A digital servo controller for a laservision video-disc player

van der Stap, R.A.M.

Award date:
1984

UNIVERSITY OF TECHNOLOGY EINDHOVEN
Department of Electrotechnology
Group Digital Systems

TECHNISCHE HOGESCHOOL EINDHOVEN
Afdeling der Elektrotechniek
Vakgroep Digitale Systemen

A DIGITAL SERVO CONTROLLER FOR A LASERVISION VIDEO-DISC PLAYER

By: R.A.M. van der Stap

Report of a graduation project,
from 830501 to 840430,
under supervision of:
Prof.Ir. A. Heetman,
Ir. J. van Lier,
Ing. L.A. van Bokhoven.

SUMMARY

The task of the LaserVision laboratory of Philips, is to develop video-disc players that have a better price/performance ratio. One way to do this is to integrate the electronics. Under certain circumstances, digital servo techniques are the most promising, for integrating the electronics of the players' servo systems. This report describes the design and construction of a digital servo controller based on a digital signal processor. From the evaluated processors, the TMS320 of Texas Instruments was the most suitable. A program was written for a P2500 personal computer, which can emulate a terminal for the evaluation module of the TMS320. This evaluation module is used to develop the digital servo controller. The controller is intended as a tool, useful for the investigation of the digital servo area. Two types of pulse modulators have been designed and tested, to replace the digital-to-analog converter. Two LaserVision servos are implemented by the controller, operating with an algorithm optimized for the elimination of periodic disturbances. The two servo-tasks are handled synchronously and thus both servos have the same sampling frequency. In this set-up, addition of a third servo is not possible, due to the limited speed of the processor. Aliasing precautions can be left out and this saves the phase margin of the system. Implementation of digital servo systems within the LaserVision system is feasible.

TABLE OF CONTENTS

TABLE OF CONTENTS

1. INTRODUCTION

In this report I will describe the work I did on my graduation project. The work was carried out at the LaserVision laboratory of the Nederlandse Philips Bedrijven B.V. under supervision of Ir. J. van Lier (Philips) and Ing. L.A. van Bokhoven (University of Technology). Responsible for the graduation was Prof. Ir. A. Heetman, the professor of the group Digital Systems.

At the LaserVision laboratory, LaserVision video-disc players are developed. The main principles of the player have been developed. Now, the main task of the laboratory is to develop new features and most important: to develop players with a better price/performance ratio. This can be done in several ways. One method is to integrate the electronics into integrated circuits.

A LaserVision video-disc player comprises several servo systems. To integrate the electronics part of these systems, several methods are available:

- an analog integration technique can be used.
- a switched-capacitor technique can be used, which makes it a sampled system.
- digital signal processing techniques can be applied, which not only makes it a sampled system, but also causes the signal to be quantized.

The objective is to integrate all servo systems of the player in one component, if possible including matching sequence and control logic. This to reduce the number of components as much as possible. Only digital signal processing techniques make the system (aside from sample-and-hold and analog-to-digital conversion) temperature independent and facilitates integrating sequence and control logic in the system. Also this technique is best suited to implement adaptive systems in.

My assignment was to design and construct a digital servo controller, based on a microprocessor, that could perform the function of at least two servo systems. In addition to this, the controller would have to operate without DAC (Digital-to-Analog Converter) and a special, predictive, control algorithm would have to be used. The controller should be implemented in an existing LaserVision video-disc player and with it, it should be possible to develop certain control algorithms and compensation filters as well as deliver the proof that digital servo systems can be feasible in a LaserVision video-disc player.

The LaserVision system will be described in chapter 2. Most of the material presented in chapter 2 is taken from Philips pu-

blications and is only re-edited by me. Chapter 2 can be skipped by readers who are already familiar with the principles of the LaserVision system. Chapter 3 presents an introduction to servo systems in general and digital servo systems in particular. This introduction is included because it represents the orientating work I had to do, to be able to handle the assignment. It can be skipped by readers who are familiar with servo technique. In chapter 4 the assignment is described in more detail with all implicating requirements. The choice of the components used for the controller is treated in chapter 5 and the tools, that formed the environment in which the controller was developed, are described in chapter 6. In chapter 7 then, all parts of the controller hardware are described, and chapter 8 gives an inside view of the controller software. The results, conclusions and recommendations are presented in chapter 9. Listings of the software and schematics of the hardware can be found in the appendices, as well as a short instruction guide to the usage of the prototype of the controller.

I would like to express my gratitude towards all people who assisted me in my project. In particular I would like to thank Ir. J. van Lier and Ing. L.A. van Bokhoven for their support and Ir. C. Hezemans and A. van Dijk for their advice on servo technical subjects.

2. THE LASERVISION SYSTEM

2.1 Optical systems for storage purposes

Many systems are known for (semi-) permanent storage of information. Among them are the magnetic tape (audio-, video- and datastorage), the magnetic disc (datastorage) and the audiorecord. The disclike media have the advantage of direct access, which means one can access a piece of information without the need to access irrelevant information. This direct access feature was also thought desirable for videostorage. Therefore, in 1969, investigations were started at the Philips Research Laboratories to develop a disclike storage medium for video.

One of the problems was the density with which the information was to be stored. Videodisc would still not be possible if the idea of optical read-out had not been born. Light permits, because of its small wavelength, very small things to be discriminated on the surface of a disc (in the order of microns). The solution was to store the information in a series of miniscule pits in a reflecting surface. The thus prerecorded disc can be read out by an intense light beam. In the resulting system, the lightbeam is produced by a helium-neon laser.

From the laservision system for videostorage, two other optical systems evoluated: the compact disc system for audiostorage and the DOR (Digital Optical Recorder) system for data storage. While the laservision system stores its information in an analog fashion (as will be explained later), the other two systems store their information digitally. This makes the compact disc system also suitable for data storage. Of the three systems DOR is the only one that can also record data. The laser in the recorder can 'burn in' data (but in the contrary to the magnetic disc, occupied space on a disc is never released after erasing data). The other two systems need prerecorded discs.

2.2 The optical disc in the laservision system

In the laservision system, the audio and video information on the disc are stored in a spiral track, starting at the inside at a fixed diameter and moving to the outside (figure 2.1 and 2.2). Average track pitch (the distance between the tracks) is 1.6 micron. The information in the track exists of small depressions, called 'pits' with variable distance and length. The pit width is 0.4 micron, the pit depth is approximately 0.1 micron. Coded in this way, one television picture requires a disc surface of 0.6 square millimeters.

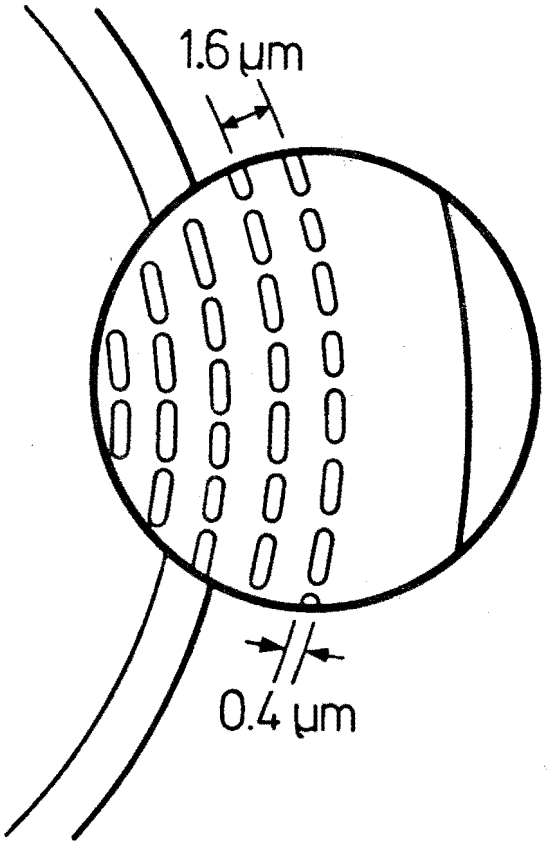


Figure 2.1: The disc surface.

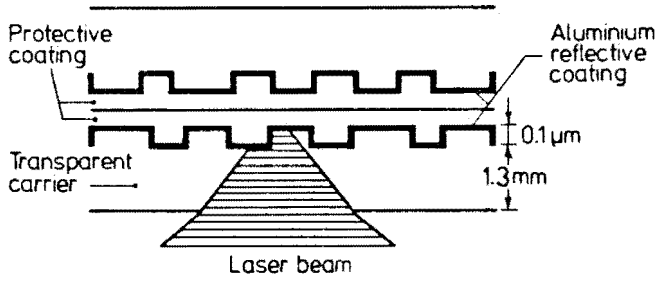


Figure 2.2: Cross section of the disc, showing the protection of the information.

There are two types of discs which are called CAV or active play and CLV or long play. CAV stands for Constant Angular Velocity which means that the speed of rotation is constant (1500 rpm for PAL/SECAM and 1800 rpm for NTSC). This type of disc has a maximum playing time of 36 minutes per side and allows special playing modes such as stills, slow motion etc., which will be explained in detail later. The other type is CLV, Constant Linear Velocity. Here the speed of rotation decreases inversely proportional with the read-out diameter, as

a result of which, more information can be stored on the disc, leading to a maximum playing time of about one hour per side. On the other hand these discs can only be played in a continuous way i.e. in the normal playing mode.

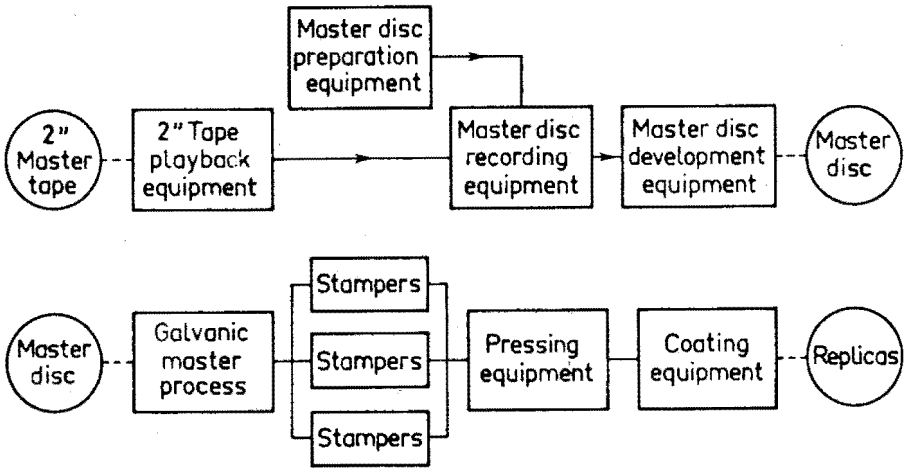


Figure 2.3: Video disc mastering and replication.

The production process for a video disc is more or less comparable with that used for conventional gramophone records. First, a master recording is made. It consists of a glass plate with a photosensitive layer deposited on one side. The coded signal of the information to be stored modulates the beam of a 100 mW laser which 'writes' the information in the surface of the master disc.

'Cutting' is done on a real time basis, that is, it requires only as much time as the programme lasts. In principle, every normal type of TV signal source (camera, magnetic tape, slide and film scanner) can be connected to the master recorder. In practice, however, 2" magnetic recording tape will be used as a programme carrier.

Exposure to the laser beam is followed by a development process, which leaves a pattern of pits on the master, from which via a galvanic process stampers are made. These stampers are used for disc production in a way similar to pressing gramophone records, or by means of a recently developed completely new process typically adapted to Laservision disc requirements.

After pressing, an extremely thin metal coating of no more than about 0.04 micron thick is deposited on the information side which is then sealed with a protective layer. As a final operation two discs are glued together resulting in one disc that can be read from both sides.

2.3 Optical principles

The most important advantages of the optical system are the contact free read-out of the information, as a result of which wear of disc or read-out device is non-existant and the possibility of an effective protection of the information on the disc against the influence of dust, fingerprints, etc. The optical principles used are based on light-ray diffraction, a phenomenon that occurs, if the object dimensions are of the same order of magnitude as the wavelength of light. Well-known is the light diffraction by means of a narrow slit (figure 2.4 a). An analogous situation occurs, if a light beam impinges on a reflecting surface with depressions of the shape and size of the afore-mentioned pits. Particularly strong is the effect, if a light spot, comparable in size to the wavelength of light, is concentrated on a pit with accurately chosen depth (figure 2.4 b). In the case of a flat surface nearly all the light is reflected and can be picked up by a photodiode fitted in the beam path, whereas if a depression is present, the major part of the light is deflected and accordingly less light is detected by the photodiode (figure 2.4 c).

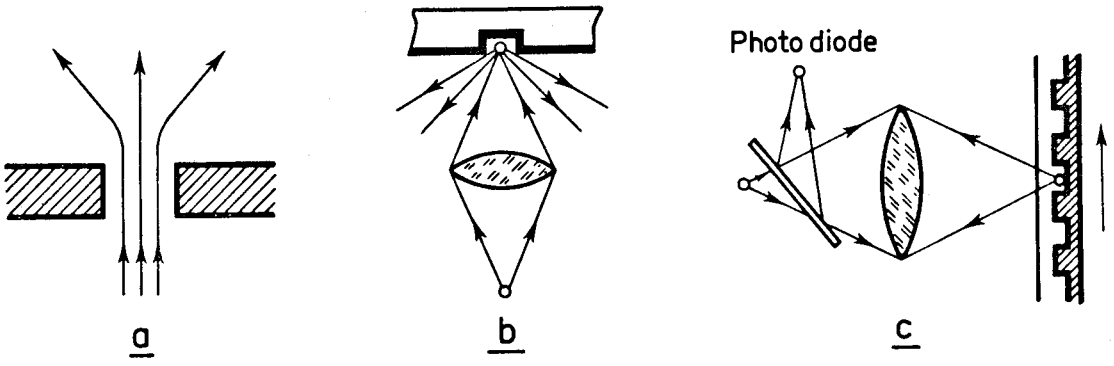


Figure 2.4: a) Diffraction of light in a narrow slit.
b) Deflection when the light spot is focused on a 'pit' of well-defined depth.
c) Modulation of the reflected light by means of 'pits' passing the light spot and resulting in a corresponding signal on the photodiode.

In order to store on a 30-cm diameter disc a video programme of half an hour for the CAV-type and one hour for the CLV-type, an extremely high information density is required. There is, however, a fundamental lower limit to the size of the details, that can be read, depending on the wavelength of the

light and the numerical aperture (N.A.) of the objective lens. The numerical aperture is defined as the product of the refraction index and the sinus of the angle between the optical axis and the outermost light ray contributing to the imaging: N.A.=n sin a (figure 2.5).

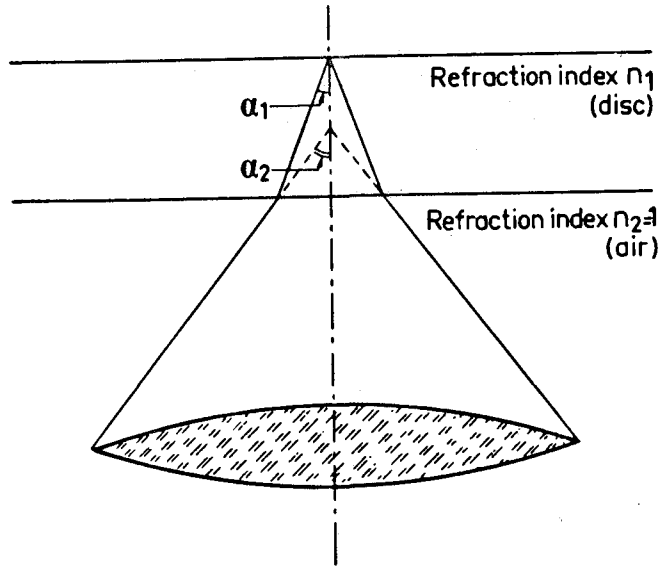


Figure 2.5: Numerical aperture.

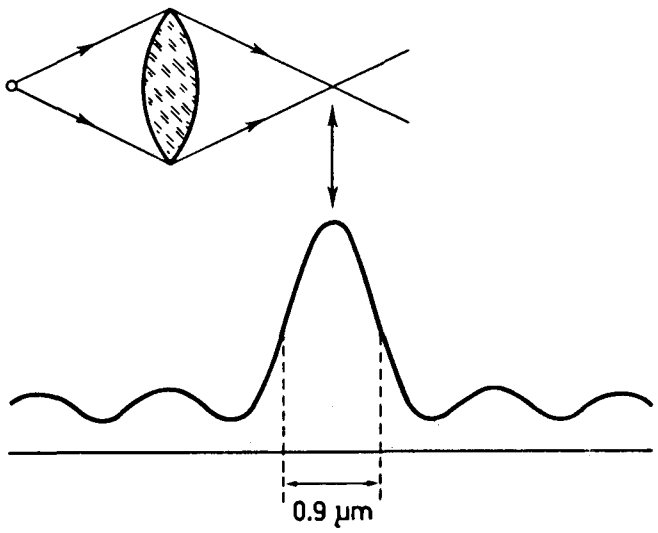


Figure 2.6: The half intensity diameter of the focused spot.

Due to diffraction at the lens aperture the light 'spot' in reality is a spot with around it annuli of decreasing brightness. If spot diameter is defined as the half-intensity diameter it is found that for N.A.=0.4 and a wavelength of the

light used 0.63 micron, the minimum (half-intensity) spot diameter is 0.9 micron (figure 2.6). This indicates that there is a maximum spatial frequency that can be read with this system, which in fact is directly proportional to the numerical aperture of the objective lens and inversely proportional with the wavelength of the light used.

The wavelength of the light being determined by the laser, the cut-off frequency of the signal at a given speed of rotation depends only on the numerical aperture of the read-out objective lens. Limitations here are, to a certain extent, the price of the objective lens, since with increasing N.A. its construction is more complicated, but much more the decrease in focal depth, which is inversely proportional to the square of the numerical aperture. Thus higher demands are to be met on disc flatness or by the focusing system servo. The high information density on the disc and the resulting microscopically small information details require special measures for the protection of this information. It is clear that even very small dust particles may have a detrimental effect and result in great many signal drop- outs. However, by using the transparent plastic disc itself as a protective layer, dust, fingerprints and scratches on its surface are out of focus and thus have no or hardly any influence on the amount of reflected light. It is important that the thickness of the protective layer is large in relation to the focusing depth. It can be seen that a thickness if 1.3 mm for one disc side in this respect offers no problem whatsoever. For a thin, disposable disc the protective layer could be reduced to 300 micron or even 200 micron if less stringent quality demands are accepted.

2.4 Coding

In the Laservision optical video disc there is a single information track in which all the information is stored for the reproduction of a colour- television programme with two sound channels and data signals. The non- lineaarity of the master recording process limits the choice of possible enncoding techniques, and a two-level signal recording was found to be the most attractive solution. On this track the information is encoded in the length and the spacing of the pits or, in other words, for a rotating disc in the repetition frequency, determined by the average length of the pits, and a pulse-width modulation of the frequency, determined by the modulation of the length of the pits (figure 2.7). The high frequency signal is FM-modulated by the composite video signal with top sync at 6.76 MHz, black at 7.10 MHz and white at 7.90 MHz. Including the resulting side bands, the frequency spectrum encompasses a frequency range until approximate 2.5 MHz at the lower side (figure 2.8).

The audio signals are equally FM modulated on carriers of 684 kHz and 1066 kHz with a frequency swing of 100 kHz. The audio carriers modulate the video FM-signal resulting after limiting in a pulse-width modulation that is used to modulate the intensity of a laser beam in the masterrecording machine.

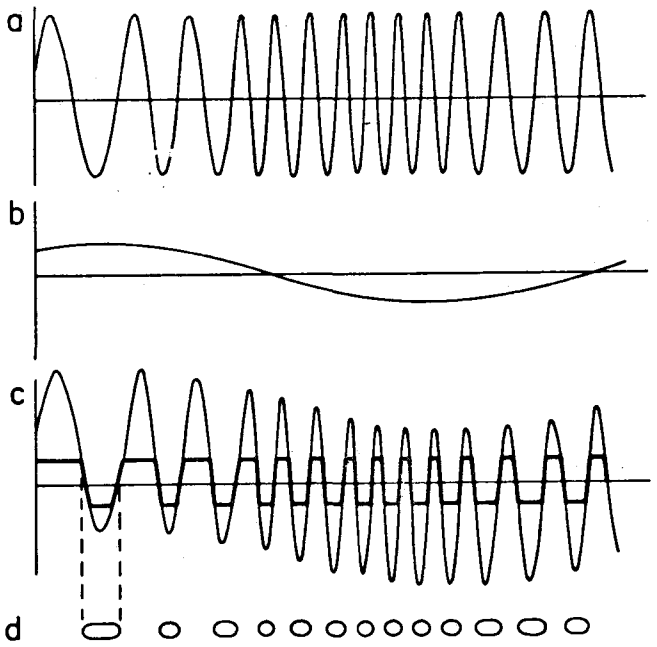


Figure 2.7: The modulation of the signal.

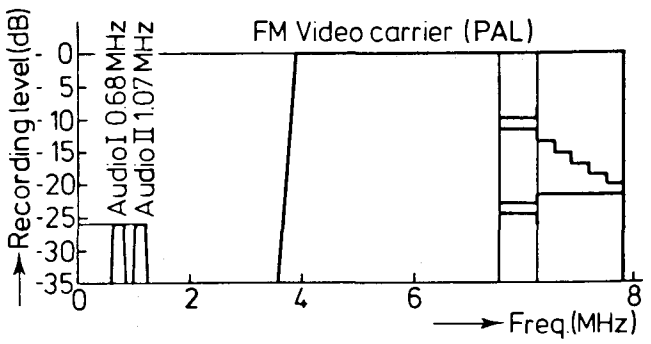


Figure 2.8: Electrical coding.

Apart from video and audio information, the disc contains a number of signals, inserted in the non-visible lines of the blanking periods. These are e.g. VIR (Vertical Interval Reference) and VIT (Vertical Interval Test) signals for test purposes (lines 19, 20, 332, 333) and further digital address signals for various purposes (lines 16, 17, 18, 329, 330, 331).

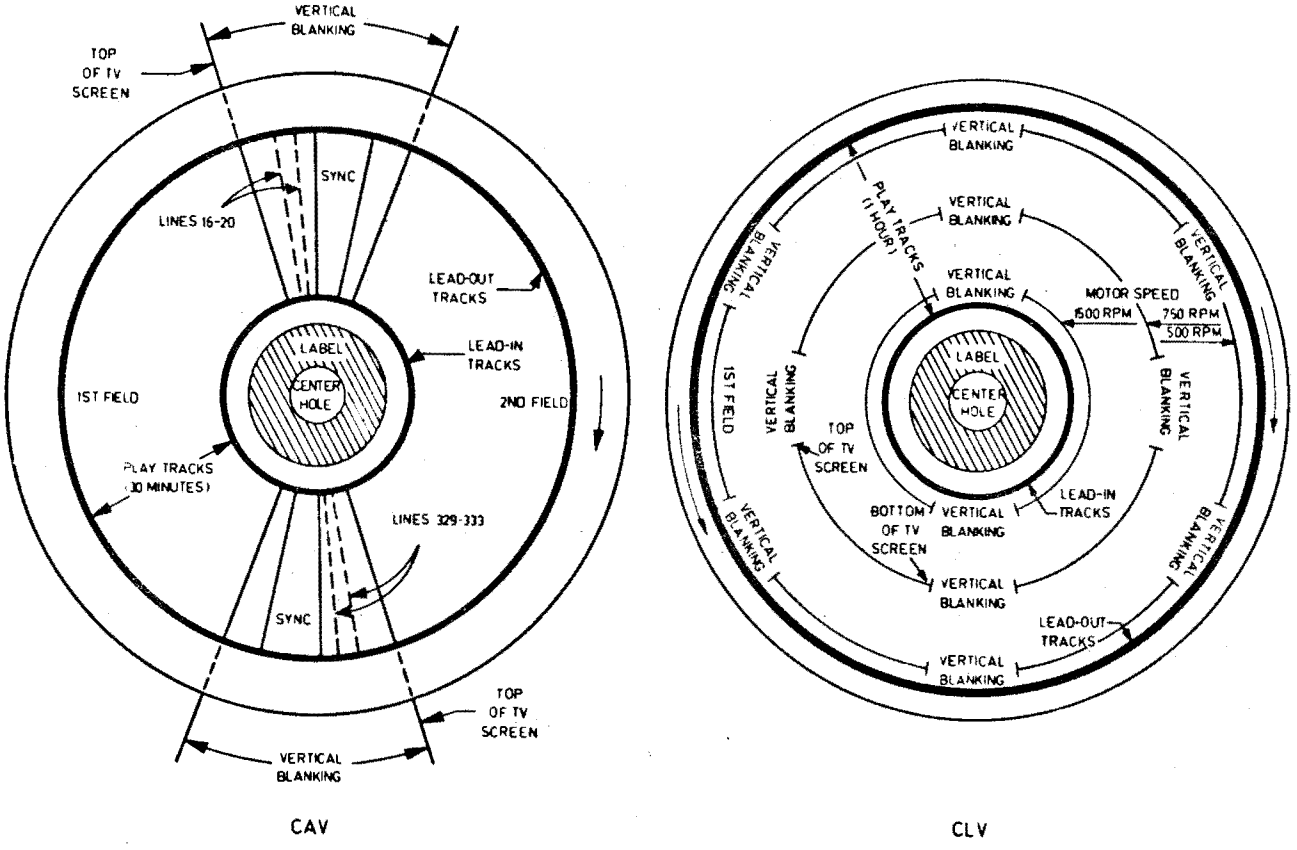


Figure 2.9: Comparison between CAV and CLV discs.

The address signals have the following functions:

LEAD-IN TRACKS. A minimum of 900 tracks prior to the start of the actual programme contain a start code which sends the read-out objective to the beginning of the programme on the disc at nine times its normal speed.

LEAD-OUT TRACKS. A minimum of 600 tracks immediately after the end of the programme contain an end code which sends the read-out objective back to the start at 75 times its normal speed. A muting circuit in the player suppresses video and audio signals during this period.

PROGRAMME AREA. There has to be distinguished between CAV and CLV types of discs:

CAV (ACTIVE PLAY)

- Picture code consisting of a picture number by means of which each picture of a programme can be identified and displayed on the TV screen if desired. It is always present in CAV dics in one field of a full frame and can be succeeded in the next field by a stop code which causes the player to inter-

rupt the normal playing mode and perform a still picture.

- Chapter code consisting of a chapter number by means of which a chapter can be identified. This information can be displayed on the TV screen if desired. If a chapter code is present it can interrupt a search action as soon as the start of a chapter is reached.

CLV (LONG PLAY)

- Continuous play code is always present in CLV discs and disables the player functions still picture, slow motion, fast motion and reverse play.

- Time code. Instead of the picture code as used in CAV discs a time code is always present in CLV discs during the programme area. It contains a time coding with hour and minutes indication indicating the elapsed time of a programme which can be displayed on the TV screen if desired.

- An optional chapter code.

2.5 Focusing

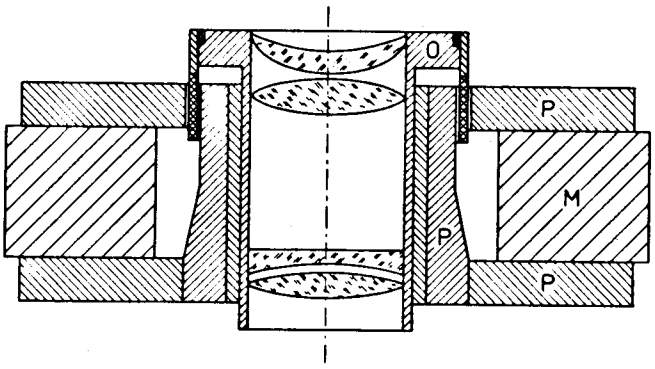


Figure 2.10: The moving objective.

Microscopically small information details have to be read from the disc. Therefore, an objective lens is required with a large numerical aperture and thus with a small depth of focus. With N.A.=0.4 a maximum out-of-focus is allowed of 2 micron. In view of the tolerances in disc and in player contruction this accuracy can only be realised by means of a servo-control system including a moving read objective that can follow the undulations of the disc (figure 2.10).

The mounting of the objective closely resembles a loudspeaker voice coil. When a current is driven through the coil, the objective is accellerated, in a direction and with an accelleration depending on the direction and the strength of the cur-

rent.

For understanding the way in which the control signal is obtained, it is necessary first to have a closer look at the photodiode on which the light- beam, reflected by the disc, falls (figure 2.11). In fact the diode consists of three segments of which the middle one is composed of four quadrants. The light beam will, when correctly focused, create a circular spot on the middle diode. All four quadrants will receive an equal amount of light. However on its way to the diode the reflected beam passes a cylindrical (astigmatic) lens. The lens gives the beam two focal lines perpendicular to each other. This means that the circle spot will become an ellips when the objective is not correctly focused. Then the amount of light received by each quadrant will not be the same and a difference signal can be derived.

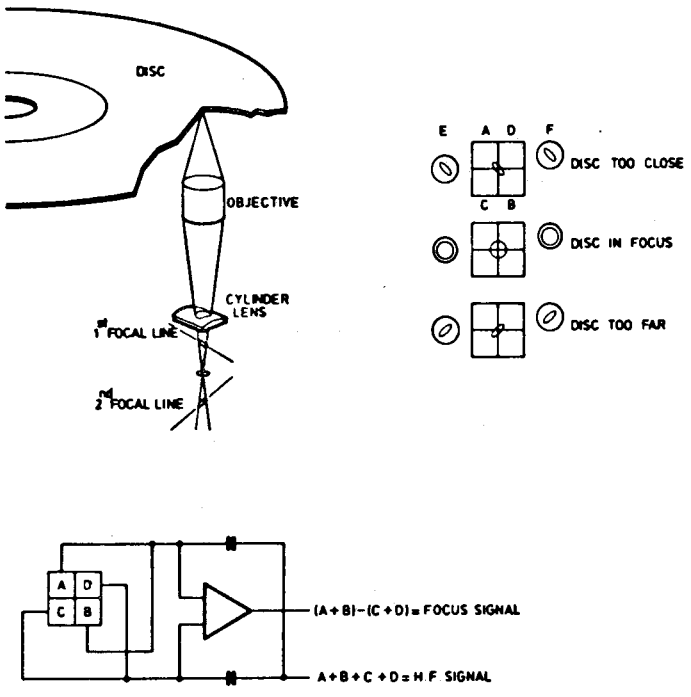


Figure 2.11: The principle of astigmatic focusing.

The largest amplitudes in vertical movement of the disc occur at the speed of rotation (25 Hz) and decrease rapidly for higher frequencies. Disc specification sets the maximum value of the accelleration at 10g but the player can cope with higher accellerations in order to provide a sufficient safety margin.

2.6 Radial tracking

The information on the disc is contained in a spiral that is read from the inside to the outside. For this purpose the 'read' objective and other optical elements are mounted on a sledge, driven by a small DC-motor, and moving radially under the disc. With an average track pitch of 1.6 micron and a speed of rotation of the disc of 1500 rpm, this means an average linear speed of the sledge of 2.5 mm per minute. The scanning light beam has to remain focused on the track with a radial accuracy of 0.1 micron, a requirement that cannot be met by a purely mechanical guidance system. By varying the speed of the drive motor by incorporating it in a servocontrol system certain slow corrections are possible. However, to cope with the effects caused by e.g. eccentricity of the spinning disc additional measures are required and use is made of a pivoting mirror by means of which the light spot can move radially over the disc. This mirror is mounted in an assembly resembling the construction of a moving coil galvano- meter where the coil is part of the radial servo-control circuit. The information in the track can only be read out optically and the deviation if the beam from the centre of the track can therefore also only be measured optically (figure 2.12). For this purpose two auxilliary beams of light are used which are slightly displaced from the centre line of the track, in opposite directions, so that they are partly on and partly alongside the track.

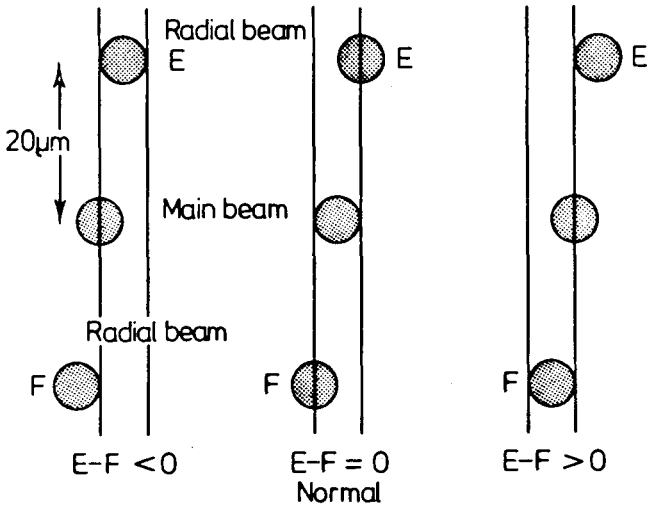


Figure 2.12: The principle of track following.

After reflection at the disc, the two auxilliary beams fall each on their own photodiode (E and F in figure 2.11). The difference signal of the two diodes after amplification passes a low-pass filter with a cut-off frequency of 20 kHz and is then used as an error signal in the control system. If the average position of the mirror deviates from its zero position, the average current is used to control the slide motor for corrections of the position of the sledge.

Unroundness of the track and unavoidable tolerances in player construction may result in a total eccentricity of 100 microns for the combination which means that at 25 Hz a reduction of at least 60 dB is required to keep the read spot within a tenth of a micron from the track.

2.7 Time error compensation

A television picture consists of lines that have to be 'written' in an exactly defined time interval (64 microseconds for PAL).

Deviations result in a distorted picture and moreover in phase errors causing colour abberations. In view of this, broadcast TV transmitters are crystal controlled for stability and the receivers are equipped with synchronisation circuits that ensure that even with deteriorated signals e.g. in fringe areas, a perfect synchronisation is maintained. In a video disc player, however, we are faced with a number of problems as a result of which the linear speed of the track, as seen by the objective is not constant. The causes if this difficulty are to be found in deviations if the disc and in unavoidable tolerances on the centring of the disc on the turntable.

The eccentricity of the track is the main cause ot time errors located at 25 Hz. Since the player/disc combination allows for a maximum eccentricity of 100 microns it can be calculated that the resulting time error is 11.5 micro- seconds. Only a maximum error of 10 nanoseconds is permitted.

To minimize time errors the phase of the line synchronisation pulses is compared with that of a crystal-controlled oscillator and the derived signal is fed to the motor-speed control circuit. Furthermore, a second pivoting mirror, scanning the track tangentially, is used. A signal derived from the burst is used to control the movements of the tangential mirror.

Another type of time error occurs in playing modes where 'track jumping' is incvolved. In these cases it is necessary that all corresponding synchronisation pulses on adjacent tracks are positioned on a straight radial line.

2.8 Special playing modes

The laservision system offers several special playing modes when CAV discs are used. This makes the system emminently suitable for e.g. instructive programmes. The special playing modes are an exclusive feature of an optical read-out system and originate from the possibility to move rapidly with the light beam from one track to another during the fly-back period.

The special playing modes are:

- Still picture: a reverse jump after every revolution (figure 2.13 a).
- Reverse motion at normal speed: a reverse jump after every half revolution (figure 2.13 b).
- Fast forward at three times normal speed: a forward jump every half revolution (figure 2.13 c).
- Slow motion: still picture and at adjustable times a forward (or reverse) jump.

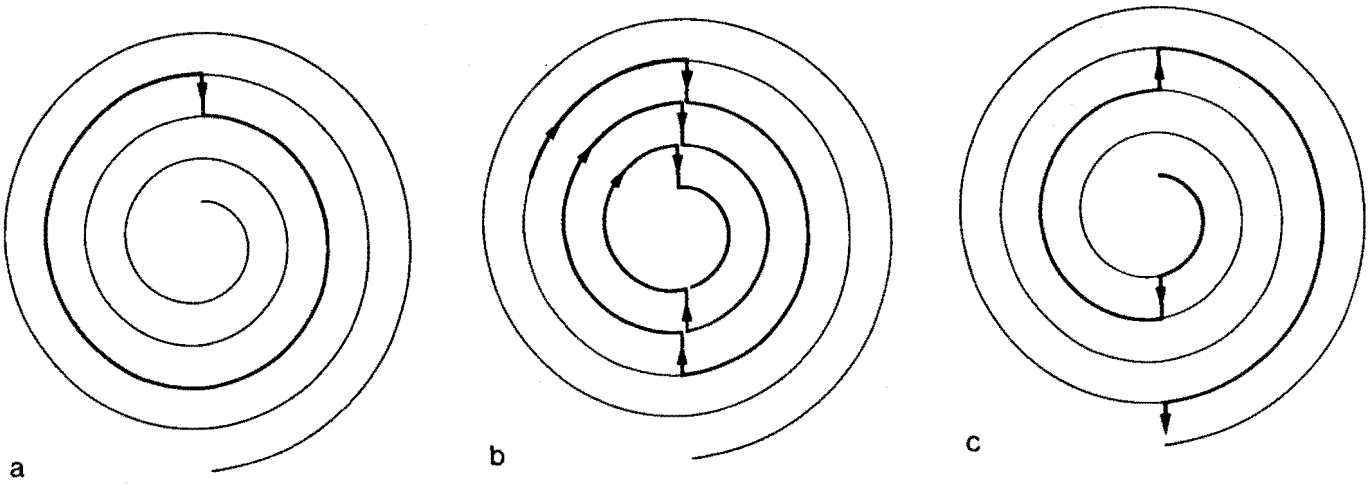


Figure 2.13: Special playing modes.

To enable the beam to jump from one track to another during the fly-back period, use is made of the fact that the open control loop of the radial servo has a low resonance frequency, so that it behaves like a ballistic galvanometer. The beam jump is effected by opening the control loop and applying an accellerating current pulse followed by an equally large retarding pulse through the coil of the radial tracking mirror and then closing the loop again.

2.9 The light-path

The first part of the light-path (figure 2.14) is the source: a 1 mW Helium-Neon laser. This source produces monochromatic light with a wavelength of 632.8 nm. The light is coherent and linearly polarized. The next part is the grating. It splits the main beam into three beams with intensities 1:4:1 (higher order beams are not used). The spotlens focuses the three beams on a plane about 10 mm behind the lens. The three beams are deflected by 90 degrees by the beam splitter. The polarisation of the light is changed from linearly to circularly by the 1/4 lambda plate. The light is deflected again by 90 degrees by the folding mirror. The light passes the collimator lens which turns the diverging beams into parallel beams. The light is deflected by the radial tracking mirror, a moving mirror, mounted within a coil and pivoting on a vertical spindle. The light is deflected by the tangential tracking mirror, a moving mirror, mounted within a coil and pivoting on a horizontal spindle. The light is focused on the disc by the objective, a moving microscope objective 20 x N.A. 0.40. The light follows the same path back, but its polarisation is turned by the 1/4 lambda plate to linear, 90 degrees turned in respect to the original beams. Therefore it is not deflected by the beam splitter and ends up on the photodiodes (after passing the cylinder lens, described in the section focusing).

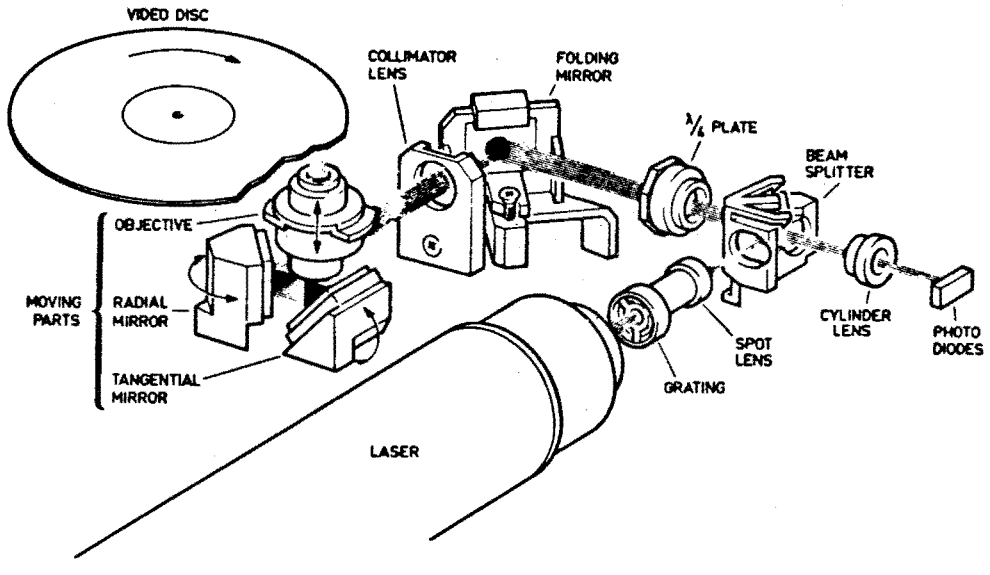


Figure 2.14: The light path.

3. AN INTRODUCTION TO SERVO SYSTEMS

3.1 Introduction.

A servo system may be defined as follows (1): a control system is referred to as a servo if the output is designed to follow as closely as possible a given reference signal. In the special case where the reference signal is constant, we talk about a regulator rather than a servo. By this definition, the LaserVision servos are actually regulators. However, since in all other articles on LaserVision those controllers are referred to as servos, we will do so also.

The LaserVision focus servo e.g. controls the relative position of the objective to the disc i.e. the optical distance (in which the different lightspeeds in air and the substrate are accounted for) between them. The focus servo has to keep this distance constant within a few microns, whilst the moving range of the objective has to be 6 mm. The disturbances, i.e. the movements of the disc have to be compensated for. These disturbances are limited by the disc specification (8) according to figure 3.1. The vertical disc stroke (d(t)) convoluted with the inverse Laplace transform of the transfer function in figure 3.1 (h(t)) must be smaller than 2E-6. This results in a limit to the acceleration, that must be given to the objective, to reduce the disturbance sufficiently. The disc specification states that this accelleration maximally may be 9 g (9 times the earth gravity accelleration).

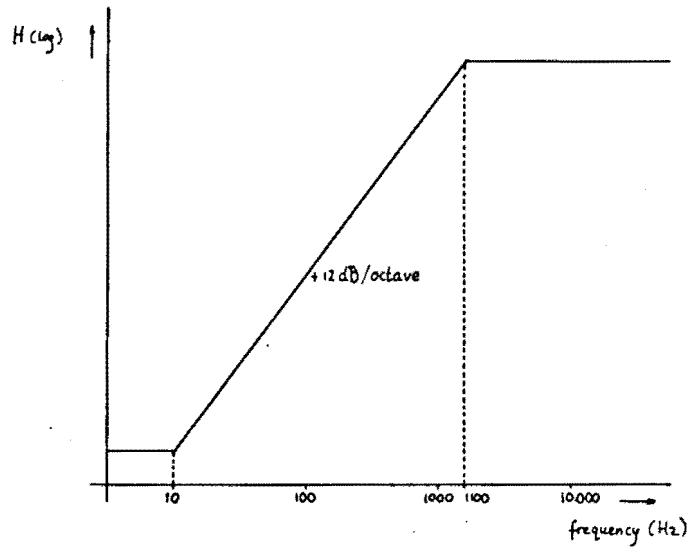


Figure 3.1: Disc specification of surface disturbances.

In addition to this restriction the disc stroke must stay

within 1 mm in each direction. This example gives an indication of the precision achievable with help of a servo system.

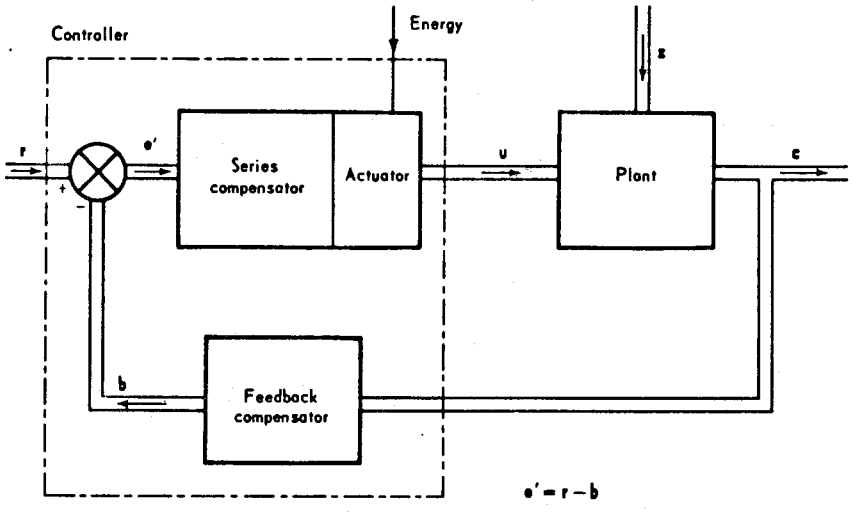


Figure 3.2: Typical servo structure.

An error vector is defined: E(t)=R(t)-C(t), and we then are able to judge the quality of the servo by its ability to minimize in some sense this error vector.

The basic system structure is depicted in figure 3.2. The controller portion of the overall systems incorporate:

1. The actuator, which supplies the control force U. The actuator requires external power for its operation and may, in some instances, be an inseparable part of the plant itself.

2. The comparator-compensator, which constitutes the 'brain' of the system. This portion of the controller usually consists of passive networks, the main functions of which are to shape the dynamics of the signals before they enter the actuator portion. Quite often, it is necessary to include active networks (integrators, for instance) in the compensator design.

In this introductory chapter we will discuss servos with linear controllers only. For more details on linear (and nonlinear) control, the reader is referred to the standard works (1,2).

3.2 Analysis and synthesis tools for linear servos.

In this section the tools are discussed by using the focus

servo as an example. This serves two purposes: one, it is easier this way to explain the use of the tools and two, one gets to know the working of the focus servo.

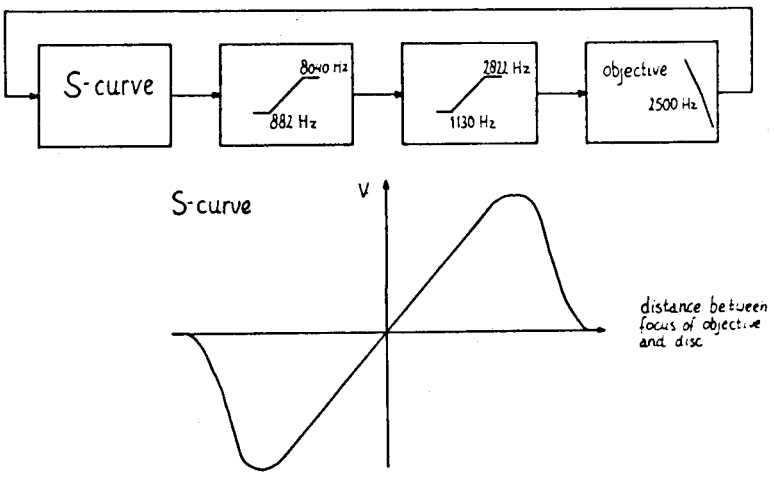


Figure 3.3. The LaserVision focus servo, the S-curve is the amplitude response of the quadruple photodiodes.

Figure 3.3. depicts the LaserVision focus servo. The focus servo consists of a second order lead-lag network. The output of the lead-lag network is amplified and fed to the actuator. The actuator (the 'loudspeaker' coil) is actually part of the process. The process is the objective that focuses the laser beam on the disc. The movement of the disc surface perpendicular to the disc (due to the disc-rotation) is the disturbance fed into the system. The reflected beam falls on the quadruple photodiode. As is explained in section 1.5, the difference signal obtained from these diodes is a measure for the defocusing. Once the laser beam is focused, this signal is used as an error signal. The error signal is fed into the lead-lag network.

The complex transfer function of the total open loop system can be written as follows:

$$H(s) = K_r \cdot \frac{1+s\tau_1}{1+s\tau_2} \cdot \frac{1+s\tau_3}{1+s\tau_4} \cdot \frac{K_p}{s^2(1+s\tau_5)}$$

The frequency transfer function as:

$$H(j\omega)=K_r\cdot\frac{1+j\omega\tau_1}{1+j\omega\tau_2}\cdot\frac{1+j\omega\tau_3}{1+j\omega\tau_4}\cdot\frac{K_p}{(j\omega)^1(1+j\omega\tau_5)}$$

A third description method is the state-variable method, but this method will not be discussed further.

Servo systems can be analyzed in the time domain and in the frequency domain. The specifications of the focus servo are given in the frequency domain. The reason for this is quite clear: although the disturbance signal is largely periodic (with a period of one revolution of the disc) it is of random origin and its identification is made on the basis of its spectral composition. It is also much easier to test the servo by means of sinusiodal test-signals. The testing can be done while the player is in operation. Therefore, time domain analysis will not be discussed here.

3.2.1 bode plots

The frequency transfer function can be written as:

$$20\log|H| = 20\log K + 10\log[1+(\omega T_1)^2]+\cdots-20N\log\omega-10\log[1+(\omega T_2)^2]-\cdots$$

$$\angle H = \tan^{-1}(\omega T_1)+\cdots-N\frac{\pi}{2}-\tan^{-1}(\omega T_2)-\cdots$$

For the focus servo this becomes:

$$20\log|H| = 20\log K_r\cdot K_p + 10\log[1+(\omega\tau_1)^2]+10\log[1+(\omega\tau_3)^2]-40\log\omega-10\log[1+(\omega\tau_2)^2]$$
$$-10\log[1+(\omega\tau_4)^2]+10\log[1+(\omega\tau_5)^2]$$

$$\angle H = \tan^{-1}(\omega\tau_1)+\tan^{-1}(\omega\tau_3)-\pi-\tan^{-1}(\omega\tau_2)-\tan^{-1}(\omega\tau_4)-\tan^{-1}(\omega\tau_5)$$

Bode suggested to plot these functions on semi-log paper. The gain is expressed in db (20log) and the phase is expressed in radians. On semi-log paper, the function -20Nlogw will be a straight line with a negative slope of -6N db per octave. The terms of the form $10\log[1+(\omega T)^2]$ are also handled very conveniently. If omega is much larger than 1/T, the term approaches the asymptote 20log(wT). This is a straight line intersecting the 0 db level at the frequency $\omega=1/T$ (break frequency). If omega is much smaller than 1/T, the term approaches the abcissa. At the break frequency, the term equals 3 db. To obtain the overall gain, one only has to sum up terms which consist basically of straightline elements. The same goes for the overall phase. The bode plots of the focus servo are shown in figure 3.4.

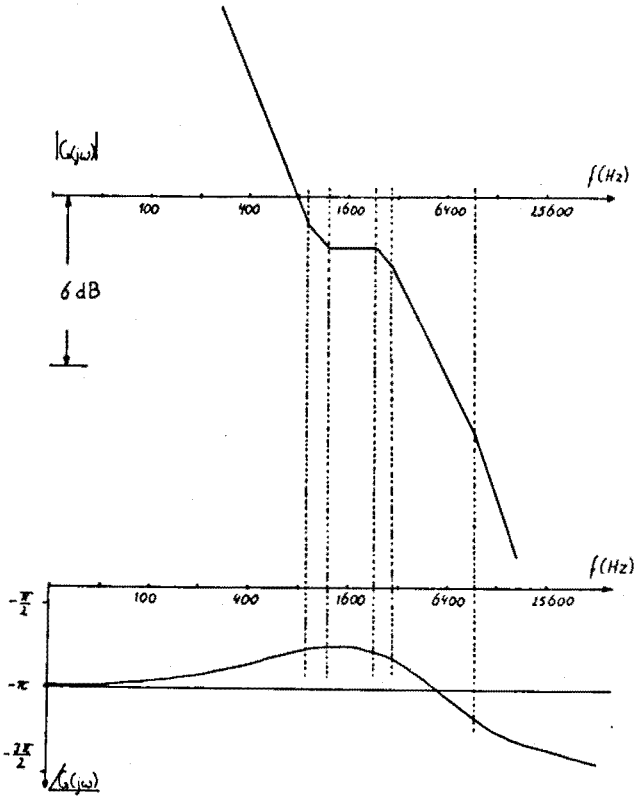


Figure 3.4: The bode plots of the focus servo.

3.2.2 nyquist diagrams

Contrary to the bode method, by which method two plots are needed (one for gain and one for phase) the nyquist method presents the two characteristics in one figure. A nyquist diagram is easily constructed and it provides a great deal of information concerning the nature of response, relative stability, and steady-state errors of a linear servo system.

The nyquist method leans heavily on the theory of conformal mapping. Consider a function G(s), where s is a complex variable. This theory states that, for every point in the s- plane, a corresponding point G(s) is found in the G(s)-plane. One can say that the function G maps the points in the s- plane onto the points in the G(s)-plane. This is done such that a small square in the s-plane is mapped on a small square in the G(s)-plane. In general, the squares are of different size and are differently oriented with respect to the coordinate axes. Of course, the function G(s) is the open- loop transfer function and s is the complex frequency. The positive imaginary axis of the s-plane is mapped onto the G(s)-plane and the resulting

line depicts G(jw) (for positive frequencies). In one figure this gives the gain (the distance between the point in the G(s)-plane and the origin) and the phase (the angle between the abcissa and the line that connects the point with the origin) for positive frequencies. A servo system is stable if and only if the mapping of the entire right half of the s- plane does not contain G(s)=-1. The nyquist diagram of the focus servo is represented in figure 3.5.

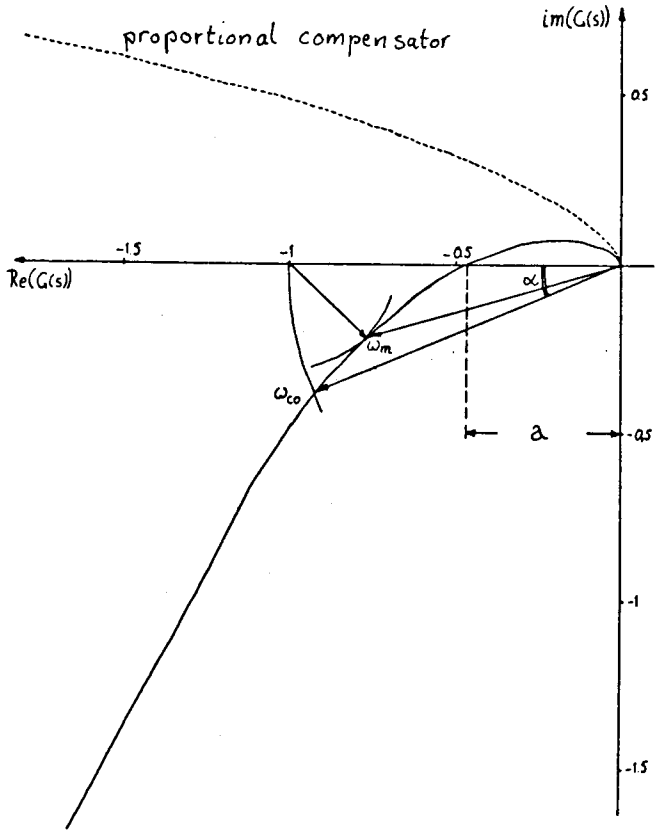


Figure 3.5: The Nyquist diagram of the focus servo.

The nyquist diagram is also a means to determine the relative stability of a servo system. The less stable a system is, the closer the eigenvalues are to the jw axis, the closer the G(jw) plot will be to the critical point -1. The effect of the degree of stability on the closed loop frequency transfer function can readily be seen from the nyquist plot.

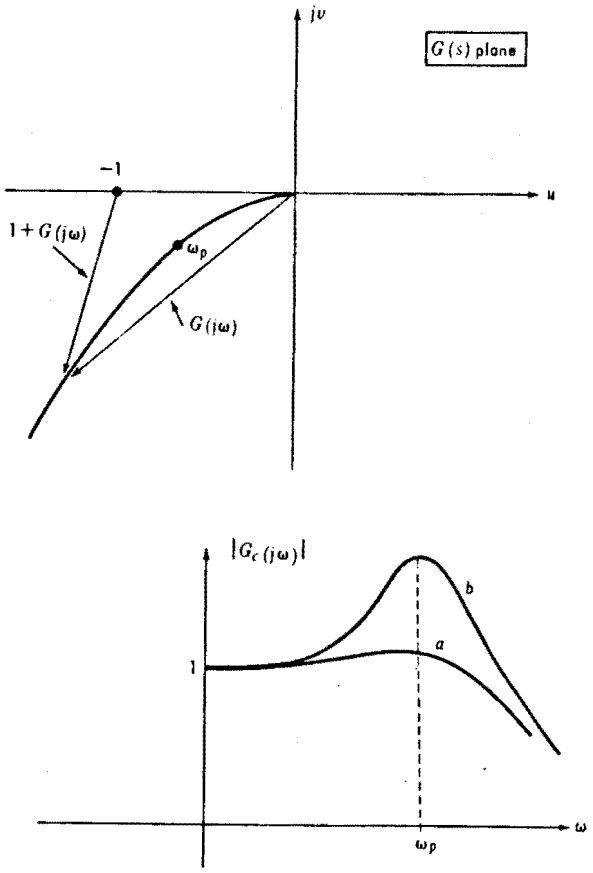


Figure 3.6: Correlation between nyquist plot and closed-loop frequency transfer function.

Figure 3.6 shows a resonance peak at the frequency where the nyquist plot passes close to the critical point -1. An other measure of the degree of stability can be obtained by means of the gain and phase margins. The gain margin (GM) is defined as the gain increase needed to drive the system in to instability. With reference to figure 3.7 this means:

$$GM \triangleq \frac{1}{a} - 1$$

The phase margin (PM) is defined in association with the crossover frequency, that is, the frequency at which the open-loop gain equals unity:

$$PM \triangleq \angle G(j\omega_{co}) - 180^\circ$$

With reference to figure 3.5, we can say that the focus servo is a stable system with GM=2, PM=23 degrees and a resonance peak at 4100 Hz. These of course are the theoretical values obtained with an open-loop gain of 10^8. This figure also demonstrates the function of the lead-lag networks. To this purpose, the nyquist diagram of a focus servo with a proportional compensator is plotted in the same figure (the dotted line). One can clearly see that the resulting servo is unstable.

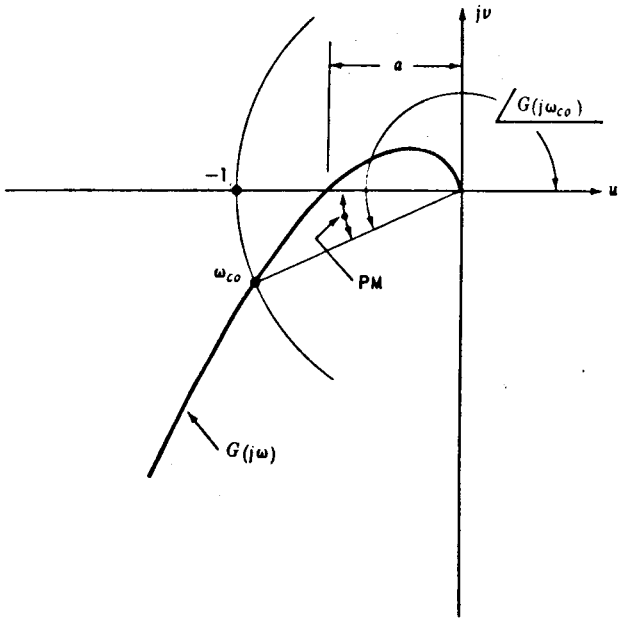


Figure 3.7: Definition of GM and PM in terms of the nyquist plot.

3.2.3 the root-locus method (RLM)

The RLM is the most important synthesis technique for single-input-single-output linear servomechanisms. The RLM is a mixed graphical-analytical procedure that can be used even in the study of moderately nonlinear systems and systems with slowly time-varying parameters.

The closed-loop eigenvalues or the poles of the closed-loop transfer function

$$G_c(s) = \frac{G(s)}{1 + H(s) \cdot G(s)}$$

completely determine the response of the servomechanism. by

means of the RLM one can find the traces (=loci) in the s-plane of the eigenvalues as they, under influence of changing parameter values, move in distinct patterns (root loci) accross the s plane. The root-locus method makes it possible to obtain the loci without direct solution of the characteristic equation. And for that matter the order of the system is not important.

If we possess full information concerning the location of the closed-loop poles and zeros, we can obtain directly the time and the frequency response for a system. Furthermore, it is possible from the pole-zero configuration to draw some extremely valuable conclusions concerning the relative importance of the different eigenvalues.

All points on the loci must satisfy the characteristic equation.

$$K\frac{\prod^{q}(s-Z)}{\prod^{n}(s-P)} = -1 \qquad \begin{array}{l} Z = HG\text{-zeros} \\ P = HG\text{-poles} \end{array}$$

Being a complex equation it can be separated into two parts: a modular part and an angular part.

The construction of root-loci is done by the 9 rules stated below:

1. The closed-loop zeros coincide with the G zeros and the H poles.
2. The root-loci wil always be symmetrical with respect to the sigma axis. Sections of the sigma axis may constitute part of the loci.
3. The general character of the root-loci is obtained by satisfying the angular part:

$$\sum^{q}\angle(s-Z) - \sum^{n}\angle(s-P) = -\pi + \vartheta 2\pi \qquad \vartheta = 0,1,2,\cdots$$

   The specific positions of the eigenvalues along the loci are determined from the modular part:

$$K\frac{\prod^{q}|s-Z|}{\prod^{n}|s-P|} = 1$$

4. There are a total of n root-locus branches, each starting from one of the n HG poles for a gain setting that approaches zero.
5. For very high loop gain, the root-loci will either approach any one of the q HG zeros or end up in infinity.

6. Those n-q loci which tend to infinity will do so along straight-line asymptotes.
7. All asymptotes must pass through the 'center of gravity' (CG) of the HG pole-zero cluster. The CG is computed on the basis that all poles have the mass +1 and all zeros the mass -1.
8. The angle measured relative to the sigma axis at which a locus leaves a pole will be referred to as 'angle of departure'. It can be computed from the formula:

$$\varphi_{dep.} = \pi + \sum_{i=1}^{q} \angle(P_k - Z_i) - \sum_{j=1,\neq k}^{n} \angle(P_k - P_j)$$

9. The angle at which a locus arrives at a zero will be referred to as 'angle of arrival'. It can be computed from the formula:

$$\varphi_{arr.} = -\pi + \sum_{j=1}^{n} \angle(Z_k - P_j) - \sum_{i=1,\neq k}^{q} \angle(Z_k - Z_i)$$

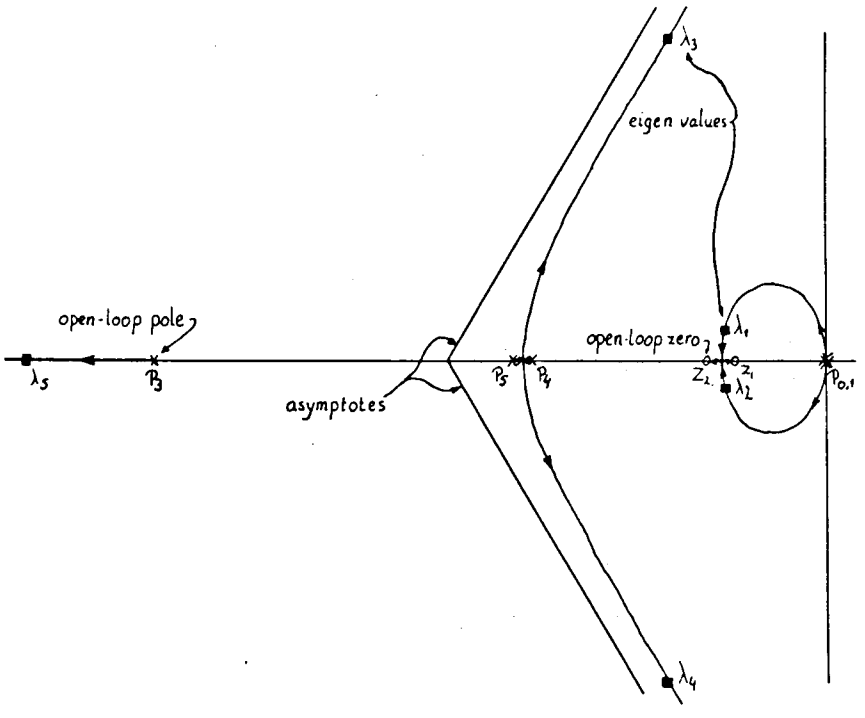


Figure 3.8: The root-locus diagram of the focus servo.

10. The break-away points can be found by solving the angular

equation for a (variable) point close to the abcissa and replacing the angles by their tangents.

Figure 3.8 depicts the root-locus diagram of the focus servo. From this plot one can obtain the time response in the following way: derive the n closed-loop eigenvalues and the m closed-loop zeroes. Then the closed-loop transfer function can be written in the form:

$$G_c(s) = K_c \frac{\prod^{m}(s-z_i)}{\prod^{n}(s-\lambda_j)} = \frac{\prod^{n}(-\lambda_j)\cdot\prod^{m}(s-z_i)}{\prod^{m}(-z_i)\cdot\prod^{n}(s-\lambda_j)}$$

From this equation one can obtain the time response to a step input by successively performing factoring and a reverse Laplace transformation (after multiplication with 1/s for the step function).

We can obtain the closed-loop frequency transfer function from the closed-loop transfer function:

$$G_c(j\omega) = \frac{\prod^{n}(-\lambda_j)\cdot\prod^{m}(j\omega-z_i)}{\prod^{m}(-z_i)\cdot\prod^{n}(j\omega-\lambda_j)}$$

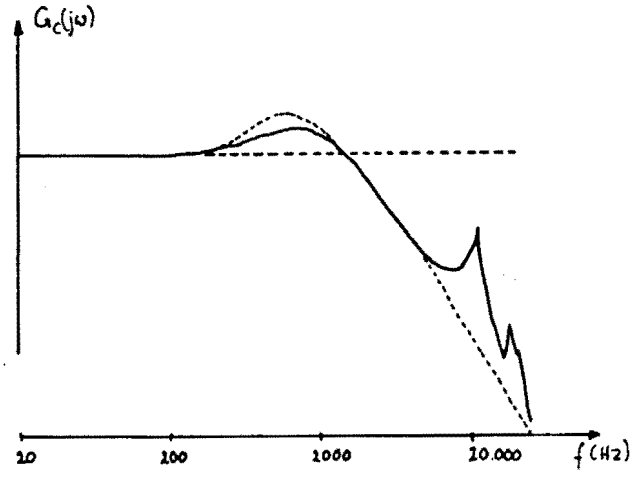


Figure 3.9: Frequency response of the focus servo.

and this equation can be solved graphically. By means of this method one can immediately see at which frequency one can expect a resonance and how well an oscillatory effect at that frequency is damped. Also one can see that the dynamic respon-

se of the system is largely dictated by the two complex conjugated eigenvalues (at the higher frequency). Those poles are called dominant. The focus servo responds dynamiccally as if it were a spring-mass system. Figure 3.9 depicts the theoretical (dotted) and practical frequency response of the focus servo.

3.3 Design methods for analog servos.

When synthesizing a servo system, one must be able to judge the quality of the designed system. To be able to do so, a set of performance specifications must be specified. Such a set can be composed of the following elements (1):

Frequency-domain specifications:
1. Bandwidth
2. Phase margin (and crossover frequency)
3. Gain margin
4. M peak (resonance peak and peak frequency)
5. Deviation ratio
6. Error-constant-bandwidth ratio
7. Output impedance
8. Gain-bandwidth product

Time-domain specifications (in response to a step input):
1. Delay time
2. Rise time
3. Percentage overshoot
4. Settling time
5. Final (static) value of error

One of the major problems occurring in almost every servo system is saturation. The saturation effect occurs when an error-signal, before or more likely after filtering, should become larger than the largest possible signal in the circuit. Since this is not possible, the signal will become the largest possible signal in the circuit (in most instances the supply voltage). The effect adds an important non-linearity to the transfer function of the servo which makes a predictable performance impossible. By choosing the right amplification values in the right places, the designer should try to prevent saturation to happen. If this is impossible, the effects should be minimized. Clearly the occurence of saturation is a sixth time-domain specification.

The synthesis methods can be classified into three different approaches (1):

The trial-and-error method.

Using this method, one selects a (suboptimal) set of performance specifications, a configuration and a set of parameters. Subsequently, while the performance specifications are not met, the design goes through several iterations in which the chosen sets are subject to changes. The method leads to fast results if the proces of iterations converges.

The analytical method.

The design procedure starts out with a performance specification consisting of a single figure of merit. Next, a configuration and a desired output are chosen. Subsequently the closed-loop transfer function is chosen such that the figure of merit is minimized. As long as the configuration is fixed, the problem reduces to the relatively simple search for those controller parameter values that minimize the index of performance. If not, the analytical method is very difficult to use.

The optimal design method.

The method starts out with an index of performance that as adequately as possible incorporates all the factors that add to the cost of the performance. The only constraints imposed upon the solution are those that relate to the physical restrictions on the magnitudes of the components of the state and control-force vector (using state-variable description). Next, a control-force vector is sought that minimizes the chosen performance criterion. This is a very difficult task and in most practical cases one must be content with a numerical solution. It may be difficult to implement the optimum strategy into hardware and the computational requirements may be to severe to render the optimum solution practically feasible. This method gives a unique solution that is truly optimal. It can be applied to nonlinear and time-variant systems. It is the only technique that can handle multiple-input-multiple-output systems.

3.4 Digital servo systems

3.4.1 description

The output sequence for any linear time-invariant system is obtained from the input sequence by the repeated evaluation of the convolution sum relation (5):

$$y[n] = \sum_{k=-\infty}^{\infty} h[k] \cdot x[n-k] \qquad -\infty < n < \infty$$

where h[n] is the response of the system to the unit impulse sequence. The convolution sum equation is very similar in form to the convolution integral that describes the operation of a continuous-time linear time-invariant system. In contrast to the analog system, however, the convolution sum equation serves not only as a theoretical description of discrete linear time-invariant systems in general, but it can be used to implement certain types of linear systems.

As in the analog case, Fourier analysis is a valuable tool in the theory and design of discrete signals and systems. The discrete-time Fourier transform representation is defined by the equations:

$$X(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} x[n] \cdot e^{-j\omega nT}$$

$$x[n] = \frac{T}{2\pi} \int_{-\pi}^{\pi} X(e^{j\omega T}) \cdot e^{j\omega nT} d\omega$$

A notable property of $X(e^{j\omega T})$ is that it is always a periodic function of ω with period $2\pi/T$.

In the analog case, the Laplace transform is often more useful and convenient than the Fourier transform, because it can be used to represent a wider class of signals and because algebraic expressions involving the Laplace transform are less cumbersome than those involving Fourier transforms. For the same reasons the z-transform is often preferred to the Fourier transform for discrete sequences. The z-transform representation is defined by:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] \cdot z^{-n}$$

$$x[n] = \frac{1}{2\pi j} \oint_C X(z) \cdot z^{n-1} dz$$

where C is a closed contour lying in the region of convergence of the power series. Comparison of the Fourier transform and the z-transform shows that:

$$X(e^{j\omega T}) = X(z)\Big|_{z=e^{j\omega T}}$$

i.e., the Fourier transform, when it exists is just the z-

transform evaluated on a circle of radius one in the complex z-plane.

One of the most important reasons for the use of frequency domain representation is the result that if y[n] is the output of a linear time- invariant system, then its z- transform (and thus its Fourier transform) satisfies the equation:

$$Y(z) = H(z)\,X(z)$$

where H(z) and X(z) are the z-transforms of the unit sample response of the system and the input to the system, respectively.

Another advantage of the Fourier transform representation is that it provides a very convenient means of showing the relationship between a sequence of samples and the original analog signal from which the samples were obtained. Specifically, if x[n]=xa(nT), then:

$$X(e^{j\omega T}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_a(\omega + 2\pi k/T)$$

where $X_a(\omega)$ is the Fourier transform of the analog signal xa(t). From this relationship it is clear that there is a possibility that the images of the Fourier transform may overlap and since they are added together, it would be impossible to unscramble the effects of this aliasing distortion. Figure 3.10. illustrates the implications of this for two sampling rates. No aliasing distortion occurs if $X_a(\omega)$ is bandlimited (the highest frequency at which $X_a(\omega)$ contains energy is called the Nyquist frequency) and if the sampling frequency is greater than twice the Nyquist frequency. Thus, an analog signal to be sampled must be bandlimited to at the most half the sampling frequency. Most analog signals are contaminated with additive noise, so analog lowpass filtering is almost always necessary prior to sampling. This effect can also be achieved by sampling at high rates and perform the lowpass filtering digitally. After that the sampling rate can be reduced by decimating (throwing away samples).

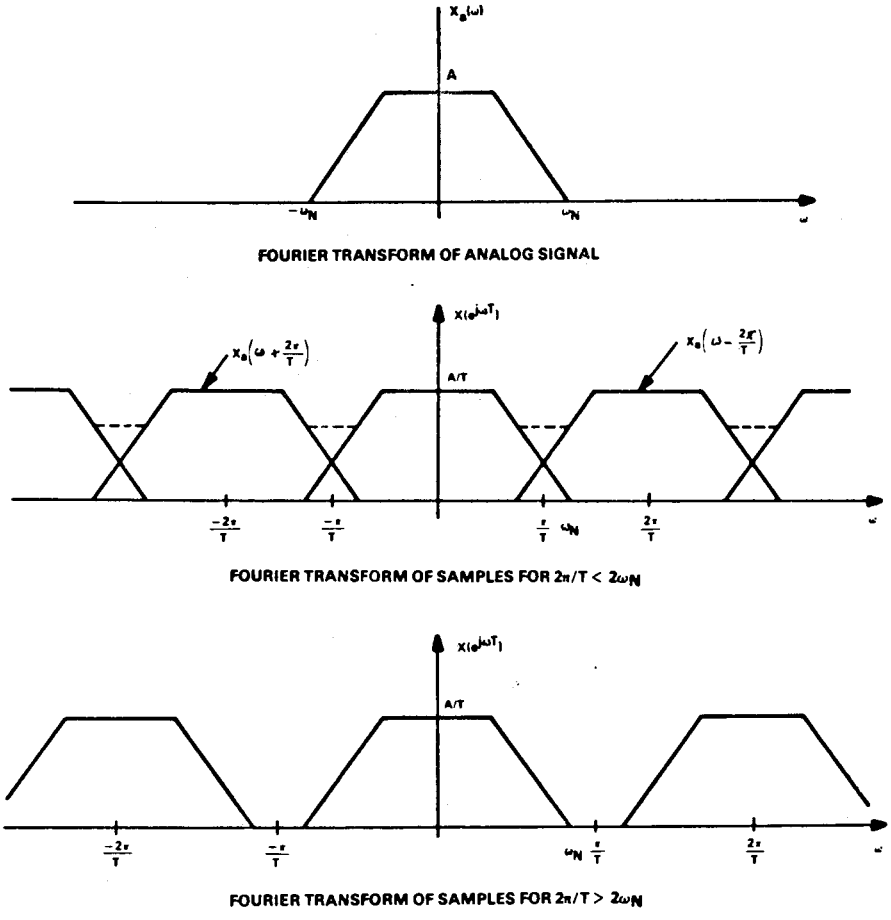


Figure 3.10: Aliasing distortion.

Discrete time systems usually are described by means of their difference equation. From the difference equation, the z-transform can be obtained by replacing a one-sample-period-delay by a multiplication by 1/z. Z stands for exp(-sT), where T stands for the sample period.

A difference equation:

$$p_0 e[k] + p_1 e[k-1] + p_2 e[k-2] = q_0 u[k] + q_1 e[k-1] + q_2 e[k-2]$$

The matching z-transform:

$$G(z^{-1}) = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2}}{q_0 + q_1 z^{-1} + q_2 z^{-2}}$$

From the difference equation, one can see that typical arith-

metic operations in a digital servo are a summation of a series of products.

3.4.2 Obtaining a discrete model from an analog model

One should realize that a linear servo (regulator) system consists of an amplifier and a filter (to shape the dynamics). For analog filters exist several techniques for translation to digital filters (3). Two desirable properties of any mapping from continuous to discrete space are:

1. The j-omega axis in the s plane should be mapped to the unit circle in the z plane.

2. Points in the left half of the s plane (RE[s]<0) should be mapped inside the unit circle.

We will present the most frequent used methods:

A. Mapping of differentials:

This is the simplest way to digitize a continuous system. One replaces the differentials in the differential equation of the continuous system with finite differences. This way one obtaines directly a difference equation that approximates the given differential equation. But by using backward differences, the left half of the s plane is mapped onto a circle with centre RE[z]=1/2 and radius 1/2. By using forward differences the j-omega axis of the s plane is mapped onto RE[z]=1. Though it is possible, using generalized differences, to obtain the desired conformal mapping, it is very difficult and therefore seldomly used.

B. Impuls invariant transformation:

The second technique for digitizing an analog filter is called the impuls invariant transformation. The characteristic property preserved by this transformation is that the impulse response of the resulting digital filter is a sampled version of the impulse response of the analog filter. In consequence of this result, the frequency response of the digital filter is an aliased version of the frequency response of the corresponding analog filter (for the aliasing effect, see subsection 3.4.1). This means that the analog filter must be bandlimited to half the sample frequency and it generally requires that a guard filter be used to guarantee that the analog filter be suitably bandlimited prior to transformation. Clearly, as the sample frequency becomes higher, the effects of aliasing become neglegible and the digital and analog frequency responses become comparable.

C. Bilinear transformation:

A simple conformal mapping from the s plane to the z plane which has the two desired properties which is algebraic in nature and which maps the entire j- omega axis in the s-plane to the unit circle in the z-plane is the bilinear transformation defined by

$$s \rightarrow \frac{2}{T}\frac{(1-z^{-1})}{(1+z^{-1})}$$

Since the entire j-omega axis of the s plane is mapped onto the unit circle in the z plane, the aliasing errors inherent with impulse invariant transformations are eliminated. There is a highly nonlinear relationship, however, between the analog frequency (Ω) and the digital frequency (ω). The nature of this nonlinearity is seen by the definition of the bilinear transformation:

$$j\Omega \rightarrow \frac{2}{T}\frac{(1-e^{-j\omega T})}{(1+e^{-j\omega T})}$$

which can be written as:

$$\Omega \rightarrow \frac{2}{T}\tan\left(\frac{\omega T}{2}\right)$$

This shows that the digital frequency response will be a warped version of the analog frequency response. By pre- warping the cut-off frequencies of the analog filter this effect can be compensated for. Furthermore, it has the properties that realizable, stable continuous systems are mapped to realizable, stable digital systems. One disadvantage of this technique is that the frequency response of the continuous system must be piece-wise constant to compensate for the effects of the nonlinear relation between analog and digital frequencies. Also neither the impulse response nor the phase response of the analog filter is preserved in a digital filter obtained by bilinear transformation.

D. Matched z transformation:

A fourth technique for digitizing an analog filter is called the matched z transformation and is a direct mapping from poles and zeros in the s plane to poles and zeros in the z plane. The mapping has the property that an s-plane pole (zero) at s=-a maps to a z-plane pole (zero) at z=exp(-aT) where T is the sampling period. It should be clear that the continuous transfer function H(s) should be in factored form to apply the

matched z transformation.

The matched z transformation is not a suitable mapping in cases when the analog system has zeros with center frequencies greater than half the sampling frequency or when the analog system is an all-pole system. In general, use of the impulse invariant or bilinear transformation is to be preferred over the matched z transformation.

3.4.3 cascading linear systems

When cascading linear systems, the z-transforms of the systems may be multiplied when there is a sampler between the systems. When not, the Laplace transforms must be multiplied first before z-transformation is applied. Of course, like with the Laplace transformation, multiplication of the z- transforms is the same as convolution of the respective impulse responses (2).

3.4.4 proportional and integral behaviour

For processes with proportional behaviour, the gain is obtained by using the final value theorem:

$$K = \frac{e(k\to\infty)}{u(k\to\infty)} = \lim_{z\to 1} G(z) = \frac{p_0+p_1+\cdots+p_m}{q_0+q_1+\cdots+q_m}$$

Processes with integral behaviour have a pole at z=1:

$$G(z) = \frac{p_0+p_1z^{-1}+\cdots+p_mz^{-m}}{(1-z^{-1})(q_0'+q_1'z^{-1}+\cdots+q_{m-1}'z^{-m+1})}$$

If b differs from 0, the system has a jump discontinuity at k=0. However, for most real processes b is zero because, for synchronous sampling at the input and output, at least the lag behaviour of the actuators and sensors avoids the jump discontinuity.

3.4.5 poles and stability of discrete time systems

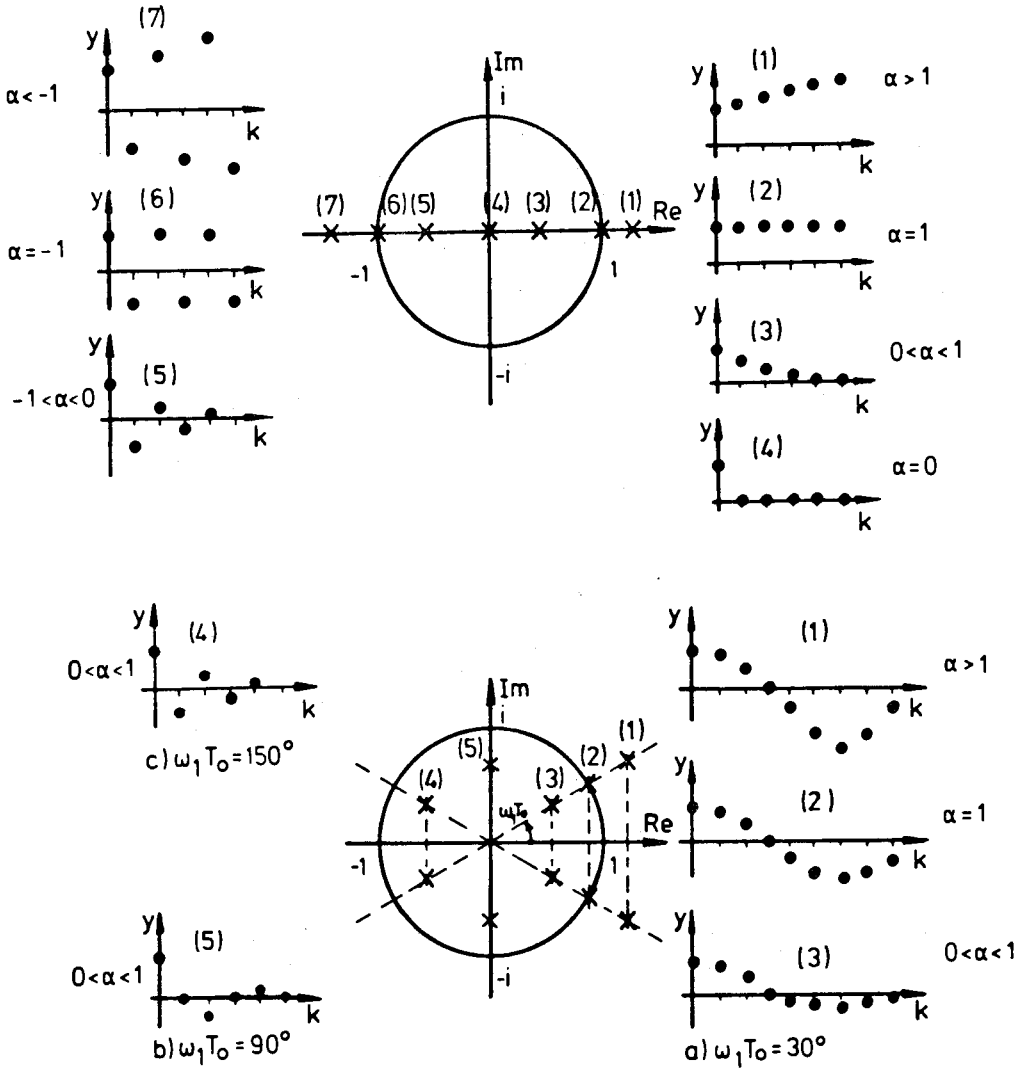


Figure 3.11: Poles of discrete time systems and the corresponding eigen behaviour.

The poles in the s-plane and the z-plane are related by: $p_d = e^{-p_a T}$ which means that s-poles lead to positive z-poles. Therefore, a negative z-pole has no corresponding s-pole.

A linear system is asymptotically stable if after an initial perturbation it returns to the equilibrium point (2). This is the case if the system poles are located inside the unit circle. Figure 3.11 illustrates the behaviour of a system with one real pole or one conjugate complex pole pair.

3.4.6 the microprocessor as digital controller

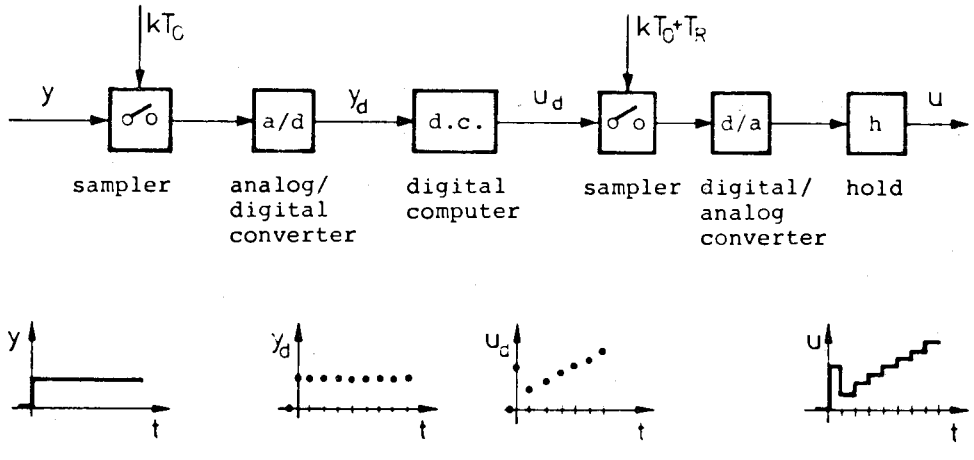


Figure 3.12: The computer as sampled data controller.

A microprocessor based controller design is depicted in figures 3.12 and 3.13. To process analog signals by a microprocessor, those signals have to be sampled, hence the presence of the sample-and-hold amplifier in the schematic of figure 3.13.

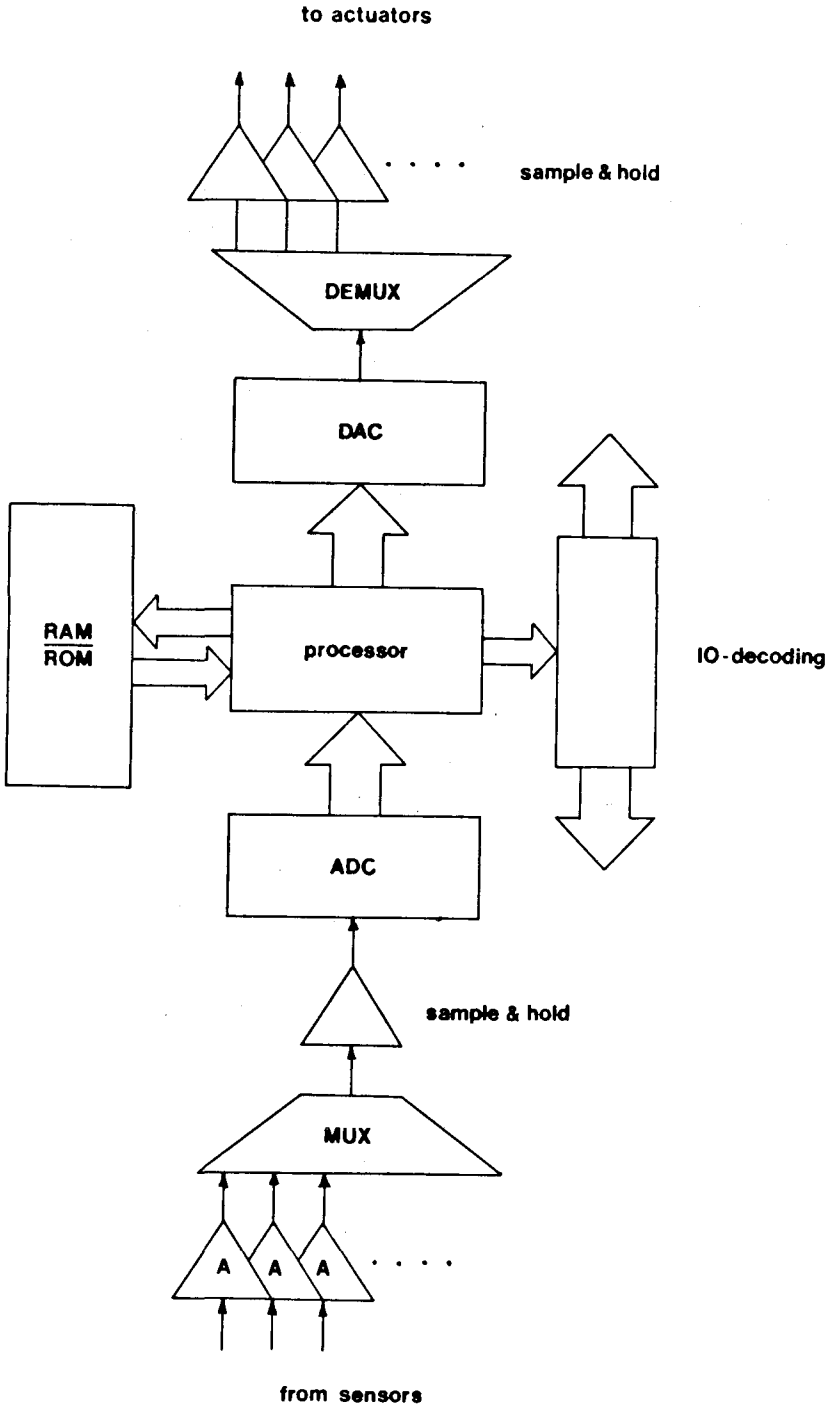


Figure 3.13: A microprocessor based controller.

Furthermore, the signals have to be converted to digital data by an ADC (Analog to Digital Converter). To apply the processed signal to the actuator, a DAC (Digital to Analog Converter) is necessary. That is of course, if the actuator needs analog signals (when the actuator is e.g. a switch or a stepper motor, DA conversion is not necessary). A microprocessor

is an expensive device (compared with analog electronics). Still it is relatively cheap if it can be used to perform more than one control function. Therefore a multiplexer and individual amplifiers (to ensure that the signal range matches with the input range of the ADC and the output range of the DAC) are needed on both sides of the system. Also if more than one control function is incorporated the driver amplifiers have to be of the sample- and-hold type. Apart from input and output, the microprocessor also needs memory for program storage and storage of intermediate data, and circuits which decode the control signals for the ADC, DAC, etc.

3.4.7 intrinsic advantages of digital servo systems

A digital servo system can be designed such that it can be programmed. This provides an enormous flexibility. When a designer disposes over a suitable development system, parameters can be changed within minutes.

Digital servo systems can be easily simulated by computers.

Digital servo systems are very well suited to be integrated in ic's. This could make digital servo systems compact and cheap.

With digital servo systems, algorithms are more easily made adaptive. This could make servo systems respond more adequately to repeating types of errors and adapt themselves to varying process parameters.

3.4.8 intrinsic disadvantages of digital servo systems

When using digital systems in servo controllers two main problems occur:

DELAY

All extra delays in a servo controller have to be minimized. The reason for this is that an extra delay reduces the Phase Margin of the total system. A small Phase Margin can give cause to instabilities. Delays caused by digitizing an analog system can be caused by:

1. Anti-aliasing filtering. This effect was discussed in 3.1.1. The delay caused by these filters depends on the order of the filter and the Q-factor.

2. Time necessary to calculate the new output samples. This delay is dependend on the speed of the processor or hardware.

3. The zero-order hold filter behind a Digital-to-Analog Converter (if necessary) causes a delay of half a sampling period.

Except for the calculation time, these effects can be diminished by increasing the overall sampling rate.

QUANTIZATION

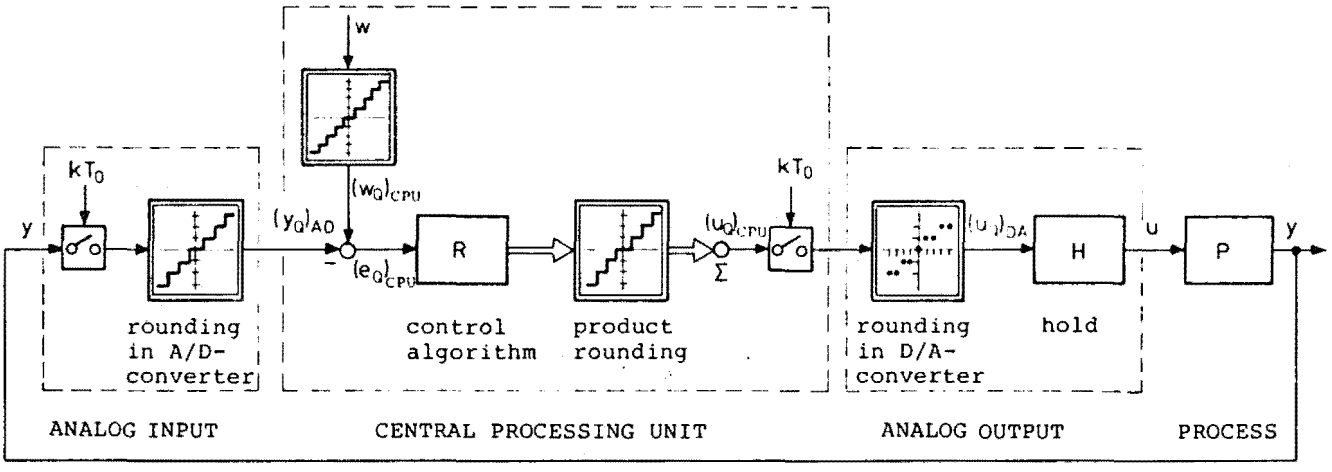


Figure 3.14: Non-linearities due to quantization.

As is shown in figure 3.14, quantization of amplitudes arises at several places in digital controllers. The principal causes can be summarized as follows (3):

1. Quantization of variables: rounding or truncating of the controlled variables in the DAC (Digital/Analog Converter), ADC (Analog/Digital Converter) or CPU (Central Processing Unit) due to limited word size.

2. Quantization of coefficients: rounding or truncating of the controller parameters.

3. Quantization of intermediate results in the control algorithm (e.g. the rounding of products).

Quantization can cause the following effects:

1. Quantization noise can be generated in the ADC and this acts as white noise on the controlled variable. Its variance cannot be decreased by any control. This leads to undesirable changes of the manipulated variable which can be larger than one quantization unit of the DAC.

2. Offsets and limit cycles (constant amplitude oscillations) can arise due to quantization.

The resulting amplitudes of quantization noise, offsets or limit cycles in the controlled variable are at least one quantization unit of the ADC. Limit cycles arise particulaly with strongly acting control algorithms. They can disappear if the controller gain is reduced. The describing function or the direct method of Ljapunov can be used in stability investigations for the detection of limit cycles. The simplest investigation of quantization effects is obtained by simulation. This is true particularly if quantizations occur at more than one location.

The influence of rounding effects in the controller parameters can usually be neglected, even in the case of fixed point representation. The only effect to be mentioned is that the frequency response changes a little.

Isermann (2) gives a set of conclusions as to how undesired quantization effects in digital control loops can be avoided:

1. The word lengths of the ADC and DAC and the numerical range of the CPU must be sufficiently large and coordinated.

2. The word lengths or dynamic range at all quantization locations must be utilized as much as possible, by appropriate scaling of variables.

3. To avoid excessive quantization errors of factors and products, the CPU word length for fixed point calculations must be significantly larger than that of the ADC.

4. If limit cycles arise for a given digital controller, the controller parameters should be modified to give a weaker controller action (detuned).

5. For feedforward control algorithms and digital filters one must take care of the dead band effect (offset caused by product rounding, which are multiples of the quantization units of the products) around the steady state.

The word length of the ADC should be chosen such that its quantization error is smaller than the static and dynamic errors of the sensors. For digital control the word length of the DAC can be taken such that one quantization unit if the manipulated variable arises, after transfer through the process, in about one quantization unit of the ADC.

4. THE ASSIGNMENT AND REQUIREMENTS

4.1 Recapitulation of the assignment

Before we discuss the requirements that have to be met, we first recapitulate the assignment. In chapter 1, the assignment was lined out rather roughly. Here we will go into it in more detail. Actually, the assignment consists of four parts. These parts are chosen such that in the end a completely working prototype was constructed. During the project the evoluating prototype had to be tested in combination with a real LaserVision player. The resulting controller had to be a trial system with which experiments (for a digital servo) could be set up quickly.

1. Design and construct a digital servo controller to control the focus process using a microprocessor, a DAC and an ADC. This boils down to selecting for all components a proper wordlength and speed accordingly to the necessary accuracy and sample frequency, selecting a proper microprocessor with matching development tools, selecting a suitable DAC and ADC and adding the necessary digital and analog circuitry. Also a program for the microprocessor needs to be written and for that purpose an appropriate control algorithm has to be selected. A digital filter must be constructed (to be performed in software). In the software a start-up facility must be provided. Compare the performance of this controller with the original (analog) servo controller. To this end, appropriate performance criteria must be selected and these criteria must be applied to both the analog and the digital controller.

2. Eliminate the DAC, that is to say, replace it with a circuit (and matching software) which could more easily be integrated in an ic and be controlled by digital signals. Make an inventory of the methods of controlling analog actuators with digital signals and select the most useful one. Design and construct the necessary extra hardware and software and implement it in the system.

3. Include in the system a second servo. The software for this servo has to be executed by the same microprocessor that executes the software for the first servo. The problem is to design a real-time 'multi-tasking operating system'. In the original assignment, it was intended to implement the radial servo as second servo. Due to the limitation of the duration of the project, this was not possible and instead a second focus servo (in a second player) was implemented.

4. change the controller to a predictive type, i.e. a learning system. Since the disturbances to be compensated for by the servo's are very highly corrolated between successive revolu-

tions of the video disc, it is possible to predict with high accuracy what the disturbance at a given moment will look like (6). This makes it possible to anticipate disturbances and steer the actuator in advance. By this method it would be possible to reduce the bandwidth of the filter and thus reduce the auditive noise, produced by the actuator. For this part of the assignment, it is necessary to include in the circuit a relatively large memory. this memory is required to 'remember' at each given moment the situation or state of the servo at one revolution of the video disc earlier. Also it is necessary to devise some means to synchronize this memory to the in principle variable revolutional speed of the disc.

4.2 The requirements resulting from the assignment

4.2.1 basic requirements

These requirements include:

- a lower limit to the sample frequency: the bandwidth of the focus servo should be 1.6 kHz. To have no problems caused by instabilities due to the sampling rate, this sampling rate should be at least 16 kHz (2). This poses a strong restriction on the choice of the processor and the converters.

- a lower limit to the wordlength: in view of the fact that the system was to be a try-out system, the input wordlength had to be rather high. Also we didn't want to make the ADC automatically scalable, which perhaps would make the system unnecessary complex (but which would lower the demand on input wordlength). We came to the rather arbitrary, but definitely overmeasured choice of 12 bits. The wordlength of the processor should then be at least 16 bits and the wordlength of the DAC at least 10 bits (4).

- samples should be taken at equidistant time intervals and results should be output as soon as is possible after the taking of a sample (i.e. the hold-command to the sample-and-hold amplifier).

4.2.2 the elimination of the DAC

Replacement of analog control of an actuator by some means of digital control should not deteriorate the performance of the digital servo system. Furthermore, the digital control of the actuator should not increase the auditive noise produced by the actuator.

4.2.3 the including of a second servo process

The processes should not be hindered by each other. This means that whether process 0 is starting or running or doing nothing at all, process 1 mustn't know anything about it. In fact, the most stringent requirement, which resulted from the control theory, is that samples should be taken with equidistant time-intervals. Also the control force value must be output as soon as possible after the taking of the sample. Any other process than the servo filters, which is not time-critical has to be interleaved with the time-critical processes in such a manner that the above stated demands are not violated.

4.2.4 the predictive approach

As was mentioned earlier, the processor must be able to address an extra memory space, large enough to hold the state of both servos for one revolution. The actual size of the memory depends on the sample frequency and the word-length of the states (in- or output values) of the servos. Theoretical investigations (6,7) have shown, that the bandwidth of the memory loop should be smaller than the bandwidth of the servo loop without the memory. This means, that possible the input values of the memory should be low pass filtered before entering in the memory. Of course, the loading and unloading of the memory loop should enlarge the delay, between the input of a sample and the output of a control force value, as little as possible, because enlarging this delay means reduction of the phase margin. The phase margin in its turn is a measure for the stability of a system. For synchronization of this memory to the revolutions of the video disc, a tacho- generator, which is coupled to the disc spinning motor, must be used. This can be the original tacho-generator (18 cycles per revolution) which is built into the motor itself, or it can be another (perhaps more accurate) generator. In any case, pulses derived from the tacho signal must be processed by the microprocessor. This is a typical example of a non-time-critical possibly concurrent process. Of course, if the two servos are not integrated in the same player, two of these processes are needed and the memory should be split up in two sections. When a player is scanning, the predictive controller should be disabled.

4.2.5 interaction with the video disc player

When the servos are operated in a real player which, aside from the servos, functions normally, it is not enough for the servo controller just to perform the servo filter function. The actuators have to be moved in an area in which the error signal has a well defined value. For instance the objective

has to be raised so that the focusing point of the laserbeam is within 10 micron from the reflective layer of the disc. Only then the servo can keep the objective focused on the disc. It may be needed to repeat this action for several times when once is not successful. Safeties have to be built in to guard against malfunctions, e.g. the objective hitting the disc surface or irregularities when there is no disc on the player. During normal operation, there are several signals from the player that have to be monitored. Already mentioned are the spinning motor tacho signal and the signal which indicates that the player is in the scanning mode. On behalf of the focus servo, the focus-position-indication (FPI) signal has to be monitored. This signal indicates whether the objective is within the focusing range and is derived from the sum signal of the four quadrants of the focusing diode on which the reflected (main-) laserbeam falls. On behalf of the radial servo, the track-position-indication (TPI) signal must be monitored. TPI indicates if the servo is following on the track or between the tracks and is derived from the two diodes on which the reflected secondary laserbeams fall.

The players control board must receive the same signals which it would get from the original servos. Otherwise the player might turn off the laser or stop the disc rotation. Among these are signals which indicate that the servos are in operation and are in particular important during the startup sequence and during possible calamities.

It may be necessary to low-pass filter the error signals because of the electro-magnetical noise the player produces. This extra filtering introduces an extra lag in the system and this might affect the stability of the system. The spinning motor and the laser power supply (a switched mode supply) are possible origins of noise in the form of spikes.

5. THE SELECTION OF COMPONENTS

In this chapter, the choice of components will be justified. Since the resulting circuit was to be a trial system, I did not restrict the choice to the cheapest components, that perform just good enough. At the outset, I wanted to make sure that the performance of the circuit would not be hindered by any limitations of the components. Limitations of any kind can be built in anyhow, if necessary. The above stated method also was imposed by the fact that any orders had to be issued as soon as possible, within three weeks from the start of the project. A possible long delivery time could delay the project much. So there was not much time to study over the choice of components nor was there much time to study the servo theory in advance. Only a minimum amount of theory was digested before a choice was made.

5.1 The microprocessor

5.1.1 the selection criteria

The eventual choice had to be based on the following criteria:

- The microprocessor should be equipped with fast hardware for arithmetic calculations. A hardware multiplier is a desired feature.

- Since the microprocessor not only has to perform a digital filter but also several logic control tasks, the processor has to be equipped with, preferrably parallel, IO ports. The instruction set therefore has to include bit-manipulation-, program control- and subroutine instructions.

- An interrupt facility is a desired feature to insure fast response to external events.

- With regard to the predictive controller, the processor should be able to control a large RAM, either internally or externally to the processor.

- The microprocessor should be able to perform all tasks stand-alone, that is to say, without being a peripheral to another processor.

- Not only the processor itself must be readily available, but also the appropriate development support tools, such as an evaluation module or emulator. Also it would be very convenient to have cross support software, such as a cross support assembler and a simulator.

- The delivery time for processor and support should be as short as possible.

5.1.2 the evaluated microprocessors.

The first criterion already narrows the choice down to one class of processors: the digital signal processors. Those processors generally are fast and optimized for implementations of digital filters. This of course means that they are certainly not optimized for controller tasks (criterion 2), we actually want the best of both worlds. This section presents a bird's-eye view of the evaluated digital signal processors.

2920 SIGNAL PROCESSOR (intel)

This signal processor is specifically designed to replace analog subsystems in real time processing applications. It has built in A-to-D and D-to-A converters with 4 analog input lines and 8 analog/TTL output lines. It has 25 bit precision and a 24 bit instruction word. As development tools are available: an emulator, a compiler and a diversity of filter software. Still this processor cannot be used because it is non-interruptable, there are no program control instructions, signal bandwitdth is maximally 10 kHz, there is no hardware multiplier on board and the memory, which the processor can control is too small (40x25 bits, internally).

iACX96 (8096, 8396) 16 BIT MICROCONTROLLERS (intel)

this processor isn't a real signal processor, but a very fast microcontroller with a 16 bit data (and instruction) wordlength, an internal program memory of 8 kbytes and a 232 byte register file. the memory is externally extendable to 64 kbytes. Provided on-chip are a high-speed serial port which can detect and cause transients, a watch-dog timer and a pulse width modulator. iACX96 is interruptable by 8 sources, contains four 16 bit software timers and five 8 bit IO ports and features a 250 ns instruction cycle. The processor doesn't contain a hardware multiplier but does provide a multiplication and a division instruction (6.5 microseconds). Loops are possible and the processor maintains a stack in memory. iACX96 looks promising though it is perhaps not as fast as a real signal processor in arithmetic. Still it is not a suitable candidate because samples were not available nor were there development tools.

uPD7720 DIGITAL SIGNAL PROCESSOR (NEC)

The 7720 is a fast signal processor with built in 16x16 hardware multiplier (250 ns multiplication) and features a 250 ns instruction cycle. A data word is 13 bits long and an instruc-

tion word 23 bits. Several actions are performed in parallel. The processor is interruptable, loops are possible and a hardware 4x9 bit stack is provided. The processor is readily available and so are an emulator, a compiler and various application programs. The processor has 512 words (23 bits) of program memory, 510 words (13 bits) of parameter memory and 128 words (16 bits) of data memory. A serial input- and a serial output port are provided. This processor is not suitable because memory is not externally extendable and the 8 bit parallel port is only suited as microprocessor interface where the 7720 operates as a peripheral.

S2811 DIGITAL SIGNAL PROCESSOR (AMI)

This signal processor contains a hardware multiplier (12x12 bit), a 16 bit accumulator 256x17 bit program memory (ROM), 128x16 bit parameter memory (ROM) and 128x16 bit data memory (RAM). It is provided with a serial input port, a serial output port and a microprocessor interface (8 bit wide). Program loops are possible and so are subroutines. The S2811 is not suitable because it is only usable as a microprocessor peripheral. As such the memory is not extendable and the signal processor is not interruptable.

HD61810 DIGITAL SIGNAL PROCESSOR (Hitachi)

The HD61810 is a fast processor (250 ns instruction cycle) realized in CMOS technology . It is the only single chip signal processor I encountered with integrated floating decimal point arithmetic hardware in the form of a hardware multiplier and a 20 bits ALU with a double accumulator. The data word is 16 bit long (12 bit mantissa and 4 bit exponent) and an instruction is 22 bit long. The data memory on-chip is 200 words long and furthermore the chip contains a 128x16 bit coefficient ROM and a 512x12 bit instruction ROM. The HD61810 is equipped with a microprocessor interface (8 and 16 bit wide) and a serial input and serial output interface to transfer data from an ADC to the processor and from the processor to a DAC. The processor is interruptable and loops and subroutines are possible. The HD61810 is not a suitable candidate because the memory is not externally extendable, there are no parallel IO ports (when using the processor in a stand-alone mode) and development tools were not yet available. From the processor itself only samples were available.

TMS320 DIGITAL SIGNAL PROCESSOR (Texas Instruments)

This processor is usable stand-alone only, has a 200 ns instruction cycle, 16 bit data- and instruction words. It comprises a hardware (16x16 bit) multiplier, a 32 bit accumulator and ALU, an interrupt mechanism and a 4 level hardware stack. It has 8 16 bit wide parallel IO ports, 1536x16 bit on-board

ROM and 144x16 bit on-board RAM. The program ROM is externally extendable to 4096 words in which memory (with special instructions) can also be written (for tables). An evaluation board, a cross-assembler and a simulator are available.

In short: of the evaluated (signal) processors, the TMS320 of Texas Instruments is the only suitable candidate. In the next section, we will discuss the TMS320 in more detail. The reader can find all details about the TMS320 in (5).

5.1.3 the TMS320 digital signal processor

ARCHITECTURE

A block diagram of the TMS320 is shown in figure 5.1. The TMS320 is designed with a modified Harvard architecture. A Harvard architecture is an architecture in which program memory and data memory lie in two seperate spaces. There is a separate program and data bus and this permits a full overlap of instruction fetch and execution. This is clearly demonstrated by figure 5.1. One part of the chip contains the program bus, the program ROM (1536x16 bit), the stack (4x12 bit), the program counter register and the controller. The other part of the chip contains the data bus, the data RAM (144x16 bit), the double precision 32-bit ALU/accumulator, the fast (200 ns) 16x16-bit multiplier, a barrel shifter for shifting data memory words into the accumulator, a shifter that shifts the accumulator into the data RAM and two autoincrementing/decrementing registers used for indirect data adressing and loop counting. Unlike with a straight Harvard architecture however, there is a data path between the program and the data bus, which makes a data read from program memory and an indirect subroutine call possible.

PROGRAM MEMORY

The TMS320 can be ordered in two versions: the TMS320M10 and the TMS32010. The TMS320M10 can be used in two modes: the microprocessor mode in which all program memory lies off-chip and the microcomputer mode in which part of the memory (1536 x16-bit) lies on-chip and the rest off-chip. The TMS32010 can operate only in the microprocessor mode. Both types can address a maximum amount of 4Kx16-bit words of program memory. Instructions in off-chip memory can only be executed at full speed, which means fast memories with access times of under 100 ns are required.

DATA MEMORY

Data memory is the 144x16-bit on-chip RAM. Instruction operands are fetched from this RAM. Data can be read into the RAM

from a peripheral using the IN instruction or read from program memory using the TBLR instruction. The OUT instruction will write a word from the data memory to a peripheral, while a TBLW instruction will write a word from data memory to program memory. In this case, of course, the program memory will

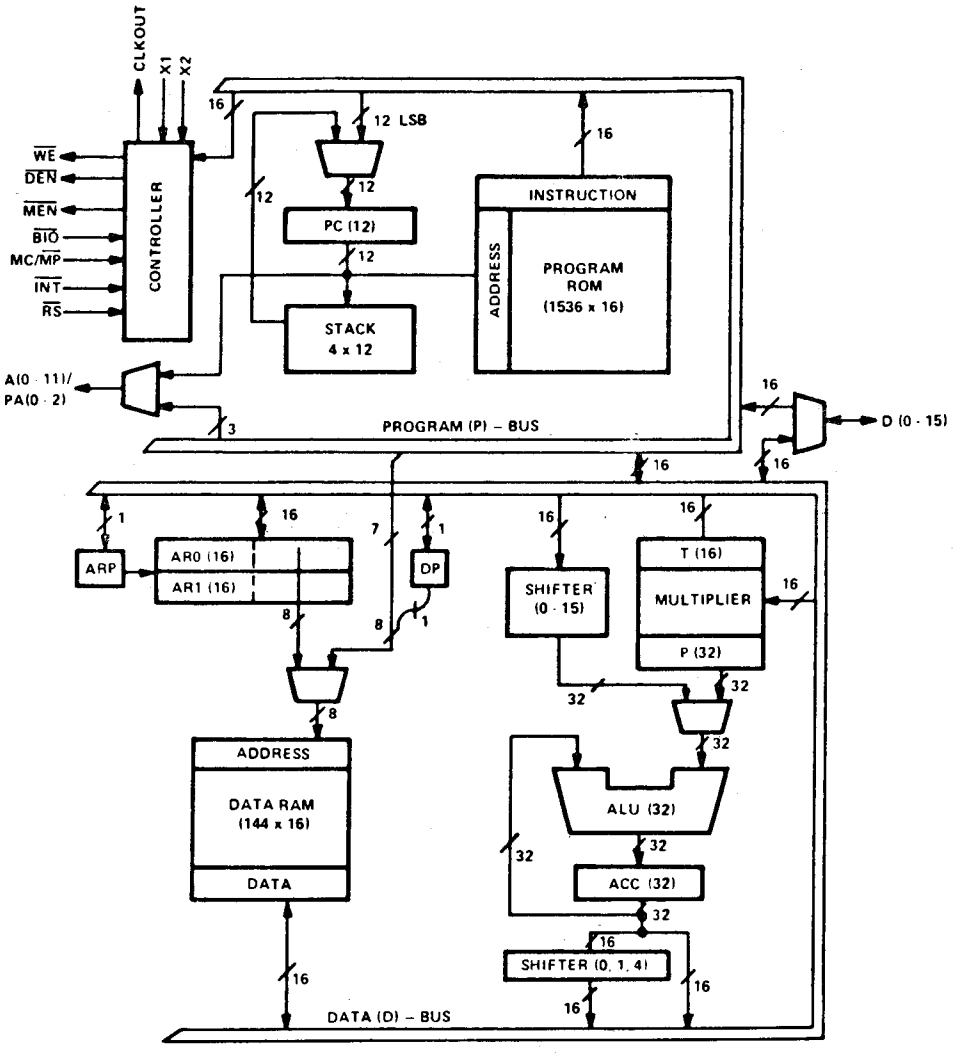


Figure 5.1: The architecture of the TMS320M10.

have to be off-chip RAM. This clearly indicates one method of extending the data memory. This method however leads to a trade-off between program memory-space and extra data memory-space. Furthermore, if frequent accesses to the extra memory are needed, it is a slow method since the TBLR and TBLW instructions take three cycles to execute. Also they use one level of the stack. With some additional hardware, the IN and OUT instructions provide another method. One could use one IO port to fill an external addressing register and one as external memory data bus. Or use two autoincrementing addressing regis-

ters for reading and writing. Since the TMS320 has 8 IO ports available, several possibilities exist. This method is faster, using the autoincrement method, since the IN and OUT instructions only take two cycles to execute.

ADDRESSING MODES

The TMS320 offers two addressing modes: direct and indirect addressing. In the direct addressing mode, the address of an operand is part of the instruction (seven bits). This address is concatenated with the so called data page pointer (DP a hardware 1-bit register) to address the full 144 words. Page zero contains 128 words and page one contains 16 words. Usually page one will contain infrequently accessed system variables. DP is part of the processor status register and can therefore be stored in data memory using the SST (store status) instruction. The indirect addressing mode uses the lower 8 bits of the auxiliary registers as an address for a data operand. The auxiliary register in question is pointed to by the auxiliary register pointer (ARP) which is part of the status register. The auxiliary registers can be automatically incremented because the lower 9 bits of the register form a circular counter. This feature also facilitates using the auxiliary registers for loop control using the BANZ (Branch on Auxiliary register Non Zero) instruction. This instruction also decrements the current auxiliary register. As the auxiliary registers are 16 bit wide, they also can be used as temporary storage registers. The auxiliary registers can be stored in and loaded from data RAM using the SAR respectively the LAR instructions.

ARITHMETIC ELEMENTS

There are four basic arithmetic elements: the ALU, the accumulator, the multiplier and the shifters. All arithmetic operations are performed using two's complement arithmetic. Most arithmetic instructions will access a word in the data RAM, either directly or indirectly, and pass it through the barrel shifter. This shifter can left-shift a word 0 to 15 bits, depending on the value specified by the instruction. The data word then enters the ALU where it is loaded into or added/subtracted from the accumulator. After a result is obtained in the accumulator, it can be stored in the data RAM. Since the accumulator is 32 bits, both halves must be stored separately. The accumulator is always destination and the primary operand. A parallel left-shifter is present at the accumulator output to aid in scaling results as they are being moved to the data RAM. The accumulator can be set in a special mode called the overflow mode. This mode is indicated by the overflow mode (OVM) 1-bit register which is part of the status word. In this mode, an overflow will cause the largest/smallest representable value of the ALU to be loaded into the accumulator. This

models the saturation processes inherent in analog systems. If an overflow occurs the OV-bit in the status word will be set. A special branch instruction (BOV) will perform a branch and reset the OV-bit on this condition. A variety of other accumulator conditions can be tested by conditional branch instructions. The multiplier consists of the T-register, the P-register and the multiplier array. In order to use the multiplier, a multiplicand must be loaded into the T-register, then a multiplication is performed with an operand from the data memory and the (32-bit wide) result is found in the P-register. The product can then be loaded into, added to, or subtracted from the accumulator. The contents of the P-register cannot be restored without altering other registers. Therefore, an interrupt will be inhibited until the instruction following a multiplication has been executed. This then will have to be an instruction which combines the P-register with the accumulator.

SPECIAL INSTRUCTIONS

Already mentioned are the BOV and BANZ instructions. In addition to these two, there are two instructions that make pipelined multiply and accumulate operations at 400 ns rates possible: the LTA (Load T-register and Add P-register to the accumulator) and the LTD (Load T-register, shift indirectly addressed data location one address further and add P-register to the accumulator) instructions. Next, there are an instruction that shifts data words in memory one location ahead (DMOV; an operation also performed by the LTD instruction), an instruction that performs a branch when the signal on the BIO testpin is zero (BIOZ) and an instruction that uses the 12 lower order bits of the accumulator for a subroutine call address (CALA).

DRAWBACKS

In the instruction set one misses rotate instructions for the accumulator contents, in and out instructions for the accumulator contents and relative branch instructions. Since there are only three memory strobes available that are mutually exclusive, TBLW and OUT use the same memory strobe. This means that, if one wants to use both instructions, one has to decode the full memory address to detect on the outside the difference between the two instructions (bit 3 to 11 are zero means TBLW). There is no difference at all when memory locations 0 to 7 are addressed, so TBLW cannot be used on those locations. The built-in overflow mechanism only works with additions. This means that, since one often has to multiply and use numbers of different accuracy and therefore has to shift after accumulation, one still has to add extra instructions to prevent an overflow resulting in sign reversal.

5.2 Other important components

As was mentioned before, a 12 bit wordlength was chosen for the converters. Upper limits were put to the conversion time: 10 microseconds for the ADC (inclusively stabilizing time for the sample-and-hold amplifier) and 1 microsecond for the DAC. This gives a total delay for conversions of 11 microseconds. If we take a sampling rate of 16 KHz (ten times the required bandwidth of the focus servo), the sampling gives a delay of 31.25 microseconds (half the sampletime). Of course the calculation of the output force also takes time. If we tolerate at 1600 Hz a phase shift of 35 degrees, we can afford a maximum delay of 60.8 microseconds. This leaves for the calculation a maximum of 18.55 microseconds, which corresponds with a total of 92 instruction cycles (executed at a rate of 5 M instructions per second). This should be sufficient since most instructions of the TMS320 take only one cycle to execute. Furthermore, all major components had to be microprocessor compatible. Since the components had to be obtained quickly, a selection was made out of components that were in stock by the dealers.

THE ANALOG-TO-DIGITAL CONVERTER

Chosen was the HI-5712 ADC from Harris Electronics. This converter of the successive approximation type, features an 8 microseconds conversion time for a 12 bit conversion controlled by an internal or external clock. 10 Volts and 20 volts full scale input levels are possible, and a +10V precision reference is available. The data can be output in 2's complement or binairy output code, in a resolution of 6,8,10 or 12 bits. In addition to a serial output line, a parallel, 12 bits wide data path is available.

THE DIGITAL-TO-ANALOG CONVERTER

Chosen was the MP7622 DAC from Micro Power Systems. This converter is a 12 bit monolithic multiplying DAC. The MP7622 accepts AC or DC reference voltages and multiplies in all four quadrants. It has three 4-bits input registers, all individually addressable. A 12-bit DAC register follows the input registers so the DAC is double buffered and digital feed-through is reduced to a minimum. Note: the DAC register cannot be disabled by means of the chip-enable line so it has to be filled with a zero value explicitly. Low output capacitance allows the MP7622 to achieve very fast settling times: less than one microsecond for a full scale response to 0.01% when utilizing a high speed output amplifier.

THE SAMPLE-AND-HOLD AMPLIFIER

Chosen was the HA-2425 fast sample-and-hold amplifier of Harris Electronics. It is a monolithic circuit consisting of a high performance operational amplifier with its output in series with an ultra-low leakage analog switch and a MOSFET input unity gain amplifier. The circuit features a acquisition time of 5 microseconds (time between hold to sample/track transition and valid output voltage) and an aperture time of 30 nanoseconds (time between sample/track to hold transition and valid output voltage) and has a TTL compatible control input.

THE ANALOG MULTIPLEXER

The need for a multiplexer is obvious since we want to replace more than one servo system and thus want to sample more than one error signal. Chosen was the HI-508A 8 channel CMOS analog multiplexer from Harris Electronics. It has TTL/CMOS compatible multiplexed address and enable lines and features break-before-make switching with an access time of 500 nanoseconds.

6. THE DEVELOPMENT ENVIRONMENT

The environment in which the digital servo controller was developed, and by environment I mean the tools available to assist in developing the prototype, was formed by two pieces of equipment and a communications program written by myself. The first piece of equipment was the evaluation module, available for the TMS320, the second a P2500 personal computer, performing the task of an intelligent terminal among other things. An overview of the development environment is given in figure 6.1. In the following paragraphs, all items of the environment will be discussed. For detailed information on the equipment, the reader is referred to (9,10,11,12). A listing of the communications program can be found in APPENDIX IV.

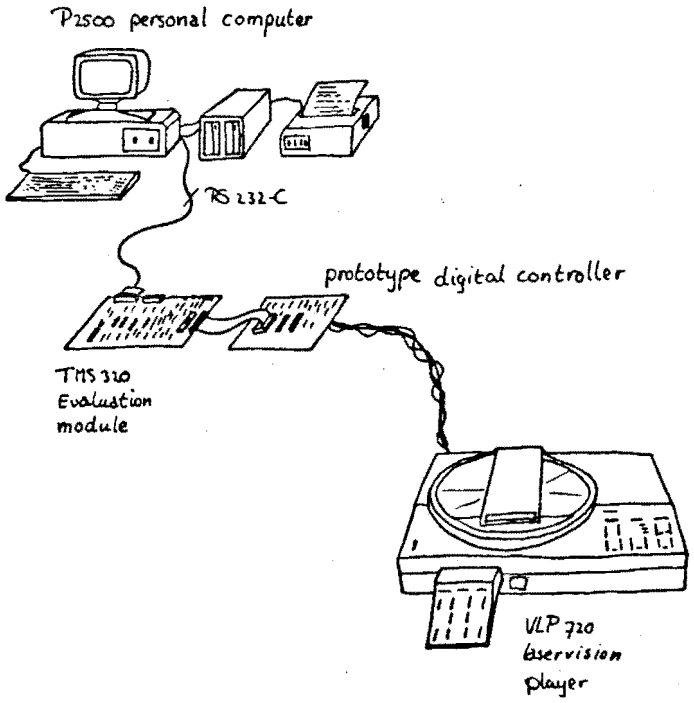


Figure 6.1: The development environment (with one player).

6.1 The TMS320-EVM Evaluation Module

The Evaluation Module (EVM) is a single board development system for the TMS320 digital signal processor. It can be used to edit programs, assemble them to executable code, debug the programs 'in circuit' and load them in an Eprom.

The board is controlled by a TMS9995 16-bits microprocessor which can communicate with the outside world through two RS-

232C serial ports. One of the ports is to be used for a terminal, the other for a host computer or a line printer. Furthermore, there is a connection on-board for an audio cassette recorder, which can be used to store files. The TMS9995 stores the assembled code in a 8 kbytes large dual-ported RAM, from which the on-board TMS32010 (the microprocessor version of the TMS320) fetches its instructions. All pins of the TMS32010 are directly connected via a 40-lead flat-cable to a connector, which can be inserted in a 40-pins dual-in-line ic-socket in the prototype. The lines are unbuffered and the flat-cable is very short to prevent delays to arise due to such a connection. Also there is a 28-pins zero-insertion-force socket on the board that can contain a TMS2764 Eprom. Such an Eprom can be programmed on-board.

Due to the fact that I wanted one of the first samples of the EVM to become available, the EVM I received was not able to let the TMS32010 execute code at its maximal speed (20 MHz clock frequency gives a 200 ns instruction cycle), but only at three quarters of that speed (15 MHz gives a 266.66 ns instruction cycle). The choice was either that, or an extra delay of three months for my graduation project. Still a 15 MHz clock frequency seemed to be fast enough. When not, the TMS 32010 itself is able to run at 20 MHz but in that case it will have to execute code out of Eprom on the prototype and debugging is no longer possible.

The operating system of the EVM (executed by the TMS9995) supports several debugging facilities: all registers and memory locations of the TMS320 can be inspected and altered, there are commands for loading and saving data- and program memory and processor state, a single step facility is provided. There is a possibility to set breakpoints for program execution, however, if for a breakpoint an event count of more than one is set, program execution is no longer in real time. This is caused by the fact that after occurrence of a breakpoint, the TMS9995 'interrogates' the TMS320 to check its registers and state. Furthermore the EVM does not contain a hardware event counter so this function has to be performed in software. The firmware contains a simple line oriented editor with which textfiles can be created, and a prom-utility with which Eproms can be programmed, verified and read. There is also a line-by-line assembler which can translate program textfiles from the RS-232 input ports or from the audio-cassette input port to TMS320 code. Symbolic labels can be used and the assembler produces a listing (to one of the RS-232 ports). Furthermore the firmware contains a patch assembler (one way to alter the stored TMS320 instruction code, the other way is to alter program memory locations on a hexa-decimal level) and a reverse assembler, with which code can be translated in instructions (mnemonics and hexa-decimal parameters).

6.2 The P2500 Personal Computer

To be able to operate the EVM, a terminal was needed. Since in that case one needs another device to store information and a terminal was not directly available but a personal computer was available it was decided to use the personal computer. For that purpose, the computer had to be provided with a serial I/O card and a program, capable of emulating a terminal as well as transferring files to and from the EVM. The program is described in the next section.

The P2500 personal computer is a Z80 based modular microcomputer. It is provided with a monitor, a separate keyboard, a dot-matrix printer and two 300 Kbyte floppy drives. It is supplied with the U.C.S.D p-system, which is very well suited for program development.

The P2500 was used to create program textfiles for the TMS320, to write and execute programs for digital filter construction, to write and execute the terminal emulation program and to write reports with. The system was also used to store TMS320 code files on floppy disk.

A drawback of the U.C.S.D. p-system is the fact that created textfiles can maximally be 16 Kbytes long. Many programs, especially when ample provided with comment, are longer than that and thus have to be split into several files.

6.3 The EVMdriver Communications Program

EVMdriver, the communications or terminal-emulating program, was set up in such a way, that it was fast to create. Thus possibly it is not the most elegant solution to the problem. However it performs its task quite well, but perhaps a trifle too slow. All operations on floppy disk are performed on a floppy disk with volume name USER1:.

The program consists of an initialisation part and a wait loop in which it waits for an input on a menu and executes a case-statement. The case-statements alternatives present the program options:

a. transparent mode

In this option, the program passes all characters raw (without translation) from the keyboard to the serial port and from the serial port to the screen. Characters are echoed by the EVM and not internally. The RS-232 handshake lines have to be operated by software, so to prevent buffer overflow, the DTR line is set false after one character is read from the RS-232 buf-

fer (because of the baud-rate of 9600, there are more than one characters in the buffer at that moment). Once the buffer is empty again, the DTR line is set true. By typing ^l, the program is set in the log-mode. In this mode the printer is set to 8 lines per inch and 16.5 characters per inch and all characters which appear on the screen are also sent to the printer. The log-mode is exit by typing ^l a second time and then the printer is reset to 6 lines per inch and 10 characters per inch. The transparent mode is exit by typing ^c.

b. log mode

The log-mode can also be set and reset from the main menu.

c. getfile

This option facilitates reading a file from floppy disk into a buffer in main memory. The program checks if volume USER1: is in drive 2 and if the requested file is present. In case of a textfile, all comment lines (starting with a '*') are omitted and all comment from column 25 is omitted. This is done to save time while sending textfiles to the EVM. After loading the file, the program automatically returns to the main menu.

d. storefile

This option facilitates saving to buffer in main memory in a diskfile. Files can be of type 'text' or 'data'. The program checks if volume USER1: is in drive 2 and if the requested file is present. If the file is already present, the program asks for a confirmation of the save. After saving the file, the program automatically returns to the main menu.

e. download

This part of the program sends the contents of the buffer to the serial port. Characters received from the serial port are sent to the screen and, if the log-mode is set, to the printer. This part of the program also contains the above described software handshaking. The download mode is exit when the either the user types ^c or the program receives the EVM monitor prompt from the serial port. This prompt is the question mark ('?').

f. upload

This part of the program sends a carriage return to the serial port as the end of a EVM monitor command. Successively all characters received from the serial port are placed in the buffer. Only typed ESC characters are transmitted to the EVM to abort the command in progress, the upload can be exit by typing ^c. The upload is automatically exit when the program

receives a prompt ('?') from the EVM.

g. quit

Entering this option causes an abort of the program.

6.4 Usage of the environment

As was mentioned above, program textfiles are created on the P2500 personal computer, using the text editor of the U.C.S.D. p-system. Successively EVMdriver is executed and in the transparent mode, the EVM assembler is invoked. Then the first program textfile is transferred from floppy to the buffer and downloaded to the EVM. After the last character of the file is sent, the user exits the download mode by typing ^c and gets the next textfile, etc. When the whole program is assembled, the resulting codefile can be uploaded using the SPM (Save Program Memory) command of the EVM monitor and the upload mode of EVMdriver. Then the buffer can be stored in a floppy diskfile. Since the assembling procedure can take a long time, especially when an assembler listing is required, it is advisable to save the codefile immediately after the assembly is finished. Also it is advisable to print out the label table, which makes it easier to find instructions or data locations in the program memory.

7. THE DESCRIPTION OF THE HARDWARE

In this chapter, the most important circuits of the prototype are described. The schematics of the hardware can be found in appendix I. For a description of the components, the reader is referred to chapter 5 and the components' data sheets.

7.1 The sample timer

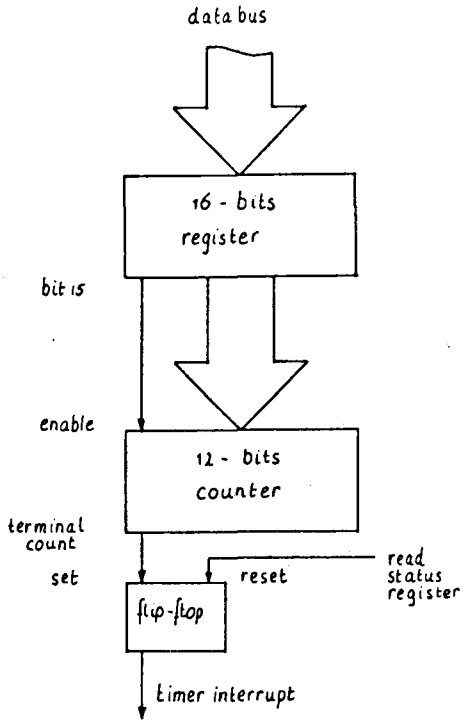


Figure 7.1: The programmable sample timer.

The sample timer is included in the prototype to be insured of an accurate sampling frequency. The timer can generate an interrupt after each sample period. The sample period can be up to 4096 instruction cycles long (with a 266.6 ns instruction cycle this gives a sampling frequency as low as 916 Hz).

The sample timer is depicted in figure 7.1 and consists of two 8-bits registers, three 4-bit synchronous counters and a D-flip-flop. The registers are loaded with a value, and this value is loaded parallel in the counters after the counters have reached a contents of all ones. At the same time the flip-flop is filled with a one. The flip-flop is cleared when the status register (see section 7.7) is read.

The registers have to be loaded with a two's complement number

indicating the number of instruction cycles that have to be counted, with a negative sign. A positive number or zero number inhibits counting. The counters count instruction-clock-cycles in an incrementing mode.

7.2 The analog input stage

The function of the analog input stage is to provide samples of the analog error signals on request and digitize those samples (quantize the samples to a number of specified levels and code the resulting level in a number of bits). To this purpose, the analog input stage contains a ADC (Analog-to-Digital-Converter) with an accuracy of 12 bits and a sample-and-hold amplifier with a unity gain amplification (see section 5.2). Due to the fact that more than one signal have to be sampled, the stage contains a 8-channel multiplexer with which one of the available signals can be selected. Since the dynamic range of the sampled signals have to match the input range of the ADC, each input is provided with its own scaling amplifier.

At the outset the analog input stage also contained an anti-aliasing filter. This filter, of course, should prevent aliasing errors caused by frequency components in the error signals of higher frequency than half the sample frequency, and thus improve the quality of the filter. However, the anti-aliasing filter introduced so much phase shifting where as the frequency components in the error signal above half the sample frequency contained so little energy, that one could hardly speak of an improvement. Therefore, the filter was removed from the analog input stage.

When the processor issues a command to take a sample and start an A-to-D conversion, the sample-and-hold amplifier is set in the hold mode. It takes some time for the amplifier to reach a stable output level and thus the start of a conversion will have to be delayed. Therefore, the command is delayed for 5 microseconds by a one-shot and then passed to the ADC. The processor can issue the command by writing a word in a register. At the same time, the processor selects the multiplexer address, enables the ADC and the DAC and starts the taking of a sample. The register is called the hardware-status-register. The bit which starts the taking of a sample is not written in a physical register but in a flip-flop. The flip-flop is reset by reading the contents of the ADC. This resets the sample-and-hold amplifier to the sampling/tracking mode and thus assures the sample-and-hold amplifier of enough time between successive holds.

A high-to-low transition on the converter status output indicates that a conversion is finished. The signal from this

output (STATUS-ADC) is inverted and used to clock a D-flip-flop, which is loaded with a 'high'. The flip-flop is reset by reading the status register and the output signal is called CONVERSION-READY.

7.3 The analog output stage

The function of the analog output stage is to convert the digitally filtered error signal to an analog signal that can be used to drive the servo actuator. To this purpose, the analog output stage contains a 12-bits DAC (Digital-to-Analog Converter, see section 5.2) and a complementary transistor power stage.

The DAC is enabled by setting a bit in the hardware-status-register. The data sent to the DAC has to be coded binairy instead of two's complement so the sign bit of the data word (bit 15) has to be inverted. The DAC supplies an output current and two operational amplifiers are needed to transform this current in a (bipolar) output voltage. Disabling the DAC doesn't turn off the output current of the DAC. This means the power amplifier has to be switched off when the DAC is disabled.

The power amplifier is constructed from a complementary transistor amplifier stage with power transistors and an operational amplifier. The amplification of the transistor stage is reduced to 2 by internal feedback and the total amplification is reduced to 1.2 by feedback to the input of the operational amplifier. The operational amplifier itself is provided with frequency dependent feedback to remove oscillations. The power amplifier is switched off by pulling the output of the operational amplifier to ground by means of a J-FET. The DAC-enable signal is compared with 2.5 Volts by an operational amplifier resulting in a signal equal to either one of the supply voltages. This signal switches the J-FET. The two zener diodes form a safeguard against an overvoltage on the input of the transistor stage.

7.4 Digital means of controlling an actuator

As was mentioned in section 4.1, the ojective, when eliminating the DAC, was to obtain a circuit to control the actuator with, that could more easily be integrated. If the actuator can be controlled using pulses, the major part of the circuit can be composed of digital electronics. Only the power output stage cannot be integrated in the same chip with the controller. With pulsed control, a coil is used, to store current in. The acceleration of the objective is proportional to this current according to the formula:

$$a = B * i * l / m$$

in which a= accelleration
B= the strength of the magnetic field
i= the current
l= the length of the current path through the field
m= the mass of the objective

Pulses can be used to increase or decrease the current in the coil more or less in the same way as the DAC does. At the outset, the idea was to use the voice coil of the objective itself for this purpose, but during testing it became clear that the pulses produced too much interference in the video circuits. Therefore, a coil was incorporated in the power output stage.

Since there was not much time left for extra calculations in software (see chapter 8), a type of pulse modulator was sought that could be transparent to the processor, i.e. the processor can address and load the modulator in the same way it loads the DAC and the overall performance of the system does not change by replacing the DAC by a modulator.

7.4.1 the duty cycle modulator (DCM)

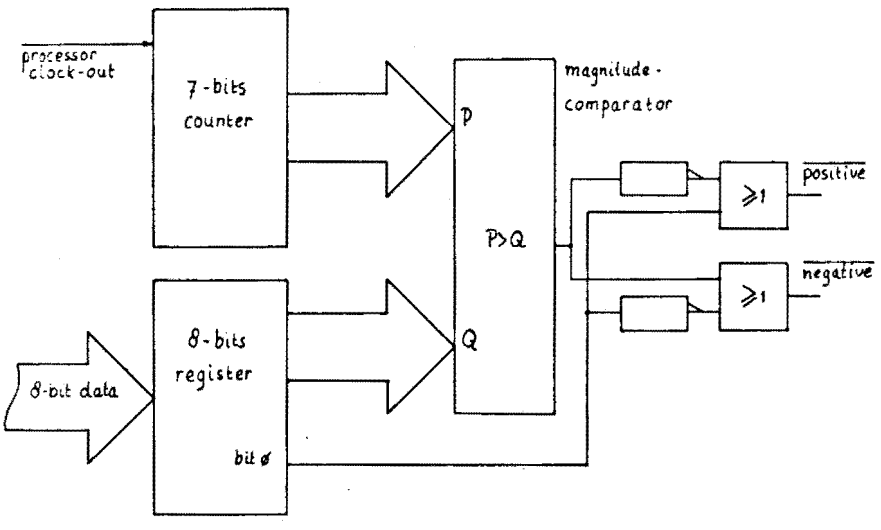


Figure 7.2: The modulator.

The first pulse modulator is depicted in figure 7.2. It produces a variation on pulse width modulation we called duty cycle modulation. The modulator produces a series of pulses of variable length and constant repetition frequency.

The modulator functions as follows: the processor can write its control force value in a register (8-bit). From these 8 bit, 7 bits are compared with the contents of a counter, msb (most significant bit) with msb, etc. The one bit left over is the sign-bit. If the sign-bit indicates a positive value and the register contents is smaller than the counter contents the 'non-positive' output is low. If the sign-bit indicates a negative value and the register contents is larger than the counter contents (due to coding of values in two's complement code) the 'non-negative' output is low. The 8 bit counter counts processor instruction cycles. This means the shortest pulse that can be made in this manner is two instructions cycles (533.3 nanoseconds) long. The 7 msb's of the counter are used for comparison. If the 7 lsb's (least significant bits) of the counter are used, the shortest pulse is 266.6 nanoseconds long, but this pulse is considered too short, regarding the heigth of the pulses of 12 Volts. The modulator period is thus 256 (2 to the power 8) instruction cycles long (68.3 microseconds) long. With this type of modulator it is necessary that the sampling frequency is the same as the modulator frequency (14648 Hz).

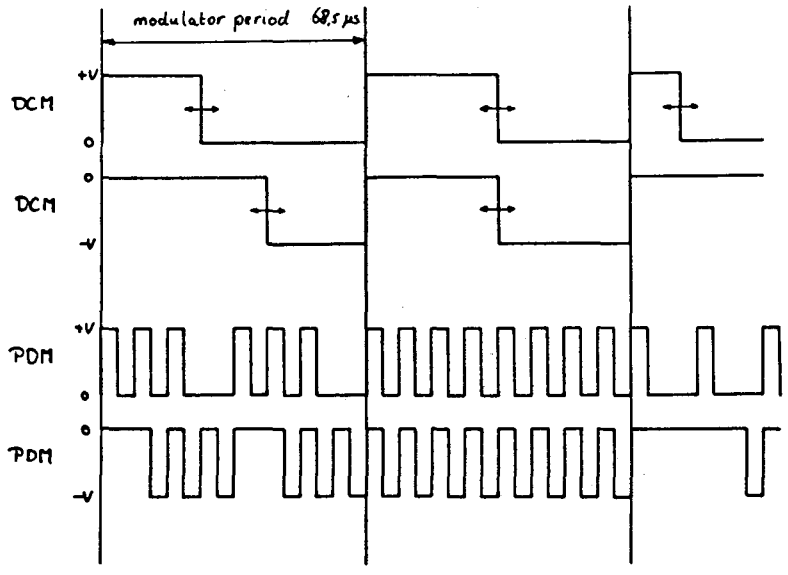


Figure 7.3: The effects of DCM and PDM for positive and negative signals.

The duty cycle modulator has several disadvantages, some intrinsic to the type of modulation, others which can be accounted to the modulator I made:

1. The modulator frequency is audible (14648 Hz).

2. Though after one modulation period, the current through the

coil is the same as when a DAC is used, the objective displacement is not. This effect is worsened by the large modulation period compared with the system time-constants.

3. Due to the two's complement coding, for negative pulses a delay is introduced, which is dependent on the pulse width and can, for the smallest pulse, amount to an entire modulator period. This effect is illustrated by figure 7.3 and can be eliminated by conversion (in soft- or hardware) of the coded value to a sign+magnitude coded value.

4. In my duty cycle modulator, the start of the modulator period is not coupled with the loading of the register. This also produces a delay, dependent on the value written and the moment the value is written relative to the start of the modulator period. This effect is illustrated by figure 7.4 and can be eliminated by coupling the loading of the register with the start of the modulation period.

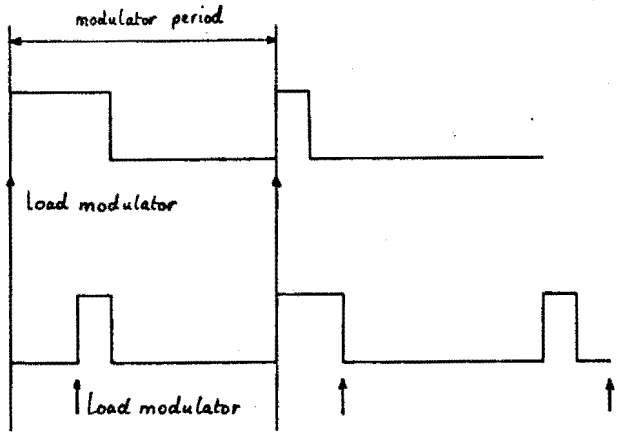


Figure 7.4: The DCM with different load moments.

The effect of all above stated disadvantages can be weakened by decreasing the modulation period. Unfortunately, since the smallest pulse length is fixed, this results in a reduction of output word-length.

7.4.2 the pulse density modulator (PDM)

To overcome the intrinsic disadvantages of the DCM, I changed the circuit of the DCM described in 7.4.1 in such a manner that it evenly distributes pulses (of the smallest length) over the modulator period. I called this modulator a pulse-density modulator. The only change made to the modulator in figure 7.2 was that the 7 bits of the register are compared with the 7 bits of the counter msb with lsb, etc. The resulting effect is compared to the DCM effect in figure 7.3.

The disadvantages of the duty-cycle modulator are present in this modulator but the effects are not noticeable. The PDM has as an advantage over the DCM that the sampling frequency does not necessarily have to be coupled to the modulator frequency.

In the prototype, the PDM makes use of the same counter as the DCM does. The modulator period, therefore will be the same for both modulators.

7.4.3 the power output stage for the modulators

As was mentioned before, from the outset, I wanted to use the objective coil to integrate the pulses. Therefore, the output stage had to produce positive and negative pulses of 12 Volts high and minimally 533.3 nanoseconds long. Especially in case of the PDM, the pulses are very short, where as the current through the coil can be high (maximally 1 Ampere). For this reason, I used HEXFETs as switching elements of which the switching speed is independent of the current through the device. The HEXFET is a special type of power-MOSFET fabricated by International Rectifier. Unfortunately only n-type HEXFETs were available to me so the output stage could not be a straightforward complementary amplifier stage. The HEXFETs available can maximally endure a drain-source current of 0.5 Ampere, so in the prototype, for each HEXFET, two are put in parallel.

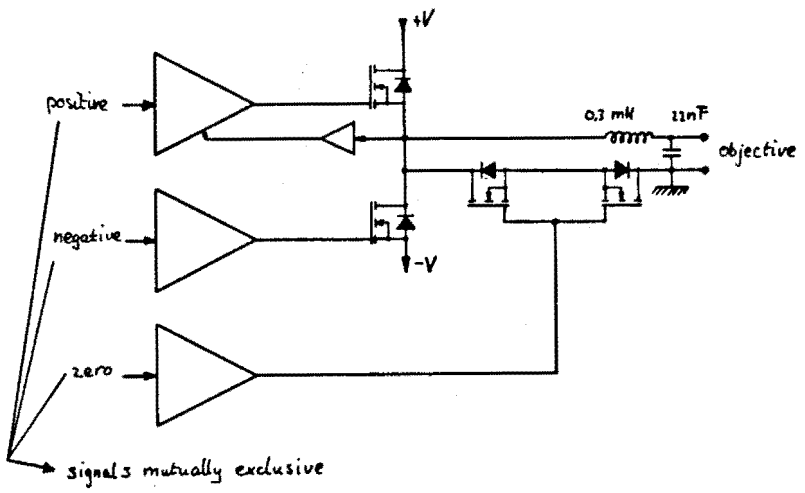


Figure 7.5: The modulator power output stage.

An important datum is that the current through the coil may only be increased or decreased by pulses. This means that in between two pulses, the current must stay the same. In traditional applications of pulsed control (high-power motor control, etc.), there is a switch from the coil to each of the

power supplies. Each switch is bridged by a diode, which with the power supplies form the so called free-wheeling circuit. When both switches are closed and a current is flowing through the coil, the voltage over the coil peaks above one of the supply voltages so the current can flow through one of the diodes through the power supply back to the coil. However, this method has two disadvantages: one, the peaking of the coil voltage produces a lot of interference and two, it reduces the current considerably. Using short pulses, especially in case of the PDM, the current will never become strong enough. It will, in the periods between pulses, be reduced to zero. This is not an acceptable situation. To overcome this problem, I had to add a bidirectional switch that can short-circuit the coil. This switch forms an actively controlled low-impedance free-wheeling circuit. Of course, care had to be taken, to insure that the signals controlling the switches were mutually exclusive.

The resulting output stage is depicted in figure 7.5. HEXFET A switches the coil to the positive supply, HEXFET B to the negative supply. HEXFETs C and D form the bidirectional switch (the HEXFET conducts from drain to source and the built-in free-wheeling diode from source to drain). When the gate-source voltage is above +4 Volts, the HEXFETs begin to conduct, but a large current can flow above +7 Volts. The HEXFETs power supplies are +12 and -12 Volts. It is sufficient to switch the gate of HEXFET B between -12 and -5 Volt, and the gates of HEXFETs C and D between +12 and -6 Volts. The gate of HEXFET A has to be switched from 0 to +19 Volts, but when HEXFET B opens, the source-voltage of A becomes negative. An extra circuit has to be added to make sure that the gate-voltage also becomes negative when B opens. To switch the gate-voltages of the HEXFETs, three transistor driver stages were added. In all cases BSX20 p-n-p switching transistors were used. The stages for the HEXFETs B and C-D were provided with level shifters to be able to switch negative voltages from TTL input signals. All three stages were fitted with an emitter follower transistor, to be able to drive the 140 nF input capacity per (double) HEXFET. The driver stage for HEXFET A switches the gate between 0 and +19 Volts, for HEXFET B between -15 and 0 Volts and for the HEXFETs C and D between -15 and +19 Volts.

7.5 The player interface

The player interface comprises a number of circuits. Their function is to scale several player signals to TTL levels, detect transitions on some of them and make the signals attainable for the processor.

The FPI (Focus Position Indication) signal must be monitored.

This signal is switched between -12 and +12 Volts. A simple one-transistor inverter with clamping diode reduces this to +5 and 0.7 Volts. The circuit is present in duplicate (FPI0 and FPI1) since the prototype must perform two focus servos.

Two SCAN signals (in the player documentation called OBS) must be monitored. During normal play, this signal is 0 Volt, and during the scanning of a player +12 Volts. However, during special playing modes the signal varies between 0 and +2.5 Volts. With help of an operational amplifier, the decision level (high-low) is placed at +5 Volts and the signal is transformed in: +15 Volts during scan and -15 Volts during non-scan. The simple one-transistor inverter with clamping diode transformes these levels to 0 Volts during scan and +5 Volts during non-scan. The output signals are called SCAN0 and SCAN1.

The circuit described in the previous paragraph, is also used to transform the T0 and T1 signal (the tacho-signals produced by the tacho-generators on the spinning motors of both players). With one difference: the decision level high- low is placed at 0 Volts. In addition to this transformation, the signals derived in this way is used to clock a D-flip-flop. Thus, on a low-to-high transition, a high is clocked in the flip-flop. This makes it possible for the processor to detect a high-to-low transition on the tacho-signals and thus count the tacho-cycles (in the software description called: tacho-pulses). The output signals of the flip-flops are called TACH00 and TACH01. The flip-flop is cleared by a dummy-read of the processor. An IN-instruction on IO-port 2 will clear the TACH00 flip-flop and an IN-instruction on IO-port 3 will clear the TACH01 flip-flop.

Two switches are placed on the prototype, intended to be used for enabling the servos manually and independently of each other. These signals, ON0 and ON1, are TTL compatible.

To make the signals attainable to the processor, a 16-bit register (physically formed by two 8-bit buffers) can place the signals on the data-bus. In addition to the player signals, also the signals TIMER and CONVERSION-READY are placed on the bus. This logical register is called the status register. From the chip-enable signal of this register (called NON-READ-STATUS), the positive going edge is detected by means of a synchronous positive edge detector. It operates as follows: by means of a D-flip-flop, which is clocked by the processor instruction-clock, the NON-READ-STATUS signal is delayed by one clock-cycle. In parallel, the signal is inverted and 'or'ed with the output of the D-flip-flop. The resulting signal is active (low) after a high-to-low transition occurred on the NON-READ-STATUS signal. The positive (trailing) edge is used to clear the interrupt flip-flops.

The status word is read as follows:

BIT 0: TIMER
BIT 1: FPI0
BIT 2: TACHO0
BIT 3: ON0
BIT 4: FPI1
BIT 5: TACHO1
BIT 6: ON1
BIT 7: CONVERSION-READY
BIT 8: SCAN0
BIT 9: SCAN1

7.6 The interrupt structure

The extra hardware, necessary for the interrupt handling, is limited to an or-gate and an inverter. With this circuit, the signals TIMER and CONVERSION-READY are 'or'ed to produce the low-active processor interrupt signal (NON-INT). On an interrupt, the processor can proceed by reading the status register. The hardware status word will show which device originated the interrupt and the interrupt condition will be reset.

7.7 IO-address decoding

The IO-address is decoded from the processor address lines 0-2 by two 8-out-of-3 decoders. One of them provides all read-signals and is enabled by the processor IO-read strobe signal NON-DEN (Data ENable). The other decoder provides all write- and load-signals and is enabled by the processor IO-write strobe signal NON-WE (Write Enable) and ENIOW (ENable IO-Write). ENIOW is high when the address lines 3-11 are low. The signal is made by 'nor'ing the address lines in pairs (A11 is just inverted) and 'wired-and'ing the outputs with diodes and a pull-up resistor. ENIOW is necessary to make a difference to the hardware between an OUT-instruction and a TBLW-instruction (see subsection 5.1.3, the last paragraph).

7.8 Provisions for external memory

On the prototype, two sockets are prepared for 2764 type Eproms. These Eproms can be used as program ROM. No provisions were made for external memory for temporary storage purposes. During the last part of the project, program memory was used for temporary storage of data, but since program memory was located on the EVM, there was no need for alterations of the prototype hardware (aside from alterations to the IO-addressing circuitry, see section 7.7). It was principally a shortage of (project-) time that forced me to make use of the

EVM's program memory for storage purposes. On the other hand, it was also the most obvious method.

If a future user of the prototype would want to add external RAM, I would recommend to use some of the address space of the program memory. This method uses the least extra hardware and since there is only a small area on the circuit board left open, it might come in handy. Of course, this is feasible only when the total amount of memory required for instructions combined with the desired amount of data memory, does not exceed 4K words.

8. THE DESCRIPTION OF THE SOFTWARE

In this chapter, all aspects of the prototype software are treated. First is described how the filter was chosen and how the parameters were found. Discussed is, what is done and what could be done to optimize filter parameters. Also the principles and functioning of the predictive controller are described. Next, the different aspects of the control program for the digital servo controller are discussed.

8.1 Servo-aspects of the software

The principal function of the software is to render stability to the servo system. This necessitates a compensation filter to be made in software. Theoretical aspects of digital compensation filters are treated in section 1.4. This section treates the realization.

8.1.1 obtaining a suitable digital filter

Our starting point, when making a digital compensation filter for the focus servo, was the analog compensation filter from the LaserVision player. This is a second order lead-lag network with zeroes at 882 and 1130 Hz, and poles at 2822 and 8040 Hz. The idea behind using this filter was that it functions quite well. It thus is particularly suited to test the hardware with. At least to start with.

The different methods to translate an analog filter to a digital filter are described in subsection 3.4.2. The difference equation obtained by translation, was directly solved in software, to make full use of the efficient TMS320 instruction set. This resulted in an IIR- (Infinite Impulse Response) filter in a direct form depicted in figure 8.1. The first method I tried was the matched z-transformation. Measurements showed that the phase characteristics of the resulting filter did not resemble the original characteristics and the total system was not stable. Next, with sufficient pre-warping of the time constants, the bilinear transformation was performed. This method produced a suitable filter of which the amplitude and phase characteristics correspond very well with the analog filter.

The sampling frequency, used in the system, first was fixed on 20 kHz. Only one focus servo was implemented then. Also a second order anti-aliasing filter was integrated in the system. It had a cut-off frequency at 7 kHz. Later measurements showed that above 7 kHz no frequency components with sufficient energy were in the error signal so the anti-aliasing filter was

omitted. This greatly enlarged the phase- and gain-margin so the sampling frequency could be lowered. In section 7.4 it was mentioned that the sampling frequency should be 14648 Hz. Later on in this chapter we will show that even with this frequency, there is hardly enough time to perform all tasks for two focus servos. Without anti-aliasing filter and with a sampling frequency of 14648 Hz, the amplification factor can be set as high as the original (analog) compensator amplification while maintaining a stable system.

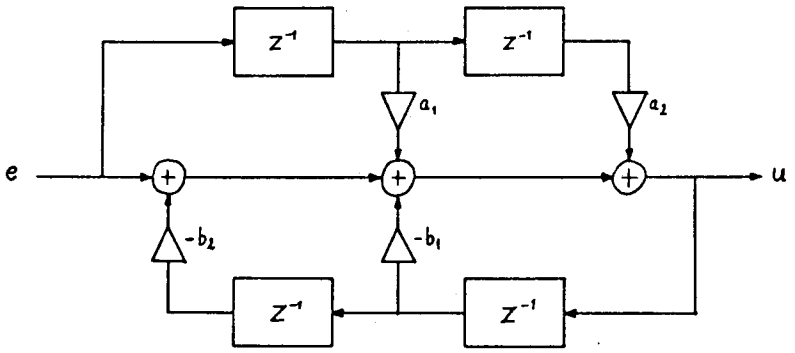


Figure 8.1: The digital filter.

Table 8.1: Filter parameters.

T / par.	1E-5	2.5E-5	5E-5	6.83E-5
P_0	4.4432	3.7654	3.2087	3.0574
P_1	-8.3229	-6.4263	-4.6551	-3.9183
P_2	3.9061	2.7409	1.6856	1.2512
Q_1	1.5852	1.0887	0.4274	-0.0108
Q_2	-0.6263	-0.1886	-0.0254	0.0350

$$u = Q1 \cdot u^{-1} + Q2 \cdot u^{-2} + P0 \cdot e + P1 \cdot e^{-1} + P2 \cdot e^{-2}$$

For the transformation itself a small computer program was written that calculates the parameters for this specific filter for several sampling frequencies and amplification factors. Table 8.1 shows several possibilities.

8.1.2 optimizing digital filter parameters

The reason for optimizing digital filter parameters is obvious. The analog filter we started from may not be ideal and even if it is, the transformation method may introduce changes in the amplitude- and phase-characteristics. The purpose can be to obtain the desired amplitude- and phase-characteristics but it can also be to obtain the desired impulse response (to respond 'ideally' to a specific type of errors).

Most methods for optimization of filter parameters depend largely on iterative computer simulation. The start out from a specific set of parameters. Then the computer program varies one or some of the parameters and observes the effect on the performance of the filter. Using a performance criterium to judge the performance by, the program iterates until the desired performance is obtained (or a certain computer calculation time limit is exceeded).

An optimization program (called OPTIMUM) was written by me of which the listing can be found in APPENDIX III. The purpose was to obtain an 'ideal' response to quadratic errors (which would induce a specified acceleration on the objective). It was said by some people on the servo and control departement that a quadratic error was a good 'worst-case' error found on discs with airbubbles imbedded in the substrate. The program simulates the servo for a short period and uses the accumulated quadrated error as a performance criterium. The program varies the parameters one at a time using the Newton-Raphson minimization method. The program did not produce filter parameters resulting in a stable filter. The non-linearities produced by the built in saturation could be a reason, or a too short simulation period. It also could be that the Newton-Raphson method should be applied to all parameters at once instead of one at a time, but whatever the reason, the results were not usable. As was mentioned before, this field of investigations was not assigned to me and I could not spend much time on it. I had to leave the program unfinished.

8.1.3 additional structures for predictive control

As was mentioned in section 4.1, focusing disturbances are highly periodic with the rotational frequency of the video disc. Since this is a priory information about the error signal, using this information in our controller design might im-

prove the performance. The theory of the control of periodic disturbances is treated extensively in (6,7).

The controller based on periodic disturbances is depicted in figure 8.2. In addition to the compensation filter, one can see in the picture a delay-line which delays the signal for one period of the disc revolutional frequency. The delay loop is a comb-filter giving extra gain for frequencies, where the phase of k(s)exp(-sT) is zero or a multiple of 2π. The frequencies where the peaks of the comb-filter occur are:

$$\omega_n = n \cdot 2\pi / T_R \qquad n = 0, 1, 2, \cdots$$

i.e. the rotational frequency of the disc and its harmonics. The reduction of the servo is given by:

$$\frac{1}{1 + P(s) \cdot R(s)} = \frac{1 - k(s) \cdot e^{-sT_R}}{(1 + P(s))\left(1 - \frac{k(s)}{1+P(s)} \cdot e^{-sT_R}\right)}$$

in which: P = transfer function of process, filter included
R = transfer function of delay-loop
k = transfer function of loop-filter
T = revolutional period of the disc

which for the peak frequencies is given by:

$$\frac{1 - k(j\omega_n)}{\left(1 + P(j\omega_n)\right)\left(1 - \frac{k(j\omega_n)}{1+P(j\omega_n)}\right)} \sim \frac{1 - k(j\omega_n)}{1 + P(j\omega_n)} \quad \text{when} \quad \left|\frac{k(j\omega_n)}{1 + P j\omega_n}\right| << 1$$

To obtain an error which is as small as possible, we choose a k(jw) as close to 1 as possible. A sufficient condition of stability then is:

$$\left|\frac{k(j\omega)}{1 + P(j\omega)}\right| < 1$$

To satisfy this equation under worst case conditions, k(s) has to be chosen low-pass. In our program there was not much time left for calculations of k(s) so it was kept as simple as possible:

$$u = \frac{15}{16} u^{-1} + \frac{1}{32} e + \frac{1}{32} e^{-1}$$

which is a first order low pass filter with a cut-off frequency at 300 Hz.

A very high burden on the software is the extra administration needed to maintain the delayline in software. A pointer has to be maintained, which points at the memory location where the processor is reading/writing at the moment. The length of the

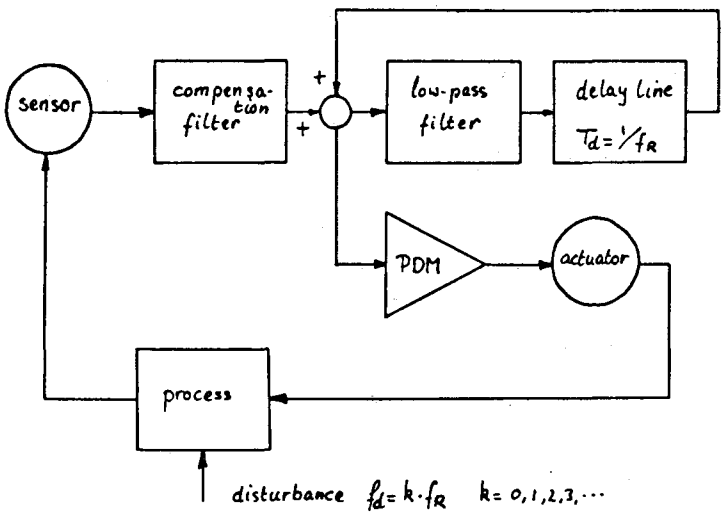


Figure 8.2: The controller based on periodic disturbances.

delayline must automatically be adapted to the revolutional speed of the disc. For this purpose, the number of samples per revolution as well as the number of tachopulses per revolution have to be counted. Once, 18 tachopulses are counted (during one revolution the delayline is filled), the counted numbers have to be cleared, the pointer must be reset, the end of the delayline must be updated, and the delay-output must be switched into the loop. This worsens the worst case number of instructions needed per sampling period. For both focus servos, separate delaylines are needed. There is not enough memory available to store each sample in a separate location when CLV-type discs are used so two samples must be stored in one location. This means at alternative sample times, two values are written in the delayline and two values are read from the delayline, and one must test for odd and even turns. Also two values must be assembled into one word and afterwards be disassembled again.

8.2 The control program for the prototype

In this section the assembler program for the TMS320 is described, that controls the prototype and performs two focus servos independently of each other with the in subsection 8.1.3 mentioned delayline management included. The program may

seem to be a little spaghetty like and unnecessary complex. However, one must remember that only 128 instruction cycles per sample period per process were available. This time constraint forced me to use every trick available to save instructions. So at some places pieces of code are duplicated to save time. Both processes and filter routines are, but for the memory locations used, exactly alike.

8.2.1 the principle of handling two tasks

There are two methods of handling two different real-time controller tasks by one processor. Let us call them the asynchronous method and the synchronous method. With the asynchronous method, execution time is granted to a process 'on demand', e.g. when external hardware indicates that action should be taken (by means of an interrupt) control is passed to the specific process and perhaps an other process is temporarily interrupted. With the synchronous method, execution time is cut in time-slices and each process is granted one time-slice (or a fixed number of time-slices) in a round-robin fashion.

From the requirements stated in subsection 4.2.3 it follows that the synchronous method is the most suitable method. In our case, we have two tasks of equal importance, i.e. for both servos performing the compensation filter. These tasks have to be performed on specified time intervals, which are (in this case) for both tasks of equal length. Now it also is obvious to give the sample timer, of which the former function was to indicate a sample moment to the processor, the new function of indicating the beginning of a new time-slice to the processor. This could be done on a polled basis but this costs extra instructions, so it is done on interrupt basis. The same goes for the indication of the end of a conversion cycle.

In the software a status register is maintained with flags that indicate the state of the processes. A process can be RUNNING or NOT-RUNNING which means the process is performing the task of the compensation filter respectively starting-up the servo (or waiting for the ON-switch to be flipped). For each process there is a flag indicating a conversion in progress (this flag is valid only when the process is NOT-RUNNING) called CONVERTING. Also there is a flag indicating the process ACTIVE in the present time-slice. The bits in the software status register:

BIT 0: CONVERTING0
BIT 1: not-used
BIT 2: RUNNING0
BIT 3: not-used
BIT 4: CONVERTING1
BIT 5: not-used

BIT 6: RUNNING1
BIT 7: not-used
BIT 8: ACTIVE (process-bit)

Furthermore there is a delay-status register with which the delaylines are controlled. A delayline can be ACTIVE, which means write operations are performed and RUNNING, which means read operations are performed. A delayline is set ACTIVE when a player is not scanning and a tachopulse is detected. Then after one revolution, the delayline is set RUNNING. The delay-status register:

BIT 0: DELAY0-ACTIVE
BIT 1: DELAY0-RUNNING
BIT 2: DELAY1-ACTIVE
BIT 3: DELAY1-RUNNING

8.2.2 initialization

This part of the program loads the data memory of the TMS320 with values from the program memory, such as filter parameters, masks, count-values,etc. The overflow mode of the processor is set. Furthermore, the sample timer is loaded with an appropriate value and all flip-flops in the hardware are reset by dummy reads to IO-ports 0 to 3. Finally a branch is performed to the start of process 0.

8.2.3 the interrupt handlers

There are two possible interrupt causes: the end of a timer period and the end of a conversion cycle. On an interrupt, the processor checks the NON-BIO pin, to which the timer output (active low) is connected, with a BIOZ instruction (see section 5.2). The BIOZ instruction performs a branch to the timer interrupt handler.

On a conversion interrupt, the processor inputs the converted value, checks which process is ACTIVE and for that process it checks if the process is RUNNING. If the active process is running the processor branches to the appropriate filter routine (which can thus be seen as a part of the conversion interrupt handler), if not the processor resets the CONVERTING flag for the active process, enables interrupts and performs a return. The filter routine is described later on in this chapter.

On a timer interrupt, the processor reads the hardware status register, saves the accumulator, performes a process switch, starts a conversion, restores the accumulator and performs a return. The process switch is done by swapping the stack-value

underneath TOS (Top Of Stack) with a stored value. This means that, after return from the timer interrupt handler, control is passed to the routine that had control before the interrupt was acknowledged. This routine then will not pass control to the process from which it was called. This process switching will be illustrated in subsection 8.2.6. The processor starts a conversion by writing a memory location (called CONV) to IO-port 0 with bit 6 set. The same word also specifies the multiplexer channel (see section 7.2). The processes alter the value in CONV so that on some times a conversion is started by the timer interrupt handler and on other times not. Each time the proper multiplexer channel is selected. The exact location in the code of the conversion start is important. The reason why will be explained in subsection 8.2.6.

8.2.4 the focus servo process

The focus servo process consist of three parts: the start/restart section, the test loop and the timer wait routine.

The timer wait routine is called by the process during the start/restart section for timing purposes. It sets the multiplexer channel and the converting value (CONV, see 8.2.3) ready for the other process. Then it sets the auxiliary register pointer to 0 (this is expected during the filter routine) and saves the auxiliary registers. Then it waits in a loop until the process bit (ACTIVE in the software status register) has been flipped. This means that a timer interrupt has occurred and the process is switched. Then the timer wait routine performs a return (to the other process).

The function of the start/restart section is to bring the objective in the focusing position and to pull the objective down when something goes wrong. The restart part resets the RUNNING flag and the conversion capability for the process. Then it pulls the objective down with half the maximal energy for 10 sample periods. This is timed with calls to the timer wait subroutine. Note: these are 20 timer periods! This is so because the timer interrupt handler switches the process. After the 10 sample periods, the process waits approximately 1 second. After each call to the timer wait routine, the auxiliary registers are restored (they might be changed by the other process). Then the process continues with the start section. There, the process waits in a timed loop until the ON-switch is flipped. If this has happened, all delayline variables are reset. Also the delayline status is reset and the compensation filter is initialized. Then the objective is kicked up (with half the maximal energy) for 10 sample periods. Successively a float-voltage is placed on the objective and the process waits with a time-limit of approximately 1 second until FPI becomes true. If the time-limit expires, the

program comes in an endless loop. If FPI becomes true, the converting capability for that process is set and the timer wait routine is called. The timer interrupt handler then will start a conversion. In a (non-timed) loop the CONVERTING flag is polled to see if a converter interrupt was handled (the handler will also read the sensor value). If in the mean time, FPI has become false, the process is restarted. If the sensor value stays under a certain value, another conversion is started, and so on. If not, the RUNNING flag is set and the timer wait routine is called. The timer interrupt routine will start another conversion. The process then enters the test loop.

The test loop is a loop in which non-time-critical hardware variables are tested and the necessary action is taken. The loop will be interrupted (at specified places) by converter interrupts, which will cause execution of the filter routine (because the RUNNING flag is set). The hardware variables are tested through flags in a memory location in which the hardware status register is read by the timer interrupt handler. The ON-switch and FPI are tested and if one of them is false, the process is restarted. If the player is scanning and if the tacho flip-flop is set, the number of tachopulses is incremented, the tacho flip-flop is reset and the delayline is set ACTIVE. If the player is not scanning, the delayline variables and status are reset and the multiplication factor (amplification factor) is changed (this is to have a different amplification during both modes DELAY-ON/DELAY-OFF).

8.2.5 the focus filter routine

The focus filter routine is actually a part of the converter interrupt handler and is executed when the process is RUNNING.

First the multiplexer channel is set for the other process and so is the conversion value (CONV). Then the difference equation of the compensator filter is calculated. The result must be multiplied with a factor. To prevent overflow causing sign reversal from happening, an equivalent value is first added to and then subtracted from the result (or vice versa when the result is negative). A value from the delayline is added (the lower byte on even turns and the high byte on odd turns; both are zero when delay is not RUNNING) and the result is output to the modulator. If the delay is not ACTIVE, the routine waits for the process bit to change and then exits the routine. If the delay is ACTIVE the output value is used as an input to the difference equation of a first order low-pass filter (with binairy multiplication factors):

$$u = \frac{15}{16}u^{-1} + \frac{1}{32}e + \frac{1}{32}e^{-1}$$

and the result is quantized to an 8-bits resolution. Then, separate actions are taken on even and odd turns. On even turns, the value is inserted as the high byte in the word to be written in the delayline. If the delay is RUNNING, the multiplication factor (for the compensation filter) during delay is set, a word is read from the delayline at the pointer, and the word is disassembled in a high byte and a low byte. Then the number of samples is incremented and the routine is exit. On odd turns, the value is inserted as the low byte in the word to be written in the delayline. The word is written at the pointer location and the pointer is incremented. If the number of tachopulses equals 18, the number of tachopulses and the number of samples are cleared, the pointer is reset to the beginning of the delayline and the delay is set RUNNING. Then the routine is exit.

8.2.6 the coherence of the program

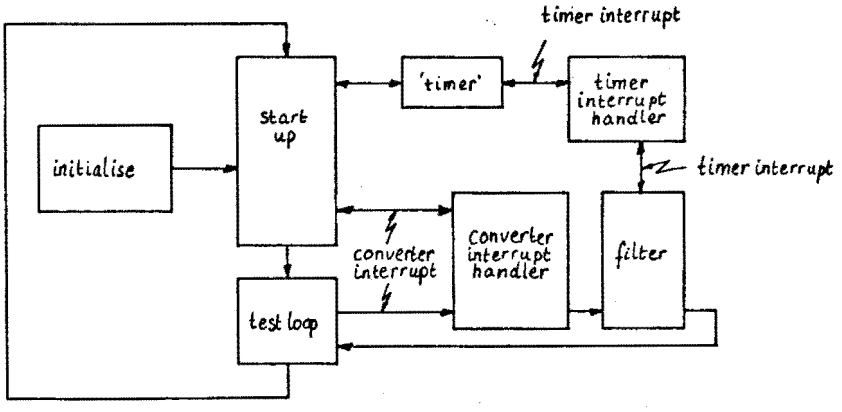


Figure 8.3: Schematic of program (only one process shown).

With one process, the program should not be difficult to understand. Referring to figure 8.3, from initialization, the processor executes first the startup section and then the testloop. From the startup section, the timer wait routine is called (for timing and synchronization purposes) and after a timer interrupt, control is transferred back to the process. When the startup section is interrupted by the converter, control is transferred back to the process directly. However, when the test loop is interrupted by the converter, control is

that timer interrupts always occur during execution of a sub- or interrupt routine! The timer interrupt handler thus can be called a level 2 routine. When during the timer interrupt routine the stack is manipulated, on the sub-TOS level, the level 1 routines will return to the swapped process. This is also illustrated by figure 8.4, which shows two processes in the RUNNING state. It also shows how little time is left. Per process, 128 instruction cycles are available. Worst case, 122 cycles are needed.The only margin left is in the execution time of the processes (test loops).

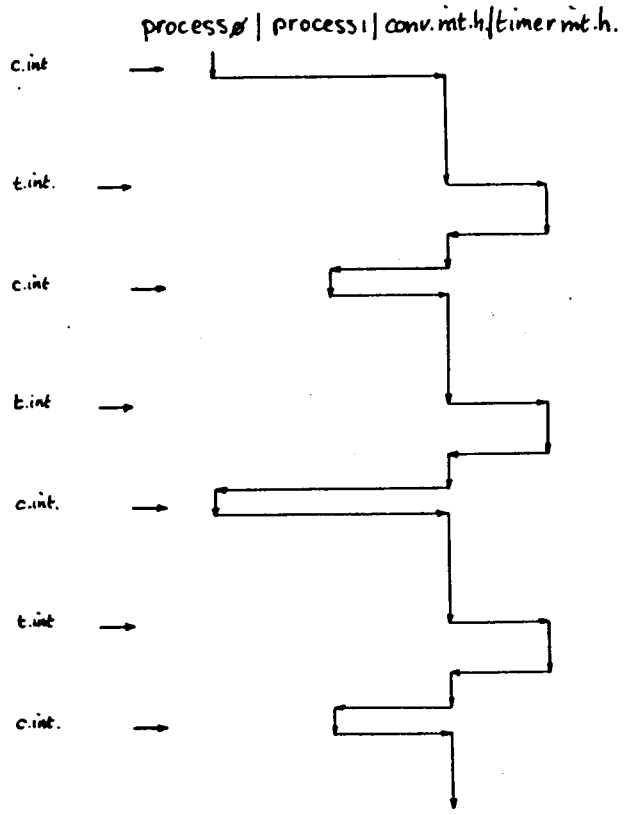


Figure 8.4: Time diagram of two processes running.

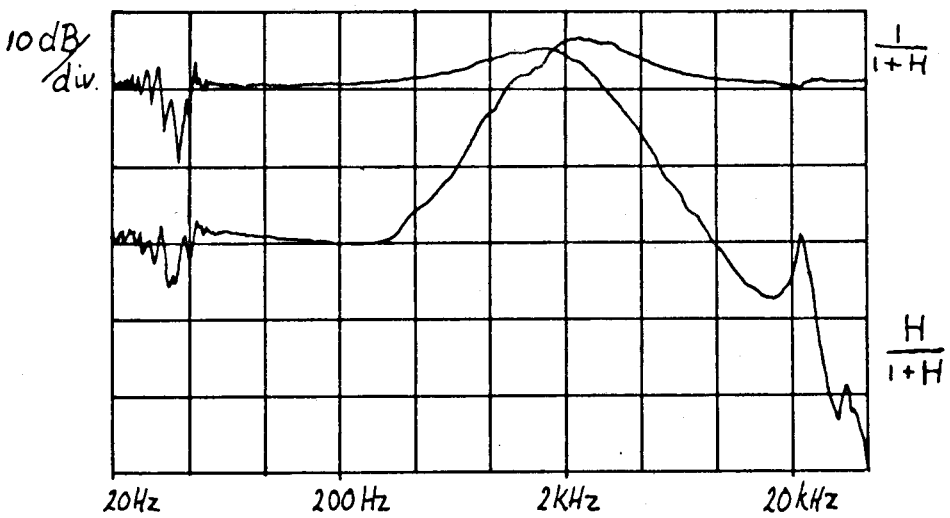
Figure 8.4 also indicates another problem: the interrupts are caused by each other. The timer interrupt handler starts a conversion which causes a conversion interrupt. This conversion interrupt causes execution of the filter routine (when the process is running) but during part of the filter routine interrupts are disabled (because the filter calculation may not be disturbed). Also during the filter routine, a timer interrupt must be handled. Now, if the timer interrupt occurs during the time in which interrupts are disabled, the interrupts is delayed. Then the start of the next conversion is delayed, etc. So this condition must be avoided. This is done

by determining when the conversion should be started and that is why the location of the conversion start in the timer interrupt handler is extremely important. Note: during normal operations a disabling of the interrupts during the filter routine would then be superfluous; the disabling was built in for testing purposes.

9. CONCLUSIONS AND RECOMMENDATIONS

In this chapter, the obtained results are presented, some conclusions are stated and my recommendations are given.

9.1 Results



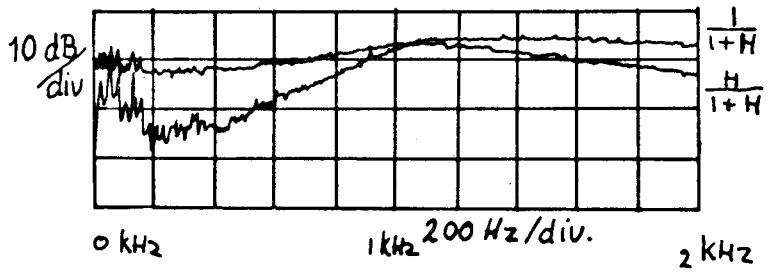


Figure 9.1.a: Bandwidth measurement of the analog focus servo (on a logarithmic scale)
Figure 9.1.b: Bandwidth measurement of the digital focus servo (on a linear scale)

With respect to the original focus servo, the servo bandwidth was not high enough. Both servos used the same amplification factor. The bandwidth is defined as the frequency at which the modulus of the closed loop transfer function equals one. Figure 9.1.a shows H/(1+H) (the closed loop transfer function; H is the open loop transfer function) and 1/(1+H) of the original servo. Figure 9.1.b shows the same functions of the digital servo, operating with a 14648 Hz sampling frequency. The reason for this discrepancy must be sought in a not very accurate translation of the original compensation filter and the influence of the extra delay caused by the sampling frequency (34.1 microseconds causes at 1 kHz a 12 degrees phaseshift).

The territory of the wordlength is largely left unexplored. The input wordlength was at all times kept at 12 bits. The delayline wordlength was taken 8 bits and the modulator wordlenght was also 8 bit from the beginning. The output wordlength was, by means of experiment, reduced: the servo still was able to focus when the output wordlength was 3 bits. However, the error signal became somewhat erratic.

From the beginning, the sampling frequency was set at 20 kHz. Then, on behalf of the DCM (Duty Cycle Modulator) and the second process, it was reduced to 14648 Hz (see section 7.4). At that sampling frequency no troublesome effects of aliasing could be detected. Figure 9.2 depicts the energy spectrum of the error signal. It shows that the sampling frequency can be lowered somewhat more before aliasing becomes a serious problem. However, the stability poses a more stringent demand on the sampling frequency. I did not search for the lower limits on the sampling frequency. Figure 9.2 shows that the energy of the error signal is nearly constant over a large frequency range (due to birefringence caused by the laser and spikes coused by the power supply). The aliasing errors caused by this can be better followed by the objective but they are less amplified (than with the non-sampled system). Also the contribution to the sound-level by aliasing errors is not clear. They could be deminished by e.g. incorporating a mediating mechanism in the sampler in stead of using a low-pass filter.

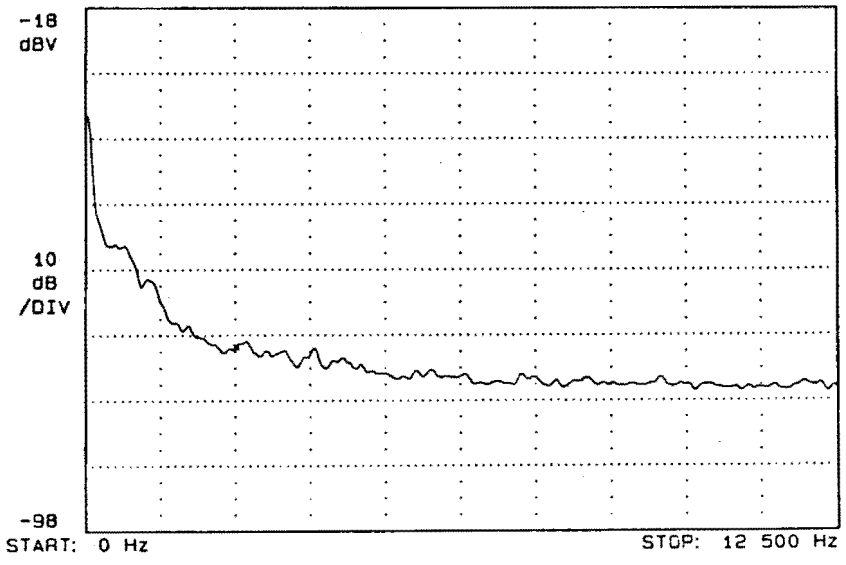


Figure 9.2: Energy spectrum of the focus error signal.

As was explained in section 7.4, the duty cycle modulator does not perform well. One problem is intrinsic to duty cycle modu-

lation and two other problems are caused by the specific realization. The resulting servo is still stable, but the gain must be taken half of the gain from the servo with pulse density modulator or DAC. As a result from this poor stability, addition of the delayline to the servo to make the predictive controller had an opposite effect: the stability became even poorer. Between operation with a DAC or the pulse density modulator, no difference could be detected. The PDM was used at both 20 kHz and 14648 Hz sampling frequencies. In both cases it performed excellent and the modulator frequency was 14648 Hz.

The addition of the predictive controller produced on the open loop transfer function the desired effect. In figure 9.3, the peaks of the comb-filter can be seen. Unfortunately we had to give the low-pass filter a very low cut-off frequency. This was done because there was only time enough to make it a first order filter. If the transition band of the filter could be smaller, then the effect of the controller could be stronger on higher frequencies. The reduction of the controller on low frequencies is 20 dB better than with the non-predictive controller. This is illustrated by figure 9.4 which shows 1/(1+H) (with H the open-loop transfer function of the predictive controller).

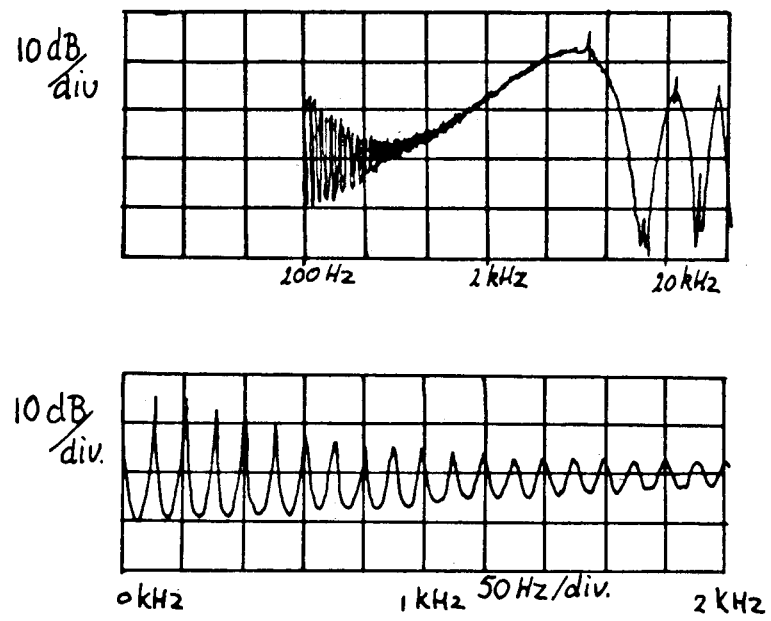


Figure 9.3: The open-loop transfer function (H) of the predictive controller.

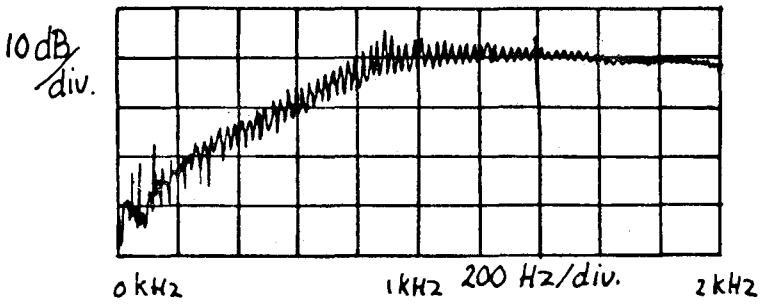


Figure 9.4: 1/(1+H) of the predictive controller.

The reduction of the non-predictive controller is poor for low frequencies. This is caused by the small number of input levels per millivolt. A total of 4096 input levels are provided by the 12-bit ADC but the input range is large to be able to start-up the servo well. If a small number of levels are available, a relatively slowly varying input signal will cause infrequent level changes. This degrades the differentiating effect of the compensation filter considerably. This is why the response is poor for low frequencies.

9.2 Conclusions

One of the major topics on which a conclusion should be stated is whether digital servo systems is meaningful for LaserVision. One could give a positive answer on this question when:

1. more than one servo loops can be integrated in one ic with control included. This means, aside from the compensation filter, also the sequence logic and control for special playing modes in one integrated circuit. The servos would have to be integrated in a dedicated ic. Dedicated ic's can provide fast arithmetic for the filters as well as dedicated circuitry for control and guarding functions. One could also think of a semi-customized ic on which a fast processor is located, flash-converters for A-to-D conversion and dedicated hardware for guarding functions, pulse modulators, etc. A dedicated ic would have to be programmable to some extent, to be able to adjust the ic to changes in the servo hardware. Bottle-necks when realizing such an ic probably will be the number of IO-pins required and the chip-area needed (probably over 20 square millimeters).

2. adaptive or learning systems are desired. These systems can provide a better performance, or with equal performance reduce the (auditive) noise level. Also these systems can be used to adjust themselves to the properties of e.g. a LaserVision deck (which can due to production circumstances be variable). This last quality would reduce the production costs, because

the servo (and the deck) no longer needs to be tuned.

3. cross-talk between the different servos is a serious problem. A multiple-input-multiple-output controller could compensate for this cross-talk. Such a controller is more easily developed, simulated and is cheaper when it is a digital controller.

The TMS320 is a suitable processor for a demonstration/try-out system. But for an application of a servo controller it is not suited at all. Too many additional components are needed. Being as fast as it is, it is not fast enough to perform all three servos while the accuracy is far too high. Thus it makes a digital servo controller too expensive for LaserVision purposes.

A third servo process is not implementable in the present set up. If the sampling frequency of the focus servo could be lowered and the radial servo could run on the same sampling frequency, and if the processor could run on 20 MHz, then there would be room for the tangential servo. This servo then, would also have to run on the same sampling frequency as the focus servo does. This still leaves us with the slide drive and the spinning motor drive. Still, I strongly advise to use a synchronous system for granting execution time to each task, this will guarantee for each servo a constant sampling frequency.

9.3 Recommendations

I would advise to watch the developments at the Compact-Disc laboratory as close as possible. They are developing and implementing the technique of integrating digital servo systems in dedicated ic's. One should watch for the differences (in scale and in principle) between the Compact-Disc servos and the LaserVision servos and try to work out methods to compensate for those differences.

The ADC could be provided with a programmable reference so the input range of the ADC could be chosen by the microprocessor. This would eliminate the disadvantages of the fixed input ranges mentioned in section 9.1. Another, and perhaps cheaper, method if one integrates the converter, is to distribute the levels of the ADC on a non-linear bases, i.e. few levels for large input signals and many levels for small input levels. Because of the non-linear nature of the ADC, the servo would be difficult to develop.

The feasibility of the radial mirror, radial slide, tangential mirror and spinning motor servos still has to be examined be-

fore a total integration of all servo systems could be tried.

Also the lower limits of the sampling frequencies as well as the wordlengths for all servos must be explored.

The digital compensation filters should be optimized directly. Translation of analog filters is not enough because optimal analog filters do not necessarily result in optimal digital filters.

A search should be made for adaptive systems, usable in a LaserVision surrounding. Investigated should be what could be adaptive in the servos and how that should be implemented.

REFERENCES

1. O.I. Elgard, Control Systems Theory,
McGraw-Hill, 1976.

2. R. Iserman, Digital Control Systems,
Berlin: Springer-Verlag, 1981.

3. L.R. Rabiner and B. Gold, Theory and
Application of Digital Signal Processing,
Englewood Cliffs, NJ: Prentice-Hall, 1975.

4. R.M. Aarts, A digital controller for servo systems,
Nat. Lab. Techn. Note 171/83.

5. TMS32010 USER's GUIDE,
Texas Instruments, 1983.

6. Ir. K.A Immink and J. van Gerwen, The control
of periodic disturbances, Nat. Lab. Report 5440.

7. Ir. K.A. Immink, A controller design based on
periodic disturbances, Nat. Lab. Techn. Note 204/77.

8. Standard on a pre-recorded optical reflective video-disc
system 'LaserVision' 50Hz/625 lines- PAL and SECAM, IEC,
Techn. Committee Nr.60: recording, Sub-Committee 60B:
video recording, april 1983.

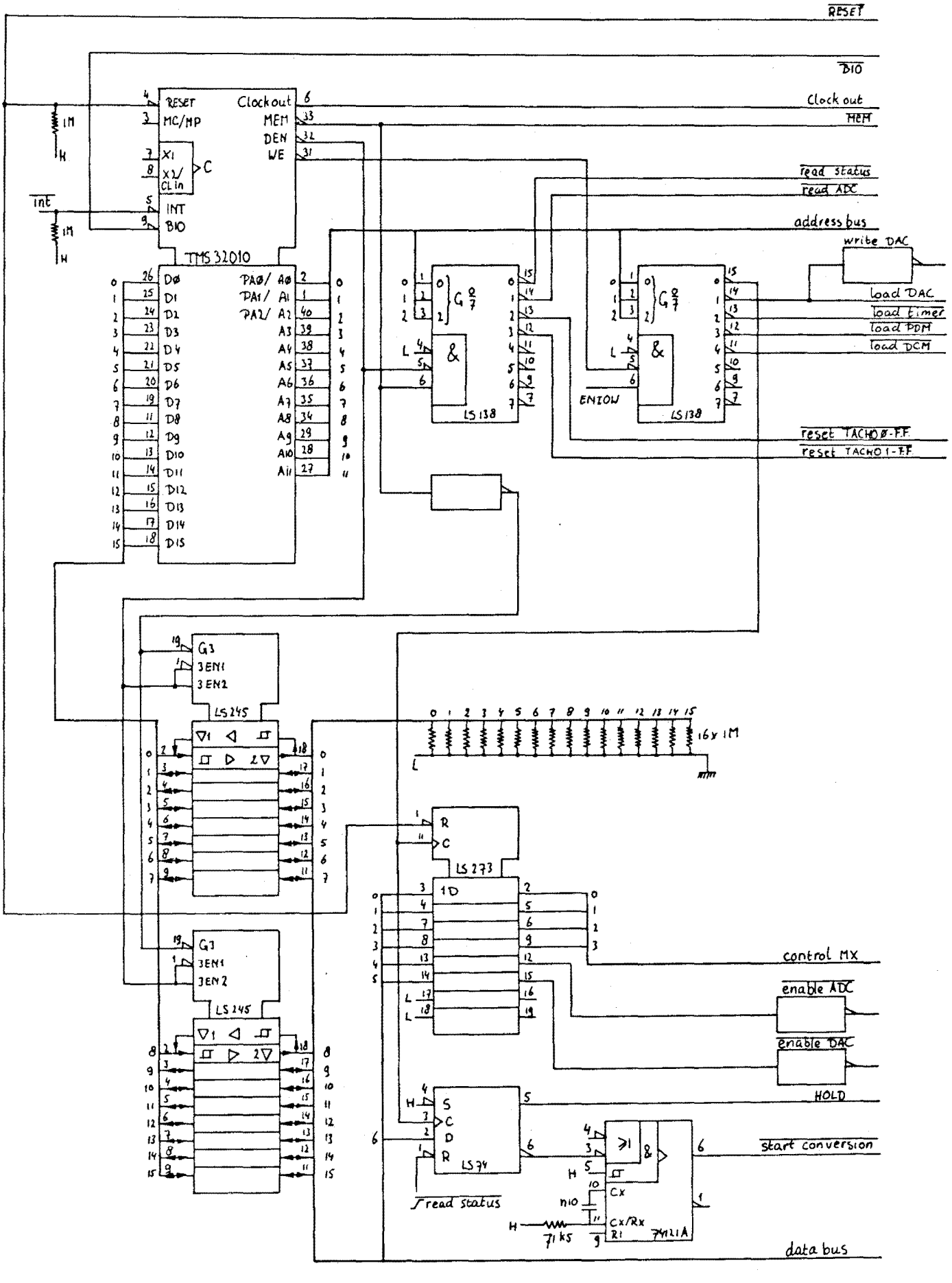
9. P2251 Total-System Set (TSS) reference manual,
Philips, 1982.

10. P2500 system reference manual,
Philips, 1982.

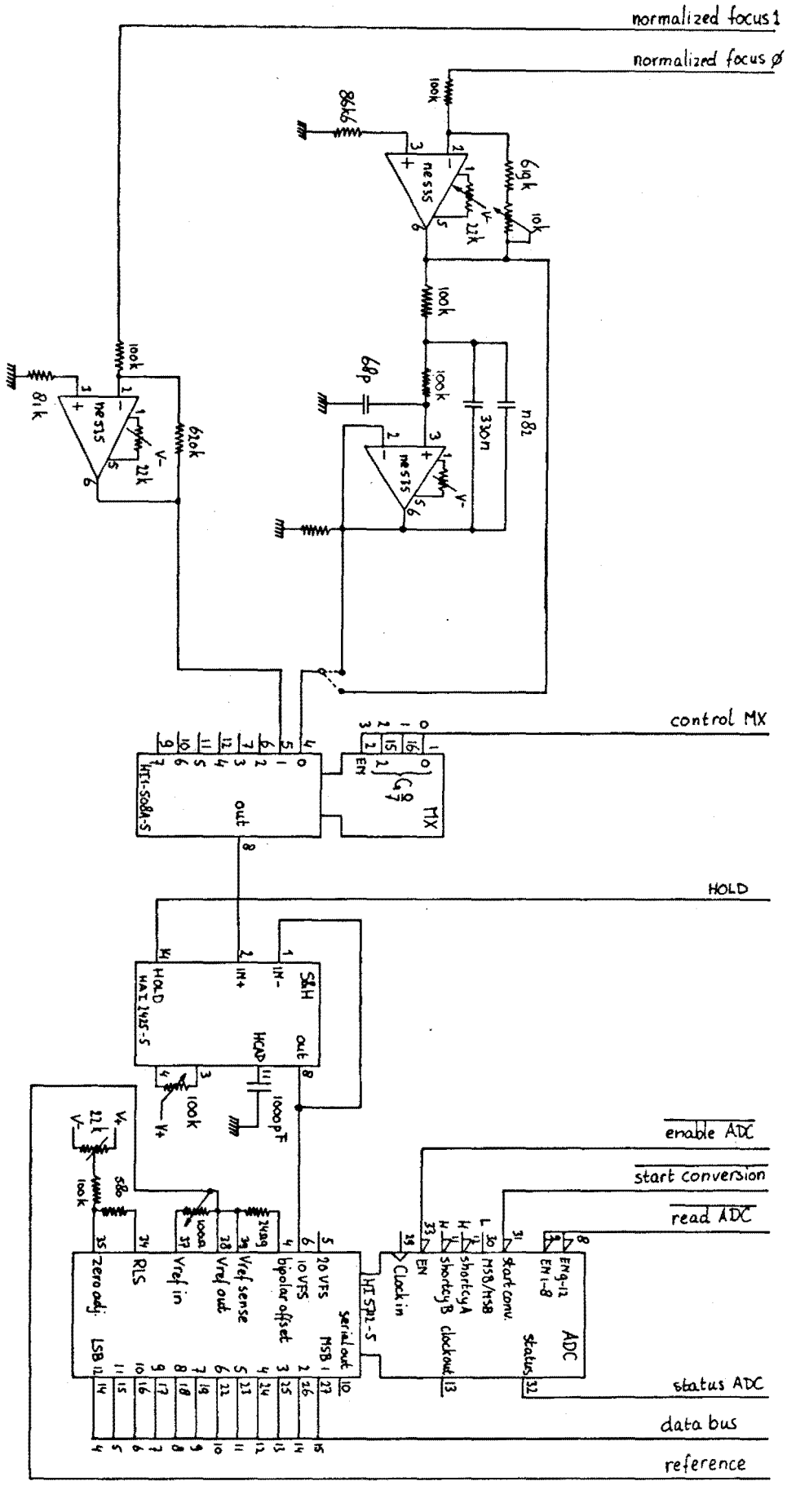
11. TMS320 Evaluation Module reference manual,
Texas Instruments, 1983.

12. 22VP720 service manual,
Philips, 1982.

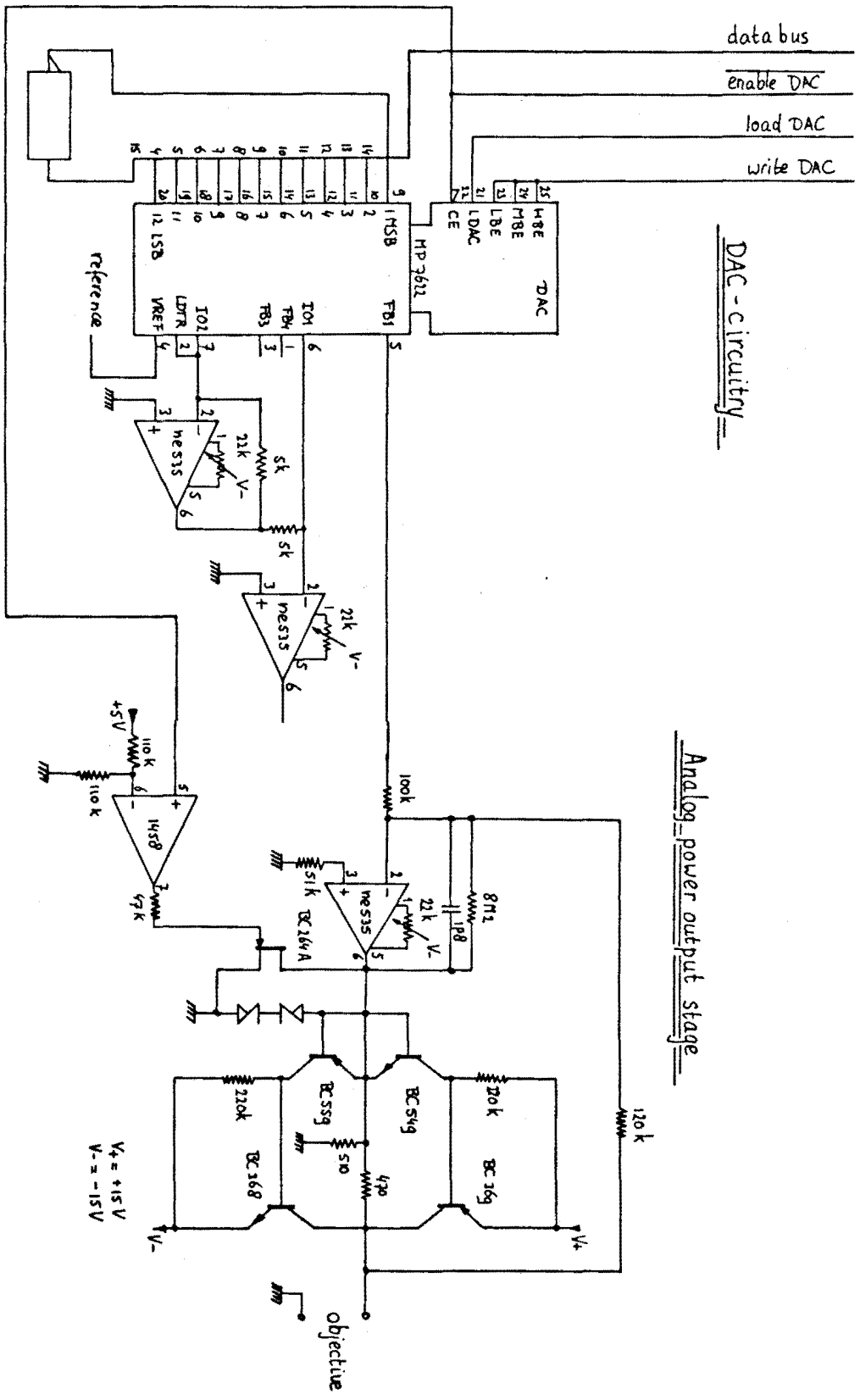
APPENDIX I
Schematics prototype hardware: processor; control circuitry



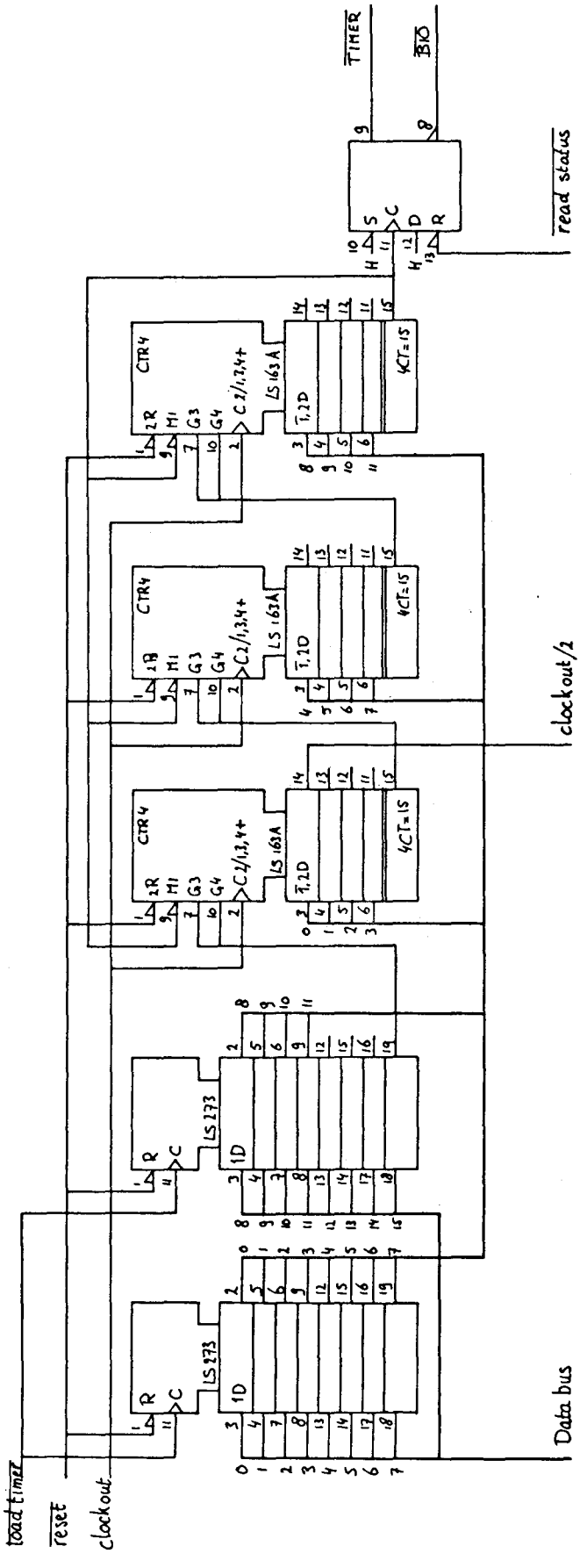
APPENDIX I
Schematics prototype hardware: analog input stage



APPENDIX I
Schematics prototype hardware: analog output stage

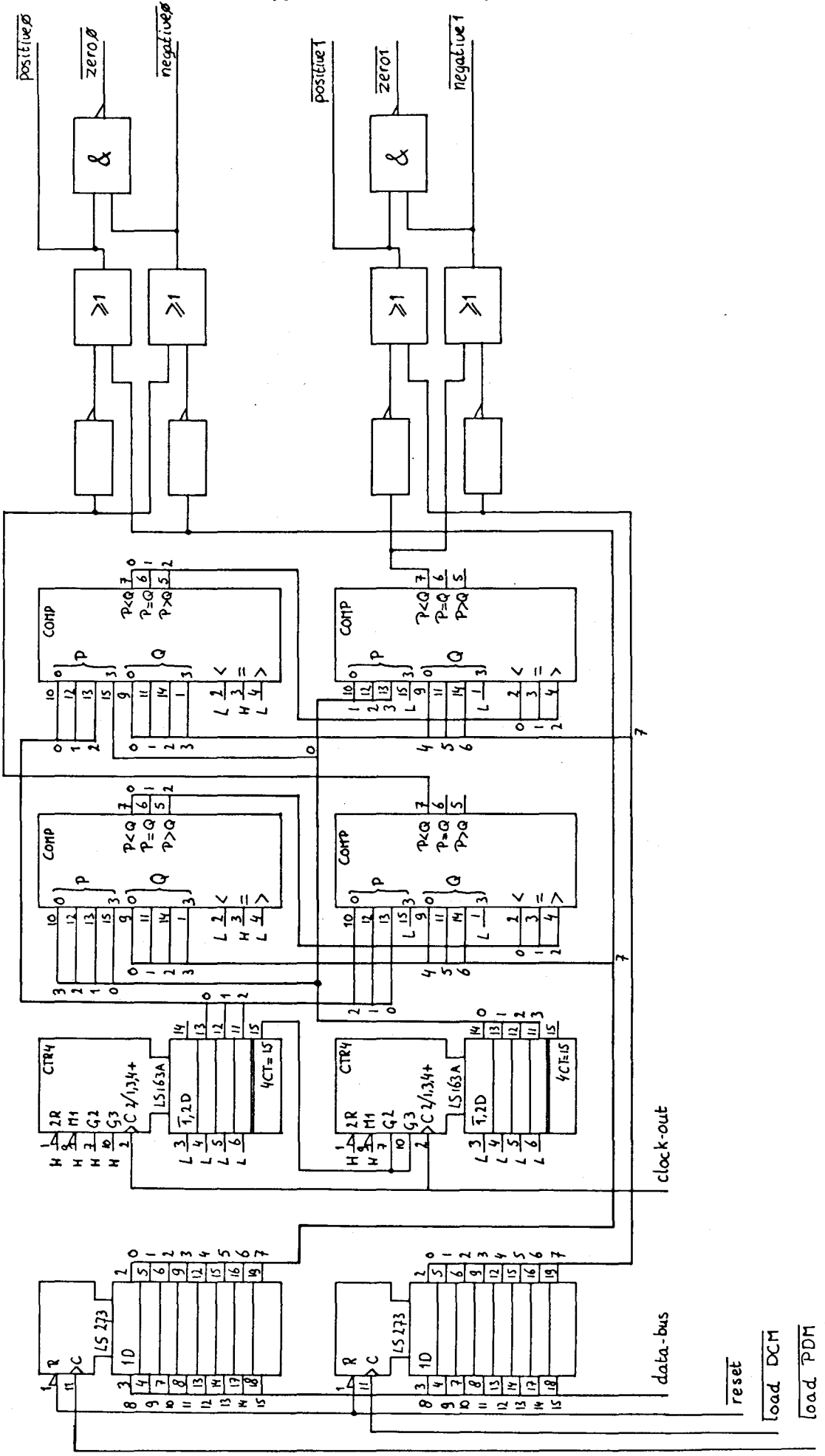


APPENDIX I
Schematics prototype hardware: programmable sample timer

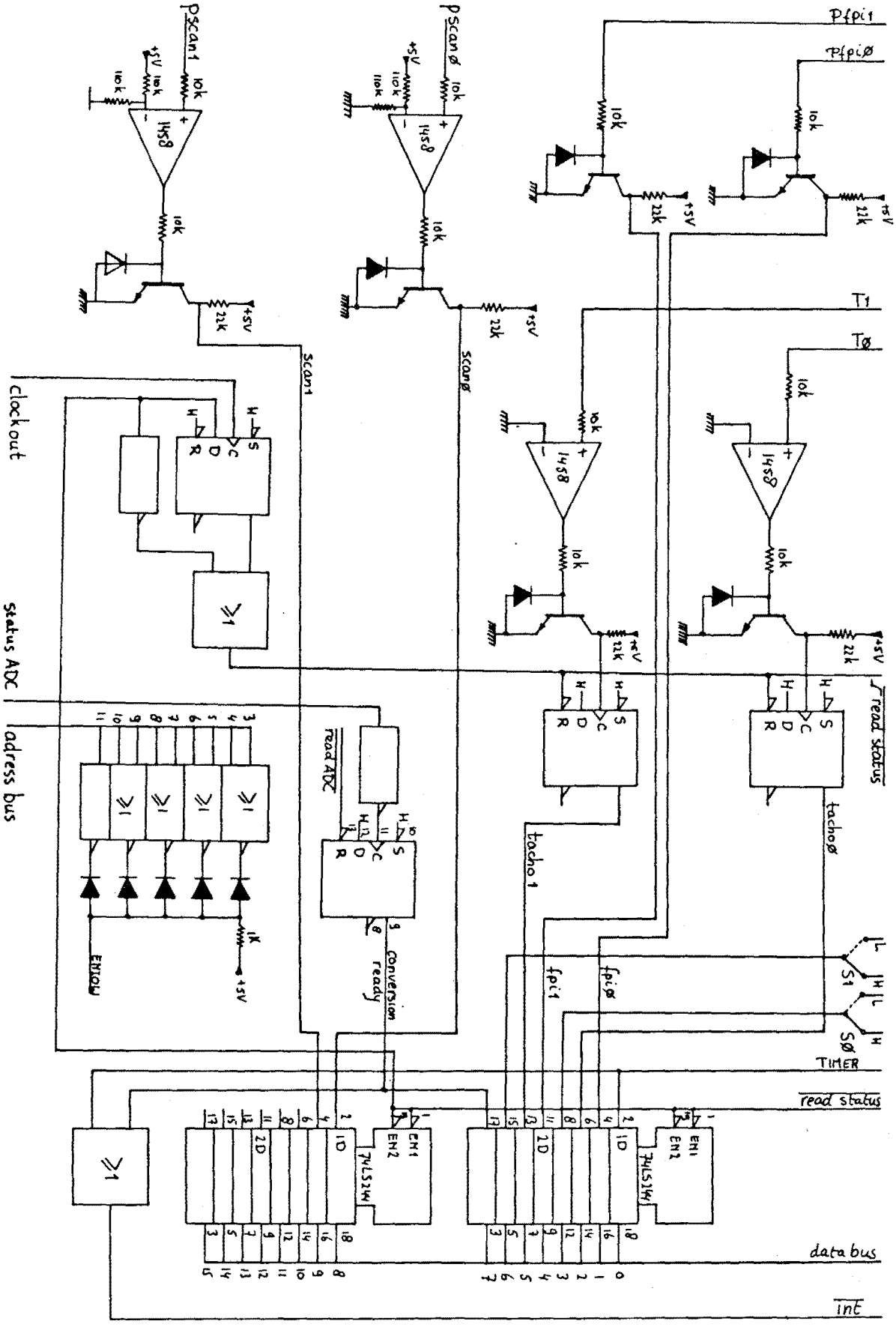


APPENDIX I

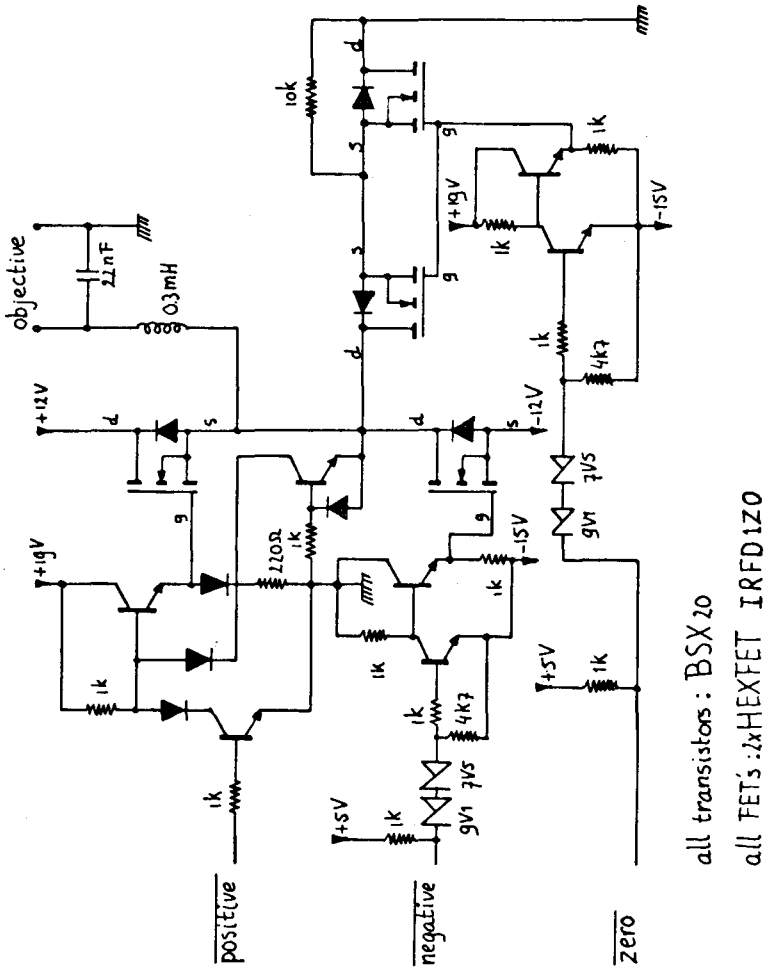
Schematics prototype hardware: the pulse modulators

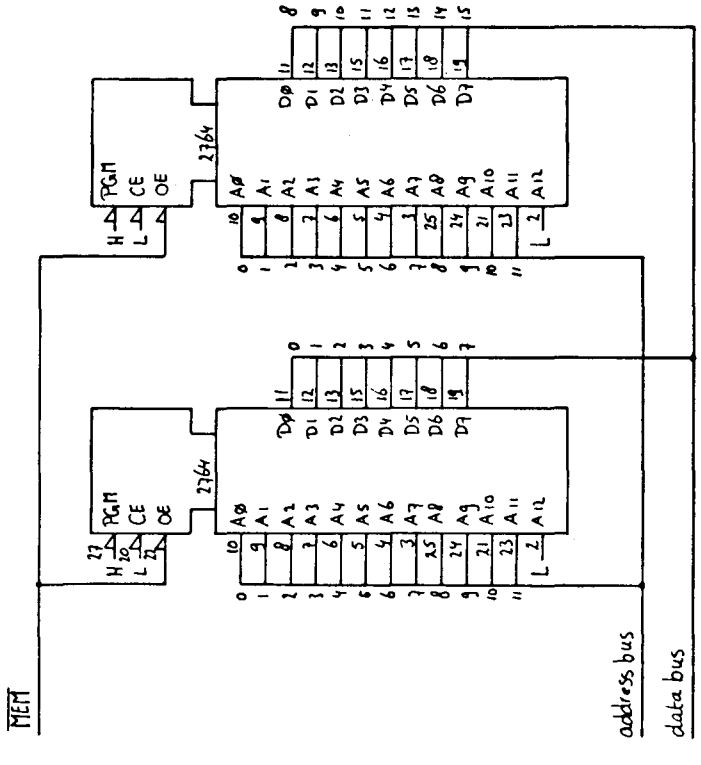


APPENDIX I
Schematics prototype hardware: player interface
other circuitry



APPENDIX I
Schematics prototype hardware: modulator power stage
Eprom sockets





APPENDIX II
Listing prototype software

```
* Dual Digital Servo Program ** VERSION (840405/12.00) *
*
*
***** FUNCTIONAL DESCRIPTION *****
*
* This is the assembler source listing of the program for the
* TMS320 Digital Signal Processor. The program performs all
* tasks necessary for two LaserVision focus servos. The program
* is interrupt driven. Interrupts from an external timer cause
* a context switch between the two processes and the start of
* a A-to-D conversion of the error signal of one of the two.
* Under normal circumstances, an interrupt from the ADC causes
* execution of one of the two filter routines. Both processes
* and filter routines are, aside from the memory addresses
* used, exactly the same. The processes consist of a part,
* which brings the objective in a focusing position and, in
* case of an error, pulls the objective away from the disc, a
* part which monitors the player signals and a routine which
* waits for a timer interrupt. A process can be stopped/star-
* ted by flipping the on/off switch. The program hangs up if
* one of the processes is started while the particular player
* is off or holds no disk. The process should be started while
* the disk is nearly at its full turning speed. If the process
* is started earlier, the acceleration is to high for the pre-
* dictive controller to be stable and the objective will drop
* out of focus. The process will try once more to focus the
* laser beam and will be successful the second time.
*
***** END OF FUNCTIONAL DESCRIPTION *****
*
*
***** ASSEMBLER DEFINITIONS *****
ar0     equ     0       * auxiliary register 0
ar1     equ     1       * auxiliary register 1
pa0     equ     0       * port address 0
pa1     equ     1       * port address 1
pa2     equ     2       * port address 2
pa3     equ     3       * port address 3
pa4     equ     4       * port address 4
hwstat  equ     0       * hardware status
swstat  equ     1       * software status
one     equ     2       * one
minus   equ     3       * minus one
timval  equ     4       * timer load value
hibyte  equ     5       * high byte mask
lobyte  equ     6       * low byte mask
float0  equ     7       * focus 0 float value
interm  equ     8       * intermediate storage location
mult0   equ     9       * focus 0 multiplication value
minniv  equ     10      * minimal value of focus error
maxw    equ     11      * wait loop count value
psave   equ     12      * save location for return PC
```

APPENDIX II
Listing prototype software

```
apsave  equ     13      * previous return PC
par     equ     15      * start of parameter block
lpar    equ     20      * end of parameter block
chan0   equ     21      * channel 0 selection value
conv0   equ     22      * selection value for timer int. handler
nrtac0  equ     23      * number of tachopulses per revolution
nrsam0  equ     24      * number of samples per revolution
wrtep0  equ     27      * pointer in delayline 0
stali0  equ     28      * start of delayline 0
stoli0  equ     29      * end of delayline 0
ovflo0  equ     30      * overflow value 0
actua0  equ     31      * actuator value 0
drive0  equ     32      * previous actuator value 0
sens0   equ     33      * sensor value 0
osens0  equ     34      * previous sensor value 0
error0  equ     35      * before previous sensor value 0
mult1   equ     36      * focus 1 multiplication value
chan1   equ     37      * channel 1 selection value
conv1   equ     38      * selection value for tim. int. handler
nrtac1  equ     39      * number of tachopulses per revolution
nrsam1  equ     40      * number of samples per revolution
wrtep1  equ     43      * pointer in delayline 1
stali1  equ     44      * start of delayline 1
stoli1  equ     45      * end of delayline 1
float1  equ     46      * focus 1 float value
ovflo1  equ     47      * overflow value 1
actua1  equ     48      * actuator value 1
drive1  equ     49      * previous actuator value 1
sens1   equ     50      * sensor value 1
osens1  equ     51      * previous sensor value 1
error1  equ     52      * before previous sensor value 1
rdwrd0  equ     53      * read word 0 (from delayline)
wrwrd0  equ     54      * write word 0 (to delayline)
rdwrd1  equ     55      * read word 1 (from delayline)
wrwrd1  equ     56      * write word 1 (to delayline)
hgwrd0  equ     57      * high word 0 (high byte from rdwrd0)
lowrd0  equ     58      * low word 0 (low byte from rdwrd0)
hgwrd1  equ     59      * high word 1 (high byte from rdwrd1)
lowrd1  equ     60      * low word 1 (low byte from rdwrd1)
del0    equ     61      * lowpass filter value 0
prdel0  equ     62      * previous lowpass filter value 0
dlstat  equ     63      * status of delaylines
conv    equ     64      * selection value for tim. int. handler
accl    equ     65      * save location for accumulator low
acch    equ     66      * save location for accumulator high
ar0sa0  equ     67      * save location for aux. reg.0 focus 0
ar1sa0  equ     68      * save location for aux. reg.1 focus 0
ar0sa1  equ     69      * save location for aux. reg.0 focus 1
ar1sa1  equ     70      * save location for aux. reg.1 focus 1
demul0  equ     71      * multiplication factor during delay 0
demul1  equ     72      * multiplication factor during delay 1
scmul0  equ     73      * multiplication factor during scan 0
```

APPENDIX II
Listing prototype software

```
scmul1  equ     74      * multiplication factor during scan 1
resul0  equ     75      * result of compensation filtering 0
pres0   equ     76      * previous result of compensation filt.0
ppres0  equ     77      * before previous result of comp. filt.0
resul1  equ     78      * result of compensation filtering 1
pres1   equ     79      * previous result of compensation filt.1
ppres1  equ     80      * before previous result of comp. filt.1
del1    equ     81      * lowpass filter value 1
prdel1  equ     82      * previous lowpass filter value 1
***** END OF ASSEMBLER DEFINITIONS *****
*
*
***** RESET/POWER-UP ACTION *****
        aorg    0
        symt
reset   b       main            * branch to main routine
***** END OF RESET/POWER-UP ACTION *****
*
*
***** INTERRUPT SERVICE ROUTINE *****
* The interrupt handler inspects the BIO line to check which
* signal caused the interrupt.
* In case of an interrupt caused by conversion, the handler
* checks which process is actual and, if the process is run-
* ning, performs a branch to a filter routine. If the process
* is not running, it resets the conversion parameter and
* returns to the process. A context save is not necessary.
* In case of an interrupt caused by the timer, the handler
* changes the process by changing the value on stack below
* TOS (because this part is always executed from a subroutine:
* filter or timer). Before returning to the routine, the
* handler starts a conversion (or selects a channel, depending
* on the value of CONV). During this part, the accumulator and
* the (processor-) status register are saved.
*
inter   bioz    timerh
*
* end of conversion handling
*
convh   lac     one,8       * check which process actual
        and     swstat
        bnz     focus1
focus0  in      sens0,pa1   * input conv.value and reset int.
        lac     one,2       * test if focus0 is running
        and     swstat
        bnz     filt0       * branch to filter0
        lac     one         * reset converting0
        xor     minus
        and     swstat
        sacl    swstat
        eint
        ret
```

APPENDIX II
Listing prototype software

```
focus1  in      sens1,pa1  * input conv. value and reset int.
        lac     one,6      * test if focus1 is running
        and     swstat
        bnz     filt1      * branch to filter1
        lac     one,4      * reset converting1
        xor     minus
        and     swstat
        sacl    swstat
        eint
        ret
*
* timer handling (the location of the conversion start
* is important)
*
timerh  in      hwstat,pa0 * reset interrupt cause
        sacl    accl       * save context
        sach    acch
        lac     one,8      * change proces bit
        xor     swstat
        sacl    swstat
        pop                * change proces
        out     conv,pa0   ***** start conv. or select channel
        sacl    interm
        pop
        sacl    psave
        lac     apsave
        dmov    psave
        push
        lac     interm
        push
        zalh    acch       * restore context
        add     accl
        ret
***** END OF INTERRUPT HANDLER *****
*
*
***** TIMER ROUTINES *****
* The timer routines change the channel and conversion code,
* set ar0 and save the auxiliary registers
* and wait until the process bit is changed, which is done by
* the timer interrupt handler.
* They then return to the changed process.
*
timer0  out     chan1,pa0  * set channel to 1
        lac     conv1      * set converting value to 1
        sacl    conv
        larp    ar0        * during conv.int. ar0
        sar     ar0,ar0sa0
        sar     ar1,ar1sa0
timw1   eint
        nop
        dint
```

APPENDIX II
Listing prototype software

```
        lac     one,8       * is process bit set to 1
        and     swstat
        bz      timw1
        ret
*
timer1  out     chan0,pa0   * set channel to 0
        lac     conv0       * set converting value to 0
        sacl    conv
        larp    ar0         * during conv.int. ar0
        sar     ar0,ar0sa1
        sar     ar1,ar1sa1
timw0   eint
        nop
        dint
        lac     one,8       * is process bit set to 0
        and     swstat
        bnz     timw0
        ret
***** END OF TIMER ROUTINES *****
*
*
```

APPENDIX II
Listing prototype software

```
*****  DATA TABLES   *****
floa0d  data    "700
floa1d  data    "700
ovfl0d  data    31130           * 19/20 in Q15
ovfl1d  data    31130           * 19/20 in Q15
multd0  data    10              * 10 decimaal in Q0
multd1  data    10
dmuld0  data    6               * multipl.factor during delay
dmuld1  data    6
timd    data    -128            * (2x)14650 Hz samplefreq.
een     data    1
min     data    -1
maxwd   data    183
minnd   data    "1000
param   data    4100,-12840,10018  *parameters p2-p0:div by 10
        data    115,-35         * parameters q2-q1:div by 10
hibte   data    "ff00
stli0   data    "800            * start of delay line 0
stre0   data    "802            * start of readpointer
nstl0   data    "921            * end of delay line 0
stli1   data    "c00            * start of delay line 1
stre1   data    "c02            * start of readpointer1
nstl1   data    "d21            * end of delay line 1
***** END OF DATA TABLES *****
*
*
***** FOCUS 1 PROCES *****
*
* The process consist of a start/restart section and a test
* loop. In the test loop, FPI, ON, SCAN and TACHO are tested
* and appropriate actions are taken. Interrupts are enabled
* on specified moments to avoid the need for a context save.
*
* START/RESTART SECTION *
* 10 periods -5V (down) on restart:
rst1    dint
        lac     one,6       * reset running 1
        xor     minus
        and     swstat
        sacl    swstat
        lac     chan1       * res. conv. capability chan1
        sacl    conv1
        call    timer1
cont10  lark    ar1,9
        zac
        sub     one,14
        sacl    actua1
        out     actua1,pa4
wait4   call    timer1
        lar     ar1,ar1sa1
        larp    ar1
        banz    wait4
```

APPENDIX II
Listing prototype software

```
        zac
        sacl    actua1
        out     actua1,pa4
* wait 1 second:
        lar     ar0,maxw
        lark    ar1,"50
rest1   call    timer1
        lar     ar0,ar0sa1
        lar     ar1,ar1sa1
        larp    ar0
        banz    rest1
        larp    ar1
        lar     ar0,maxw
        banz    rest1
* test for stop:
start1  call    timer1
        lac     one,6       * on1
        and     hwstat
        bz      start1
        zac
        sacl    nrsam1      * reset nr of samples
        lac     minus
        sacl    nrtac1      * reset nr of tachopulses
        sacl    hgwrd1      * reset read words
        sacl    lowrd1
        lack    "c          * reset delay 1 status
        xor     minus
        and     dlstat
        sacl    dlstat
        lac     stali1      * set pointers to start
        sacl    wrtep1
        lac     stare1
        sacl    readp1
* initialise filter values
        lac     float1
        sacl    actua1
        larp    ar0
        lark    ar0,actua1
        dmov    *+
        zac
        sacl    sens1
        lark    ar0,sens1
        dmov    *+
        dmov    *+
* 10 periods +5V (up)
        lac     one,14
        sacl    actua1
        lark    ar1,9
        out     actua1,pa4
wait5   call    timer1
        lar     ar1,ar1sa1
        larp    ar1
```

APPENDIX II
Listing prototype software

```
        banz    wait5
*  float up voltage for maximum 1 second
        larp    0
        lar     ar0,maxw
        lark    ar1,¨50
        lac     float1
        sacl    actua1
        out     actua1,pa4
* wait for FPI
wait6   banz    cont11
        larp    1
        lar     ar0,maxw
        banz    cont11
        b       stop
cont11  call    timer1
        lar     ar0,ar0sa1
        lar     ar1,ar1sa1
        larp    ar0
        lac     one,4       * fpi1
        and     hwstat
        bz      wait6
* start conversion
        lac     one,6       * set conv. capability chan1
        or      conv1
        sacl    conv1
lp12    call    timer1      * int. handler starts conv.
* wait for conversion ready
        lac     one,4       * set conversion 1
        or      swstat
        sacl    swstat
lp11    eint                * conv.int.handler reads sensor
        nop
        dint
        lac     one,4       * conversion ready
        and     swstat
        bnz     lp11
*test FPI
        lac     one,4       * fpi1
        and     hwstat
        bz      rst1
* test absolute value focus error
        lark    ar1,osens1
        larp    ar1
        dmov    *-
        dmov    *
        lac     sens1
        abs
        sub     minniv
        blz     lp12
        lac     one,6       * set running 1
        or      swstat
        sacl    swstat
```

APPENDIX II
Listing prototype software

```
        call    timer1      * tim.int. origins from subr.
*
* TEST LOOP *
loop1   dint
        lac     one,4       * test if fpi1 true
        and     hwstat
        eint
        bz      rst1        * if not restart
        dint
        lac     one,6       * test if on1 switch true
        and     hwstat
        eint
        bz      rst1        * if not restart
        dint
        lac     one,9       * test if player 1 scanning
        and     hwstat
        eint
        bz      scan1
nscan1  dint                * not scanning then
        lac     one,5       * test if tacho1 active
        and     hwstat
        eint
        bz      loop1       * if not skip
        dint
        lac     nrtac1      * if so increment nr of tachopulses
        add     one
        eint
        sacl    nrtac1
        in      interm,pa3  * reset tacho1
        dint
        lac     one,2       * set delay active
        or      dlstat
        eint
        sacl    dlstat
        b       loop1
scan1   dint                * scanning then
        lack    ~3          * reset delay active and running
        and     dlstat
        eint
        sacl    dlstat
        dint
        zac                 * clear high and low read word
        sacl    hgwrd1
        eint
        sacl    lowrd1
        dint
        lac     minus       * reinitialise the number of tacho-
        sacl    nrtac1      * pulses and
        zac
        eint
        sacl    nrsam1      * the number of samples.
        dint
```

APPENDIX II
Listing prototype software

```
        lac     stali1      * reset write pointer to start
        eint
        sacl    wrtep1
        dint
        lac     scmul1      * change multiplication factor
        sacl    mult1
        eint
        b       loop1
*****  END OF FOCUS 1 PROCESS *****
*
*
***** FOCUS 0 PROCES *****
*
* The process consist of a start/restart section and a test
* loop. In the test loop, FPI, ON, SCAN and TACHO are tested
* and appropriate actions are taken. Interrupts are enabled
* on specified moments to avoid the need for a context save.
*
* START/RESTART SECTION *
* 10 periods -5V (down) on restart:
rst0    dint
        lac     one,2       * reset running 0
        xor     minus
        and     swstat
        sacl    swstat
        lac     chan0       * reset conv. capability for chan0
        sacl    conv0
        call    timer0
cont8   lark    ar1,9
        zac
        sub     one,14
        sacl    actua0
        out     actua0,pa3
wait1   call    timer0
        lar     ar1,ar1sa0
        larp    ar1
        banz    wait1
        zac
        sacl    actua0
        out     actua0,pa3
* wait 1 second:
        lar     ar0,maxw
        lark    ar1,"50
rest0   call    timer0
        lar     ar0,ar0sa0
        lar     ar1,ar1sa0
        larp    ar0
        banz    rest0
        larp    ar1
        lar     ar0,maxw
        banz    rest0
* test for stop:
```

APPENDIX II
Listing prototype software

```
start0  call    timer0
        lac     one,3       * on0
        and     hwstat
        bz      start0
        zac
        sacl    nrsam0      * reset nr of samples
        lac     minus
        sacl    nrtac0      * reset nr of tachopulses
        sacl    hgwrd0      * reset read words
        sacl    lowrd0
        lack    ¨3          * reset delay 0 status
        xor     minus
        and     dlstat
        sacl    dlstat
        lac     stali0      * set pointer to start
        sacl    wrtep0
        lac     stare0
        sacl    readp0
* initialise filter values
        lac     float0
        sacl    actua0
        larp    ar0
        lark    ar0,actua0
        dmov    *+
        zac
        sacl    sens0
        lark    ar0,sens0
        dmov    *+
        dmov    *+
* 10 periods +5V (up)
        lac     one,14
        sacl    actua0
        lark    ar1,9
        out     actua0,pa3
wait2   call    timer0
        lar     ar1,ar1sa0
        larp    ar1
        banz    wait2
*  float up voltage for maximum 1 second
        larp    0
        lar     ar0,maxw
        lark    ar1,¨50
        lac     float0
        sacl    actua0
        out     actua0,pa3
* wait for FPI
wait3   banz    cont9
        larp    ar1
        lar     ar0,maxw
        banz    cont9
        b       stop
cont9   call    timer0
```

APPENDIX II
Listing prototype software

```
        lar     ar0,ar0sa0
        lar     ar1,ar1sa0
        larp    ar0
        lac     one,1       * fpi0
        and     hwstat
        bz      wait3
* start conversion
        lac     one,6       * set conv. capability for chan0
        or      conv0
        sacl    conv0
lp02    call    timer0      * interrupt handler starts conversion
* wait for conversion ready
        lac     one         * set conversion 0
        or      swstat
        sacl    swstat
lp01    eint                * conv. int. handler reads sensor
        nop
        dint
        lac     one         * conversion ready
        and     swstat
        bnz     lp01
*test FPI
        lac     one,1       * fpi0
        and     hwstat
        bz      rst0
* test absolute value focus error
        lark    ar1,osens0
        larp    ar1
        dmov    *-
        dmov    *
        lac     sens0
        abs
        sub     minniv
        blz     lp02
        lac     one,2       * set running 0
        or      swstat
        sacl    swstat
        call    timer0      * timer int. must origin from subr.
*
* TEST LOOP *
loop0   dint
        lac     one,1       * test if fpi0 true
        and     hwstat
        eint
        bz      rst0        * if not restart
        dint
        lac     one,3       * test if on0 switch true
        and     hwstat
        eint
        bz      rst0        * if not restart
        dint
        lac     one,8       * test if player 0 scanning
```

APPENDIX II
Listing prototype software

```
        and     hwstat
        eint
        bz      scan0
nscan0  dint            * not scanning then
        lac     one,2   * test if tacho0 active
        and     hwstat
        eint
        bz      loop0   * if not skip
        dint
        lac     nrtac0  * if so increment nr of tachopulses
        add     one
        eint
        sacl    nrtac0
        in      interm,pa2 * reset tacho0
        dint
        lac     one,0   * set delay active
        or      dlstat
        eint
        sacl    dlstat
        b       loop0
scan0   dint            * scanning then
        lack    "c      * reset delay active and running
        and     dlstat
        eint
        sacl    dlstat
        dint
        zac             * clear high and low read word
        sacl    hgwrd0
        eint
        sacl    lowrd0
        dint
        lac     minus   * reinitialise number of tacho-
        sacl    nrtac0  * pulses and
        zac
        eint
        sacl    nrsam0  * number of samples.
        dint
        lac     stali0  * reset write pointer to start
        eint
        sacl    wrtep0
        dint
        lac     scmul0  * change multiplication factor
        sacl    mult0
        eint
        b       loop0
***** END OF FOCUS 0 PROCES *****
*
*
***** STOP ROUTINES *****
* stop:
stop    dint
        zac
```

APPENDIX II
Listing prototype software

```
        sacl    actua0
        out     actua0,pa3      * PWM0
        out     actua0,pa4      * PWM1
        out     actua0,pa0      * disable system
elo1    nop
        b       elo1
***** END OF STOP ROUTINES *****
*
*
```

APPENDIX II
Listing prototype software

```
***** FILTER 0 ROUTINE *****
*
* The filter routine changes the channel and conversion code,
* performs the filter calculations, calculates the value for
* the delay line and performs administration for the delay
* line. The delay line administration is split up in a part
* for even turns and one for odd turns. This is possible be-
* cause a delayline value contains two results. The filter
* calculations are copied to speed up execution. On execution:
* arp=0, on finish: arp=0. The interrupt routine has input
* the conversion value.
*
filt0   out     chan1,pa0   * change channel
        lac     conv1       * change conversion code
        sacl    conv
act0    lac     one         * test if delay 0 even
        and     nrsam0
        bnz     odd0
even0   lark    ar1,par     * perform filter diff. equation
        lark    ar0,error0
        lt      *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *+,ar0
        lta     *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *,ar0
        apac
        blz     leqz00      * protect against overflow
        addh    ovflo0
        subh    ovflo0
        b       stor00
leqz00  subh    ovflo0
        addh    ovflo0
stor00  sach    actua0,1    * shift 1 due to mult.
        lt      actua0
        mpy     mult0
        pac
        sacl    actua0
        zalh    actua0
        addh    lowrd0
        add     lowrd0,14
        sach    resul0
        eint
        out     resul0,pa3
        lac     one         * test if delay active else return
        and     dlstat
        bz      retur0
        zalh    del0        * lpf input delayline fc 310 HZ
```

APPENDIX II
Listing prototype software

```
        sub     del0,12
        add     resul0,11
        add     pres0,11
        dmov    resul0
        sach    del0
        lac     del0        * quantize result by 8 bits
        and     hibyte
evenn0  sacl    wrwrd0
        lac     one,1       * test if delay running
        and     dlstat
        bz      isam0
        lac     wrtep0      * read word at write pointer
        tblr    rdwrd0
        lac     demul0
        sacl    mult0
nendr0  lac     rdwrd0      * disassemble read word in
        and     hibyte      * high byte
        sacl    hgwrd0
        lac     rdwrd0,8    * and low byte
        sacl    lowrd0
isam0   lac     nrsam0      * update number of samples
        add     one
        sacl    nrsam0
        ret
odd0    lark    ar1,par     * perform filter diff. equation
        lark    ar0,error0
        lt      *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *+,ar0
        lta     *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *,ar0
        apac
        blz     leqz01      * protect against overflow
        addh    ovflo0
        subh    ovflo0
        b       stor01
leqz01  subh    ovflo0
        addh    ovflo0
stor01  sach    actua0,1    * shift 1 due to mult.
        lt      actua0
        mpy     mult0
        pac
        sacl    actua0
        zalh    actua0
        addh    hgwrd0
        add     hgwrd0,14
        sach    resul0
```

APPENDIX II
Listing prototype software

```
        eint
        out     resul0,pa3
        lac     one         * test if delay active else return
        and     dlstat
        bz      retur0
        zalh    del0        * lpf input delayline fc 310 HZ
        sub     del0,12
        add     resul0,11
        add     pres0,11
        dmov    resul0
        sach    del0
        lac     del0,8      * quantize result by 8 bits
        sach    ppres0
        lac     ppres0
        and     lobyte
        adds    wrwrd0      * assemble write word
        sacl    wrwrd0
        lac     wrtep0      * write write word
        tblw    wrwrd0
        add     one         * increment write pointer
        sacl    wrtep0
        lack    18          * test if nr of tachopulses 18
        sub     nrtac0
        bnz     isam0
        sacl    nrtac0      * clear nr of tachopulses
        sacl    nrsam0      * clear nr of samples
        lac     stali0      * reset write pointer
        sacl    wrtep0
        lac     one,1       * set delay running
        or      dlstat
        sacl    dlstat
        ret
retur0  lac     one,8       * test process bit
        and     swstat
        bz      retur0
        ret
***** END OF FILTER 0 ROUTINE *****
*
*
***** FILTER 1 ROUTINE *****
*
* The filter routine changes the channel and conversion code,
* performs the filter calculations, calculates the value for
* the delay line and performs administration for the delay
* line. The delay line administration is split up in a part
* for even turns and one for odd turns. This is possible be-
* cause a delayline value contains two results. The filter
* calculations are copied to speed up execution. On execution:
* arp=0, on finish: arp=0. The interrupt routine has input
* the conversion value.
*
filt1   out     chan0,pa0   * change channel
```

APPENDIX II
Listing prototype software

```
        lac     conv0       * change conversion code
        sacl    conv
act1    lac     one         * test if delay 1 even
        and     nrsam1
        bnz     odd1
even1   lark    ar1,par     * perform filter diff. equation
        lark    ar0,error1
        lt      *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *+,ar0
        lta     *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *,ar0
        apac
        blz     leqz10      * protect against overflow
        addh    ovflo1
        subh    ovflo1
        b       stor10
leqz10  subh    ovflo1
        addh    ovflo1
stor10  sach    actua1,1    * shift 1 due to mult.
        lt      actua1
        mpy     mult1
        pac
        sacl    actua1
        zalh    actua1
        addh    lowrd1
        add     lowrd1,14
        sach    resul1
        eint
        out     resul1,pa4
        lac     one,2       * test if delay active else return
        and     dlstat
        bz      retur1
        zalh    del1        * lpf input delayline fc 310 HZ
        sub     del1,12
        add     resul1,11
        add     pres1,11
        dmov    resul1
        sach    del1
        lac     del1        * quantize result by 8 bits
        and     hibyte
evenn1  sacl    wrwrd1
        lac     one,3       * test if delay running
        and     dlstat
        bz      isam1
        lac     wrtep1      * read word at write pointer
        tblr    rdwrd1
```

APPENDIX II
Listing prototype software

```
        lac     demul1
        sacl    mult1
nendr1  lac     rdwrd1      * disassemble read word in
        and     hibyte      * high byte
        sacl    hgwrd1
        lac     rdwrd1,8    * and low byte
        sacl    lowrd1
isam1   lac     nrsam1      * update number of samples
        add     one
        sacl    nrsam1
        ret
odd1    lark    ar1,par     * perform filter diff. equation
        lark    ar0,error1
        lt      *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *+,ar0
        lta     *-,ar1
        mpy     *+,ar0
        ltd     *-,ar1
        mpy     *,ar0
        apac
        blz     leqz11      * protect against overflow
        addh    ovflo1
        subh    ovflo1
        b       stor11
leqz11  subh    ovflo1
        addh    ovflo1
stor11  sach    actua1,1    * shift 1 due to mult.
        lt      actua1
        mpy     mult1
        pac
        sacl    actua1
        zalh    actua1
        addh    hgwrd1
        add     hgwrd1,14
        sach    resul1
        eint
        out     resul1,pa3
        lac     one,2       * test if delay active else return
        and     dlstat
        bz      retur1
        zalh    del1        * lpf input delayline fc 310 HZ
        sub     del1,12
        add     resul1,11
        add     pres1,11
        dmov    resul1
        sach    del1
        lac     del1,8      * quantize result by 8 bits
        sach    ppres1
```

APPENDIX II
Listing prototype software

```
        lac     ppres1
        and     lobyte
        adds    wrwrd1      * assemble write word
        sacl    wrwrd1
        lac     wrtep1      * write write word
        tblw    wrwrd1
        add     one         * increment write pointer
        sacl    wrtep1
        lack    18          * test if nr of tachopulses 18
        sub     nrtac1
        bnz     isam1
        sacl    nrtac1      * clear nr of tachopulses
        sacl    nrsam1      * clear nr of samples
        lac     stali1      * reset write pointer
        sacl    wrtep1
        lac     one,3       * set delay running
        or      dlstat
        sacl    dlstat
        ret
retur1  lac     one,8       * test process bit
        and     swstat
        bnz     retur1
        ret
***** END OF FILTER 1 ROUTINE *****
*
*
```

APPENDIX II
Listing prototype software

```
***** INITIALISATION OF VARIABLES *****
main    lack    een
        tblr    one
        lack    min
        tblr    minus
        lack    ovfl0d
        tblr    ovflo0
        lack    ovfl1d
        tblr    ovflo1
        lack    multd0
        tblr    mult0
        tblr    scmul0
        lack    dmuld0
        tblr    demul0
        lack    dmuld1
        tblr    demul1
        lack    multd1
        tblr    mult1
        tblr    scmul1
        lack    maxwd
        tblr    maxw
        lack    minnd
        tblr    minniv
        lack    floa0d
        tblr    float0
        lack    floa1d
        tblr    float1
        lack    hibte
        tblr    hibyte
        lack    stli0
        tblr    stali0
        lack    nstl0
        tblr    stoli0
        lack    stre0
        tblr    stare0
        lack    stli1
        tblr    stali1
        lack    nstl1
        tblr    stoli1
        lack    stre1
        tblr    stare1
        lack    255
        sacl    lobyte
        lack    start1
        sacl    apsave
        zac
        sacl    dlstat
        sacl    swstat          * process 0 active
*
* initialise filter parameters
        lark    ar1,5
        lark    ar0,par
```

APPENDIX II
Listing prototype software

```
        lack    param
par1    larp    0
        tblr    *+,ar1
        add     one
        banz    par1
*
* initialise channel and conversion codes
        lack    "18             * adc enab, dac disab, chan 0
        sacl    chan0
        sacl    conv0
        lack    "19             * adc enab, dac disab, chan 1
        sacl    chan1
        sacl    conv1
***** END OF INITIALISATION OF VARIABLES
*
*
***** INITIALISATION OF HARDWARE *****
        sovm                    * set overflow mode
        lack    timd            * initialise timer
        tblr    timval
        out     timval,pa2
        in      hwstat,pa1      * reset converting interrupt
        in      hwstat,pa2      * reset tacho ff 0
        in      hwstat,pa3      * reset tacho ff 1
        in      hwstat,pa0      * reset timer interrupt
        b       start0          * startup system by
*                                 branching to proces 0
***** END OF INITIALISATION OF HARDWARE *****
*
*
***** END OF PROGRAM *****
        end
```

APPENDIX III
Listing Optimum

```
PROGRAM OPTIMUM;(*VERSION 840422/22.00 *)

(* By this program, parameters of the difference equation can
be optimized, such that the focus servo maximally reduces
errors of a quadratic form. The program optimizes one parame-
ter at a time using a newton-raphson minimization method on a
quadratic criterion function. The criterion function is the
quadrated error signal plus a weighted quadrated control force
signal, summated over a certain time span in which the servo
must follow a given disturbance. The heigth of the disturbance
is adapted automatically to the quality of the servo. From the
start parameters have to be given, that are 'in the neighbour-
hood of the ideal parameters. Also a weigh-factor has to be
given to give the control force a relative importance to the
error in the calculation of the criterion function. The program
continuously displays its progress on the screen and afterwards
results are printed. During execution the printer must be on-
line. *)


CONST t5 = 5.30E-5;      (* time-constant of process *)
TYPE  par=ARRAY [1..9] of REAL;
VAR   a,b,c,d,e,f,g,B1,C1,k,K1,S,S1,S2,pi,hr,hp:REAL;
      max,mult,p1,p2,p3,p4,p5,q0,q1,q2,q3,q4,q5,factor:REAL;
      choice,bsomtot,i,j,l,m,n,cr,raphsom,bersom:INTEGER;
      er,ep,ur,up:ARRAY [0..5] of REAL;
      queued: ARRAY [0..4] of INTEGER;
      ervast:BOOLEAN;
      p:par;
      printer:FILE of CHAR;


PROCEDURE display(p:par;kf,factor,mult:real;raphsom,bersom,
                  bsomtot:integer;ervast,urvast:boolean);

(* procedure display displays the current situation on the
screen, in a defined format, i.e. at all times, the same pa-
rameter is at the same location on the screen. The parameters
are displayed, the drive factor (the weigh-factor for the
control force in the criterion function), the disturbance
factor (the factor with which the disturbance function is
multiplied), the value of the criterion function, the number
of times the criterion function was calculated (criterion
function cycles) for this newton-raphson cycle, whether the
control force has been saturated, whether the servo dropped
out of focus, the number of newton-raphson cycles so far and
the total of criterion function cycles so far. *)

VAR home:integer;
BEGIN
   home:=1;
   unitwrite(2,home,1);
```

APPENDIX III
Listing Optimum

```
   writeln('p[1] (q1) =',p[1],'       ');
   writeln('p[2] (q2) =',p[2],'       ');
   writeln('p[3] (q3) =',p[3],'       ');
   writeln('p[4] (q4) =',p[4],'       ');
   writeln('p[5] (p0) =',p[5],'       ');
   writeln('p[6] (p1) =',p[6],'       ');
   writeln('p[7] (p2) =',p[7],'       ');
   writeln('p[8] (p3) =',p[8],'       ');
   writeln('p[9] (p4) =',p[9],'       ');
   writeln;
   writeln('drive factor =',factor);
   writeln('disturbance factor =',mult,'      ');
   writeln;
   writeln('criterion function  =',kf,'      ');
   writeln('criterion function cycles  =',bersom,'   ');
   writeln;
   IF urvast THEN writeln('signal saturated')
   ELSE writeln('                        ');
   IF ervast THEN writeln('servo dropped out of focus')
   ELSE writeln('                       ');
   writeln;
   writeln('newton raphson cycles  =',raphsom);
   writeln('total of criterion function cycles  =',bsomtot);
END; (* of display *)


FUNCTION scurve(d:real;VAR test:boolean):real;

(* in this function the output of the sensor circuit is calcu-
lated, given a distance from the focus of the objective to the
disc surface. Also this function supplies a boolean which in-
dicates whether the distance is within the servo limits (the
servo is 'in-focus'). *)

VAR help:real;
BEGIN
   IF abs(d)>21e-6 THEN BEGIN help:=0;test:=true END
   ELSE IF abs(d)<=7e-6 THEN help:=d*71.3e3*(sqr(143e3*d)-3)
        ELSE help:=-d*sqr(3-143e3*abs(d))/28e-6;
   scurve:=help*10;
END; (* of scurve *)


PROCEDURE berekenKF(p:par;VAR ervast:BOOLEAN;VAR som:REAL);

(* procedure berekenKF calculates the criterion function. To
this purpose, it simulates the servo for a specified number of
samples and feeds it a disturbance. The criterion function is
the summated quadrated error and (weighted) quadrated control
force divided by the number of samples of the simulation. If
the servo drops out of focus, the calculation is aborted. The
gravity is not simulated. *)
```

APPENDIX III
Listing Optimum

```
VAR urvast:BOOLEAN;
    fout: real;
BEGIN
   for m:=0 to 5 do ep[m]:=0;
   for m:=0 to 5 do er[m]:=0;
   for m:=0 to 5 do ur[m]:=0;
   for m:=0 to 5 do up[m]:=0;
   som:=0;j:=0;
   urvast:=FALSE;ervast:=FALSE;
   WHILE NOT (j >= 120)  DO
   BEGIN
      (* the process: *)
      up[5]:=up[4];
      up[4]:=up[3];
      up[3]:=up[2];
      up[2]:=up[1];
      up[1]:=up[0];
      up[0]:=ur[0];
      ep[5]:=ep[4];
      ep[4]:=ep[3];
      ep[3]:=ep[2];
      ep[2]:=ep[1];
      ep[1]:=ep[0];
      ep[0]:=q1*ep[1]+q2*ep[2]+q3*ep[3]+
             p1*up[0]+p2*up[1]+p3*up[2];
      (* the controller: *)
      (* calculate disturbance: *)
      IF j>40 THEN fout:=0
              ELSE fout:=mult*1e-6*sqr(sin(pi*j/40));
      fout:=ep[0]+fout;
      er[4]:=er[3];
      er[3]:=er[2];
      er[2]:=er[1];
      er[1]:=er[0];
      er[0]:=scurve(fout,ervast);
      ur[4]:=ur[3];
      ur[3]:=ur[2];
      ur[2]:=ur[1];
      ur[1]:=ur[0];
      ur[0]:=p[1]*ur[1]+p[2]*ur[2]+p[3]*ur[3]+p[4]*ur[4]
             +p[5]*er[0]+p[6]*er[1]+p[7]*er[2]+p[8]*er[3]
             +p[9]*er[4];
      (* saturation of signal: *)
      IF ur[0]>10 THEN BEGIN ur[0]:=10;urvast:=TRUE END
      ELSE IF ur[0]<-10 THEN BEGIN ur[0]:=-10;urvast:=TRUE END;
      (* update criterion function: *)
      if abs(fout)>7e-6
      then som:=som+sqr(fout*71.3e4*(sqr(143e3*fout)-3))
                                    +factor*sqr(ur[0])
      else som:=som+sqr(fout*sqr(143e3*abs(fout)-3)/28e-5)
                                    +factor*sqr(ur[0]);
```

APPENDIX III
Listing Optimum

```
      j:=j+1;
   END;
   som:=som/j;
   bersom:=bersom+1;bsomtot:=bsomtot+1;
   display(p,som,factor,mult,raphsom,bersom,bsomtot,
                                              ervast,urvast);
END;(* of berekenKF *)


PROCEDURE newraph(i:integer;VAR p:par;VAR S:real);

(* This procedure minimizes the criterion function by varying
one parameter (indicated by the variable 'i'). This is done by
newton-raphson minimization. For this method, the first and
the second derivative (of the criterion function to the para-
meter) are calculated. If the first derivative is large enough
the parameter is adapted. The step is bound to a maximum. If,
after the step the new criterion function turns out to be lar-
ger than the previous one, the step is halved and taken in the
opposite direction (if after 5 steps back, the criterion func-
tion still is larger, the original parameter is recovered). *)

VAR p1,p2:par;
    ddSdp,dSdp,S1,S2,S3,hulp,tus1,tus2 : REAL;
    o:INTEGER;
BEGIN
   (* initialize newton-raphson search *)
   bersom:=0;
   for n:=1 to 9 do p1[n]:=p[n];
   for n:=1 to 9 do p2[n]:=p[n];
   o:=0;
   (* newton raphson: *)
   REPEAT
      S3:=S;
      (* calculate the 1st and 2nd derivative of S to p[i]: *)
      p1[i]:=p[i]+1e-6;p2[i]:=p[i]-1e-6;
      berekenKF(p1,ervast,S1);berekenKF(p2,ervast,S2);
      dSdp:=(S-S2)/1e-6;ddSdp:=(-2*S+S1+S2)/1e-12;
      IF abs(dSdp)>1e-6 THEN
      BEGIN
         (* adapt p[i]: *)
         tus1:=p[i];
         tus2:=S;
         if abs(p[i])<1 then max:=0.05
                        else max:=abs(0.05*p[i]);
         IF ddSdp>1e-6 THEN hulp:=dSdp/ddSdp ELSE hulp:=S/dSdp;
         IF abs(hulp) max THEN hulp:=max*hulp/abs(hulp);
         p[i]:= p[i]-hulp;
         n:=0;
         berekenKF(p,ervast,S);
         if abs(hulp)>1e-6 then
         begin if S>S1 then begin
```

APPENDIX III
Listing Optimum

```
                                p[i]:=p1[i];
                                S:=S1
                              end
                        else if S>S2 then begin
                                            p[i]:=p2[i];
                                            S:=S2;
                                          end;
          end;
          WHILE (S-tus2>1e-8) and (n < 5)  DO
          BEGIN
             p[i]:=(p[i]+tus1)/2;n:=n+1;
             berekenKF(p,ervast,S);
          END;
          o:=o+1;
          IF n=5  THEN begin p[i]:=tus1;S:=tus2;o:=20 end;
       END
       ELSE o:=20;
    UNTIL ((abs(S-S3)<1e-8) OR (o>5));
    IF o>10 THEN write('+') ELSE write('*');
    raphsom:=raphsom+1;
END;(* of newraph *)

BEGIN

    (* INITIALISATIE *)
    ervast:=true;cr:=13;k:=0.1;K1:=45;hr:=1/14650;
    hp:=1/14650;
    (* the weigh-factor has to be entered: *)
    factor:= (*****)
    raphsom:=0;bersom:=0;bsomtot:=0;
    REWRITE(printer,'printer:');
    writeln(printer,'PROGRAM DIGREGoptimum');
    writeln(printer,'startvalues are:');
    (* The start parameters must be entered here: *)
    p[1]:=  (*****)
    p[2]:=  (*****)
    p[3]:=  (*****)
    p[4]:=  (*****)
    p[5]:=  (*****)
    p[6]:=  (*****)
    p[7]:=  (*****)
    p[8]:=  (*****)
    p[9]:=  (*****)
    writeln(printer,'factor = ',factor:9);

    (* calculation of process parameters by matched
       z-transformation: *)
    a:=exp(-hp/t5);
    b:=K1*sqr(hp)/2;
    c:=K1*sqr(t5);
    q1:=2+a;
    q2:=-1-2*a;
```

APPENDIX III
Listing Optimum

```
   q3:=a;
   p1:=b-K1*hp*t5+c*(1-a);
   p2:=b*(1-a)+K1*hp*t5*(1+a)-c*2*(1-a);
   p3:=-b*a-K1*hp*t5*a+c*(1-a);
   writeln(printer,'proces parameters');
   writeln(printer,'q1 =',q1:9);
   writeln(printer,'q2 =',q2:9);
   writeln(printer,'q3 =',q3:9);
   writeln(printer,'q4 =',q4:9);
   writeln(printer,'q5 =',q5:9);
   writeln(printer,'p1 =',p1:9);
   writeln(printer,'p2 =',p2:9);
   writeln(printer,'p3 =',p3:9);
   writeln(printer,'p4 =',p4:9);
   writeln(printer,'p5 =',p5:9);

   (* initialization of the other parameters *)
   S:=0;
   l:=0;
   mult:=45;
   pi:=4*arctan(1);

   (* de optimization: *)
   WHILE ervast DO
   BEGIN
      mult:=mult-5;
      berekenKF(p,ervast,S);
   END;
   writeln(printer,'regelaar parameters');
   writeln(printer,'q1 =',p[1]:9);
   writeln(printer,'q2 =',p[2]:9);
   writeln(printer,'q3 =',p[3]:9);
   writeln(printer,'q4 =',p[4]:9);
   writeln(printer,'p0 =',p[5]:9);
   writeln(printer,'p1 =',p[6]:9);
   writeln(printer,'p2 =',p[7]:9);
   writeln(printer,'p3 =',p[8]:9);
   writeln(printer,'p4 =',p[9]:9);
   writeln(printer,'disturbance factor=',mult:9);
   writeln(printer,'KF =',S:9);
   REPEAT
      S2:=S;
      FOR i:=1 to 9 DO
      CASE i OF    (* choose optimize parameters here: *)
      1,2,5,6,7:newraph(i,p,S);
      END (* of case *);
      l:=l+1;
      write('o');
      (* adapt disturbance factor: *)
      if mult ≠ 100 then
      BEGIN
         mult:=mult+5;
```

APPENDIX III
Listing Optimum

```
            berekenKF(p,ervast,a);
            IF ervast THEN mult:=mult-5;
         END;
      UNTIL ((abs(S-S2)<1e-8) OR (1>10));

      (* report results: *)
      writeln(printer,'resultaat parameters');
      writeln(printer,'q1 =',p[1]:9);
      writeln(printer,'q2 =',p[2]:9);
      writeln(printer,'q3 =',p[3]:9);
      writeln(printer,'q4 =',p[4]:9);
      writeln(printer,'p0 =',p[5]:9);
      writeln(printer,'p1 =',p[6]:9);
      writeln(printer,'p2 =',p[7]:9);
      writeln(printer,'p3 =',p[8]:9);
      writeln(printer,'p4 =',p[9]:9);
      writeln(printer,'disturbance factor=',mult:9);
      writeln(printer,'KF =',S:9);
      close(printer,LOCK);
END.
```

APPENDIX IV
Listing EVMdriver

```
PROGRAM EVMdriver (*VERSION 840214/15.00*);
USES (*$U supply.code*)supplydir,
     (*$U user1:evmprocs.code*)RS232;
(*************** program description ******************)
(*  This program emulates a terminal (RS-232C) to     *)
(*  use with the Evaluation Module of the TMS320      *)
(*  signal-processor. All information sent to the     *)
(*  screen, can also be sent to the printer (log).    *)
(*  In addition to this terminal mode (called trans-  *)
(*  parent), it is possible to store information      *)
(*  from the EVM in a buffer and send the contents    *)
(*  of the buffer to the EVM. The buffer in its       *)
(*  turn can be stored in a diskfile (and vice-ver-   *)
(*  sa). The buffer contains ASCII-coded charac-      *)
(*  ters.                                             *)
(*  Assumptions: EVM echoes all characters.           *)
(*               After ESC, EVM returns to monitor.   *)
(*               All files are to be on disk volume   *)
(*               'USER1:'                             *)
(*  The program is designed to run on a P2500 micro-  *)
(*  computer under the UCSD-p operating system and    *)
(*  uses the system file SUPPLY.CODE and the user-    *)
(*  file EVMPROCS.CODE.                               *)
(*  Declarations between accolades are declared in    *)
(*  the unit RS232 in the file EVMPROCS.CODE.         *)
(*************** initialization ***********************)
CONST quit=113;          (* 'q' *)
      logprint=12;       (* CNTL 'l' *)
      upload=117;        (* 'u' *)
      download=100;      (* 'd' *)
      store=115;         (* 's' *)
      getfile=103;       (* 'g' *)
      transparent=116;   (* 't' *)
      promptm=63;        (* prompt of EVM-monitor: '?' *)
      prompte=42;        (* prompt of EVM-editor: '*' *)
      esc=27;            (* ESC *)
      exitmode = 3;      (* CNTL 'c' *)
      systerm=2;         (* screen and keyboard *)
     (*printer=6;*)
(*TYPE karakter= 0..255;*)
VAR (*prep_par: PACKED ARRAY [0..15] OF karakter; *)
   (* res_par: PACKED ARRAY [0..2] OF karakter;   *)
    (*cntr_par: PACKED ARRAY [0..3] OF karakter;  *)
    (*stat_par: PACKED ARRAY [0..9] OF karakter;  *)
    (*set_print: PACKED ARRAY [0..2] OF karakter; *)
     res_print: PACKED ARRAY [0..3] OF karakter;
    (*read_par: PACKED RECORD                              *)
    (*            par: PACKED ARRAY [0..3] OF karakter;    *)
    (*            ptr: ^karakter;                          *)
    (*            len: integer;                            *)
    (*          END;                                       *)
    (*writ_par: PACKED RECORD                              *)
```

APPENDIX IV
Listing EVMdriver

```
    (*            par: PACKED ARRAY [0..3] OF karakter;  *)
    (*            ptr: ^karakter;                        *)
    (*            len: integer;                          *)
    (*          END;                                     *)
     linepoint,bfpoint,bepoint,index,result
     (*,i,prbfpoint*): INTEGER;
     queued: ARRAY [0..29] of INTEGER;
     pause,dtr, log,charread,txstop,go_on: BOOLEAN;
     buffer: PACKED ARRAY [0..8191] of karakter;
     xon,xoff,choice,chars,chark(*,cr,sp,lf*): karakter;
     fijlname,fijltijp,volname,outstr:STRING;
     stfiled,getfiled:FILE of karakter;
     stfilet,getfilet:FILE of CHAR;
     hup:CHAR;

BEGIN
  init;                (*initialization procedure from RS232*)
  set_print[0]:=27;              (*esc*)
  set_print[1]:=56;              (*8lpi*)
  set_print[2]:=29;              (*16.5cpi*)
  res_print[0]:=27;              (*esc*)
  res_print[1]:=54;              (*6lpi*)
  res_print[2]:=30;              (*10cpi*)
  res_print[3]:=13;              (*cr*)
  xon := 17; xoff := 19; dtr := true;
  choice:=0;chars:=0;chark:=0;cr:=13;lf:=10;sp:=32;
  log:=FALSE;charread:=FALSE;txstop:=FALSE;go_on:=FALSE;
  bfpoint:=0;prbfpoint:=0;bepoint:=0;index:=0;result:=0;

(************** main program loop ****************************)

WRITELN(OUTPUT,'enter mode:^l(og,u(pload,d(ownload,s(tore'
                                ,',g(et,t(ransparent');
UNITREAD(systerm,choice,1);    (* inquire at systerm which *)
WHILE choice ≠ quit DO         (* mode is to be entered *)
BEGIN
CASE choice of

(*$I user1:logprint    *)      (* separate parts of the pro- *)
(*$I user1:download    *)      (* gram stored in separate    *)
(*$I user1:upload      *)      (* text-files.                *)
(*$I user1:getfile     *)
(*$I user1:store       *)
(*$I user1:transp      *)

END                            (*of case*);
WRITELN(output);
WRITE(output,'enter mode:q(uit,^l(og,u(pload,');
WRITELN(output,'d(ownload,s(tore,g(et,t(ransparent');
UNITREAD(systerm,choice,1);
END                            (*of while*)
END                            (*of EVMdriver*).
```

APPENDIX IV
Listing EVMdriver

```
(* EVMdriver module USER1:logprint.text                    *)

(* The functions of the module logprint:                   *)
(* reverse the logstatus                                   *)
(* if the logmode must be on then emit a message with      *)
(* manual acknowledge when the printer is on-line          *)
(* and set the printer to a denser typeface                *)

logprint        :
       BEGIN
          log:=not log;          (* reverse the logstatus *)
          IF log THEN
          BEGIN
             WRITE('log on, is printer on-line?[cr]');
             readln(hup);
             writeln;
             unitwrite(printer,set_print,3);
          END
          ELSE BEGIN
             WRITELN('log off');
             unitwrite(printer,res_print,4);
          END;
       END;(*of logprint*)
```

APPENDIX IV
Listing EVMdriver

```
(* EVMdriver-module USER1:download                              *)

(* This module sends the contents of the buffer (up to  *)
(* the point to which the buffer is filled) to the EVM. *)
(* The download can be aborted by typing ^c and inter-  *)
(* rupted by typing ESC. The download mode will be exit *)
(* automatically when the P2500 receives the EVM moni-  *)
(* tor prompt ('?'). All characters received from the   *)
(* EVM will be displayed on screen and (if the logmode  *)
(* is on) printed out.                                  *)

download        :
       BEGIN
          (************** initialise download **************)
          txstop:=FALSE;
          pause:=FALSE;
          bepoint:=0;
          chars:=0;
          chark:=0;
          i:=0;
          (************************************************)
          WRITELN(output,'loading file to EVM (download)');
          (* eliminating empty lines at end and beginning: *)
          WHILE buffer[bfpoint-2] in [cr,sp]
                                 DO bfpoint:=bfpoint-1;
          WHILE buffer[bepoint] =cr DO bepoint:=bepoint+1;
          WHILE (chars≠promptm) AND (chark≠exitmode) DO
          BEGIN
             IF (bepoint < bfpoint) AND (NOT txstop)
                                 AND (NOT pause) THEN
             (* service serial output (EVM) *)
             BEGIN
                wr_evm (buffer[bepoint]);
                if buffer[bepoint]=cr then pause:=true;
                bepoint:=bepoint+1;
             END;
             (* service keyboard *)
             unitstatus(systerm,queued,1);
             IF queued[0]>0 THEN
             BEGIN
                UNITREAD(systerm,chark,1);
                IF chark = esc THEN
                (* only ESC characters are Xmitted to EVM,
                                          and stop Xmit *)
                BEGIN
                   wr_evm (chark);
                   txstop:=TRUE;
                END;
             END;
             (* service serial input (EVM),
                                 screen (and printer) *)
             if bepoint<bfpoint THEN
```

APPENDIX IV
Listing EVMdriver

```
            BEGIN
               IF st_evm THEN
               BEGIN
                  chars := rd_evm;i:=0;
                  if chars ≠ lf then
                  BEGIN
                     UNITWRITE(systerm,chars,1);
                     (* all characters except ^l are printed *)
                     IF log and (chars≠logprint)
                     THEN fill_prbf(chars);
                  END;
               END
               ELSE i:=i+1;
               if (i>5) then pause:=false;
            END
            ELSE BEGIN
               if st_evm then
               BEGIN
                  IF dtr THEN BEGIN
                    cntr_par[3] := 0 + 10; (* dtr false *)
                    unitwrite (255,cntr_par,4);
                    dtr := false;
                  END;
                  chars := rd_evm;
                  if chars≠lf then
                  BEGIN
                     unitwrite (systerm,chars,1);
                     (* all characters except ^l are printed *)
                     IF log and (chars≠logprint)
                     THEN fill_prbf(chars);
                  END
               END
               ELSE
                 IF NOT dtr THEN BEGIN
                   cntr_par[3] := 128 + 10; (* dtr true *)
                   unitwrite (255,cntr_par,4);
                   dtr := true;
                 END;
            END;
      END;
      END;(*of download*)
```

APPENDIX IV
Listing EVMdriver

```
(* EVMdriver-module USER1:upload                                    *)

(* This module loads a file from the EVM into the buf-             *)
(* fer. It issues a carriage return to the EVM to fi-              *)
(* nish up a command given by the user. Next all cha-              *)
(* racters received from the EVM are loaded into the               *)
(* buffer, displayed on the screen and, if the logmode             *)
(* is on, printed out. The upload is aborted by typing             *)
(* ^c, and interrupted by typing ESC. The upload mode              *)
(* is exit automatically when the P2500 receives the               *)
(* EVM monitor prompt ('?').                                       *)

upload          :
       BEGIN
          (********** initialise upload *************)
          bfpoint:=0;
          chars:=cr;       (* character from RS232 port *)
          charread:=TRUE;
          chark:=0;        (* character from keyboard   *)
          txstop:=false;
          i:=0;
          (*****************************************)
          WRITELN(output,'loading file from EVM (upload)');
          wr_evm(chars);
          chars := rd_evm;
          WHILE (chars≠60) AND (chark≠exitmode)
                           AND (chars≠promptm)
                           AND (NOT txstop) DO
          BEGIN
             (* service screen/printer *)
             IF charread THEN
             BEGIN
                buffer[bfpoint]:=chars;bfpoint:=bfpoint+1;
                UNITWRITE(systerm,chars,1);
                IF log and (chars≠logprint)
                THEN fill_prbf(chars);
                charread:=FALSE;
             END;
             (* service keyboard *)
             UNITSTATUS(systerm,queued,1);
             IF queued[0] > 0 THEN
             BEGIN
                UNITREAD(systerm,chark,1);
                (* only ESC characters are Xmitted to EVM *)
                IF chark=esc THEN wr_evm(chark);
             END;
             (* service serial input (EVM) *)
             IF st_evm THEN
             BEGIN
                if dtr then  BEGIN
                   cntr_par[3]:=0+10; (*set dtr false*)
                   unitwrite(255,cntr_par,4);
```

APPENDIX IV
Listing EVMdriver

```
                dtr:=false;
              END;
              chars := rd_evm;i:=0;
              if chars ≠ lf then charread:=TRUE;
            END ELSE BEGIN
                if not dtr then BEGIN
                   cntr_par[3]:=128+10; (*set dtr true*)
                   unitwrite(255,cntr_par,4);
                   dtr:=true;
                END;
              i:=i+1;
              END;
            IF i>5 THEN txstop:=TRUE;
         END;
         (* output prompt to screen *)
         UNITWRITE(systerm,chars,1);
      END;(*of upload*)
```

APPENDIX IV
Listing EVMdriver

```
(* EVMdriver-module USER1:getfile                              *)

(* This module loads a file from USER1 in the buffer.         *)
(* The buffer is 8 kbytes long. The program checks if         *)
(* USER1 is in drive 2 and if the requested file is on        *)
(* it. The file must be of type TEXT or DATA. From            *)
(* text-files, all comment lines (starting with '*')          *)
(* are deleted and so is comment from column 25 and on.       *)

getfile             :
       BEGIN
          (********* initialise getfile ************)
          bfpoint:=0;
          go_on:=FALSE;
          (*****************************************)
          WRITELN(output,'get diskfile in buffer');
          (* inquire filename *)
          WRITELN(output,'enter filename:');
          READLN(input,fijlname);
(*$I-*)      result:=Read_Volname(5,volname);
          (* check if USER1 in drive #2 *)
          IF (result ≠ 0) OR (volname ≠ 'USER1') THEN
(*$I+*)      BEGIN
             WRITELN(output,
                     'Insert USER1 in drive2, and type <CR>');
             READLN(input,hup);
          END;
          (* check if file present *)
          IF Search_File(5,fijlname,index)=-10 THEN
             WRITELN(output,'file non-existent')
          ELSE go_on:=TRUE;
          result:=Dispose_Dir;
          IF go_on THEN
          BEGIN
              fijlname:=CONCAT('USER1:',fijlname);
              fijltijp:=COPY(fijlname,POS('.',fijlname)+1,4);
              (* in case file of type text, chars have to *)
              (* be converted to ASCII code *)
              IF fijltijp = 'text' THEN
              BEGIN
                 RESET(getfilet,fijlname);
                 WHILE NOT EOF(getfilet) AND (bfpoint < 8192) DO
                 BEGIN
                    REPEAT (* delete comment-lines: *)
                    BEGIN
                       read(getfilet,hup);
                       IF (hup='*') THEN readln(getfilet)
                       ELSE
                       BEGIN
                          buffer[bfpoint]:=ord(hup);
                          bfpoint:=bfpoint+1
                       END;
```

APPENDIX IV
Listing EVMdriver

```
                END UNTIL (hup≠'*');
                (* read the remaining part of the line to
                   the comment: *)
                linepoint:=0;
                WHILE NOT (EOLN(getfilet) OR
                          ((hup='*') AND (linepoint>25))) DO
                BEGIN
                   READ(getfilet,hup);
                   buffer[bfpoint]:=ord(hup);
                   bfpoint:=bfpoint+1;
                   linepoint:=linepoint+1;
                END;
                (* delete spaces at the end of the line: *)
                IF ((hup='*') AND (linepoint 25)) THEN
                BEGIN
                   bfpoint:=bfpoint-1;
                   WHILE buffer[bfpoint-1]=sp DO
                   bfpoint:=bfpoint-1;
                END;
                READLN(getfilet);
                buffer[bfpoint]:=cr;bfpoint:=bfpoint+1;
             END;
             CLOSE(getfilet);
          END;
          (* in case file of type data,
             no conversion is necessary *)
          IF fijltijp = 'data' THEN
          BEGIN
             RESET(getfiled,fijlname);
             WHILE NOT EOF(getfiled) AND (bfpoint < 8192) DO
             BEGIN
                buffer[bfpoint]:=getfiled^;
                GET(getfiled);bfpoint:=bfpoint+1;
             END;
          END;
          CLOSE(getfiled);
       END;
       END;(*of getfile*)
```

APPENDIX IV
Listing EVMdriver

```
(* EVMdriver module USER1:store.text                          *)

(* This module stores the buffer in main memory in a         *)
(* diskfile on volume USER1. The module checks if volu-      *)
(* me USER1 is in drive 2 and if the file to which the       *)
(* buffer must be written is already present, if so the      *)
(* program asks for an acknowledge to overwrite.             *)
(* The file type may be TEXT or DATA.                        *)

store          :
       BEGIN
          (*************** initialise store **************)
          bepoint:=0;
          go_on:=FALSE;
          (**********************************************)
          WRITELN(output,'store buffer in diskfile');
          (* inquire filename *)
          WRITELN(output,'enter filename:');
          READLN(input,fijlname);
(*$I-*)   result:=Read_Volname(5,volname);
          (* check if USER1 in drive #2 (UNITNO=5) *)
          IF (result ≠ 0) OR (volname ≠ 'USER1') THEN
(*$I+*)   BEGIN
             WRITELN(output,
                     'Insert USER1 in drive 2, and type <CR>');
             READLN(input,hup)
          END;
          (* check if file already present *)
          IF Search_File(5,fijlname,index)≠ -10 THEN
          BEGIN
             WRITELN(output,
                     'file already present, overwrite?[y,n]');
             READ(keyboard,hup);
             IF hup='y' THEN go_on:=TRUE;
          END ELSE go_on:=TRUE;
          result:=Dispose_Dir;
          IF go_on THEN
          BEGIN
              fijlname:=CONCAT('USER1:',fijlname);
              fijltijp:=COPY(fijlname,POS('.',fijlname)+1,4);
              (* if file of type text, ASCII codes have *)
              (* to be converted to characters *)
              IF fijltijp = 'text' THEN
              BEGIN
                 REWRITE(stfilet,fijlname);
                 WHILE bepoint < bfpoint DO
                 BEGIN
                    IF buffer[bepoint]=cr THEN
                                        WRITELN(stfilet)
                    ELSE WRITE(stfilet,chr(buffer[bepoint]));
                    bepoint:=bepoint+1;
                 END;
```

APPENDIX IV
Listing EVMdriver

```
            CLOSE(stfilet ,LOCK);
        END;
        (* if file of type data,
                no conversion is necessary *)
        IF fijltijp = 'data' THEN
        BEGIN
            REWRITE(stfiled,fijlname);
            WHILE bepoint < bfpoint DO
            BEGIN
                stfiled^:=buffer[bepoint];
                PUT(stfiled);bepoint:=bepoint+1;
            END;
            CLOSE(stfiled ,LOCK);
        END;
    END;
    END;(*of store*)
```

APPENDIX IV
Listing EVMdriver

```
(* EVMdriver-module USER1:transp.text                    *)

(* This module performs the terminal emulation task and *)
(* is therefore called transparent. It passes all cha-  *)
(* racters (except ^c) transparently between RS232 port *)
(* and keyboard/screen. When the logmode is set, all    *)
(* characters displayed on the screen are also printed  *)
(* out. The logmode is exit by typing ^c. RS232 hand-   *)
(* shaking is done in software with help of the routi-  *)
(* nes in the unit RS232, except for the CTS-check      *)
(* which is done automatically.                         *)

transparent     :
       BEGIN
          (*************** initialise transparent *********)
          charread:=FALSE;
          chark:=0;
          (************************************************)
          WRITELN(output,
                  'transparent mode; exit by typing <CNTL c>');
          WHILE chark ≠ exitmode DO
          BEGIN
             (* handle keyboard input: *)
             IF charread THEN
             BEGIN
                IF chark=logprint THEN
                BEGIN
                   log:=not log;
                   IF log THEN
                   BEGIN
                      WRITE('log on, is printer on-line?[cr]');
                      readln(hup);
                      writeln;
                      unitwrite(printer,set_print,3);
                   END
                   ELSE BEGIN
                      WRITELN('log off');
                      unitwrite(printer,res_print,4);
                   END;
                END
                ELSE wr_evm (chark);
                charread:=FALSE
             END;
             (* handle serial port input: screen and printer *)
             IF st_evm THEN
             BEGIN
                IF dtr THEN BEGIN
                  cntr_par[3] := 0 + 10; (* dtr false *)
                  unitwrite (255,cntr_par,4);
                  dtr := false;
                END;
                chars := rd_evm;
```

APPENDIX IV
Listing EVMdriver

```
            if chars≠lf then (* linefeeds are filtered *)
            BEGIN
               unitwrite (systerm,chars,1);
               IF log and (chars≠logprint)
               THEN fill_prbf(chars);
            END
         END
         ELSE
           IF NOT dtr THEN BEGIN
             cntr_par[3] := 128 + 10; (* dtr true *)
             unitwrite (255,cntr_par,4);
             dtr := true;
           END;
         UNITSTATUS(systerm,queued,1);
         IF queued[0] > 0 THEN
         BEGIN
             UNITREAD(systerm,chark,1);
             charread:=TRUE
         END;
      END;
   END;
```

APPENDIX IV
Listing EVMdriver

```
UNIT RS232;

(* This unit contains all procedures used in conjunction with
the RS232 interface of the P2500 and the printer. *)

INTERFACE

CONST printer = 6;
TYPE karakter= 0..255;
VAR prep_par: PACKED ARRAY [0..15] OF karakter;
    res_par: PACKED ARRAY [0..2] OF karakter;
    cntr_par: PACKED ARRAY [0..3] OF karakter;
    set_print:PACKED ARRAY [0..2] OF karakter;
    prbuf: PACKED ARRAY [0..79] of karakter;
    stat_par: PACKED ARRAY [0..9] OF karakter;
    read_par: PACKED RECORD
                par: PACKED ARRAY [0..3] OF karakter;
                ptr: ^karakter;
                len: integer;
              END;
    writ_par: PACKED RECORD
                par: PACKED ARRAY [0..3] OF karakter;
                ptr: ^karakter;
                len: integer;
              END;
    i,prbfpoint: INTEGER;
    cr: karakter;
    sp,lf: karakter;

PROCEDURE init;
FUNCTION st_evm:boolean;
FUNCTION rd_evm:karakter;
PROCEDURE wr_evm(data:karakter);
PROCEDURE fill_prbf(prchar:karakter);

IMPLEMENTATION

PROCEDURE init;

(* This procedure initialises the serial IO port of the P2500.
It sets the port at 9600 BAUD, 7 data bits, 2 stop bits and
detection of a CTS false condition. Also the DTR line is set
true. *)

VAR i: integer;
BEGIN
  (* prepare *)
  FOR i := 0 TO 15 DO prep_par[i] := 0;
  prep_par[1] := 32;
  prep_par[12] := 128+32+24; (* cts_check + 7_bits + 2_stops *)
  prep_par[13] := 13;
  prep_par[14] := 13;
```

APPENDIX IV
Listing EVMdriver

```
  unitwrite(255,prep_par,16);
  (* reset *)
  res_par[0] := 1;
  res_par[1] := 32;
  res_par[2] := 0;
  unitwrite (255,res_par,3);
  (* control *)
  cntr_par[0] := 2;
  cntr_par[1] := 32;
  cntr_par[2] := 0;
  cntr_par[3] := 128+10; (* set dtr *)
  unitwrite (255,cntr_par,4);
  (* status *)
  FOR i := 0 TO 9 DO stat_par[i] := 0;
  stat_par[0] := 3;
  stat_par[1] := 32;
  (* read *)
  WITH read_par DO BEGIN
    par[0] := 4;
    par[1] := 34;
    par[2] := 0;
    par[3] := 0;
    new(ptr);
    len := 1;
  END;
  (* write *)
  WITH writ_par DO BEGIN
    par[0] := 5;
    par[1] := 33;
    par[2] := 0;
    par[3] := 0;
    new(ptr);
    len := 1;
  END;
END; (* init *)

FUNCTION st_evm;

(* This function inquires the status of the RS232 port. The
function is true when a byte is received from the port. *)

BEGIN
  unitwrite(255,stat_par,10);
  st_evm := stat_par[5] > 0;
END; (* wr_evm *)

FUNCTION rd_evm;

(* This function reads a byte from the RS232 buffer *)

BEGIN
  unitwrite(255,read_par,8);
```

APPENDIX IV
Listing EVMdriver

```
  rd_evm := read_par.ptr^;
END; (* rd_evm *)

PROCEDURE wr_evm;

(* This procedure writes a byte to the RS232 port. *)

BEGIN
  writ_par.ptr^ := data;
  unitwrite(255,writ_par,8);
END; (* wr_evm *)

PROCEDURE fill_prbf;

(* This procedure loads a character in an array (80 charac-
ters long). When the array is full or a carriage return is
stored, the array is written to the printer port. *)

BEGIN
   if not (prchar in [8,15,135,151,137,153])
   then prbuf[prbfpoint]:=prchar;
   if prchar in [8,15,135,151] then prbfpoint:=prbfpoint-1
                               else prbfpoint:=prbfpoint+1;
   if (prchar=cr) or (prbfpoint=79)  then
   BEGIN
      if prbfpoint>1 then unitwrite(printer,prbuf,prbfpoint);
      prbfpoint:=0;
   END
END; (* of fill_prbf *)

END.
```