Technische Universiteit Delft
Faculteit der Electrotechniek

Aard Taakverslag
Omvang 48 pag.
Datum oktober 1986

Vakgroep Telecommunicatie en verkeers-begeleidingssystemen (laboratorium voor transmissie van informatie).

Auteur J.W.Vieveen
Titel Simulatie van een tweetraps DSM Codec.

Korte Inhoud:

Om de kwaliteit van een Delta Sigma modulator Codec (CoDer - DECoder) op te voeren kan men het geïntegreerde verschilsignaal opnieuw coderen en versturen samen met het DSM signaal. Deze signalen kunnen dan later gecombineerd worden. Er treed nu een behoorlijke verbetering op in de signaal-ruis verhouding. De signaal-ruis verhouding wordt echter niet verdubbeld, zoals o.a. door Steele (lit.1) bij Delta Modulatoren wordt aangegeven, onder aanname van RC gevormde Gaussische ingangssignalen. Ook Bergen Henegouwen heeft onderzoek gedaan naar 2-bit DSM codecs (lit.2).

Nadelig voor de bouw van de tweetraps DSM codec is de toegenomen complextiteit. De DSM wordt namelijk dubbel uitgevoerd, en de beide signalen moeten gemultiplext verzonden worden. Bovendien dient men bij het samenvoegen van de signalen rekening te houden met faseverschuivingen tussen de beide codecs.

De signaal-ruisverhouding kan met behulp van noise-shaping wel iets worden verhoogd, maar niet genoeg om de resultaten acceptabel te maken voor een praktische realisatie van een tweetraps DSM.

Mentor: Ir. A.J.R.M. Coenen.

Inhoudsopgave

Inleiding

In dit rapport wordt een Delta Sigma Modulator beschreven, waarvan het geïntegreerde verschilsignaal opnieuw gecodeerd en verzonden wordt. Volgens Steele (lit.1) mag men dan de signaal-ruis verhouding van een enkele DSM-coder verdubbelen, met toepassing van een RC shaped Gaussisch ingangssignaal en mits rekening is gehouden met delays in de filters. Het compenseren voor de delays is nodig, daar anders fase verschillen kunnen optreden. Met een computersimulatie is de tweetraps codec gesimuleerd, met de reden dat veel variabelen gemakkelijk ingesteld kunnen worden op een computer.

Na de inleiding zal in de tekst eerst een enkele DSM worden besproken. In hoofdstuk 3 zal de tweetraps DSM bekeken worden, waarna in hoofdstuk 4 enige resultaten getoond zullen worden. Tenslotte zal in hoofdstuk 5 geprobeerd worden de resultaten wat te verbeteren en waar nodig de consequenties voor een praktische realisatie aan te geven. Conclusies zijn vermeld in hoofdstuk 6. In hoofdstuk 5 worden aspecten van de simulatie zelf behandeld.

2:Opbouw van een D.S.M.

2.1 Theorie van de DSM

Een Delta Sigma Modulator weinig complex van opzet. Omdat een DSM betrekkelijk eenvoudig te bouwen is, weinig componenten bevat en daardoor goedkoop is, is het interessant hier onderzoek naar te doen. Bovendien biedt de DSM code veel mogelijkheden tot hoog frequent digitale filteroperaties, wat bijvoorbeeld met PCM moeilijker te realiseren is.

Delta Sigma Modulatie is eigenlijk een bijzonder geval van Puls Code Modulatie (PCM), waarin de amplitude van een te bemonsteren signaal op discrete tijdstippen volgens een vaste sample-frequentie bemonsterd wordt in amplitude discrete waarden. Bij PCM is het aantal discrete waarden 2 tot de macht n, met n de lengte van de rij bits die per sample gevormd worden. Bij waarden van n boven de 6 à 7 komt men al op zeer acceptabele waarden voor de signaal-ruis verhouding. Een DSM is eigenlijk een PCM met n=1. Het signaal heeft twee nivo's.

De kwaliteit van de DSM hangt af van het oversampelings-mechanisme plus het toepassen van noise-shaping. De signaal ruis verhouding is dan op zijn gunstigst:

$$S/N = 2 + 30 * \log(b)$$

formule 2.1

b is de oversamplingsfactor gedefinieerd door:

$$b = \frac{f_s}{[2 * f_h]}.$$

formule 2.2

Hierbij is f_s = bemonsterfrequentie
f_h = hoogste frequentie in het basisband signaal.

Voor bewijs van deze formules wordt verwezen naar Coenen (lit. 3).

Het merendeel van onderzoek op dit gebied is er op gericht de signaal-ruis verhouding van een Delta Sigma Modulator codec systeem zo hoog mogelijk te krijgen bij een gelijk blijvende bitrate, zoals ook deze taakopdracht.

De reden dat een DSM gebruikt wordt in plaats van PCM, is de gemakkelijke codering. Wil men een videosignaal verzenden, dan is een basisband van 0-5 Mhz noodzakelijk. Vooral de lage (basisband) frequenties zijn belangrijk, omdat het oog gevoeliger is voor laagfrequente storingen dan voor hoogfrequente. Bij videotransmissie is een conventionele PCM-coder nogal complex, en dus duur. Want om de hoge bewerkingssnelheid te halen zijn snelle en dus dure componenten nodig, die bovendien de nodige energie verbruiken. Kortom, (nog) geen opstelling om de consument in huis te geven. Een DSM codec heeft als groot voordeel dat de broncode gelijk is aan de lijncode. De oversampeling bepaalt de kwaliteit van de DSM codec, echter voor acceptabele beeldkwaliteit zijn hoge bitrates nodig. Onderzoek wordt gedaan de verhouding beeldkwaliteit / bitrate zo hoog mogelijk te krijgen.

De maximale amplitude van het ingangsignaal is ook van belang. In formule:

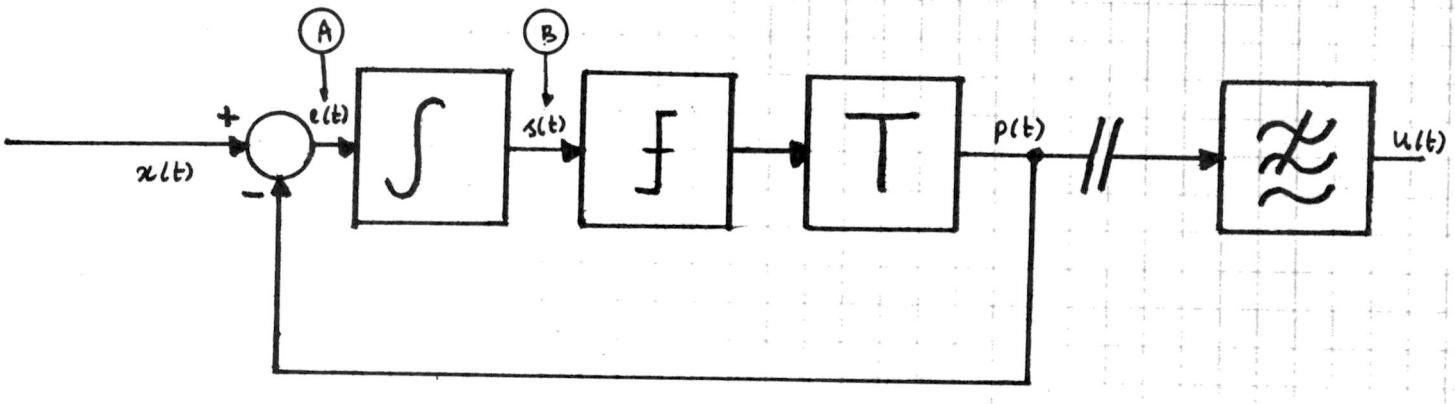
$$U_{max} = U * (1 - 2b).$$

formule 2.3

met U = amplitude uitgang van DSM
U max = maximale ingangs amplitude.

Deze beperking is nodig omdat in termen van F.M. , de zwaai van het uitgangssignaal binnen de basisband onverantwoord groot wordt. Zie voor bewijs Coenen (lit.3).

2.2 Realisatie van een DSM

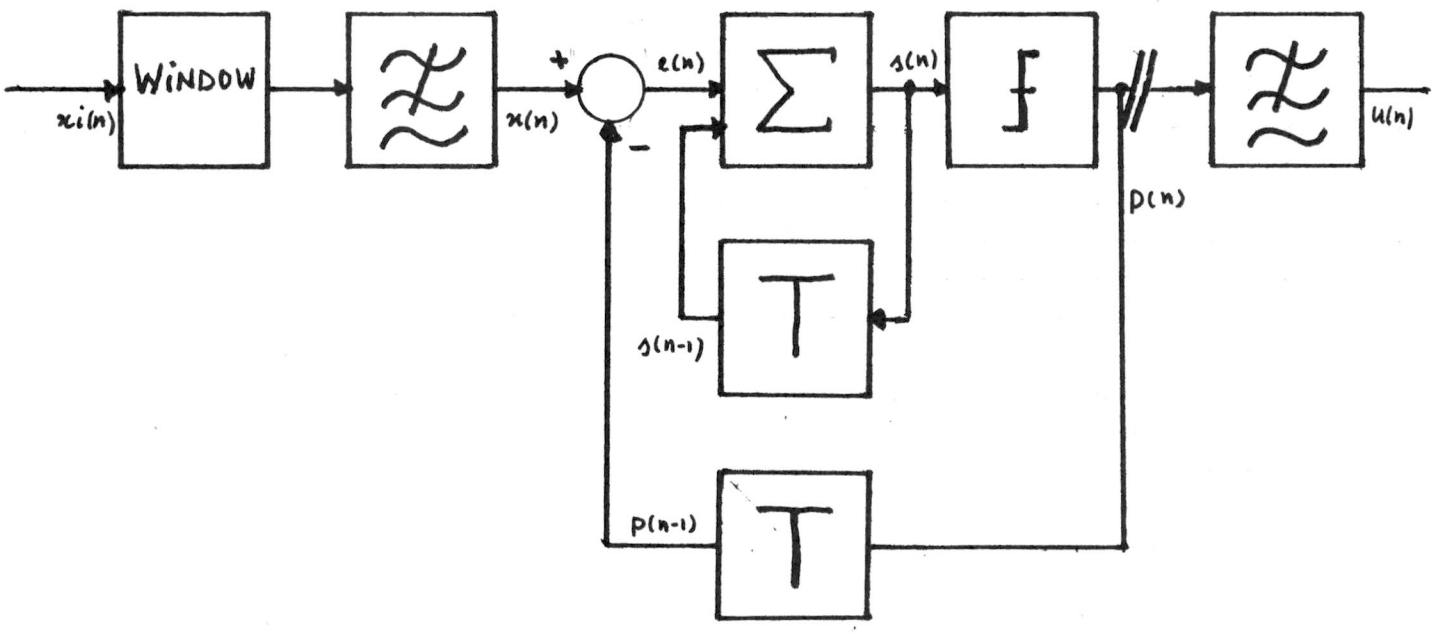


Figuur 2.1
Delta Sigma modulator schema.

Fig. 2.1 geeft een voorbeeld van de werking van een DSM. Het ingangsignaal x(t) (basisbandbegrensd) wordt verminderd met het uitgangsignaal p(t), zodat een amplitude-verschil e(t) ontstaat (punt A). Dit signaal op het hierna te noemen aftrekpunt bevat driehoeks ruis, d.w.z. het ruisvermogen neemt toe met toenemende frequentie. Er is namelijk

emperisch bepaald dat het ruis spectrum van s(t) voor de comperator vlak is. Het signaal op punt A zal verder verschilsignaal genoemd worden. Vervolgens word dit signaal geintegreerd. Dit geintegreerde verschilsignaal s(t) is aangeduid op punt B. Van s(t) wordt het teken bepaald door de comperator. Afhankelijk van dit signaal wordt een hoog dan wel een laag signaal p(t) verzonden en teruggekoppeld. Eenmaal verzonden hoeft men alleen het basisband gemiddelde van het signaal p(t) te bepalen (laagdoorlaat filteren) om het signaal x(t) terug te winnen.

2.3 Simulatie van een DSM op een computer.



Figuur 2.2
Schema voor een computergesimuleerde DSM.

Fig. 2.2 geeft een voorbeeld van een gesimuleerde DSM. Allereerst wordt het ingangsignaal $x_i(n)$ "getapered" tot x(n). Dit betekent dat het ingangsignaal aan begin en eind van het sample-array vermenigvuldigd wordt met een cosinus-kwadraad functie om kop en staart van de array vloeiend aan te laten sluiten. Dit is namelijk een voorwaarde voor een

bewerking met behulp van de fouriertransformatie. De Fouriertranformatie gaat namelijk uit van een herhalend patroon. Zou de kop niet vloeiend op de staart aansluiten, dan ontstaat er een stapfunctie welke het spectrum beïnvloedt. In het schema van fig 2.2 is dit het blokje genaamd "window".

De integrator is vervangen door een combinatie van een sommator met zijn 1 klokpulsvertraagde uitgang s(n-1). Deze integrator werkt allen op discrete tijdstippen, terwijl in de werkelijkheid de integrator continu integreert.

Wat verder opvalt is dat de terugkoppeltijdvertraging in de terugkoppellus is geplaatst. Dit zou betekenen dat de uitgang synchroon en zonder tijdsvertraging zou volgen. In een praktische realisatie wordt deze tijd gebruikt om de comperator in te stellen, maar aangezien de signalen reeds tijddiscreet zijn, is het geheel op deze manier geïmplementeerd. Het enige voordeel is dat de array's van ingang en uitgang geen tijdvertraging ten opzichte van elkaar krijgen. Dat de praktische realisatie wel een tijdsvertraging heeft, moet goed onthouden worden. Voor signaal-ruis berekeningen heeft dit verder geen consequenties. Voor simulaties van een DSM codec bij lage oversample-factoren wordt verwezen naar Ossewaarde (lit.4).

2.4 Resultaten van een simulatie

In fig 2.3 is een DSM ingangsignaal x(n) getekend met er doorheen het gefilterde uitgangssignaal u(n). De signalen zijn basisband gefilterd op 5 MHz. De signaal frequentie is 1.1 MHz genomen. Dit is niet precies 1 MHz, omdat rekening is gehouden om bij het signaal de kop op de staart vloeiend over te laten lopen. In 512 samples is 1.1 MHz een goed passende waarde. De amplitude van het ingangsignaal ligt op 0.7, bij een eenheidsuitsturing. Duidelijk zijn in de toppen enige vervormingen te zien.

In fig 2.4 is het DSM signaal p(n) te zien, gemengd met

het gefilterde DSM signaal u(n). De gegevens van deze simulatie zijn gelijk aan die van figuur 2.3. Vervolgens staat op fig 2.5 het verschilsignaal e(n). Duidelijk is de vorm van het ingangssignaal er in te herkennen. Let op de periodiciteit die in het signaal voorkomt.

In fig 2.6 is het geïntegreerde verschilsignaal s(n) afgedrukt. Ook hier is weer duidelijk de vorm van het ingangssignaal te herkennen.
De signaal- ruisverhouding is hierbij berekend. Hiervoor is genomen een som van kwadratisch ingangssignaal gedeeld door een som van kwadratisch verschil van in en uitgang. In formule:

$$S/N = 10 \log \frac{(u(t))^2}{(u(t) - x(t))^2}$$

formule 2.4

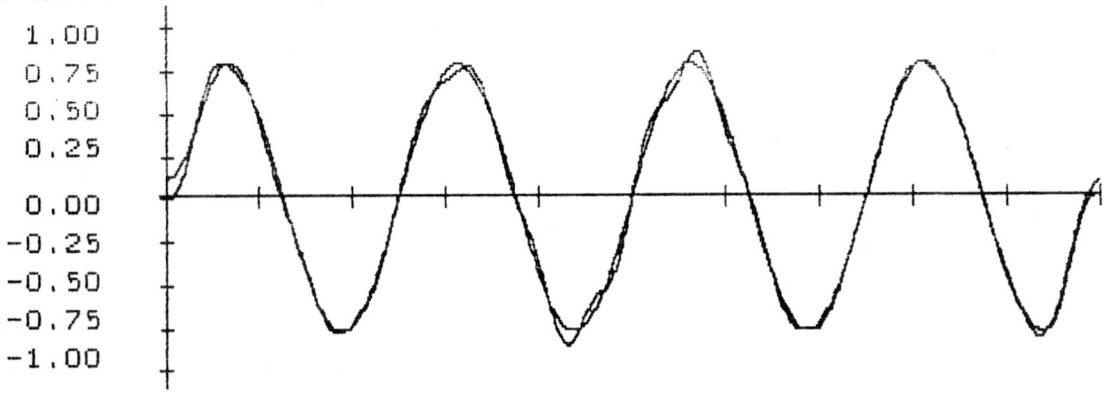
of in het tijddiscrete analogon:

$$S/N = 10 \log \frac{\Sigma (u(n))^2}{\Sigma (u(n) - x(n))^2}$$

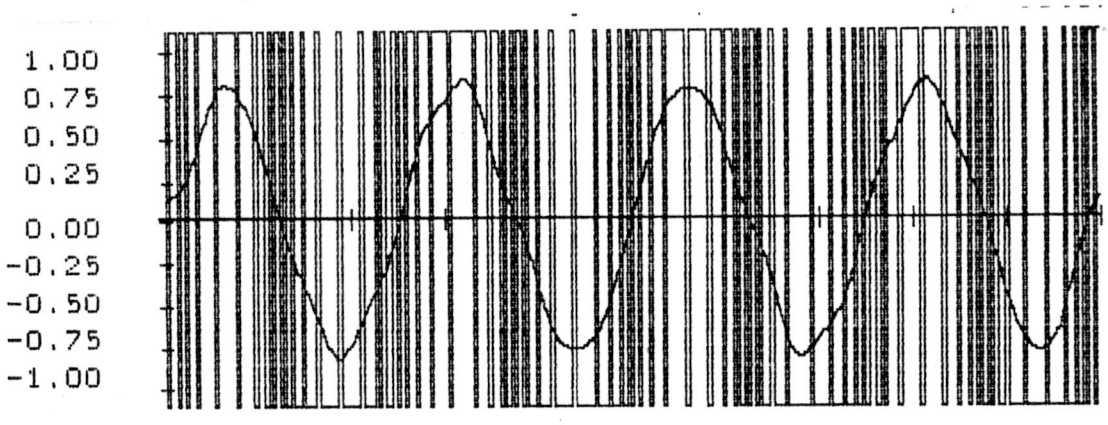
formule 2.5

De som werd alleen over het middenstuk van het sample-array genomen, de begin en eindstukken bevatten inschakelverschijnselen en overgangs-verschijnselen, veroorzaakt door onder andere het laag doorlaat filter. Voor de signaal- ruisverhouding zijn zo de eerste 10 en laatste 10 samples niet meegenomen. Een sinus van 1 Mhz met een basisband van 5 Mhz komt zo op een signaal-ruisverhouding van 23 dB, wat een te verwachten waarde is. Zie hiervoor ook Ossewaarde e.a. (lit.4) Deze waarde is de signaal-ruisverhouding tussen signaal x(n) en u(n). In het vervolg

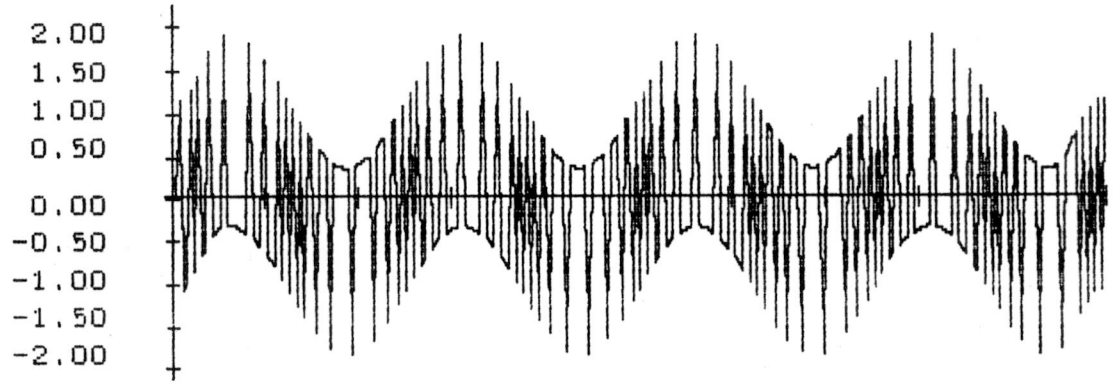
zal bij de signaal-ruis verhouding van deze twee signalen uitgegaan worden. De amplitude is dan maximaal gekozen, ongeveer 0.7, bij een eenheidsuitsturing op p(n).



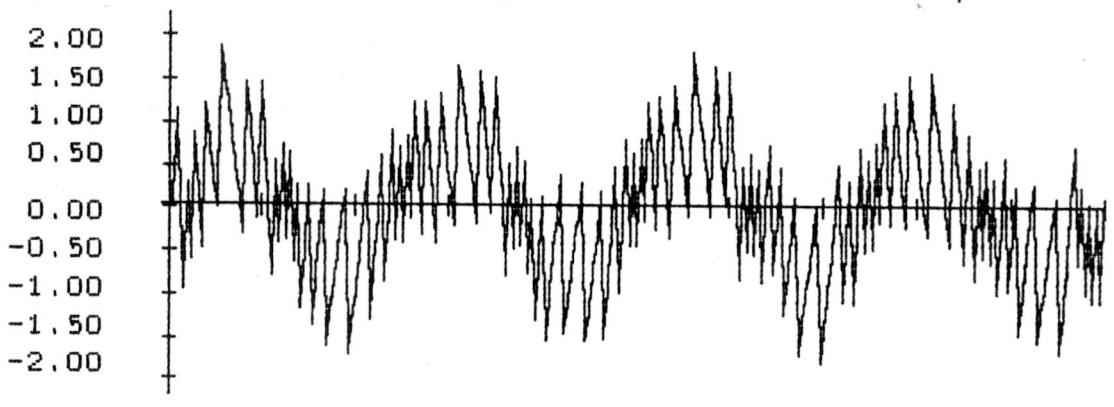
Figuur 2.3
Originele signaal x(n) met gefilterd DSM signaal u(n).



Figuur 2.4
DSM signaal p(n) met gefilterd DSM signaal u(n).



Figuur 2.5
Het verschil signaal e(n).



Figuur 2.6
Het signaal na de integrator s(n).

3:Toepassen van een tweetraps DSM

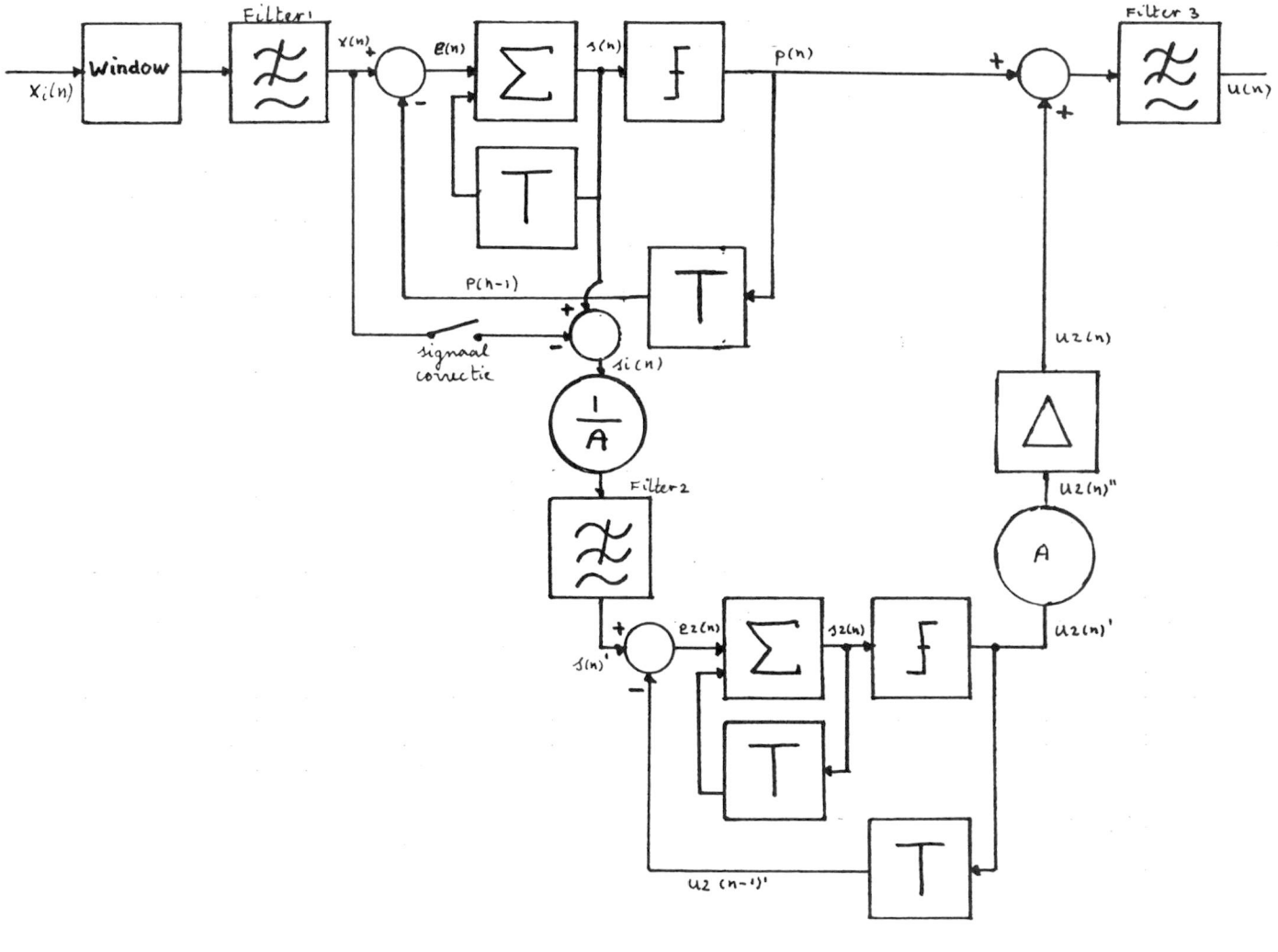
3.1 basisschema van een tweetraps DSM

In fig 3.1 staat het basisschema van een tweetraps Delta Sigma Modulator. Eerst zal kort de werking ervan besproken worden. In de lijn tussen in- en uitgang herkennen we weer een 1 traps delta sigma modulator. Het bijzondere aan deze modulator is, dat het geïntegreerde verschilsignaal s(n) basisbandgefilterd naar een tweede DSM wordt gevoerd. Zoals in hoofdstuk 2 al aan de orde is gekomen, blijkt hier vlakke ruis aanwezig te zijn. Eerdere pogingen van de vakgroep om direct het verschilsignaal e(n) te coderen zijn matig gebleken, omdat door de grote amplitude van de hoge basisband frequentiecomponenten van dit signaal deze na basisbandfiltering nauwelijks in amplitude terug te dringen waren. In het verschilsignaal e(n) is namelijk driehoeksruis aanwezig, waardoor de hoge frequentie componenten hier grote amplitudes hebben ten opzichte van de lage frequentie componenten. De hoge frequentie componenten zijn hier dus een belemmering om het signaal e(n) door een tweede DSM codec goed te kunnen coderen.

Het geïntegreerde verschilsignaal s(n) wordt basisbandgefilterd, (filter 2) om aliasing (vouwoverspraak) in de tweede codec te voorkomen. Nadat het geïntegreerde verschilsignaal s(n) gefilterd is, wordt dit met een constante vermenigvuldigd (factor 1/A). Dit wordt gedaan om de tweede DSM een qua amplitude zo gunstig mogelijke waarde te geven (zie formule 2.3), wat resulteerd in signaal s(n)'. De uitgang van de tweede DSM moet voor gebruik natuurlijk eerst gedifferentieerd worden, omdat we met het geïntegreerde verschilsignaal te doen hebben. Tenslotte wordt het geheel van signalen u2(n) en p(n) samengevoegd en opnieuw basisband gefilterd (filter 3). Dit filteren gebeurt natuurlijk pas in de decoder op afstand, maar bij deze simulatie werd het gelijk gedaan, mede om in en uitgang te

kunnen vergelijken.

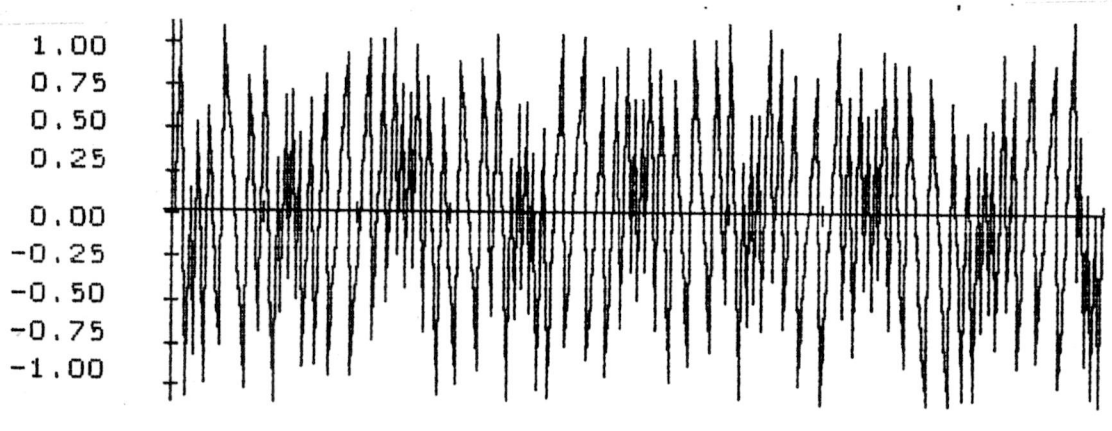
Opgemerkt moet worden dat in de simulatie alle filters van een ideaal laagdoorlaattype zijn. Deze filters zijn niet causaal en zijn dus van het type 'lineaire fase'. In 'analoge' werkelijkheid zullen steile filters een behoorlijke faseverschuiving tot gevolg kunnen hebben.



Figuur 3.1
Schema tweetraps DSM met signaal correctie.

3.2 Werking van signaal-correctie

In het schema staat ook een schakelaar afgebeeld die het ingangsignaal aftrekt van het geïntegreerde verschilsignaal (signaal-correctie). Er blijkt namelijk dat het ingangssignaal op het geïntegreerde verschilsignaal het ingangsignaal gesuperponeerd aanwezig is. Het voordeel van deze aftrekking is dat het zoverkregen signaal si(n) een vlakker verdeeld amplitude verloop heeft, en de tweede DSM daardoor twee maal efficiënter aangestuurd kan worden. Deze factor 2 die het ingangsignaal si(n) t.o.v. het signaal s(n) in amplitude scheelt, levert dus een behoorlijke winst voor de tweede DSM op. Een tekening van gecorrigeerde geïntegreerde verschilsignaal s(n) is gegeven in fig 3.2. Vergelijk dit signaal met fig. 2.6.



Figuur 3.2
Het gecorrigeerde geïntegreerd verschilsignaal.

Natuurlijk is geprobeerd om de invloed van het aftrekken van het signaal x(n) van s(n) te berekenen of te beredeneren. Allereerst zijn niet alle DSM bewerkingen in het Z-domein uit te drukken. Dit komt bijvoorbeeld door het niet-lineair zijn van de comperator.

Als beredeneerd wordt wat de gevolgen zijn van het aftrekken van x(n) bij s(n) dan kan dat het beste vergeleken worden met het geval zonder aftrekken.
Als s(n) niet met x(n) verminderd wordt, is de werking van de tweede DSM in principe geheel correct. Het signaal s(n) wordt namelijk gecodeerd en later zo goed mogelijk weer gereconstrueerd (in fig. 3.1: u2(n)'). Het enige nadeel is de dubbele amplitude ten opzichte van de eenheidsuitsturing die in het signaal voorkomt (zie ook fig. 2.6). In het geval met aftrekken van x(n) bij s(n) wordt het uitgangssignaal u2(n)' dus weer de gedecodeerde s(n) met daarvan afgetrokken de gedecodeerde x(n), de signalen x(n) en s(n) zijn superponeerbaar. Omdat het geïntegreerde foutsignaal s(n) nog gedifferentieerd moet worden, wordt ook de gesuperponeerde x(n) gedifferentieerd.

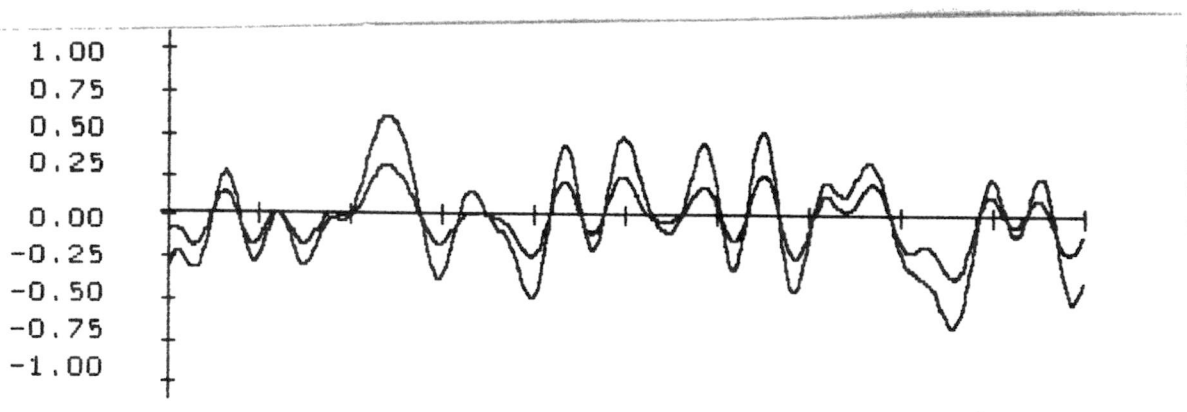
Nu lijkt het erop dat het deel afkomstig van x(n) een fout introduceert, afhankelijk van amplitude en frequentie van het ingangsignaal. Dit is bij de simulatie echter niet naar voren gekomen. Bij een 2 traps DSM waar het signaal x(n) niet van s(n) afgetrokken werd moest het signaal u2(n) met een 1T vertraagd voordat het bij het signaal van de eerste DSM kon worden opgeteld. Dit is gedaan om de beste signaal-ruisverhouding te krijgen. Werd de signaal-correctie (=aftrekken van x(n) bij s(n)) toegepast, dan moest het uitgangsignaal u2(n) voor of na de filtering 1T vertraagd worden. In het ideale geval (testopstelling) dat de twee-traps DSM het signaal Si(n) rechtstreeks doorverbond met u2(n) was leverde de signaal- gecorrigeerde tweetraps DSM in vergelijking met een niet signaal-gecorrigeerde tweetraps DSM dezelfde signaal-ruisverhouding op, n.l. rond de 100 dB.

Concluderend mag gezegd worden dat de signaal-correctie op het uitgangssignaal alleen een tijdverschuiving tot gevolg heeft. Zodoende zijn de hierna volgende resultaten allen met signaal-correctie bij een tweetraps DSM.

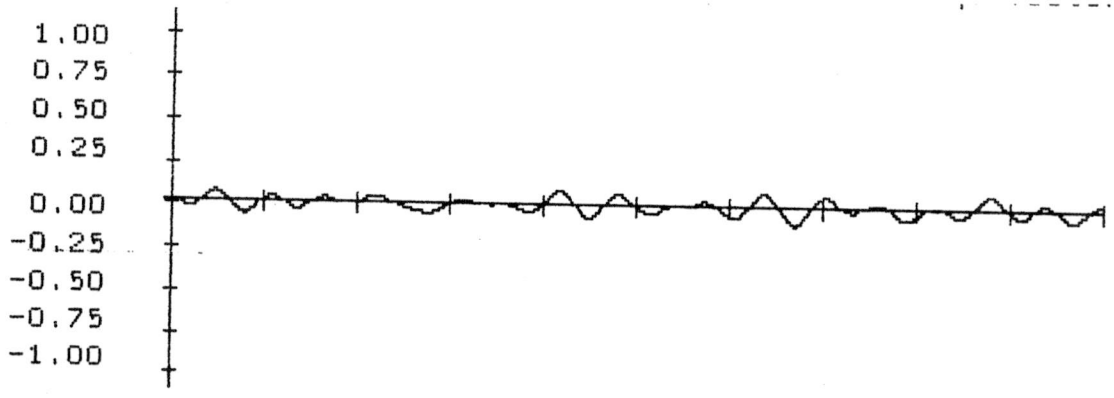
3.3 Enige resultaten van de tweetraps DSM

De resultaten zijn het beste te zien aan de hand van enkele plaatjes. Afbeelding 3.3 laat 2 signalen zien: Ten eerste het gefilterde geïntegreerde verschilsignaal s(n)', dit signaal is vermenigvuldigd met 1/A. A is hier 0.5, de topwaarde van het signaal word zo ongeveer 0.65. Later zullen nog enkele andere waarden van A geprobeerd worden. De frequentie van het ingangssignaal xi(n) is 1.1 MHz, met een amplitude van 0.70 bij een eenheidsuitsturing op p(n). De filters zijn allen met een afsnijfrequentie op 5 Mhz.

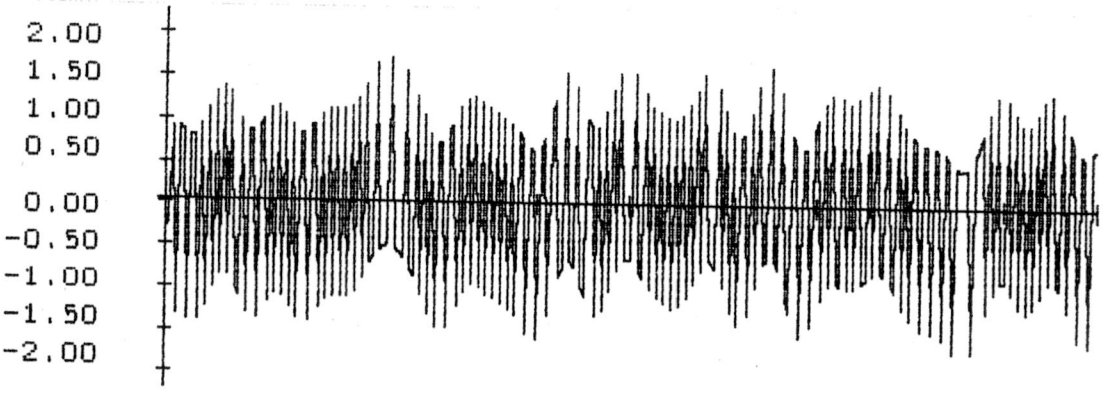
Het tweede signaal in afbeelding 3.3 is het signaal nadat het door de tweede DSM is gecodeerd, gefilterd en weer vermenigvuldigd is met een factor A (signaal u2(n)''). Tenslotte staat het gedifferentieerde signaal u2(n) van de tweede DSM in fig 3.4. Wat opvalt is de kleine amplitude en de onregelmatige structuur. Een plaatje van ingangssignaal en tweetraps DSM uitgang heeft weinig zin: de ingang wordt zo goed gevolgd dat verschillen niet meer op het oog te zien zijn. Voor de volledigheid zijn in fig 3.5 en 3.6 het verschilsignaal e2(n) resp. het geïntegreerde verschilsignaal s2(n) getekend.



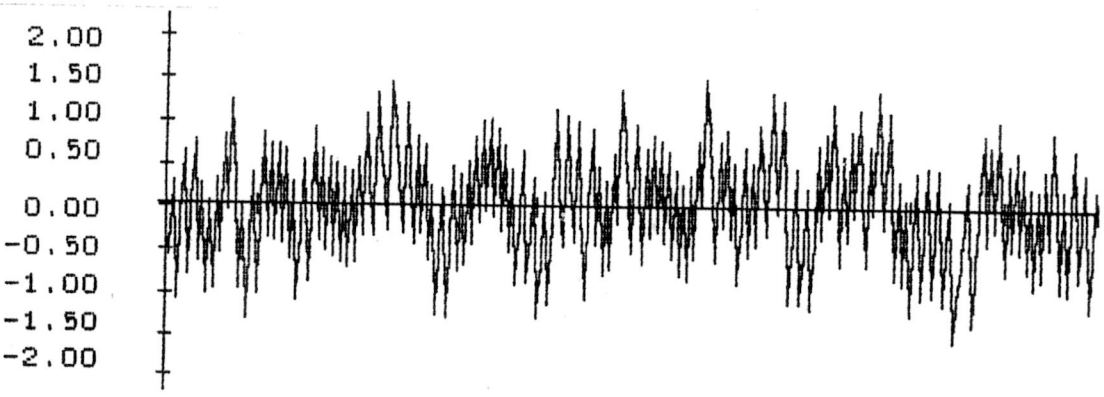
Figuur 3.3
Gefilterd geïntegreerd verschilsignaal si(n) en gefilterd uitgang u2(n)' van tweede DSM.



Figuur 3.4
Gedifferentieerde uitgang u2(n) van de tweede DSM.



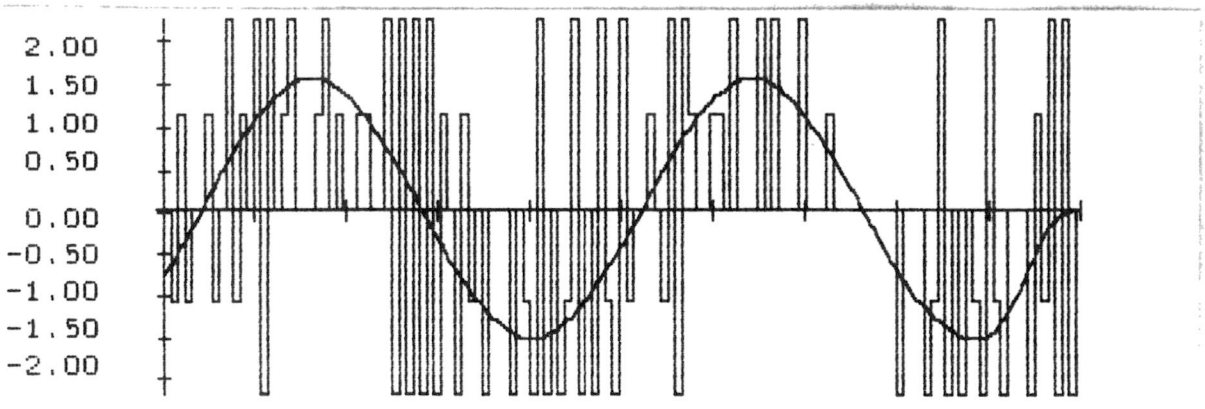
Figuur 3.5
Het verschilsignaal e2(n) op de tweede DSM.



Figuur 3.6
Het geïntegreerde verschilsignaal s2(n) op de tweede DSM.

Wat belangrijk is om op te merken, is dat het gefilterde geïntegreerde verschilsignaal s(n) van de eerste DSM niet ideaal is om te coderen. Dit signaal heeft namelijk een vrij lage amplitude, vergeleken met de ideale maximum amplitude, afgezien van een paar pieken. Zou dit signaal gecodeerd worden in een enkel traps DSM, dan is een minder goede signaal- ruis verhouding te verwachten dan met een sinusvormig signaal met maximale amplitude uitsturing. Zo zal de bijdrage aan de signaal-ruisverhouding van de tweede DSM hoogstens gelijk zijn aan de signaal-ruis verhouding van de signalen si(n) en u2(n). Als de signaal- ruis verhoudingen van de eerste en tweede DSM opgeteld worden, is zo geen verdubbeling van de signaal- ruisverhouding van de eerste DSM te verwachten, maar een iets lagere signaal- ruis verhouding.

Fig. 3.7 laat als illustratie een eerste DSM signaal met een daarbij opgeteld tweede DSM uitgangssignaal (p(n)+u2(n)) zien. Het gefilterde signaal u(n) is er doorheen getekend. De grafiek is wat vergroot in de breedte om de pulsen beter te kunnen onderscheiden.



Figuur 3.7
Eerste DSM met tweede DSM signaal samengevoegd. Er door heen het gefilterde resultaat. Figuur laat laatste 125 Samples zien.

4 Berekeningen aan het model.

4.1 S/N verhoudingen van een twee traps DSM.

De resultaten van enkele metingen staan in tabel 4.1. In de bovenste rij staan de S/N verhoudingen van een eentraps DSM, zonder Noise Shaping of andere pogingen tot verbetering van de S/N verhouding. Bij de metingen kan men in werkelijkheid ongeveer 3 dB optellen om de waarden bij een praktische realisatie te verkrijgen. Dit komt doordat de simulatie uitgaat van tijddiscrete signalen in plaats van continue signalen.

Onder de 1e rij resultaten van van tabel 4.1 staan de S/N verhoudingen van de twee traps DSM bij diverse signaalfrequenties. De 3e rij wordt later besproken. De resultaten van de tweede rij kunnen niet zondermeer met die van de eerste rij vergeleken worden. De eerste rij van tabel 4.1 heeft namelijk een enkele DSM met 70 Mhz zonder noise shaping, dus 70 Mbit, terwijl de twee traps DSM 2 * 70 Mhz gebruikt, dus 140 Mbit. Om te vergelijken zijn tabellen 4.2 en 4.3 toegevoegd. Hierin staan in tabel 4.2 de signaal ruis verhoudingen van een enkele DSM op 140 Mbit en in tabel 4.3 de signaal ruis verhoudingen van een tweetraps DSM op 2 * 35 Mhz = 70 Mbit.

Opgemerkt dient te worden dat er te weinig punten zijn om een betrouwbare weergave in grafiek vorm te geven.

De conclusie die getrokken kan worden als de enkele DSM met een tweetraps DSM (beide op 70 Mbit) vergeleken wordt, blijkt de twee traps DSM ongeveer 10 dB onder de enkele DSM te zitten. Worden de beide DSM's op 140 Mbit vergeleken, dan zit de signaal- ruis verhouding van de tweetraps DSM gemiddeld een factor 5 dB boven de signaal- ruis verhouding van de enkele DSM. Als de tweetraps DSM op 70 Mbit werkt, is de oversampelingsfactor 4.5 voor de tweetraps DSM. Deze waarde is te laag om de DSM's goed te laten werken.

Het uiteindelijke doel om op 2 x 35 Mbit een beter resultaat te krijgen met een tweede DSM in plaats van een enkele DSM, zal volgens deze simulatie niet slagen.

De signaal- ruiswaarden op 140 Mbit zijn bij de twee-traps DSM hoger dan een normale DSM op 140 Mbit, maar echter niet zo hoog als gehoopt. Volgens Steele (lit.1) codeert de tweede DSM namelijk zo, dat de signaal ruisverhouding van de tweede coder bij die van de hoofdcoder mag worden opgeteld. De winst is echter niet 24 dB, maar ongeveer 13 dB. Deels is dit te wijten aan de niet ideale amplitudeverdeling van het geintegreerde verschilsignaal s(n)', en deels ook aan het vervormen van het signaal si(n) door het laagdoorlaatfilter (filter 2). Dit filter "strijkt" het signaal min of meer glad, waardoor wat informatie verloren gaat.

sign.freqentie	0.5 Mhz	1 Mhz	2 Mhz	3 Mhz	4 Mhz
enkele DSM	27.3 dB	25.9 dB	25.1 dB	26.4 dB	18.1 dB
2-traps DSM	39.7 dB	37.0 dB	38.1 dB	38.3 dB	40.7 dB
noise shaped 2-traps DSM	39.0 dB	40.1 dB	42.6 dB	41.6 dB	40.3 dB

Tabel 4.1
Signaal-ruis verhoudingen bij verschillende frequenties.
Alle ingangsamplitudes zijn maximaal genomen. Alle filters zijn ideaal met afsnijfrequentie op 5 MHz. De factor A is 0.5, de bemonsterfrequenties van alle DSM zijn 70 MHz. Window is cosinus kwadraad functie op 5% van begin en eind waarden van samplearray xi(n).

sign.freqentie	0.5 Mhz	1 Mhz	2 Mhz	3 Mhz	4 Mhz
enkele DSM 140 Mhz	40.5 dB	34.1 dB	29.3 dB	35.6 dB	29.1 dB

Tabel 4.2
Signaal-ruis verhouding enkele DSM op 140 MHz.
Voor gegevens zie tabel 4.1.

sign.freqentie	0.5 Mhz	1 Mhz	2 Mhz	3 Mhz	4 Mhz
2-traps DSM 2 * 35 Mhz	17.3 dB	17.3 dB	16.6 dB	14.4 dB	17.8 dB

Tabel 4.3
Signaal-ruis verhouding tweetraps DSM op 2 x 35 Mhz.
Voor gegevens zie tabel 4.1.

4.2 Variëren van constanten.

In de computersimulatie zijn enkele in de praktijk constante waarden veranderd. Men denke hierbij aan de factor A, of de waarde van de afsnijfrequentie van het tweetraps-filter (filter 2).

Om te beginnen is de waarde van de vermenigvuldiging veranderd. In fig 3.1 is dit aangegeven met de factoren 1/A en A. In hoofdstuk 2 is al ter sprake gekomen dat een juiste amplitude aan de ingang van de DSM betere resultaten geeft dan een te grote of te kleine. Verschillende waarden zijn afgedrukt in tabel 4.4. Hiernaar kijkend valt een piek op rond

A = 0.40. Dit was ook wel te verwachten, als men kijkt naar de amplitude van het signaal Si(n), dit nivo ligt hier rond 0.35 (zie ook fig 3.3).
Het dal bij A = 0.60 kan in dit verslag niet verklaard worden. Andere waarden van A dicht rond A = 0.60 wezen aan dat het gebied rond A = 0.60 een lage signaal/ruis verhouding geeft.

A	0.80	0.70	0.60	0.50	0.40	0.30	0.20
2-tr.DSM S/N	32.5 dB	37.3 dB	31.5 dB	37.0 dB	41.0 dB	39.1 dB	36.2 dB

Tabel 4.4
De invloed van de correctiefactor A.
Bemonsterfrequentie beide DSM's op 70 Mhz, alle filters ideaal op 5 MHz. Window is cosinus kwadraad functie op 5% van begin en eind van sample array xi(n).

De afsnijfrequentie van het tweetraps DSM filter (filter 2) is ook van belang. In de praktijk kan namelijk alleen een filter met een brede transitiedemping worden gebruikt, bovendien moet het een lineair fase filter zijn. Als een niet-lineair-fase filter gebruikt word, ontstaan looptijdsverschillen tussen verschillende frequentiecomponenten. Het ontvanger filter (filter 3 in fig 3.1) kan wel steil zijn, omdat beide DSM componenten u2(n) en p(n) door hetzelfde filter en gelijkertijd gefilterd kunnen worden.

De resultaten van het variëren van de afsnijfrequentie van filter 2 staan in tabel 4.5. Hieruit blijkt dat men heel goed een laagdoorlaat filter met een hogere afsnijfrequentie kan gebruiken. Een voorzichtige conclusie kan zijn dat men de afsnijfrequentie van het filter beter wat hoger dan 5 Mhz kan kiezen, de resultaten worden dan beter. Er moet wel opgelet worden dat men dan de factor A dan wat vergroot, immers

doordat de afsnijfrequentie van de filtering wat hoger komt komen er hogere pieken in het signaal ten gevolge van de hogere frequentie componenten. De tegenvallende waarde van het 9 Mhz filter komt waarschijnlijk omdat er daar precies wat storing (door filtering) in blijft zitten. Een simulatie met meer samplewaarden (b.v. 10.000 punten) zou beter zijn.

filter corr.DSM	4 Mhz	5 Mhz	7 Mhz	9 Mhz	11 Mhz	13 Mhz
2-traps DSM S/N	27.5 dB	37.0 dB	38.3 dB	35.6 dB	39.3 dB	34.1 dB

Tabel 4.5
De invloed van de filterfrequentie van filter 2. Correctie factor A = 0.5, verdere gegevens als in tabel 4.4.

4.3 Betrouwbaarheid van de uitkomsten.

Natuurlijk vraagt U zich met al deze waarden af, welke betrouwbaarheid de getallen hebben. Een opsomming van factoren die bekeken zijn volgt nog. Wat is nu zo belangrijk? Ten eerste moeten het eerste DSM signaal p(n) met het daarbij behorende foutsignaal u2(n) in fase zijn. Dit foutsignaal is dan ook verschoven in de tijd, tot 1/2 T vooruit en 1/2 T achteruit in de tijd. De vertraging 1/2 T bijvoorbeeld werd veroorzaakt door het samplearray slap te filteren volgens de formule: array[n] = (array[n-1] + array[n])/2. Dit maakte het resultaat alleen maar slechter, of er nu voor of achterwaarts werd geschoven.

Zoals al eerder ter sprake is geweest, is de simulatie ook uitgevoerd zonder DSM codering, de array's zijn dus letterlijk doorgeschoven. In de figuren komt dit neer op x(n) door te verbinden met p(n) en si(n) door te verbinden met u2(n)". In een enkele DSM leverde dat voor de berekende

signaal-ruisverhouding een waarde van 245 dB op, en in de tweetraps DSM, waarbij wel in de tweetraps DSM gedifferentieerd is, ruim 100 dB. Dit zakt zo sterk omdat de rekennauwkeurigheid van de gebruikte computer slechts tot 1E-38 ging met 11 significante cijfers. In ieder geval was de 100 dB geen beperkende factor voor de nauwkeurigheid.

Een andere controle voor de complete tweetraps DSM was de werking van het laatste filter (filter 3 uit fig 3.1). Dit filter kan namelijk gesplitst worden in twee (identieke) filters, waarbij het filter aan de tweetraps DSM kant nog voor de differentiator geschoven kan worden. Dit is gecontroleerd en ik verkreeg dezelfde resultaten als met een filter met de gesommeerde signalen u2(n) en p(n).

Er is verder geprobeerd het tweetraps signaal wat te versterken of te verzwakken ten opzichte van het eerste DSM signaal, echter met verminderingin het resultaat.

Tenslotte is er geprobeerd de signaal-correctie weer te compenseren door er later weer het (DSM-bewerkte) signaal p(n) bij u2(n)'' op te tellen. Dit verslechterde het resultaat. De verklaring van dit verschijnsel valt buiten de taak.

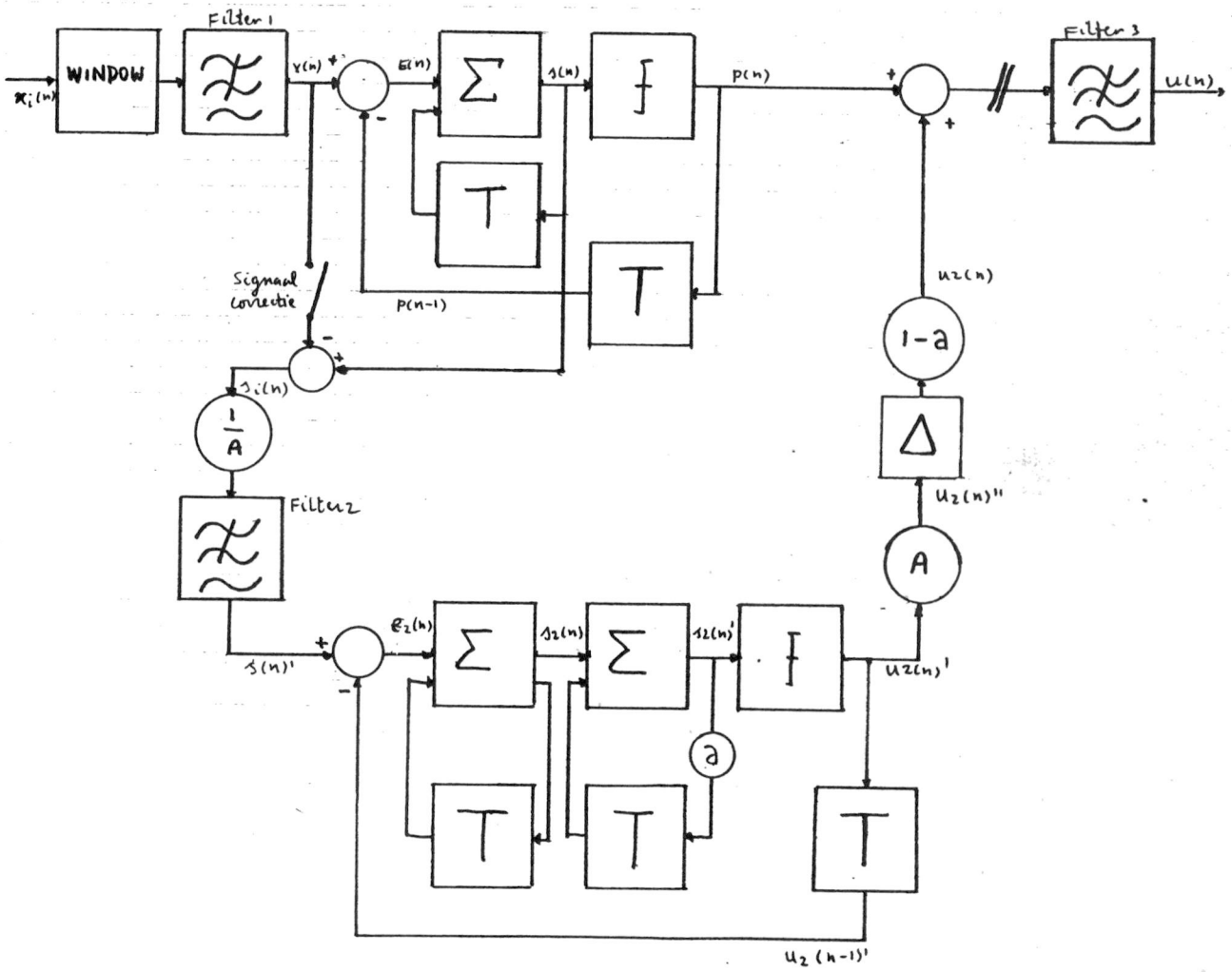
5 : Verbeteringen aan een tweetraps DSM.

5.1 Noise shaping aan de tweetraps DSM

Om de simulatieresultaten wat te verbeteren, is geprobeerd noise-shaping toe te passen. Dit is te zien in fig. 5.1. De noise-shaping is alleen toegepast op het tweede DSM gedeelte, omdat anders faseverschuivingen zouden ontstaan tussen de eerste DSM signaal p(n) en het signaal u2(n). Deze situatie is wel gesimuleerd, maar de resultaten waren verslechterd.

De noise-shaping wordt gevormd door de integratie aktie plus een kleine extra integratie in de voorwaardse weg van de DSM. In figuur 5.1 wordt s2(n) bewerkt tot s2(n)'. De in figuur 5.1 getekende realisatie resulteerd in een versterking van het signaal u2(n) met een factor 1/(1-a). Het signaal wordt daarom met een vermenigvuldiging van (1-a) gecorrigeerd. Enige verbetering treedt zo wel op. Opvallend is echter dat de verbetering hoofdzakelijk op de middenfrequenties, 1 - 3 Mhz plaatsvind.

Enige resultaten staan in tabel 4.1, zodat een goede vergelijking mogelijk is met coders zonder noise-shaping. Als waarde voor a (de noise-shaping factor) is tijdens de simulatie 0.25 genomen. Dit gaf de beste resulaten, zoals ook al door Ossewaarde en van Bergen Henegouwen (lit.2 en lit.4) is aangegeven.



Figuur 5.1
Schema van een tweetraps DSM met extra noise-shaping.

5.2 Simulatie aspecten.

De waarden die berekend zijn, vallen eigenlijk wat tegen. Hiervoor kunnen verschillende oorzaken debet aan zijn. Ten eerste werkt de simulatie met tijddiscrete en amplitude gekwantiseerde signalen, wat in de analoge werkelijkheid niet zo is.

Ten tweede scheelt het aantal samplewaarden waarbij de simulatie uitgevoerd wordt. Verschillen in signaal- ruis verhoudingen bij een verschillend aantal samples zijn wel bepaald. Bij een simulatie over 1024 punten gaf een bijna dezelfde signaal- ruis verhouding als een van 256 punten. Voorwaarde is wel de juiste frequentie te coderen, daar ander kop en staart van het sample-array niet vloeiend aan sluiten. Men dient op te passen met het filteren: als er in de eerste sample (waar soms een inschakel verschijnsel tot uitdrukking komt) een verkeerde en ongecontroleerde waarde zit, levert dit verkeerde filterresultaten op.

Ten derde is deze simulatie geheel digitaal en werkt dus met discrete tijdstippen. In werkelijkheid onstaan echter snel kleine faseverschillen, terwijl dat tijddiscreet onder de 1/2 T nauwelijks meer is te achterhalen.

Ten vierde zijn de filters in mijn geval ideaal, wat dus in werkelijkheid ook niet voor kan komen. Bovendien zijn ze in de simulatie allen gelijk, iets dat in werkelijkheid ook niet kan. Enkele digitale filters zijn ook gesimuleerd (herhaalde reschthoeksweging), maar deze gaven nauwelijks afwijkende resultaten.

6 : Conclusies

De tweetraps DSM, waarbij het geïntegreerde verschilsignaal van een normale DSM apart gecodeerd wordt en later bijgevoegd, kan in sommige gevallen een verbetering geven t.o.v. een DSM op dubbele samplefrequentie. De verbetering is echter niet zo veel als dat theoretisch verwacht zou kunnen worden. Debet hieraan is hoofdzakelijk het feit dat het geïntegreerde foutsignaal nog hoge frequentiecomponenten bevat, welke moeilijk codeerbaar zijn.

Andere verbeteringen als noise shaping in de tweetraps DSM konden de signaal-ruisverhouding niet op een acceptabele waarde krijgen.

Hoewel deze methode verbetering opleverd, loont het niet de moeite het in praktijk te gaan gebruiken. Men is namelijk het gemak van de oorspronkelijke DSM kwijt: men heeft namelijk een tijdmultiplex ingevoerd, waardoor de tweetraps DSM aanzienlijk uitgebreider is geworden, en waarschijnlijk lopen hier de kosten teveel op.

Appendix 1:

Gegevens over het simulatieprogramma.

De gebruikte software

Het simulatieprogramma is uiteindelijk geschreven in Pascal. Gebruikt is Turbo-Pascal version III, van Borland. Het programma is in bijlage 1 afgedrukt. Het laatste blad van het programma is een mini-handleiding. Oorspronkelijk ben ik ook op een andere machine, een Textronics desktop computer, aan een basic versie begonnen, maar daar liep ik vast op de fast-fourier transformatie, die rare resonantie verschijnselen vertoonde. Ook hulp van deskundigen konden deze problemen niet verhelpen.

Er is door mij voor de simulatie een fast-fourier-transformatie ontwikkeld volgens het Butterfly principe. Deze procedure heb ik zelf ontwikkeld, daar andere mij ter beschikking gestelde procedures niet snel genoeg waren. De procedure is gecontroleerd met de standaard definitie van de fourier transformatie.

Een ander set procedures is voor de grafische representatie van de grafieken, zowel op printer als op scherm. Resultaten zijn reeds getoond in de figuren.

De gebruikte hardware

De basic versie die helaas niet afgemaakt is, is geschreven op een tektronix desktop computer, met hoge grafische resolutie.

De pascal versie is geschreven op een IBM-compatibele Olivetti M24, welke de vakgroep inmiddels zelf in huis heeft. Deze computer heeft een grafische kaart waarbij de resolutie tot 320 x 200 is gebruikt. Dit is niet op alle IBM (achtige) machines standaard aanwezig.

De gebruikte printer is een normale standaard matrixprinter (grafisch, Epson-compatibel) welke intelligent door het programma bestuurd wordt.

Literatuurlijst

1: Steele, R. - Peak signal-to-noise ratio formulas for multistage delta modulation with RC-shaped Gaussian input signals.
The Bell Technical Journal vol.61,no 3, maart 1982.

2: Bergen Henegouwen, E.A.J.M. - Analyse van een aantal digitale 1-bit codecs.
Afstudeerverslag T.U.Delft, code 05-1-565-28-209, 1 februari 1984.

3: Coenen, A.J.R.M. - Digitale Modulatie.
College dictaat Informatie transmissie techniek.
T.U. Delft, 1983.

4: Ossewaarde, G. - Computersimulaties van digitale 1-bit codecs met lage oversample-factoren.
Afstudeerverslag T.U. Delft, code 05-1-565-28-211, 24 oktober 1983.

```
  1: program taak;
  2: {$C-}
  3: (*
  4:
  5:
  6:                    <<< VIVI - SOFT >>>
  7:                    Jan Vieveen
  8:                    DSM codec experiment
  9:                    Copyright  1986
 10:
 11:
 12: Dit programma rekent een correctie/gewone DSM door,
 13: en tekent grafieken op een IBM matrix printer.
 14:
 15: *)
 16:
 17: label      stop;
 18:
 19:
 20: const
 21: samples                  = 256;   (* aantal discrete tijdstippen *)
 22: bijschriftkleur          = 1;     (* kleuren voor grafiek        *)
 23: askleur                  = 1;     (* 1 = aan, 0 = uit            *)
 24: grafiek1kleur            = 1;
 25: grafiek2kleur            = 1;
 26:
 27:
 28: type
 29: str10                    = string[10];       (* string types voor functies *)
 30: str80                    = string[80];       (* en procedures              *)
 31: anystr                   = string[255];
 32: charset                  = set of char;
 33: werkarray                = array[1..samples] of real;
 34: tekenarray               = array[1..640,1..25] of integer;
 35:
 36: var
 37: recht,printing,genest,dc,                  (* halveT is hoofdlus 1/2T vertraagd,constant is      *)
 38: constant,halveT             : boolean;     (* voor printer,als recht is true dan  hoekige grafiek*)
 39: a0,a1,a2,a3,a4,a5,a6,a7,b0,                (* genest betekent met correctie van fase sig.        *)
 40: sinarray                    : werkarray;   (* Deze variabelen gelden in het hele programma       *)
 41: tekening                    : tekenarray;  (* 2 dimensionaal array voor de tekening              *)
 42: ch,tc                       : char;        (* Dit zijn dummy karakters                           *)
 43: x,y,n,i                     : integer;     (* x/y zijn voor assenkruis, rest dummy               *)
 44: m,fs,sf,u,v1,v2,integrator,afsnijfreq,     (* fs=freq.van sampelen, sf=freq.van ingangsign.      *)
 45: afsnijfreq2,A,tape,                        (* afsnijfreq2 is voor de corr.DSM                    *)
 46: schaal,uitgang,m1,m2,m3,m4   : real;       (* u=ampl.ingangsignaal,v1=vergel.spanning comp.      *)
 47:                                            (* integrator cummuleert ingangsignaal                *)
 48:
 49:
 50: (*  UpcaseStr converts a string to upper case *)
 51:
 52: function UpcaseStr(S : Str80) : Str80;
 53: var
 54:   P : Integer;
 55: begin
 56:   for P := 1 to Length(S) do
 57:     S[P] := Upcase(S[P]);
 58:   UpcaseStr := S;
 59: end;
 60:
 61: (*  ConstStr returns a string with N characters of value C *)
 62:
 63: function ConstStr(C : Char; N : Integer) : Str80;
 64: var
 65:   S : string[80];
 66: begin
 67:   if N < 0 then
```

```
  68:     N := 0;
  69:   S[0] := Chr(N);
  70:   FillChar(S[1],N,C);
  71:   ConstStr := S;
  72: end;
  73:
  74: (*  Beep sounds the terminal bell or beeper *)
  75:
  76: procedure Beep;
  77: begin
  78:   Write(^G);
  79: end;
  80:
  81: {* inputStr leest een string in van lengte L op plaats (X+1,Y+1),           *}
  82: {* beindiging door ingeven van karakter uit charset, welke retourneerd in TC *}
  83:
  84: procedure InputStr(var S     : AnyStr;
  85:                        L,X,Y : Integer;
  86:                        Term  : CharSet;
  87:                    var TC    : Char    );
  88: const
  89:   UnderScore  =  '_';
  90: var
  91:   P : Integer;
  92:   Ch: Char;
  93:   overwrite:boolean;
  94: begin
  95:   GotoXY(X + 1,Y + 1); Write(S,ConstStr(UnderScore,L - Length(S)));
  96:   P := 0;
  97:   overwrite:=true;
  98:   gotoxy(64,23);
  99:   write('mode: overwrite');
 100:   clreol;
 101:   repeat
 102:     GotoXY(X + P + 1,Y + 1); Read(Kbd,Ch);
 103:     if ch = #27 then
 104:        begin
 105:        read(kbd,ch);
 106:        if ch = #75  then ch := ^S;  {* left          ,vorige   letter*}
 107:        if ch = #77  then ch := ^D;  {* right         ,volgende letter*}
 108:        if ch = #72  then ch := ^F;  {* down          ,omlaag         *}
 109:        if ch = #80  then ch := ^A;  {* up            ,omhoog         *}
 110:        if ch = #59  then ch := ^Y;  {* functionkey 1 ,veeg uit,volgen*}
 111:        if ch = #60  then ch := ^Z;  {* functionkey 2 ,klaar          *}
 112:        if ch = #61  then ch := ^V;  {* functionkey 3 ,insert/overwrit*}
 113:        if ch = #62  then ch := ^G;  {* functionkey 4 ,veeg letter    *}
 114:        if ch = #115 then ch := ^E;  {* ctrl left     ,vorig art      *}
 115:        if ch = #116 then ch := ^X;  {* ctrl right    ,volgend art    *}
 116:        if ch = #71  then ch := ^Z;  {* ctrl up       ,klaar          *}
 117:        end;
 118:     if ch = #9 then ch := ^F;       {* S1            ,omlaag         *}
 119:     case Ch of
 120:       #32..#255 : if P < L then
 121:                   begin
 122:                     if not overwrite then
 123:                        begin
 124:                         if Length(S) = L then Delete(S,P,1);
 125:                         end
 126:                     else
 127:                         begin
 128:                         delete(S,P+1,1);
 129:                         end;
 130:                     P := P + 1;
 131:                     Insert(Ch,S,P);
 132:                     Write(Copy(S,P,L));
 133:                   end
 134:                   else Beep;
```

```
      ^S        : if P > 0 then
                    P := P - 1
                  else Beep;
      ^D        : if P < Length(S) then
                    P := P + 1
                  else Beep;
      ^A        : P := 0;
      ^F        : P := Length(S);
      ^G        : if P < Length(S) then
                  begin
                    Delete(S,P + 1,1);
                    Write(Copy(S,P + 1,L),UnderScore);
                  end;
      ^H,#127   : if P > 0 then
                  begin
                    Delete(S,P,1);
                    Write(^H,Copy(S,P,L),UnderScore);
                    P := P - 1;
                  end
                  else Beep;
      ^Y        : begin
                    Write(ConstStr(UnderScore,Length(S) - P));
                    Delete(S,P + 1,L);
                    ch := ^M;
                  end;
      ^V        : begin
                      overwrite:=not overwrite;
                      gotoxy(70,23);
                      if overwrite then write('overwrite')
                      else write('insert');
                      clreol;
                   end;
    else
      if not (Ch in Term) then Beep;
    end;  {of case}
  until Ch in Term;
  P := Length(S);
  GotoXY(X + P + 1,Y + 1);
  Write('' :L - P);
  TC := Ch;
end;

{* select drukt prompt af, en laat kiezen uit term. Resultaat in upcase in TC *}

procedure Select(    Prompt : Str80;
                     Term   : CharSet;
                 var TC     : Char    );
var
  Ch : Char;
begin
  GotoXY(1,23); Write(Prompt,'? ' ); ClrEol;
  repeat
    Read(Kbd,Ch);
    TC := Upcase(Ch);
    if not (TC in Term) then
      Beep;
  until TC in Term;
  Write(Ch);
end;

{* Convert leest getallen in volgens inputstr, met z cijfers achter de komma *}

procedure convert(var getal  : real;
                      l,x,y,z: integer;
                      term   : charset;
                  var tc     : char  );
```

```
201: var   g : anystr;
202:     code : integer;
203: begin
204: repeat
205:     str(getal:1:z,g);
206:     inputstr(g,1,x,y,term,tc);
207:     val(g,getal,code);
208:     if code <> 0 then beep;
209: until code = 0;
210: end;
211:
212:
213:
214:
215: {*****************************************************************************}
216: {********** MS-DOS Functions   by Jan Vieveen  ******************************}
217: {*****************************************************************************}
218:
219: function Date: str10;
220: type
221:   regpack = record
222:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
223:             end;
224:
225: var
226:   recpack:       regpack;                {record for MsDos call}
227:   month,day:     string[2];
228:   year:          string[4];
229:   dx,cx:         integer;
230:
231: begin
232:   with recpack do
233:   begin
234:     ax := $2a shl 8;
235:   end;
236:   MsDos(recpack);                        { call function }
237:   with recpack do
238:   begin
239:     str(cx,year);                        {convert to string}
240:     str(dx mod 256,day);                    { " }
241:     str(dx shr 8,month);                    { " }
242:   end;
243:   if length(day)   = 1 then insert('0',day,1);
244:   if length(month) = 1 then insert('0',month,1);
245:   year := copy(year,3,2);
246:   date := day+'-'+month+'-'+year;
247: end;
248:
249:
250: function time: str10;
251: type
252:   regpack = record
253:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
254:             end;
255:
256: var
257:   recpack:          regpack;             {assign record}
258:   ah,al,ch,cl,dh:   byte;
259:   hour,min,sec:     string[2];
260:
261: begin
262:   ah := $2c;                             {initialize correct registers}
263:   with recpack do
264:   begin
265:     ax := ah shl 8 + al;
266:   end;
267:   intr($21,recpack);                     {call interrupt}
```

```
268:   with recpack do
269:   begin
270:     str(cx shr 8,hour);                  {convert to string}
271:     str(cx mod 256,min);                       { " }
272:     str(dx shr 8,sec);                         { " }
273:   end;
274:   if length(min)=1 then insert('0',min,1);
275:   if length(sec)=1 then insert('0',sec,1);
276:   time := hour+':'+min+':'+sec;
277: end;
278:
279:
280: function disk: integer;
281: type
282:   regpack = record
283:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
284:             end;
285: var
286:   recpack:       regpack;                {record for MsDos call}
287:
288: begin
289: with recpack do  ax := $19 shl 8;
290: MSDOS(recpack);
291: with recpack do disk := (ax mod 256) + 1;
292: end;
293:
294:
295: function day:str10;
296: type
297:   regpack = record
298:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
299:             end;
300:
301: var
302:   recpack:       regpack;                {record for MsDos call}
303:   dag    :       string[1];
304:
305: begin
306: with recpack do
307:    begin
308:    ax := $2A shl 8;
309:    end;
310: MSDOS(recpack);
311: with recpack do str(ax mod 256, dag);
312: case dag of
313:    '0':day:='zondag';
314:    '1':day:='maandag';
315:    '2':day:='dinsdag';
316:    '3':day:='woensdag';
317:    '4':day:='donderdag';
318:    '5':day:='vrijdag';
319:    '6':day:='zaterdag';
320:    end;
321: end;
322:
323: procedure setdate;
324: type
325:   regpack = record
326:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
327:             end;
328:
329: var
330:   recpack    :       regpack;               {record for MsDos call}
331:   x,y        :       integer;
332:   ch         :       char;
333:   dummy      :       real;
```

```
335: x := wherex;
336: y := wherey;
337: with recpack do
338: repeat
339:    gotoxy(x,y);  dummy := 0;
340:    write('geef dag  :');
341:    convert(dummy,2,x+11,y-1,0,[^M],ch);
342:    dx := round(dummy);
343:    gotoxy(x,y+1);dummy := 0;
344:    write('geef maand:');
345:    convert(dummy,2,x+11,y,0,[^M],ch);
346:    dx := dx + (round(dummy) shl 8);
347:    gotoxy(x,y+2);dummy := 0;
348:    write('geef jaar :');
349:    convert(dummy,4,x+11,y+1,0,[^M],ch);
350:    cx := round(dummy);
351:    ax := $2B shl 8;
352:    MSDOS(recpack);
353: until (ax mod 256 = 0);
354: end;
355:
356: procedure settime;
357: type
358:   regpack = record
359:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
360:             end;
361:
362: var
363:   recpack   :       regpack;               {record for MsDos call}
364:   x,y       :       integer;
365:   ch        :       char;
366:   dummy     :       real;
367: begin
368: x := wherex;
369: y := wherey;
370: with recpack do
371: repeat
372:    gotoxy(x,y);    dummy := 0;
373:    write('geef uur  :');
374:    convert(dummy,2,x+11,y-1,0,[^M],ch);
375:    cx := round(dummy) shl 8;
376:    gotoxy(x,y+1);  dummy := 0;
377:    write('geef min. :');
378:    convert(dummy,2,x+11,y,0,[^M],ch);
379:    dx := 0;
380:    cx := cx + round(dummy);
381:    ax := $2D shl 8;
382:    MSDOS(recpack);
383: until (ax mod 256) = 0;
384: end;
385:
386:
387: function DOSversion : str10;
388: type
389:   regpack = record
390:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
391:             end;
392: var
393:   recpack:       regpack;              {record for MsDos call}
394:   i      :       integer;
395:   up,down:       string[2];
396: begin
397: with recpack do ax := $30 shl 8;
398: MSDOS(recpack);
399: with recpack do
400:    begin
401:    i := ax mod 256;
```

```
402:    str(i,up);
403:    i := ax shr 8;
404:    str(i,down);
405:    end;
406: DOSversion := up+'.'+down;
407: end;
408:
409: procedure diskfree(var mem:real ; var drive:integer);
410: type
411:   regpack = record
412:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
413:             end;
414: var
415:   recpack:       regpack;               {record for MsDos call}
416: begin
417: with recpack do
418:    begin
419:    ax := $36 shl 8;
420:    dx := drive;             {* 0 = default drive *}
421:    end;
422: MSDOS(recpack);
423: with recpack do                      {* bx bevat available clusters *}
424:    begin                             {* dx bevat clusters/drive     *}
425:    mem := bx;                        {* cx bevat bytes/sectors      *}
426:    if (ax mod 256) = 255 then drive := -1;
427:    end;
428: end;
429:
430: procedure selectdisk( var drive:integer);
431: type
432:   regpack = record
433:               ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
434:             end;
435: var
436:   recpack :       regpack;               {record for MsDos call}
437:   get     :       real;
438:   code    :       integer;
439:
440: begin
441: if drive (= 0 then
442:    begin
443:    drive := disk;
444:    end;
445: drive := drive - 1;
446: with recpack do
447:    begin
448:    ax := $E shl 8;
449:    dx := drive;
450:    end;
451: MSDOS(recpack);
452: with recpack do drive := ax mod 256;
453: end;
454:
455:
456: procedure show(var rr,ri : werkarray);
457: var n : integer;
458:
459: begin
460: write(lst,#27,'4');
461: for n := 1 to samples do
462:   begin
463:   write(lst,n:13,' ',rr[n]:12:9,' ',ri[n]:12:9);
464:   if n mod 2 = 0 then writeln(lst);
465:   end;
466: end;
467:
468:
```

```
469: {* Deze procedure vult een array met sinus waarden           *}
470:
471: procedure setupsin;
472: var n:integer;
473: begin
474: for n := 1 to samples do
475:    sinarray[n] := sin(2*pi*(n-1)/samples);
476: end;
477:
478: {* SIn en COs functions voor arraywaardes van sinus en cosisinus    *}
479:
480: function si(var n:integer) : real;
481: begin
482: if n ) 0 then
483:    si := sinarray[abs(n)]
484: else
485:    si := -1*sinarray[abs(n)]
486: end;
487:
488: function co(var n:integer) : real;
489: begin
490: co := sinarray[ (abs(n) + samples shr 2)]
491: end;
492:
493:
494:
495: {************************************************************************}
496: {********** Fouriertransformatie door Jan Vieveen *********************}
497: {************************************************************************}
498:
499: procedure ft(var rr,ri : werkarray);
500: var part,step,stap,n,p,lengte,halvelengte,inf1,inf2   :integer;
501: intf                                                  :real;
502: hulprr,hulpri                                         :werkarray;
503:
504: begin
505: lengte      := samples;
506: halvelengte := samples shr 1;
507: for n := 1 to round(   ln(samples)/ln(2)   ) do
508:    begin
509:    hulprr := rr;
510:    hulpri := ri;
511:    fillchar(rr,sizeof(werkarray),0);
512:    fillchar(ri,sizeof(werkarray),0);
513:    for p := 1 to halvelengte do
514:       begin
515:       part  := round(  int( p*(1 shl n)/lengte - 0.000001)  );
516:       intf  := part/( 1 shl n );
517:       inf1  := round(intf * samples) + 1;
518:       step  := (lengte * part) shr n  ;
519:       stap  := step +  (lengte shr n) ;
520:       rr[p] := hulprr[p + step] + hulprr[p + stap] * co(inf1)
521:                                 - hulpri[p + stap] * si(inf1);
522:       ri[p] := hulpri[p + step] + hulprr[p + stap] * si(inf1)
523:                                 + hulpri[p + stap] * co(inf1);
524:       rr[p + halvelengte] := hulprr[p + step] - hulprr[p + stap]*co(inf1)
525:                                               + hulpri[p + stap]*si(inf1);
526:       ri[p + halvelengte] := hulpri[p + step] - hulprr[p + stap]*si(inf1)
527:                                               - hulpri[p + stap]*co(inf1);
528:       end;
529:    end;
530: for n := 2 to samples do
531:    begin
532:    rr[n] := rr[n] * 2;
533:    ri[n] := ri[n] * 2;
534:    end;
```

```
535: end;
536:
537: procedure ift(var rr,ri : werkarray);
538: var part,step,stap,n,p,lengte,halvelengte,inf1,inf2   :integer;
539: intf                                                  :real;
540: hulprr,hulpri                                         :werkarray;
541:
542: begin
543: lengte      := samples;
544: halvelengte := samples shr 1;
545: n := round(   ln(samples)/ln(2)   );
546: repeat
547:    begin
548:    hulprr := rr;
549:    hulpri := ri;
550:    fillchar(rr,sizeof(werkarray),0);
551:    fillchar(ri,sizeof(werkarray),0);
552:    for p := 1 to halvelengte do
553:       begin
554:       part  := round(  int( p*(1 shl n)/lengte - 0.000001)  );
555:       intf  := part/( 1 shl n );
556:       inf1  := -1 * (round(intf * samples) + 1);
557:       step  := (lengte * part) shr n  ;
558:       stap  := step +  (lengte shr n) ;
559:       rr[p + stap] := 0.5 * (co(inf1) * (hulprr[p] - hulprr[p + halvelengte])
560:                           - si(inf1) * (hulpri[p] - hulpri[p + halvelengte]));
561:       ri[p + stap] := 0.5 * (co(inf1) * (hulpri[p] - hulpri[p + halvelengte])
562:                           + si(inf1) * (hulprr[p] - hulprr[p + halvelengte]));
563:       rr[p + step] := 0.5 *             (hulprr[p] + hulprr[p + halvelengte]);
564:       ri[p + step] := 0.5 *             (hulpri[p] + hulpri[p + halvelengte]);
565:       end;
566:    end;
567: n := n - 1;
568: until n = 0;
569: end;
570:
571: procedure taper(var ar : werkarray; var deel : real);
572: var n : integer;
573: begin
574: for n := 1 to round(deel*samples/2) do
575:    ar[n] := ar[n] * sqr(sin( (n-1)*pi / (deel*samples - 2) ) );
576: if deel <> 0 then
577: for n := round(samples * (1 - deel/2)) to samples do
578:    ar[n] := ar[n] * sqr(sin( (samples-n)*pi / (samples*deel) ) );
579: end;
580:
581:
582: {* Dit is ook een FastFouriertransformatie, maar zonder gebruik van array's *}
583: {* voor de sinus en cosinus (dit ter illustratie van de methode).          *}
584:
585: procedure fast(var rr,ri : werkarray);
586: var part,step,stap,n,p,lengte,halvelengte  :integer;
587: intf                                       :real;
588: hulprr,hulpri                              :werkarray;
589:
590: begin
591: lengte      := samples;
592: halvelengte := samples shr 1;
593: for n := 1 to round(   ln(samples)/ln(2)   ) do
594:    begin
595:    hulprr := rr;
596:    hulpri := ri;
597:    fillchar(rr,sizeof(werkarray),0);
598:    fillchar(ri,sizeof(werkarray),0);
599:    for p := 1 to halvelengte do
600:       begin
601:       part  := round(  int( p*(1 shl n)/lengte - 0.000001)  );
```

```
602:        intf  := part/( 1 shl n );
603:        step  := (lengte * part) shr n  ;
604:        stap  := step +  (lengte shr n) ;
605:        rr[p] := hulprr[p + step] + hulprr[p + stap] * cos(intf * 2 * pi)
606:                                  - hulpri[p + stap] * sin(intf * 2 * pi);
607:        ri[p] := hulpri[p + step] + hulprr[p + stap] * sin(intf * 2 * pi)
608:                                  + hulpri[p + stap] * cos(intf * 2 * pi);
609:        rr[p + halvelengte] := hulprr[p + step] + hulprr[p + stap]*cos(2*pi*(intf+0.5))
610:                                                - hulpri[p + stap]*sin(2*pi*(intf+0.5));
611:        ri[p + halvelengte] := hulpri[p + step] + hulprr[p + stap]*sin(2*pi*(intf+0.5))
612:                                                + hulpri[p + stap]*cos(2*pi*(intf+0.5));
613:        end;
614:    end;
615: for n := 1 to samples do
616:    begin
617:    ri[n] := ri[n]/samples;
618:    rr[n] := rr[n]/samples;
619:    end;
620: end;
621:
622: {* "vulin" codeert binair de tekening                                    *)
623: {* Deze procedure wordt gebruikt door drawp, identiek aan draw in pascal, *)
624: {* alleen drawp vult een array "tekening". Vulin checkt of het getekende  *)
625: {* punt al getekend is, zoniet alsnog.                                    *)
626:
627: procedure vulin(var x,y: integer);
628: var rij,hoogte,bulky,i : integer;
629: begin
630: rij := 1 + round(  (y - y mod 8) / 8 );
631: hoogte := 7 - y mod 8;
632: bulky := tekening[x,rij];
633: bulky := bulky shr hoogte;
634: if bulky mod 2 = 0 then
635:    tekening[x,rij] := tekening[x,rij] + 1 shl hoogte;
636: end;
637:
638: procedure drawp(x1,y1,x2,y2 : integer);
639: var x,y                        : integer;
640:
641: begin
642: x1 := x1 + 1;                        {* schaling (array loopt v 1..640*)
643: x2 := x2 + 1;                        {* y wordt automatisch in "vulin" geschaald *)
644: x  := x1;                            {* x en y zijn variabelen *)
645: y  := y1;
646: if x1 ) 640 then x1 := 640;
647: if x2 ) 640 then x2 := 640;
648: if y ) 200 then y := 200;
649: vulin(x,y);
650: if (x () x2) or (y () y2) then
651: repeat
652:    if abs((x - x2)/(abs(x1 - x2)+1)) ) abs((y - y2)/(abs(y1 - y2)+1)) then
653:        begin
654:        if x ( x2 then
655:            x := x + 1
656:        else
657:            x := x - 1;
658:        end
659:    else
660:        begin
661:        if y ( y2 then
662:            y := y + 1
663:        else
664:            y := y - 1;
665:        end;
666:    vulin(x,y);
667: until (x = x2) and (y = y2);
668: end;
```

```
669:
670: {* Hiresp veegt de tekening voor op de printer                    *}
671:
672: procedure hiresp;
673: var n,m : integer;
674: begin
675: for n := 1 to 640 do
676:    for m := 1 to 25 do
677:       tekening[n,m] := 0;
678: end;
679:
680: {* drukaf drukyt tekening op IBM printer, M = max. amplitude        *}
681: {* De printer wordt "Intelligent" bestuurd                          *}
682:
683: procedure drukaf(var m:real);
684: var n,p,i,j,code    : integer;
685:    step,hulp        : real;
686:    klaar            : boolean;
687:    mededeling       : anystr;
688:    tc               : char;
689:
690: begin
691: mededeling := '';
692: step := 0;
693: writeln(lst,#27,#51,#21,   #27,#52);   {* set linefeed to 3*naalden, near letter quality *}
694: write(lst,' samples:',samples:5,' uitgang:',uitgang:5:2,' samplefreq:',fs:7:2,' freq. filter :',afsnijfreq:7:2);
695: writeln(lst,' A =',A:3:2);
696: writeln(lst);
697: writeln(lst,' schaal :',schaal:5:0,' ingang :',u:5:2,'  sign.freq:',sf:7:2,' freq. filter2:',afsnijfreq2:7:2);
698: writeln(lst);
699: for n := 1 to 25 do
700:    begin
701:    if (n=2)or(n=4)or(n=6)or(n=8)or(n=11)or(n=13)or(n=15)or(n=17)or(n=19) then
702:       begin
703:       write(lst,(m - m*step/4):8:2);
704:       step := step + 1;
705:       end
706:    else
707:       write(lst,'':8);
708:    code := 0;
709:    j := 640;
710:    repeat
711:       if tekening[j,n] <> 0 then
712:          begin
713:          code := j;
714:          end;
715:       j := j - 1;
716:    until (code <> 0) or (j = 0);
717:    write(lst,#27,#76,chr(code mod 256),chr(round((code - code mod 256)/256)) );
718:   for p := 1 to code do
719:       begin
720:       write(lst,chr(tekening[p,n]));
721:       end;
722:    if code <> 0 then writeln(lst);
723:    gotoxy(1,24);
724:    writeln(n:10,code:10);
725:    end;
726: gotoxy(1,23);
727: write('Geef een mededeling voor onder de grafiek:');clreol;
728: gotoxy(1,24);clreol;
729: writeln(lst);
730: inputstr(mededeling,80,0,23,[^M],tc);
731: writeln(lst,mededeling);
732: for p := 1 to 2 do
733:    writeln(lst);
734: end;
735:
```

```
736:
737: (* Deze procedures vullen een werkarray met een functie             *)
738:
739: procedure sinus(var a:werkarray );
740: var n : integer;
741: begin
742: for n := 1 to samples do
743:    a[n] := u * sin((sf/fs) * 2 * pi * (n-1));
744: end;
745:
746:
747: procedure zaagtand(var a : werkarray);
748: var n : integer;
749:     m : real;
750: begin
751: m := 2*u*sf/fs;
752: a[1] := -u;
753: for n := 2 to samples do
754:    begin
755:    a[n] := a[n-1] + m;
756:    if a[n] > u then
757:        a[n] := -u;
758:    end;
759: end;
760:
761:
762: procedure driehoek(var a : werkarray);
763: var n : integer;
764:     m : real;
765: begin
766: m := 4*u*sf/fs;
767: a[1] := -u;
768: for n := 2 to samples do
769:    begin
770:    a[n] := a[n-1] + m;
771:    if abs(a[n]+m) > u then
772:        m := -m;
773:    end;
774: end;
775:
776:
777: procedure blok(var a : werkarray);
778: var n : integer;
779: m,ing : real;
780: begin
781: ing := u;
782: m := 0.5 * fs/sf;
783: for n := 1 to samples do
784:    begin
785:    a[n] := ing;
786:    if m < n then
787:        begin
788:        m := m + 0.5 * fs/sf;
789:        ing := -ing;
790:        end;
791:    end;
792: end;
793:
794:
795: procedure delta(var a : werkarray);
796: var n : integer;
797:     m : real;
798: begin
799: m :=  fs/sf;
800: for n := 1 to samples do
801:    begin
802:    if m < n then
```

```
803:        begin
804:        a[n] := u;
805:        m := m + fs/sf;
806:        end
807:    else
808:        a[n] := 0;
809:    end;
810: end;
811:
812:
813: {*deze procedure zet alle globale variabelen op nul *}
814:
815: procedure initialisatie;
816: var i : integer;
817:
818: begin
819: setupsin;
820: for i := 1 to samples do
821:    begin
822:    a0[i] := 0;
823:    a1[i] := 0;
824:    a2[i] := 0;
825:    a3[i] := 0;
826:    a4[i] := 0;
827:    a5[i] := 0;
828:    a6[i] := 0;
829:    b0[i] := 0;
830:    end;
831: fs := 70.0;
832: sf := 18*fs/(samples-1);
833: v1 := 0;
834: v2 := 0;
835: schaal := 2;
836: uitgang := 1;
837: afsnijfreq := 5.5;
838: afsnijfreq2 := 5.5;
839: u  := (1 - (4*afsnijfreq/fs) );
840: recht := false;
841: printing := false;
842: A := 0.5;
843: tape := 0.1;
844: constant := false;
845: end;
846:
847: {* start is het begin menu met instelling van de variabelen   *}
848:
849: procedure start;
850: var i : integer;
851:    ch : char;
852:
853: const  term : charset = [^E,^I,^H,^X,^Z];
854:
855: begin
856: clrscr;
857: gotoxy(1,1);
858: writeln('Datum is ',day,' ',date);
859: writeln('DOS-version is ',dosversion);
860: gotoxy(1,4);
861: beep;
862: writeln('sample frequentie ..........');
863: writeln('frequentie ingangsignaal ...              = aantal maal ',fs/(samples-1):6:5);
864: writeln('amplitude ingangsignaal ....');
865: writeln('amplitude comperator 1 .....');
866: writeln('amplitude comperator 2 .....');
867: writeln('schaalfactor (1..max) ......');
868: writeln('uitgang ....................');
```

```
869: writeln('Afsnijfreqentie filter .....');
870: writeln('Afsnijfreqentie filter2 ....');
871: writeln('Tussen verzwakking .........');
872: writeln('Taperfactor ................');
873: i := 1;
874: sf := sf*(samples-1)/fs;
875: repeat
876:    case i of
877:    1:convert(fs,12,29,3,2,term,ch);
878:    2:convert(sf,12,29,4,2,term,ch);
879:    3:convert(u ,10,29,5,2,term,ch);
880:    4:convert(v1,10,29,6,2,term,ch);
881:    5:convert(v2,10,29,7,2,term,ch);
882:    6:repeat
883:        convert(schaal,4,29,8,0,term,ch);
884:      until (schaal )= 2) and (schaal ( samples);
885:    7:convert(uitgang,10,29,9,2,term,ch);
886:    8:convert(afsnijfreq,10,29,10,2,term,ch);
887:    9:convert(afsnijfreq2,10,29,11,2,term,ch);
888:    10:convert(A,4,29,12,2,term,ch);
889:    11:convert(tape,5,29,13,3,term,ch);
890:    end;
891:    if (ch=^I) or (ch=^M) or (ch=^X) then
892:        if i = 11 then
893:            i := 1
894:        else
895:            i := i+1
896:    else
897:        if ch = ^E then
898:            if i = 1 then
899:                i := 11
900:            else i := i-1;
901: until (ch=^M) and (i=1) or (ch=^Z);
902: sf := sf * fs/(samples-1);
903: select('Genestte structuur  J)a / N)ee ',['J','N'],ch);
904: genest := (ch = 'J');
905: select('DC correctie toepassen J)a / N)ee ',['J','N'],ch);
906: dc := (ch = 'J');
907: select('Hoofd DSM ½ T vertragen (slap filter) ',['J','N'],ch);
908: halveT := (ch='J');
909: select('kies een ingangsignaal S)inus, Z)aagtand, D)riehoek, dE)lta ,B)lok ',['S','Z','D','E','B'],ch);
910: case ch of
911: 'S':sinus(a0);
912: 'Z':zaagtand(a0);
913: 'D':driehoek(a0);
914: 'E':delta(a0);
915: 'B':blok(a0);
916: end;
917: taper(a0,tape);
918: clrscr;
919: end;
920:
921: {* assen tekenen tekent assen op scherm in 640 x 200 mode  (helaas 1 kleur)  *}
922:
923: procedure assentekenen(var m : real ; a:werkarray);
924: var n : integer;
925:
926: begin
927: draw(40,0,40,160,askleur);
928: draw(35,80,639,80,askleur);
929: if printing then
930:    begin
931:    drawp(40,0,40,160);
932:    drawp(35,80,639,80);
933:    end;
934: for n := 1 to 10 do
935:    begin
```

```
936:    draw(39+n*60,76,39+n*60,84,askleur);
937:    if printing then drawp(39+n*60,76,39+n*60,84);
938:    end;
939: if not constant then
940:    m := 0.5;
941: for n := 1 to samples do
942:    if abs(a[n]) > m then
943:        m := abs(a[n]);
944: m := round(m + 0.49);
945: for n := 1 to 9 do
946:    begin
947:    draw(36,n * 18 - 9,44, n * 18 - 9,askleur);
948:    if printing then drawp(36,n * 18 - 9,44, n * 18 - 9);
949:    if n < 6 then
950:        gotoxy(1,n * 2)
951:    else
952:        gotoxy(1,n * 2 + 1);
953:    write((m * (5 - n)/4):2:1);
954:    end;
955: gotoxy(75,9);
956: write(samples);
957: end;
958:
959:
960: {* tekengrafiek tekent grafiek in a, met kleur gegeven en m maxamplitude *}
961:
962: procedure tekengrafiek( m : real ; a : werkarray ; kleur : integer);
963: var n,x1,x2,y1,y2 : integer;
964:     dummy1,dummy2 : real;
965:
966: begin
967: dummy1 := 596/(samples-schaal);
968: dummy2 := 80/m;
969: for n := round(schaal) to samples do
970:    begin
971:    x1 := 40 +      round( (n-schaal    ) *  dummy1);
972:    x2 := 40 +      round( (n-schaal + 1) *  dummy1);
973:    y1 := 80 + -1 * round(a[(n-1)] * dummy2);
974:    y2 := 80 + -1 * round(a[(n  )] * dummy2);
975:    if recht then
976:        begin
977:        draw(x1,y1,x1,y2,kleur);
978:        draw(x1,y2,x2,y2,kleur);
979:        if printing then
980:            begin
981:            drawp(x1,y1,x1,y2);
982:            drawp(x1,y2,x2,y2);
983:            end;
984:        end
985:    else
986:        begin
987:        draw(x1,y1,x2,y2,kleur);
988:        if printing then  drawp(x1,y1,x2,y2);
989:        end;
990:    end;
991: end;
992:
993: {* begin van hoofdprogramma             *}
994:
995: begin
996: clrscr;
997: gotoxy(10,10);
998: write('Programma initialiseert variabelen........');
999: initialisatie;
1000: repeat
1001:    schaal := 2;
1002:    printing := false;
```

```
1003:    start;
1004:    fillchar(b0,sizeof(werkarray),0);
1005:    ft(a0,b0);                                    {* filteren van ingang *)
1006:    for n := round(afsnijfreq*samples/fs + 1) to samples do
1007:        begin
1008:        a0[n] := 0;
1009:        b0[n] := 0;
1010:        end;
1011:    ift(a0,b0);
1012:
1013:    a1[1] := a0[1];                               {*  start hoofd-DSM    *)
1014:    integrator := a1[1];
1015:    a2[1] := integrator;
1016:    m1 := integrator;
1017:    a3[1] := 0;
1018:    for n := 2 to samples do
1019:        begin
1020:        a1[n] := a0[n] - a3[n-1];
1021:        integrator := integrator + a1[n];
1022:        a2[n] := integrator;
1023:        if a2[n] ) v1 then
1024:            a3[n] := uitgang
1025:        else
1026:            a3[n] := -uitgang;
1027:        end;
1028:    b0 := a3;
1029:    if halveT then
1030:        begin
1031:        for n := 2 to samples do
1032:            a3[n] := ( b0[n] + b0[n-1] )/2;
1033:       end;                                  {* einde procedure 1e DSM modulator *)
1034:    fillchar(b0,sizeof(werkarray),0);
1035:    a7 := a3;
1036:
1037:    if genest then
1038:        begin
1039:        a4 := a2;
1040:        if dc then for n := 1 to samples do
1041:            begin
1042:            a4[n] := a4[n] - a0[n];
1043:            end;
1044:
1045:        fillchar(b0,sizeof(werkarray),0);
1046:        ft(a4,b0);
1047:        for n := round(afsnijfreq2*(samples)/fs + 1) to samples  do
1048:            begin
1049:            a4[n] := 0;
1050:            b0[n] := 0;
1051:            end;
1052:        ift(a4,b0);
1053:        for n := 1 to samples do
1054:            a4[n] := a4[n]/A;
1055:        a5[1] := a4[1];
1056:        integrator := a5[1];
1057:        a6[1] := integrator;
1058:        a2[1] := 0;
1059:        for n := 2 to samples do
1060:            begin
1061:            a5[n] := a4[n] - a2[n-1];
1062:            integrator := integrator + a5[n];
1063:            a6[n] := integrator;
1064:            if a6[n] ) v2 then
1065:                a2[n] := 1
1066:            else
1067:                a2[n] := -1;
1068:            end;
1069:        b0 := a2;
```

```
1070:       for n := 1 to samples do
1071:           begin
1072:           a2[n] := A*a2[n] ;
1073:           end;
1074:
1075:       b0 := a2;a7[1] := 0; a7[samples] :=0;  {*b0 is dummy voor differentiator *)
1076:                a1[1] := 0; a1[samples] :=0;
1077:       for n := 2 to (samples) do
1078:           a1[n] := (b0[n] - b0[n-1]);  {* differentieëren *}
1079:
1080:       for n := 2 to samples do
1081:           if dc then a7[n-1] := a1[n] + a3[n-1]
1082:           else a7[n] := a1[n] + a3[n-1];
1083:       a6 := a7;
1084:       end;     {* of if genest..... *}
1085:
1086:   fillchar(b0,sizeof(werkarray),0);
1087:   ft(a7,b0);
1088:   for n := round(afsnijfreq*samples/fs + 1) to samples do
1089:       begin
1090:       a7[n] := 0;
1091:       b0[n] := 0;
1092:       end;
1093:   ift(a7,b0);
1094: b0 := a7;
1095:   hires;
1096:   assentekenen(m,a0);
1097:   tekengrafiek(m,a0,grafiek2kleur);
1098:   repeat
1099:       gotoxy(1,1);
1100:       write('Schaal:',schaal:4:0);
1101:       gotoxy(12,1);
1102:       if recht then
1103:           write(' /   rechte  grafiek  /')
1104:       else
1105:          write(' / vloeiende grafiek  /');
1106:       if printing then
1107:           write(' printer AAN / ')
1108:       else
1109:           write(' printer UIT / ');
1110:       write(fs:6:2);
1111:       select('teken grafiek 0..7, V)eeguit, Q)uit, S)chaal G)roter/K)leiner, R)echt/scheef ',
1112:       ['0','1','2','3','4','5','6','7','8','V','Q','G','K','R','S','W','P','D','C'],ch);
1113:       case ch of
1114:       '0':begin
1115:           assentekenen(m,a0);
1116:           tekengrafiek(m,a0,grafiek1kleur);
1117:           end;
1118:       '1':begin
1119:           assentekenen(m,a1);
1120:           tekengrafiek(m,a1,grafiek2kleur);
1121:           end;
1122:       '2':begin
1123:           assentekenen(m,a2);
1124:           tekengrafiek(m,a2,grafiek2kleur);
1125:           end;
1126:       '3':begin
1127:           assentekenen(m,a3);
1128:           tekengrafiek(m,a3,grafiek2kleur);
1129:           end;
1130:       '4':begin
1131:           assentekenen(m,a4);
1132:           tekengrafiek(m,a4,grafiek2kleur);
1133:           end;
1134:       '5':begin
1135:           assentekenen(m,a5);
1136:           tekengrafiek(m,a5,grafiek2kleur);
```

```
1137:            end;
1138:        '6':begin
1139:            assentekenen(m,a6);
1140:            tekengrafiek(m,a6,grafiek2kleur);
1141:            end;
1142:        '7':begin
1143:            assentekenen(m,a7);
1144:            tekengrafiek(m,a7,grafiek2kleur);
1145:            end;
1146:        'B':begin
1147:            assentekenen(m,b0);
1148:            tekengrafiek(m,b0,grafiek2kleur);
1149:            end;
1150:        'V':begin
1151:            hires;
1152:            if printing then hiresp;
1153:            end;
1154:        'W':begin
1155:            m1 := 0;
1156:            m3 := 0;
1157:            m4 := 0;
1158:            for n := round(tape * samples) to (samples-round(tape * samples)) do
1159:                begin
1160:                m4 := m4 + sqr(a0[n]);
1161:                m1 := m1 + sqr(b0[n] - a0[n]);
1162:                end;
1163:            m := 10 * ln(m4/m1)/ln(10);
1164:            gotoxy(1,24);write('S/R verhouding is ',m:10:2,' dB');
1165:            if printing then
1166:                begin
1167:                writeln(lst,'S/R verhouding is ',m:4:3,' dB.');
1168:                writeln(lst);
1169:                if dc then write(lst,'D.C. gecorrigeerd, ');
1170:                if halveT then write(lst,' hoofd DSM ½T vertraagd, ');
1171:                if genest then write(lst,' gecorrigeerde DSM');
1172:                writeln(lst);
1173:                end;
1174:            end;
1175:        'G':if schaal + 1 ( samples then
1176:                schaal := schaal + 1;
1177:        'K':if schaal - 1 )= 2 then
1178:                schaal := schaal - 1;
1179:        'S':begin
1180:            schaal := schaal + 50;
1181:            if schaal ) samples then
1182:                schaal := schaal - samples + 1;
1183:            end;
1184:        'R':recht := not recht;
1185:        'P':printing := not printing;
1186:        'D':drukaf(m);
1187:        'C':begin
1188:            constant := not constant;
1189:            if constant then
1190:                begin
1191:                gotoxy(1,24);
1192:                write('Geef grafiekwaarde:                                                    ');
1193:                convert(m,5,20,23,1,[^M],ch);
1194:                end;
1195:            end;
1196:        end;
1197:    until ch = 'Q';
1198:    select('Nog een proef doen   J)a / N)ee ',[^M,'J','N'],ch);
1199: until ch = 'N';
1200: clrscr;
1201: end.
1202:
```

```
1203:
1204:
1205: {*  Dit is een tabel om naast de computer te leggen voor functies van toetsen *}
1206: {*                                                                            *}
1207: {*  toets       normale DSM                 Gecorrigeerde DSM                *}
1208: {*  ------------------------------------------------------------------------- *}
1209: {*  0        ingangsignaal                  ingangsignaal                    *}
1210: {*  1        foutsignaal                    gediff.signaal correctie DSM =   *}
1211: {*                                            correctie signaal ongefilterd  *}
1212: {*  2        geïntegreerd foutsignaal       uitgang corr.DSM ampl. gecorr.   *}
1213: {*  3        uitgang DSM                    uitgang hoofd-DSM                *}
1214: {*  4        XXXXXXXXXXX                    gefilterd en ampl.gecorr.        *}
1215: {*                                            geïntegreerd foutsignaal       *}
1216: {*  5        XXXXXXXXXXX                    foutsignaal corr.DSM             *}
1217: {*  6        XXXXXXXXXXX                    geïnt.foutsign. corr.DSM         *}
1218: {*  7        gefilterde DSM                 gefilterd hoofd+corr.DSM         *}
1219: {*  8        = toets 7                      = toets 7                        *}
1220: {*                                                                            *}
1221: {*  C = aanhouden van hoogst voorkomende amplitude  aan/uit                   *}
1222: {*  R = staafdiagram / vloeiende grafiek                                      *}
1223: {*  G = schaal beginpunt grafiek groter                                       *}
1224: {*  K = schaal beginpunt grafiek kleiner                                      *}
1225: {*  S = schaal grafiek 50 omhoog                                              *}
1226: {*  V = veeg grafiek                                                          *}
1227: {*  W = bereken Signaal/Ruis verhouding                                       *}
1228: {*  Q = Stoppen                                                               *}
1229: {*                                                                            *}
1230: {*  P = printer aan/uit                                                       *}
1231: {*  D = drukaf grafiek op printer                                             *}
1232: {*  ------------------------------------------------------------------------- *}
```