameterene 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 temperatur 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 parameterene 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 operasjonsparametere som vi har brukt under syntese. Verdiene er basert på Alcatel Mietec's Worst Case Industrial (WCIND) prosess parametere.

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 teknologibiblioteket fra prosesshuset. Dette er statiske modeller som bygger på gjennomsnittlig fan–out for en krets av en gitt størrelse (19). Dette gir grunnlaget for å regne ut forsinkelsesbidraget fra lederne. Lasten forbundet med å drive en inverter kalles en standardlast. Denne standardlasten bruker syntesevertøyet som måleenheten for last. Kapasitansen til en inverter er gitt i teknologibiblioteket. 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 multiplikasjonskreten

Resultatene bygger på syntesekjøringer med Synopsys Design Compiler, alle script og innstillinger er gitt i appendiks C. Synteseresultatene for kretsen, som ble pipelinet manuelt, med to pipelinetrinn er gitt i tabell 8.2. Total celleareal for kretsen er 4084 portekvivalenter med et tidskrav på 66 MHz.

Del av kretsen	Antall
Number of ports:	78
Number of nets:	2086
Number of cells:	1927
Number of references:	119
Combinational area:	3417 portekviv.
Noncombinational area:	667 portekviv.
Total cell area:	4084 portekviv.

Tabell 8.2 Synteseresultater for flyttallsmultiplikasjonskretsen, med to pipelinetrinn.

Synteseresultatene for kretsen, som ble pipelinet manuelt, med tre pipelinetrinn er gitt i tabell 8.3. Totalt celleareal er 4487 portekvivalenter når tidskravet på 74 MHz er oppfylt. (66MHz er hastigheten på vår systemklokke, med 74 MHz har vi litt å gå på.)

Del av kretsen	Antall
Number of ports:	78
Number of nets:	2241
Number of cells:	2024
Number of references:	105
Combinational area:	3288 portekviv.
Noncombinational area:	1199 portekviv.
Total cell area:	4487 portekviv.

Tabell 8.3 Syntese resultatene for flyttallsmultiplikasjonskretsen med tre pipelinetrinn.

Vi valgte her kretsen med tre pipelinetrinn for å sikre oss at kretsen ville fungere tilfredsstillende i systemet.

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 multiplikasjons prosessen. Standard VHDL–biblioteker kommenteres ikke.

9.1 Egendefinerte VHDL–biblioteker

For enkelt å ha muligheten til å forandre på ordbredder og sammensetninger av data og kontrollord, 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.1, definert multiplikasjonskretsens 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 ordbredden 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 fpmult.vhd som er gjengitt i appendiks A.2.

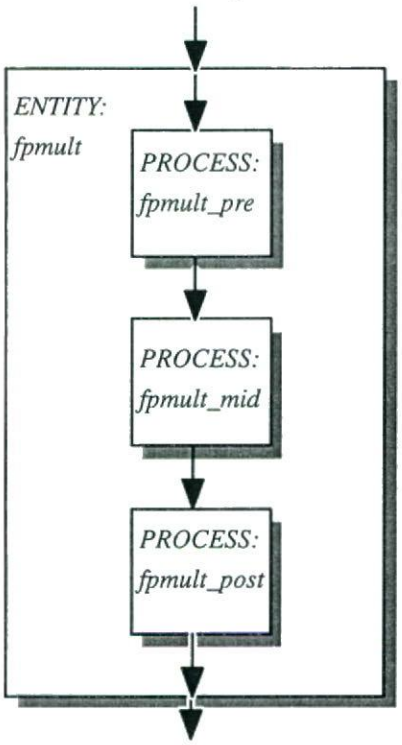
Den beskrevne multiplikasjonskrets, med tre pipelinetrinn, ble det naturlig å dele opp i tre interne *processer,* én for hvert pipelinetrinn. Disse tre *processene* fungerer uavhengig av hverandre og påvirkes kun av signaler utenifra og signaler *processene* imellom. Fordi multiplikasjon er en seriell prosess, 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, globale signaler og alle innsignalene beskrives. Deretter kommer 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 utenomverdenen 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, som så settes 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 Multiplikasjonsenhetens inn– og utgangssignaler

Multiplikasjonskretsen har fem inn– og fire utgangssignaler disse er:

Innsignaler:

- multiplicand, multiplier, clk, resetn og hold

Utsignaler:

- product, mult_ufl, mult_ofl og mult_ivo

Hvis resetn og hold er passive, produserer multiplikasjonsenheten ett resultat pr klokke–periode. Svaret, product, er forsinket med tre klokkeperioder i forhold til inngangs–verdiene, multiplicand og multiplier, da det er 3 pipelinetrinn i kretsen. Når resetn er aktiv, nullstilles alle registere i kretsen. Når hold er aktiv, fryses kretsen.

9.4 Detaljert gjennomgang av VHDL–koden fpmult.vhd

Koden er gjengitt i sin helhet i appendiks A.2. Det følgende er en kortfattet gjennomgang av modulene som inngår i vår implementasjon av flyttallsmultiplikatoren.

Vår implementasjon av flyttallsmultiplikasjonskretsen består av følgende enheter:

- Enhet som sjekker om tallene inn til multiplikasjonskretsen er gyldige.
- Enhet som sørger for at produktet får riktig fortegn.
- Enhet som summer eksponentene til de to multiplikandene.
- Enhet som multipliserer de to multiplikandenes mantisser.
- Normaliseringsenhet.
- Avrundingsenhet.
- Normaliseringsenhet som normaliserer etter avrunding.
- 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* (multiplicand) og *b* (multiplier)
- Utsignaler: *a*, *b*, a_uendelig, b_uendelig, a_nan, b_nan, a_null og b_null, exception, res_v, ivo_v.

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. Hvis vi har et untakstilfelle, settes exeption til én og res_v tilordnes riktig unntaksverdien.

Fortegnsmodul

- Innsignaler: a.sign og b.sign
- Utsignaler: absign

• Funksjon: $a.sign \otimes b.sign$

Enheten beregner produktets fortegn på grunnlag av de to multiplikandenes fortegn.

Eksponent addisjons modul

• Innsignaler: a.exp og b.exp

• Utsignal: exps_v, underflow og overflow

Enheten summerer de to multiplikandenes eksponenter og setter summen på modulens utgang, exps_v. Modulen setter også ut under– og overflow hvis eksponentverdiene skulle tilsi det. Underflow settes hvis den mest signifikante biten i a.exp, b.exp og exps_v er én. Overflow settes hvis den mest signifikante biten i a.exp, b.exp og exps_v er null.

Mantissemultiplikasjon

• Innsignaler: a.mant og b.mant

• Utsignaler: fullproduct

• Funksjon: $a.mant \cdot b.mant$

Enheten utfører en heltallsmultiplikasjon av de to multiplikandenes matisser. Multiplikasjonen utføres i en $16 \cdot 16$ biters binær multiplikasjonsmatrise. Multiplikand a settes inn horisontalt og multiplikand b settes inn vertikalt.

Tilordning av g,r og s

• Innsignaler: fullproduct

• Utsignaler: flags_v_guard, flags_v_round og flags_v_sticky

Enheten tilordner guardbiten til øverste biten i nedre halvdel av fullproduct. Round biten tilordnes verdien til den nest øverste biten i nedre halvdel av fullproduct. Sticky biten tilordnes verdien av en OR–operasjon av de resterende bitene i nedre halvdel av fullproduct.

Normalisering av resultatet

• Innsignaler: fullproduct, flags_v_guard, flags_v_round og flags_v_sticky

• Utsignaler: mant_v, flags_v_guard, flags_v_round og flags_v_sticky

Enheten justerer fullproduct slik at den ledende eneren befinner seg i posisjon *MSB–1*. Det kan her bare forekomme høyreskift. Guard, round og sticky oppdateres også på grunnlag av skiftoperasjonene. Eksponenten til svaret justeres også i henhold til skiftingen av fullproduct.

Avrunding av resultatet

- Innsignaler: mant_v
- Utsignaler: mant_r, carry_msb

Enheten avrunder resultatet etter regler gitt i kapittel 3 og justerer om nødvendig eksponenten.

Normalisering av resultatet etter avrunding

- Innsignaler: mant_r, carry_msb
- Utsignaler: norm_prod

Enheten normaliserer resultatet etter regler gitt i kapittel 3.

Eksponent inkrementering

- Innsignaler: exps_v
- Utsignaler: expadj

Enheten justerer eksponenten på grunnlag av normaliseringene.

Sjekk på gyldig tall ut

- Innsignaler: expadj, normprod, overflow, underflow
- Utsignaler: oflow_v, uflow_v, mantout og expout

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

Tilordning av utganger

- Innsignaler: product, res_v, exception
- Utsignaler: res, mult_ivo, mult_ofl, mult_ufl

Enheten tilordner de interne variablers verdier til utgangene.

10 KONKLUSJON

Vi har i denne rapporten beskrevet en teknologiuavhengig implementasjon av en multiplikasjonskrets 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 teknologibibliotek.

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

Multiplikasjonskretsen, som inneholder tre pipelinetrinn, har en total størrelse på 4487 portekvivalenter. Kretsen med tre pipelinetrinn går på en klokkehastighet på 74 MHz. Vi har også implementert en utgave av kretsen med to pipelinetrinn og med en klokkehastighet på 66 MHz og total størrelse på 4084 porterekvivalenter.

Litteratur

(1) Adams G, Bose B K, Pei L, Wang A (1985): The Design of a Floating–Point Prosessor Unit, Report No. UCB/CSD 86/259, University of California Berkley.

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

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

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

(5) Blom H, Gundersen R: Pipelinet 128–punkters radix–2 FFT–prosessor med vektorlengdeuavhengig ytelse med full utnyttelse av aritmetiske enheter, FFI/RAPPORT 2000/00885, Forsvarets forskningsinstitutt.

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

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

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

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

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

(11) IEEE (1985): IEEE Standard 754–1985 for Binary Floating–Point Arithmetic, IEEE.

(12) IEEE (1987): IEEE Standard 1076–1987 IEEE Standard VHDL Reference Manual, IEEE.

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

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

(15) Mou Z J, Jutand F (1992): "Overturned–Stairs" Adder Trees and Multipler Design, IEEE Transactions on Computers, Vol. 41, No. 8, 940–948.

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

(17) Reusens P (1981): "Fixed–Point High–Speed Parallel Multipliers in VLSI", Springer–Verlag, New York.

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

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

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

(21) Wallace C S (1964): "A Suggestion For a Fast Multiplier", IEEE Transactions on Electronics and Computers, Vol. EC–13, 14–17.

APPENDIKS

A VHDL–KODE FOR FLYTTALLSMULTIPLIKASJONSKRETSEN

Vi vil i de følgende avsnitt gjengi VHDL–kode for multiplikasjonskretsen og det egendefinerte typebiblioteket.

A.1 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;
```

A.2 VHDL–filen fpmult.vhd

Her følger VHDL–koden for multiplikasjonskretsen med tre pipelinetrinn.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

LIBRARY common_lib;
USE common_lib.types.ALL;

ENTITY fpmult IS
   PORT(
     clk          : IN  std_logic; -- system clock
     resetn       : IN  std_logic; -- system reset
     hold         : IN  std_logic; -- hold signal inhibiting changes
     multiplicand : IN  nfloat;    -- argument for multiplication
     multiplier   : IN  nfloat;    -- argument for multiplication
     product      : OUT nfloat;    -- final pipelined product
     mult_ofl     : OUT std_logic; -- overflow from multiplication
     mult_ufl     : OUT std_logic; -- underflow from multiplication
     mult_ivo     : OUT std_logic  -- ivo from multiplication
       );
END fpmult;

ARCHITECTURE pipe3 OF fpmult IS

  constant ZeroExp_c    : std_logic_vector(exphi downto
0):=(others=>'0');
  constant AllOnesExp_c : std_logic_vector(exphi downto
0):=(others=>'1');
```

```
  constant ZMc    : std_logic_vector(manhi downto
0):="1000000000000000";
  constant ZeroMant_c    : std_logic_vector(manhi downto
0):=(others=>'0');
  constant nan_mant : std_logic_vector(manhi downto 0) :=
"1010000000000000";
  constant AllOneVec_c : std_logic_vector(exphi downto 0) :=
(others=>'1');
  constant ZeroVec_c    : std_logic_vector(exphi downto 0) :=
(others=>'0');
  constant ZeroVec      : std_logic_vector(manhi downto 0) :=
(others=>'0');
  constant ExpAllOnesVec_c :std_logic_vector(exphi downto
0):=(others=>'1');
  constant ExpAllZeroVec_c :std_logic_vector(exphi downto
0):=(others=>'0');
  signal res1_s, res2_s: ffloat;
  signal exception1_s,exception2_s : boolean;
  signal absign1_s,absign2_s : std_logic;
  signal ivo1_s,ivo2_s : std_logic;
  signal underflow1_s,underflow2_s : std_logic;
  signal overflow1_s,overflow2_s : std_logic;
  signal ExpS1_s,ExpS2_s  : std_logic_vector(exphi downto 0);
  signal m_8_s, m_9_s, m_10_s: std_logic_vector(30 downto 0);
  signal b11a15_s,b14a13_s,b15a12_s,b14a15_s, b15a14_s, b15a15_s :
std_logic;
  signal s1_s : std_logic_vector(11 downto 0);
  signal b12a14_s,b12a15_s,b13a14_s,b13a15_s,b14a14_s,b15a13_s :
std_logic;
  signal fullproduct_s : std_logic_vector(2*manhi+1 downto 0);

begin
  fpmult_pre: process(clk,resetn)
    variable a            : ffloat;
    variable b            : ffloat;
    variable res_v        : ffloat;
    variable exception    : boolean;
    variable ivo_v        : std_logic;
    variable a_oo, a_nan, b_oo, b_nan, ab_null : std_logic;
    variable Exp1_v      : std_logic_vector(exphi downto 0);
    variable Exp2_v      : std_logic_vector(exphi downto 0);
    variable ExpS_v      : std_logic_vector(exphi downto 0);
    variable underflow   : std_logic;
    variable overflow    : std_logic;
--    variable AddToExp,AddToExp2 : boolean;
    variable absign, nansign  : std_logic;
--array mult
    variable aaa : std_logic_vector(15 downto 0);
    variable bbb : std_logic_vector(15 downto 0);
    variable sss : std_logic_vector(11 downto 0);
    variable m_0, m_1, m_2,m_3,m_4 : std_logic_vector(30 downto 0);
    variable m_5,m_6 : std_logic_vector(30 downto 0);
    variable m_7,m_8,m_9,m_10 : std_logic_vector(30 downto 0);
--    variable m_11,m_12 : std_logic_vector(30 downto 0);
```

```
    variable b0a0, b1a0, b2a0, b3a0, b4a0, b5a0, b6a0, b7a0
:std_logic;
    variable b0a1, b1a1, b2a1, b3a1, b4a1, b5a1, b6a1, b7a1
:std_logic;
    variable b0a2, b1a2, b2a2, b3a2, b4a2, b5a2, b6a2, b7a2
:std_logic;
    variable b0a3, b1a3, b2a3, b3a3, b4a3, b5a3, b6a3, b7a3
:std_logic;
    variable b0a4, b1a4, b2a4, b3a4, b4a4, b5a4, b6a4, b7a4
:std_logic;
    variable b0a5, b1a5, b2a5, b3a5, b4a5, b5a5, b6a5, b7a5
:std_logic;
    variable b0a6, b1a6, b2a6, b3a6, b4a6, b5a6, b6a6, b7a6
:std_logic;
    variable b0a7, b1a7, b2a7, b3a7, b4a7, b5a7, b6a7, b7a7
:std_logic;
    variable b0a8, b1a8, b2a8, b3a8, b4a8, b5a8, b6a8, b7a8
:std_logic;
    variable b0a9, b1a9, b2a9, b3a9, b4a9, b5a9, b6a9, b7a9
:std_logic;
    variable b0a10,b1a10,b2a10,b3a10,b4a10,b5a10,b6a10,b7a10
:std_logic;
    variable b0a11,b1a11,b2a11,b3a11,b4a11,b5a11,b6a11,b7a11
:std_logic;
    variable b0a12,b1a12,b2a12,b3a12,b4a12,b5a12,b6a12,b7a12
:std_logic;
    variable b0a13,b1a13,b2a13,b3a13,b4a13,b5a13,b6a13,b7a13
:std_logic;
    variable b0a14,b1a14,b2a14,b3a14,b4a14,b5a14,b6a14,b7a14
:std_logic;
    variable b0a15,b1a15,b2a15,b3a15,b4a15,b5a15,b6a15,b7a15
:std_logic;
    variable b8a0, b9a0, b10a0, b11a0, b12a0, b13a0, b14a0, b15a0
:std_logic;
    variable b8a1, b9a1, b10a1, b11a1, b12a1, b13a1, b14a1, b15a1
:std_logic;
    variable b8a2, b9a2, b10a2, b11a2, b12a2, b13a2, b14a2, b15a2
:std_logic;
    variable b8a3, b9a3, b10a3, b11a3, b12a3, b13a3, b14a3, b15a3
:std_logic;
    variable b8a4, b9a4, b10a4, b11a4, b12a4, b13a4, b14a4, b15a4
:std_logic;
    variable b8a5, b9a5, b10a5, b11a5, b12a5, b13a5, b14a5, b15a5
:std_logic;
    variable b8a6, b9a6, b10a6, b11a6, b12a6, b13a6, b14a6, b15a6
:std_logic;
    variable b8a7, b9a7, b10a7, b11a7, b12a7, b13a7, b14a7, b15a7
:std_logic;
    variable b8a8, b9a8, b10a8, b11a8, b12a8, b13a8, b14a8, b15a8
:std_logic;
    variable b8a9, b9a9, b10a9, b11a9, b12a9, b13a9, b14a9, b15a9
:std_logic;
    variable b8a10,b9a10,b10a10,b11a10,b12a10,b13a10,b14a10,b15a10
:std_logic;
```

```
    variable b8a11,b9a11,b10a11,b11a11,b12a11,b13a11,b14a11,b15a11
:std_logic;
    variable b8a12,b9a12,b10a12,b11a12,b12a12,b13a12,b14a12,b15a12
:std_logic;
    variable b8a13,b9a13,b10a13,b11a13,b12a13,b13a13,b14a13,b15a13
:std_logic;
    variable b8a14,b9a14,b10a14,b11a14,b12a14,b13a14,b14a14,b15a14
:std_logic;
    variable b8a15,b9a15,b10a15,b11a15,b12a15,b13a15,b14a15,b15a15
:std_logic;
--
  begin
    if (resetn = '0') then
     res1_s.mant  <= (others => '0');
     res1_s.exp   <= (others => '0');
     res1_s.sign  <= '0';
     exception1_s <= false;
     absign1_s    <= '0';
     ivo1_s       <= '0';
     underflow1_s <= '0';
     overflow1_s  <= '0';
     ExpS1_s      <= (others => '0');
     m_8_s        <= (others => '0');
     m_9_s       <= (others => '0');
     m_10_s      <= (others => '0');
     b12a14_s    <= '0';
     b12a15_s    <= '0';
     b13a14_s    <= '0';
     b13a15_s    <= '0';
     b14a14_s    <= '0';
     b11a15_s    <= '0';
     b14a13_s    <= '0';
     b15a12_s    <= '0';
     b14a15_s    <= '0';
     b15a13_s    <= '0';
     b15a14_s    <= '0';
     b15a15_s    <= '0';
     s1_s         <= (others => '0');

    elsif (clk ='1' and clk'event) then
      if (hold = '0') then
     a.mant := '1'&multiplicand.mant;
     a.exp  :=     multiplicand.exp;
     a.sign :=     multiplicand.sign;
     b.mant := '1'&multiplier.mant;
     b.exp  :=     multiplier.exp;
     b.sign :=     multiplier.sign;
     ivo_v := '0';
     res_v.exp   := (others => '0');
     res_v.mant  := (others => '0');
     res_v.sign  := '0';
     exception := false;
     a_oo    := '0';
     a_nan   := '0';
```

```
    b_oo    := '0';
    b_nan   := '0';
    ab_null := '0';
    absign  := '0';
    nansign := '0';
--sign
    absign := (a.Sign xor b.Sign);
--Input ooook???
    if ((a.exp=ZeroExp_c) or (b.exp=ZeroExp_c)) then -- null eller
denorm
      res_v.exp   := ZeroExp_c;
      res_v.mant  := '1' & ZeroMant_c(manhi-1 downto 0);
      res_v.sign  := '0';--absign;
      ivo_v       := '0';
      exception := true;
      ab_null := '1';
    end if;
    if (a.exp=AllOnesExp_c) then --oo eller nan
      if (a.mant=ZMc) then --oo
        res_v.exp   := AllOnesExp_c;
        res_v.mant  := '1' & ZeroMant_c(manhi-1 downto 0);
        res_v.sign  := absign;--a.sign;
        ivo_v       := '0';
        exception := true;
        a_oo := '1';
      else --nan
        res_v.exp   := AllOnesExp_c;
        res_v.mant  := nan_mant;
        res_v.sign  := a.sign;
        nansign     := a.sign;
        ivo_v       := '0';
        exception := true;
        a_nan := '1';
      end if;
    end if;
    if (b.exp=AllOnesExp_c) then --oo eller nan
      if (b.mant=ZMc) then --oo
        res_v.exp   := AllOnesExp_c;
        res_v.mant  := '1' & ZeroMant_c(manhi-1 downto 0);
        res_v.sign  := absign;--b.sign;
        ivo_v       := '0';
        exception := true;
        b_oo := '1';
      else --nan
        res_v.exp   := AllOnesExp_c;
        res_v.mant  := nan_mant;
        res_v.sign  := b.sign;
        nansign     := b.sign;
        ivo_v       := '0';
        exception := true;
        b_nan := '1';
      end if;
    end if;
-------------------------------------------------------------------
```

```
-------
     if ((a_oo = '1' and b_nan = '1') or
         (b_oo = '1' and a_nan = '1')) then
       res_v.exp    := AllOnesExp_c;
       res_v.mant   := nan_mant;
       res_v.sign   := nansign;--b.sign;
       ivo_v        := '0';
       exception := true;
     end if;
     if (a_nan = '1' and b_nan = '1') then
       res_v.exp    := AllOnesExp_c;
       res_v.mant   := nan_mant;
       res_v.sign   := a.sign;
       ivo_v        := '0';
       exception := true;
     end if;
     if ((a_nan = '1' or b_nan = '1') and ab_null = '1') then
       res_v.exp    := AllOnesExp_c;
       res_v.mant   := nan_mant;
--     res_v.sign   := '0';
       ivo_v        := '0';
       exception := true;
     end if;
     if ((a_oo = '1' or b_oo = '1') and ab_null = '1') then
       res_v.exp    := AllOnesExp_c;
       res_v.mant   := nan_mant;
--     res_v.sign   := '0';
       ivo_v        := '1';
       exception := true;
     end if;
--   if (a_oo = '1' and b_oo = '1' ) then
--     res_v.exp    := AllOnesExp_c;
--     res_v.mant   := '1' & ZeroMant_c(manhi-1 downto 0);
--     res_v.sign   := absign;
--     ivo_v        := '0';
--     exception := true;
--   end if;
     if ((a_oo = '1' or b_oo = '1') and
         ((a.exp=ZeroExp_c) or (b.exp=ZeroExp_c))) then
       res_v.exp    := AllOnesExp_c;
       res_v.mant   := nan_mant;
       res_v.sign   := '0';
       ivo_v        := '1';
       exception := true;
     end if;


--exp add
     Exp1_v(Exp1_v'high) := not a.exp(a.exp'high);
     Exp1_v(Exp1_v'high-1 downto Exp1_v'low) :=
       a.exp(a.exp'high-1 downto a.exp'low);
     Exp2_v(Exp2_v'high) := not b.exp(b.exp'high);
     Exp2_v(Exp2_v'high-1 downto Exp2_v'low) :=
```

```
      b.exp(b.exp'high-1 downto b.exp'low);
    --2 & 4
    ExpS_v := unsigned(Exp1_v) + unsigned(Exp2_v) + '1';
    --3
    ExpS_v(ExpS_v'high) := not ExpS_v(ExpS_v'high);
    --Oflw and Uflw check
    underflow := '0';
    overflow := '0';
    if ((Exp1_v(Exp1_v'high)='1') and
        (Exp2_v(Exp2_v'high)='1') and
        (ExpS_v(ExpS_v'high)='1')) then
      underflow := '1';
    elsif (((Exp1_v(Exp1_v'high)='0') and
         (Exp2_v(Exp2_v'high)='0') and
         (ExpS_v(ExpS_v'high)='0')) or
           (ExpS_v=AllOneVec_c)) then
      overflow := '1';
    end if;

--(s)mult
--   ap := unsigned(a.mant);
--   bp := unsigned(b.mant);
--   fullproduct_u := ap*bp;
--   fullproduct := std_logic_vector(fullproduct_u);

    aaa := a.mant;
    bbb := b.mant;
--skrive um10 xor  se MANO

      b0a0  := bbb(0) and  aaa(0);
      b0a1  := bbb(0) and  aaa(1);
      b0a2  := bbb(0) and  aaa(2);
      b0a3  := bbb(0) and  aaa(3);
      b0a4  := bbb(0) and  aaa(4);
      b0a5  := bbb(0) and  aaa(5);
      b0a6  := bbb(0) and  aaa(6);
      b0a7  := bbb(0) and  aaa(7);
      b0a8  := bbb(0) and  aaa(8);
      b0a9  := bbb(0) and  aaa(9);
      b0a10 := bbb(0) and aaa(10);
      b0a11 := bbb(0) and aaa(11);
      b0a12 := bbb(0) and aaa(12);
      b0a13 := bbb(0) and aaa(13);
      b0a14 := bbb(0) and aaa(14);
      b0a15 := bbb(0) and aaa(15);


      b1a0 := bbb(1) and aaa(0);
      b1a1 := bbb(1) and aaa(1);
      b1a2 := bbb(1) and aaa(2);
      b1a3 := bbb(1) and aaa(3);
      b1a4 := bbb(1) and aaa(4);
      b1a5 := bbb(1) and aaa(5);
      b1a6 := bbb(1) and aaa(6);
```

```
        b1a7 := bbb(1) and aaa(7);
        b1a8 := bbb(1) and aaa(8);
        b1a9 := bbb(1) and aaa(9);
        b1a10 := bbb(1) and aaa(10);
        b1a11 := bbb(1) and aaa(11);
        b1a12 := bbb(1) and aaa(12);
        b1a13 := bbb(1) and aaa(13);
        b1a14 := bbb(1) and aaa(14);
        b1a15 := bbb(1) and aaa(15);

        b2a0 := bbb(2) and aaa(0);
        b2a1 := bbb(2) and aaa(1);
        b2a2 := bbb(2) and aaa(2);
        b2a3 := bbb(2) and aaa(3);
        b2a4 := bbb(2) and aaa(4);
        b2a5 := bbb(2) and aaa(5);
        b2a6 := bbb(2) and aaa(6);
        b2a7 := bbb(2) and aaa(7);
        b2a8 := bbb(2) and aaa(8);
        b2a9 := bbb(2) and aaa(9);
        b2a10 := bbb(2) and aaa(10);
        b2a11 := bbb(2) and aaa(11);
        b2a12 := bbb(2) and aaa(12);
        b2a13 := bbb(2) and aaa(13);
        b2a14 := bbb(2) and aaa(14);
        b2a15 := bbb(2) and aaa(15);

--B-0-1-2
        sss(0) :=   b0a0;
        sss(1) :=  (b0a1) xor b1a0;
        m_0(0) :=  (b0a1) and b1a0;

        m_0(1) :=  (b0a2) xor b1a1 xor (b2a0);
        m_0(2) := ((b0a2  and b1a1) or
             (b0a2  and b2a0) or
             (b1a1  and b2a0));

        m_0(3) :=  (b0a3) xor (b1a2) xor (b2a1);
        m_0(4) := ((b0a3  and b1a2) or
             (b0a3  and b2a1) or
             (b1a2  and b2a1));

        m_0(5) :=  (b0a4) xor (b1a3) xor (b2a2);
        m_0(6) := ((b0a4  and b1a3) or
             (b0a4  and b2a2) or
             (b1a3  and b2a2));

        m_0(7) :=  (b0a5) xor (b1a4) xor (b2a3);
        m_0(8) := ((b0a5  and b1a4) or
             (b0a5  and b2a3) or
             (b1a4  and b2a3));

        m_0(9) :=  (b0a6)  xor (b1a5) xor (b2a4);
        m_0(10) := ((b0a6  and b1a5) or
```

```
        (b0a6  and b2a4) or
        (b1a5  and b2a4));

    m_0(11) := (b0a7) xor (b1a6) xor (b2a5);
    m_0(12):= ((b0a7  and b1a6) or
        (b0a7  and b2a5) or
        (b1a6  and b2a5));

    m_0(13):=  (b0a8) xor (b1a7) xor (b2a6);
    m_0(14):= ((b0a8  and b1a7) or
        (b0a8  and b2a6) or
        (b1a7  and b2a6));

    m_0(15):=  (b0a9) xor (b1a8) xor (b2a7);
    m_0(16):= ((b0a9  and b1a8) or
        (b0a9  and b2a7) or
        (b1a8  and b2a7));

    m_0(17):=  (b0a10) xor (b1a9) xor (b2a8);
    m_0(18):= ((b0a10 and b1a9) or
        (b0a10 and b2a8) or
        (b1a9 and b2a8));

    m_0(19):=  (b0a11) xor (b1a10) xor (b2a9);
    m_0(20):= ((b0a11 and b1a10) or
        (b0a11 and b2a9) or
        (b1a10 and b2a9));

    m_0(21):=  (b0a12) xor (b1a11) xor (b2a10);
    m_0(22):= ((b0a12 and b1a11) or
        (b0a12 and b2a10) or
        (b1a11 and b2a10));

    m_0(23):=  (b0a13) xor (b1a12) xor (b2a11);
    m_0(24):= ((b0a13 and b1a12) or
        (b0a13 and b2a11) or
        (b1a12 and b2a11));

    m_0(25):=  (b0a14) xor (b1a13) xor (b2a12);
    m_0(26):= ((b0a14 and b1a13) or
        (b0a14 and b2a12) or
        (b1a13 and b2a12));

    m_0(27):=  (b0a15) xor (b1a14) xor (b2a13);
    m_0(28):= ((b0a15 and b1a14) or
        (b0a15 and b2a13) or
        (b1a14 and b2a13));

    m_0(29) := (b1a15) xor (b2a14);
    m_0(30) := (b1a15) and (b2a14);

--B-3-4-5

    b3a0  := bbb(3) and  aaa(0);
```

```
b3a1  := bbb(3) and  aaa(1);
b3a2  := bbb(3) and  aaa(2);
b3a3  := bbb(3) and  aaa(3);
b3a4  := bbb(3) and  aaa(4);
b3a5  := bbb(3) and  aaa(5);
b3a6  := bbb(3) and  aaa(6);
b3a7  := bbb(3) and  aaa(7);
b3a8  := bbb(3) and  aaa(8);
b3a9  := bbb(3) and  aaa(9);
b3a10 := bbb(3) and aaa(10);
b3a11 := bbb(3) and aaa(11);
b3a12 := bbb(3) and aaa(12);
b3a13 := bbb(3) and aaa(13);
b3a14 := bbb(3) and aaa(14);
b3a15 := bbb(3) and aaa(15);


b4a0  := bbb(4) and aaa(0);
b4a1  := bbb(4) and aaa(1);
b4a2  := bbb(4) and aaa(2);
b4a3  := bbb(4) and aaa(3);
b4a4  := bbb(4) and aaa(4);
b4a5  := bbb(4) and aaa(5);
b4a6  := bbb(4) and aaa(6);
b4a7  := bbb(4) and aaa(7);
b4a8  := bbb(4) and aaa(8);
b4a9  := bbb(4) and aaa(9);
b4a10 := bbb(4) and aaa(10);
b4a11 := bbb(4) and aaa(11);
b4a12 := bbb(4) and aaa(12);
b4a13 := bbb(4) and aaa(13);
b4a14 := bbb(4) and aaa(14);
b4a15 := bbb(4) and aaa(15);

b5a0  := bbb(5) and aaa(0);
b5a1  := bbb(5) and aaa(1);
b5a2  := bbb(5) and aaa(2);
b5a3  := bbb(5) and aaa(3);
b5a4  := bbb(5) and aaa(4);
b5a5  := bbb(5) and aaa(5);
b5a6  := bbb(5) and aaa(6);
b5a7  := bbb(5) and aaa(7);
b5a8  := bbb(5) and aaa(8);
b5a9  := bbb(5) and aaa(9);
b5a10 := bbb(5) and aaa(10);
b5a11 := bbb(5) and aaa(11);
b5a12 := bbb(5) and aaa(12);
b5a13 := bbb(5) and aaa(13);
b5a14 := bbb(5) and aaa(14);
b5a15 := bbb(5) and aaa(15);

--B-3-4-5

sss(2) :=   m_0(0) xor m_0(1);
```

```
m_1(0) :=   m_0(0) and m_0(1);

m_1(1) :=   m_0(3) xor b3a0;
m_1(2) :=   m_0(3) and b3a0;

m_1(3) :=   b3a1   xor b4a0;
m_1(4) :=   b3a1   and b4a0;

m_1(5) :=  (b3a2) xor (b4a1) xor (b5a0);
m_1(6) := ((b3a2  and b4a1) or
     (b3a2  and b5a0) or
     (b4a1  and b5a0));

m_1(7) :=  (b3a3) xor (b4a2) xor (b5a1);
m_1(8) := ((b3a3  and b4a2) or
     (b3a3  and b5a1) or
     (b4a2  and b5a1));

m_1(9) :=  (b3a4) xor (b4a3) xor (b5a2);
m_1(10):= ((b3a4  and b4a3) or
     (b3a4  and b5a2) or
     (b4a3  and b5a2));

m_1(11):=  (b3a5) xor (b4a4) xor (b5a3);
m_1(12) := ((b3a5  and b4a4) or
     (b3a5  and b5a3) or
     (b4a4  and b5a3));

m_1(13) :=  (b3a6)  xor (b4a5) xor (b5a4);
m_1(14) := ((b3a6  and b4a5) or
      (b3a6  and b5a4) or
      (b4a5  and b5a4));

m_1(15) := (b3a7) xor (b4a6) xor (b5a5);
m_1(16):= ((b3a7  and b4a6) or
     (b3a7  and b5a5) or
     (b4a6  and b5a5));

m_1(17):=  (b3a8) xor (b4a7) xor (b5a6);
m_1(18):= ((b3a8  and b4a7) or
     (b3a8  and b5a6) or
     (b4a7  and b5a6));

m_1(19):=  (b3a9) xor (b4a8) xor (b5a7);
m_1(20):= ((b3a9  and b4a8) or
     (b3a9  and b5a7) or
     (b4a8  and b5a7));

m_1(21):=  (b3a10) xor (b4a9) xor (b5a8);
m_1(22):= ((b3a10 and b4a9) or
     (b3a10 and b5a8) or
     (b4a9  and b5a8));

m_1(23):=  (b3a11) xor (b4a10) xor (b5a9);
```

```
        m_1(24):= ((b3a11 and b4a10) or
             (b3a11 and b5a9) or
             (b4a10 and b5a9));

        m_1(25):=  (b3a12) xor (b4a11) xor (b5a10);
        m_1(26):= ((b3a12 and b4a11) or
             (b3a12 and b5a10) or
             (b4a11 and b5a10));

        m_1(27):=  (b3a13) xor (b4a12) xor (b5a11);
        m_1(28):= ((b3a13 and b4a12) or
             (b3a13 and b5a11) or
             (b4a12 and b5a11));

        m_1(29):=  (b3a14) xor (b4a13) xor (b5a12);
        m_1(30):= ((b3a14 and b4a13) or
             (b3a14 and b5a12) or
             (b4a13 and b5a12));

-- 0-1-2 + 3-4-5

        sss(3) :=   m_0(2)  xor m_1(0) xor m_1(1);
        m_2(0) := ((m_0(2)  and m_1(0)) or
             (m_0(2)  and m_1(1)) or
             (m_1(0)  and m_1(1)));

        m_2(1) :=   m_0(4)  xor m_0(5) xor m_1(3);
        m_2(2) := ((m_0(4)  and m_0(5)) or
             (m_0(4)  and m_1(3)) or
             (m_0(5)  and m_1(3)));

        m_2(3) :=   m_0(6)  xor m_0(7) xor m_1(5);
        m_2(4) := ((m_0(6)  and m_0(7)) or
             (m_0(6)  and m_1(5)) or
             (m_0(7)  and m_1(5)));

        m_2(5) :=   m_0(8)  xor m_0(9) xor m_1(7);
        m_2(6) := ((m_0(8)  and m_0(9)) or
             (m_0(8)  and m_1(7)) or
             (m_0(9)  and m_1(7)));

        m_2(7) :=   m_0(10)  xor m_0(11) xor m_1(9);
        m_2(8) := ((m_0(10)  and m_0(11)) or
             (m_0(10)  and m_1(9)) or
             (m_0(11)  and m_1(9)));

        m_2(9) :=   m_0(12)  xor m_0(13) xor m_1(11);
        m_2(10):= ((m_0(12)  and m_0(13)) or
             (m_0(12)  and m_1(11)) or
             (m_0(13)  and m_1(11)));

        m_2(11):=   m_0(14)  xor m_0(15) xor m_1(13);
        m_2(12):= ((m_0(14)  and m_0(15)) or
             (m_0(14)  and m_1(13)) or
```

```
     (m_0(15)  and m_1(13)));

m_2(13):=   m_0(16)  xor m_0(17) xor m_1(15);
m_2(14):= ((m_0(16)  and m_0(17)) or
     (m_0(16)  and m_1(15)) or
     (m_0(17)  and m_1(15)));

m_2(15):=   m_0(18)  xor m_0(19) xor m_1(17);
m_2(16):= ((m_0(18)  and m_0(19)) or
     (m_0(18)  and m_1(17)) or
     (m_0(19)  and m_1(17)));

m_2(17):=   m_0(20)  xor m_0(21) xor m_1(19);
m_2(18):= ((m_0(20)  and m_0(21)) or
     (m_0(20)  and m_1(19)) or
     (m_0(21)  and m_1(19)));

m_2(19):=   m_0(22)  xor m_0(23) xor m_1(21);
m_2(20):= ((m_0(22)  and m_0(23)) or
     (m_0(22)  and m_1(21)) or
     (m_0(23)  and m_1(21)));

m_2(21):=   m_0(24)  xor m_0(25) xor m_1(23);
m_2(22):= ((m_0(24)  and m_0(25)) or
     (m_0(24)  and m_1(23)) or
     (m_0(25)  and m_1(23)));

m_2(23):=   m_0(26)  xor m_0(27) xor m_1(25);
m_2(24):= ((m_0(26)  and m_0(27)) or
     (m_0(26)  and m_1(25)) or
     (m_0(27)  and m_1(25)));

m_2(25):=   m_0(28)  xor m_0(29) xor m_1(27);
m_2(26):= ((m_0(28)  and m_0(29)) or
     (m_0(28)  and m_1(27)) or
     (m_0(29)  and m_1(27)));

m_2(27):=   m_0(30)  xor b2a15 xor m_1(29);
m_2(28):= ((m_0(30)  and b2a15) or
     (m_0(30)  and m_1(29)) or
     (b2a15    and m_1(29)));

m_2(29):=   b3a15  xor b4a14 xor b5a13;
m_2(30):= ((b3a15  and b4a14) or
     (b3a15  and b5a13) or
     (b4a14  and b5a13));


--B-6-7-8
     b6a0  := bbb(6) and  aaa(0);
     b6a1  := bbb(6) and  aaa(1);
     b6a2  := bbb(6) and  aaa(2);
     b6a3  := bbb(6) and  aaa(3);
     b6a4  := bbb(6) and  aaa(4);
```

```
b6a5  := bbb(6) and  aaa(5);
b6a6  := bbb(6) and  aaa(6);
b6a7  := bbb(6) and  aaa(7);
b6a8  := bbb(6) and  aaa(8);
b6a9  := bbb(6) and  aaa(9);
b6a10 := bbb(6) and aaa(10);
b6a11 := bbb(6) and aaa(11);
b6a12 := bbb(6) and aaa(12);
b6a13 := bbb(6) and aaa(13);
b6a14 := bbb(6) and aaa(14);
b6a15 := bbb(6) and aaa(15);


b7a0 := bbb(7) and aaa(0);
b7a1 := bbb(7) and aaa(1);
b7a2 := bbb(7) and aaa(2);
b7a3 := bbb(7) and aaa(3);
b7a4 := bbb(7) and aaa(4);
b7a5 := bbb(7) and aaa(5);
b7a6 := bbb(7) and aaa(6);
b7a7 := bbb(7) and aaa(7);
b7a8 := bbb(7) and aaa(8);
b7a9 := bbb(7) and aaa(9);
b7a10 := bbb(7) and aaa(10);
b7a11 := bbb(7) and aaa(11);
b7a12 := bbb(7) and aaa(12);
b7a13 := bbb(7) and aaa(13);
b7a14 := bbb(7) and aaa(14);
b7a15 := bbb(7) and aaa(15);

b8a0 := bbb(8) and aaa(0);
b8a1 := bbb(8) and aaa(1);
b8a2 := bbb(8) and aaa(2);
b8a3 := bbb(8) and aaa(3);
b8a4 := bbb(8) and aaa(4);
b8a5 := bbb(8) and aaa(5);
b8a6 := bbb(8) and aaa(6);
b8a7 := bbb(8) and aaa(7);
b8a8 := bbb(8) and aaa(8);
b8a9 := bbb(8) and aaa(9);
b8a10 := bbb(8) and aaa(10);
b8a11 := bbb(8) and aaa(11);
b8a12 := bbb(8) and aaa(12);
b8a13 := bbb(8) and aaa(13);
b8a14 := bbb(8) and aaa(14);
b8a15 := bbb(8) and aaa(15);

sss(4) :=   m_2(0) xor m_2(1) xor m_1(2);
m_3(0) := ((m_2(0) and m_2(1)) or
     (m_2(0) and m_1(2)) or
     (m_2(1) and m_1(2)));

m_3(1) :=   m_1(4) xor m_2(3);
m_3(2) :=   m_1(4) and m_2(3);
```

```
m_3(3) :=   m_1(6) xor m_2(5) xor b6a0;
m_3(4) := ((m_1(6) and m_2(5)) or
     (m_1(6) and b6a0) or
     (m_2(5) and b6a0));

m_3(5) :=   b6a1 xor b7a0;
m_3(6) :=   b6a1 and b7a0;

m_3(7) :=  (b6a2) xor (b7a1) xor (b8a0);
m_3(8) := ((b6a2  and b7a1) or
     (b6a2  and b8a0) or
     (b7a1  and b8a0));

m_3(9) :=  (b6a3) xor (b7a2) xor (b8a1);
m_3(10):= ((b6a3  and b7a2) or
     (b6a3  and b8a1) or
     (b7a2  and b8a1));

m_3(11):=  (b6a4) xor (b7a3) xor (b8a2);
m_3(12):= ((b6a4  and b7a3) or
     (b6a4  and b8a2) or
     (b7a3  and b8a2));

m_3(13):=  (b6a5) xor (b7a4) xor (b8a3);
m_3(14):= ((b6a5  and b7a4) or
     (b6a5  and b8a3) or
     (b7a4  and b8a3));

m_3(15) :=  (b6a6)  xor (b7a5) xor (b8a4);
m_3(16) := ((b6a6  and b7a5) or
      (b6a6  and b8a4) or
      (b7a5  and b8a4));

m_3(17) := (b6a7) xor (b7a6) xor (b8a5);
m_3(18):= ((b6a7  and b7a6) or
      (b6a7  and b8a5) or
      (b7a6  and b8a5));

m_3(19):=  (b6a8) xor (b7a7) xor (b8a6);
m_3(20):= ((b6a8  and b7a7) or
      (b6a8  and b8a6) or
      (b7a7  and b8a6));

m_3(21):=  (b6a9) xor (b7a8) xor (b8a7);
m_3(22):= ((b6a9  and b7a8) or
     (b6a9  and b8a7) or
     (b7a8  and b8a7));

m_3(23):=  (b6a10) xor (b7a9) xor (b8a8);
m_3(24):= ((b6a10 and b7a9) or
     (b6a10 and b8a8) or
     (b7a9 and b8a8));
```

```
        m_3(25):=  (b6a11) xor (b7a10) xor (b8a9);
        m_3(26):= ((b6a11 and b7a10) or
             (b6a11 and b8a9) or
             (b7a10 and b8a9));

        m_3(27):=  (b6a12) xor (b7a11) xor (b8a10);
        m_3(28):= ((b6a12 and b7a11) or
             (b6a12 and b8a10) or
             (b7a11 and b8a10));

        m_3(29):=  (b6a13) xor (b7a12) xor (b8a11);
        m_3(30):= ((b6a13 and b7a12) or
             (b6a13 and b8a11) or
             (b7a12 and b8a11));

-- 0-1-2 + 3-4-5 + 6-7-8

        sss(5) :=   m_2(2)  xor m_3(0) xor m_3(1);
        m_4(0) := ((m_2(2)  and m_3(0)) or
             (m_2(2)  and m_3(1)) or
             (m_3(0)  and m_3(1)));

        m_4(1) :=   m_2(4)  xor m_3(3);
        m_4(2) :=   m_2(4)  and m_3(3);


        m_4(3) :=   m_1(8)  xor m_2(7) xor m_3(5);
        m_4(4) := ((m_1(8)  and m_2(7)) or
             (m_1(8)  and m_3(5)) or
             (m_2(7)  and m_3(5)));

        m_4(5) :=   m_1(10)  xor m_3(6) xor m_3(7);
        m_4(6) := ((m_1(10)  and m_3(6)) or
             (m_1(10)  and m_3(7)) or
             (m_3(6)   and m_3(7)));

        m_4(7) :=   m_1(12)  xor m_3(8) xor m_3(9);
        m_4(8) := ((m_1(12)  and m_3(8)) or
             (m_1(12)  and m_3(9)) or
             (m_3(8)   and m_3(9)));

        m_4(9) :=   m_1(14)  xor m_3(10) xor m_3(11);
        m_4(10):= ((m_1(14)  and m_3(10)) or
             (m_1(14)  and m_3(11)) or
             (m_3(10)  and m_3(11)));

        m_4(11):=   m_1(16)  xor m_3(12) xor m_3(13);
        m_4(12):= ((m_1(16)  and m_3(12)) or
             (m_1(16)  and m_3(13)) or
             (m_3(12)  and m_3(13)));

        m_4(13):=   m_1(18)  xor m_3(14) xor m_3(15);
        m_4(14):= ((m_1(18)  and m_3(14)) or
             (m_1(18)  and m_3(15)) or
```

```
        (m_3(14)  and m_3(15)));

    m_4(15):=   m_1(20)  xor m_3(16) xor m_3(17);
    m_4(16):= ((m_1(20)  and m_3(16)) or
        (m_1(20)  and m_3(17)) or
        (m_3(16)  and m_3(17)));

    m_4(17):=   m_1(22)  xor m_3(18) xor m_3(19);
    m_4(18):= ((m_1(22)  and m_3(18)) or
        (m_1(22)  and m_3(19)) or
        (m_3(18)  and m_3(19)));

    m_4(19):=   m_1(24)  xor m_3(20) xor m_3(21);
    m_4(20):= ((m_1(24)  and m_3(20)) or
        (m_1(24)  and m_3(21)) or
        (m_3(20)  and m_3(21)));

    m_4(21):=   m_1(26)  xor m_3(22) xor m_3(23);
    m_4(22):= ((m_1(26)  and m_3(22)) or
        (m_1(26)  and m_3(23)) or
        (m_3(22)  and m_3(23)));

    m_4(23):=   m_1(28)  xor m_3(24) xor m_3(25);
    m_4(24):= ((m_1(28)  and m_3(24)) or
        (m_1(28)  and m_3(25)) or
        (m_3(24)  and m_3(25)));

    m_4(25):=   m_1(30)  xor m_3(26) xor m_3(27);
    m_4(26):= ((m_1(30)  and m_3(26)) or
        (m_1(30)  and m_3(27)) or
        (m_3(26)  and m_3(27)));

    m_4(27):=   m_3(28)  xor m_3(29) xor b4a15;
    m_4(28):= ((m_3(28)  and m_3(29)) or
        (m_3(28)  and b4a15) or
        (m_3(29)  and b4a15));

    m_4(29):=   m_3(30)  xor b7a13 xor b8a12;
    m_4(30):= ((m_3(30)  and b7a13) or
        (m_3(30)  and b8a12) or
        (b7a13    and b8a12));

-- 0-1-2 + 3-4-5 + 6-7-8 II

    sss(6) :=   m_3(2)  xor m_4(0) xor m_4(1);
    m_5(0) := ((m_3(2)  and m_4(0)) or
        (m_3(2)  and m_4(1)) or
        (m_4(0)  and m_4(1)));

    m_5(1) :=   m_2(6)  xor m_3(4)  xor m_4(3);
    m_5(2) := ((m_2(6)  and m_3(4)) or
        (m_2(6)  and m_4(3)) or
        (m_3(4)  and m_4(3)));
```

```
m_5(3) :=   m_2(8)  xor m_2(9) xor m_4(5);
m_5(4) := ((m_2(8)  and m_2(9)) or
     (m_2(8)  and m_4(5)) or
     (m_2(9)  and m_4(5)));

m_5(5) :=   m_2(10)  xor m_2(11) xor m_4(7);
m_5(6) := ((m_2(10)  and m_2(11)) or
     (m_2(10)  and m_4(7)) or
     (m_2(11)  and m_4(7)));

m_5(7) :=   m_2(12)  xor m_2(13) xor m_4(9);
m_5(8) := ((m_2(12)  and m_2(13)) or
     (m_2(12)  and m_4(9)) or
     (m_2(13)  and m_4(9)));

m_5(9) :=   m_2(14)  xor m_2(15) xor m_4(11);
m_5(10):= ((m_2(14)  and m_2(15)) or
     (m_2(14)  and m_4(11)) or
     (m_2(15)  and m_4(11)));

m_5(11):=   m_2(16)  xor m_2(17) xor m_4(13);
m_5(12):= ((m_2(16)  and m_2(17)) or
     (m_2(16)  and m_4(13)) or
     (m_2(17)  and m_4(13)));

m_5(13):=   m_2(18)  xor m_2(19) xor m_4(15);
m_5(14):= ((m_2(18)  and m_2(19)) or
     (m_2(18)  and m_4(15)) or
     (m_2(19)  and m_4(15)));

m_5(15):=   m_2(20)  xor m_2(21) xor m_4(17);
m_5(16):= ((m_2(20)  and m_2(21)) or
     (m_2(20)  and m_4(17)) or
     (m_2(21)  and m_4(17)));

m_5(17):=   m_2(22)  xor m_2(23) xor m_4(19);
m_5(18):= ((m_2(22)  and m_2(23)) or
     (m_2(22)  and m_4(19)) or
     (m_2(23)  and m_4(19)));

m_5(19):=   m_2(24)  xor m_2(25) xor m_4(21);
m_5(20):= ((m_2(24)  and m_2(25)) or
     (m_2(24)  and m_4(21)) or
     (m_2(25)  and m_4(21)));

m_5(21):=   m_2(26)  xor m_2(27) xor m_4(23);
m_5(22):= ((m_2(26)  and m_2(27)) or
     (m_2(26)  and m_4(23)) or
     (m_2(27)  and m_4(23)));

m_5(23):=   m_2(28)  xor m_2(29) xor m_4(25);
m_5(24):= ((m_2(28)  and m_2(29)) or
     (m_2(28)  and m_4(25)) or
     (m_2(29)  and m_4(25)));
```

```
      m_5(25):=   m_2(30)  xor b5a14 xor m_4(27);
      m_5(26):= ((m_2(30)  and b5a14) or
           (m_2(30)  and m_4(27)) or
           (b5a14    and m_4(27)));

      m_5(27):=   m_4(29)  xor b5a15 xor b6a14;
      m_5(28):= ((m_4(29)  and b5a15) or
           (m_4(29)  and b6a14) or
           (b5a15    and b6a14));

      m_5(29):=   b6a15  xor b7a14 xor b8a13;
      m_5(30):= ((b6a15  and b7a14) or
           (b6a15  and b8a13) or
           (b7a14  and b8a13));

--B-9-10-11
      b9a0  := bbb(9) and  aaa(0);
      b9a1  := bbb(9) and  aaa(1);
      b9a2  := bbb(9) and  aaa(2);
      b9a3  := bbb(9) and  aaa(3);
      b9a4  := bbb(9) and  aaa(4);
      b9a5  := bbb(9) and  aaa(5);
      b9a6  := bbb(9) and  aaa(6);
      b9a7  := bbb(9) and  aaa(7);
      b9a8  := bbb(9) and  aaa(8);
      b9a9  := bbb(9) and  aaa(9);
      b9a10 := bbb(9) and aaa(10);
      b9a11 := bbb(9) and aaa(11);
      b9a12 := bbb(9) and aaa(12);
      b9a13 := bbb(9) and aaa(13);
      b9a14 := bbb(9) and aaa(14);
      b9a15 := bbb(9) and aaa(15);


      b10a0 := bbb(10) and aaa(0);
      b10a1 := bbb(10) and aaa(1);
      b10a2 := bbb(10) and aaa(2);
      b10a3 := bbb(10) and aaa(3);
      b10a4 := bbb(10) and aaa(4);
      b10a5 := bbb(10) and aaa(5);
      b10a6 := bbb(10) and aaa(6);
      b10a7 := bbb(10) and aaa(7);
      b10a8 := bbb(10) and aaa(8);
      b10a9 := bbb(10) and aaa(9);
      b10a10 := bbb(10) and aaa(10);
      b10a11 := bbb(10) and aaa(11);
      b10a12 := bbb(10) and aaa(12);
      b10a13 := bbb(10) and aaa(13);
      b10a14 := bbb(10) and aaa(14);
      b10a15 := bbb(10) and aaa(15);

      b11a0 := bbb(11) and aaa(0);
      b11a1 := bbb(11) and aaa(1);
```

```
b11a2 := bbb(11) and aaa(2);
b11a3 := bbb(11) and aaa(3);
b11a4 := bbb(11) and aaa(4);
b11a5 := bbb(11) and aaa(5);
b11a6 := bbb(11) and aaa(6);
b11a7 := bbb(11) and aaa(7);
b11a8 := bbb(11) and aaa(8);
b11a9 := bbb(11) and aaa(9);
b11a10 := bbb(11) and aaa(10);
b11a11 := bbb(11) and aaa(11);
b11a12 := bbb(11) and aaa(12);
b11a13 := bbb(11) and aaa(13);
b11a14 := bbb(11) and aaa(14);
b11a15 := bbb(11) and aaa(15);

sss(7) :=   m_5(0) xor m_5(1) xor m_4(2);
m_6(0) := ((m_5(0) and m_5(1)) or
     (m_5(0) and m_4(2)) or
     (m_5(1) and m_4(2)));

m_6(1) :=   m_4(4) xor m_5(3);
m_6(2) :=   m_4(4) and m_5(3);

m_6(3) :=   m_4(6) xor m_5(5) xor b9a0;
m_6(4) := ((m_4(6) and m_5(5)) or
     (m_4(6) and b9a0) or
     (m_5(5) and b9a0));

m_6(5) :=   b9a1 xor b10a0;
m_6(6) :=   b9a1 and b10a0;

m_6(7) :=  (b9a2) xor (b10a1) xor (b11a0);
m_6(8) := ((b9a2  and b10a1) or
     (b9a2  and b11a0) or
     (b10a1  and b11a0));

m_6(9) :=  (b9a3) xor (b10a2) xor (b11a1);
m_6(10):= ((b9a3  and b10a2) or
     (b9a3  and b11a1) or
     (b10a2  and b11a1));

m_6(11):=  (b9a4) xor (b10a3) xor (b11a2);
m_6(12):= ((b9a4  and b10a3) or
     (b9a4  and b11a2) or
     (b10a3  and b11a2));

m_6(13):=  (b9a5) xor (b10a4) xor (b11a3);
m_6(14):= ((b9a5  and b10a4) or
     (b9a5  and b11a3) or
     (b10a4  and b11a3));

m_6(15) :=  (b9a6)  xor (b10a5) xor (b11a4);
m_6(16) := ((b9a6  and b10a5) or
      (b9a6  and b11a4) or
```

```
      (b10a5  and b11a4));

m_6(17) := (b9a7) xor (b10a6) xor (b11a5);
m_6(18):= ((b9a7  and b10a6) or
      (b9a7  and b11a5) or
      (b10a6  and b11a5));

m_6(19):=  (b9a8) xor (b10a7) xor (b11a6);
m_6(20):= ((b9a8  and b10a7) or
      (b9a8  and b11a6) or
      (b10a7  and b11a6));

m_6(21):=  (b9a9) xor (b10a8) xor (b11a7);
m_6(22):= ((b9a9  and b10a8) or
     (b9a9  and b11a7) or
     (b10a8  and b11a7));

m_6(23):=  (b9a10) xor (b10a9) xor (b11a8);
m_6(24):= ((b9a10 and b10a9) or
     (b9a10 and b11a8) or
     (b10a9 and b11a8));

m_6(25):=  (b9a11) xor (b10a10) xor (b11a9);
m_6(26):= ((b9a11 and b10a10) or
     (b9a11 and b11a9) or
     (b10a10 and b11a9));

m_6(27):=  (b9a12) xor (b10a11) xor (b11a10);
m_6(28):= ((b9a12 and b10a11) or
     (b9a12 and b11a10) or
     (b10a11 and b11a10));

m_6(29):=  (b9a13) xor (b10a12) xor (b11a11);
m_6(30):= ((b9a13 and b10a12) or
     (b9a13 and b11a11) or
     (b10a12 and b11a11));

b12a0 := bbb(12) and aaa(0);
b12a1 := bbb(12) and aaa(1);
b12a2 := bbb(12) and aaa(2);
b12a3 := bbb(12) and aaa(3);
b12a4 := bbb(12) and aaa(4);
b12a5 := bbb(12) and aaa(5);
b12a6 := bbb(12) and aaa(6);
b12a7 := bbb(12) and aaa(7);
b12a8 := bbb(12) and aaa(8);
b12a9 := bbb(12) and aaa(9);
b12a10 := bbb(12) and aaa(10);
b12a11 := bbb(12) and aaa(11);
b12a12 := bbb(12) and aaa(12);
b12a13 := bbb(12) and aaa(13);
b12a14 := bbb(12) and aaa(14);
b12a15 := bbb(12) and aaa(15);
```

```
--B-12

      sss(8) :=   m_5(2)  xor m_6(0) xor m_6(1);
      m_7(0) := ((m_5(2)  and m_6(0)) or
           (m_5(2)  and m_6(1)) or
           (m_6(0)  and m_6(1)));

      m_7(1) :=   m_5(4)  xor m_6(3);
      m_7(2) :=   m_5(4)  and m_6(3);


      m_7(3) :=   m_4(8)  xor m_5(7) xor m_6(5);
      m_7(4) := ((m_4(8)  and m_5(7)) or
           (m_4(8)  and m_6(5)) or
           (m_5(7)  and m_6(5)));

      m_7(5) :=   m_6(6) xor m_6(7);
      m_7(6) :=   m_6(6) and m_6(7);

      m_7(7) :=   b12a0  xor m_6(8) xor m_6(9);
      m_7(8) := ((b12a0  and m_6(8)) or
           (b12a0  and m_6(9)) or
           (m_6(8)   and m_6(9)));

      m_7(9) :=   b12a1    xor m_6(10) xor m_6(11);
      m_7(10):= ((b12a1    and m_6(10)) or
           (b12a1    and m_6(11)) or
           (m_6(10)  and m_6(11)));

      m_7(11):=   b12a2  xor m_6(12) xor m_6(13);
      m_7(12):= ((b12a2  and m_6(12)) or
           (b12a2  and m_6(13)) or
           (m_6(12)  and m_6(13)));

      m_7(13):=   b12a3  xor m_6(14) xor m_6(15);
      m_7(14):= ((b12a3  and m_6(14)) or
           (b12a3  and m_6(15)) or
           (m_6(14)  and m_6(15)));

      m_7(15):=   b12a4  xor m_6(16) xor m_6(17);
      m_7(16):= ((b12a4  and m_6(16)) or
           (b12a4  and m_6(17)) or
           (m_6(16)  and m_6(17)));

      m_7(17):=   b12a5  xor m_6(18) xor m_6(19);
      m_7(18):= ((b12a5  and m_6(18)) or
           (b12a5  and m_6(19)) or
           (m_6(18)  and m_6(19)));

      m_7(19):=   b12a6  xor m_6(20) xor m_6(21);
      m_7(20):= ((b12a6  and m_6(20)) or
           (b12a6  and m_6(21)) or
           (m_6(20)  and m_6(21)));
```

```
        m_7(21):=   b12a7  xor m_6(22) xor m_6(23);
        m_7(22):= ((b12a7  and m_6(22)) or
             (b12a7  and m_6(23)) or
             (m_6(22)  and m_6(23)));

        m_7(23):=   b12a8  xor m_6(24) xor m_6(25);
        m_7(24):= ((b12a8  and m_6(24)) or
             (b12a8  and m_6(25)) or
             (m_6(24)  and m_6(25)));

        m_7(25):=   b12a9  xor m_6(26) xor m_6(27);
        m_7(26):= ((b12a9  and m_6(26)) or
             (b12a9  and m_6(27)) or
             (m_6(26)  and m_6(27)));

        m_7(27):=   b12a10   xor  m_6(28)  xor m_6(29);
        m_7(28):= ((b12a10   and  m_6(28)) or
             (b12a10   and  m_6(29)) or
             (m_6(28)  and  m_6(29)));

        m_7(29):=   m_6(30)  xor b11a12 xor b12a11;
        m_7(30):= ((m_6(30)  and b11a12) or
             (m_6(30)  and b12a11) or
             (b11a12   and b12a11));

-- 0-1-2 + 3-4-5 + 6-7-8 + 9-10-11 + 12

        sss(9) :=   m_6(2)  xor m_7(0) xor m_7(1);
        m_8(0) := ((m_6(2)  and m_7(0)) or
             (m_6(2)  and m_7(1)) or
             (m_7(0)  and m_7(1)));

        m_8(1) :=   m_5(6)  xor m_6(4)  xor m_7(3);
        m_8(2) := ((m_5(6)  and m_6(4)) or
             (m_5(6)  and m_7(3)) or
             (m_6(4)  and m_7(3)));

        m_8(3) :=   m_4(10)  xor m_5(9) xor m_7(5);
        m_8(4) := ((m_4(10)  and m_5(9)) or
             (m_4(10)  and m_7(5)) or
             (m_5(9)   and m_7(5)));

        m_8(5) :=   m_4(12)  xor m_7(6) xor m_7(7);
        m_8(6) := ((m_4(12)  and m_7(6)) or
             (m_4(12)  and m_7(7)) or
             (m_7(6)   and m_7(7)));

        m_8(7) :=   m_4(14)  xor m_7(8) xor m_7(9);
        m_8(8) := ((m_4(14)  and m_7(8)) or
             (m_4(14)  and m_7(9)) or
             (m_7(8)   and m_7(9)));

        m_8(9) :=   m_4(16)  xor m_7(10) xor m_7(11);
        m_8(10):= ((m_4(16)  and m_7(10)) or
```

```
     (m_4(16)  and m_7(11)) or
     (m_7(10)  and m_7(11)));

m_8(11):=   m_4(18)  xor m_7(12) xor m_7(13);
m_8(12):= ((m_4(18)  and m_7(12)) or
     (m_4(18)  and m_7(13)) or
     (m_7(12)  and m_7(13)));

m_8(13):=   m_4(20)  xor m_7(14) xor m_7(15);
m_8(14):= ((m_4(20)  and m_7(14)) or
     (m_4(20)  and m_7(15)) or
     (m_7(14)  and m_7(15)));

m_8(15):=   m_4(22)  xor m_7(16) xor m_7(17);
m_8(16):= ((m_4(22)  and m_7(16)) or
     (m_4(22)  and m_7(17)) or
     (m_7(16)  and m_7(17)));

m_8(17):=   m_4(24)  xor m_7(18) xor m_7(19);
m_8(18):= ((m_4(24)  and m_7(18)) or
     (m_4(24)  and m_7(19)) or
     (m_7(18)  and m_7(19)));

m_8(19):=   m_4(26)  xor m_7(20) xor m_7(21);
m_8(20):= ((m_4(26)  and m_7(20)) or
     (m_4(26)  and m_7(21)) or
     (m_7(20)  and m_7(21)));

m_8(21):=   m_4(28)  xor m_7(22) xor m_7(23);
m_8(22):= ((m_4(28)  and m_7(22)) or
     (m_4(28)  and m_7(23)) or
     (m_7(22)  and m_7(23)));

m_8(23):=   m_4(30)  xor m_7(24) xor m_7(25);
m_8(24):= ((m_4(30)  and m_7(24)) or
     (m_4(30)  and m_7(25)) or
     (m_7(24)  and m_7(25)));

m_8(25):=   b7a15    xor m_7(26) xor m_7(27);
m_8(26):= ((b7a15    and m_7(26)) or
     (b7a15    and m_7(27)) or
     (m_7(26)  and m_7(27)));

m_8(27):=   b10a13   xor m_7(28) xor m_7(29);
m_8(28):= ((b10a13   and m_7(28)) or
     (b10a13   and m_7(29)) or
     (m_7(28)  and m_7(29)));

m_8(29):=   m_7(30)  xor b10a14 xor b11a13;
m_8(30):= ((m_7(30)  and b10a14) or
     (m_7(30)  and b11a13) or
     (b10a14   and b11a13));

-- 0-1-2 + 3-4-5 + 6-7-8 + 9-10-11 + 12 III
```

```
sss(10):=   m_7(2)  xor m_8(0) xor m_8(1);
m_9(0) := ((m_7(2)  and m_8(0)) or
     (m_7(2)  and m_8(1)) or
     (m_8(0)  and m_8(1)));

m_9(1) :=   m_5(8)  xor m_7(4)  xor m_8(3);
m_9(2) := ((m_5(8)  and m_7(4)) or
     (m_5(8)  and m_8(3)) or
     (m_7(4)  and m_8(3)));

m_9(3) :=   m_5(10)  xor m_5(11) xor m_8(5);
m_9(4) := ((m_5(10)  and m_5(11)) or
     (m_5(10)  and m_8(5)) or
     (m_5(11)  and m_8(5)));

m_9(5) :=   m_5(12)  xor m_5(13) xor m_8(7);
m_9(6) := ((m_5(12)  and m_5(13)) or
     (m_5(12)  and m_8(7)) or
     (m_5(13)  and m_8(7)));

m_9(7) :=   m_5(14)  xor m_5(15) xor m_8(9);
m_9(8) := ((m_5(14)  and m_5(15)) or
     (m_5(14)  and m_8(9)) or
     (m_5(15)  and m_8(9)));

m_9(9) :=   m_5(16)  xor m_5(17) xor m_8(11);
m_9(10):= ((m_5(16)  and m_5(17)) or
     (m_5(16)  and m_8(11)) or
     (m_5(17)  and m_8(11)));

m_9(11):=   m_5(18)  xor m_5(19) xor m_8(13);
m_9(12):= ((m_5(18)  and m_5(19)) or
     (m_5(18)  and m_8(13)) or
     (m_5(19)  and m_8(13)));

m_9(13):=   m_5(20)  xor m_5(21) xor m_8(15);
m_9(14):= ((m_5(20)  and m_5(21)) or
     (m_5(20)  and m_8(15)) or
     (m_5(21)  and m_8(15)));

m_9(15):=   m_5(22)  xor m_5(23) xor m_8(17);
m_9(16):= ((m_5(22)  and m_5(23)) or
     (m_5(22)  and m_8(17)) or
     (m_5(23)  and m_8(17)));

m_9(17):=   m_5(24)  xor m_5(25) xor m_8(19);
m_9(18):= ((m_5(24)  and m_5(25)) or
     (m_5(24)  and m_8(19)) or
     (m_5(25)  and m_8(19)));

m_9(19):=   m_5(26)  xor m_5(27) xor m_8(21);
m_9(20):= ((m_5(26)  and m_5(27)) or
     (m_5(26)  and m_8(21)) or
```

```
        (m_5(27)  and m_8(21)));

    m_9(21):=   m_5(28)  xor m_5(29) xor m_8(23);
    m_9(22):= ((m_5(28)  and m_5(29)) or
        (m_5(28)  and m_8(23)) or
        (m_5(29)  and m_8(23)));

    m_9(23):=   m_5(30)  xor b8a14 xor m_8(25);
    m_9(24):= ((m_5(30)  and b8a14) or
        (m_5(30)  and m_8(25)) or
        (b8a14    and m_8(25)));

    m_9(25):=   b8a15  xor b9a14    xor m_8(27);
    m_9(26):= ((b8a15  and b9a14)   or
        (b8a15  and m_8(27)) or
        (b9a14  and m_8(27)));

    m_9(27):=   m_8(29)  xor b9a15 xor b12a12;
    m_9(28):= ((m_8(29)  and b9a15) or
        (m_8(29)  and b12a12) or
        (b9a15    and b12a12));

    m_9(29):=   b10a15  xor b11a14 xor b12a13;
    m_9(30):= ((b10a15  and b11a14) or
        (b10a15  and b12a13) or
        (b11a14  and b12a13));

--B-13-14-15
    b13a0  := bbb(13) and  aaa(0);
    b13a1  := bbb(13) and  aaa(1);
    b13a2  := bbb(13) and  aaa(2);
    b13a3  := bbb(13) and  aaa(3);
    b13a4  := bbb(13) and  aaa(4);
    b13a5  := bbb(13) and  aaa(5);
    b13a6  := bbb(13) and  aaa(6);
    b13a7  := bbb(13) and  aaa(7);
    b13a8  := bbb(13) and  aaa(8);
    b13a9  := bbb(13) and  aaa(9);
    b13a10 := bbb(13) and aaa(10);
    b13a11 := bbb(13) and aaa(11);
    b13a12 := bbb(13) and aaa(12);
    b13a13 := bbb(13) and aaa(13);
    b13a14 := bbb(13) and aaa(14);
    b13a15 := bbb(13) and aaa(15);


    b14a0 := bbb(14) and aaa(0);
    b14a1 := bbb(14) and aaa(1);
    b14a2 := bbb(14) and aaa(2);
    b14a3 := bbb(14) and aaa(3);
    b14a4 := bbb(14) and aaa(4);
    b14a5 := bbb(14) and aaa(5);
    b14a6 := bbb(14) and aaa(6);
    b14a7 := bbb(14) and aaa(7);
```

```
b14a8 := bbb(14) and aaa(8);
b14a9 := bbb(14) and aaa(9);
b14a10 := bbb(14) and aaa(10);
b14a11 := bbb(14) and aaa(11);
b14a12 := bbb(14) and aaa(12);
b14a13 := bbb(14) and aaa(13);
b14a14 := bbb(14) and aaa(14);
b14a15 := bbb(14) and aaa(15);

b15a0 := bbb(15) and aaa(0);
b15a1 := bbb(15) and aaa(1);
b15a2 := bbb(15) and aaa(2);
b15a3 := bbb(15) and aaa(3);
b15a4 := bbb(15) and aaa(4);
b15a5 := bbb(15) and aaa(5);
b15a6 := bbb(15) and aaa(6);
b15a7 := bbb(15) and aaa(7);
b15a8 := bbb(15) and aaa(8);
b15a9 := bbb(15) and aaa(9);
b15a10 := bbb(15) and aaa(10);
b15a11 := bbb(15) and aaa(11);
b15a12 := bbb(15) and aaa(12);
b15a13 := bbb(15) and aaa(13);
b15a14 := bbb(15) and aaa(14);
b15a15 := bbb(15) and aaa(15);

sss(11) :=   m_9(0) xor m_9(1) xor m_8(2);
m_10(0) := ((m_9(0) and m_9(1)) or
      (m_9(0) and m_8(2)) or
      (m_9(1) and m_8(2)));

m_10(1) :=   m_8(4) xor m_9(3);
m_10(2) :=   m_8(4) and m_9(3);

m_10(3) :=   m_8(6) xor m_9(5) xor b13a0;
m_10(4) := ((m_8(6) and m_9(5)) or
      (m_8(6) and b13a0) or
      (m_9(5) and b13a0));

m_10(5) :=   b13a1 xor b14a0;
m_10(6) :=   b13a1 and b14a0;

m_10(7) :=  (b13a2) xor b14a1 xor (b15a0);
m_10(8) := ((b13a2  and b14a1) or
      (b13a2  and b15a0) or
      (b14a1  and b15a0));

m_10(9) :=  (b13a3) xor b14a2 xor (b15a1);
m_10(10):= ((b13a3  and b14a2) or
      (b13a3  and b15a1) or
      (b14a2  and b15a1));

m_10(11):=  (b13a4) xor b14a3 xor (b15a2);
m_10(12):= ((b13a4  and b14a3) or
```

```
            (b13a4  and b15a2) or
            (b14a3  and b15a2));

        m_10(13):=  (b13a5) xor b14a4 xor (b15a3);
        m_10(14):= ((b13a5  and b14a4) or
            (b13a5  and b15a3) or
            (b14a4  and b15a3));

        m_10(15) :=  (b13a6) xor b14a5 xor (b15a4);
        m_10(16) := ((b13a6  and b14a5) or
             (b13a6  and b15a4) or
             (b14a5  and b15a4));

        m_10(17) := (b13a7) xor b14a6 xor (b15a5);
        m_10(18):= ((b13a7  and b14a6) or
            (b13a7  and b15a5) or
            (b14a6  and b15a5));

        m_10(19):=  (b13a8) xor (b14a7) xor (b15a6);
        m_10(20):= ((b13a8  and b14a7) or
            (b13a8  and b15a6) or
            (b14a7  and b15a6));

        m_10(21):=  (b13a9) xor (b14a8) xor (b15a7);
        m_10(22):= ((b13a9  and b14a8) or
            (b13a9  and b15a7) or
            (b14a8  and b15a7));

        m_10(23):=  (b13a10) xor (b14a9) xor (b15a8);
        m_10(24):= ((b13a10 and b14a9) or
            (b13a10 and b15a8) or
            (b14a9 and b15a8));

        m_10(25):=  (b13a11) xor (b14a10) xor (b15a9);
        m_10(26):= ((b13a11 and b14a10) or
            (b13a11 and b15a9) or
            (b14a10 and b15a9));

        m_10(27):=  (b13a12) xor (b14a11) xor (b15a10);
        m_10(28):= ((b13a12 and b14a11) or
            (b13a12 and b15a10) or
            (b14a11 and b15a10));

        m_10(29):=  (b13a13) xor (b14a12) xor (b15a11);
        m_10(30):= ((b13a13 and b14a12) or
            (b13a13 and b15a11) or
            (b14a12 and b15a11));

-- alle I
--pipe1
    res1_s       <= res_v      ;
    exception1_s <= exception;
    absign1_s    <= absign   ;
    ivo1_s       <= ivo_v      ;
```

```
    underflow1_s <= underflow;
    overflow1_s  <= overflow ;
    ExpS1_s      <= ExpS_v     ;
    m_8_s        <= m_8     ;
    m_9_s        <= m_9     ;
    m_10_s       <= m_10     ;
    b12a14_s     <= b12a14;
    b12a15_s     <= b12a15;
    b13a14_s     <= b13a14;
    b13a15_s     <= b13a15;
    b14a14_s     <= b14a14;
    b11a15_s     <= b11a15;
    b14a13_s     <= b14a13;
    b15a12_s     <= b15a12;
    b14a15_s     <= b14a15     ;
    b15a13_s     <= b15a13     ;
    b15a14_s     <= b15a14     ;
    b15a15_s     <= b15a15     ;
    s1_s         <= sss;
     end if;
   end if;
 end process fpmult_pre;
--piplining 1. forsøk


 fpmult_mid: process(clk,resetn)
   variable res_v        : ffloat;
   variable exception    : boolean;
   variable ivo_v        : std_logic;
   variable ExpS_v      : std_logic_vector(exphi downto 0);
   variable underflow    : std_logic;
   variable overflow     : std_logic;
   variable fullproduct : std_logic_vector(2*manhi+1 downto 0);
--     variable mant_vv  : std_logic_vector(2*manhi+1 downto 0);
--     variable mant_v  : std_logic_vector(manhi downto 0);
--     variable mant_rr : std_logic_vector(manhi+1 downto 0);
--     variable mant_r : std_logic_vector(manhi downto 0);
--     variable normprod : std_logic_vector(manhi downto 0);
--     variable mantout : std_logic_vector(manhi downto 0);
--     variable flags_v_guard : std_logic;
--     variable flags_v_round : std_logic;
--     variable flags_v_sticky : std_logic;
--     variable AddToExp,AddToExp2 : boolean;
--     variable expout : std_logic_vector(exphi downto 0);
--     variable carryMSB : std_logic;
--     variable Oflw_v : std_logic;
--     variable Uflw_v  : std_logic;
   variable absign  : std_logic;
--     variable expadj : unsigned(exphi downto 0);
--array mult
   variable sss : std_logic_vector(31 downto 0);
   variable m_8, m_9,m_10,m_11,m_12,m_13 : std_logic_vector(30
downto 0);
   variable anp, bnp : std_logic_vector(16 downto 0);
```

```
    variable snp : unsigned(16 downto 0);
    variable b12a14,b12a15,b13a14,b13a15,b14a14 :std_logic;
    variable b11a15,b14a13,b15a12,b14a15,b15a13,b15a14,b15a15
:std_logic;
--
  begin
    if (resetn = '0') then
      res2_s.mant  <= (others => '0');
      res2_s.exp   <= (others => '0');
      res2_s.sign  <= '0';
      exception2_s <= false;
      absign2_s    <= '0';
      ivo2_s       <= '0';
      underflow2_s <= '0';
      overflow2_s  <= '0';
      ExpS2_s      <= (others => '0');
      fullproduct_s<= (others => '0');
    elsif (clk ='1' and clk'event) then
      if (hold = '0') then
     res_v       := res1_s      ;
     exception   := exception1_s;
     absign      := absign1_s   ;
     ivo_v       := ivo1_s      ;
     underflow   := underflow1_s;
     overflow    := overflow1_s ;
     ExpS_v      := ExpS1_s     ;
     m_8         := m_8_s     ;
     m_9         := m_9_s     ;
     m_10        := m_10_s     ;
--   m_11        := m_11_s     ;
     b12a14    := b12a14_s;
     b12a15    := b12a15_s;
     b13a14    := b13a14_s;
     b13a15    := b13a15_s;
     b14a14    := b14a14_s;
     b11a15      := b11a15_s     ;
     b14a13      := b14a13_s     ;
     b15a12      := b15a12_s     ;
     b14a15      := b14a15_s   ;
     b15a13      := b15a13_s   ;
     b15a14      := b15a14_s   ;
     b15a15      := b15a15_s   ;
     sss(11 downto 0) := s1_s;


      sss(12) :=   m_9(2)  xor m_10(0) xor m_10(1);
      m_11(0) := ((m_9(2)  and m_10(0)) or
            (m_9(2)  and m_10(1)) or
            (m_10(0)  and m_10(1)));

      m_11(1) :=   m_9(4)  xor m_10(3);
      m_11(2) :=   m_9(4)  and m_10(3);
```

```
m_11(3) :=   m_8(8)  xor m_9(7) xor m_10(5);
m_11(4) := ((m_8(8)  and m_9(7)) or
      (m_8(8)  and m_10(5)) or
      (m_9(7)  and m_10(5)));

m_11(5) :=   m_8(10)  xor m_10(6) xor m_10(7);
m_11(6) := ((m_8(10)  and m_10(6)) or
      (m_8(10)  and m_10(7)) or
      (m_10(6)  and m_10(7)));

m_11(7) :=   m_8(12)  xor m_10(8) xor m_10(9);
m_11(8) := ((m_8(12)  and m_10(8)) or
      (m_8(12)  and m_10(9)) or
      (m_10(8)  and m_10(9)));

m_11(9) :=   m_8(14)  xor m_10(10) xor m_10(11);
m_11(10):= ((m_8(14)  and m_10(10)) or
      (m_8(14)  and m_10(11)) or
      (m_10(10) and m_10(11)));

m_11(11):=   m_8(16)  xor m_10(12) xor m_10(13);
m_11(12):= ((m_8(16)  and m_10(12)) or
      (m_8(16)  and m_10(13)) or
      (m_10(12) and m_10(13)));

m_11(13):=   m_8(18)  xor m_10(14) xor m_10(15);
m_11(14):= ((m_8(18)  and m_10(14)) or
      (m_8(18)  and m_10(15)) or
      (m_10(14) and m_10(15)));

m_11(15):=   m_8(20)  xor m_10(16) xor m_10(17);
m_11(16):= ((m_8(20)  and m_10(16)) or
      (m_8(20)  and m_10(17)) or
      (m_10(16) and m_10(17)));

m_11(17):=   m_8(22)  xor m_10(18) xor m_10(19);
m_11(18):= ((m_8(22)  and m_10(18)) or
      (m_8(22)  and m_10(19)) or
      (m_10(18) and m_10(19)));

m_11(19):=   m_8(24)  xor m_10(20) xor m_10(21);
m_11(20):= ((m_8(24)  and m_10(20)) or
      (m_8(24)  and m_10(21)) or
      (m_10(20) and m_10(21)));

m_11(21):=   m_8(26)  xor m_10(22) xor m_10(23);
m_11(22):= ((m_8(26)  and m_10(22)) or
      (m_8(26)  and m_10(23)) or
      (m_10(22) and m_10(23)));

m_11(23):=   m_8(28)  xor m_10(24) xor m_10(25);
m_11(24):= ((m_8(28)  and m_10(24)) or
      (m_8(28)  and m_10(25)) or
      (m_10(24) and m_10(25)));
```

```
        m_11(25):=   m_8(30)  xor m_10(26) xor m_10(27);
        m_11(26):= ((m_8(30)  and m_10(26)) or
             (m_8(30)  and m_10(27)) or
             (m_10(26) and m_10(27)));

        m_11(27):=   b11a15   xor m_10(28) xor m_10(29);
        m_11(28):= ((b11a15   and m_10(28)) or
             (b11a15   and m_10(29)) or
             (m_10(28) and m_10(29)));

        m_11(29):=   b14a13   xor m_10(30) xor b15a12;
        m_11(30):= ((b14a13   and m_10(30)) or
             (b14a13   and b15a12) or
             (m_10(30) and b15a12));


-- alt II

        sss(13):=    m_10(2)  xor m_11(0) xor m_11(1);
        m_12(0) := ((m_10(2)  and m_11(0)) or
             (m_10(2)  and m_11(1)) or
             (m_11(0)  and m_11(1)));

        m_12(1) :=   m_9(6)  xor m_10(4)  xor m_11(3);
        m_12(2) := ((m_9(6)  and m_10(4)) or
             (m_9(6)  and m_11(3)) or
             (m_10(4) and m_11(3)));

        m_12(3) :=   m_9(8)  xor m_9(9) xor m_11(5);
        m_12(4) := ((m_9(8)  and m_9(9)) or
             (m_9(8)  and m_11(5)) or
             (m_9(9)  and m_11(5)));

        m_12(5) :=   m_9(10)  xor m_9(11) xor m_11(7);
        m_12(6) := ((m_9(10)  and m_9(11)) or
             (m_9(10)  and m_11(7)) or
             (m_9(11)  and m_11(7)));

        m_12(7) :=   m_9(12)  xor m_9(13) xor m_11(9);
        m_12(8) := ((m_9(12)  and m_9(13)) or
             (m_9(12)  and m_11(9)) or
             (m_9(13)  and m_11(9)));

        m_12(9) :=   m_9(14)  xor m_9(15) xor m_11(11);
        m_12(10):= ((m_9(14)  and m_9(15)) or
             (m_9(14)  and m_11(11)) or
             (m_9(15)  and m_11(11)));

        m_12(11):=   m_9(16)  xor m_9(17) xor m_11(13);
        m_12(12):= ((m_9(16)  and m_9(17)) or
             (m_9(16)  and m_11(13)) or
             (m_9(17)  and m_11(13)));
```

```
      m_12(13):=   m_9(18)  xor m_9(19) xor m_11(15);
      m_12(14):= ((m_9(18)  and m_9(19)) or
            (m_9(18)  and m_11(15)) or
            (m_9(19)  and m_11(15)));

      m_12(15):=   m_9(20)  xor m_9(21) xor m_11(17);
      m_12(16):= ((m_9(20)  and m_9(21)) or
            (m_9(20)  and m_11(17)) or
            (m_9(21)  and m_11(17)));

      m_12(17):=   m_9(22)  xor m_9(23) xor m_11(19);
      m_12(18):= ((m_9(22)  and m_9(23)) or
            (m_9(22)  and m_11(19)) or
            (m_9(23)  and m_11(19)));

      m_12(19):=   m_9(24)  xor m_9(25) xor m_11(21);
      m_12(20):= ((m_9(24)  and m_9(25)) or
            (m_9(24)  and m_11(21)) or
            (m_9(25)  and m_11(21)));

      m_12(21):=   m_9(26)  xor m_9(27) xor m_11(23);
      m_12(22):= ((m_9(26)  and m_9(27)) or
            (m_9(26)  and m_11(23)) or
            (m_9(27)  and m_11(23)));

      m_12(23):=   m_9(28)  xor m_9(29) xor m_11(25);
      m_12(24):= ((m_9(28)  and m_9(29)) or
            (m_9(28)  and m_11(25)) or
            (m_9(29)  and m_11(25)));

      m_12(25):=   m_9(30)  xor b12a14    xor m_11(27);
      m_12(26):= ((m_9(30)  and b12a14)   or
            (m_9(30)  and m_11(27)) or
            (b12a14   and m_11(27)));

      m_12(27):=   m_11(29)  xor b12a15 xor b13a14;
      m_12(28):= ((m_11(29)  and b12a15) or
            (m_11(29)  and b13a14) or
            (b12a15    and b13a14));

      m_12(29):=   b13a15  xor b14a14 xor b15a13;
      m_12(30):= ((b13a15  and b14a14) or
            (b13a15  and b15a13) or
            (b14a14  and b15a13));


-- alle III

      sss(14):=    m_11(2)  xor m_12(0) xor m_12(1);
      m_13(0) := ((m_11(2)  and m_12(0)) or
            (m_11(2)  and m_12(1)) or
            (m_12(0)  and m_12(1)));

      m_13(1) :=   m_11(4)  xor m_12(3);
```

```
m_13(2) :=   m_11(4)  and m_12(3);

m_13(3) :=   m_11(6)  xor m_12(4) xor m_12(5);
m_13(4) := ((m_11(6)  and m_12(4)) or
      (m_11(6)  and m_12(5)) or
      (m_12(4)  and m_12(5)));

m_13(5) :=   m_11(8)  xor m_12(6) xor m_12(7);
m_13(6) := ((m_11(8)  and m_12(6)) or
      (m_11(8)  and m_12(7)) or
      (m_12(6)  and m_12(7)));

m_13(7) :=   m_11(10)  xor m_12(8) xor m_12(9);
m_13(8) := ((m_11(10)  and m_12(8)) or
      (m_11(10)  and m_12(9)) or
      (m_12(8)   and m_12(9)));

m_13(9) :=   m_11(12)  xor m_12(10) xor m_12(11);
m_13(10):= ((m_11(12)  and m_12(10)) or
      (m_11(12)  and m_12(11)) or
      (m_12(10)  and m_12(11)));

m_13(11):=   m_11(14)  xor m_12(12) xor m_12(13);
m_13(12):= ((m_11(14)  and m_12(12)) or
      (m_11(14)  and m_12(13)) or
      (m_12(12)  and m_12(13)));

m_13(13):=   m_11(16)  xor m_12(14) xor m_12(15);
m_13(14):= ((m_11(16)  and m_12(14)) or
      (m_11(16)  and m_12(15)) or
      (m_12(14)  and m_12(15)));

m_13(15):=   m_11(18)  xor m_12(16) xor m_12(17);
m_13(16):= ((m_11(18)  and m_12(16)) or
      (m_11(18)  and m_12(17)) or
      (m_12(16)  and m_12(17)));

m_13(17):=   m_11(20)  xor m_12(18) xor m_12(19);
m_13(18):= ((m_11(20)  and m_12(18)) or
      (m_11(20)  and m_12(19)) or
      (m_12(18)  and m_12(19)));

m_13(19):=   m_11(22)  xor m_12(20) xor m_12(21);
m_13(20):= ((m_11(22)  and m_12(20)) or
      (m_11(22)  and m_12(21)) or
      (m_12(20)  and m_12(21)));

m_13(21):=   m_11(24)  xor m_12(22) xor m_12(23);
m_13(22):= ((m_11(24)  and m_12(22)) or
      (m_11(24)  and m_12(23)) or
      (m_12(22)  and m_12(23)));

m_13(23):=   m_11(26)  xor m_12(24) xor m_12(25);
m_13(24):= ((m_11(26)  and m_12(24)) or
```

```
            (m_11(26)  and m_12(25)) or
            (m_12(24)  and m_12(25)));

      m_13(25):=   m_11(28)  xor m_12(26) xor m_12(27);
      m_13(26):= ((m_11(28)  and m_12(26)) or
            (m_11(28)  and m_12(27)) or
            (m_12(26)  and m_12(27)));

      m_13(27):=   m_11(30)  xor m_12(28) xor m_12(29);
      m_13(28):= ((m_11(30)  and m_12(28)) or
            (m_11(30)  and m_12(29)) or
            (m_12(28)  and m_12(29)));

      m_13(29):=   b14a15  xor b15a14 xor m_12(30);
      m_13(30):= ((b14a15  and b15a14  ) or
            (b14a15  and m_12(30)) or
            (b15a14  and m_12(30)));

--FINAL ADDER

      anp := ('0'&m_13(30)&m_13(28)&m_13(26)&m_13(24)&m_13(22)
              &m_13(20)&m_13(18)&m_13(16)&m_13(14)&m_13(12)
              &m_13(10)&m_13(8)& m_13(6)& m_13(4)& m_13(2)&m_13(0));
      bnp := ('0'&b15a15&  m_13(29)&m_13(27)&m_13(25)&m_13(23)
              &m_13(21)&m_13(19)&m_13(17)&m_13(15)&m_13(13)
              &m_13(11)&m_13(9)&m_13(7)&m_13(5)&m_13(3)&m_13(1));

      snp := unsigned(anp) + unsigned(bnp) + m_12(2);

      sss(31 downto 15) := std_logic_vector(snp(16 downto 0));
      fullproduct := sss;
--pipe1
     res2_s       <= res_v      ;
     exception2_s <= exception;
     absign2_s    <= absign   ;
     ivo2_s       <= ivo_v      ;
     underflow2_s <= underflow;
     overflow2_s  <= overflow ;
     ExpS2_s      <= ExpS_v     ;
     fullproduct_s <= fullproduct ;
      end if;
    end if;
  end process fpmult_mid;
--piplining


  fpmult_post: process(clk,resetn)
    variable res_v        : ffloat;
    variable exception    : boolean;
    variable ivo_v        : std_logic;
    variable ExpS_v      : std_logic_vector(exphi downto 0);
    variable underflow    : std_logic;
    variable overflow     : std_logic;
    variable fullproduct : std_logic_vector(2*manhi+1 downto 0);
```

```
    variable mant_vv  : std_logic_vector(2*manhi+1 downto 0);
    variable mant_v   : std_logic_vector(manhi downto 0);
    variable mant_rr : std_logic_vector(manhi+1 downto 0);
    variable mant_r : std_logic_vector(manhi downto 0);
    variable normprod : std_logic_vector(manhi downto 0);
    variable mantout : std_logic_vector(manhi downto 0);
    variable flags_v_guard : std_logic;
    variable flags_v_round : std_logic;
    variable flags_v_sticky : std_logic;
    variable AddToExp,AddToExp2 : boolean;
    variable expout : std_logic_vector(exphi downto 0);
    variable carryMSB : std_logic;
    variable Oflw_v : std_logic;
    variable Uflw_v  : std_logic;
    variable absign  : std_logic;
    variable expadj : unsigned(exphi downto 0);
--array mult


--
  begin
    if (resetn = '0') then
      product.mant  <= (others => '0');
      product.exp   <= (others => '0');
      product.sign  <= '0';
      mult_ofl <= '0';
      mult_ufl <= '0';
      mult_ivo <= '0';

    elsif (clk ='1' and clk'event) then
      if (hold = '0') then
     res_v       := res2_s      ;
     exception   := exception2_s;
     absign      := absign2_s   ;
     ivo_v       := ivo2_s      ;
     underflow   := underflow2_s;
     overflow    := overflow2_s ;
     ExpS_v      := ExpS2_s     ;
     fullproduct := fullproduct_s;

--gr(ap)s
     Flags_v_Guard  := fullProduct(manhi);
     Flags_v_Round  := fullProduct(manhi-1);
     Flags_v_Sticky := '0';
     Sticky_Lbl:
     for Index_i in (manhi-2) downto 0 loop
       Flags_v_Sticky := Flags_v_Sticky or fullProduct(Index_i);
     end loop Sticky_Lbl;

--(u)norm(al)
     Mant_vv:=fullProduct(fullProduct'high-1 downto fullProduct'low)
& '0';
     AddToExp := false;
     if (fullProduct(fullProduct'high)='1') then
```

```
      AddToExp := true;
      Mant_vv    := fullProduct;
      Flags_v_Sticky := flags_v_Sticky or flags_v_Round;
      Flags_v_Round  := flags_v_Guard;
      Flags_v_Guard  := '0';
    end if;
    mant_v := mant_vv(mant_vv'high downto manhi+1);

 --Rounding
    Mant_rr := '0' & Mant_v;
    if ((flags_v_Round='1') and
        (flags_v_Sticky='1' or Mant_v(Mant_v'low)='1')) then
      Mant_rr := unsigned('0' & Mant_v) + '1';
    end if;
    Mant_r := Mant_rr(Mant_rr'high-1 downto Mant_rr'low);
    CarryMSB := Mant_rr(Mant_rr'high);

--Normalization2
    AddToExp2 := false;
    NormProd := mant_r;
    if (CarryMSB = '1') then
      AddToExp2  := true;
      NormProd := CarryMSB&mant_r(mant_r'high downto mant_r'low+1);
    end if;

--exp adjust
    Expadj := unsigned(Exps_v);
    if (Addtoexp or Addtoexp2) then
      Expadj := unsigned(Exps_v) + '1';
    end if;

--ExceptionCheck
    Oflw_v  := '0';
    Uflw_v  := '0';
    ExpOut  := std_logic_vector(expadj);
    MantOut := normprod;
       --Overflow
    if ((Overflow = '1') or
        (Underflow= '0' and
         (std_logic_vector(expadj) = ExpAllOnesVec_c))) then
      Oflw_v  := '1';
      ExpOut  := ExpAllOnesVec_c;
      MantOut := '1' & ZeroVec(normprod'high-1 downto 0);
      --Underflow
    elsif ( Underflow='1' or
         (std_logic_vector(expadj) = ExpAllZeroVec_c)) then
      Uflw_v  := '1';
      ExpOut  := ExpAllZeroVec_c;
      MantOut := ZeroVec;
    end if;

   --If we have any exception inputs, the flags
   --for Overflow and underflow shall not be put out.
```

```
      if (Exception) then
        product.mant <= res_v.mant(manhi-1 downto 0);
        product.exp  <= res_v.exp ;
        product.sign <= res_v.sign;
        mult_ofl <= '0';
        mult_ufl <= '0';
      else
        product.mant <= mantout(manhi-1 downto 0);
        product.exp  <= expout;
        if (underflow = '1' or
            (overflow = '0' and
             std_logic_vector(expadj) = ExpAllZeroVec_c)) then
          product.sign <= '0';
        else
          product.sign <= absign;
        end if;
        mult_ofl <= Oflw_v;
        mult_ufl <= Uflw_v;
      end if;
      mult_ivo <= ivo_v;
       end if; --hold
     end if;--clk
  end process fpmult_post;
end pipe3;
```

B VHDL–KODE FOR TESTBENKEN OG TESTBENKINTERFACE

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

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

```
-------------------------------------------------------------------
-------
--FPMULT 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 multlib;
USE multlib.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 of input mantissa
constant outp_mantsize  : integer := (outp_wordsize-outp_expsize) -
1; -- Size of output mantissa

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 := (     0,     0,   200,   100);
    constant exp1_inc1   : params := (     1,    -1,    -1,    -1);
    constant exp1_stop1  : params := (     2,    -1,    -1,    -1);
    constant exp1_start2 : params := (   125,    -1,    -1,    -1);
    constant exp1_inc2   : params := (     1,    -1,    -1,    -1);
    constant exp1_stop2  : params := (   129,    -1,    -1,    -1);
    constant exp1_start3 : params := (   253,    -1,    -1,    -1);
    constant exp1_inc3   : params := (     1,     1,     1,     1);
    constant exp1_stop   : params := (   255,   255,   200,   100);

    constant exp2_start  : params := (     0,     0,   200,   100);
    constant exp2_inc1   : params := (     1,    -1,    -1,    -1);
    constant exp2_stop1  : params := (     2,    -1,    -1,    -1);
    constant exp2_start2 : params := (   125,    -1,    -1,    -1);
    constant exp2_inc2   : params := (     1,    -1,    -1,    -1);
    constant exp2_stop2  : params := (   129,    -1,    -1,    -1);
    constant exp2_start3 : params := (   253,    -1,    -1,    -1);
    constant exp2_inc3   : params := (     1,     1,     1,     1);
    constant exp2_stop   : params := (   255,   255,   200,   100);

    constant mant1_start : params := (     0, 32767,     0,     0);
    constant mant1_inc   : params := (     1,     1,     1,     1);
    constant mant1_stop  : params := (    60, 32768,     4, 32768);
    constant mant1_type  : params := (     2,     0,     1,     0);
    constant mant2_start : params := (     0,     0,     0,     0);
    constant mant2_inc   : params := (     1,  4096,     1,     1);
    constant mant2_stop  : params := (    60,  4096, 32768,     4);
    constant mant2_type  : params := (     2,     0,     0,     1);


COMPONENT fpmult
   PORT(
        clk          : IN  std_logic; -- system clock
        resetn       : IN  std_logic; -- system reset
        hold         : IN  std_logic; -- hold signal inhibiting
changes
        multiplicand : IN  nfloat;    -- argument for multiplication
        multiplier   : IN  nfloat;    -- argument for multiplication
        product      : OUT nfloat;    -- final pipelined product
        mult_ofl     : OUT std_logic; -- overflow from
multiplication
        mult_ufl     : OUT std_logic; -- underflow from
multiplication
        mult_ivo     : OUT std_logic  -- ivo from multiplication


   );
END COMPONENT;
for all : fpmult use entity multlib.fpmult;
```

```
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
        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);
signal facina, facinb : slv_vector;
signal facres : std_logic_vector(outp_wordsize-1 downto 0);
signal dutina, dutinb, dutres: nfloat;
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;
signal dut_op :  std_logic := '0'; -- Setter DUT til M-e addere



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 : fpmult
    PORT MAP(
             clk          => clk,
             resetn       => resetn,
             hold         => hold,
             multiplicand => dutina,
             multiplier   => dutinb,
               product      => dutres,
             mult_ofl     => dut_ofl,
             mult_ufl     => dut_ufl,
             mult_ivo     => dut_ivo

             );


  FACGEN : facitgen
    PORT MAP(
             ina => facina(dutpipelen-1),
             inb => facinb(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 Sign1, Sign2                    : std_logic;
    variable Exponent1, Exponent2            :
std_logic_vector(inp_expsize-1 downto 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 one
              mantisse(0) := '1';
            when 2 =>
              mantisse := (others => '1');                    --
all ones
            when others =>
              mantisse := (others => '1');                    --
ones, bit0 zero
              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 one
```

```
            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
       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');
                   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;

    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);
        finishdly(i) <= finishdly(i-1);
      END LOOP delayline;
      facina(0) <= dutina.sign&dutina.exp&dutina.mant;
      facinb(0) <= dutinb.sign&dutinb.exp&dutinb.mant;
      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) /=
"11111111000000000000000")              -- ingen av inngangene
uendelig
                and (facinb(dutpipelen-1)(22 downto 0) /=
"11111111000000000000000")) 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) =
"11111111000000000000000"))             -- 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) =
"11111111000000000000000"))) 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 0 og ingen input
                    and (facinb(dutpipelen-1)(22 downto 15) /=
"00000000"))) then                      -- 0 eller denormalisert
            tb_ufl := '1';
        end if;

        out_data(outp_wordsize-1 downto 0):=
dutres.sign&dutres.exp&dutres.mant;
        if(out_data /= facres or tb_ofl /= dut_ofl or tb_ivo /=
dut_ivo or tb_ufl /= dut_ufl) then
          dataerror <= true;
          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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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), right, 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 multiplikasjonskretsen.

D.1 C–filen sim_fpmul.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 [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[6];

  while (read_data(fd_in, 2, data)) {
    printf("%06X \n", fpmul_24bit(data[0], data[1]));
  }
}
```

D.2 C–filen 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 C–biblioteks 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

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 velges slik at eksponentverdien alltid er stigende. Fortegnene starter som 0 og skifter til 1.

parameternavn	**sett1**	**sett2**	**sett3**	**sett4**
exp1 start value :	0	0	200	100
exp1 inc1 value :	1			
exp1 stop1 value :	2			
exp1 start2 value :	125			
exp1 inc2 value :	1			
exp1 stop2 value :	129			
exp1 start3 value :	253			
exp1 inc3 value :	1	1	1	1
exp1 stop value :	255	255	200	200
exp2 start value :	0	0	200	100
exp2 inc1 value :	1			
exp2 stop1 value :	2			
exp2 start2 value :	125			
exp2 inc2 value :	1			
exp2 stop2 value :	129			
exp2 start3 value :	253			
exp2 inc3 value :	1	1	1	1
exp2 stop value :	255	255	200	100
mant1 start value :	0	32767	0	0
mant1 inc value :	1	1	1	1
mant1 stop value :	60	32768	4	32768
mant1 type :	2	0	1	0

mant2 start value :	0	0	0	0
mant2 inc value :	1	4096	1	1
mant2 stop value :	60	4096	32768	4
mant2 type :	2	0	0	1

Alle datasettene kjøres med alle kombinasjoner av fortegn.