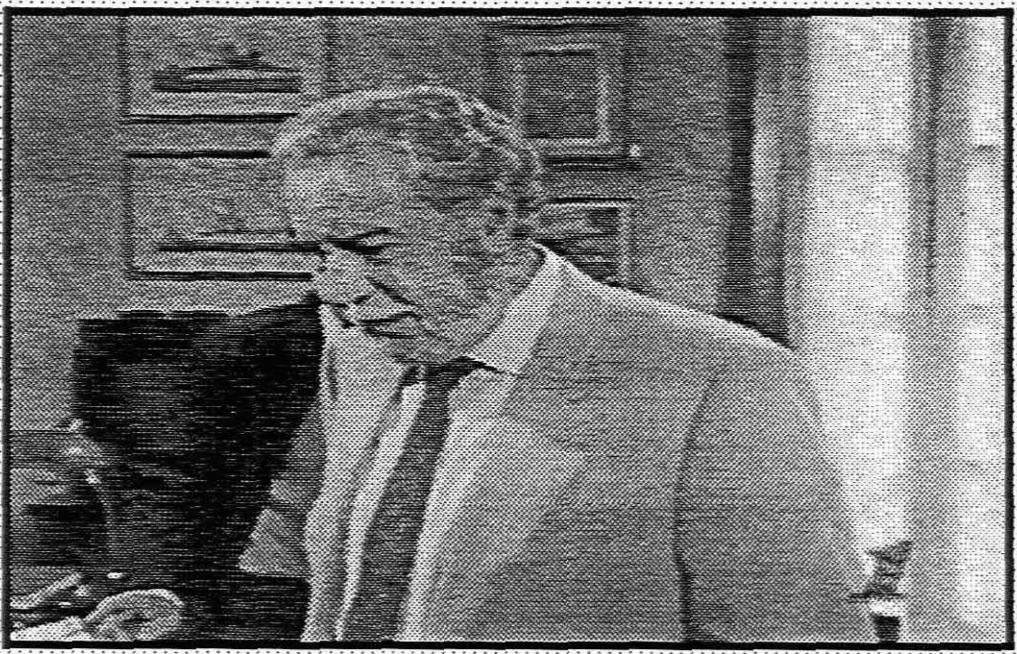
5)

The development of a Video Frame Grabber for a PC



By N.P.Stodart

Thesis submitted in partial fulfillment of the requirements for the Master Diploma in Technology to the Department of Electrical Engineering (light current) at the Cape Technikon

Cape Town
South Africa
October 1993

Declaration

I declare that the contents of this thesis represents my own work and that the opinions contained here are my own. It has not been submitted before for any examination at this or any other institute.

N.P.Stodart

NStodart

(Signature)

Abstract

This thesis describes the design and development of a computer vision system. The system (Video Frame Grabber) will give PC-Users the potential to capture any visual image into the memory of a computer. This computer intelligible image opens the way for new development in computer photography, Image recognition and Desktop Publishing.

Opsomming

Hierdie verhandeling beskryf die ontwerp van 'n stelsel wat rekenaarsig moontlik maak. Die stelsel (Video Raam Stoorder) sal dit vir die PR- gebruiker moontlik maak om enige sigbare beeld vas te lê in die geheue van 'n rekenaar. As die rekenaar eers die beeld verstaan open dit die weg na nuwe ontwikkeling in rekenaar fotografie, beeld herkenning en persoonlike rekenaar uitgewery.

Acknowledgements

Special thanks to the following people who made it possible to complete this Master Diploma project.

- Mr P.H. Kleynhans for all the arrangements for a room to work in and his friendly advice during my study period.
- Mr C.P.Walser as my Mentor.
- My parents for their moral and financial support during the whole of my studies.
- All the staff from the Electrical Engineering department of the Cape Technikon for their interest and support during the development of this project
- All my friends for their moral support during the development of this project.
- My Heavenly Father who gave me the insight to complete this project.

TABLE OF CONTENTS

TABLE CONTENTS (Continued) **PAGE**

TABLE CONTENTS (Continued) **PAGE**

TABLE CONTENTS (Continued) **PAGE**

TABLE CONTENTS (Continued) PAGE

TABLE CONTENTS (Continued) **PAGE**

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued) **PAGE**

LIST OF ILLUSTRATIONS (Continued) **PAGE**

DRAWING

LIST OF ILLUSTRATIONS (Continued) **PAGE**

1. INTRODUCTION

The personal computer made its appearance in the early 1980's. From this time the PC has developed to a sophisticated technical wonder. The minimum standard for computers is getting higher day by day.

Along with the development of the PC has come a great development in PC accessories such as Sound Blaster cards, colour laser printers, high resolution colour monitors and many more.

The ultimate in development will be computer vision. Here follows a quick overview of the current techniques in use to achieve computer vision.

1.1 Hand Image Scanners

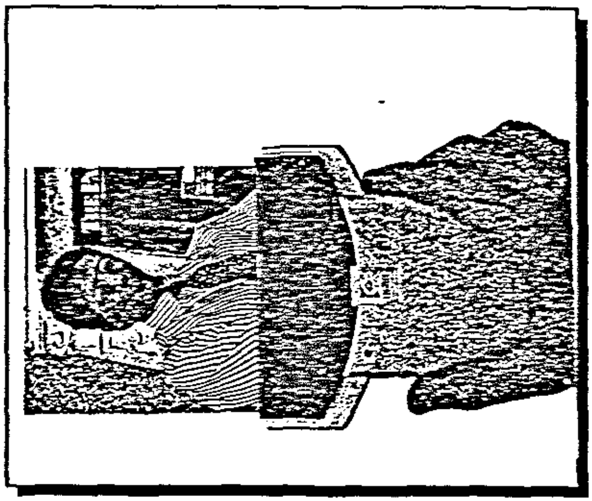


Figure 1.1 Handheld Scanner

This system consists of a handheld device with optical sensors which scans in an image while the operator is slowly moving the scanner over a photo or any document. A analog signal from the optical sensors is digitized and stored in the computer's memory. The digital image can be manipulated by a PC to achieve any desired output.

1.2 Full Page Image Scanners

This system uses the same scanning technique as the handheld image scanner. The only difference is that the graphic image is inserted into the scanner in the same manner as you would insert a paper into a photocopier machine.

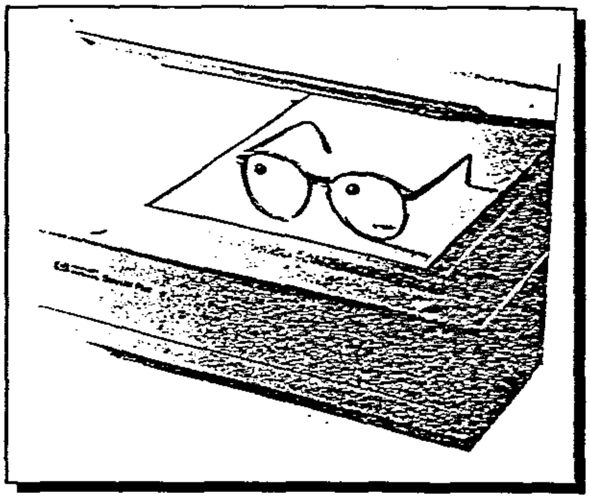


Figure 1.2 Full Page Scanner

1.3 Photographic Scanners

The photographic Scanner uses a special adapted 35 mm camera to take a picture of the image to be imported into the PC. A film is not used but the image is developed onto an array of CCD (Charge coupled device) sensors in the camera. This image stored on the CCD Array may then be downloaded to a PC for further processing.

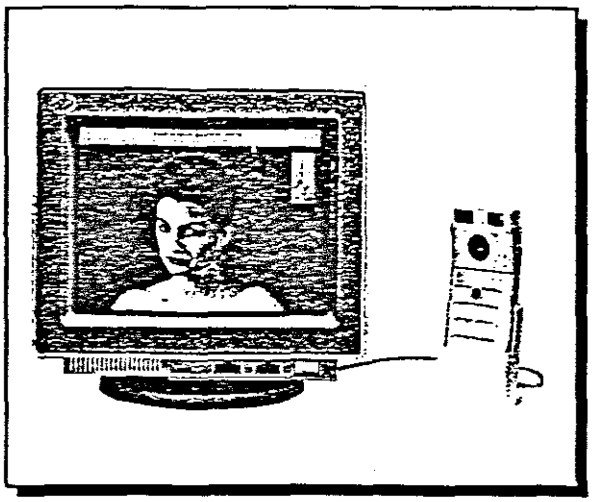


Figure 1.3 Photo-Scanner

1.4 Video Scanners

In a video scanning system the analog video signal from a video camera is digitized and imported into the PC. There are two methods in use.

1.4.1 Slow scan technique

This technique uses more than one video frame to construct a single image. For this method the video signal must be stationary for at least a few seconds before it may be captured and downloaded to a PC.

1.4.2 Real time video frame grabber

A digitized image is constructed by using only one video frame of the video source. With this method it would be possible to capture events directly without any delay.

A very large variety of image scanners are available but they are very expensive and are only available to graphic design engineers. This leaves a large gap in the private sector for a scanner for small business and home computer users. Based on the above survey an image scanning system was designed and constructed by using a real time video frame grabber.

2. SYSTEM INVESTIGATION

As seen from the introduction there are a number of methods in use to achieve computer vision and some of the methods are improving day by day. In this chapter a review is given of the reasons for using a video frame grabber for a capturing unit in the proposed design system.

2.1 Why use a Video Frame Grabber?

The video frame grabber is the only scanner which uses standard equipment such as video cameras and video machines. The rest of the scanners use devices specially designed for computer image scanning. Home video photography has developed greatly in recent years. These developments may be used in full when a video frame grabber is being used.

There is no need for the development of any photographs or any waiting time before a desired output is achieved. It can store any sequence of photographs on a normal video tape. With an average of 5 seconds per still picture needed a total of 2160 pictures may be stored on a single 180 minute VHS video tape. To store the same number of digital pictures onto a hard disk will take (2160 * ± 480 Kilobytes per picture) = 1.0125 Giga bytes of space. The first storage method mentioned is the most cost efficient way of storing a very large number of pictures.

Shoot any scene now and download it at a later stage to a computer.

2.2 Proposed design system

The frame grabber consists of 3 main sections as shown below in the basic block diagram.

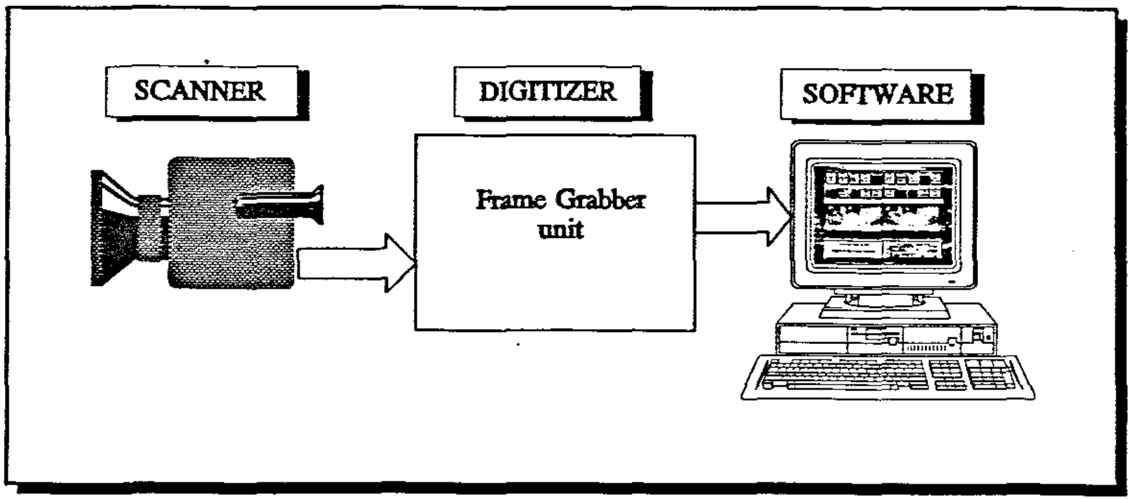


Figure 2.1 Block diagram of the Frame Grabber system

2.2.1 Scanner Section

In figure 2.1 a video camera is shown as the scanner for this unit. The scanner section may be any of the following standard video sources:

- Video camera with a composite PAL video output.
- Video machine with a composite PAL video output.
- Television with a composite PAL video output.

The quality of the picture scanned is directly related to the video quality obtained from the video source.

2.2.2 Digitizing Section

The digitizer unit takes a standard analog PAL video signal and converts it into a digital signal. Only one video frame is digitized and stored in the internal static ram of the digitizing unit. A very high speed analog to digital converter is used for the digitizing of the video signal. The whole process of how a frame is captured and digitized will be described in the circuit description section in chapters 8 to 13.

2.2.3 Software Section

The digitized analog signal needs to be converted into picture information to be displayed on the screen. The detecting of this digital video information is done by a software program written with the high level programming language Turbo Pascal. A thorough investigation into the conversion from a digital video signal to an actual picture will be discussed in chapter 3. A complete listing of the Turbo Pascal program FR30.PAS may be found in appendix B.

3 SOFTWARE DEVELOPMENT

Turbo Pascal is one of the few high level languages to offer build in routines for the handling of high resolution graphics screens. The following main criteria were taken into consideration before the program was written.

3.1 Design Considerations

- Be compatible with the following graphics adapters

Graphic Adapters	720 *348	640 *350	640 *400	640 *480	800 *600	1024 *768
Hercules	B/W	-----	-----	-----	-----	-----
EGA	-----	16	16	-----	-----	-----
VGA	-----	256	256	256	16	16
Super VGA	-----	256	256	256	256	256

- Access 512 k of external Static RAM in the Frame Grabber unit
- Fully mouse operated in all the graphics modes
- 64 Grey scales on the standard VGA Graphics Adapter
- Show the digitized video signal as
    - * Graph (Normal oscilloscope display)
    - * Picture (Grey or Black and White)
- Software control over picture brightness and contrast
- Convert the captured image to the PCX graphics format for easier manipulation by standard desktop publishing software.

3.2 Hardware / Software trade off

In designing this section certain specifications were laid down by the hardware section.

3.2.1 Sampling Rate

A sampling rate of 12 MHz is used to sample the analog video signal. This sampling rate was a trade off between good resolution and the cost of static ram needed to store the sampled signal. According to Nyquist sampling theory the sampling rate must be at least twice the maximum frequency to be sampled. A television video signal has a bandwidth of 5,5 MHz which set the minimum sampling rate to at least 11 MHz.
To sample both ODD/EVEN fields of a video signal the sampling period must be exactly 40 ms.

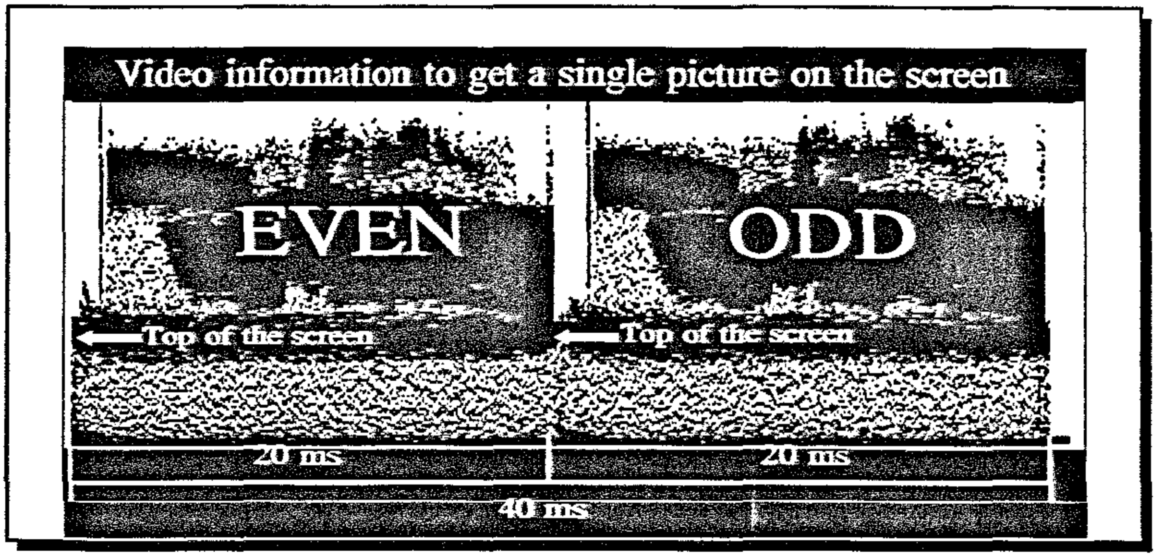


Figure 3.1 A full video frame

The graph in Fig 3.2 may be used for a quick look-up of the memory size needed to sample a full video picture.

The memory size needed may be calculated as follows:

```
Memory needed = Sample period * Sampling rate
              =      40 ms     *     12 Mhz
              =  480000 bytes
```

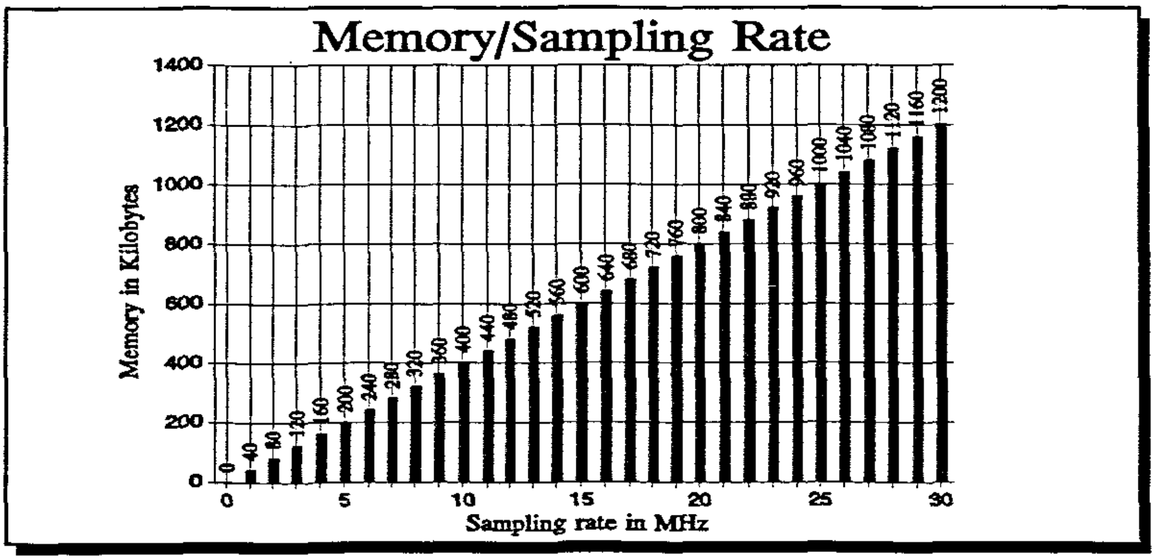


Figure 3.2 Sampling Rate to Memory Size comparison

It can be seen from Figure 3.2 that the memory size needs to be doubled for a sampling rate of twice the frequency to store the same amount of picture information.

3.2.2 Data format

The data format is determined by the bit resolution of the analog to digital converter. For the frame grabber unit a 7-bit Flash A-D Converter (ADC 207MC) is used. For accurate decoding of the video information a single bit is used to store only sync information. 8-Bit parallel data is used for communication between the PC and the frame grabber unit. The format of this 8-bit data is shown in Figure 3.3. D0..D6 is used for the digitized video information. D7 is used for sync information only.

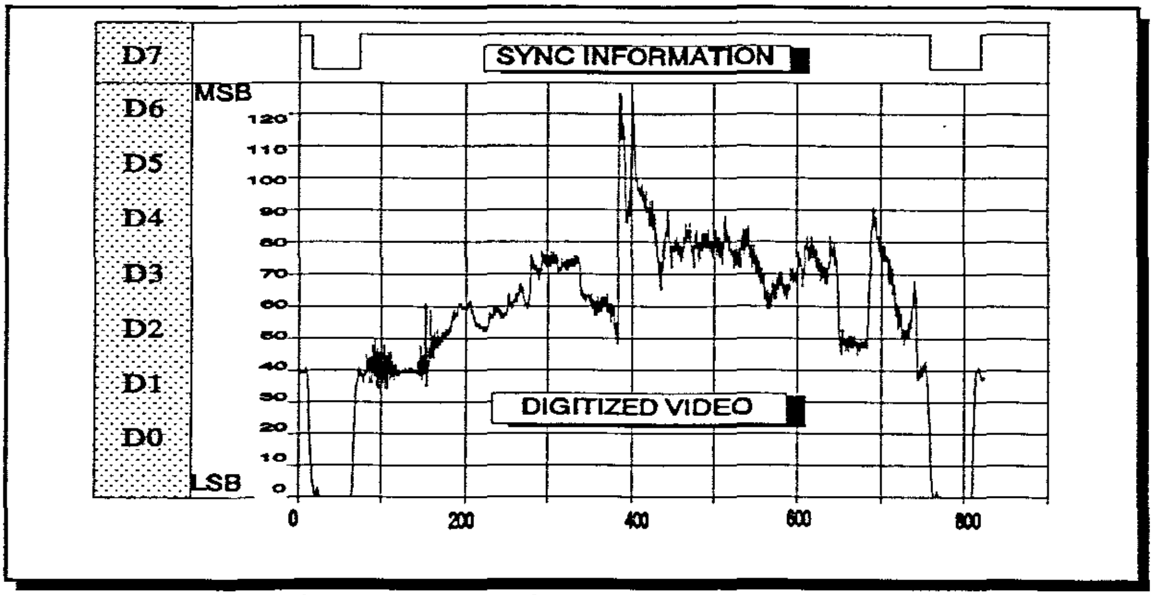


Figure 3.3 8-Bit data configuration

The separate bit for sync information is used to get an accurate indication of when sync is occurring. The sync information may be detected from the digitized analog signal by software routines, but it is not as accurate as the digital sync information obtained from a sync separator (LM1881) due to level triggering on a non specific value.

3.2.3 Addressing Mode (Frame Grabber Unit)

To access the data stored in the Static Ram a special format is used. A single picture consist of 480000 bytes of digital information. To access a single byte in a memory block of 512 kilobytes a unique 19 bit address is needed. The communication between the PC and the Frame grabber unit is limited to an 8 bit parallel format. The 19 bit address needs to be divided into 3 to be transferred over the 8 bit link.

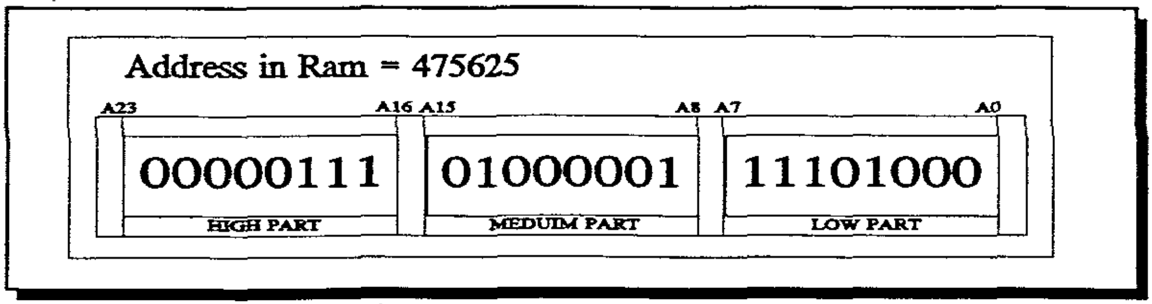


Figure 3.4 Addressing Mode

A description of handshaking and parallel data communication may be found in the hardware description of the address decoder section.

The following procedure MUST be followed to gain access to the frame grabber unit.

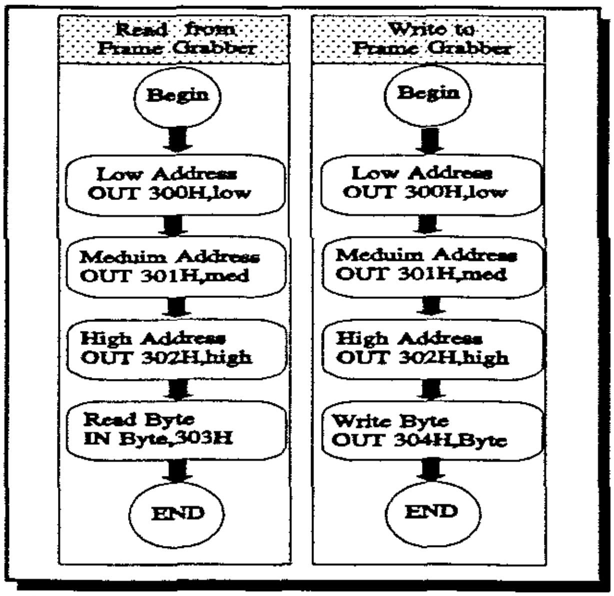


Figure 3.5 Data Management

3.2.4 The VGA Palette

Standard VGA graphics adapters can display 256 colours simultaneously on the screen. Every pixel on the screen may have a unique colour value. The colour values are determined by the settings in the VGA-Palette. The VGA palette is an array of 256 palette colours. The format of a single palette colour is shown below.

Palette Colour[0] =	R [Value 0..63]	Red value for pixel
	G [Value 0..63]	Green value for pixel
	B [Value 0..63]	Blue value for pixel

The values of the RGB settings will give a unique colour to Palette colour[0]. With this method any 256 colours may be used out of a range of 262144 colours.

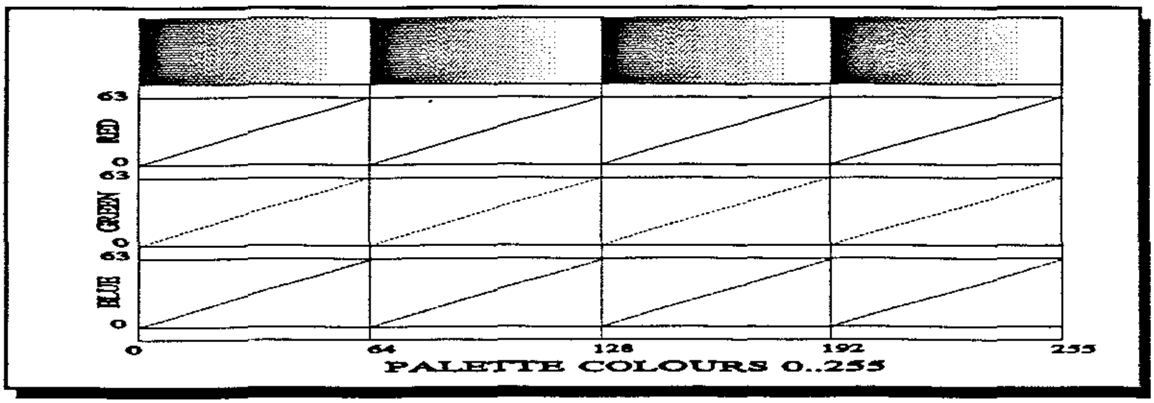


Figure 3.6 Grey Scale palette settings

Figure 3.6 shows the value for the RGB registers to get 64 Grey shades with a standard VGA graphics adapter.

3.3 Video detection

The video detection is done by a software routine written in Turbo Pascal. A full listing of the software program FR30.PAS is shown in Appendix B. The procedure "ShowGraphArray" in Appendix B is used for video detection.

A video signal consists of picture information as well as information for horizontal and vertical synchronisation of the picture on the screen. The horizontal and vertical sync information is detected by software and is used for placing the picture information in the correct screen position.

The picture information in the video signal is detected and converted into pixel information to be displayed. In figure 3.7 a portion of the video signal with sync and video information is clearly shown.

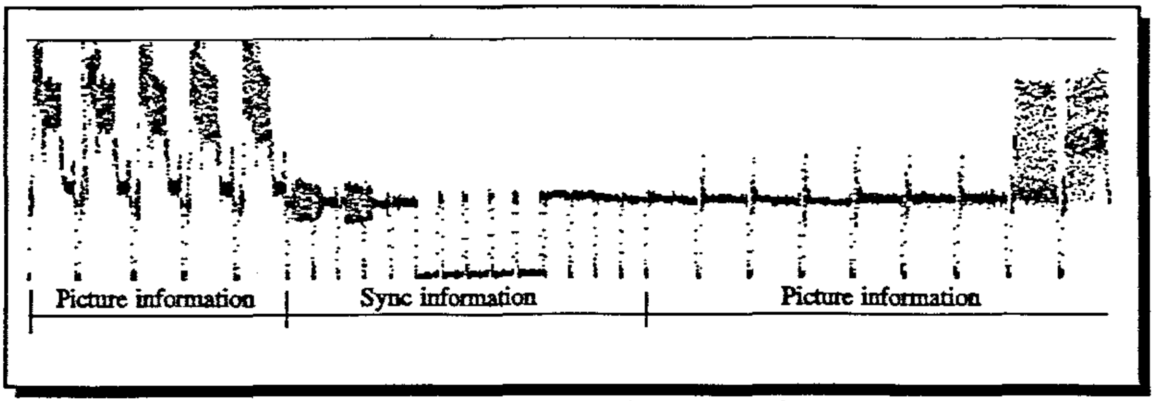


Figure 3.7 Sync and Picture information

The hardware section is designed to store a full video frame starting with the EVEN field for every captured frame. This ensures that the starting sync pulses for every captured frame are exactly similar.

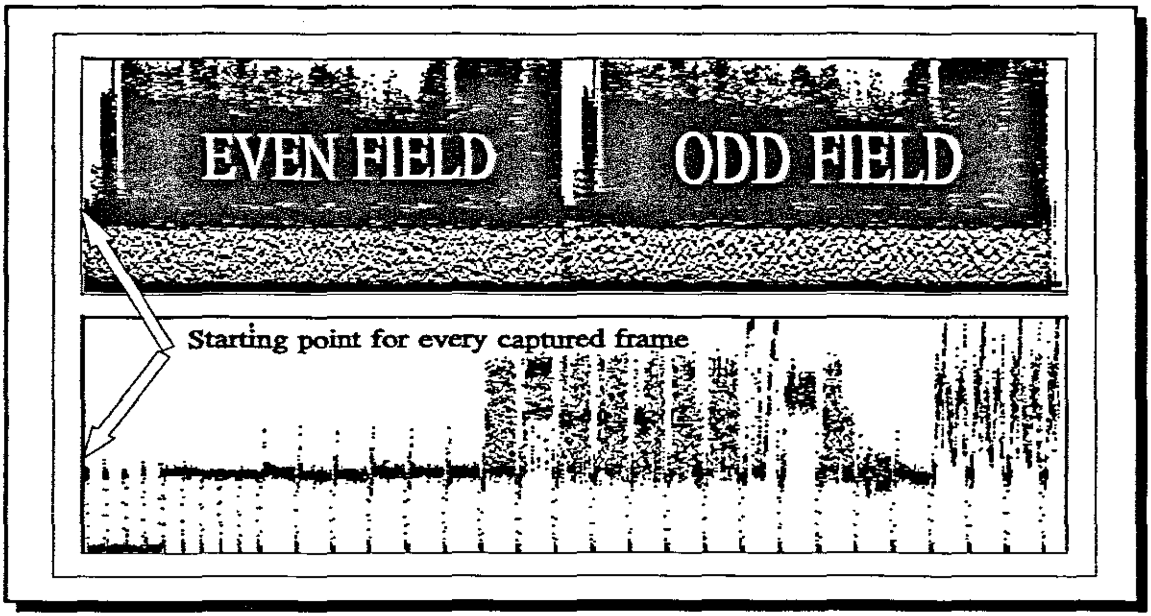


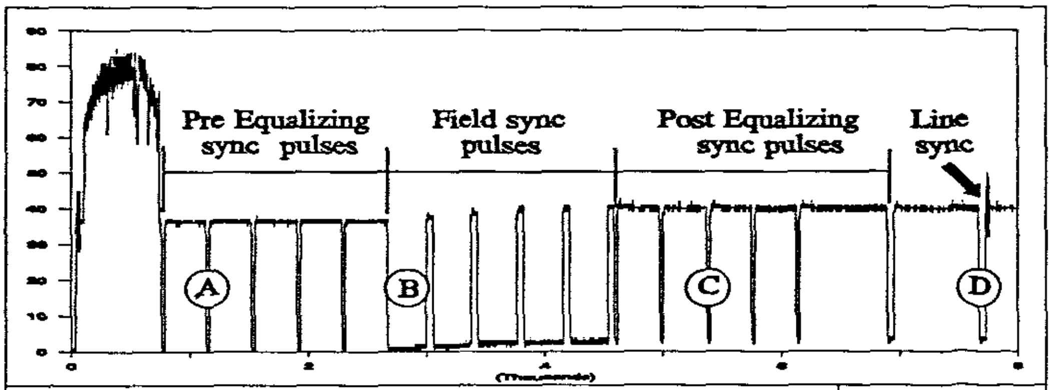
Figure 3.8 Starting point of each captured frame

The software needs to distinguish between the following different types sync pulses:

- Line sync pulses
- Pre equalization sync pulses
- Post equalization sync pulses
- Field sync pulses

These sync pulses must not be displayed on the screen but are used for the correct placing of picture information on the screen. The width of the sync pulses is used for sync recognition and identification. A software counter (SynchCount) is used to determine the low time of the sync pulse. The low time for each of these sync pulses is fixed to within a few microseconds.

Figure 3.9 shows all the sync pulses and their fixed lengths in microseconds.



	Samples at 12 MHz
● A Pre Equalizing pulses Low time = 2.35 microseconds	28
● B Field Sync pulses Low time = 27.3 microseconds.	328
● C Post Equalizing pulses Low time =2.35 microseconds	28
● D Line sync pulse Low time = 4.7 microseconds	56

Figure 3.9 Sync pulse specifications

The sampling rate for the frame grabber unit is fixed at 12 MHz. Time may be directly converted to samples taken by the following formula.

Samples taken = Time in microseconds x Sampling rate in MHz
= 4,7 x 12
= 56,4 Samples for the line sync low level duration

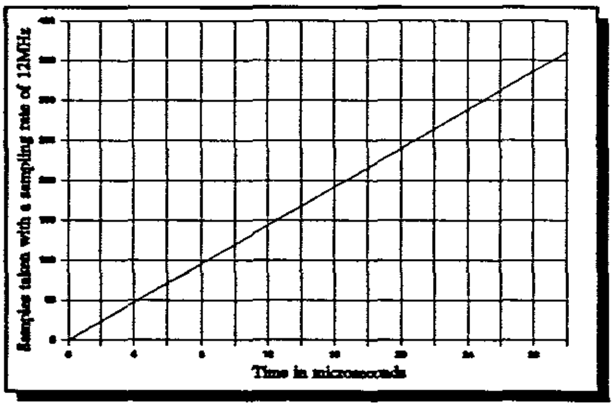


Figure 3.10 Samples - Time

Figure 3.10 may be used for a quick determination of the samples used for a specific time in microseconds.

In figure 3.11 it can clearly be seen when the sync counter and horizontal counters starts in the software program. The value for the sync counter reached at line sync detection, is used for detecting the start of a video line. (Normally this gives a count of between 57-59)

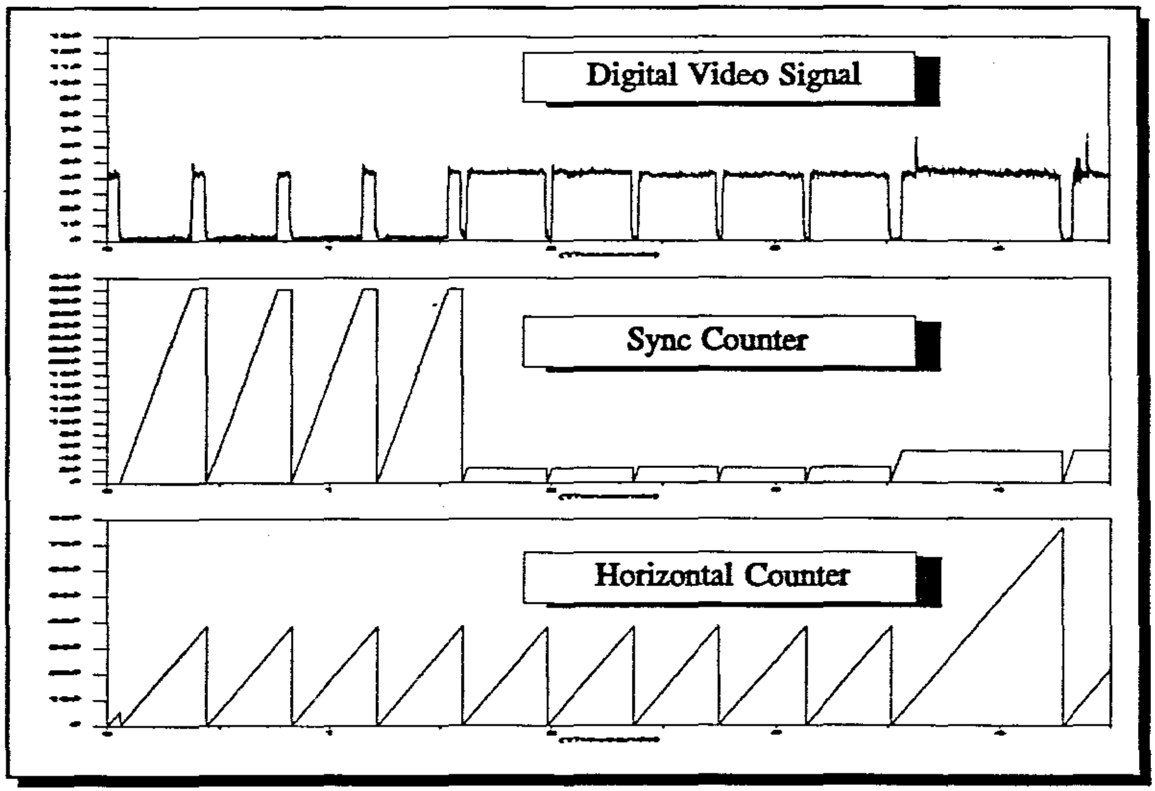


Figure 3.11 Software counters for sync recognition

The horizontal counter is used to synchronise every picture line with the start of a line sync pulse. The starting point (falling edge) of a line sync was found to be more stable than the rising edge, for starting a horizontal position counter to place a pixel on the screen.

The first few video lines consist of TV test signals or are blank for a video camera signal. These first few video lines must not be displayed but must be skipped. The video picture starts at line 24 as shown in figure 3.12

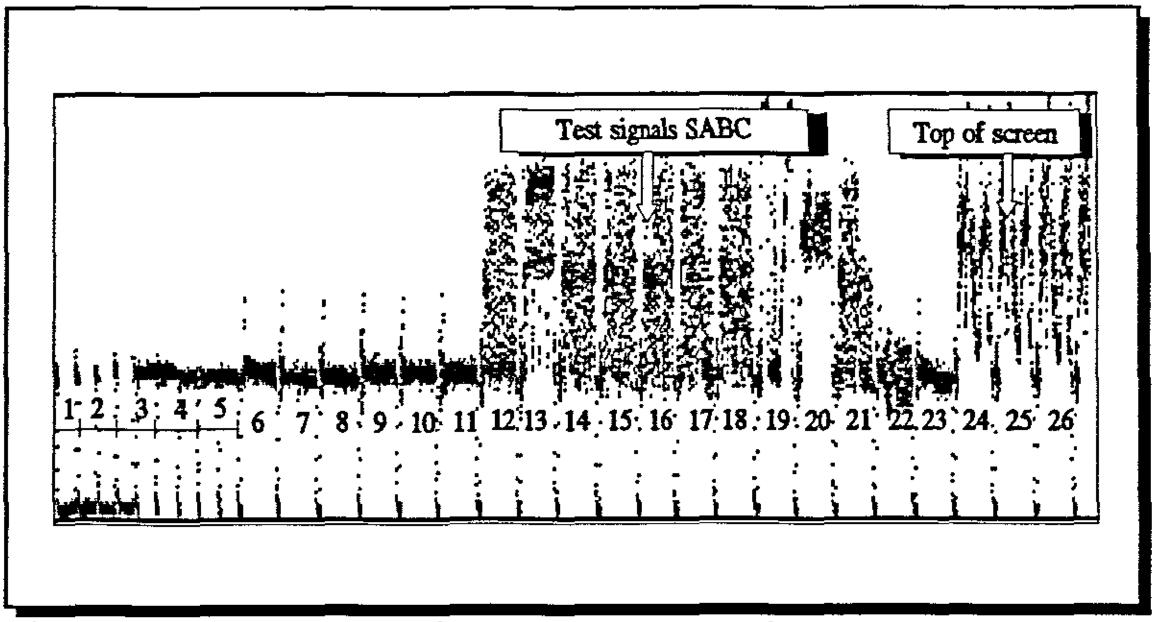


Figure 3.12 Blank lines on the top of the screen

These blank or test lines are present in both the ODD and EVEN fields of the captured data. A video line is detected when a valid line sync pulse is found and the value for the Topcounter (Lines to be skipped at the top of the screen) is bigger than 23.

As seen from figure 3.13 the video counter starts at a point where the horizontal counter reaches the value of 150 or bigger. The video counter starts at this point to skip the colour burst information found at the beginning of each video line.

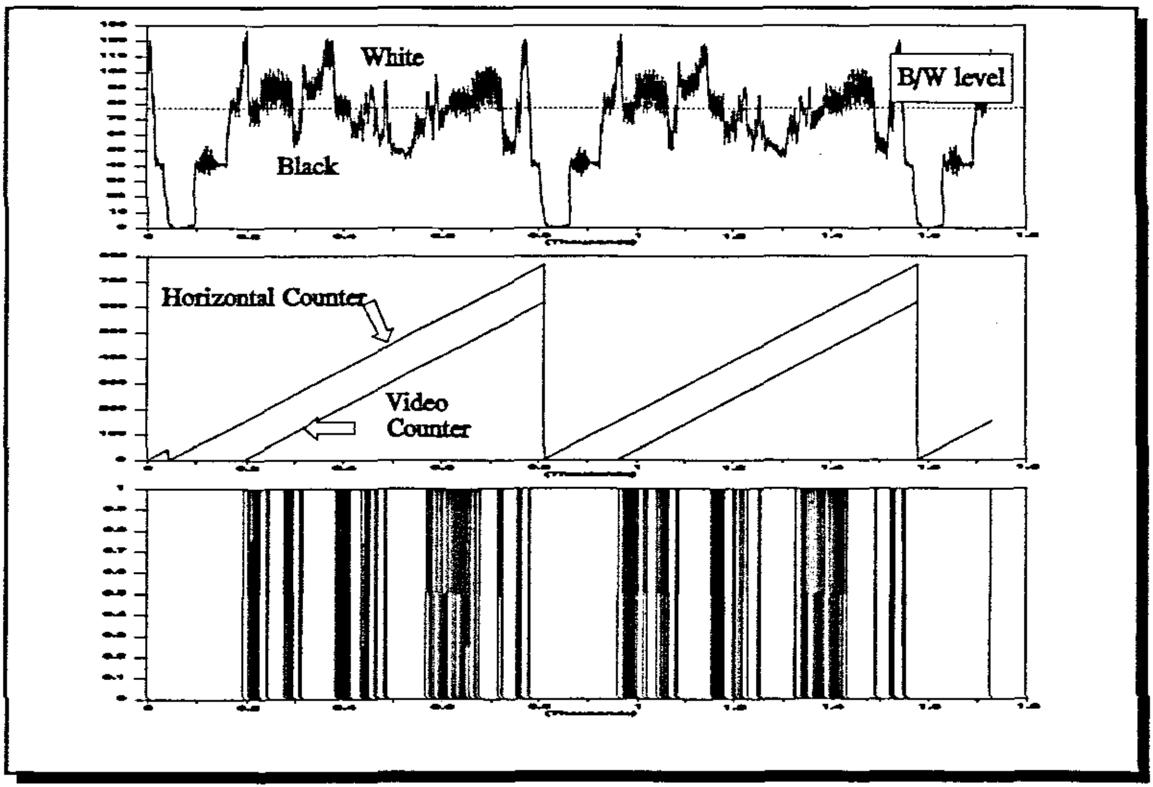


Figure 3.13 Black and white pixel conversion

3.3.1 <u>Black and White detection</u>

A fixed level is set for the variable BW level. For sampled values bigger than this BW level the pixel colour is set to white and for lower levels the pixel colour is set to black. Figure 3.13 show one video line converted to a black and white picture.

3.3.2 Grey scale detection

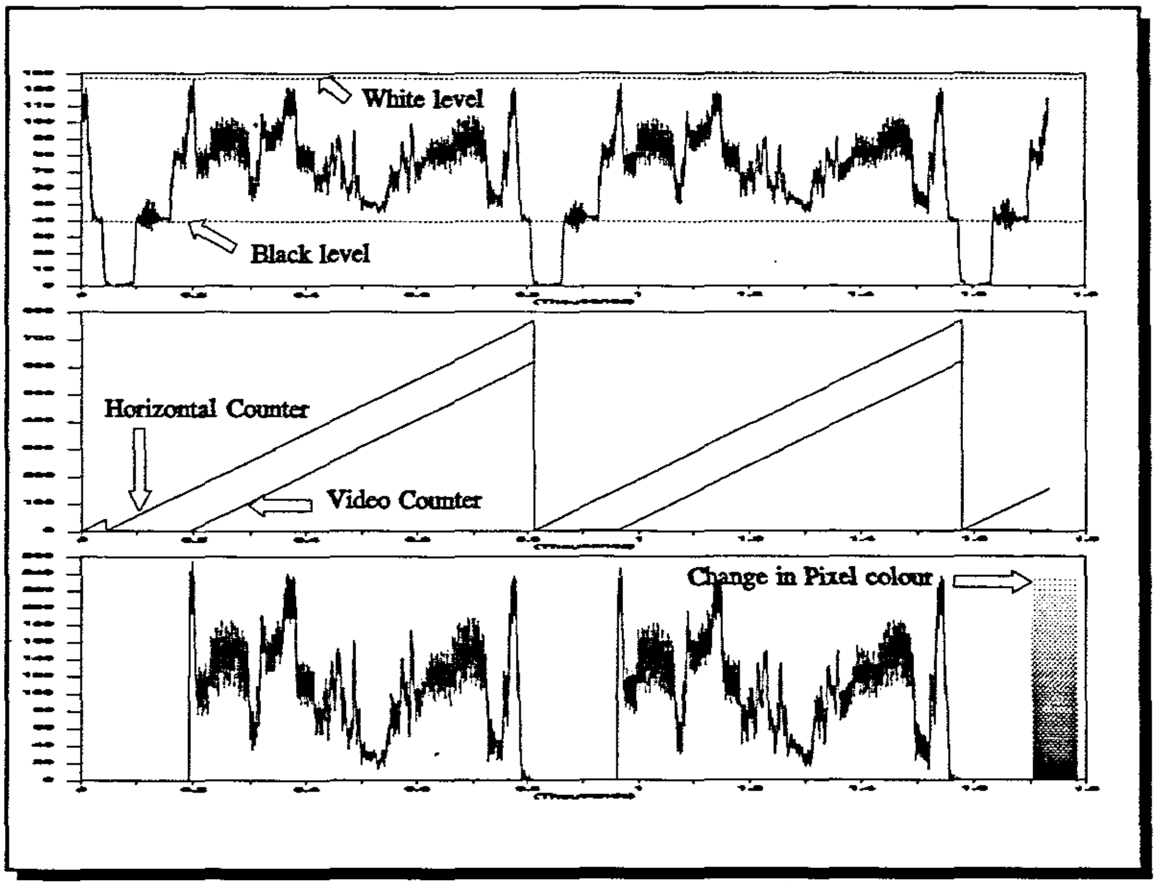


Figure 3.14 Grey Scale colour conversion

For the grey scale conversion the picture information is scaled between the white level and the black level settings. The value is converted to a colour value with the following formula:

$$\text{Pixel Colour} = \frac{(\text{ Current Byte- Black Level })}{(\text{ White level - black level})} \times \text{Max Colours}$$

$$= \frac{75-40}{127-40} \times 255$$

$$= 102.586 \quad (\text{ This value is rounded to its nearest whole number })$$

The value for the variable max colours is determined by the current graphics adapter in used is may be 1,4,16 or 256. Figure 3.14 shows a video line converted to 256 grey scales for displaying on a VGA Graphics adapter. The specifications as given in Figure 3.15 were used for the detection of a single video line.

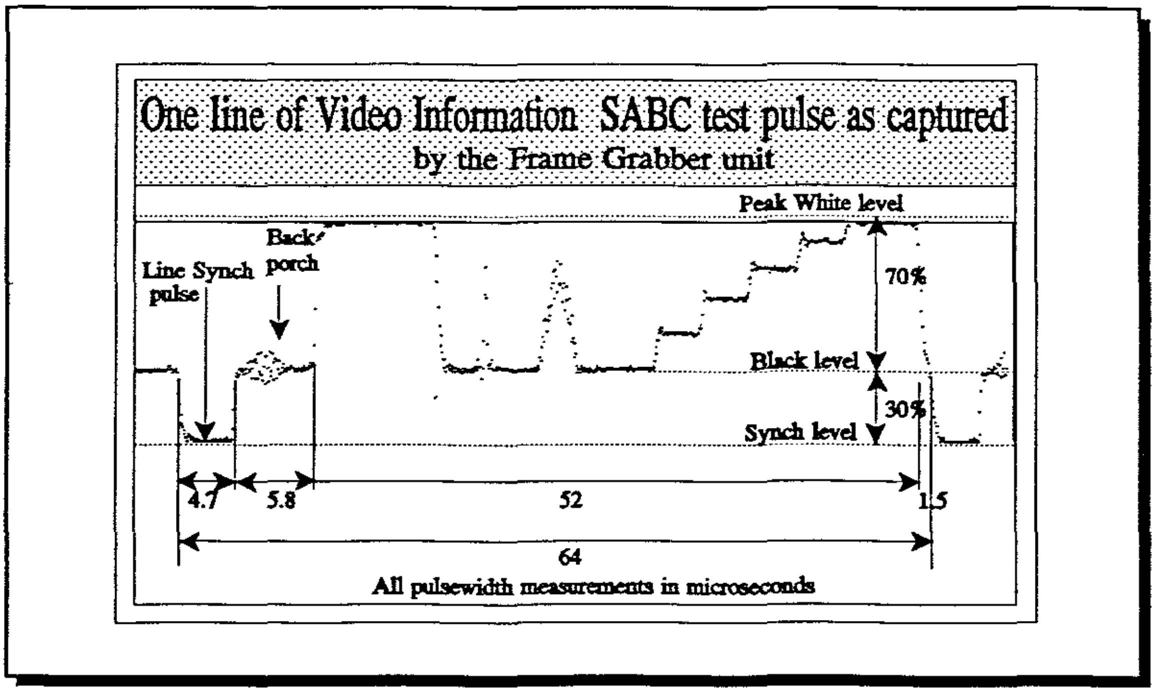

One line of Video Information SABC test pulse as captured
by the Frame Grabber unit
Peak White level
Back porch
Line Synch pulse
70%
Black level
30%
Synch level
4.7
5.8
52
1.5
64
All pulsewidth measurements in microseconds

Figure 3.15 SABC Line specification as used for detection

Figure 3.16 shows a full Block diagram of HOW to detect a video line.

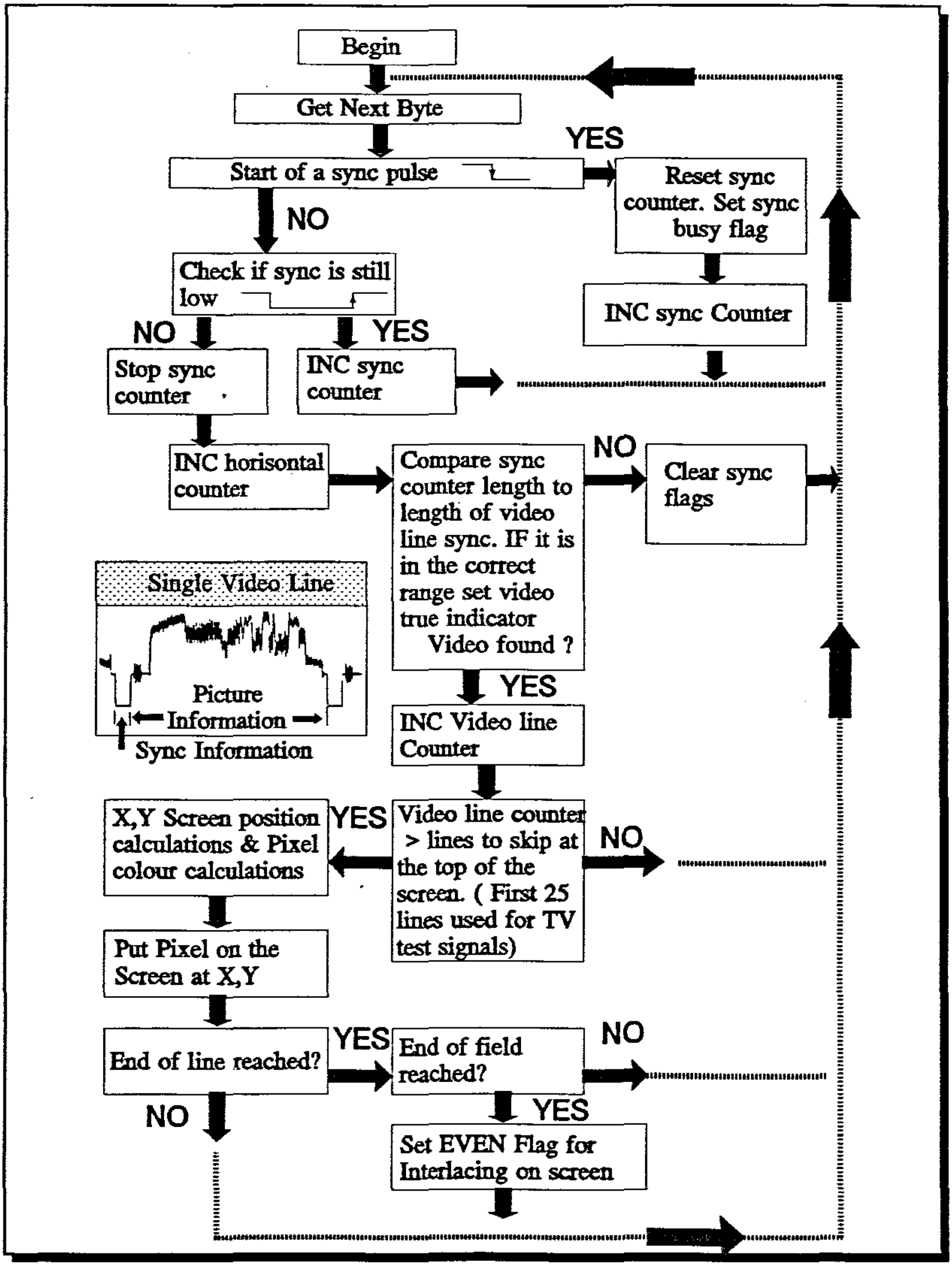

Begin
Get Next Byte
YES
Start of a sync pulse
Reset sync counter. Set sync busy flag
NO
Check if sync is still low
INC sync Counter
NO
YES
Stop sync counter
INC sync counter
INC horisontal counter
Compare sync counter length to length of video line sync. IF it is in the correct range set video true indicator Video found ?
NO
Clear sync flags
Single Video Line
Picture Information
Sync Information
YES
INC Video line Counter
X,Y Screen position calculations & Pixel colour calculations
YES
Video line counter > lines to skip at the top of the screen. (First 25 lines used for TV test signals)
NO
Put Pixel on the Screen at X,Y
End of line reached?
YES
End of field reached?
NO
NO
YES
Set EVEN Flag for Interlacing on screen

Figure 3.16 Step by Step Video Conversion

3.4 PCX-File format

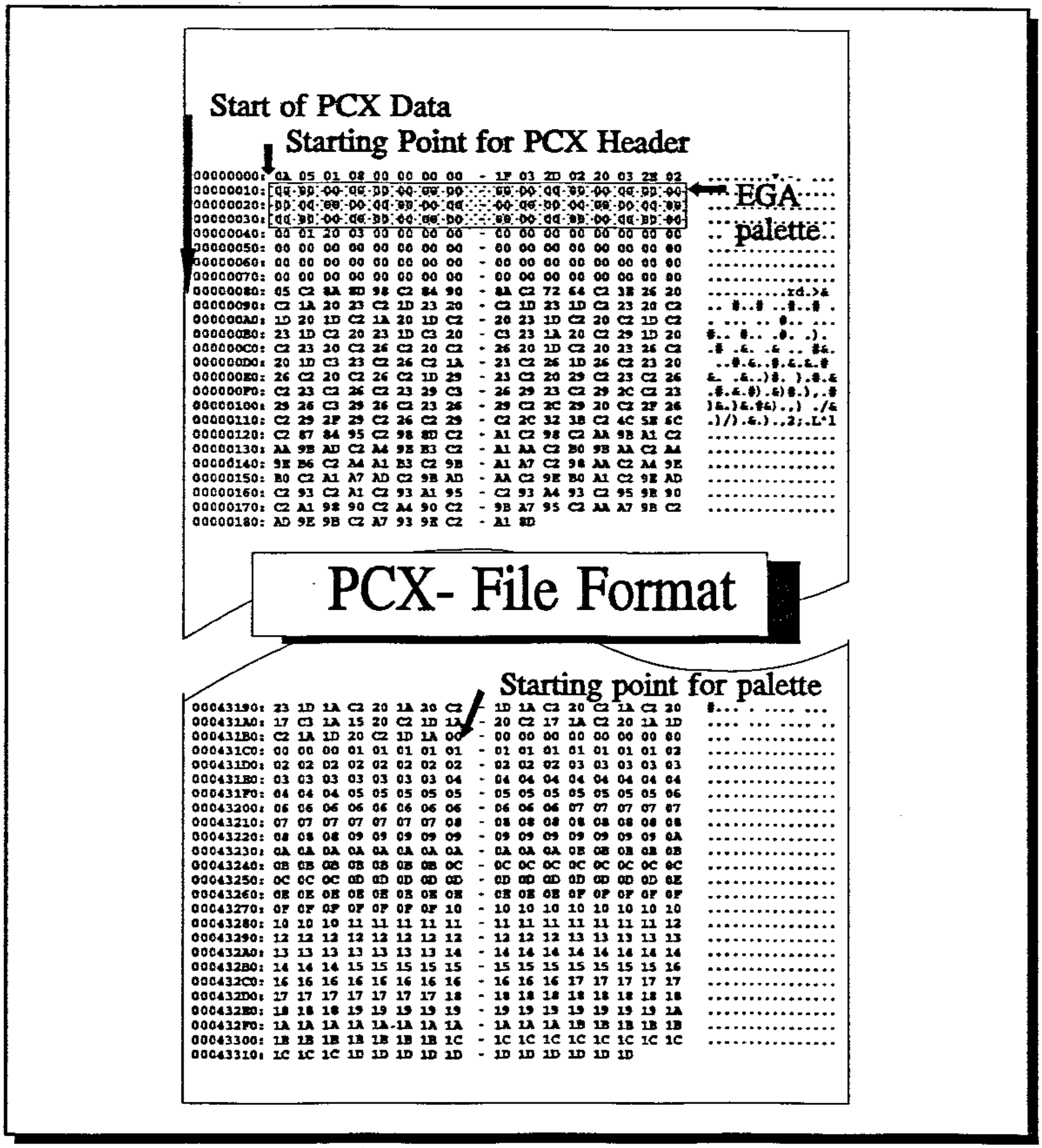
The PCX file format is one of the single most used graphic formats in the DOS world. It is the native format of the PC-Paintbrush series of programs and was developed by ZSoft Corporation. Most of the major graphics programs allow the importing and/or exporting of graphic screens in this format.

This format is used to make the frame grabbing software FR30.EXE compatible with other graphics programs. The PCX file format allows the storing of several different screen formats to a file for usage with graphic programs.

The PCX file consists of 3 major sections

- **Header** This section gives descriptive information on the size of the image, screen mode to use and palette information for EGA and CGA graphic adapters.
- **Data Section** This section contains the data to construct a single picture. The data in this section is crunched for saving storage space.
- **VGA Palette** Contains the settings for the VGA colour palette to achieve 256 colours.

See Figure 3.17 for the file format as used for a VGA PCX file.



```
00000000: 0A 05 01 08 00 00 00 00  - 1F 03 2D 02 20 03 2E 02   ..........-. ...
00000010: [illegible]                                           ................
00000020: [illegible]                                           ................
00000030: [illegible]                                           ................
00000040: 00 01 20 03 00 00 00 00  - 00 00 00 00 00 00 00 00   ................
00000050: 00 00 00 00 00 00 00 00  - 00 00 00 00 00 00 00 00   ................
00000060: 00 00 00 00 00 00 00 00  - 00 00 00 00 00 00 00 00   ................
00000070: 00 00 00 00 00 00 00 00  - 00 00 00 00 00 00 00 00   ................
00000080: 05 C2 8A 8D 98 C2 84 90  - 8A C2 72 64 C2 3E 26 20   ..........rd.>&
00000090: C2 1A 20 23 C2 1D 23 20  - C2 1D 23 1D C2 23 20 C2   .. #..# ..#..# .
000000A0: 1D 20 1D C2 1A 20 1D C2  - 20 23 1D C2 20 C2 1D C2   . ... .. #.. ...
000000B0: 23 1D C2 20 23 1D C3 20  - C3 23 1A 20 C2 29 1D 20   #.. #.. .#. .).
000000C0: C2 23 20 C2 26 C2 20 C2  - 26 20 1D C2 20 23 26 C2   .# .&. .& .. #&.
000000D0: 20 1D C3 23 C2 26 C2 1A  - 23 C2 26 1D 26 C2 23 20    ..#.&..#.&.&.#
000000E0: 26 C2 20 C2 26 C2 1D 29  - 23 C2 20 29 C2 23 C2 26   &. .&..)#. ).#.&
000000F0: C2 23 C2 26 C2 23 29 C3  - 26 29 23 C2 29 2C C2 23   .#.&.#).&)#.),.#
00000100: 29 26 C3 29 26 C2 23 26  - 29 C2 2C 29 20 C2 2F 26   )&.)&.#&).,) ./&
00000110: C2 29 2F 29 C2 26 C2 29  - C2 2C 32 3B C2 4C 5E 6C   .)/).&.).,2;.L'l
00000120: C2 87 84 95 C2 98 8D C2  - A1 C2 98 C2 AA 9B A1 C2   ................
00000130: AA 9B AD C2 A4 9E B3 C2  - A1 AA C2 B0 9B AA C2 A4   ................
00000140: 9E B6 C2 A4 A1 B3 C2 9B  - A1 A7 C2 98 AA C2 A4 9E   ................
00000150: B0 C2 A1 A7 AD C2 9B AD  - AA C2 9E B0 A1 C2 9E AD   ................
00000160: C2 93 C2 A1 C2 93 A1 95  - C2 93 A4 93 C2 95 9E 90   ................
00000170: C2 A1 98 90 C2 A4 90 C2  - 9B A7 95 C2 AA A7 9B C2   ................
00000180: AD 9E 9B C2 A7 93 9E C2  - A1 8D

00043190: 23 1D 1A C2 20 1A 20 C2  - 1D 1A C2 20 C2 1A C2 20   #... . .... ...
000431A0: 17 C3 1A 15 20 C2 1D 1A  - 20 C2 17 1A C2 20 1A 1D   .... ... .... ..
000431B0: C2 1A 1D 20 C2 1D 1A 00  - 00 00 00 00 00 00 00 00   ... ............
000431C0: 00 00 00 01 01 01 01 01  - 01 01 01 01 01 01 01 02   ................
000431D0: 02 02 02 02 02 02 02 02  - 02 02 02 03 03 03 03 03   ................
000431E0: 03 03 03 03 03 03 03 04  - 04 04 04 04 04 04 04 04   ................
000431F0: 04 04 04 05 05 05 05 05  - 05 05 05 05 05 05 05 06   ................
00043200: 06 06 06 06 06 06 06 06  - 06 06 06 07 07 07 07 07   ................
00043210: 07 07 07 07 07 07 07 08  - 08 08 08 08 08 08 08 08   ................
00043220: 08 08 08 09 09 09 09 09  - 09 09 09 09 09 09 09 0A   ................
00043230: 0A 0A 0A 0A 0A 0A 0A 0A  - 0A 0A 0A 0B 0B 0B 0B 0B   ................
00043240: 0B 0B 0B 0B 0B 0B 0B 0C  - 0C 0C 0C 0C 0C 0C 0C 0C   ................
00043250: 0C 0C 0C 0D 0D 0D 0D 0D  - 0D 0D 0D 0D 0D 0D 0D 0E   ................
00043260: 0E 0E 0E 0E 0E 0E 0E 0E  - 0E 0E 0E 0F 0F 0F 0F 0F   ................
00043270: 0F 0F 0F 0F 0F 0F 0F 10  - 10 10 10 10 10 10 10 10   ................
00043280: 10 10 10 11 11 11 11 11  - 11 11 11 11 11 11 11 12   ................
00043290: 12 12 12 12 12 12 12 12  - 12 12 12 13 13 13 13 13   ................
000432A0: 13 13 13 13 13 13 13 14  - 14 14 14 14 14 14 14 14   ................
000432B0: 14 14 14 15 15 15 15 15  - 15 15 15 15 15 15 15 16   ................
000432C0: 16 16 16 16 16 16 16 16  - 16 16 16 17 17 17 17 17   ................
000432D0: 17 17 17 17 17 17 17 18  - 18 18 18 18 18 18 18 18   ................
000432E0: 18 18 18 19 19 19 19 19  - 19 19 19 19 19 19 19 1A   ................
000432F0: 1A 1A 1A 1A 1A-1A 1A 1A  - 1A 1A 1A 1B 1B 1B 1B 1B   ................
00043300: 1B 1B 1B 1B 1B 1B 1B 1C  - 1C 1C 1C 1C 1C 1C 1C 1C   ................
00043310: 1C 1C 1C 1D 1D 1D 1D 1D  - 1D 1D 1D 1D 1D 1D
```

Figure 3.17 PCX File format

3.4.1 Definition of a PCX Header

The header of the PCX-file is used to describe the image stored. This section gives descriptive information of the video mode used, screen dimensions of the picture and palette colours used for EGA/CGA video modes. Table 3.1 shows the format of a PCX header for a 800x558 256 colour VGA picture.

Description	Byte Size	Data
Maker: PCX identification Bit	1 Byte	10h
Version: PCX identification Bit	1 Byte	5h
Code: PCX identification Bit	1 Byte	1h
BPP Number of Bits per Pixel	1 Byte	8h
X1:Left X coordinate for Picture	2 Bytes	0
Y1:Upper Y Coordinate for Picture	2 Bytes	0
X2:Right X Coordinate for Picture	2 Bytes	799
Y2:Bottom Y Coordinate for Picture	2 Bytes	557
HRES: Number of horizontal pixels	2 Bytes	800
VRES: Number of vertical pixels	2 Bytes	558
Triplet: EGA/CGA palette	3*16 Bytes	0
VMode: Current video mode	1 Byte	#0h VGA
NPlanes Number of byte planes used	1 Byte	#1h VGA
BPL Bytes per horizontal line used	2 Bytes	800
HeaderSpace 60 Bytes for expansion	60 Bytes	#0h

Table 3.1 PCX Header format

3.4.2 Data section

The data section consists of data for the reconstruction of a video screen. The data of a PCX file is crunched by a special mathematical algorithm to minimize disk storage space for picture information. Figure 3.18 shows the algorithm for crunching data to the PCX data format.

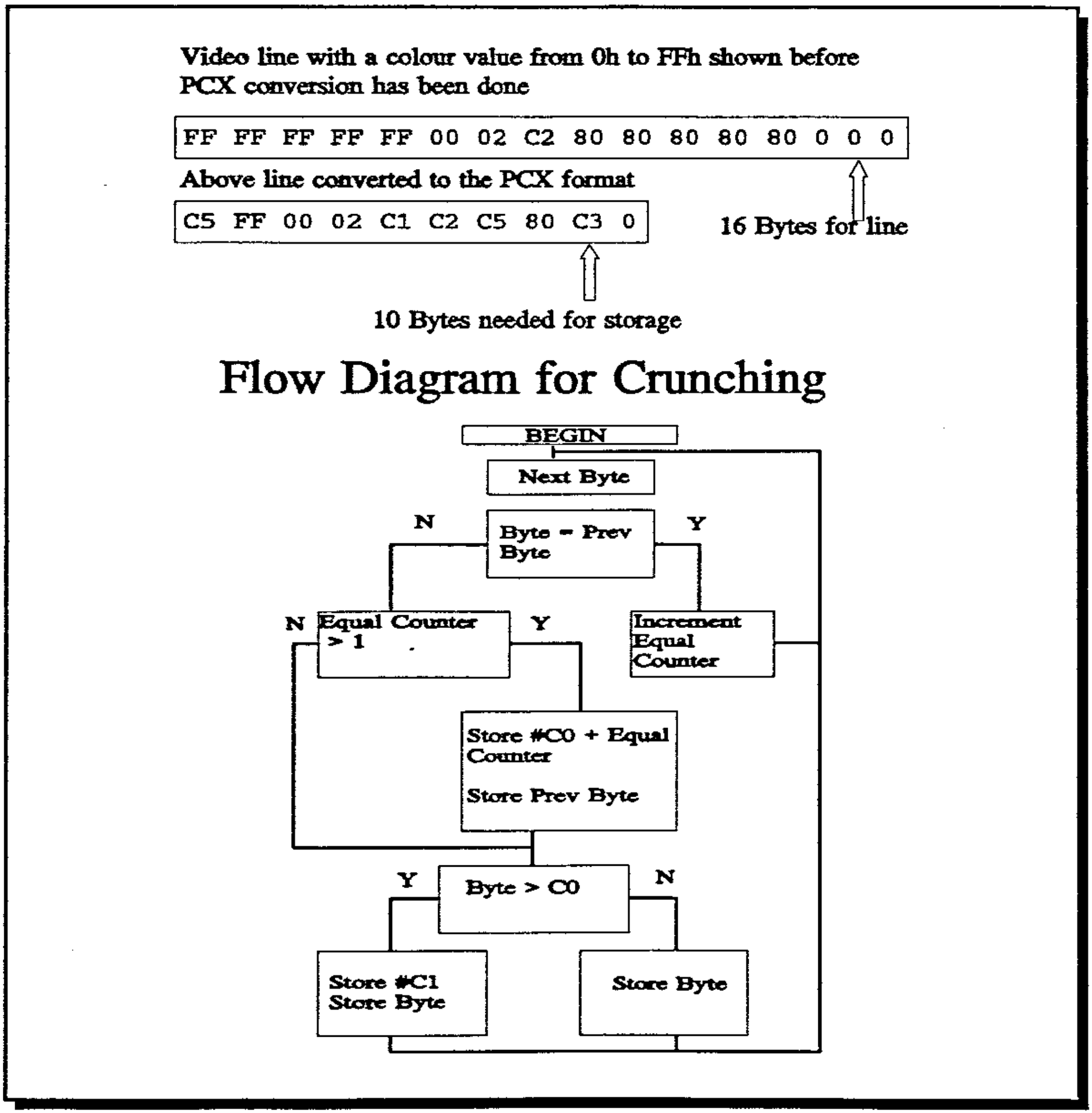

Video line with a colour value from 0h to FFh shown before PCX conversion has been done
FF FF FF FF FF 00 02 C2 80 80 80 80 80 0 0 0
Above line converted to the PCX format
C5 FF 00 02 C1 C2 C5 80 C3 0
16 Bytes for line
10 Bytes needed for storage
Flow Diagram for Crunching
BEGIN
Next Byte
N
Byte = Prev Byte
Y
N
Equal Counter > 1
Y
Increment Equal Counter
Store #C0 + Equal Counter
Store Prev Byte
Y
Byte > C0
N
Store #C1
Store Byte
Store Byte

Figure 3.18 PCX-Data crunching algorithm

The data crunching techniques detects a change in byte value. For a constant range of values eg 10 #FFh 's a counter value is stored together with the byte value. The starting value for the counter is set to #C0h. For a single byte bigger than #C0h a count value of #C1h together with the byte value is stored. For a single byte smaller than #C1h the byte value is stored.

3.4.3 VGA Palette

The VGA palette consists of 3*256 bytes of information for the selecting of a unique 256 colour palette. The VGA palette setting is the same as used in paragraph 3.2.4 for the setting of a grey scale VGA palette.

4. OPERATING INSTRUCTIONS (Software)

4.1 Minimum Requirements

- IBM Compatible PC or one of the following XT,286,386,486
- 640 Kb RAM
- 360k Floppy Drive
- Hard disk (Needs at least 3 Mbyte for correct operation)
- Any one of the following graphic adapters - Hercules
  - CGA,EGA,VGA

(A VGA graphics adapter is necessary to show the grey scales which are generated by the software. On all the other graphic adapters only a black and white image will be possible.)

4.2 Software Installation

The frame grabber comes with two 360k floppies which contains the Frame Grabber program as well as a single captured example. The disks are numbered as follow:

- 1 Install and program disk.
- 2 Picture example disk.

4.2.1 Instalment Procedure

The installation process is shown as an installation between Drive A, a 360k Floppy Drive to Drive C the hard disk.

- Boot from the hard disk and exit to the DOS prompt as shown eg. C:\
- Change to the floppy drive C:\A: [ENTER]
- Insert install program disk [1] into drive A:
- Type INSTALL C: to install program disk to drive C
- Insert example disk [2] into drive A:
- Type INSTALL C: to install example to drive A:
- This is the end of the install procedure.

The Program Disk and the example are installed onto the hard disk in the directory \FRAMEGRAB

- Change to the Directory C:\FRAMEGRAB
- Type FR30 to start the program.

4.3 Software Functions

4.3.1 F1 (Save File)

"Save File" stores a captured image to disk. Two different methods of storing the captured data may be used:

Normal Save saves the image as it is obtained from the frame grabber. The image is stored without any modifications to the data as captured.

PCX Save saves the image in a special PCX graphic format. The PCX format is a graphics format used by most of the popular desktop publishing software programs available on the market.

The image is stored as a 800 X 558 image with 256 grey scales. The PCX-Image may only be modified by special Desktop Publishing software.

4.3.2 F2 (Select File)

Retrieves any data file stored with the normal save command in function F1. Best option for editing of an existing picture

4.3.3 F3 & F4 (Start Address & Stop Address)

A captured image consists of 480000 bytes of information. Only a portion of a whole video frame may be shown as a Graph on a XY axis type display by changing the start and stop addresses.

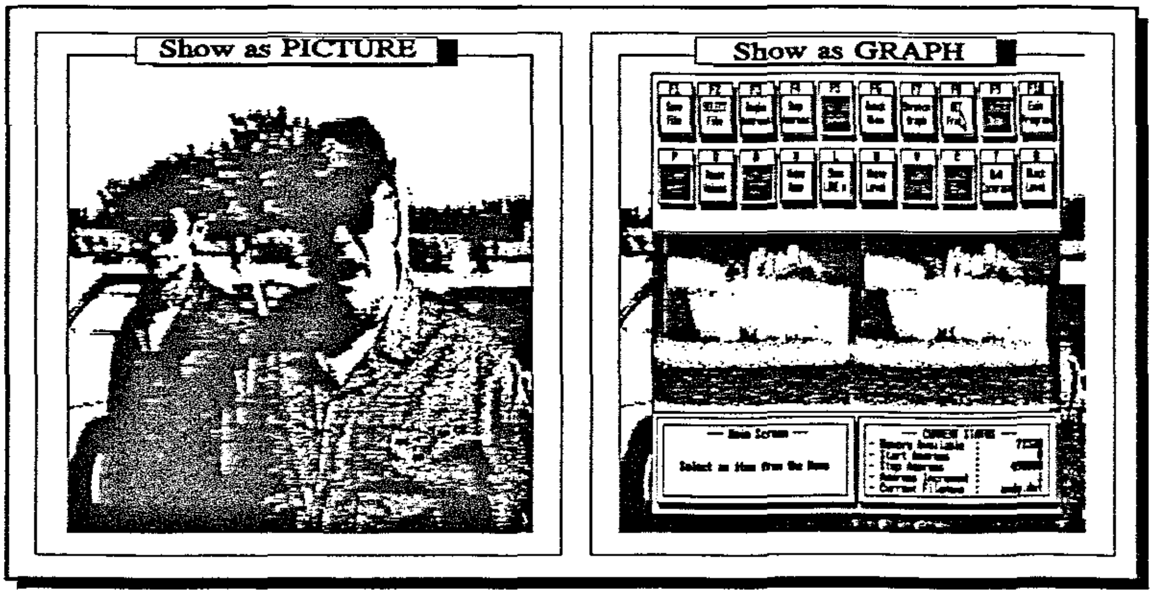


Figure 4.1 Show as PICTURE / Graph

4.3.4 F5 (Show as PICTURE / Show as Graph)

This function key act as a toggle between the graph display and the picture display modes available in this program. Show as picture will convert the captured data into a picture which will be displayed on the screen. Show as graph gives an oscilloscope type of display of the Data captured. Press F8 for a display.

4.3.5 F6 (Quick View)

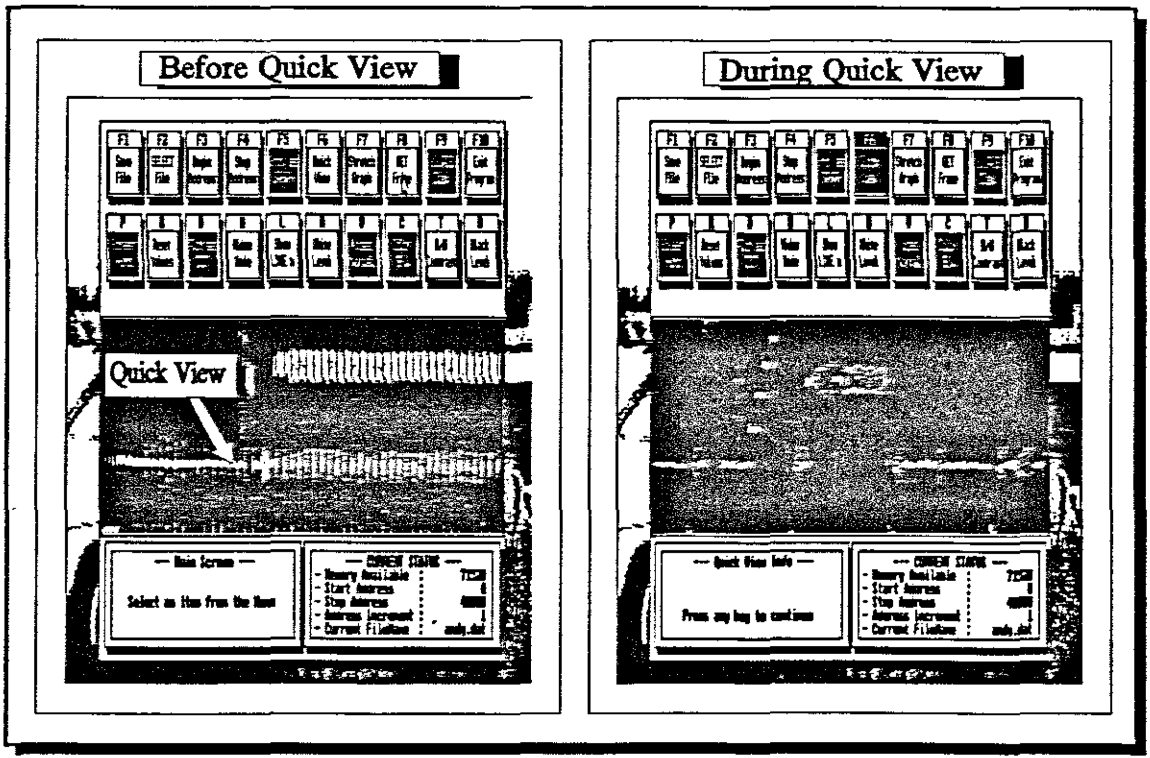


Figure 4.2 Quick expansion of Graph View

The quick view function is dependable on the show mode selected by F5. For the picture mode a quick view is shown of the last captured Picture. In the graph mode a fixed portion of the graph display is enlarged for easy location of special patterns. Figure 4.2 shows an example of the quick view operation in the graph mode. Move the curser to a point on the graph for quick expansion. When the location is reached press F6 for a quick expansion from that point on. After the expansion the screen will return to it original state before quick view were activated.

4.3.6 F7 (Stretch Graph)

The stretch function is only available in the graph display mode. Move the cursor to the first point in the graph to be stretched press ENTER to enter the first coordinate. Move the cursor to the next point to be stretch The graph will be stretched around these two points.

4.3.7 F8 (GET Frame)

F8 is the key to get new information from the frame grabber or disk. F8 will redraw the screen to the parameters set by the different functions.

4.3.8 F9 (Source Disk / Grabber)

F9 acts as a toggle between disk operation or direct mode where new data is retrieved directly from the frame grabber unit.

4.3.9 F10 (Exit Program)

Exit out of the FR30 program and return back to DOS

4.3.10 P (Screen Auto / 1:1)

P acts as a toggle between a scaled full screen display of the picture information or a 1:1 direct display. With the Scale setting to auto the picture is automatically scaled to fill the full screen.

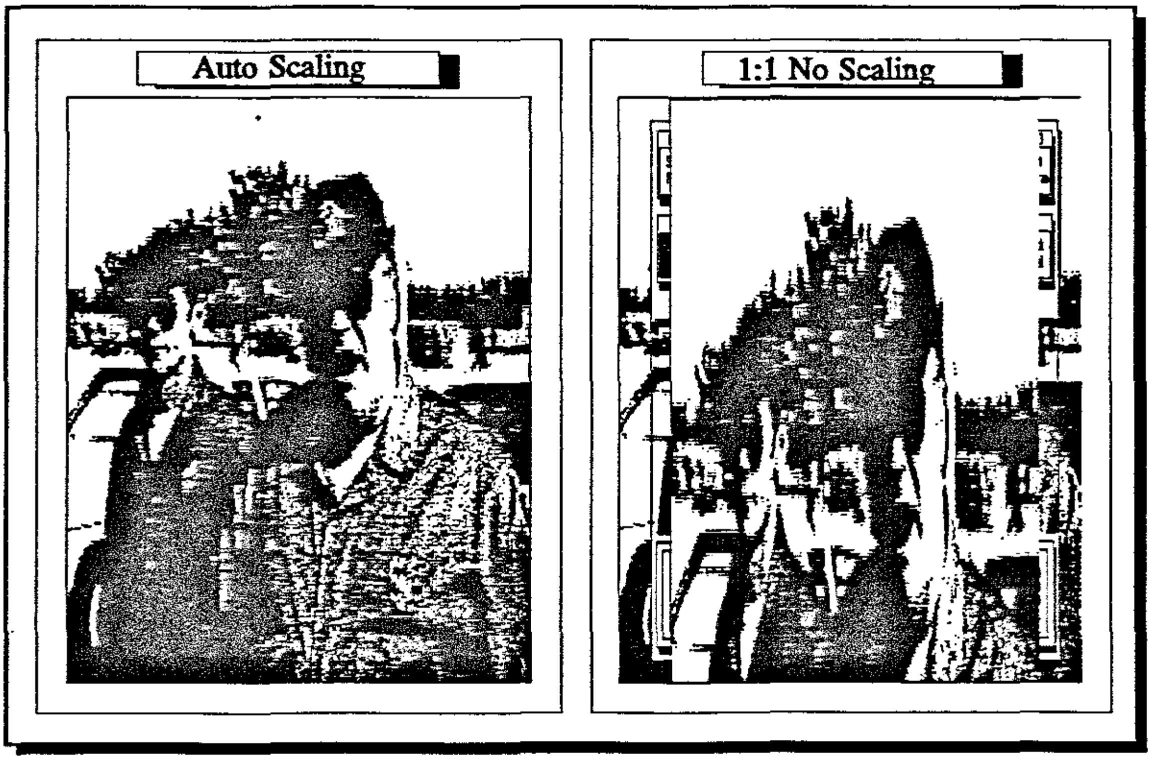


Figure 4.3 Scale set to AUTO and to 1:1

The scale setting 1:1 shows the picture without any scaling. This mode works the best on VGA graphics adapters capable of showing 800x600 resolution with a 256 colour palette.

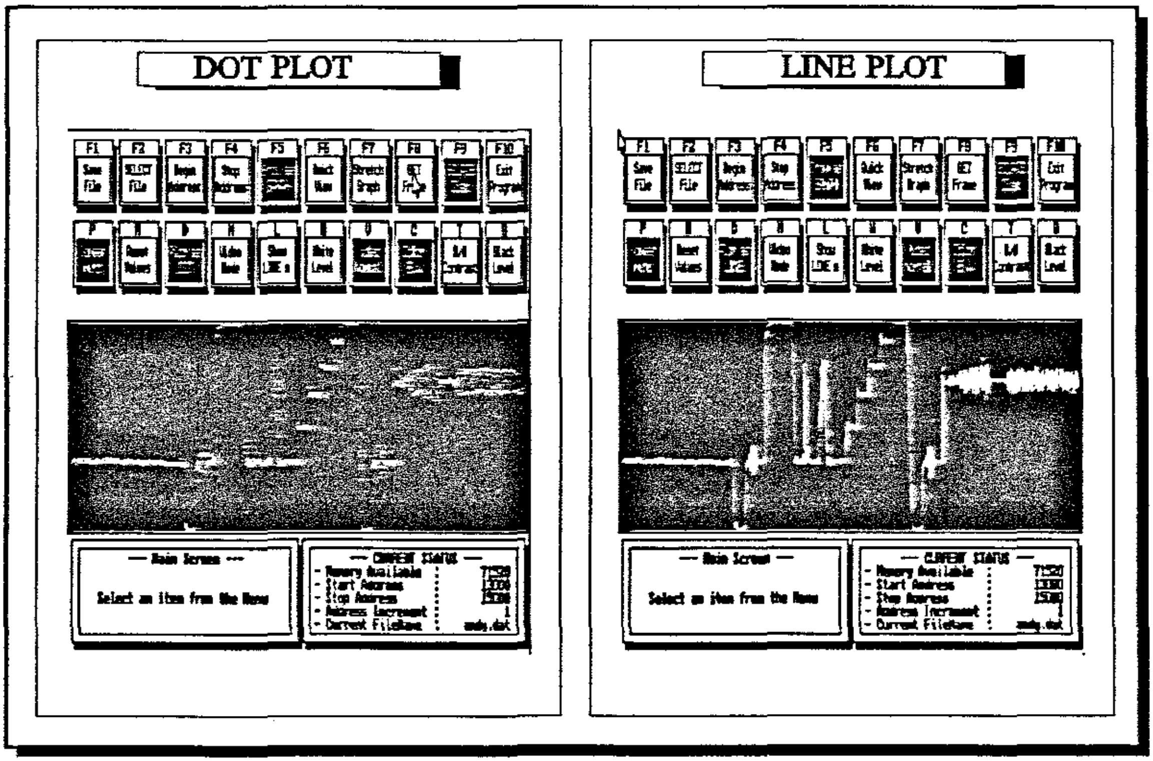
With auto scaling interference patterns may occur but with the 1:1 scale mode no interference patterns are visible.

4.3.11 R (Reset values)

Reset all the variables used to their original values at start up. This function is very useful when a new image is imported for a new set up of the screen and draw variables.

4.3.12 D (Plot as Line/Dot)

In the graph mode the graph may be plotted as single dots or the dot may be connected to get a better indication of how the waveform looks.



— Main Screen —

Select an item from the Menu

— CURRENT STATUS —
- Memory Available : [illegible]
- Start Address : [illegible]
- Stop Address : [illegible]
- Address Increment : 1
- Current FileName : [illegible].dat

Figure 4.4 Dot and Line Plots

4.3.13 M (Video Mode)

The frame grabber software supports the following graphic modes and adapters. - 1 Hercules 720x348

- 2 CGA 2 Colours 640x200
- 3 CGA 4 Colours 320x200
- 4 EGA/VGA Adapter 640x350
- 5 Extended VGA

SVGA256 256 colour	SVGA16 16 colour	SVGA32K 32768 colours	TWEAK256 256 colour	TWEAK16 16 colour
320x200	320x200	320x200	320x400	704x528
640x400	640x400	640x400	320x480	720x540
800x600	800x600	800x600	360x480	736x552
1024x768	1024x768	1024x768	400x564	768x576
640x350	640x350	640x350	400x600	784x588

4.3.14 L (Show LINE N)

This option selects a single video line to be displayed as a graph. Press L 33 to show line 33.

4.3.15 W (White Level)

Selects a new value for the current setting of the white level. The standard value for this setting is 127. Dark pictures may be lightened by lowering the value of the white level setting.

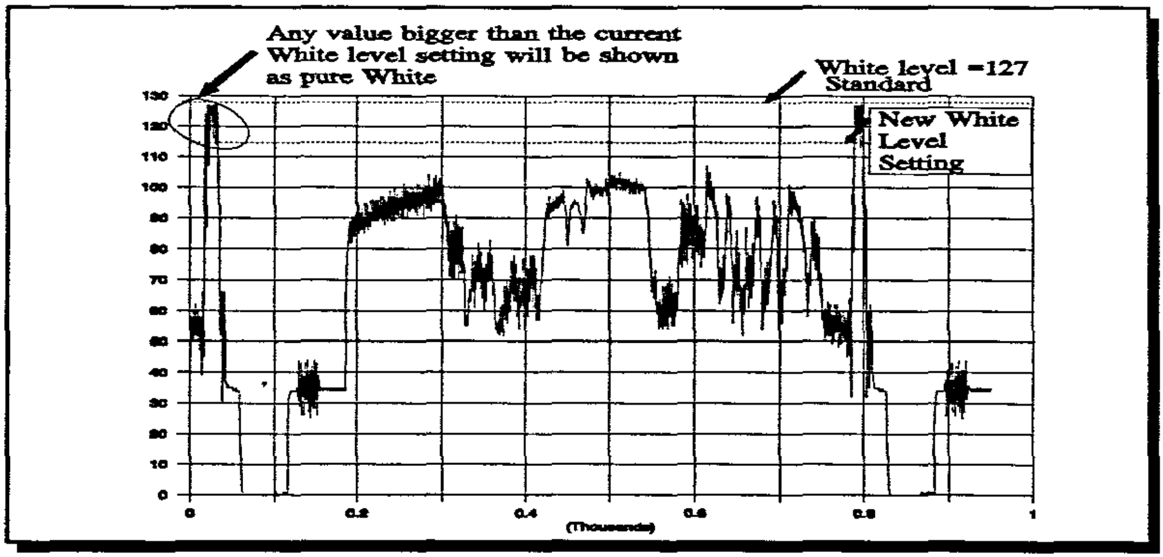


Figure 4.5 White level selection

4.3.16 <u>V (Video Normal/Invert)</u>

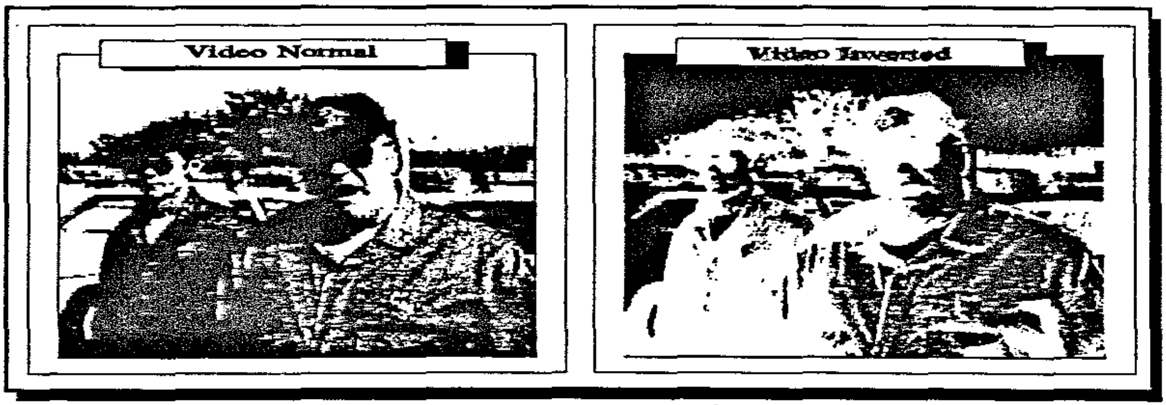


Figure 4.6 Normal and Inverted display

V serves as an toggle switch between normal screen display any inverted screen display. Some of the current screen dump software programs requires the screen to be inverted to give a good output when a screen is captured .

4.3.17 C (Color B/W / GrayScale)

C serves as a toggle switch between a black and white display of the picture or a 64 grey scaled picture.

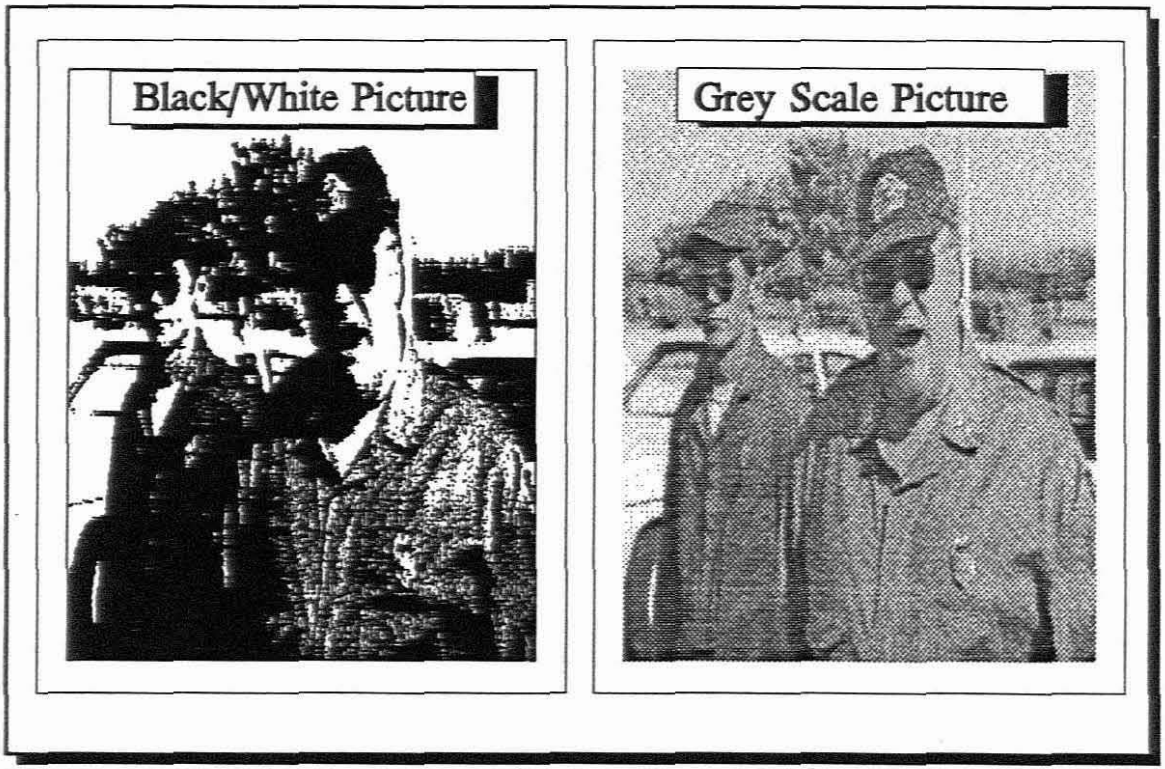


Figure 4.7 Picture shown in Black and White & Grey Scale

4.3.18 T (B/W Contrast)

The B/W contrast is the toggle point setting between black & white while the program is in the B/W mode for picture conversion. The preset value for the B/W contrast setting is 75. If this value is made higher more black will appear in the converted picture. Figure 4.8 shows the toggle point between black and white operation.

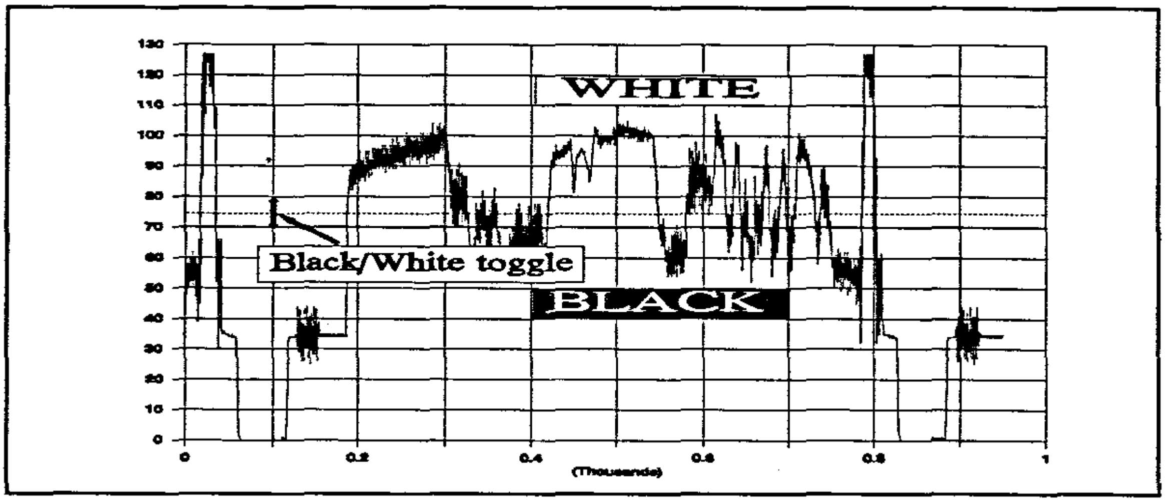


Figure 4.8 Black/White toggle point

4.3.19 <u>B (Black Level)</u>

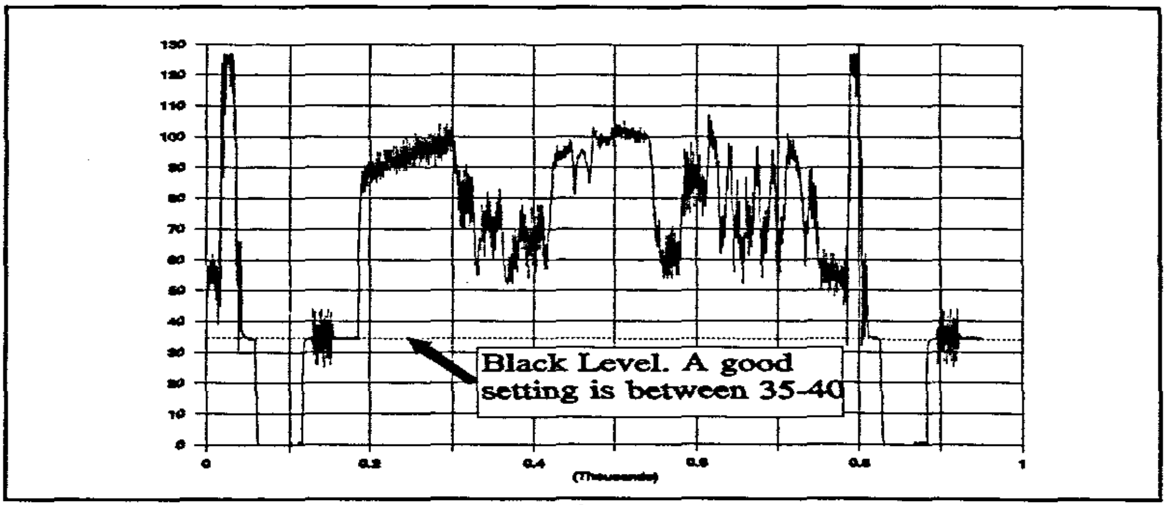


Figure 4.9 Black level adjustment

Figure 4.9 shows the black level for a captured video signal. The best way to adjust the black level is to plot the picture firstly as a graph. With this option it is possible to measure a new value for the current black level.

5 HARDWARE INSTRUCTIONS

5.1 Hardware Installation

The frame grabber unit consists of three sections:

- Address Decoder Card
- Digitizer unit
- Frame Grabber Software

5.1.1 Address Decoder Card

The address decoder card acts as an interface between the digitizer unit and the PC. The following installation procedure must be followed.

- Turn off the Computer
- Open the computer and plug the address decoder card into an empty expansion slot.
- Connect the data connection cable to the external DB 25 connector.

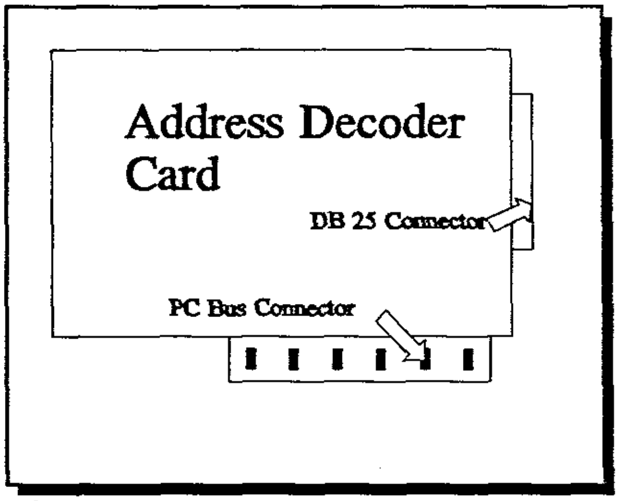


Figure 5.1 Address Decoder Card

5.1.2 Digitizer Unit

This unit is a separate unit for the digitizing of an analog video signal. The following installation procedure must be followed.

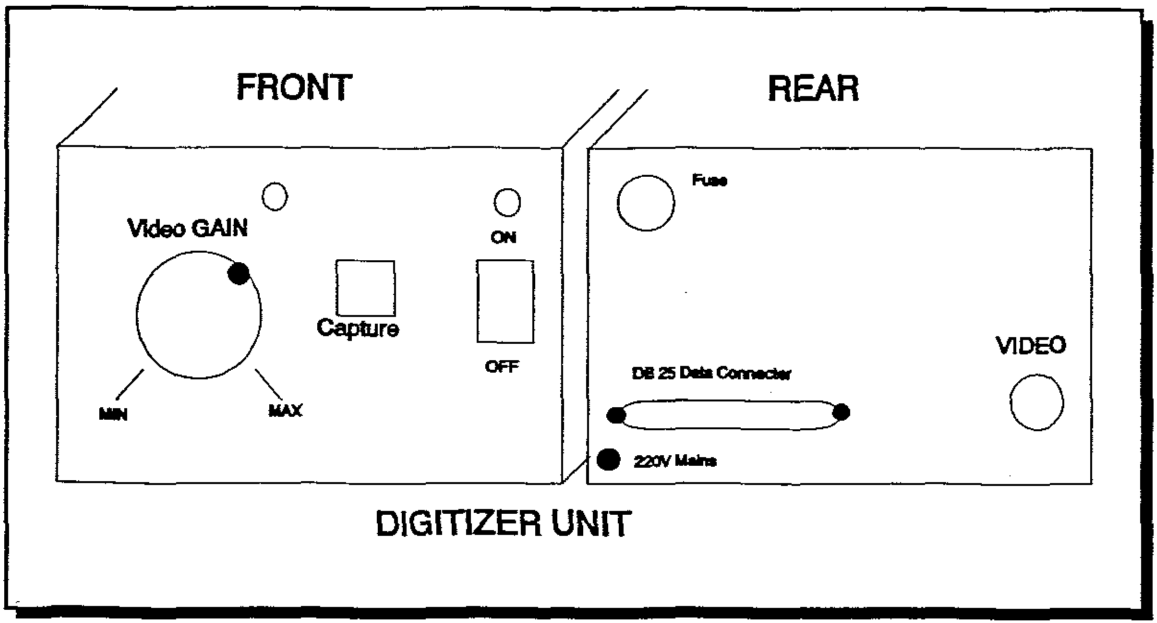


Figure 5.2 Digitizer unit

- Turn power switch to OFF
- Turn Video gain to MIN
- Connect data connection cable to DB 25 connection of this unit
- Connect external video source to VIDEO input
- Turn the power switch to ON
- Adjust the video gain until video gain indicator in just visible

5.1.3 Frame Grabber Software

The installation and operation of this software is discussed in chapter 4.

5.2 Operation of the Frame Grabber unit

Apply a video signal from a standard video source eg. (Video camera) to the video input of the digitizing unit. Adjust the video gain control until the video gain indicator is just visible. This is the fixed setting for video gain for the current video source in used. Reset this value for each new video source in use.

PRESS the CAPTURE BUTTON for capturing a desired Video Frame

This captured signal can be downloaded to the PC under control of the frame grabber software FR30.EXE for modification and storage to disk.

6. XILINX INFORMATION

6.1 Introduction

Xilinx is the tradename for a newly developed Field Programmable Gate Array. Field Programmable Gate Arrays (FPGA's) are high density (Application Specific Integrated Circuits) ASIC, that can be configured by the user. They combine the logic integration benefits of custom VLSI with the design, production and time-to-market advantages of standard products. Designers define the logic functions of the circuit and revise these functions as necessary. Thus FGPA's can be designed and verified in a few days, as opposite to several weeks for a custom array. This results in significant cost savings by reducing the risk of design changes, rescheduling and eliminating non-recurring engineering costs. For further information of the internal structure of Xilinx FPGA's see the Xilinx Programmable gate Array Data Book rev 1992 or later.

6.2 Programming steps required for programming Xilinx FGPA's

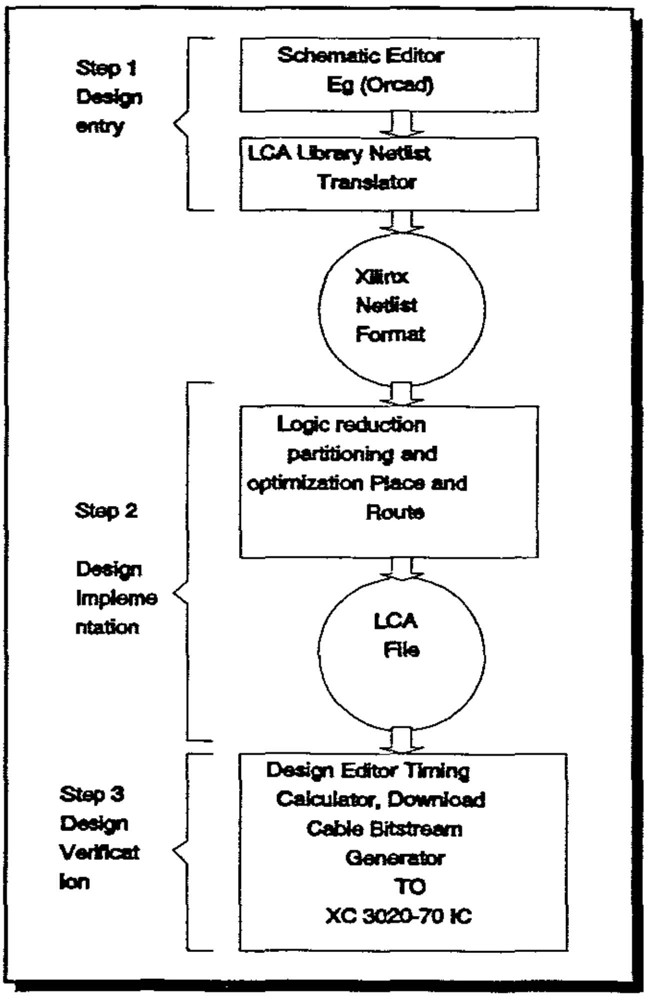
Designers can use familiar CAE tools for design entry and simulation. The Xilinx development system includes a standard netlist format, the Xilinx netlist file XNF, that provides a bridge between schematic editors or simulators and the XACT software for design implementation and real time verification. The Xilinx software requires a 386 micro computer with at least

8 Mbyte of RAM for operation.

6.2.1 **Design entry Software**

consists of libraries and netlist interfaces for standard CAE software such as FutureNet, Schemia ORCAD, ViewLogic. Programmable gate array libraries permits design entry with standard TTL functions with Boolean equations.

For this thesis the XILINX TTL schematic library interface was used for the interfacing of Xilinx to the ORCAD schematic editor.



6.2.2 **Design Implementation Software**

From the design entry point a netlist is generated which is encoded by the APR (Automatic place and route program) to re-compile the internal structure of the Xilinx FGPA.

This auto-routed internal structure of the Xilinx may then be edited by a design editor (EditLCA) for the optimization of the circuit design.

The edited LCA will then be re-compiled to a digital bitstream to be downloaded into the Xilinx IC by a specially adapted downloading cable. The bitstream may be burned into a ROM for permanent storage for the final design, after all alterations to the design are completed. The ROM bitstream may be configured to be loaded into the Xilinx FPGA at power on.

7 HARDWARE LAYOUT

This section gives an overall out lay of the hardware section of the Frame Grabber system as used.

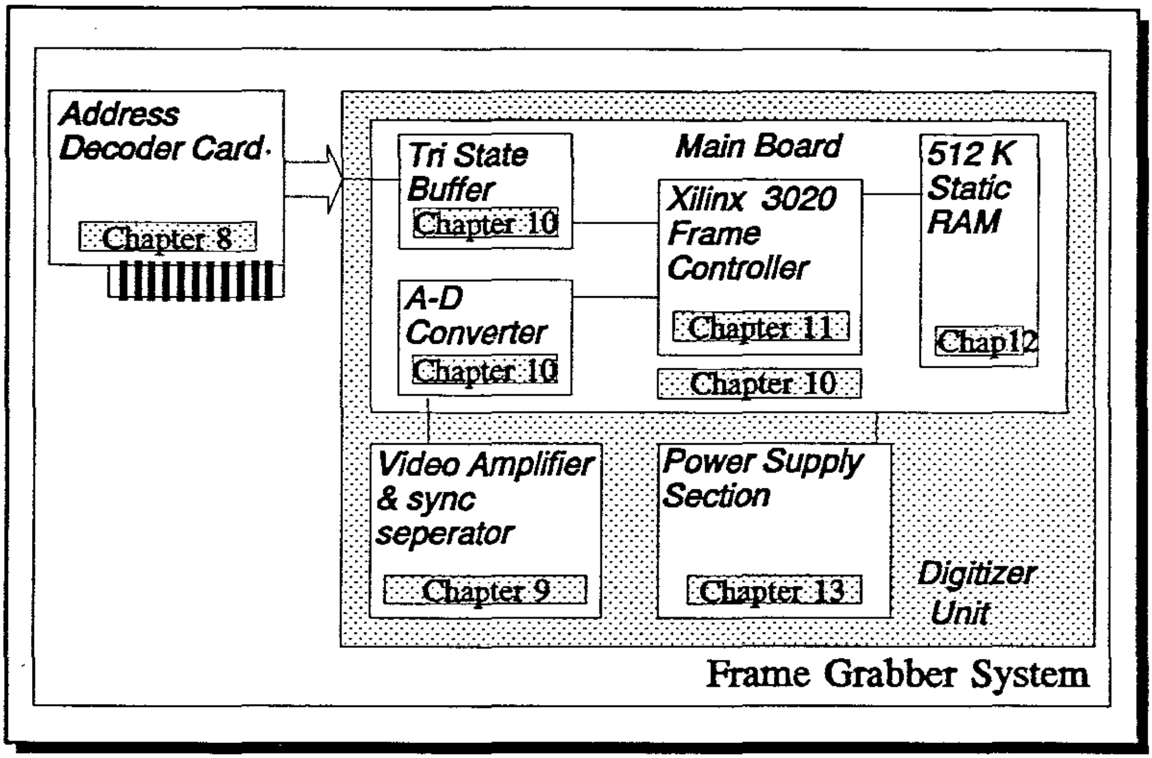


Figure 7.1 Overall hardware layout of the Frame Grabber

The hardware section of the frame grabber consists of a variety of different sections as shown in Figure 7.1. Each of this sections are described in the chapters as shown in Figure 7.1. The digitizer unit is a stand alone unit for video digitizing and the address decoder card is a plug in card designed as a interface between the digitizer unit and the PC.

8. ADDRESS DECODER

8.1 Introduction

The address decoder acts as an interface between the frame grabber unit and the personal computer for all data communication between these devices. A detailed drawing of the schematic diagram is shown in Appendix A, Drawing No 1.

8.2 Design features

The address decoder was designed by using the following main objectives.

- Fully compatible with the standard PC-Bus.
- To access 512K of external memory in the frame grabber unit.
- To Read 8-bit parallel data from the frame grabber unit.
- To Write 8-Bit parallel data to any of the 512K of memory on board the frame grabber unit for easier faultfinding and RAM error checking.
- For automatic selection between reading and writing to and from the frame grabber unit.

8.3 Hardware design

The address decoder is a plug in card designed to fit into the bus of a standard IBM Compatible PC. The address decoding is based around an 8-bit comparator and a 4 to 16 line decoder. The 4 to 16 line decoder is used to dedicate certain functions to specific I/O addresses of the PC.

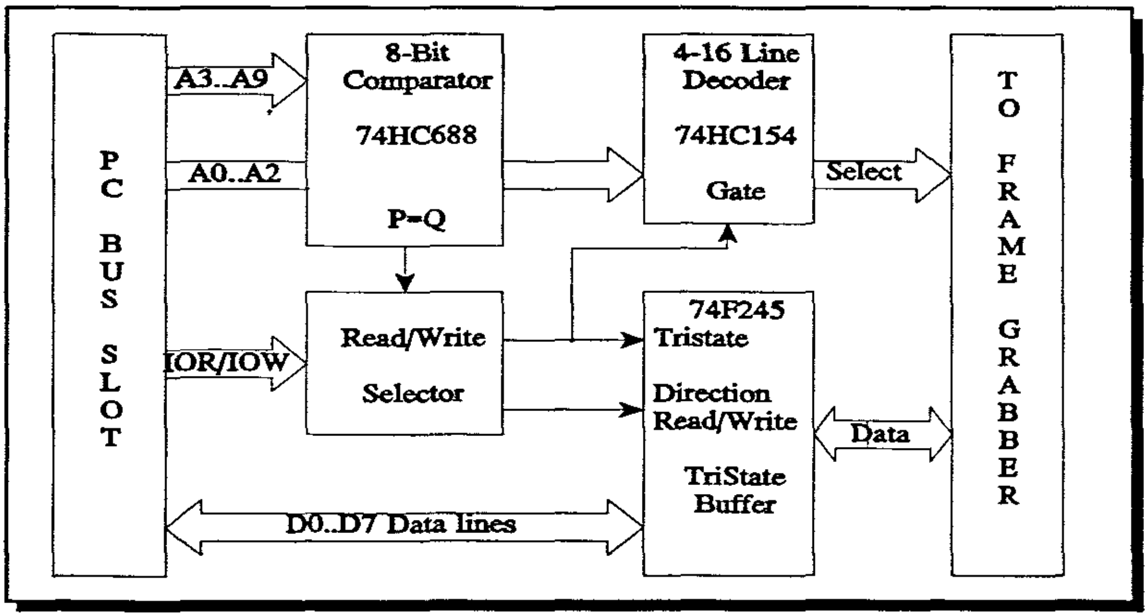


Figure 8.1 Block Diagram of the Address Decoder

The above block diagram shows a layout of the address decoder.

8.4 Theory of operation

8.4.1 Comparator Stage

A high speed CMOS 8-Bit Comparator U1 is used as an 8-bit address comparator. The addresses of the PC-Bus are monitored until a PC I/O address in the range #300h to #307 is obtained.

The AEN line is monitored for the decoding of a valid address on the PC-Bus. The address lines on the PC-Bus is monitored until the conditions as shown below are met.

AEN	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	1	0	0	0	0	0	X	X	X

Address lines A0..A2 are not compared to obtain a valid address range from 300h to 307h. P=Q will go low whenever a valid address in this range is found.

8.4.2. Read/Write Selector Stage

The read/write selector is used for the automatic selection of a read or a write to and from the frame grabber. The read write selector shuts down all control lines and data flow to and from the frame grabber if no valid address is found.

The read write selector is formed around U4 and U5. These two IC's control the direction and tristate controls of the 74HC245 tristate input/output buffer U3. The gate of the 4 to 16 line decoder (U2) is switched by this section to stop all selection controls if no valid address is found.

The IOR,IOW and the P=Q output of the 8-bit comparator is used to detect any read or write operation to the frame grabber.

Table 8.1 shows a detailed functional description of all the input and outputs of this section.

IOR	IOW	P=Q	Buffer Enable	Direction an description of operation.
0	0	0	YES = 0	Both IOR and IOW may not be low
0	0	1	NO = 1	No operation DIR=1
0	1	0	YES = 0	Read from Frame Grabber DIR=0
0	1	1	NO = 1	No operation DIR=1
1	0	0	YES = 0	Write to Frame Grabber DIR=1
1	0	1	NO = 1	No operation DIR=1
1	1	0	YES = 0	No IOR or IOW from PC DIR=1
1	1	1	NO = 1	No valid address found DIR=1

Table 8.1 Read/Write Selector input and output

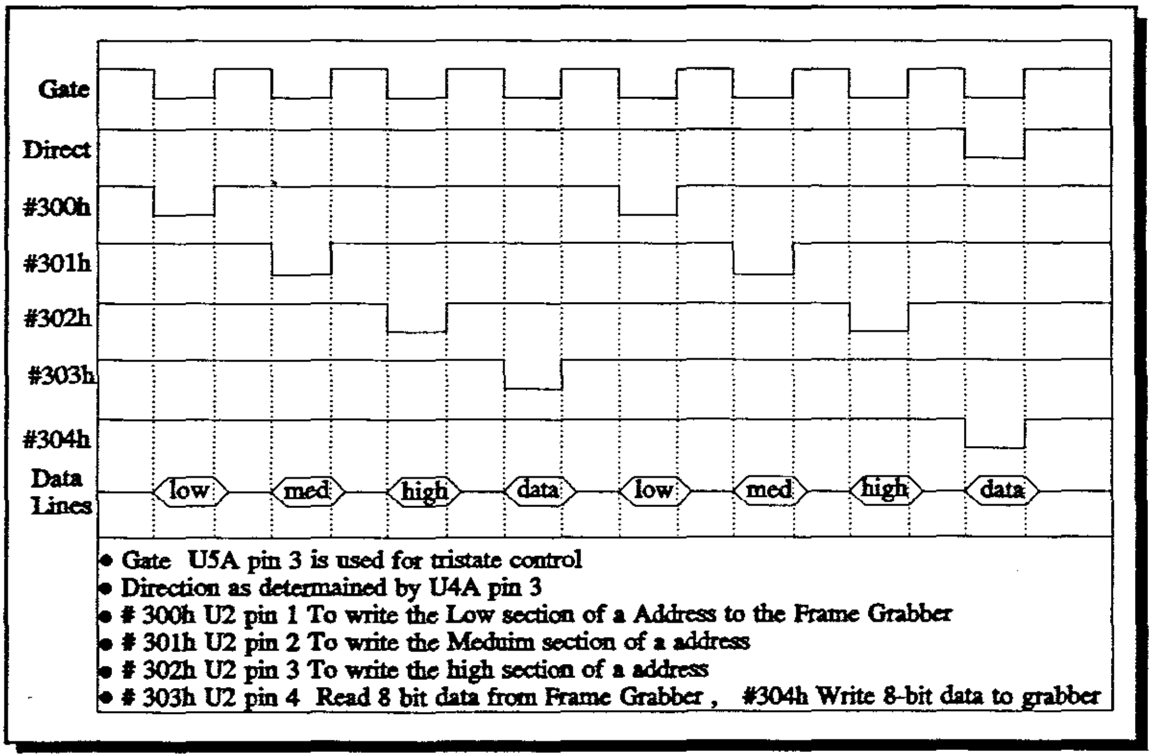
8.4.3 4-16 Line Decoder

The PC I/O addresses 300h to 304h are decoded by a 4 to 16 line decoder to form the control circuitry of the frame grabber unit. The first five outputs of the 4-16 line decoder corresponds to the addresses 300h to 304h. One of the five outputs of the decoder will be low for the duration of a valid address on the PC-Bus. The functional desciption of the first five control lines are as follow:

- 300h Supply low part of a 18 Bit address
- 301h Supply middle part of the address
- 302h Supply high part of the address
- 303h Read 8-bit data from the frame grabber

- 304h Write 8-bit data to the frame grabber.

Figure 8.2 shows a timing diagram of the programming sequence of the Frame Grabber unit.



- Gate U5A pin 3 is used for tristate control
- Direction as determained by U4A pin 3
- # 300h U2 pin 1 To write the Low section of a Address to the Frame Grabber
- # 301h U2 pin 2 To write the Meduim section of a address
- # 302h U2 pin 3 To write the high section of a address
- # 303h U2 pin 4 Read 8 bit data from Frame Grabber , #304h Write 8-bit data to grabber

Figure 8.2 Timing waveforms to control Frame Grabber

8.4.4 Tristate Buffer

The data lines of the PC are switched through a tristate buffer for protection of the PC-Bus and for better transmission over the data lines between the Frame Grabber and the PC. The timing for the switching of direction and tristate control is found in Figure 8.2.

9 VIDEO AMPLIFIER SECTION

9.1 Introduction

The video amplifier section serves as an interface between the video scanning device (normally a video camera) and the analog to digital converter. A full schematic diagram of the video amplifier and sync separator sections is shown in Appendix A drawing No 2.

9.2 Design considerations

The following main objectives were used for the designing of this section.

- Fully compatible with the standard PAL analog video output of a video machine or video camera.
- Must have a variable gain for the matching to non standard video inputs.
- The automatic clamping of the video signal to a level above 0 volt.
- A very high bandwidth amplifier for good picture resolution.
- A sync separator for detecting ODD and EVEN sync pulses for controlling the sampling period for the analog to digital converter in the Frame Grabber unit.

9.3 Hardware design

The video amplifier section is based on three very high speed operational amplifiers (3 * LM6361) with a gain bandwidth product of 50 MHz. A single operational amplifier is used to clamp the amplified video signal to a level above 0 Volt.

The separation of timing information for horizontal, vertical and odd/even field information is extracted by a single sync separator IC (LM1881N).

See Figure 9.1 for a complete block diagram of the video amplifier.

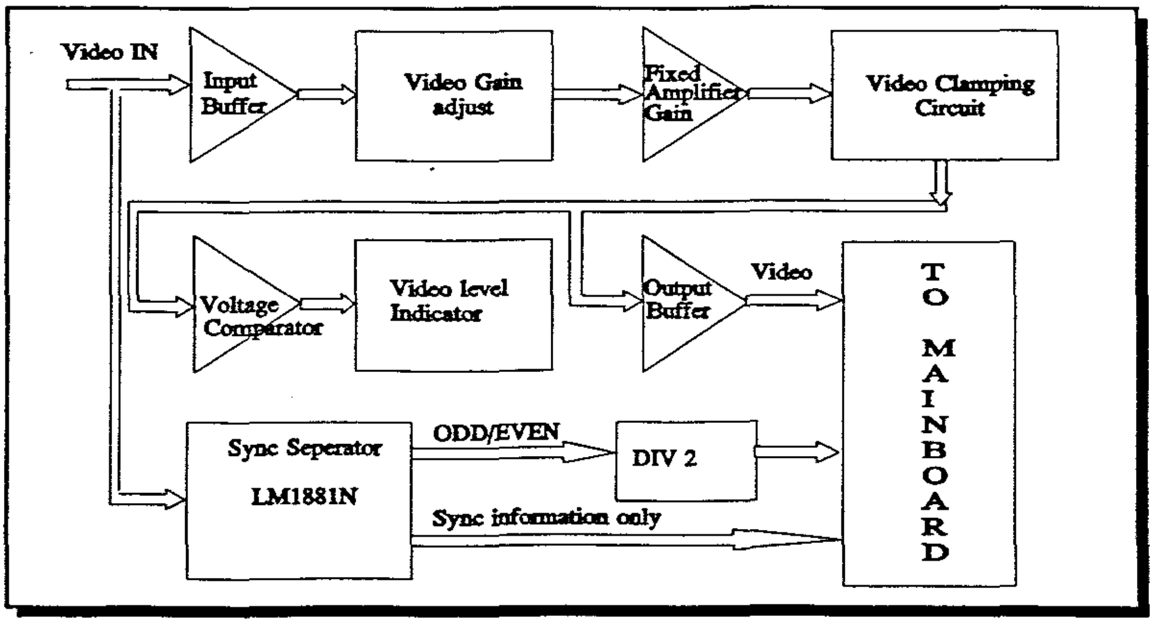


Figure 9.1 Block diagram of the video amplifier section

9.4 Theory of operation

9.4.1 Input buffer

UBUF1 serves as a non-inverting amplifier with unity gain to drive the input of the rest of the video amplifier and it's used for correct video input impedance matching. R5 is used for impedance matching between the external video source and the input of the sync separator and the input buffer UBUF1. A buffer with a very high gain bandwidth product is needed to minimise any losses in picture quality.

9.4.2 Video gain Adjust and the Fixed gain amplifier

UAMP1 serves as a non-inverting hi-speed video amplifier with a fixed gain of ±5,5. The gain may be calculated as follow:

$$\text{Gain AV} = \frac{R1 + R2}{R2} = \frac{1500 + 330}{330}$$

$$= 5{,}5454545$$

The non-inverting input of the operational amplifier is connected to a variable resistor (GAINPOT). This variable resistor is used to adjust the input video level from a minimum of 0V to the input level obtained from the input source. The video gain control is used for adjusting the video level to ±0,8 V (peak to peak). This 0,8V level will be amplified by the fixed gain amplifier UAMP1 to give a 5V (peak to peak) video level. For a better understanding of this amplifier section refer to figure 9.2 drawings No 1-3.

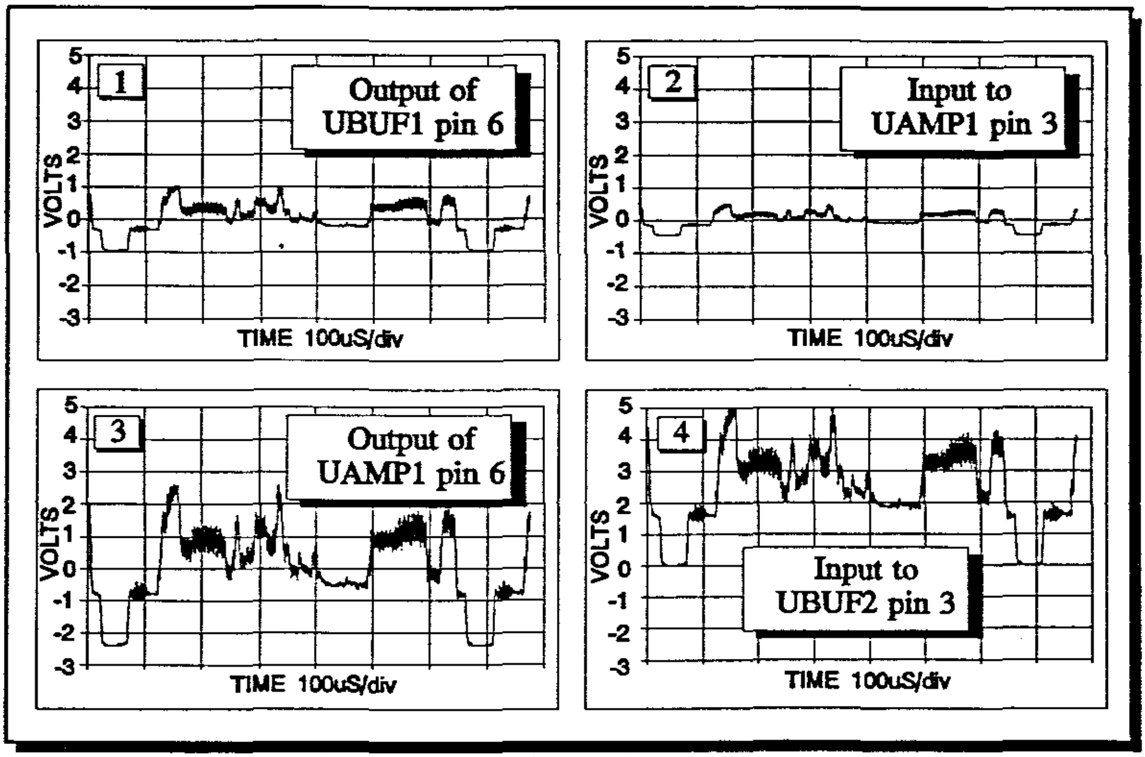


Figure 9.2 Video amplifier signals

9.4.3 Video Clamping

UCLAMP1 is used for clamping the video signal to a level above 0V. The output of UCLAMP1 is fed back to the input by diode D1. This forces the input to be at a level higher than 0V. The output of this section may be seen in Figure 9.2 drawing No 4.

9.4.4 Video level indicator and Voltage Comparator

UCOMP is used as a voltage comparator to monitor the peak video level after amplification. The comparator monitors the video level for peak level clipping. A LED is turned on as soon as the peak video level reaches a level of more than 5V. The brightness of the LED gives a good indication of the degree of peak video level clipping.

For optimal use the video gain control knob must be turn down until the LED is just not glowing.

9.4.5 Output Buffer

A high speed, high gain bandwidth product operational amplifier (UBUF2) is used as a non-inverting unity gain buffer to drive the input of the analog to digital converter on the main board. The analog to digital converter requires a good driving buffer for optimal performance.

The peak output voltage of the amplifier board is limited to 5,1V for the protection of the CMOS analog to digital converter. A 5.1V zener diode ZD1 and resistor R4 is used to clip the peak output signal to a limit of 5.1V. The peak analog input voltage to the A-D converter may not exceed the reference voltage used by more than 0,2V.

9.4.6 Sync Separator

The LM1881N is designed to strip the synchronisation signals from a composite video source that are in, or similar to, the NTSC and PAL formats. Input signals with positive polarity video (increasing signal voltage signifies increase scene brightness) from 0,5 V(p-p) to 2V (p-p) can be accommodated. The only external components required are the input coupling capacitor and a single resistor that sets internal current levels allowing the LM1881 to be adjusted for source signals with line frequencies differing from 15,625 kHz. Four major signals are available from the IC.

- Composite sync including both horizontal and vertical scan timing information.
- A vertical sync pulse.
- A burst gate and back porch clamp pulse.
- An ODD/EVEN output.

For the use of this project the composite sync and ODD/EVEN timing information was used.

The ODD/EVEN information is used to get the starting position of a video frame. The ODD/EVEN pulse is divided by 2 (UDIV2A) to give a pulse for allowing the main board to capture a full video frame consisting of both fields.

10 MAIN BOARD DESCRIPTION

10.1 Introduction

The main board of the video frame grabber contains the digitizing and storing sections of the frame grabber system. A full schematic diagram of the main board is shown in Appendix A drawing No 3 .

10.2 Theory of operation

Figure 10.1 shows an overall Block Diagram of the Main board as used. The different sections and interconnections is clearly shown in simplified form.

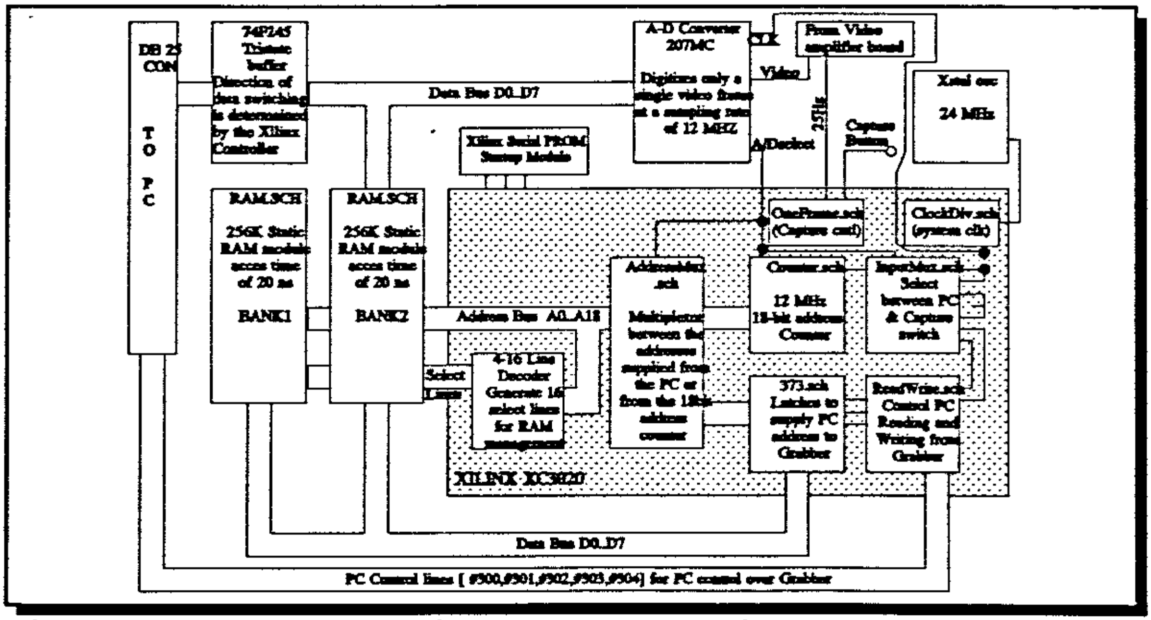

DB 25 CON
T O P C
74F245 Tristate buffer Direction of data switching is determined by the Xilinx Controller
Data Bus D0..D7
Xilinx Serial PROM Startup Module
A-D Converter 207MC
Digitizes only a single video frame at a sampling rate of 12 MHZ
CLK
Video
From Video amplifier board
25Hz
A/Dselect
Capture Button
Xtal osc
24 MHz
RAM.SCH
256K Static RAM module acces time of 20 ns
BANK1
RAM.SCH
256K Static RAM module acces time of 20 ns
BANK2
Address Bus A0..A18
Select Lines
4-16 Line Decoder Generate 16 select lines for RAM management
AddressMux.sch
Multiplexer between the addresses supplied from the PC or from the 18bit address counter
OneFrame.sch (Capture cntl)
ClockDiv.sch (system clk)
Counter.sch
12 MHz 18-bit address Counter
InputMux.sch Select between PC & Capture switch
373.sch Latches to supply PC address to Grabber
ReadWrite.sch Control PC Reading and Writing from Grabber
XILINX XC3020
Data Bus D0..D7
PC Control lines [#300,#301,#302,#303,#304] for PC control over Grabber

Figure 10.1 Block diagram of the Main board

10.2.1 Analog to Digital Conversion

The ADC-207 is the industry's first 7-bit flash converter using a high speed 1.2 micron CMOS process. This process offers some very distinctive advantages over the other process, making the ADC-207 a very unique device. The smaller geometric of the process achieves high-speed, better linearity and better temperature performance. Since the ADC-207 is a CMOS device it also has very low power consumption (250 mW)

The device draws power from a single +5V supply, and is conservatively rated for 20 MHz operation.

The ADC-207 has 128 comparators which are auto-balanced on every conversion so as to cancel out any offsets due to temperature and/or dynamic effects. The resistor ladder has a midpoint tap for use with an external voltage source to improve integral linearity beyond 7 bits. The ADC-207 also provides the user with 3 state output for easy interfacing with other components or data busses.

The reference voltage of this ADC-207 converter should be hold to 0.1% accuracy or better for optimum accuracy. This reference voltage is kept stable by a LM7805 (UREF1) in the design shown on the main board.

Figure 10.2 gives a full timing diagram of the Sampling sequence used for capturing a video frame.

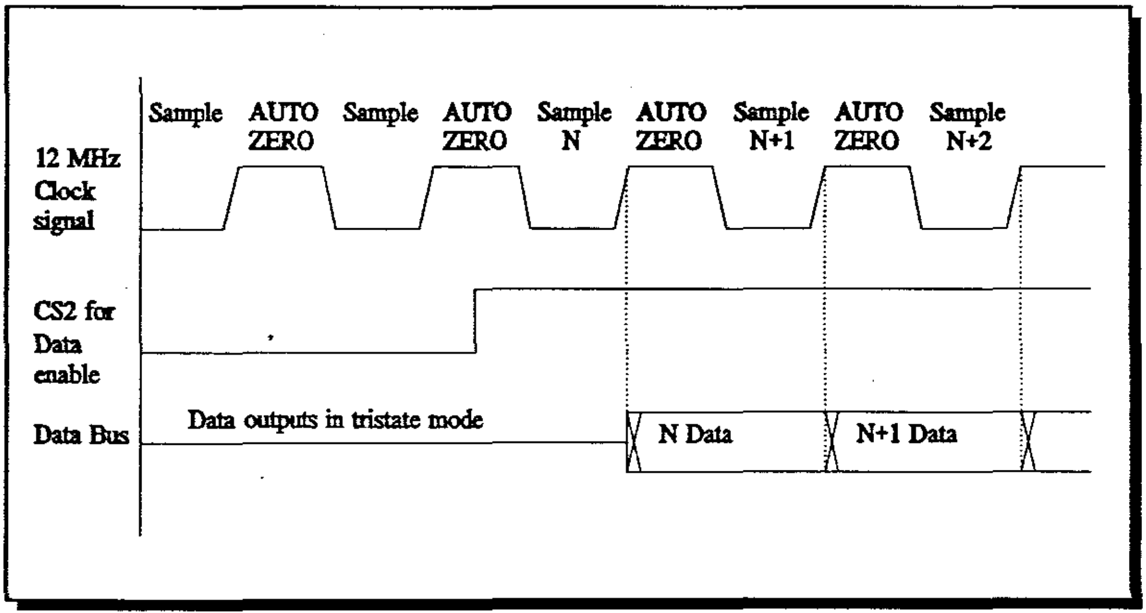


Figure 10.2 A-D timing diagram for video capturing

The A-D is at all times fully operational in the frame grabber system. The data outputs of the ADC 207 are switched onto the data bus while capturing is to taking place . The CS2 line is used for the synchronous switching of data onto the data bus while capturing a video frame. The clock signal used for the ADC-207 is the same as the 12 MHz clock signal used for updating the address counters.

10.2.2 <u>Tristate buffer (UTBUF1)</u>

UTBUF1 is used as a input buffer for the protection of the data bus from the input section. The direction switching from input to output is controlled by the frame grabber controller

(UCONTROL).

10.2.3 Xilinx XC3020 Frame Grabber Controller (UCONTROL)

The Xilinx frame grabber controller is a field programmable gate array programmed to take control of every section of the video frame grabber. This section provides all the control, address and clock lines for video frame grabbing. The pin assignment of the frame grabber controller is clearly shown in Appendix A drawing no 3. The pin definitions were set to obtain the minimum track lengths for the design of the data and address busses on the main circuit board for optimum performance at very high operating speeds of above 10 MHz. A full description of the internal structure of the Xilinx controller is given chapter 11.

10.2.4 Serial PROM (UPROM)

UPROM is a serial Xilinx programmable ROM for supplying the start-up data for initialization and programming of the Xilinx XC3020 FGPA (Field Programable gate Array).

On power on a 1 MHz clock signal is generated by the Xilinx XC3020 to down-load serial data, for initialization of the Xilinx XC3020, from the PROM. This serial data in used for the programming of the Xilinx XC 3020 to form the Xilinx video frame grabber controller. The down loading process is stopped after a successful transfer of data by the DONE/Prgm pin on the XC3020.

Any new modification or improvements may be done by replacing the information in the serial PROM by a newer version.

10.2.5 <u>Static RAM</u>

The static RAM of 512K bytes, used for storing the captured video data, is configured into 16 (32K x 8) single static RAM IC's. The RAM CY7B100 is a 262144 bit static random access memory organized as 32768 word by 8-bits using CMOS technology, and operated from a single +5V supply. Advanced circuit techniques provide both high speed and low power features with an operating current of 5 mA/MHz and a minimum cycle time of 20ns.

When CE is a logic high the device is placed in low power standby mode in which the standby current is 100 micro-ampere. This ensures very low overall RAM power consumption of the memory section of the main circuit board.

A full description of the static RAM section is given in chapter 13 showing all the timing waveforms for data manipulation.

11 XILINX FRAME GRABBER CONTROLLER

11.1 Introduction

The Xilinx frame grabber controller is used for the management of the main circuit board, including all PC I/O operations and control circuitry for storing the digitized video signal into Static RAM.

This controller supplies all the control,data and address lines necessary for the management of the frame grabber unit. A full schematic diagram of the main circuit board showing all the control, data and address lines are included in Appendix A drawing no 3.

11.2 Design considerations

For the controlling of the main circuit board, control circuitry had to be designed to work with the rest of the hardware. The following sections were needed in the design of the Xilinx frame grabber controller:

- A 18-Bit Address counter for accessing 512 Kbytes of static memory.
- 16 Address select lines for selecting a specific 32K memory block (SRAM size of 32K X 8 for a single ram IC) out of a total of 512K of available RAM on the main circuit board.
- Decoding of the address selected by the PC for reading and

writing of information between the two devices.

- Setting the capture duration to capture a full video frame, starting with the EVEN field and terminating with a ODD field.

11.3 Internal structure

The internal structure for the Xilinx Controller is shown in Appendix A drawings no 4 to 11. For the ease of explanation the drawings are shown as schematic diagrams drawn with the ORCAD-Schematic editor. A special library interface is used between the ORCAD-Schematic editor and the Xilinx Netlist Compiler. Non Standard logic units are used in the schematic drawings. They will be described together with the circuit description of each individual section. Each of the external pins of the Xilinx XC3020 must be defined accordingly to the internal circuit diagram as designed with the schematic editor. Figure 11.1 shows the definition of a few of the output pins as defined in a schematic diagram.

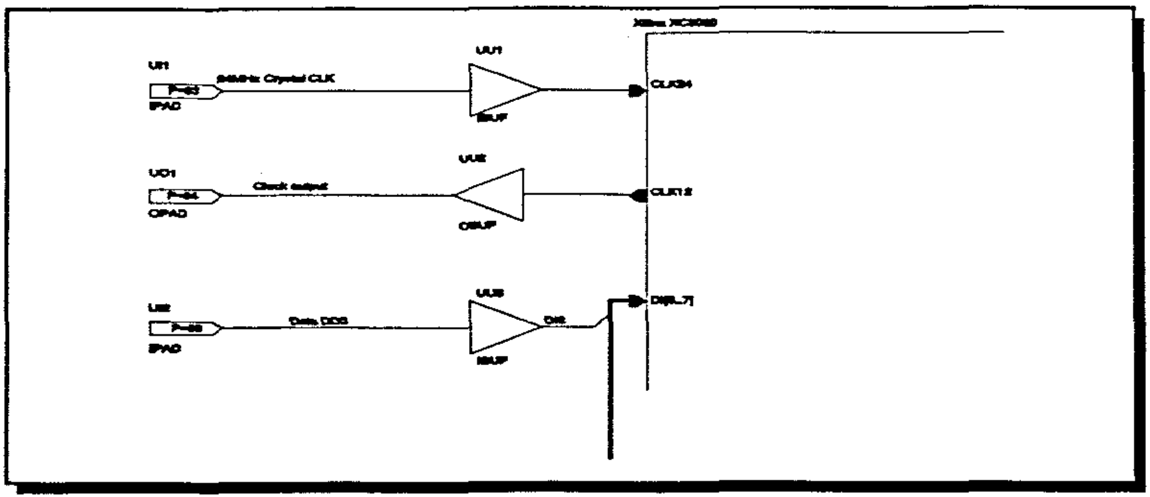


Figure 11.1 External PIN to Schematic interface

11.4 Xilinx Internal sheet connection

The internal circuitry as show in Appendix A drawing No 4 is divided into the following sections (Schematic Sheets).

11.4.1 ClockDiv.SCH (Appendix A Drawing No 5)

This section is used for the division of the crystal clock generator to a value of 12 MHz for video sampling and 44,1kHz for Audio Sampling (The audio sampling is an add on section not discussed in the thesis report).

11.4.2 InputMux.SCH (Appendix A Drawing No 6)

This section is used as a multiplexer between the PC and the frame grabber unit for controlling the read and write operations to and from the static memory on the main circuit board. During sampling the RAM controlling is done by the Frame Grabber controller. The PC takes over RAM controlling after a frame has been captured.

11.4.3 ReadWrite.SCH (Appendix A Drawing No 7)

This section is used for the decoding of the control lines as supplied by the address decoder card in the PC. These lines are divided into address latching lines and RAM read and write enable lines for the controlling of static memory on the main circuit board.

11.4.4 OneFrame.SCH (Appendix A Drawing No 8)

Oneframe.sch sheet diagram is used to synchronise the capturing process with the start of a video frame. This section sets the capturing duration to a minimum of one video frame for a click of the capture button.

11.4.5 Counter.SCH (Appendix A Drawing No 9)

This section consist of an 18-bit synchronous counter for suppling an address count while a video frame is being captured.

11.4.6 373.SCH (Appendix A Drawing No 10)

This section consist of an 18-Bit address latch. This 18-Bit address latch is used for supplying a unique address from the PC for the reading and writing to and from the static RAM of the frame grabber unit.

11.4.7 AddressMux.SCH (Appendix A Drawing No 11)

This sheet shows 18-bit address multiplexing between an address supplied by the 18-Bit counter and a latched 18-bit address supplied from the PC. The address multiplexer is used for the selection of either the PC or the Frame grabber controller to control the address bus of the main circuit board.

11.4.8 4-16 Decoder (Appendix A Drawing no 4)

This Xilinx defined module has the same functions as a 74HC154 CMOS 4 to 16 line decoder. The upper 4 address bits A15..A18 is decoded to form 16 control lines to access any of the 16 (32K x

8) Static RAM IC's.

11.5 ClockDiv.SCH - Circuit operation

The clock divider section is used for the supplying of a very stable clock signal of 12 MHz or 44,1 kHz for the synchronisation of the address lines and sampled digital data as recieved from the analog to digital converter.

The input of this section, [CLOCK] is a stable 24 MHz signal produced by a crystal oscillator on the main circuit board. The A/V line selects between a 12 MHz clock (Video sampling) and a 44,1 kHz clock (Audio sampling) frequency for the main clock for the A-D converter and the 18 Bit address counters.

The 24 MHz [CLOCK] signal is divided by a toggle flip-flop [FT0] to give a 12 Mhz clock frequency for the sampling of video information. A 44,1 kHz clock frequency for audio sampling is obtained by dividing the 24 MHz signal by a factor of 544.

The division is obtained by the combination of a 2 stage 9-Bit synchronous counter and a toggle flip-flop. The [CLOCK] 24 MHz signal is divided by 272 by UCLK1 and UCLK2. UCLK3 is used to reset the synchronous counters on a count of 272. A positive edge triggered toggle flip-flop UCLK4 is used to divide the clock frequency further by a factor of 2 and to restore the duty cycle to 50/50 for a stable clock signal of 44,1 kHz. The multiplexer UCLK5 is used for the selection between video and audio clock frequencies. A high on the A/V control line will enable a clock for video sampling.

11.6 InputMux.SCH - Circuit Operation

The input multiplexer selects between the two different modes of operation - PC Control over RAM

- Frame Grabber control over RAM

The multiplexer is designed by using standard logic gates due to the absence of a multiplexer eg 74LS157 in the standard Xilinx library. Table 11.1 gives the layout of all the input and output of the section.

AA/-BB	Input	Description	Output
1	AA1	Clock signal from ClockDiv.SCH	AA1
1	AA2	-WE of RAM (Same signal as AA1)	AA2
1	AA3	-RE of RAM = +5V	AA3
0	BB1	+ 5V to stop Address Counter	BB1
0	BB2	Write bit, Low to Write to RAM	BB2
0	BB3	Read bit, Low to read from RAM	BB3

Table 11.1 Input Multiplexer description

The AA/-BB line selects between the different modes of operation. The AA/-BB line is controlled by the OneFrame.SCH section. This line is used for capturing only a single video frame. Figure 11.2

shows the output of YY1 for a change in modes.

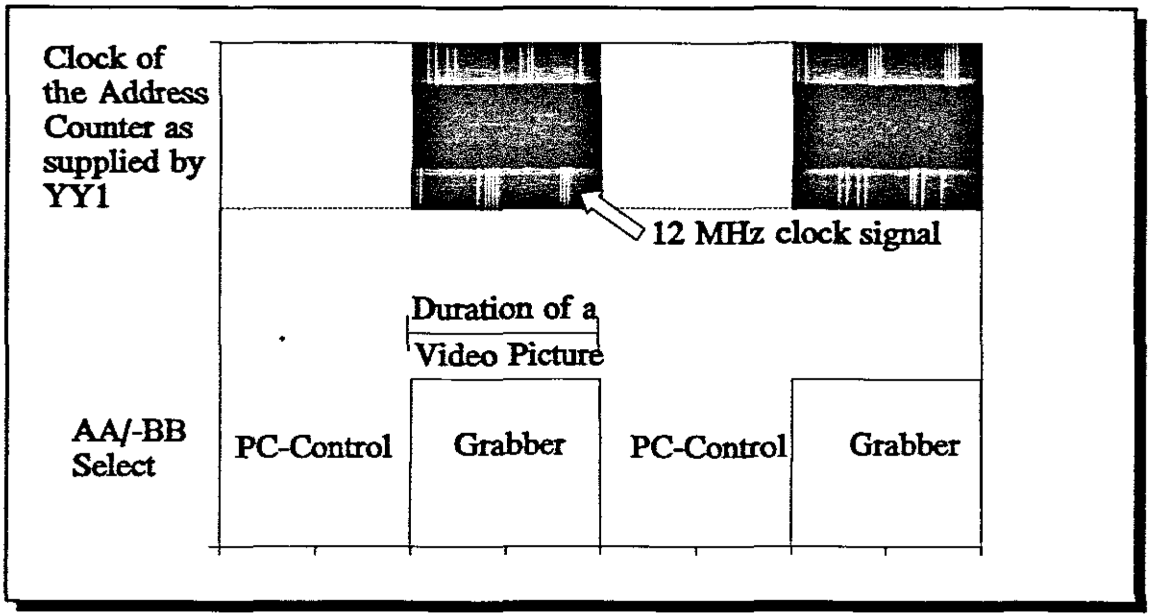

Clock of the Address Counter as supplied by YY1
12 MHz clock signal
Duration of a Video Picture
AA/-BB Select
PC-Control
Grabber
PC-Control
Grabber

Figure 11.2 Clock signal for the different modes of operation.(PC Mode or Frame Grabber mode).

11.7 ReadWrite.SCH - Circuit Operation

The ReadWrite select is used as a intelligent interface to identify the different control lines supplied by the PC. Five control lines:

- #300h Low part of address
- #301h Middle part of address
- #302h High part of address
- #303h Read data select line
- #304h Write data select line

are decoded to form the Ram Controlling lines as used in the frame grabber unit. The RE line is taken directly from the read PC line. Whenever the PC address #300h is called a low pulse on this line will appear, the latter is used for the controlling the OE line of the static RAM on the main circuit board.

The WE line may be taken directly from the Write-PC line but it was found that this line may go low during RE operations due to cross interference in the connecting cable. To prevent any accidental "writes" into the memory while reading, this line is decoded by a 5 input NAND gate [USTOP].

The direction line RD/WE-DIR changes the flow of data through the tristate input buffer on the main circuit board. The normal direction is set from PC to frame grabber for the controlling of the frame grabber unit. To read a byte from the external memory of the Frame Grabber unit the direction needs to change from the grabber to the PC. The automatic direction changing is done by the decoding of the control lines for a Read-PC pulse. This decoding is done by UDIR1 and UDIR2.

The low,med,high lines are used for the latching of the three different sections of the 18-bit PC address on the address bus.

11.8 <u>OneFrame.SCH - Circuit Operation</u>

The capture button of the video frame grabber needs to be synchronised with the video signal for the capturing to start at the correct position in the video signal. The [25 Hz] signal is derived from the ODD/EVEN field information received from the sync separator. This [25 Hz] signal is in synchronisation with the video signal and is used to synchronise the capture button to the incoming video signal. Video capturing starts as soon as the capture button is pressed. The video capturing will continue after the button has been released until a full frame has been captured. The [25 Hz] signal is used for the release delay of the capture button for capturing a full video frame. U116, U115 and U113 form the controlling circuitry for the release delay of the capture button. Figure 11.3 shows the timing diagram of the capture delay circuitry.

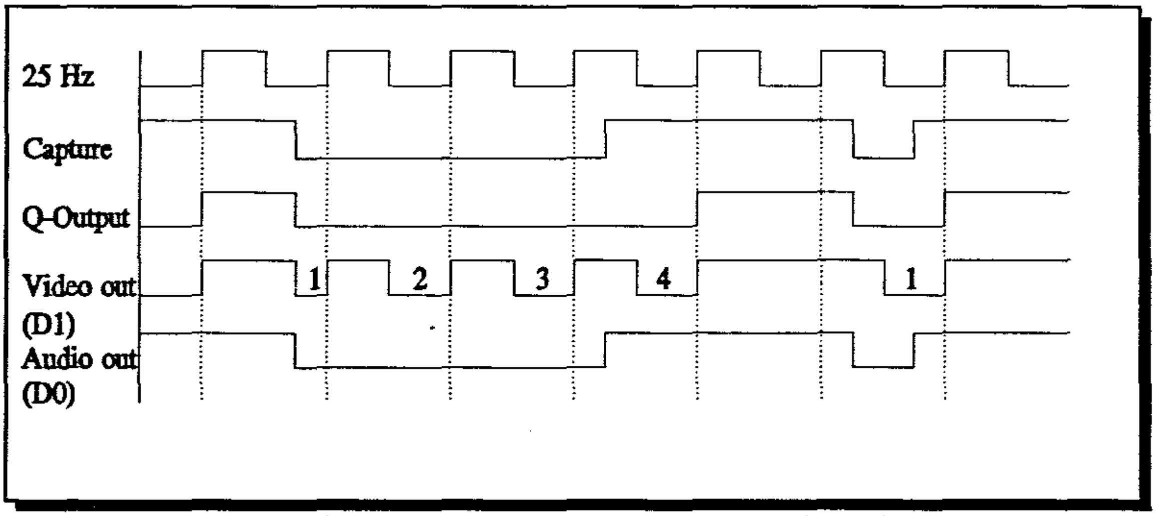


Figure 11.3 Timing waveform capture switching

11.9 <u>Counter.SCH - Circuit Operation</u>

The 18 bit address counter consists of two 8 bit and one 3 bit synchronise counters [UCNT0,UCNT1,UCNT2].The C256BCDR is a 8 bit synchronous counter with a direct reset and a carry output for cascading of the counters. The carry output of the two 8 bit counters is marked with a critical flag. This critical flag is used to minimise the track length between the two counter modules in the internal design of the XC3020 as auto-routed by the XACT auto-router. For the optimum performance the length of these lines must be a minimum to give a very low delay factor from counter to counter. The buffer UCLK is a special buffer used for driving the clock signals of high speed counters. In the Xilinx internal design all the CLK inputs must be connected to the CLOCK bus for optimal performance. The reset pins of the three counters is controlled by the Oneframe selector. Figure 11.4 shows the 25 Hz Capture pulse together with a single address line.

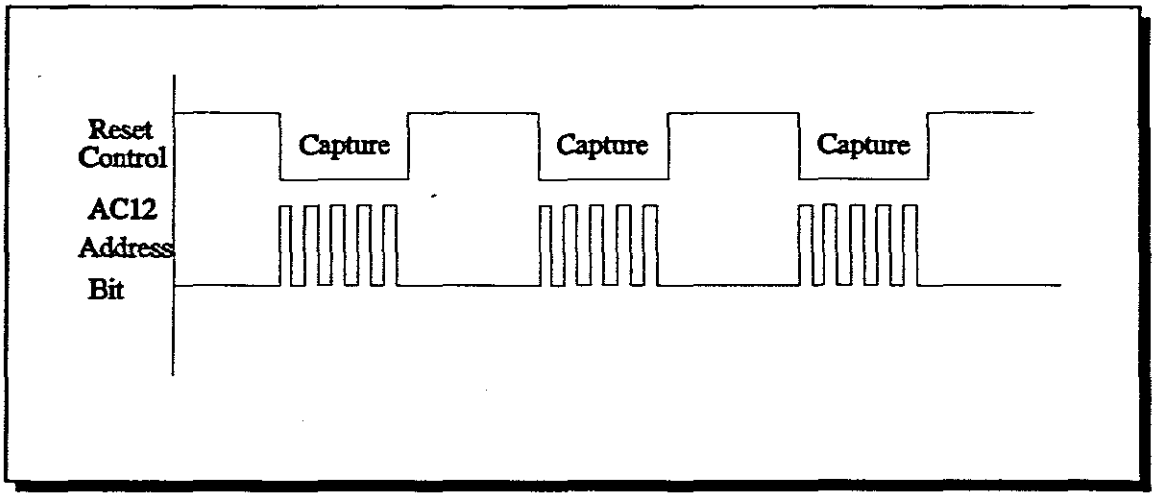


Figure 11.4 A single address line while capturing

11.10 373.SCH - Circuit Operation

The Xilinx HX373 library unit is used for the latching of data information to the internal address bus in the Xilinx frame grabber controller. The HX373 library unit is pin for pin compatible with the 74LS373.

The 373 latching circuitry is used for supplying a unique 18 bit address value on the address bus of the frame grabber unit. The gate pins G1,G2 and G3 are used for latching address data information from the PC onto the address bus of the frame grabber unit. The gate pins G1,G2,G3 corresponds to the low,med,high control lines as supplied by the address decoder card.

11.11 <u>AddressMux.SCH - Circuit Operation</u>

The 18 bit address multiplexer is based around 18, 2 to 1 line multiplexer modules as defined in the Xilinx library (Appendix A Drawing No 11).

A single control line [L/-CNT] is used for the selection between an address supplied by the address counter and the output from the 373 latching circuitry. The address multiplexer is used for giving both the PC and the frame grabber controller control over the address bus in the main circuit board.

12 MEMORY BOARD

12.1 Introduction

The frame grabber unit uses 512 k of static memory configured in 16 (32k X 8) static RAM modulues. The 32k x 8 Static RAM modules used are the same as those used for cache memory on 486 motherboards. They have very high speed data manipulation caracteristics with access times for RE/WE operations of 20 ns. Dynamic RAM's is a cheaper solution but they require extensive control circuitry for RAM refreshing while data is being transferred. Static RAM's requires no RAM refreshing and are easier to control when operating at very high data transfer rates. Appendix A drawing No 12 shows the schematic diagram of a 256K memory block as used on the main circuit board. The total memory on the frame grabber board consist of two of these 256K modules.

12.2 Theory of operation

The RAM is configured in 32K memory blocks. Each memory block is activated by a select line generated by the Xilinx frame grabber controller. The select line selects a unique memory block for reading and writing of data (Captured) information.

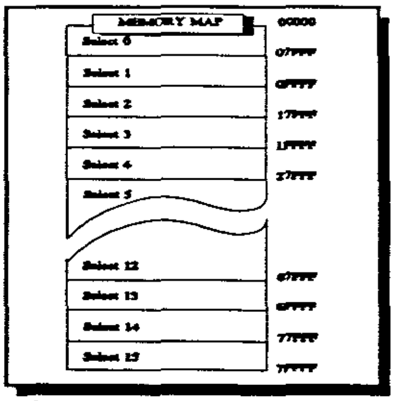


Figure 12.1 Memory Map

See figure 12.2 for the complete timing waveforms of the RAM controlling.

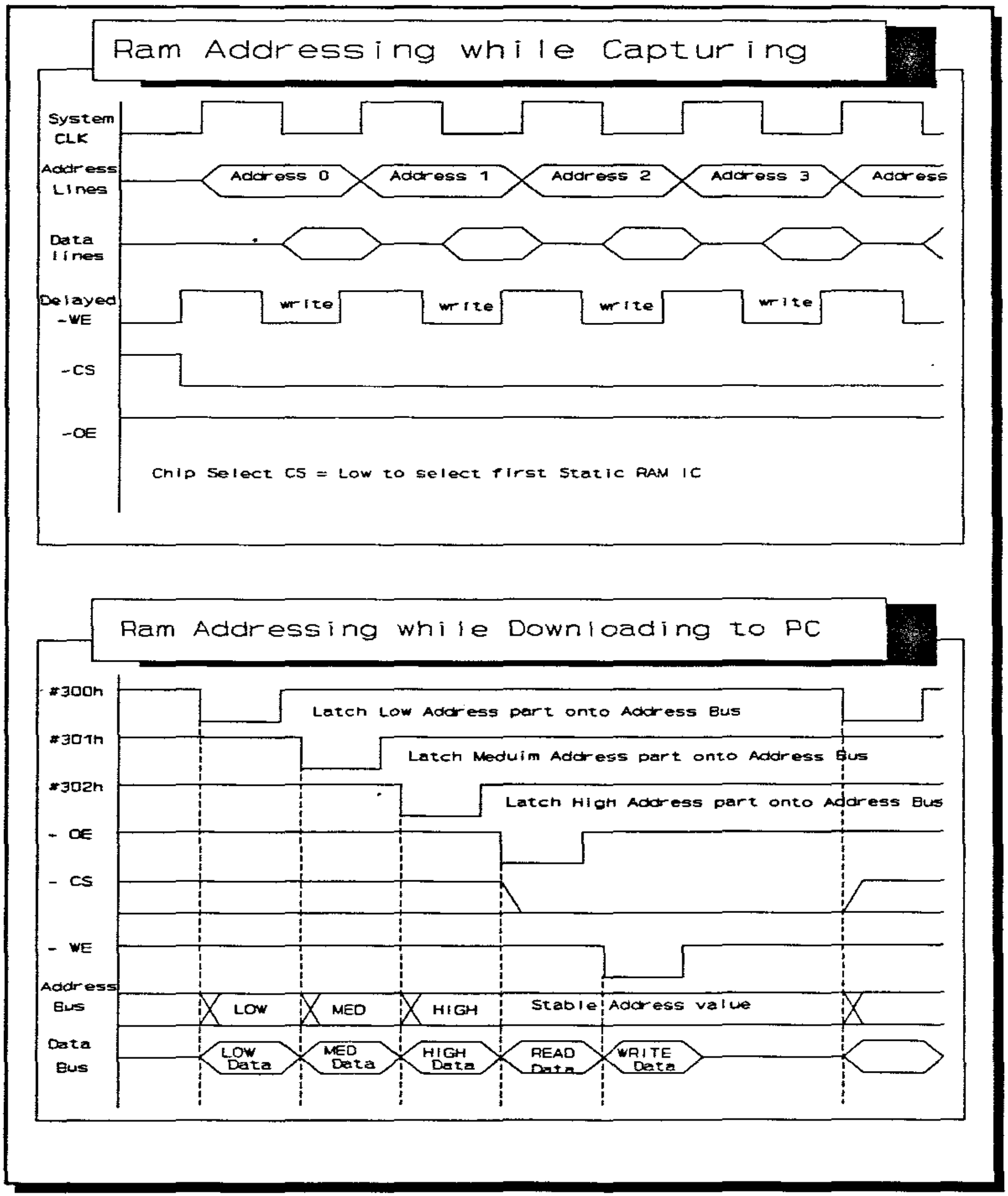

Ram Addressing while Capturing
System CLK
Address Lines
Address 0
Address 1
Address 2
Address 3
Address
Data lines
Delayed -WE
write
write
write
write
-CS
-OE
Chip Select CS = Low to select first Static RAM IC
Ram Addressing while Downloading to PC
#300h
#301h
#302h
Latch Low Address part onto Address Bus
Latch Meduim Address part onto Address Bus
Latch High Address part onto Address Bus
- OE
- CS
- WE
Address Bus
LOW
MED
HIGH
Stable Address value
Data Bus
LOW Data
MED Data
HIGH Data
READ Data
WRITE Data

Figure 12.2 Timing Diagram for Ram Accessing

13 POWER SUPPLY

13.1 Introduction

A very stable power supply is required for the accurate conversion of an analog to digital signal. The power supply section must be as noise free as possible. Drawing No 13 in appendix A shows the schematic diagram of the regulated power supply section as used in the frame grabber unit.

13.2 Design Considerations and Calculations

13.2.1 Considerations

The frame grabber unit were designed to work from a standard PC-Supply. A linear regulated power supply section is used for the final design due to the bulkiness of a PC-supply. The currents measured in Table 13.1 were measured by using a standard PC-Supply. These values were used for the final design of the regulated power supply section.

	+12V	+5V	-12V
Current drawn in (mA)	109.6	186.0	17.0

Table 13.1 Current dissipation of the Frame Grabber unit

13.2.2 Calculations

A 220/30 V centre tap mains transformer [T1] is used. The centre tap is used for giving a +21 V and -21 V dual supply rail.

Maximum DC Voltage after rectification = $15*\sqrt{2}$ - 0,7 V
= 21,213 - 0,7 V
= 20,513 V

Minimum Voltage for stable 12 V regulation = 12 + 1,5 V
= 13,5 V

{ The LM7812 requires a minimum of 1,5 V from input to output for good regulation. }

Current dissipation as measured and tabulated in Table 13.1

Total current drawn from the supply = 109,6+186,0+17 mA
= 312,6 mA

For save operation 500 mA supply current is used, with a maximum ripple voltage of 2 V (p-p).

Smoothing Capacitor Value = (I*T)/V
= (0,5 * 0,01)/2
= 2500 uF (Minimum value)

A 4700 uF capacitor were use for smoothing. See Appendix A drawing no 13 for the completed schematic diagram of the power supply section.

13.3 Circuit Operation

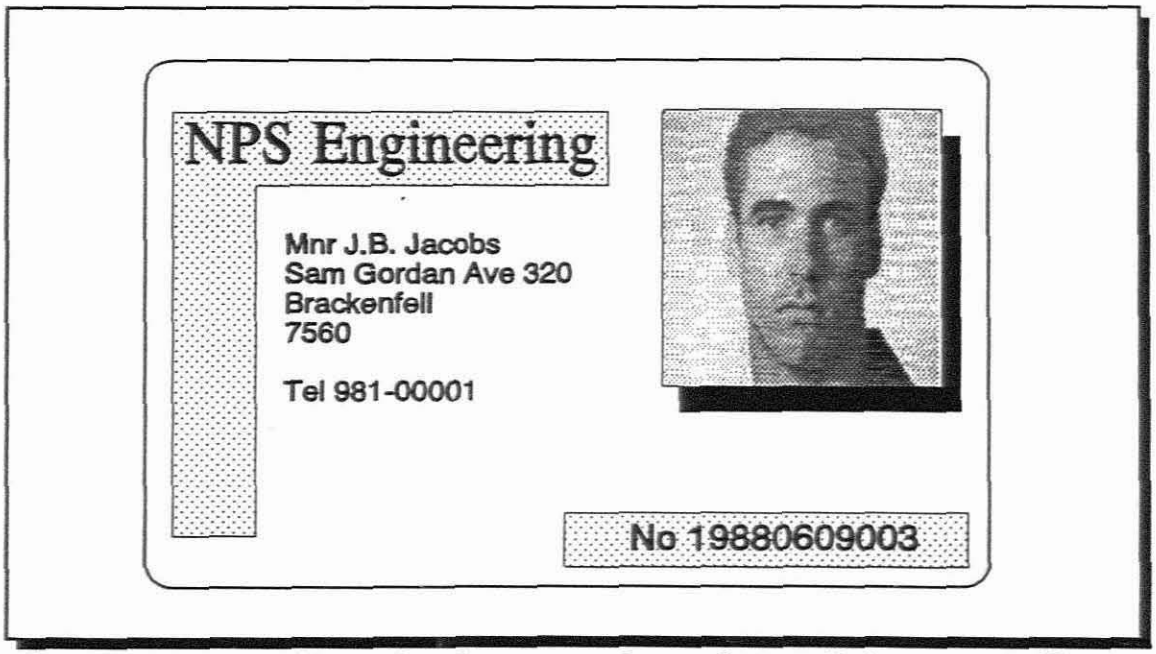
Rectification of the 15 V AC signal is done by D1,D2,D3 and D4 with C1 and C2 as the smoothing capacitors.

The positive (+21V) rectified line is regulated by two +12V and +5V regulators (U1,U2) for supplying positive supply outputs of 12V and 5V respectively. The negative (-21V) rectified line is regulated by a single -12V regulator (U3) to form a stable negative supply line.

14. FUTURE DEVELOPMENT

In the near future I would like to expand this video digitizer to a full colour video frame grabber. It can be done by splitting the video signal into it's primary components of R,G,B and luminace information. Each of these analog signals may be sampled in the same manner as used in the grey scale video frame grabber as descibed in this report.

The software program FR30.PAS can be revised to form a program for the printing of ID Card's or colour photo's. See figure 14.1 for a possible example of what can be done with the current frame grabber under control of a desktop publishing program.



NPS Engineering

Mnr J.B. Jacobs
Sam Gordan Ave 320
Brackenfell
7560

Tel 981-00001

No 19880609003

Figure 14.1 ID Card example for Future Software developments

15 GLOSSARY OF TERMS

A-D	-	Analog to Digital
B/W	-	Black and White
CCD	-	Charge Coupled Device
CGA	-	Colour Graphics Adapter
CMOS	-	Complementary Metal-Oxide Semiconductor
DOS	-	Disk Operating System
EGA	-	Enhanced Graphics Adapter
FPGA	-	Field Programmable Gate Array
IBM	-	International Business Machines
IC	-	Integrated Circuit
I/O	-	Input/Output
LED	-	Light Emitting Diode
PAL	-	Phase Alternate Line
PC	-	Personal Computer
PCB	-	Printed Circuit board
RAM	-	Random Access Memory
RGB	-	Red,Green,Blue
SABC	-	South African Broadcasting Corporation
VGA	-	Video Graphics Array

16 BIBLIOGRAPHY

Borland International. **Turbo Pascal 4.0 IBM Version.** Borland International, Inc.

Borland Oscborne McGraw-Hill. 1988. **Turbo Pascal Programmers Library second edition.** Berkley California 94710.

Cypress Semiconductor. 1991. **BiCMOS CMOS Data Book.** San Jose.

Intel Coporation. 1986. **iAPX 86/88 186/188 User's Manual Programmers Referance.** Santa Clara, Intel Literature Sales.

Intel Coporation. 1988. **Memory Products.** MC. Prospect, Intel Literature Sales.

National Semiconductors Coporation. 1988. **CMOS Data Book.** Santa Clara.

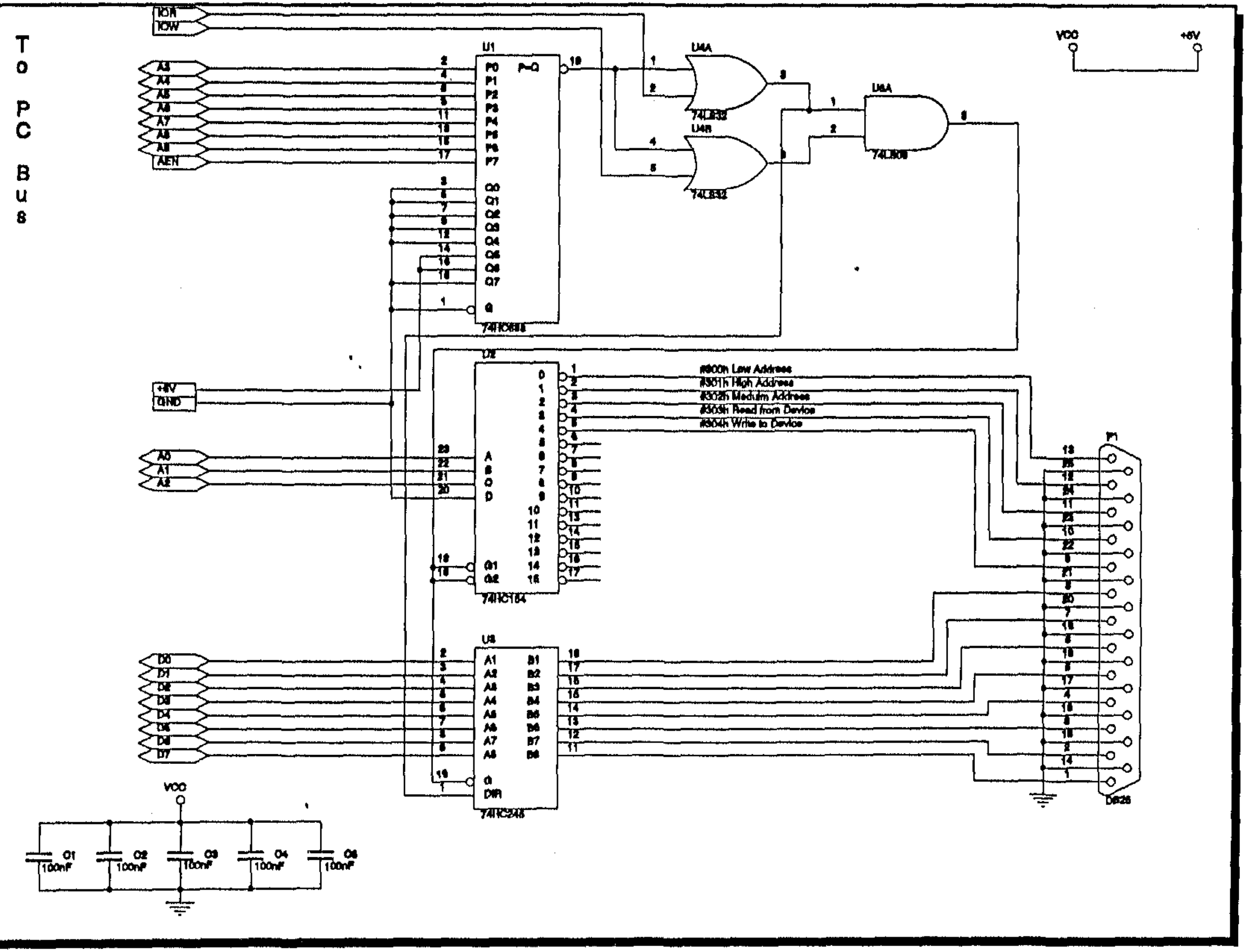
National Semiconductors Coporation. 1988. **Linear Data Book.** Santa Clara.

RS Export Centre. 1991. **RS Components Catalogue.** Corby. Northants England.

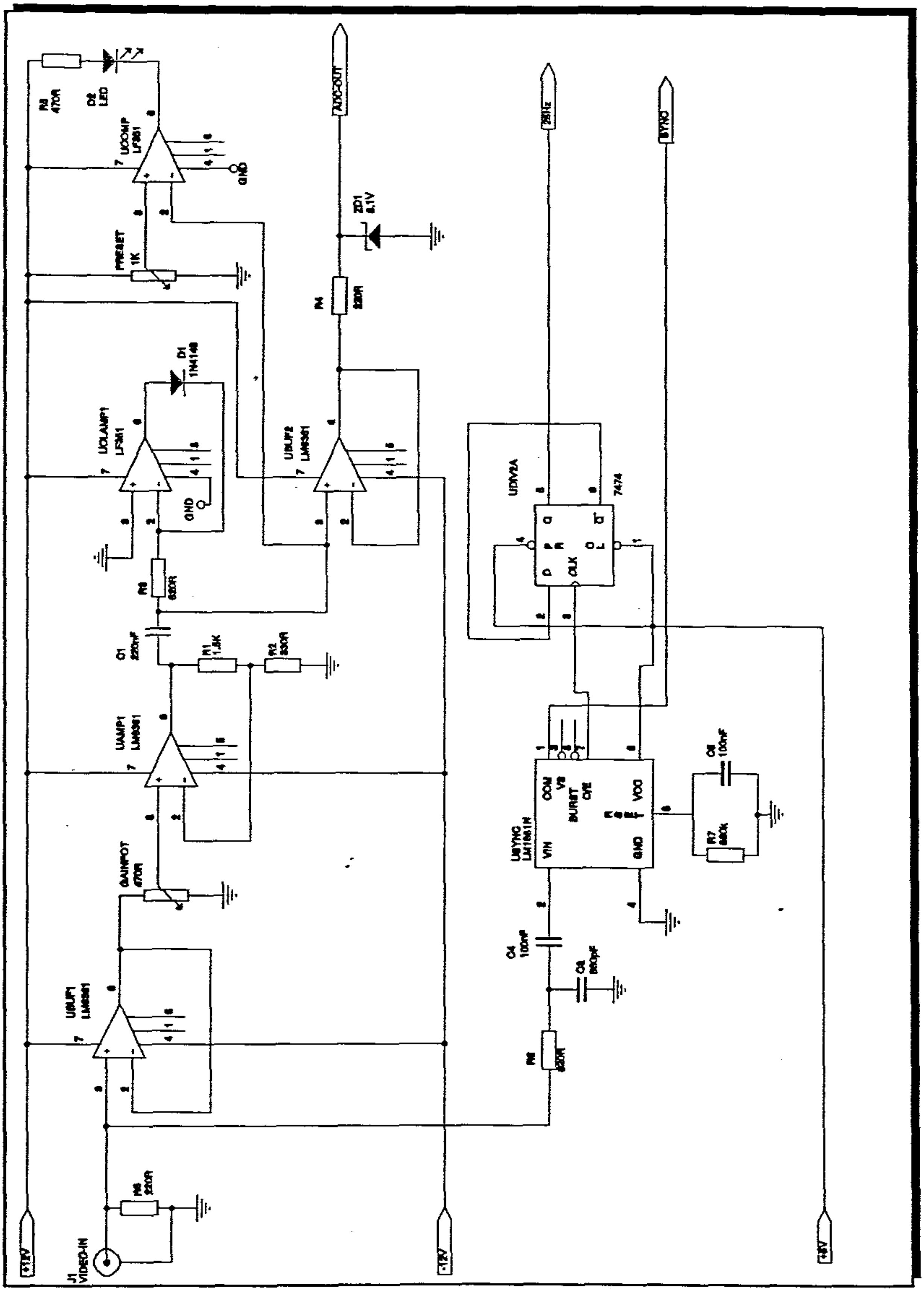
Xilinx The Programmabe Gate Array Company. 1992. **The Programmable Gate Array Data Book.** San Jose.

APPENDIX A

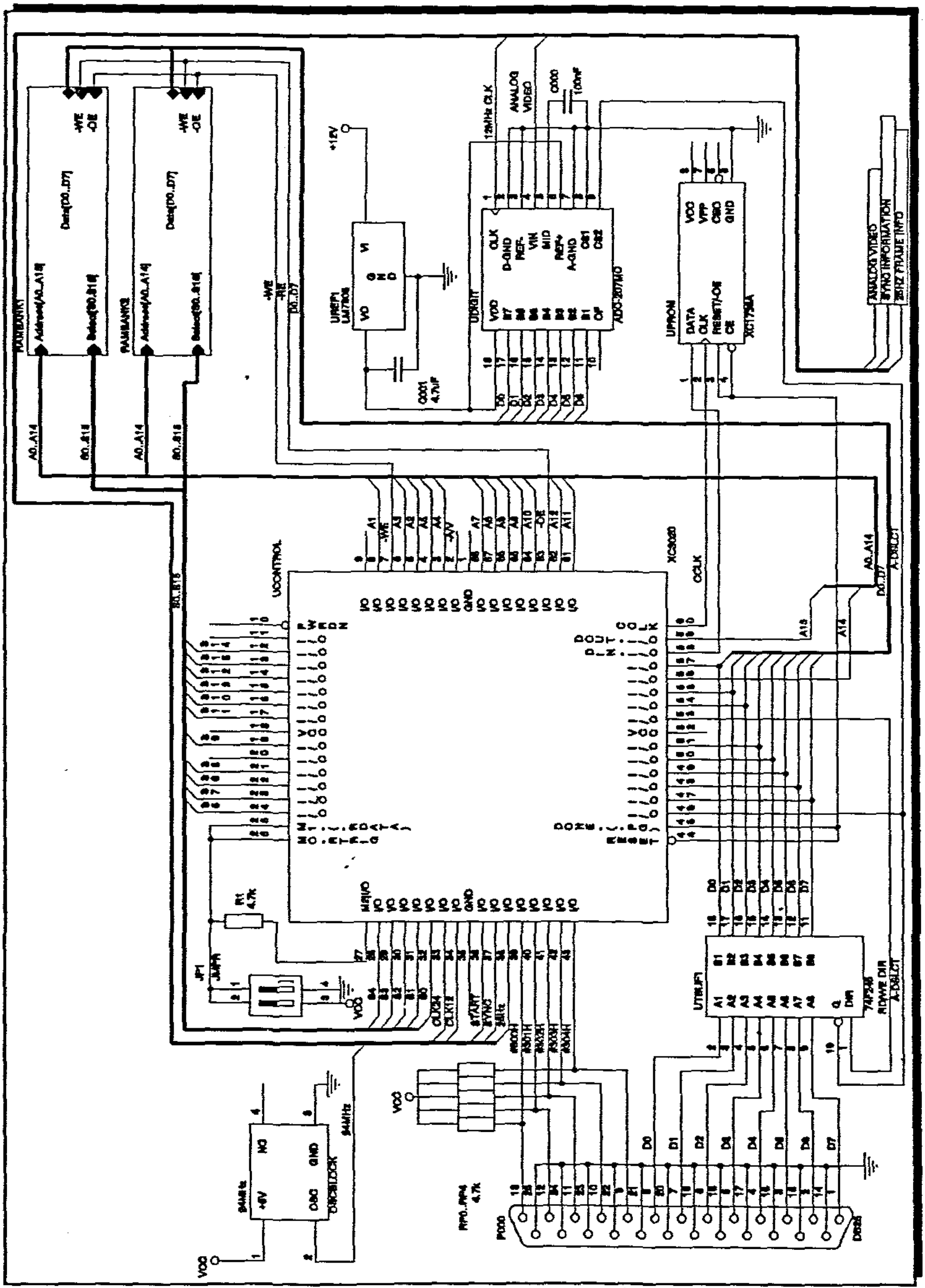
SCHEMATIC DIAGRAMS OF THE FRAME GRABBER SYSTEM



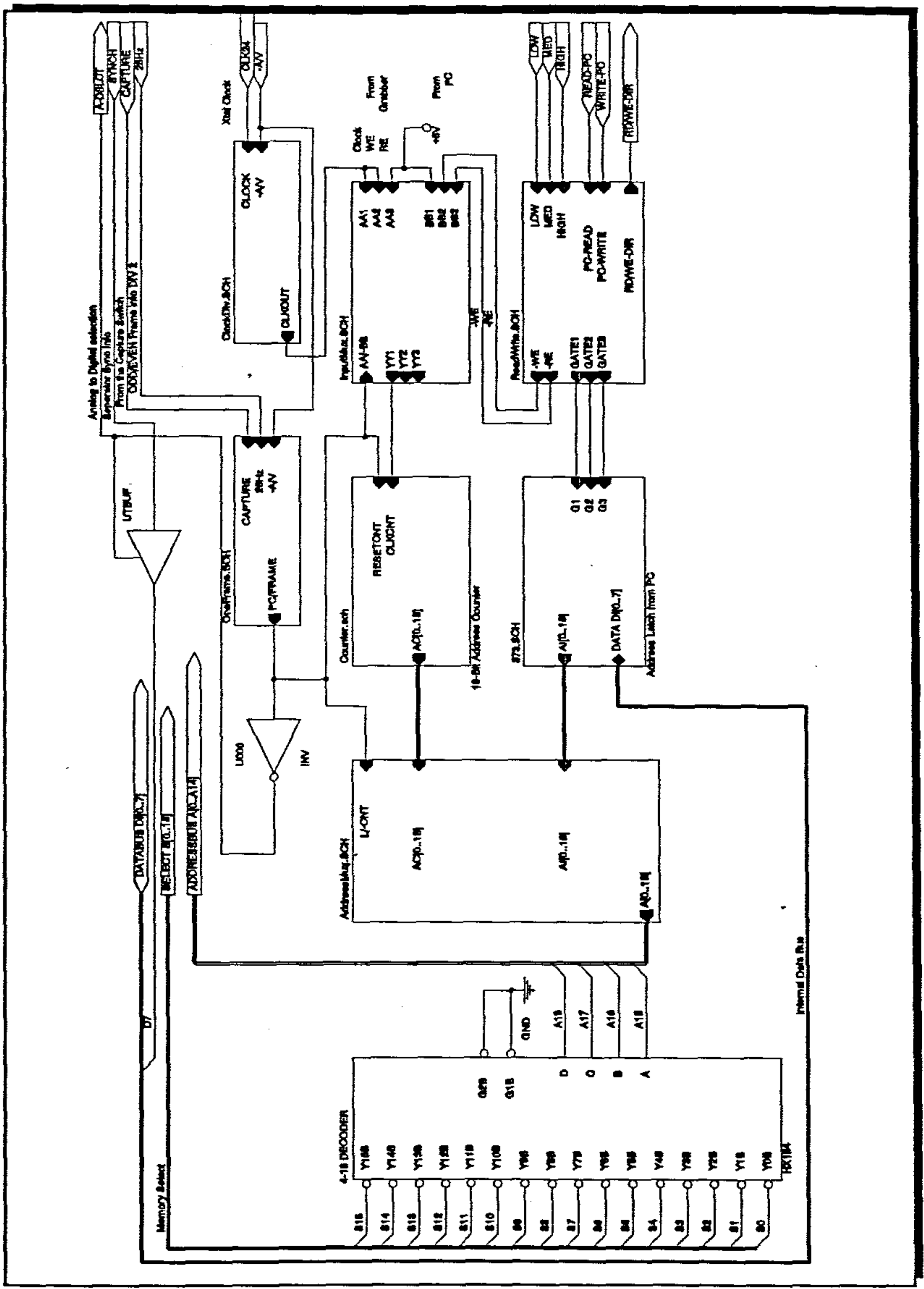
Drawing No 1 Address Decoder Schematic diagram



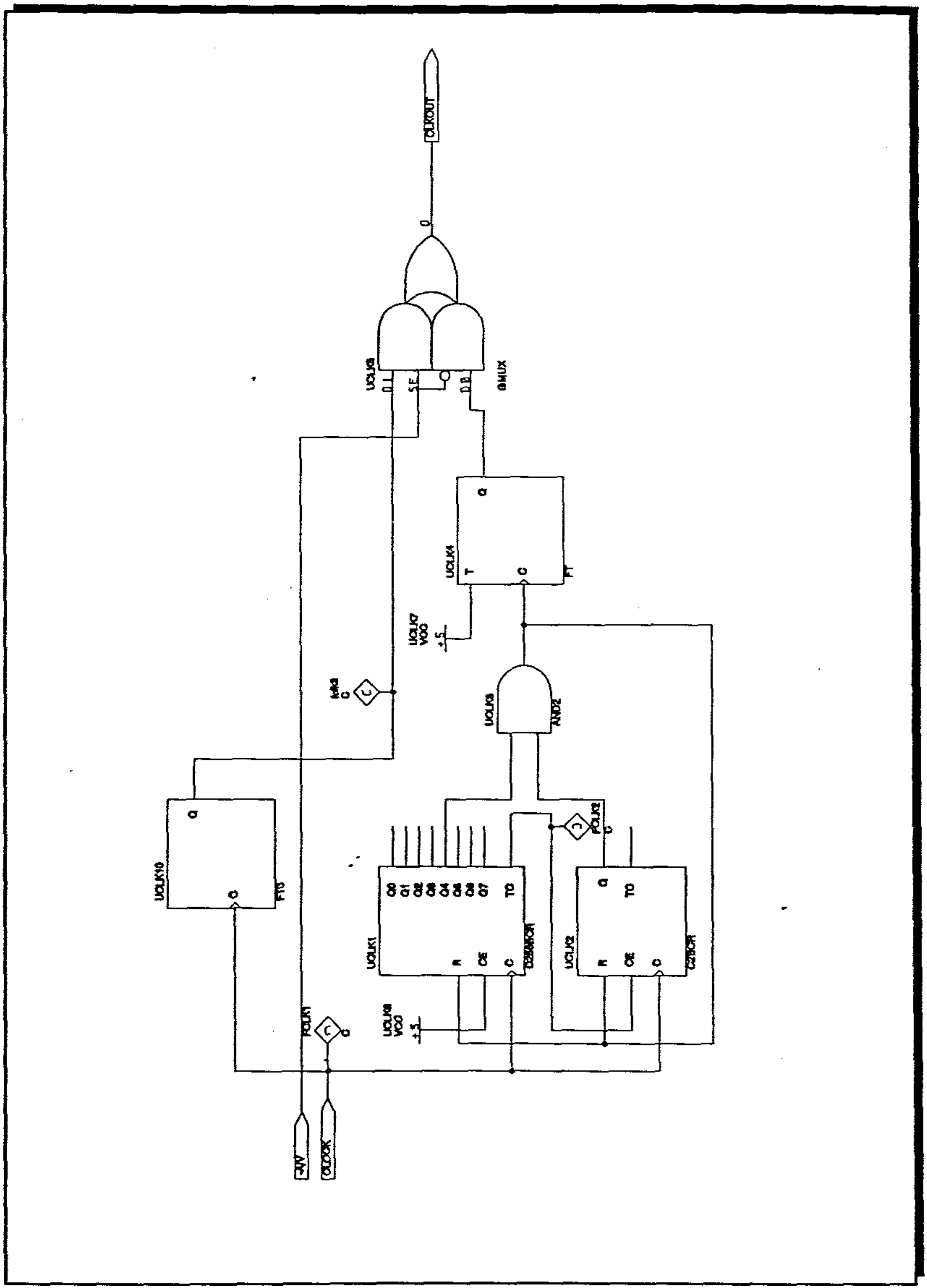
Drawing No 2   Schematic diagram of the video amplifier



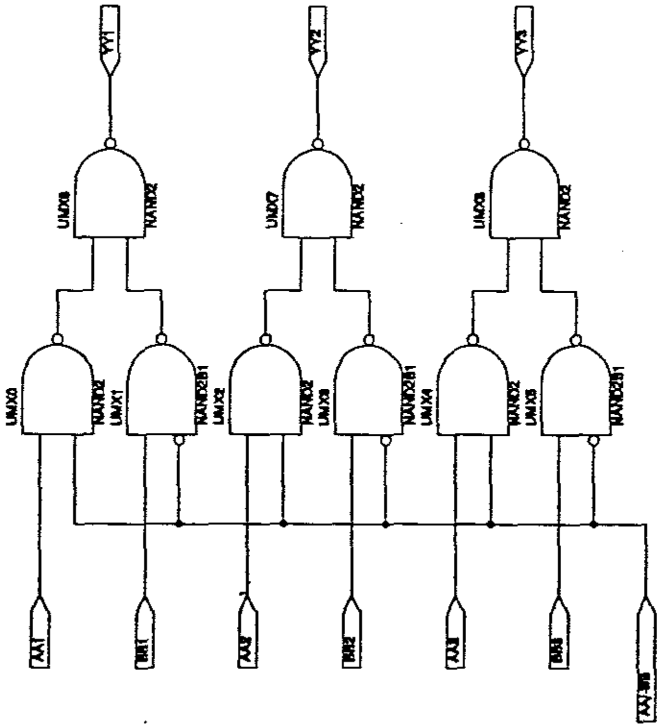
Drawing No 3   Schematic diagram of the Main Circuit board



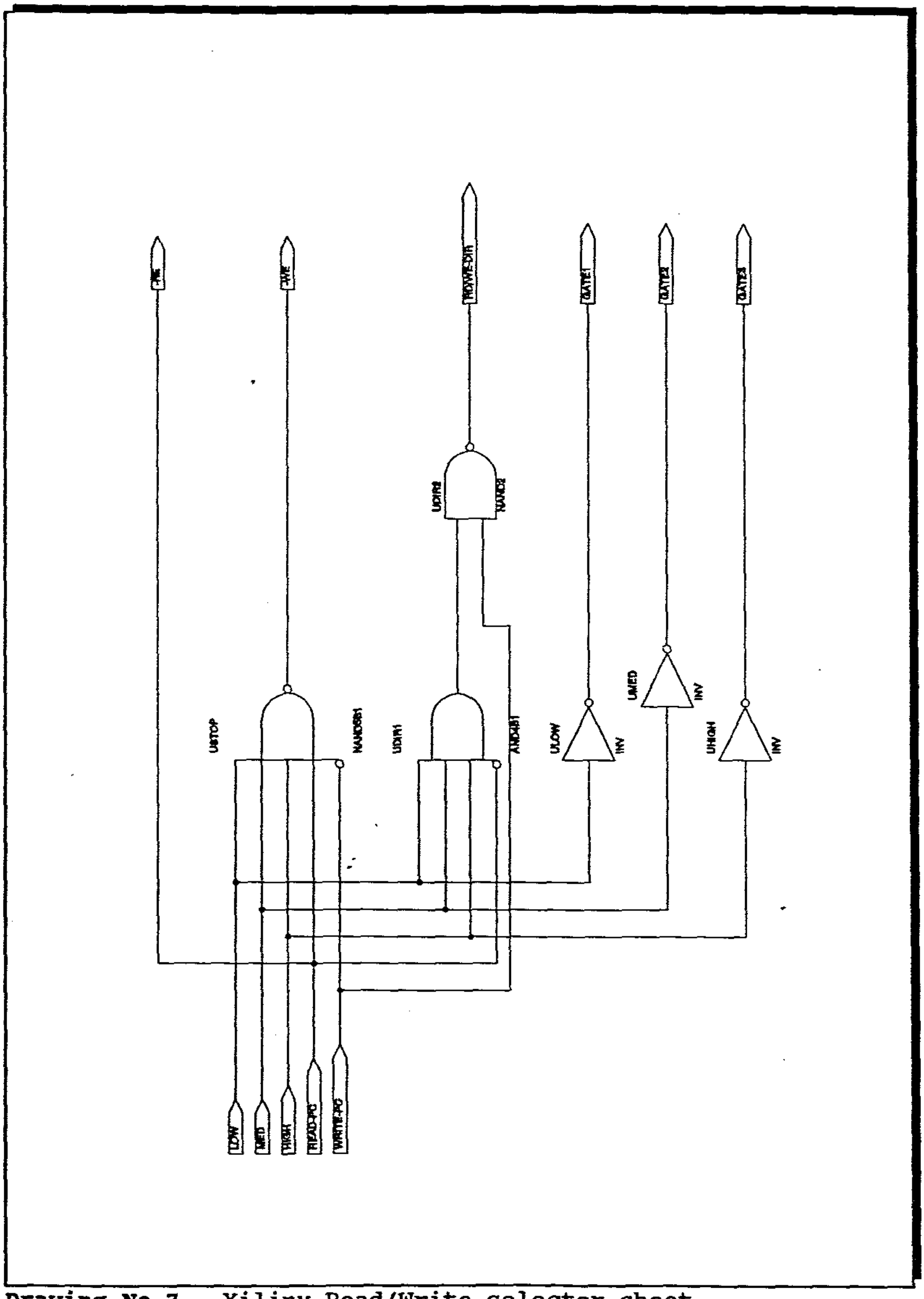
Drawing No 4   Xilinx internal sheet diagram



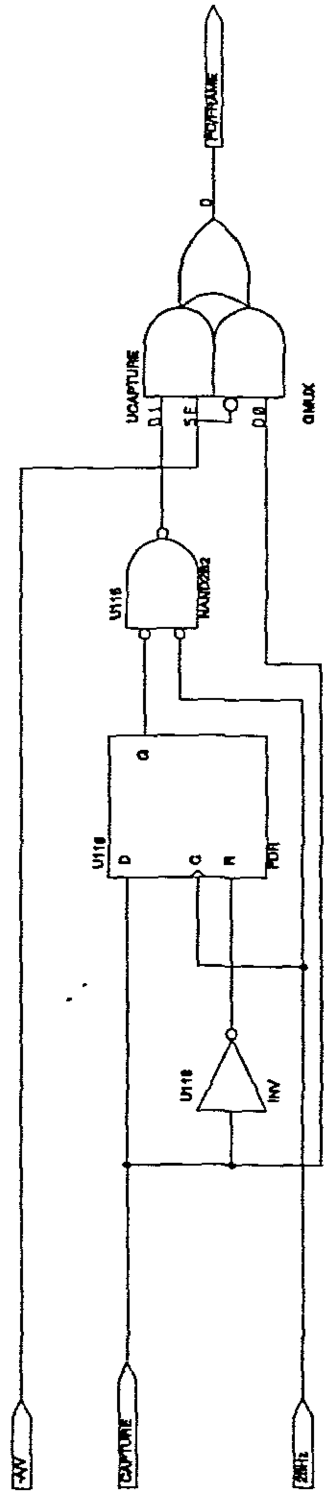
Drawing No 5 Clock divider schematic diagram



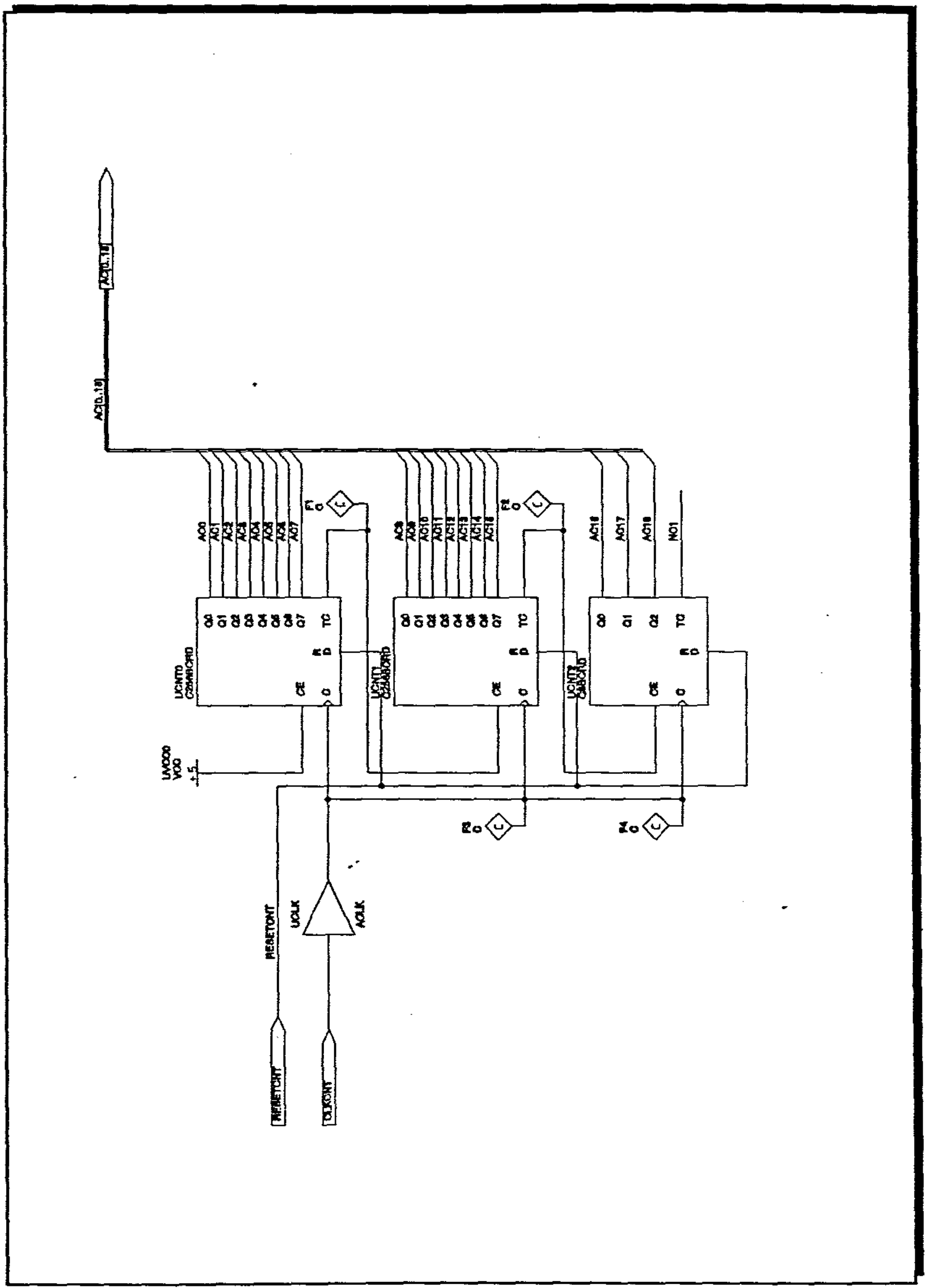
Drawing No 6 Xilinx Input multiplexer sheet



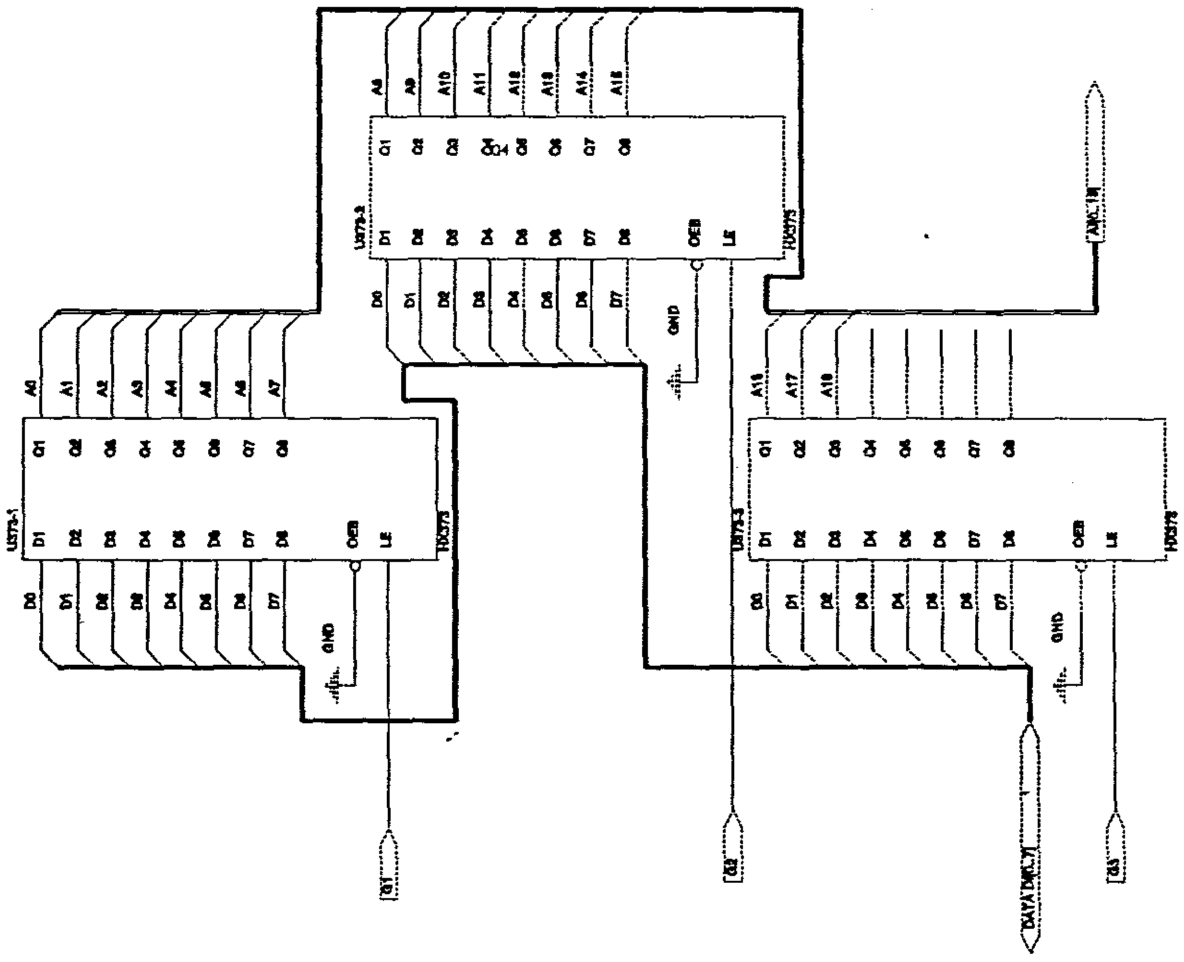
Drawing No 7   Xilinx Read/Write selector sheet



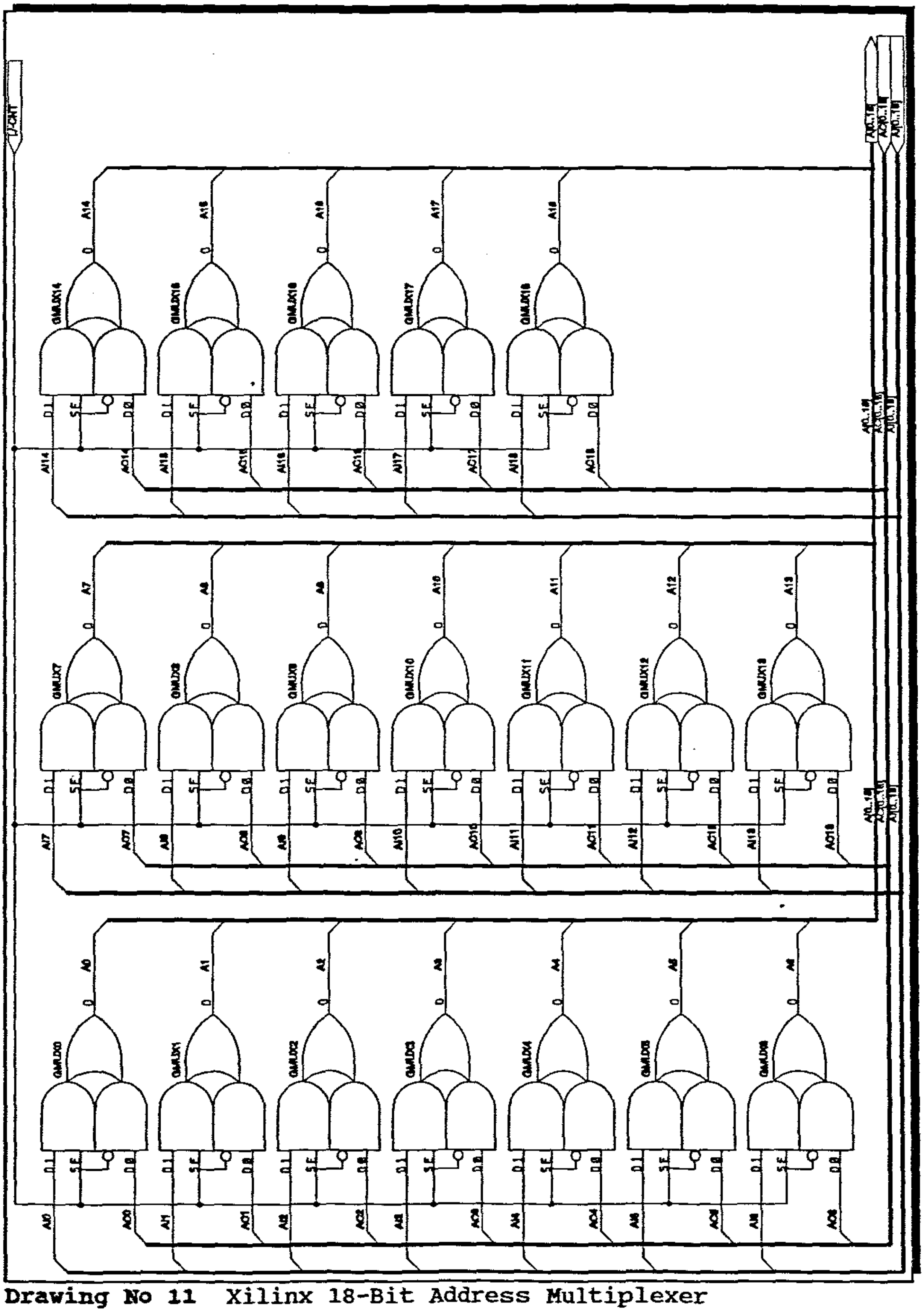
Drawing No 8 Xilinx Oneframe sheet



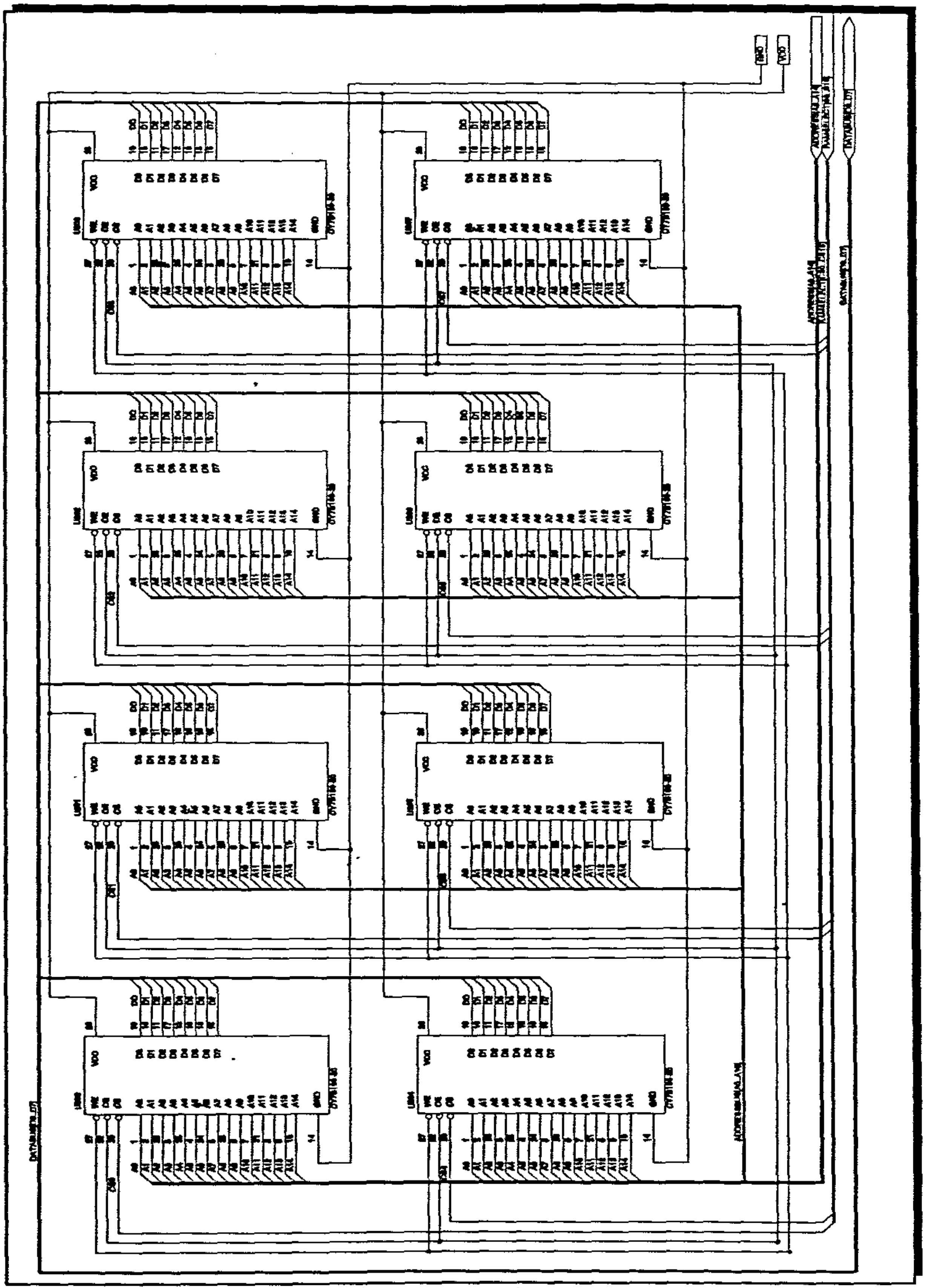
Drawing No 9 Xilinx 18-Bit Address Counter sheet



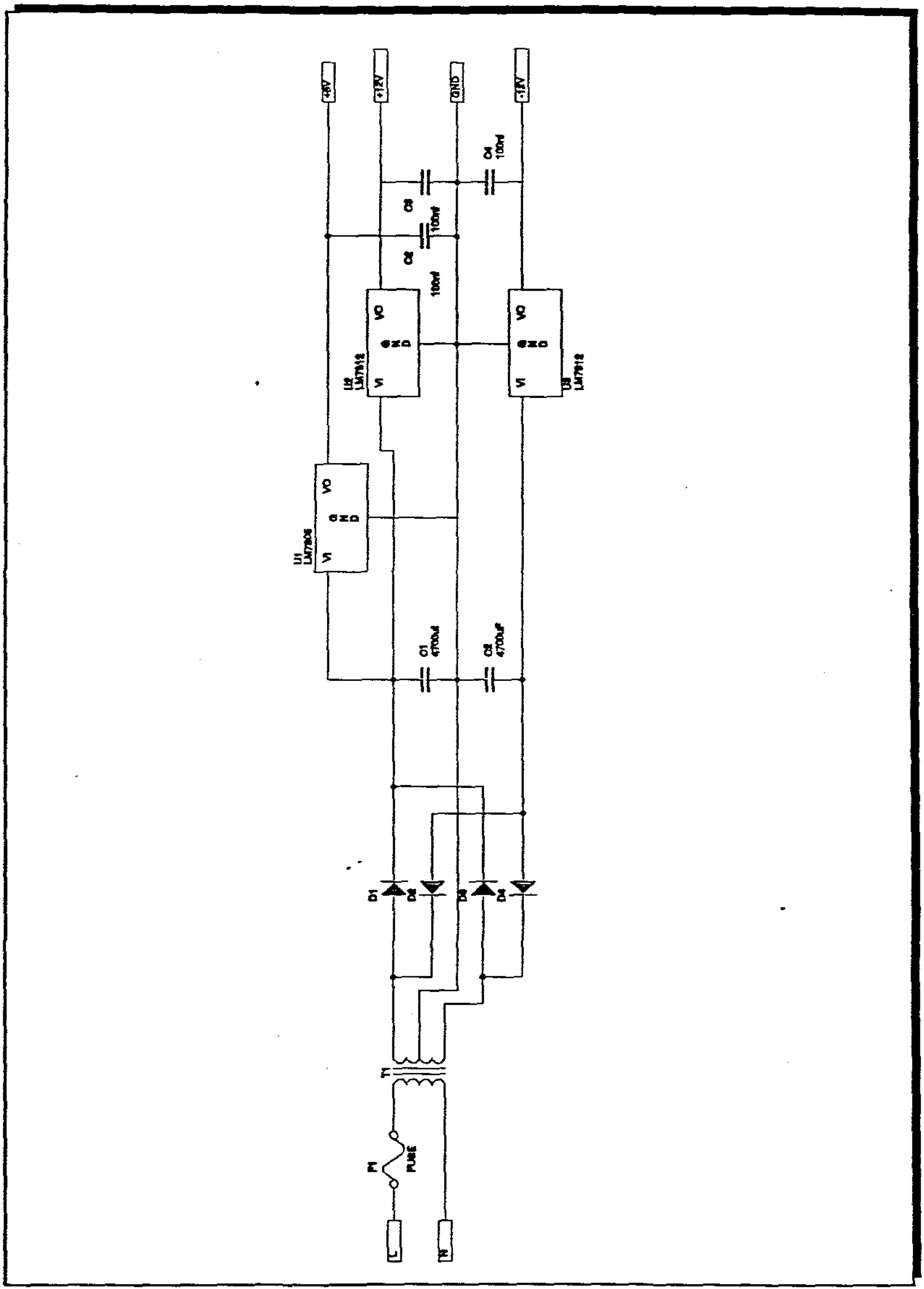
Drawing No 10 Xilinx 18-Bit Address latch sheet



Drawing No 11 Xilinx 18-Bit Address Multiplexer

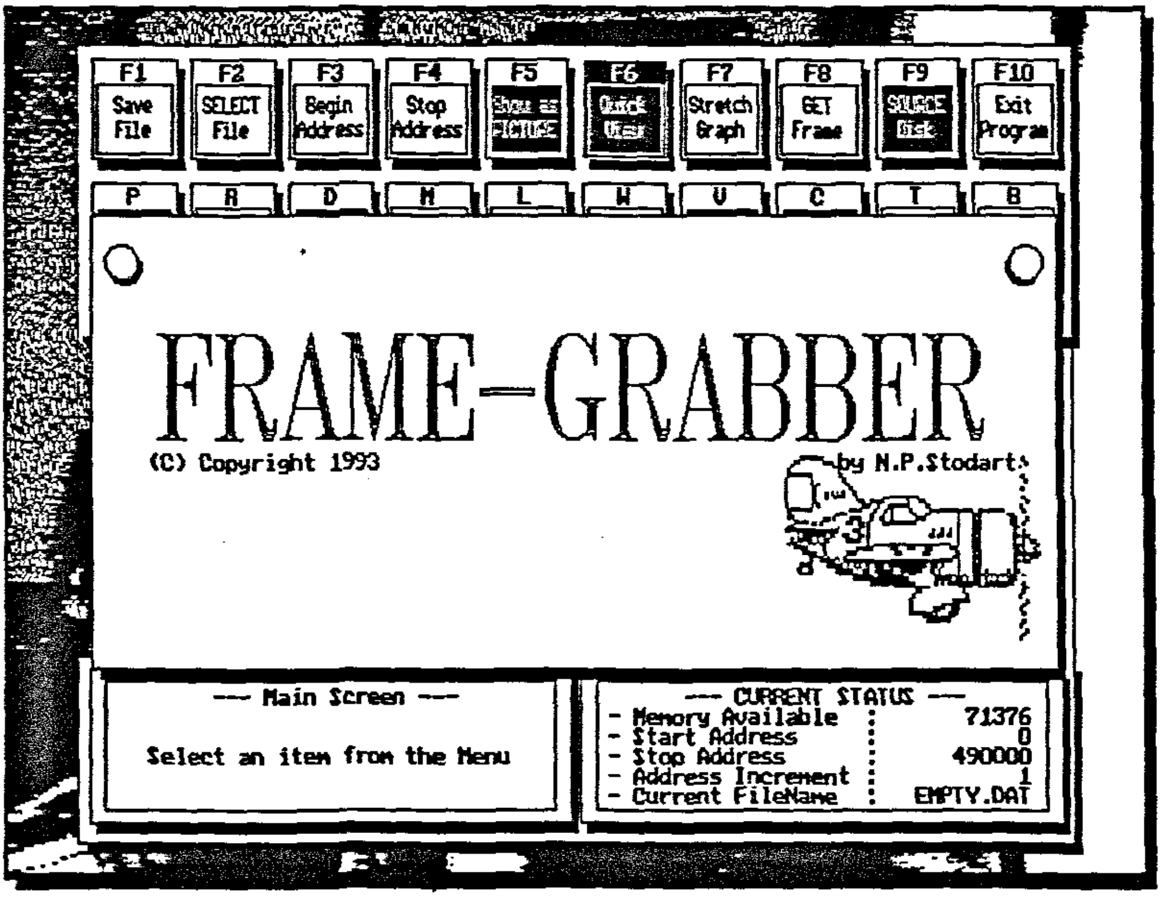


Drawing No 12 256K Static Memory sheet



Drawing No 13 Power Supply Board

Frame Grabber (FR30.PAS) Program listing



FRAME-GRABBER

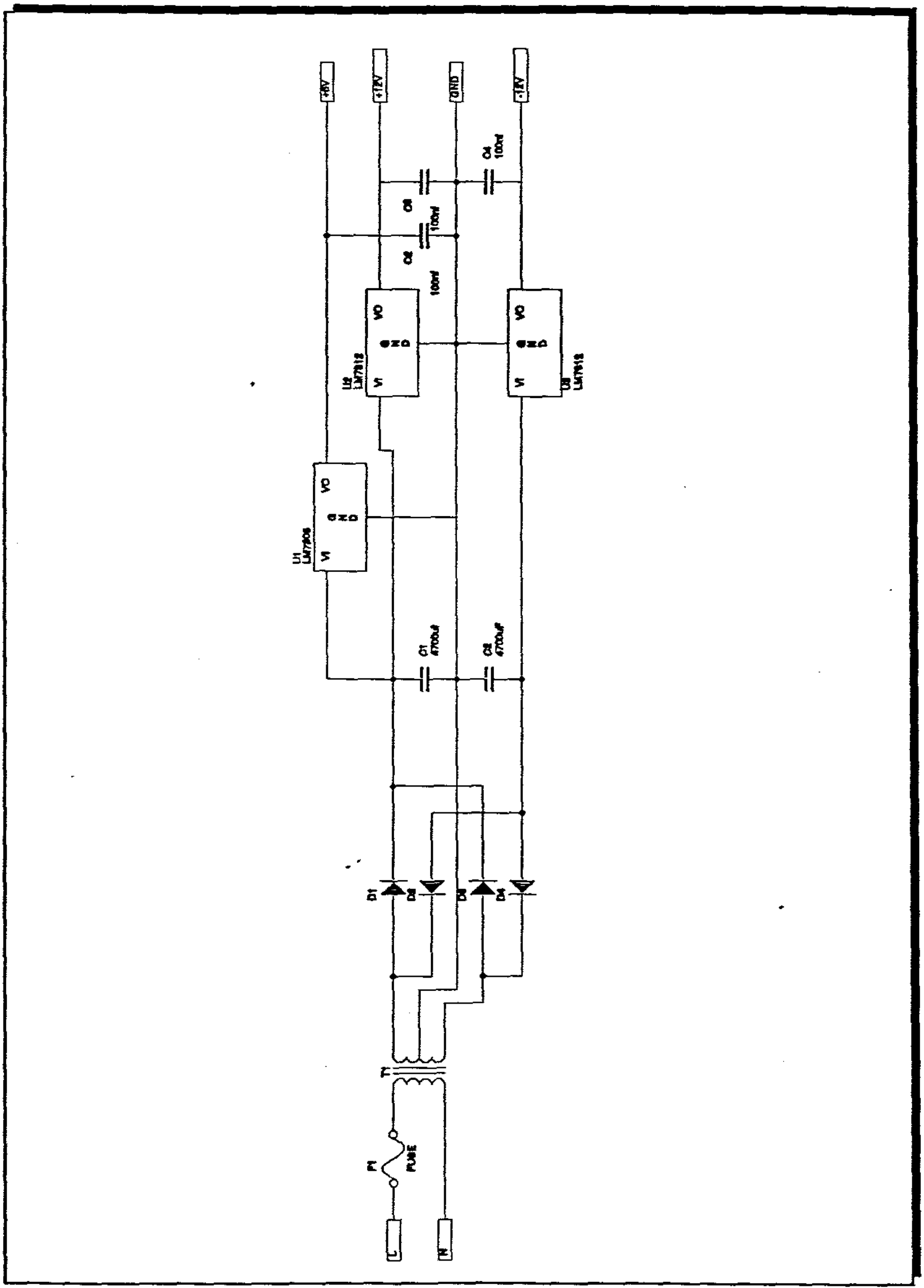
(C) Copyright 1993

by N.P.Stodart

```
--- Main Screen ---

Select an item from the Menu
```

```
--- CURRENT STATUS ---
- Memory Available  :      71376
- Start Address     :          0
- Stop Address      :     490000
- Address Increment :          1
- Current FileName  :  EMPTY.DAT
```

A full listing of the **FR30.PAS** program follows on Pages B-2. Its is inevitable that this program will be changed or added to for future developments in the frame grabber unit.



Drawing No 13 Power Supply Board

APPENDIX B

```
Program ScopeImage;

{$M 51200,0,100000}

        {     This program is used in conjunction with the
                 Frame Grabber designed by N.P.Stodart
                    It are used to get a image
          of the data stored on the screen of a computer       }

Uses Crt,Dos,Graph;

  Const
     { The screen mask is used to determain which of the curser pixels
       becomes are part of the shape (1) or of the background (0) ( it is
       ANDed with the screen contents). The Curser mask is used to determain
       which of the pixels of the cursors contribute to the color/shape of
       the curser (it is XORed with the result of the previous operation

       Screen Mask         , Curser Mask      Resulting Screen Bit
            0                    0                0
            0                    1                1
            1                    0                Unchanged
            1                    1                Inverted    }

    ScreenCurser     : Array[1..16,1..16] of byte =
                      ((0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
                       (0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1),
                       (0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1),
                       (0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1),
                       (0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1),
                       (0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1),
                       (0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1),
                       (0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1),
                       (0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
                       (0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
                       (0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1),
                       (0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,1),
                       (0,0,1,1,0,0,0,0,1,1,1,1,1,1,1,1),
                       (1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1),
                       (1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1),
                       (1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1));
    MaskCurser       : Array[1..16,1..16] of byte =
                      ((0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
                       (0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
                       (0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0),
                       (0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0),
                       (0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0),
                       (0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0),
                       (0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0),
                       (0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0),
                       (0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0),
                       (0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0),
                       (0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0),
                       (0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0),
                       (0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0),
                       (0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0),
                       (0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0),
                       (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0));

    XScreenSize          =   619;
    YScreenSize          =   319;
    DOTS                 =   613;     { Maximum Dots for graphic in
                                        the X-Direction }
    XoffSet              =   2;
    YoffSet              =   0;
    MaxBlockSize         =   16384;  {Maximum Size of Block to read}
    MaxUserMemory        =   532480; { Maximum Memory Buffer Size }
    ProgramMem           =   20000;  {Memory uses for current program}
    SpecialSet           =   [#8,#9,#13,#27]; { Special set for keyboard }


   Type
```

```
 P                  =  Pointer;
 PointArray         =  Array[1..36] of P;
 DataArray          =  Array[1..MaxBlockSize] of byte;
 Point              =  ^DataArray;
 MouseArray         =  Array[1..500] of Byte;
 VidInfo            =  Array[1..3] of integer;

Var

 VidFile             : File of VidInfo;
 VidData             : VidInfo;
 Regs                : Registers;  { Normal assmble registers used in ASM }
 CurserScreen        : MouseArray; { Pointer to Mouse Screen }
 CurserMask          : MouseArray; { Pointer to Mouse Mask }
 CurSCNImg           : MouseArray; { Image Stored at current Mouse Curser }
 OLDVECTOR           : Pointer;    { Pointer to INT 1Ch address }
 CountMouse          : Integer;    { Mouse Delay Counter }
 Xmove               : Integer;
 Ymove               : Integer;
 PrevXput            : Integer;    { Previous mouse X placement value }
 PrevYput            : Integer;    { Previous mouse Y placement value }
 CurXput             : Integer;    { Current mouse X placement value }
 CurYput             : Integer;    { Current mouse Y placement value }

 MaxMemory           : Longint;    { Maximum amount of memory free }
 MemBlock            : Longint;    { Amount of 16k blocks to be filled }
 MaxBlockRead        : Longint;    { Maximum memory for the read buffer}
 MaxBlockWrite       : Longint;    { Maximum memory for the write buffer}
 AddressPoint        : PointArray; { Array of all the starting addresses }
                                   { of D1..D36 }
 D1,D2,D3,D4,D5,D6   : Point;      { 32 Arrays of 16k each}
 D7,D8,D9,D10,D11    : Point;
 D12,D13,D14,D15,D16 : Point;
 D17,D18,D19,D20,D21 : Point;
 D22,D23,D24,D25,D26 : Point;
 D27,D28,D29,D30,D31 : Point;
 D32,D33,D34,D35,D36 : Point;

 TOP                 : Array[0..10000] of byte;
 BOT                 : Array[0..10000] of byte;
 FileInput           : String;     { FileName for to retrieve data from }
 FileOutput          : String;     { FileName to write data to }
 F                   : File;       {  FileHandler used for Input }
 PCX                 : File;       {  FileHandler used for PCX Convertion }
 FWrite              : File;       {  FileHandler used for Output to disk }
 ReadError           : Boolean;    { True if a whole file cannot be read }
 WriteError          : Boolean;    {-True when whole file cannot be written}
                                   { to disk }
 PX,PY               : Array[0..10000] of Byte;    { Points to stored image }
 ImgeSize            : Word;       { Size of Pointer P }
 Xdisplay            : Integer;    { Maximum X - Pixels on Display }
 Ydisplay            : Integer;    { Maximum Y - Pixels on Display }
 ColorsMax           : Integer;    { Maximum Colors possible }
 XWork               : Integer;    { Working Coordinates in X }
 YWork               : Integer;    { Working Coordinates in Y }
 XaxisINC            : Real;       { Increment in the Xdirection }
 GraphXCount         : Longint;    { X-Axis step-counter }
 YY                  : Char;
 KeyRead             : Char;       { Key read from keyboard buffer }

 PCXConvert          : Boolean;
 VideoModeFlag       : Boolean;
 PrevSynchTest       : Boolean;
 SynchTest           : Boolean;
 TotalVShow          : Integer;
 CheckVideo          : Boolean;    { Var for comparing }
 PrevVideoOn         : Boolean;
 VideoON             : Boolean;    { Starts of Line }
 GraphPIC            : Byte;       { Point where Video Level starts }
 SYNCHLENGTH         : Integer;    { Length of Horisontal Synch Pulse }
 SYNCHLOW            : Byte;       { Lower Point for Sync level }
 SYNCHCOUNT          : Integer;    { Counter to count Synch Length }
 HSYNCH              : Integer;    { H-Synch counter test value }
```

```
SYNCHMED            : Byte;       { Switching point for video }
SYNCHFLAG           : Boolean;    { Test flag for synch }
VIDEOFLAG           : Boolean;    { Test flag true at start of video info }
VIDEOMARKMAX        : Integer;    { MAX Value for the video Synch counter }
VIDEOMARKMIN        : Integer;    { MIN Value for the video Synch counter }
PREVSINGLE          : Byte;       { Previous value read from Device }

HorisontalDEF       : Integer;    { Length of 1 Line in pixels }
VerticalDEF         : Integer;    { Total vertical length }
FrontPorch          : Integer;    { Length of frontporch }
VideoLength         : Integer;    { Length of video signal }
VIDCount            : Integer;    { Counter to count video length }
Videolevel          : Integer;    { Point where video will start }
MaxVideoLevel       : Integer;
MinVideoLevel       : Integer;
XposPIC             : Integer;    { Current X-position while drawing PIC }
YposPIC             : Integer;    { Current Y-Position while drawing PIC }
XincPIC             : Real;       { Scaling factor to draw picture }
YincPIC             : Real;       { Scaling factor to draw picture }
InterLace           : Real;       { Scaling factor for Interlacing }
GrayScale           : Real;       { Color scaling factor }
BWLevel             : Integer;    { Point for switching in HERC mode }
ViewPICX            : Integer;    { Maximum X-Size of PIC }
ViewPICY            : Integer;    { Maximum Y-Size of PIC }
HCount              : Integer;    { Counter in X direction for PIC }
VCount              : Integer;    { Counter in INC of 2 for PIC }
VStart              : Integer;    { Starting point for PIC Display }
VcntFinal           : Integer;    { Counter in Y Direction for PIC }
EVENFLAG            : Boolean;    { True while in EVEN Field }
OLDEVENFLAG         : Boolean;    { Previous Field status }
TopCount            : Integer;    { Counter to get rid of first Video Lines }
TopStop             : Integer;    { Presetable Stop value for skip }
StartA              : Longint;    { Start address of address counters }
StopA               : Longint;    { Stop address of address counter }
Increment           : Integer;    { Increment between address }


SpecialKey          : Boolean;  { Flag for if Second ReadKey Needed }
StringKey           : String[10];

OnetoOne            : Boolean;    { Flag set for no Scale}
DiskFlag            : Boolean;    { Flag for DISK/Grabber Selection }
ColorFlag           : Boolean;    { Flag for BW / GrayScale }
DotFlag             : Boolean;    { Select Between Line / dot draw }
INVFLAG             : Boolean;    { Invert picture in B/W Mode }
GraphFlag           : Boolean;    { Select between Graph and Picture Display }
FileReadFlag        : Boolean;    { Check if file is OPEN }
DirecttoScreen      : Boolean;    { From Grabber to Screen }
DirecttoDisk        : Boolean;    { From Grabber to Disk }
HaltDraw            : Boolean;    { SuddenStop while drawing }
StretchFlag         : Boolean;    { Flag to strecth data }
StretchCount        : Integer;    { Count mouse keypresses }
QuickViewFlag       : Boolean;    { Flag used for QuickView of Graph }
PaletCounter        : Integer;    { 0..63 for 64 GrayScales }
MaxColor            : Integer;    { Colors Available }
Palet               : Array[0..255,0..2] of Byte;
StartAVar           : Longint;
StopAVar            : Longint;
MousePixel          : Integer;
MouseMovFlag        : Boolean;
MouseKeyPressed     : Boolean;
HideCurser          : Boolean;
CurserSkip          : Integer;
HeapPointer         : Pointer;
F1Mouse             : Array[0..1] of Integer;
F2Mouse             : Array[0..1] of Integer;
F3Mouse             : Array[0..1] of Integer;
F4Mouse             : Array[0..1] of Integer;
F5Mouse             : Array[0..1] of Integer;
F6Mouse             : Array[0..1] of Integer;
F7Mouse             : Array[0..1] of Integer;
F8Mouse             : Array[0..1] of Integer;
F9Mouse             : Array[0..1] of Integer;
```

```
    F10Mouse          : Array[0..1] of Integer;
    YBlock1           : Array[0..1] of Integer;

    PMouse           : Array[0..1] of Integer;
    RMouse           : Array[0..1] of Integer;
    DMouse           : Array[0..1] of Integer;
    MMouse           : Array[0..1] of Integer;
    LMouse           : Array[0..1] of Integer;
    WMouse           : Array[0..1] of Integer;
    VMouse           : Array[0..1] of Integer;
    CMouse           : Array[0..1] of Integer;
    TMouse           : Array[0..1] of Integer;
    BMouse           : Array[0..1] of Integer;
    YBlock2          : Array[0..1] of Integer;
    YBlock3          : Array[0..1] of Integer;
    PCXArray         : Array[1..801] of Byte;


{$i svga16.inc}
{$i svga256.inc}
{$i svga_put.inc}

(* Uncomment this if you link in the drivers *)
(*
procedure Svga16_driver; external;
{ $L svga16.obj }

procedure Svga256_driver; external;
{ $L svga256.obj }

procedure Twk256_driver; external;
{ $L twk256.obj }

procedure Twk16_driver; external;
{ $L twk16.obj }

procedure Svga32k_driver; external;
{ $L svga32k.obj }

procedure SvgaS3_driver; external;
{ $L svgas3.obj }
*)

var
  GraphMode, GraphDriver : integer;
  Ky : Char;
  Drv : Integer;

function WhitePixel : Word;
begin
  if (GetMaxColor > 256) then
    WhitePixel := 32767
  else
    WhitePixel := 15;
end;

{$F+}
function DetectVGA256 : Integer;
var Vid : Integer;

begin
  Vid:=VidData[3];
  DetectVGA256 := Vid;
end;

function DetectS3 : Integer;
var Vid : Integer;

begin
  Vid:=VidData[3];
  DetectS3 := Vid;
end;
```

```
function DetectVGA32k : Integer;
var Vid : Integer;

begin
  Vid:=VidData[3];
  DetectVGA32k := Vid;
end;


function DetectVGA16 : Integer;
var Vid : Integer;

begin
   Vid:=VidData[3];
   DetectVGA16 := Vid;
end;

function DetectTwk256 : Integer;
var Vid : Integer;

begin
  Vid:=VidData[3];
  DetectTwk256 := Vid;
end;

function DetectTwk16 : Integer;
var Vid : Integer;

begin
  Vid:=VidData[3];
  DetectTwk16 := Vid;
end;
{$F-}

  Procedure SetPalet;
   Begin
    For PaletCounter:=0 to 255 do
      Begin
       Palet[PaletCounter,0]:=PaletCounter;
       Palet[PaletCounter,1]:=PaletCounter;
       Palet[PaletCounter,2]:=PaletCounter;
      end;

       Regs.ah:=$10;
       Regs.al:=$12;
       Regs.BX:=0;
       Regs.ES:=Seg(Palet);
       Regs.DX:=OFS(Palet);
       Regs.CX:=256;
       Intr($10,Regs);
     end;


  Procedure SETUPGRAPHICS; { This procedure chooses the best graphicsmode available }
                           { with the current graphics card fitted to the machine.  }
  Var DetDriver            : Integer;
      DetMode              : Integer;
      GraphicDriver        : Integer;
      GraphicMode          : Integer;

 begin

   DetDriver:=Detect;
   DetectGraph(DetDriver,DetMode);
   GraphicDriver:=VidData[1];

   If (DetDriver = 9) AND (GraphicDriver = 5)  then
    Begin
     Drv:=VidData[2];
      if (Drv = 0) then
     GraphDriver := InstallUserDriver('SVGA256',@DetectVGA256)
    else if (Drv = 1) then
     GraphDriver := InstallUserDriver('SVGA16',@DetectVGA16)
  else if (Drv = 2) then
   GraphDriver := InstallUserDriver('Twk256',@DetectTwk256)
  else if (Drv = 3) then
   GraphDriver := InstallUserDriver('Twk16',@DetectTwk16)
 else if (Drv = 4) then
   GraphDriver := InstallUserDriver('Svga32k',@DetectVGA32k)
 else if (Drv = 5) then
   GraphDriver := InstallUserDriver('SvgaS3',@DetectS3);
 GraphDriver := Detect;
 InitGraph(GraphDriver,GraphMode,'');
 If GetMaxColor > 16 then  SetPalet;
 end
else
 begin
  Case GraphicDriver of
   1     :Begin DetDriver:=Detect end;
   2     :Begin DetDriver:=Detect end;
   3     :Begin DetDriver:=1; DetMode:=0; end;
   4     :Begin DetDriver:=Detect end;
 end;
  InitGraph(DetDriver,DetMode,'');
 end;
 SetFillStyle(1,MaxColor);
 Bar(0,0,50,50);
 END;


 Procedure FunctionScreen(BlockSelectTOP,BlockSelectBOT:Integer; BlockInvert:Integer);

 Var

  BigBlockSize                 : Integer;   { Size of OuterBlock of Function Select }
  SmallBlockSize               : Integer;   { Size of innerBlock of Function Select }
  BigBlocks                    : Integer;   { Number of OuterBlocks in a Line }
  SmallBlocks                  : Integer;   { Number of InnerBlocks in a Line }
  BlockCounter                 : Integer;   { Counter to count total Blocks used}
  BigBlockWork                 : Integer;   { Variable used in calculations }
  SmallBlockWork               : Integer;   { Variable used in calculations (with smallblocks) }
  TextWork                     : Integer;   { Variable to get text spacing }
  BigBlockINC                  : Real;      { Increment of Big blocks }
  SmallBlockINC                : Real;      { Increment of SmallBlocks }
  ShadowValue                  : Integer;   { Size of the shadow needed }
  GraphString1                 : String;    { Line 1 in function block }
  GraphString2                 : String;    { Line 2 in function block }
  GraphString3                 : String;    { Line 3 in function block }
  StartBlockCnt                : Integer;   { Start for block counter }
  StopBlockCnt                 : Integer;   { Stop of the block counter }
  BlockPicSize                 : Word;      { Size of screenblock }
  BlockPIC                     : Array[1..20000] of byte;   { Pointer to stored select Window }
  BlockInv                     : Boolean;
  SingleBlock                  : Boolean;   { Hilights a single block }

  Begin
   Setcolor(0);
   BigBlockSize:=52;        { Define Size of the outer block }
   BigBlocks:=10;           { Define the number of outer blocks }
   SmallBlockSize:=46;      { Define the size of a inner block }
   SmallBlocks:=10;         { Define the number of inner blocks }
   ShadowValue:=4;          { Set the value of the shadow as required }

   { Calculate the Increment rate of the outerblocks to place  "Bigblocks"
     of outer blocks evenly spread over the whole screen.          }
   BigBlockINC:=((XscreenSize+1)-(BigBlockSize*BigBlocks))/(BigBlocks+1);
   { Calcutation of the increment rate of the Inner blocks to place "SmallBlocks"
     of inner blocks }
   SmallBlockINC:=(BigBlockSize-SmallBlockSize)/2;
   YBlock1[0]:=15;
   YBlock1[1]:=43;

  IF BlockSelectTOP <= BigBlocks then
   Begin
   If BlockSelectTOP <> 0 then
    Begin
```

```
          StartBlockCnt:=BlockSelectTOP-1;
          StopBlockCnt:=BlockSelectTOP-1;
          SingleBlock:=True;
          BlockInv:=False;
          end
        else
         Begin
          StartBlockCnt:=0;
          StopBlockCnt:= BigBlocks-1;
          SingleBlock:=False;
          BlockInv:=False;
         end;


        For BlockCounter:= StartBlockCnt to StopBlockCnt do  { Place the first row of funtion blocks }
         Begin
          BlockInv:=False;
          BigBlockWork:=(BlockCounter*BigBlockSize)+Round((BlockCounter+1)*BigBlockInc);
          SmallBlockWork:=Round(SmallBlockINC);
          SmallBlockWork:=BigBlockWork+SmallBlockWork;
          SetFillStyle(0,0);              { Draw Shadow }

Bar((BigBlockWork+ShadowValue),(5+ShadowValue),(BigBlockwork+BigBlockSize+ShadowValue),(45+ShadowValue));
          SetFillStyle(1,MaxColor);     { Draw outer blocks }
          Bar(BigBlockWork,5,(BigBlockwork+BigBlockSize),45);
          Rectangle(BigBlockWork,5,(BigBlockwork+BigBlockSize),45);
                                        { Draw inner block }
          Rectangle(SmallBlockWork,15,(SmallBlockwork+SmallBlockSize),43);

          Case BlockCounter of       { Set the text for each block as needed }
          0: Begin F1Mouse[0]:=SmallBlockWork;
                   F1Mouse[1]:=SmallBlockWork+SmallBlockSize;
                   GraphString1:='F1';
                   GraphString2:='Save';  { Write a file to disk The length of the }
                   GraphString3:='File';  { depends on the current settings of  }
             end;                         { Start and Stop address }
          1: Begin F2Mouse[0]:=SmallBlockWork;
                   F2Mouse[1]:=SmallBlockWork+SmallBlockSize;
                   GraphString1:='F2';
                   GraphString2:='SELECT'; { Select file to retrieve data from }
                   GraphString3:='File';     { It is possible to select only a part }
             end;                            { of the file by entering new start & stop
                                             { Values for the address }
          2: Begin F3Mouse[0]:=SmallBlockWork;
                   F3Mouse[1]:=SmallBlockWork+SmallBlockSize;
                   GraphString1:='F3';
                   GraphString2:='Begin';    { Select new starting address }
                   GraphString3:='Address';
             end;
          3: Begin F4Mouse[0]:=SmallBlockWork;
                   F4Mouse[1]:=SmallBlockWork+SmallBlockSize;
                   GraphString1:='F4';
                   GraphString2:='Stop';     { Select new Stop address }
                   GraphString3:='Address';
             end;
          4: Begin F5Mouse[0]:=SmallBlockWork;
                   F5Mouse[1]:=SmallBlockWork+SmallBlockSize;
                   GraphString1:='F5';       { Show picture as captured source can be }
                   GraphString2:='Show as';     { from DISK or from the FRAME-GRABBER }
                   GraphString3:='PICTURE';  { Picture Display }
                   BlockINV:=True;
                   IF GraphFlag then GraphString3:='GRAPH';  { Graph -display }
             end;
          5: Begin F6Mouse[0]:=SmallBlockWork;
                   F6Mouse[1]:=SmallBlockWork+SmallBlockSize;
                   GraphString1:='F6';       { Quick Zoom In from the current curser position }
                   GraphString2:='Quick';    { Will show expanded part in the bottom window }
                   GraphString3:='View';
             end;
          6: Begin F7Mouse[0]:=SmallBlockWork;
                   F7Mouse[1]:=SmallBlockWork+SmallBlockSize;
                   GraphString1:='F7';       { Selects the part to be stretched }
                   GraphString2:='Stretch';
```

```
           GraphString3:='Graph';
     end;
  7: Begin F8Mouse[0]:=SmallBlockWork;
           F8Mouse[1]:=SmallBlockWork+SmallBlockSize;
           GraphString1:='F8';        { Get a frame as captured by the frame-grabber }
           GraphString2:='GET';
           GraphString3:='Frame';
     end;
  8: Begin F9Mouse[0]:=SmallBlockWork;
           F9Mouse[1]:=SmallBlockWork+SmallBlockSize;
           GraphString1:='F9';        { Select between Data from DISK or Direct from the }
           GraphString2:='SOURCE';      { Frame-Grabber }
           GraphString3:='Grabber';
           BlockINV:=True;
           IF DISKFLAG then GraphString3:='Disk';
     end;
  9: Begin F10Mouse[0]:=SmallBlockWork;
           F10Mouse[1]:=SmallBlockWork+SmallBlockSize;
           GraphString1:='F10';       { Quit this Program and return to DOS }
           GraphString2:='Exit';
           GraphString3:='Program';
     end
     end;  { end of case statement }

   SetTextStyle(DefaultFont,HorizDir,1);
   TextWork:=(BigBlockSize-TextWidth(GraphString1)) DIV 2;
   OUTTEXTXY((TextWork+BigBlockWork),7,GraphString1);  { Place Line 1 }
   SetTextStyle(SmallFont,HorizDir,4);
   TextWork:=(BigBlockSize-TextWidth(GraphString2)) DIV 2;
   OUTTEXTXY((TextWork+BigBlockWork),17,GraphString2); { Place Line 2 }
   TextWork:=(BigBlockSize-TextWidth(GraphString3)) DIV 2;
   OUTTEXTXY((TextWork+BigBlockWork),27,GraphString3); { Place Line 3 }
   BlockPICSize:=ImageSize(SmallBlockWork,65,(SmallBlockwork+SmallBlockSize),93);
  { GetMem(BlockPic,BlockPicSize);}
   GetImage(SmallBlockWork,15,(SmallBlockwork+SmallBlockSize),43,BlockPic);
   IF BlockInv then
    Begin
     PutImage(SmallBlockWork,15,BlockPic,NotPut);
    end;
   {Release(BlockPic);}
   BlockInv:=False;

  IF SingleBlock AND (BlockInvert=0) then
   Begin
    BlockPicSize:=ImageSize(BigBlockWork,5,(BigBlockwork+BigBlockSize),45);
  {  GetMem(BlockPic,BlockPicSize);}
    GetImage(BigBlockWork,5,(BigBléckwork+BigBlockSize),45,BlockPic);
    PutImage(BigBlockWork,5,BlockPic,NotPut);
    Delay(2000);
    PutImage(BigBlockWork,5,BlockPic,NormalPut);
    SingleBlock:=False;
    {Release(BlockPic);}
   end;
  IF BlockInvert=1 Then
   Begin
    BlockPicSize:=ImageSize(BigBlockWork,5,(BigBlockwork+BigBlockSize),45);
  {  GetMem(BlockPic,BlockPicSize);}
    GetImage(BigBlockWork,5,(BigBlockwork+BigBlockSize),45,BlockPic);
    PutImage(BigBlockWork,5,BlockPic,NotPut);
    {Release(BlockPic);}
  end;
 end;
end;

BigBlockSize:=52;      { Define the dimensions of the second function line }
BigBlocks:=10;
SmallBlockSize:=46;
SmallBlocks:=10;
ShadowValue:=4;
BigBlockINC:=((XscreenSize+1)-(BigBlockSize*BigBlocks))/(BigBlocks+1);
SmallBlockINC:=(BigBlockSize-SmallBlockSize)/2;
YBlock2[0]:=65;
YBlock2[1]:=93;
```

```
    IF BlockSelectBOT <= BigBlocks then
    Begin
     If BlockSelectBOT <> 0 then
      Begin
       StartBlockCnt:=BlockSelectBOT-1;
       StopBlockCnt:=BlockSelectBOT-1;
       SingleBlock:=True;
       end
     else
      Begin
       StartBlockCnt:=0;
       SingleBlock:=False;
       StopBlockCnt:= BigBlocks-1;
      end;

     For BlockCounter:= StartBlockCnt to StopBlockCnt do
      Begin
       BlockInv:=False;
       BigBlockWork:=(BlockCounter*BigBlockSize)+Round((BlockCounter+1)*BigBlockInc);
       SmallBlockWork:=Round(SmallBlockINC);
       SmallBlockWork:=BigBlockWork+SmallBlockWork;
       SetFillStyle(0,0);

Bar((BigBlockWork+ShadowValue),(55+ShadowValue),(BigBlockwork+BigBlockSize+ShadowValue),(95+ShadowValue));
       SetFillStyle(1,MaxColor);
       Bar(BigBlockWork,55,(BigBlockwork+BigBlockSize),95);
       Rectangle(BigBlockWork,55,(BigBlockwork+BigBlockSize),95);
       Rectangle(SmallBlockWork,65,(SmallBlockwork+SmallBlockSize),93);

       Case BlockCounter of
       0: Begin PMouse[0]:=SmallBlockWork;
                PMouse[1]:=SmallBlockWork+SmallBlockSize;
                GraphString1:='P';     { Increment Value This Value are only changeble }
                GraphString2:='Screen';   { while Reading Direct from the FRAME-Grabber }
                GraphString3:='Auto';
                BlockINV:=True;
                If OnetoOne Then GraphString3:='1:1'
          end;
       1: Begin RMouse[0]:=SmallBlockWork;
                RMouse[1]:=SmallBlockWork+SmallBlockSize;
                GraphString1:='R';      { Select reset values for all the Variables }
                GraphString2:='Reset';
                GraphString3:='Values';
          end;
       2: Begin DMouse[0]:=SmallBlockWork;
                DMouse[1]:=SmallBlockWork+SmallBlockSize;
                GraphString1:='D';      { Select the Point for BLACK/WHITE Video Toggle }
                GraphString2:='Plot as';
                GraphString3:='LINE';
                BlockINV:=True;
                IF Dotflag then GraphString3:='DOT';
          end;
       3: Begin MMouse[0]:=SmallBlockWork;
                MMouse[1]:=SmallBlockWork+SmallBlockSize;
                GraphString1:='M';      { Calculate the Minimum and Maximum Values }
                GraphString2:='Video';
                GraphString3:='Mode';
          end;
       4: Begin LMouse[0]:=SmallBlockWork;
                LMouse[1]:=SmallBlockWork+SmallBlockSize;
                GraphString1:='L';      { Selects Line to start Show as Graph }
                GraphString2:='Show';
                GraphString3:='LINE n';
          end;
       5: Begin WMouse[0]:=SmallBlockWork;
                WMouse[1]:=SmallBlockWork+SmallBlockSize;
                GraphString1:='W';      { Selects between a line or a Dot plot }
                GraphString2:='White';
                GraphString3:='Level';
          end;
       6: Begin VMouse[0]:=SmallBlockWork;
                VMouse[1]:=SmallBlockWork+SmallBlockSize;
```

```
            GraphString1:='V';     { Invert the video information }
            GraphString2:='Video';
            GraphString3:='Normal';
            BlockINV:=True;
            If INVFLAG then GraphString3:='INVERT';
        end;
     7: Begin CMouse[0]:=SmallBlockWork;
            CMouse[1]:=SmallBlockWork+SmallBlockSize;
            GraphString1:='C';          { Select between B/W or GrayScale for the output }
            GraphString2:='Color';      { of the picture to the screen , Gray Scale are only }
            GraphString3:='B/W';        { possible on a VGA-Monitor }
            BlockINV:=True;
            If Colorflag then GraphString3:='GRAY-S';
        end;
     8: Begin TMouse[0]:=SmallBlockWork;
            TMouse[1]:=SmallBlockWork+SmallBlockSize;
            GraphString1:='T';          { Selects toggle Point for B/W }
            GraphString2:='B/W';   { video this serves as the contrast control }
            GraphString3:='Contrast';
        end;
     9: Begin BMouse[0]:=SmallBlockWork;
            BMouse[1]:=SmallBlockWork+SmallBlockSize;
            GraphString1:='B';          { Selects only ODD or EVEN field to Display }
            GraphString2:='Black';      { Selects TOP Lines to SKIP }
            GraphString3:='Level';    { Place for Further expansion }
        end
        end;  { end of case statement }

      { Select the line of text to display }
      SetTextStyle(DefaultFont,HorizDir,1);
      TextWork:=(BigBlockSize-TextWidth(GraphString1)) DIV 2;
      OUTTEXTXY((TextWork+BigBlockWork),57,GraphString1);
      SetTextStyle(SmallFont,HorizDir,4);
      TextWork:=(BigBlockSize-TextWidth(GraphString2)) DIV 2;
      OUTTEXTXY((TextWork+BigBlockWork),67,GraphString2);
      TextWork:=(BigBlockSize-TextWidth(GraphString3)) DIV 2;
      OUTTEXTXY((TextWork+BigBlockWork),77,GraphString3);
      BlockPICSize:=ImageSize(SmallBlockWork,65,(SmallBlockwork+SmallBlockSize),93);
{       GetMem(BlockPic,BlockPicSize); }
      GetImage(SmallBlockWork,65,(SmallBlockwork+SmallBlockSize),93,BlockPic);
      IF BlockInv then
       Begin
        PutImage(SmallBlockWork,65,BlockPic,NotPut);
       end;
      {Release(BlockPic);}
      BlockInv:=False;

     IF SingleBlock AND (BlockInvert=0) then
      Begin
       BlockPicSize:=ImageSize(BigBlockWork,55,(BigBlockwork+BigBlockSize),95);
     {  GetMem(BlockPic,BlockPicSize);}
       GetImage(BigBlockWork,55,(BigBlockwork+BigBlockSize),95,BlockPic);
       PutImage(BigBlockWork,55,BlockPic,NotPut);
       Delay(2000);
       PutImage(BigBlockWork,55,BlockPic,NormalPut);
       SingleBlock:=False;
       {Release(BlockPic);}
      end;

    IF BlockInvert=1 Then
      Begin
       BlockPicSize:=ImageSize(BigBlockWork,55,(BigBlockwork+BigBlockSize),95);
     {  GetMem(BlockPic,BlockPicSize);}
       GetImage(BigBlockWork,55,(BigBlockwork+BigBlockSize),95,BlockPic);
       PutImage(BigBlockWork,55,BlockPic,NotPut);
       {elease(BlockPic);}
     end;
    end;
   end;
  end;

 Procedure ScreenSave(ScreenFile:String;StartingY,StoppingY:Integer);
  Var
```

```
     IMAGE              : Array[1..10000] of Byte;
     LineConstant       : Integer;
     LineCount          : Integer;
     MemSize            : Word;
     Screen             : File;
     NumWrite           : Word;
     LineStop           : Integer;
     ViewPort           : ViewPortType;

     Begin
      GetViewSettings(ViewPort);
      IF StoppingY >= GetMaxY then
      SetViewPort(0,0,GetMaxX,GetMaxY,True);
      Assign(Screen,ScreenFile);
      Rewrite(Screen,1);
      LineCount:=StartingY;
      LineConstant:=10;
      LineCount:=StartingY-LineConstant;
      Repeat
       LineCount:=LineCount+LineConstant;
       LineStop:=LineCount+LineConstant;
       If LineStop >= StoppingY Then LineStop:= StoppingY+1;
       MemSize:=ImageSize(0,LineCount,GetMaxX,LineStop);
       GetImage(0,LineCount,GetMaxX,LineStop,IMAGE);
       BlockWrite(Screen,IMAGE,MemSize,NumWrite);
      until LineStop > StoppingY;
      Close(Screen);
      SetViewPort(ViewPort.X1,ViewPort.Y1,ViewPort.X2,ViewPort.Y2,ViewPort.Clip);
     end;


   Procedure ScreenReread(ScreenFile:String; StartingY,StoppingY:Integer);
    Var
     IMAGE              : Array[1..10000] of Byte;
     LineConstant       : Integer;
     LineCount          : Integer;
     LineStop           : Integer;
     MemSize            : Word;
     Screen             : File;
     NumRead            : Word;
     ViewPort           : ViewPortType;

     Begin
      GetViewSettings(ViewPort);
      IF StoppingY >= GetMaxY then
      SetViewPort(0,0,GetMaxX,GetMaxY,True);
      Assign(Screen,ScreenFile);
      Reset(Screen,1);
      LineCount:=StartingY;
      LineConstant:=10;
      LineStop:=LineCount+LineConstant;
      LineCount:=LineCount-LineConstant;
      Repeat
       LineCount:=LineCount+LineConstant;
       LineStop:=LineCount+LineConstant;
       If LineStop >= StoppingY Then LineStop:= StoppingY+1;
       MemSize:=ImageSize(0,LineCount,GetMaxX,LineStop);
       BlockRead(Screen,IMAGE,MemSize,NumRead);
       IF (INVFlag AND NOT(GraphFlag)) AND (QUICKVIEWFLAG) then
        Begin
         PutImage(0,LineCount,IMAGE,NotPut);
        end
       Else
        Begin
         PutImage(0,LineCount,IMAGE,NormalPut);
        end;
      until LineStop > StoppingY;
      Close(Screen);
      SetViewPort(ViewPort.X1,ViewPort.Y1,ViewPort.X2,ViewPort.Y2,ViewPort.Clip);
{      Release(IMAGE); }
     end;
   Procedure ScreenSetup;
    Var
     ShadowMain          : Integer;

   { This Procedure draws the main screen for the frame-grab program }
    Begin
     ShadowMain:=5;
     SetColor(MaxColor);
     SetFillStyle(9,MaxColor);
     Bar(0,0,GetMaxX,GetMaxY);
     SetLineStyle(Solidln,0,3); { Sets line thickness to 1, color = 0 }
     Rectangle(0,0,(GetMaxX),(GetmaxY));
     SetColor(0);
     SetLineStyle(Solidln,0,1); { Sets line thickness to 1, color = 0 }
     Rectangle(2,2,(GetMaxX-2),(GetmaxY-2));
     SetColor(MaxColor);
    end;

   Procedure InnerDisplay;
   Var
    ShadowMain          : Integer;
    Top                 : Pointer;
    Bot                 : Pointer;
    MemSize             : Word;

    Begin
     ShadowMain:=5;
     Xwork:=((GetmaxX +1) div 2)  - (XScreenSize div 2 );
     Ywork:=((GetmaxY +1) div 2)  - (YScreenSize div 2 );
     Xdisplay:=XscreenSize;
     YDisplay:=YScreenSize;


     SetViewPort(0,0,GetMaxX,GetMaxY,True);
     SetFillStyle(0,0);

Bar((Xwork+ShadowMain),(Ywork+ShadowMain),(Xwork+XScreenSize+ShadowMain),(YWork+(YScreenSize-203)+Shadow
Main));

Bar((Xwork+ShadowMain),(Ywork+Ydisplay-73+ShadowMain),(Xwork+XScreenSize+ShadowMain),(YWork+YScreenSize+
ShadowMain));
     SetFillStyle(1,MaxColor);
     SetColor(0);
     SetLineStyle(Solidln,0,1); { Sets line thickness to 1, color = 0 }
     Rectangle((Xwork-1),(Ywork-1),(Xwork+XScreenSize+1),(YWork+YScreenSize+1));
     Setcolor(MaxColor);
     Bar(Xwork,Ywork,(Xwork+XScreenSize),(YWork+YScreenSize-203));
     Bar(Xwork,(Ywork+Ydisplay-73),(Xwork+XScreenSize),YWork+YScreenSize);


     SetColor(0);
     SetViewPort(Xwork,Ywork,(Xwork+XScreenSize),YWork+YScreenSize,True);
     Xdisplay:=XscreenSize;
     Ydisplay:=YscreenSize;
     Yblock3[0]:=Ydisplay-204;
     Yblock3[1]:=Ydisplay-75;


   END;  { Ends the procedure ScreenSetup }


  Procedure GETGRAPHSPEC;  { Get the specifications of current video mmode }

   BEGIN
    XDisplay := GetmaxX+1;  { Get maximum x - pixels for current mode }
    YDisplay := GetmaxY+1;  { Get maximum y - pixels for current mode }
   END;

   Function NumStr(i:longint):String; { Convert a number to a string variable }

    Var
     s : String[11];
   Begin
    Str(i,s);   { this function convert a integer type variable to a string for }
    NumStr:=s;  { for easiers handling while in the graphics modes }
  end;



Procedure GraphScreenSetup;
{ This procedure outlines of the frame for the Graphical repesentation }
{ of the video signal }
   Var Count               : Integer;

   Begin
    SetColor(0);
    SetFillStyle(1,0);   { Upper Block for Main-Graph Display }
    Bar(0,(Ydisplay-74),Xdisplay,(Ydisplay-202));
    Setcolor(MaxColor);
    Rectangle(0,(Ydisplay-74),Xdisplay,(Ydisplay-202));
    Setcolor(0);
    Rectangle(1,(Ydisplay-73),(Xdisplay-1),(Ydisplay-203));
    Yblock3[0]:=Ydisplay-204;
    Yblock3[1]:=Ydisplay-75;
  END;  { GraphSreenSetup }

  Procedure DescriptionBar(GetNew:Boolean);

  { Draw the desciption bar at the bottom }

  Var
   BigBlockSize                  : Integer;   { Size of OuterBlock of Function Select }
   SmallBlockSize                : Integer;   { Size of innerBlock of Function Select }
   BigBlocks                     : Integer;   { Number of OuterBlocks in a Line }
   SmallBlocks                   : Integer;   { Number of InnerBlocks in a Line }
   BlockCounter                  : Integer;   { Counter to count total Blocks used}
   BigBlockWork                  : Integer;   { Variable used in calculations }
   SmallBlockWork                : Integer;   { Variable used in calculations (with smallblocks) }
   TextWork                      : Integer;   { Variable to get text spacing }
   BigBlockINC                   : Real;      { Increment of Big blocks }
   SmallBlockINC                 : Real;      { Increment of SmallBlocks }
   ShadowValue                   : Integer;   { Size of the shadow needed }
   GraphString1                  : String;    { Line 1 in function block }
   GraphString2                  : String;    { Line 2 in function block }
   GraphString3                  : String;    { Line 3 in function block }
   StartBlockCnt                 : Integer;   { Start for block counter }
   StopBlockCnt                  : Integer;   { Stop of the block counter }
   TopY                          : Integer;   { Top left corner Position }
   BotY                          : Integer;   { Bottom right corner }

   Begin
    Setcolor(0);
    BigBlockSize:=300;       { Define Size of the outer block }
    BigBlocks:=2;          { Define the number of outer blocks }
    SmallBlockSize:=280;     { Define the size of a inner block }
    SmallBlocks:=2;        { Define the number of inner blocks }
    ShadowValue:=4;         { Set the value of the shadow as required }
    TopY:=250;
    BotY:=312;

    { Calculate the Increment rate of the outerblocks to place  "Bigblocks"
      of outer blocks evenly spread over the whole screen.          }
    BigBlockINC:=((XscreenSize+1)-(BigBlockSize*BigBlocks))/(BigBlocks+1);
    { Calcutation of the increment rate of the Inner blocks to place "SmallBlocks"
      of inner blocks }
    SmallBlockINC:=(BigBlockSize-SmallBlockSize)/2;

      StartBlockCnt:=0;
      StopBlockCnt:= BigBlocks-1;


    For BlockCounter:= StartBlockCnt to StopBlockCnt do  { Place the first row of funtion blocks }
     Begin
      BigBlockWork:=(BlockCounter*BigBlockSize)+Round((BlockCounter+1)*BigBlockInc);
```

```
       SmallBlockWork:=Round(SmallBlockINC);
       SmallBlockWork:=BigBlockWork+SmallBlockWork;
       SetFillStyle(0,0);             { Draw Shadow }

Bar((BigBlockWork+ShadowValue),(TopY+ShadowValue),(BigBlockwork+BigBlockSize+ShadowValue),(BotY+ShadowVa
lue));
       SetFillStyle(1,MaxColor);    { Draw outer blocks }
       Bar(BigBlockWork,TopY,(BigBlockwork+BigBlockSize),BotY);
       Rectangle(BigBlockWork,TopY,(BigBlockwork+BigBlockSize),BotY);
                                      { Draw inner block }
       Rectangle(SmallBlockWork,(TopY+5),(SmallBlockwork+SmallBlockSize),(BotY-5));
      end;
      IF GetNew then
       Begin
        ImgeSize:=Imagesize(SmallBlockWork,(TopY+5),(SmallBlockwork+SmallBlockSize),(BotY-5));
      { GetMem(PX,ImgeSize);}
        GetImage(SmallBlockWork,(TopY+5),(SmallBlockwork+SmallBlockSize),(BotY-5),PX);
       end;
     end;

  Function InttoStr(i  :  Longint):String;

  { This function converts a longinteger value to an string type for easier
    displaying in  the graphic's modes }

   Var
   SIntStr             : String[11];

   Begin

    Str(i,SIntStr);
    InttoStr:=SIntStr;

   end;  { end InttoStr }

   Function StrFill(StrCount: Integer; StringInput:String):String;

    Var
     StringFillWork      : String;
     FillSize            : Byte;
     StrFSize            : Integer;

    Begin
     StringFillwork:='                                                  ';
     StringFillWork:=StringFillWork + StringInput;
     FillSize:=ORD(StringFillwork[0]);
     StrFSize:=FillSize-Strcount;
     StringFillWork:=Copy(StringFillwork,StrFSize,FillSize);
     Strfill:=StringFillWork;
    end;

   Procedure StatusBlock(WholeBlock:Boolean);
    Var
     LineBlank          : String;
     Line1              : String;
     Line2              : String;
     Line3              : String;
     Line4              : String;
     Line5              : String;
     Line6              : String;
     ValueString        : String;
     StatusCounter      : Integer;
     YSTATUS            : Integer;

   Begin
    SetColor(0);
    SetTextStyle(DefaultFont,HorizDir,1);
    PutImage(323,255,PX,Normalput);

    Line1:='     --- CURRENT STATUS ---      ';
    Line2:='- Memory Available  : ';
    Line3:='- Start Address     : ';
    Line4:='- Stop Address      : ';
```

```
 Line5:='- Address Increment : ';
 Line6:='- Current FileName  : ';

 For StatusCounter:= 0 to 5 do
  begin
  Case StatusCounter of
   0  : Begin
         LineBlank:=Line1;
        end;
   1  : Begin
         ValueString:=InttoStr(MaxAvail);
         ValueString:=StrFill(10,ValueString);
         LineBlank:=Line2+ValueString;
        end;
   2  : Begin
         ValueString:=InttoStr(StartA);
         ValueString:=StrFill(10,ValueString);
         LineBlank:=Line3+ValueString;
        end;
   3  : Begin
         ValueString:=InttoStr(StopA);
         ValueString:=StrFill(10,ValueString);
         LineBlank:=Line4+ValueString;
        end;
   4  : Begin
         ValueString:=InttoStr(Increment);
         ValueString:=StrFill(10,ValueString);
         LineBlank:=Line5+ValueString;
        end;
   5  : Begin
         ValueString:=FileInput;
         ValueString:=StrFill(10,ValueString);
         LineBlank:=Line6+ValueString;
        end
  end;
  YStatus:=258+(Statuscounter*TextHeight(LineBlank));
  OUTTEXTXY(330,YStatus,LineBlank);
 end;
end;

Function DOTtoADR(DOT:Integer):LongInt;

 Var
  IPReal     : Real;
  Realwork   : Real;

 Begin
  RealWork:=DOT/XaxisInc;
  IPReal:=INT(RealWork);
  IF RealWork > IPReal then
   begin
    DottoADR:=Round(IPReal)+1;
   end
  Else
   Begin
    DottoADR:=Round(IPReal);
   end;
  end;


  Procedure InstructionPut(ChooseInstruction:Integer);
 Var
  LineBlank           : String;
  Line1               : String;
  Line2               : String;
  Line3               : String;
  Line4               : String;
  Line5               : String;
  Line6               : String;
  ValueString         : String;
  WorkInt             : LongInt;
  StatusCounter       : Integer;
  YSTATUS              : Integer;

Begin
 SetColor(0);
 SetTextStyle(DefaultFont,HorizDir,1);
 PutImage(17,255,PX,NormalPut);

 Case ChooseInstruction of

 1 : Begin
       Line1:='         --- Main Screen ---      ';
       Line2:=' ';
       Line3:=' ';
       Line4:='   Select an item from the Menu';
       Line5:=' ';
       Line6:=' ';
     end;
 2 : Begin
       Line1:='       --- Quick View Info ---      ';
       Line2:=' ';
       Line3:='     Select Point to ZOOM in ';
       Line4:='   Use the Mouse or Arrow keys ';
       Line5:='     to position the curser ';
       Line6:=' Press ENTER or Click Mousebutton  ';
     end;
 3 : Begin
       Line1:='       --- Quick View Info ---      ';
       Line2:=' ';
       Line3:=' ';
       Line4:=' ';
       Line5:='    Press any key to continue ';
       Line6:=' ';
     end;
 4 : Begin
       Line1:='       --- Stretch View Info ---      ';
       Line2:=' ';
       Line3:='  Select LEFT point for Stretching';
       Line4:='    Use the Mouse or Arrow keys ';
       Line5:='      to position the curser ';
       Line6:= ' Press ENTER or Click Mousebutton  ';

     end;
 5 : Begin
       Line1:='       --- Stretch View Info ---      ';
       Line2:=' ';
       Line3:=' Select RIGHT point for Stretching';
       Line4:='   Use the Mouse or Arrow keys ';
       Line5:='     to position the curser ';
       Line6:=' Press ENTER or Click Mousebutton  ';
     end;
 6 : Begin
       Line1:='       --- Reset Info ---      ';
       Line2:=' ';
       Line3:=' Reset all the variables to  ';
       Line4:='        starting state        ';
       Line5:=' ';
       Line6:=' ';
     end;
 7 : Begin
       Line1:='         --- MIN/MAX Info ---      ';
       Line2:=' ';
       Line3:=' - Minimum Value     : ';
       ValueString:=InttoStr(MinVideoLevel);
       ValueString:=StrFill(10,ValueString);
       Line3:=Line3+ValueString;
       Line4:=' - Maximum Value     : ';
       ValueString:=InttoStr(MaxVideoLevel);
       ValueString:=StrFill(10,ValueString);
       Line4:=Line4+ValueString;
       Line5:=' ';
       Line6:=' ';
     end;
 8 : Begin
```

```
      Line1:='       --- Graph Calculator ---      ';
      Line2:=' ';
      Line3:=' - Address Pixel       : ';
      WorkInt:=DottoADR(CurXput)+StartA;
      ValueString:=InttoStr(WorkInt);
      ValueString:=StrFill(8,ValueString);
      Line3:=Line3+ValueString;
      Line4:=' ';
      Line5:=' - Current Pixelvalue : ';
      WorkInt:=YDisplay-76-CurYPut;
      ValueString:=InttoStr(WorkInt);
      ValueString:=StrFill(8,ValueString);
      Line5:=Line5+ValueString;
      Line6:=' ';
    end;
 9 : Begin
      Line1:='       --- Get FRAME ---     ';
      Line2:=' ';
      Line3:='    Busy reading and plotting';
      Line4:='          Please WAIT ';
      Line5:=' ';
      Line6:=' Press any Key to ABORT Plotting';
    end;
10 : Begin
      Line1:=' ';
      Line2:=' ';
      Line3:='     --- Writing to DISK --- ';
      Line4:=' ';
      Line5:=' ';
      Line6:='          Please WAIT   ';
    end;
11 : Begin
      Line1:='  --- New Video Mode ---   ';
      Line2:=' 1. Hercules [720 * 348]   ';
      Line3:=' 2. CGA - 2 Colors         ';
      Line4:=' 3. CGA - 4 Colors         ';
      Line5:=' 4. EVA/VGA                ';
      Line6:=' 5. Extended VGA           ';
    end;
12 : Begin
      Line1:='Which driver would you like to use?';
      Line2:='  0) Svga256';
      Line3:='  1) Svga16';
      Line4:='  2) Tweak256';
      Line5:='  3) Tweak16';
      Line6:='  4) Svga32k';
    end;
13: Begin
     Line1:='  0) 320x200x256';
     Line2:='  1) 640x400x256';
     Line3:='  2) 640x480x256';
     Line4:='  3) 800x600x256';
     Line5:='  4) 1024x768x256';
     Line6:='  5) 640x350x256';
    end;
14: Begin
     Line1:='  0) 320x200x16';
     Line2:='  1) 640x200x16';
     Line3:='  2) 640x350x16';
     Line4:='  3) 640x480x256';
     Line5:='  4) 800x600x16';
     Line6:='  5) 1024x768x16';
    end;
15: Begin
     Line1:='  0) 320x400x256';
     Line2:='  1) 320x480x256';
     Line3:='  2) 360x480x256';
     Line4:='  3) 376x564x256';
     Line5:='  4) 400x564x256';
     Line6:='  5) 400x600x256';
    end;
16: Begin
     Line1:='  0) 704x528x16';
     Line2:='  1) 720x540x16';
     Line3:='  2) 736x552x16';
     Line4:='  3) 752x564x256';
     Line5:='  4) 768x576x16';
     Line6:='  5) 784x588x16';
    end;
17: Begin
     Line1:='  0) 320x200x32768';
     Line2:='  1) 640x350x32768';
     Line3:='  2) 640x400x32768';
     Line4:='  3) 640x480x32768';
     Line5:='  4) 800x600x32768';
     Line6:=' ';
    end;
18: Begin
     Line1:=' Please Select File Format ';
     Line2:='  ';
     Line3:='   1. Normal Save';
     Line4:='   2. Save in PCX-Format';
     Line5:='  ';
     Line6:=' ';
    end;
 end;


 For StatusCounter:= 0 to 5 do
  begin
  Case StatusCounter of
   0  : Begin
         LineBlank:=Line1;
        end;
   1  : Begin
         LineBlank:=Line2;
        end;
   2  : Begin
         LineBlank:=Line3;
        end;
   3  : Begin
         LineBlank:=Line4;
        end;
   4  : Begin
         LineBlank:=Line5;
        end;
   5  : Begin
         LineBlank:=Line6;
        end
  end;
  YStatus:=258+(Statuscounter*TextHeight(LineBlank));
  OUTTEXTXY(20,YStatus,LineBlank);
 end;
end;

Procedure PICScreenSetup;
 Var PICShadow                 : Integer;

 { This routine draws the screensetup as needed before a picture is
  drawn }

 Begin
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  SetColor(0);
  PicShadow:=5;
  If (GetMaxY-ViewPicY) < 15 then
  PicShadow:=0;
  SetFillStyle(1,0);
  Xwork:=((GetmaxX +1) div 2)  - ((ViewPicX+1) div 2 );
  Ywork:=((GetmaxY +1) div 2)  - ((ViewPicY+1) div 2 );
  Bar((Xwork),(Ywork),(Xwork+ViewPicX),(Ywork+ViewPicY));
  IF ((Xwork+PICShadow+ViewPicX) <= GetmaxX) AND ((Ywork+PICShadow+ViewPicY) <= GetMaxY)
  Then;
   Begin
    Bar((Xwork+PICShadow),(Ywork+PICShadow),(Xwork+PICShadow+ViewPicX),(Ywork+PICShadow+ViewPicY));
    SetColor(MaxColor);
```

```
     Rectangle((Xwork-1),(Ywork-1),(Xwork+ViewPicX+1),(Ywork+ViewPicY+1));
    end;
   SetViewPort(Xwork,Ywork,(Xwork+ViewPicX),(Ywork+ViewPicY),True);
   ClearViewPort;
 END;


 Procedure ScaleCalc(Starting,Stopping,Dot:Longint; DirectInc: Integer);

 { This procedure calculates the increment rate needed to get
   'Dot' number of X dots on the screen }

  Var

  NumberofValues              : Longint; { Total nuber of values read
                                           form I/O device or Disk }

  Begin
   NumberofValues:= ABS(Stopping-Starting) div DirectInc;
   XaxisINC:= Dot/NumberofValues;   { Calculation of x-axis increment
                                        ratio as use in the graph section }
  end; { end ScaleCalc procedure }


  Function XPixelPos(Value : Longint):Integer;

  { This Procedure converts the current addressvalue is in the counter
    to a physical pixel on the screen Thus AddressStart ( Normally 0 )
    will be placed at X-Pixel 0 and AddresStop will be placed at
    the maximum DOTS specified for the current display type }

  Var

   RealWork                   : Real; { Variable used to convert from a
                                         real to an Integer type }

  Begin
   RealWork:=Int(Value*XaxisINC);  { Give a value of 0..DOTS }
   XPixelPos:=Round(RealWork);  { Convert from Real to Integer }
  end;

Procedure ShowGraphArray(ArrayBlock :DataArray; StartArray,StopArray,VideoToggle:Integer);

  Label

   QSHOW;

  Var

   ArrayCount        : Longint; { Counter used to count through address }
   SingleByte        : Byte;    { Byte as read from disk }
   SEGAddress        : Word;    { Address of current SingleByte }
   PixelValue        : Integer; { Working Variable use to get Pixel value }
   Xpixel,Ypixel     : Integer; { X,Y-Position on the screen }
   ArrayScale        : Real;
   NumWrite          : Integer;
   GrayPCX           : Real;
   PCXColor          : Byte;

 Begin
  ArrayScale:=800/(VideoLength-150);
  GrayPCX:=255/(MaxVideoLevel-VideoLevel);
  For Arraycount:= StartArray to StopArray do
   Begin
   IF NOT(PCXConvert) then
    Begin
    IF (KeyPressed) Then    { QuickEscape while busy drawing the screen }
     Begin
      KeyRead:=ReadKey;
      If KeyRead = #0 then KeyRead:=ReadKey;   { Use to decode funtion keys}
      HaltDraw:=True;
      Goto QSHOW;
     end;
    end;
    SingleByte:=Arrayblock[ArrayCount];   { Get current SingleByte }
    PrevSynchTest:=SynchTest;
    IF SingleByte > 127 then SynchTest:=False else SynchTest:=True;

    { The SynchTest Variable is used to get the correct synch pulse info
      For this variable to work efficiently the composite synch must be
      supplied to the ODD/EVEN pin of the XILINX IC. If ODD/EVEN is supplied
      SynchTest Flag MUST be set to False }

    SEGAddress:=SEG(SingleByte);  { Current DS used for Variables }
    ASM
     PUSH DS;            { Store old DS on the stack }
     MOV DS,SEGAddress;  {Get new value for the datasegment}
     LEA SI,SingleByte;  {Getaddress of variable}
     LODSB;
     AND AL,127;         {AND with 127 to get MSB bit to 0}
     MOV SingleByte,AL;  {Save value in Variable}
     POP DS;             { Restore OLD DS Value }
    END;

{ The above machine code section remove any Synch Info Stored }


   PCXColor:=SingleByte;

   IF VideoToggle = 0 then   { Draw as a Graphic }
    Begin
     SetColor(MaxColor);
     PixelValue:=XpixelPos(GraphXCount);
     PrevSingle:=SingleByte;         { Store Current value to Prev Value }
     Ypixel:= (Ydisplay-75)-SingleByte;  { Get Y-Axis Position }
     Xpixel:=Pixelvalue+Xoffset;         { Get X-Axis Position }
     IF DotFlag then
      PutPixel(XPixel,YPixel,MaxColor)  { Put - Pixel on the Screen }
     ELSE
      If GraphXCount = 0 then   { On the first count only a pixel must be drawn }
       Begin
        MoveTo(XPixel,YPixel);
        PutPixel(XPixel,YPixel,MaxColor)
       end
      Else LineTo(XPixel,Ypixel);
    end;  { end of VideoToggle = 0 }

   If VideoToggle = 1 then
    Begin
    If (SynchTest AND NOT(PrevSynchTest)) OR SynchFlag then
     Begin
     If SynchTest AND NOT(PrevSynchTest) then HCount:=0;
      Inc(SynchCount);
      SynchFlag:=True;
      IF NOT(SynchTest) AND PrevSynchTest then
       Begin
        SynchFlag:=False;
        If (SynchCount > VideoMarkMax) AND (TopCount >= 304) then
         Begin
          TopCount:=0;
          Vcount:=1;
          YposPIC:=Round(YincPic*VCount);
          Vcount:=Vcount;
          IF PCXConvert then
           Begin
            Close(PCX);
            Assign(PCX,'EVEN');
            RESET(PCX,1);
            SEEK(PCX,0);
           end;
         end;
         SynchCount:=0;
       end;
     end;
     INC(HCount);
     If (HCount > 150) AND NOT(SynchFlag)  then
```

```
       Begin
        If (Hcount <= VideoLength)  then
        Begin
         INC(VidCount);
         If TopCount >= TopStop then
         Begin
          If ColorFlag then
           Begin
            IF SingleByte < VideoLevel then SingleByte:=VideoLevel;
            SingleByte:=Round(GrayScale*(SingleByte-VideoLevel));
            IF SingleByte > MaxColor then SingleByte:=MaxColor;
           end
          Else
           Begin
            If SingleByte >=BwLevel then SingleByte:=MaxColor else SingleByte:=0;
           end;
          XposPic:=Round(VidCount*(XincPic));
          If Vcount < TotalVShow then
           begin
            PutPixel(XposPic,YposPic,SingleByte);
            PutPixel(XposPic+1,YposPic,SingleByte);
             If GetMaxY > 500 then
              Begin
               PutPixel(XposPic,YposPic+1,SingleByte);
               PutPixel(XposPic+1,YposPic+1,SingleByte);
              end;
              If PCXConvert then
              Begin
               XposPic:=Round(VidCount*ArrayScale);
               IF PCXColor < VideoLevel then PCXColor:=VideoLevel;
               PCXColor:=Round(GrayPCX*(PCXColor-VideoLevel));
               PCXArray[XposPic]:=PCXColor;
               PCXArray[XPosPic+1]:=PCXColor;
               IF XposPic=800 then
                Begin
                 BLOCKWRITE(PCX,PCXArray,800,NumWrite);
                end;
              end;
             end;

         If Hcount = VideoLength then
          Begin
           INC(VCount);
           INC(VCount);
           VidCount:=0;
           YposPIC:=Round(YincPic*VCount);
          end;
         end;
         If Hcount = VideoLength then INC(TopCount);
        end;
       end;
      end;
     INC(GraphXCount);        (Current Position in the Graphic display)
    end;
QSHOW: end;                        { End of procedure }


   Procedure ShowGrabValue(Address:LongInt; VideoToggle:Integer);

   Label

     Grabout;

   Var
     GAddressL           : Word;     { Low address  }
     GAddressM           : Word;     { Med address  }
     GAddressH           : Word;     { High address }
     ASMAddress          : Word;     { DS for Address }
     ACNT                : LongInt;  { Delay Counter }
     AddressDelay        : LongInt;  { Address-Address delay }
     ReadDelay           : Longint;  { Address to DataValid }
   ArrayCount          : Longint; { Counter used to count through address }
   SingleByte          : Byte;    { Byte as read from disk }
   SEGAddress          : Word;    { Address of current SingleByte }
   PixelValue          : Integer; { Working Variable use to get Pixel value }
   Xpixel,Ypixel       : Integer; { X,Y-Position on the screen }

 Begin
  ASMAddress:=SEG(Address);          { Get current segment used }
  AddressDelay:=1;
  ReadDelay:=1;

- ASM
   PUSH DS;
   MOV DS,ASMAddress;
   LEA SI,Address;
   LODSW;
   MOV BX,AX;
   MOV AH,00;
   MOV GAddressL,AX;    { Low address calculation }
   MOV AL,BH;
   MOV GAddressM,AX;    { Med address calculation }
   LODSW;
   MOV GAddressH,AX;    { High address calculation }
   POP DS;
  END;

   Port[$302]:=GAddressH;   { High address #302h }
   For ACNT:= 0 TO AddressDelay do Begin end;
   Port[$301]:=GAddressM;   { Meduim address #301h }
   For ACNT:= 0 TO AddressDelay do Begin end;
   Port[$300]:=GAddressL;   { Low address #300h }
   For ACNT:= 0 TO AddressDelay do Begin end;

   SingleByte:=Port[$303];  { read form the frame grabber }
    IF (KeyPressed) Then
     Begin
      HaltDraw:=True;
      Goto GrabOUT;
     end;

   PrevSynchTest:=SynchTest;
   IF SingleByte > 127 then SynchTest:=False else SynchTest:=True;

   { The SynchTest Variable is used to get the correct synch pulse info
     For this variable to work efficiently the composite synch must be
     supplied to the ODD/EVEN pin of the XILINX IC. If ODD/EVEN is supplied
     SynchTest Flag MUST be set to False }

   SEGAddress:=SEG(SingleByte);  { Current DS used for Variables }
   ASM
    PUSH DS;             { Store old DS on the stack }
    MOV DS,SEGAddress;   {Get new value for the datasegment}
    LEA SI,SingleByte;   {Getaddress of variable}
    LODSB;
    AND AL,127;          {AND with 127 to get MSB bit to 0}
    MOV SingleByte,AL;   {Save value in Variable}
    POP DS;              { Restore OLD DS Value }
   END;

{ The above machine code section remove any Synch Info Stored }


  IF VideoToggle = 0 then   { Draw as a Graphic }
   Begin
    PixelValue:=XpixelPos(GraphXCount);
    PrevSingle:=SingleByte;          { Store Current value to Prev Value }
    Ypixel:= (Ydisplay-75)-SingleByte;  { Get Y-Axis Position }
    Xpixel:=Pixelvalue+Xoffset;         { Get X-Axis Position }
    IF DotFlag then
     PutPixel(XPixel,YPixel,MaxColor)  { Put - Pixel on the Screen }
    ELSE
     IF GraphXCount = 0 then   { On the first count only a pixel must be drawn }
    Begin
     MoveTo(XPixel,YPixel);
     PutPixel(XPixel,YPixel,MaxColor)
    end
   Else LineTo(XPixel,Ypixel);
 end;  { end of VideoToggle = 0 }

If VideoToggle = 1 then
 Begin
 If (SynchTest AND NOT(PrevSynchTest)) OR SynchFlag then
  Begin
  If SynchTest AND NOT(PrevSynchTest) then HCount:=0;
   Inc(SynchCount);
   SynchFlag:=True;
   IF NOT(SynchTest) AND PrevSynchTest then
    Begin
     SynchFlag:=False;
     If (SynchCount > VideoMarkMax) AND (TopCount >= 304) then
      Begin
       TopCount:=0;
       Vcount:=1;
       YposPIC:=Round(YincPic*VCount);
       Vcount:=Vcount;
      end;
      SynchCount:=0;
    end;
  end;
  INC(HCount);
  If (HCount > 150) AND NOT(SynchFlag)  then
   Begin

    If (Hcount <= VideoLength)  then
    Begin
     INC(VidCount);
     If TopCount > TopStop then
     Begin
     If ColorFlag then
      Begin
       IF SingleByte < VideoLevel then SingleByte:=VideoLevel;
       SingleByte:=Round(GrayScale*(SingleByte-VideoLevel));
       IF SingleByte > MaxColor then SingleByte:=MaxColor;
      end
     Else
      Begin
       If SingleByte >=BwLevel then SingleByte:=MaxColor else SingleByte:=0;
      end;
       XposPic:=Round(VidCount*(XincPic));
       If Vcount < TotalVShow then
        begin
         PutPixel(XposPic,YposPic,SingleByte);
         PutPixel(XposPic+1,YposPic,SingleByte);
         If GetMaxY > 500 then
          Begin
           PutPixel(XposPic,YposPic+1,SingleByte);
           PutPixel(XposPic+1,YposPic+1,SingleByte);
          end;
        end;
     If Hcount = VideoLength then
      Begin
       INC(VCount);
       INC(VCount);
       VidCount:=0;
       YposPIC:=Round(YincPic*VCount);
      end;
     end;
     If Hcount = VideoLength then INC(TopCount);

    end;
   end;
  end;
 INC(GraphXCount);       {Current Position in the Graphic display}
```

```
GrabOUT:    If HaltDraw then
                 Begin
                  KeyRead:=ReadKey;
                  If KeyRead = #0 then KeyRead:=Readkey;
                 end;
     end;



   Procedure GrabGraph(StartAddress,StopAddress:LongInt; AdrInc:Integer);

   Label GrabJump;

   Var
    GrabCount                  : LongInt;   { Counter for the address in grabber }

    Begin
     HaltDraw:=False;
     GraphXCount:=0;
     GrabCount:=StartAddress;
     Repeat
      ShowGrabValue(GrabCount,GraphPIC);
      If HaltDraw then Goto GrabJump;
      GrabCount:=GrabCount+AdrINC;
     Until GrabCount > StopAddress;

GrabJump:    end;


   Procedure DiskGraph(StartAddress,StopAddress:Longint);

   { This procedure reads in the current selectfile starting at
     StartAddress & StopAddress as entered Boundaries for reading a
     file }
   Label

    OutDiskGraph;

   Var

   GraphFileSize        : Longint;  { Actual filesize on disk }
   GraphSize            : Longint;  { Size to be write to RAM }
   MaxGraphSize         : Longint;  { Maximum Size to be write savely }
   BlockSize            : Integer;  { Amount of full memory blocks }
   BlockOver            : Integer;  { Left over from Memory blocks }
   BlockCntGraph        : Integer;
   NumGraph             : Word;     { Amount actually written from disk }
   GraphString          : String;

   Begin

    GraphSize:=ABS(StopAddress-StartAddress); { Amount of bytes to be read }
    MaxGraphSize:=MemBlock*MaxBlockSize; { Make sure that readsize is not }
    If GraphSize > MaxGraphSize then        { bigger than the amount of memory }
     Begin                                { available }
      GraphSize:=MaxGraphsize;
     end;

    BlockSize:=Graphsize div MaxBlockSize;
    BlockOver:=Graphsize mod MaxBlockSize;
    BlockCntGraph:=0;

    For BlockCntGraph:= 1 to BlockSize do
      Begin
        D1:=AddressPoint[BlockCntGraph];
        ShowGraphArray(D1^,1,MaxBlockSize,GraphPIC);  { Show the whole array }
        If HaltDraw then goto OutDiskGraph;
      end;


      BlockSize:=BlockSize+1;  { Store the remainder in the next block }
      D1:=AddressPoint[BlockSize];
```

```
       ShowGraphArray(D1^,1,BlockOver,GraphPIC);
       D1:=AddressPoint[1];  { Restore the correct value for D1 so that is
                               pointing to d1 and not to anyting else due
                               to the above section where the same name d1 is
                               used but the variable to which it point are
                               changed. }

OutDiskGraph:    end;


{ ************************************************************************** }
{ This part of the program are used to handle the graphics mouse It uses s
  special techniques of mouse operating. Then Mouse Curser is not displayed
  by the normal way of using the graphics curser as defined by INT 33H function
  09H. This part uses Pascal GetImage & Putimage commands along with a TSR
  program which operate on the timer tick interrupt 1Ch. This method were used
  due to the absence of a graphic mouse curser for Hercules and SVga Graphics
  mode.}
{************************************************************************** }

   Procedure MouseCurser;

   { This procedure defines a mouse curser which will work in all graphics
     modes, This procedure makes use of pascal getimage,putimage commands }

     Var
      MouseCurserColor          : Integer;  { Select mouse color }
      CurserCntRow              : Integer;  { Counter for to Initialize }
      CurserCntCol              : Integer;  { Counter for to Initialize }
      X_Offset                  : Integer;  { X Offset for PUTPIXEL }
      Y_Offset                  : Integer;  { X Offset for PUTPIXEL }
      Xmouse                    : Integer;
      Ymouse                    : Integer;
      PixelVal                  : Integer;

      Begin
       MouseCurserColor:=63; { Get maxcolor of current palette }
       X_Offset:=9;
       Y_Offset:=9;
       For CurserCntRow:= 1 to 16 do
        Begin
         For CurserCntCol:= 1 to 16 do
          Begin
           Xmouse:= CurserCntCol+X_Offset;
           YMouse:= CurserCntRow+Y_Offset;
           PixelVal:= MousecurserColor*ScreenCurser[CurserCntRow,CurserCntCol];
           Putpixel(Xmouse,Ymouse,PixelVal);  { Draw the first curser pixel }
          end;                                  { by pixel }
         end;
         ImgeSize:=ImageSize(10,10,25,25);
      {   GetMem(CurserScreen,ImgeSize); }
         GetImage(10,10,25,25,CurserScreen);   { Get the curser and store it in a pointer }
       MouseCurserColor:=63;
       X_Offset:=39;
       Y_Offset:=9;
       For CurserCntRow:= 1 to 16 do
        Begin
         For CurserCntCol:= 1 to 16 do
          Begin
           Xmouse:= CurserCntCol+X_Offset;
           YMouse:= CurserCntRow+Y_Offset;
           PixelVal:= MousecurserColor*MaskCurser[CurserCntRow,CurserCntCol];
           Putpixel(Xmouse,Ymouse,PixelVal);  { Draw Mouse Mask }
          end;
         end;
         ImgeSize:=ImageSize(40,10,55,25);
{         GetMem(CurserMask,ImgeSize);}
         GetImage(40,10,55,25,CurserMask);  { Store Mouse Mask in pointer }
         MousePixel:=8;
        end;
```

```
Function MouseMoveDetect:Boolean;

 Begin
  MouseMovFlag:=False;
  MouseKeypressed:=False;
  Regs.AX:=3;    { Get the current status of the mouse }
  Intr($33,Regs);
  IF regs.BX <> 0 then MouseKeyPressed:= True;
  CurXPut:=Regs.CX div MousePixel; { X-position indicator }
  CurYPut:=Regs.DX div MousePixel; { Y-position indicator }


  IF (PrevXput <> CurXput) OR (PrevYput <> CurYput) then
   Begin
    MouseMoveDetect:=True;
    MouseMovFlag:=True;
   end
   Else
    MouseMoveDetect:=False;
  end;



Procedure PutCurser;

{ This procedure forms the wetch and is run every 18.2 times per second
  This procedure check if the x,y position of the mouse have changed since
  it were last run. If it have changed the mouse curser is moved to its
  correct new position. The Counter CountMouse will determain how often the
  mouse curser on the screen are updated. By making the Countmouse compare
  value bigger than it current value it is possible to get a slower but
  greater incremental steps on the screen. }

 Begin
  IF Not(HideCurser) then
   Begin
    PutImage(PrevXPut,PrevYput,CurSCNImg,Normalput);
     { Restore previous Screen pixel information }
    GetImage(CurXput,CurYput,(CurXput+16),(CurYput+16),CurSCNImg);
     { Gets the Screen pixel information of the current curser position }
    PutImage(CurXput,CurYput,CurserScreen,ANDput);
     { Place the CurserScreen }
    PutImage(CurXput,CurYput,CurserMask,ORput);
     { Place the final curser }
   end;
   PrevXPut:=CurXput;
   PrevYPut:=CurYput;

 end;

Procedure MouseStarting;   { First time initialization of the mouse }

   Begin
    Regs.AX:=3;
    Intr($33,regs);
    PrevXput:=0;
    PrevYput:=0;
    ImgeSize:=ImageSize(PrevXput,PrevYput,(PrevXput+16),(PrevYput+16));
  {  GetMem(CurSCNImg,ImgeSize); }
    GetImage(PrevXput,PrevYput,(PrevXput+16),(PrevYput+16),CurSCNImg);
   end;


Procedure InitializeMouse;

 Var
  MouseIntCnt            : Integer;

 Begin
   Regs.Ax:=0;           { Reset Mouse Driver and get current Status }
   Intr($33,regs);
   Regs.Ax:=2;           { Hide the Mouse curser to use own curser }
```

```
    Intr($33,regs);

    Regs.Ax:=$F;
    Regs.CX:=1;
    Regs.DX:=1;
    Intr($33,Regs);

    Xwork:=((GetmaxX) div 2)  - (XScreenSize div 2 );
    Ywork:=((GetmaxY) div 2)  - (YScreenSize div 2 );
    Regs.CX:=0; {(Xwork-1);}
    Regs.DX:=(XScreenSize+1)*MousePixel;
    Regs.AX:=7;          { Define Horizontal curser range }
    Intr($33,regs);

    Regs.CX:=0; {(Ywork-1);}
    Regs.DX:=(YScreenSize+1)*MousePixel;
    Regs.AX:=8;          { Define Vertical Curser Range }
    Intr($33,regs);

    Regs.AX:=4;
    Regs.CX:=0;
    Regs.DX:=0;
    Intr($33,regs);      { Set the Starting Position at 0,0 }

  End;

  Procedure HideMouse;
   Begin
    PutImage(PrevXPut,PrevYput,CurSCNImg,Normalput);
    HideCurser:=True;
   end;

  Procedure ShowMouse;
   Begin
     GetImage(CurXput,CurYput,(CurXput+16),(CurYput+16),CurSCNImg);
      { Gets the Screen pixel information of the current curser position }
     PutImage(CurXput,CurYput,CurserScreen,ANDput);
      { Place the CurserScreen }
     PutImage(CurXput,CurYput,CurserMask,XORput);
      { Place the final curser }
    PrevXPut:=CurXput;
    PrevYPut:=CurYput;
    HideCurser:=False;
   end;

  Procedure MOUSE;

   Begin
                    { Define  Graphics Curser }
    InitializeMouse; { Initialize the mouse driver }
    MouseStarting;   { Define starting Position }

   end;

{ ************************************************************************ }
                  { This part of the program are used to give
                   disk-capabilities of uptil 520k Direct read
                  and write to and from the Memory of a standard
                  IBM- PC with a minimum of 580k of usable memory
                                    is possible.}
{************************************************************************ }



  Procedure GetMemoryBlocks;

   Var

    MemBlockCnt        :Integer;  { Counter used to initialize d0..d? }
    MemString          :String;
```

```
Begin
 MaxMemory:=MaxAvail-ProgramMem; { Get maximum user memory for
                                   the current machine    }

 IF MaxMemory > MaxUserMemory then
  Begin
   MaxMemory:=MaxUserMemory;      { Define Maximum user memory for a PC }
  end;                            { Base system at 520K of RAM }

 MemBlock := MaxMemory div MaxBlockSize;
 MaxBlockRead := MemBlock * MaxBlockSize;   { Set Maximum Read/Write
                                              Buffer for current machine}
 MaxBlockWrite := MaxBlockRead;
 {Writeln('Maximum Memory Buffer in Kbytes.: ',(MaxBlockRead div 1024));}

  For MemBlockCnt := 1 to MemBlock do
   Begin
{    Writeln('D',MemBlockCnt);  Test to check amount of blocks }
    Case MemBlockCnt of
         1 :  Begin New(D1); AddressPoint[1]:=Addr(D1^); end;
         2 :  Begin New(D2); AddressPoint[2]:=Addr(D2^); end;
         3 :  Begin New(D3); AddressPoint[3]:=Addr(D3^); end;
         4 :  Begin New(D4); AddressPoint[4]:=Addr(D4^); end;
         5 :  Begin New(D5); AddressPoint[5]:=Addr(D5^); end;
         6 :  Begin New(D6); AddressPoint[6]:=Addr(D6^); end;
         7 :  Begin New(D7); AddressPoint[7]:=Addr(D7^); end;
         8 :  Begin New(D8); AddressPoint[8]:=Addr(D8^); end;
         9 :  Begin New(D9); AddressPoint[9]:=Addr(D9^); end;
        10 :  Begin New(D10); AddressPoint[10]:=Addr(D10^); end;
        11 :  Begin New(D11); AddressPoint[11]:=Addr(D11^); end;
        12 :  Begin New(D12); AddressPoint[12]:=Addr(D12^); end;
        13 :  Begin New(D13); AddressPoint[13]:=Addr(D13^); end;
        14 :  Begin New(D14); AddressPoint[14]:=Addr(D14^); end;
        15 :  Begin New(D15); AddressPoint[15]:=Addr(D15^); end;
        16 :  Begin New(D16); AddressPoint[16]:=Addr(D16^); end;
        17 :  Begin New(D17); AddressPoint[17]:=Addr(D17^); end;
        18 :  Begin New(D18); AddressPoint[18]:=Addr(D18^); end;
        19 :  Begin New(D19); AddressPoint[19]:=Addr(D19^); end;
        20 :  Begin New(D20); AddressPoint[20]:=Addr(D20^); end;
        21 :  Begin New(D21); AddressPoint[21]:=Addr(D21^); end;
        22 :  Begin New(D22); AddressPoint[22]:=Addr(D22^); end;
        23 :  Begin New(D23); AddressPoint[23]:=Addr(D23^); end;
        24 :  Begin New(D24); AddressPoint[24]:=Addr(D24^); end;
        25 :  Begin New(D25); AddressPoint[25]:=Addr(D25^); end;
        26 :  Begin New(D26); AddressPoint[26]:=Addr(D26^); end;
        27 :  Begin New(D27); AddressPoint[27]:=Addr(D27^); end;
        28 :  Begin New(D28); AddressPoint[28]:=Addr(D28^); end;
        29 :  Begin New(D29); AddressPoint[29]:=Addr(D29^); end;
        30 :  Begin New(D30); AddressPoint[30]:=Addr(D30^); end;
        31 :  Begin New(D31); AddressPoint[31]:=Addr(D31^); end;
        32 :  Begin New(D32); AddressPoint[32]:=Addr(D32^); end;
        33 :  Begin New(D33); AddressPoint[33]:=Addr(D33^); end;
        34 :  Begin New(D34); AddressPoint[34]:=Addr(D34^); end;
        35 :  Begin New(D35); AddressPoint[35]:=Addr(D35^); end;
        36 :  Begin New(D36); AddressPoint[36]:=Addr(D36^); end;

     end { end of the case statement }

   end;
end;


Procedure ReleaseMemoryBlocks;

Var

 MemBlockCnt          :Integer;  { Counter used to initialize d0..d? }
 MemString            :String;

Begin

  For MemBlockCnt := 1 to MemBlock do
   Begin
```

```
    Case MemBlockCnt of
         1 :  Begin Release(D1) end;
         2 :  Begin Release(D2) end;
         3 :  Begin Release(D3) end;
         4 :  Begin Release(D4) end;
         5 :  Begin Release(D5) end;
         6 :  Begin Release(D6) end;
         7 :  Begin Release(D7) end;
         8 :  Begin Release(D8) end;
         9 :  Begin Release(D9) end;
        10 :  Begin Release(D10) end;
        11 :  Begin Release(D11) end;
        12 :  Begin Release(D12) end;
        13 :  Begin Release(D13) end;
        14 :  Begin Release(D14) end;
        15 :  Begin Release(D15) end;
        16 :  Begin Release(D16) end;
        17 :  Begin Release(D17) end;
        18 :  Begin Release(D18) end;
        19 :  Begin Release(D19) end;
        20 :  Begin Release(D20) end;
        21 :  Begin Release(D21) end;
        22 :  Begin Release(D22) end;
        23 :  Begin Release(D23) end;
        24 :  Begin Release(D24) end;
        25 :  Begin Release(D25) end;
        26 :  Begin Release(D26) end;
        27 :  Begin Release(D27) end;
        28 :  Begin Release(D28) end;
        29 :  Begin Release(D29) end;
        30 :  Begin Release(D30) end;
        31 :  Begin Release(D31) end;
        32 :  Begin Release(D32) end;
        33 :  Begin Release(D33) end;
        34 :  Begin Release(D34) end;
        35 :  Begin Release(D35) end;
        36 :  Begin Release(D36) end;

     end  { end of the case statement }

   end;
end;  { End of ReleaseMemBlock }


Procedure CleanRam;  { This procedure reset all the RAM to 0 }

Var

 ClearCount          : Integer;

 Begin

  For Clearcount:= 1 to MemBlock do
   Begin
    D1:=AddressPoint[ClearCount];    { Selects next Memoryblock }
    FillChar(D1^,Sizeof(D1^),0);
   end;

  end;  { end cleanram procedure }


Procedure Hyper_Read(StartAddress,StopAddress:Longint);

Var

 ReadFileSize        : Longint;  { Actual filesize on disk }
 ReadSize            : Longint;  { Size to be read into RAM }
 MaxReadSize         : Longint;  { Maximum Size to be read savely }
 BlockSize           : Integer;  { Amount of full memory blocks }
 BlockOver           : Integer;  { Left over from Memory blocks }
 BlockCntRead        : Integer;
 NumRead              : Word;     { Amount actually read from disk }
 ReadString           : String;

 Begin
  ReadError:=False;

  ReadFileSize:=FileSize(F); { Filesize of current file to be read }
  ReadSize:=ABS(StopAddress-StartAddress); { Amount of bytes to be read }

  MaxReadSize:=MemBlock*MaxBlockSize; { Make sure that readsize is not }
  If ReadSize > MaxReadSize then       { bigger than the amount of memory }
   Begin                               { available }
    ReadSize:=MaxReadsize;
    Readerror:=true;
   end;

  BlockSize:=Readsize div MaxBlockSize;
  BlockOver:=Readsize mod MaxBlockSize;
  Seek(F,StartAddress); {Seek the desired starting point of the file}
  BlockCntRead:=0;

  For BlockCntread:= 1 to BlockSize do

    Begin
      D1:=AddressPoint[BlockCntRead];
      Blockread(F,D1^,MaxBlockSize,NumRead);       end;

    BlockSize:=BlockSize+1;  { Store the remainder in the next block }
    D1:=AddressPoint[BlockSize];
    Blockread(F,D1^,BlockOver,NumRead);
    D1:=AddressPoint[1];  { Restore the correct value for D1 so that is
                            pointing to d1 and not to anyting else due
                            to the above section where the same name d1 is
                            used but the variable to which it point are
                            changed. }

  end;


  Procedure Hyper_Write(StartAddress,StopAddress:Longint);

Var

 WriteFileSize        : Longint;  { Actual filesize on disk }
 WriteSize            : Longint;  { Size to be write to RAM }
 MaxWriteSize         : Longint;  { Maximum Size to be write savely }
 BlockSize            : Integer;  { Amount of full memory blocks }
 BlockOver            : Integer;  { Left over from Memory blocks }
 BlockCntWrite        : Integer;
 NumWrite             : Word;     { Amount actually written from disk }
 WriteString          : String;

 Begin
  WriteError:=False;

  WriteFileSize:=FileSize(FWrite); { Filesize of current file to be read }
  WriteSize:=ABS(StopAddress-StartAddress); { Amount of bytes to be read }

  MaxWriteSize:=MemBlock*MaxBlockSize; { Make sure that readsize is not }
  If WriteSize > MaxWriteSize then       { bigger than the amount of memory }
   Begin                               { available }
    WriteSize:=MaxWritesize;
    WriteError:=true;
   end;

  BlockSize:=Writesize div MaxBlockSize;
  BlockOver:=Writesize mod MaxBlockSize;
  Seek(FWrite,StartAddress); {Seek the desired starting point of the file}
  BlockCntWrite:=0;

  For BlockCntWrite:= 1 to BlockSize do
```

```
    Begin
      D1:=AddressPoint[BlockCntWrite];
      Blockwrite(FWrite,D1^,MaxBlockSize,NumWrite);
    end;

    BlockSize:=BlockSize+1;  { Store the remainder in the next block }
    D1:=AddressPoint[BlockSize];
    Blockwrite(FWrite,D1^,BlockOver,NumWrite);
    D1:=AddressPoint[1];  { Restore the correct value for D1 so that is
                           pointing to d1 and not to anyting else due
                           to the above section where the same name d1 is
                           used but the variable to which it point are
                           changed. }

  end;

{ ******************************************************************** }

Procedure DirectTransfer(StartAddress,StopAddress:Longint);

Var

 ReadSize            : Longint;  { Size to be read into RAM }
 MaxReadSize         : Longint;  { Maximum Size to be read savely }
 BlockSize           : Integer;  { Amount of full memory blocks }
 BlockOver           : Integer;  { Left over from Memory blocks }
 BlockCntRead        : Integer;
 NumRead             : Word;     { Amount actually read from disk }
 ReadString          : String;
 CurrentAddress      : Longint;
 DirectCount         : LongInt;
 DAddressL           : Word;
 DAddressM           : Word;
 DAddressH           : Word;
 ASMAddress          : Word;
 ACnt                : Integer;
 AddressDelay        : Integer;
 ReadByte            : Byte;


 Begin
  ReadError:=False;

  ReadSize:=ABS(StopAddress-StartAddress); { Amount of bytes to be read }

  MaxReadSize:=MemBlock*MaxBlockSize;.{ Make sure that readsize is not }
  If ReadSize > MaxReadSize then     '  { bigger than the amount of memory }
   Begin                              { available }
    ReadSize:=MaxReadsize;
    Readerror:=true;
   end;

  BlockSize:=Readsize div MaxBlockSize;
  BlockOver:=Readsize mod MaxBlockSize;
  BlockCntRead:=0;
  DirectCount:=0;
  CurrentAddress:=StartAddress;

  For BlockCntread:= 1 to BlockSize do

    Begin
      ASMAddress:=SEG(CurrentAddress);
      D1:=AddressPoint[BlockCntRead];
      For DirectCount:= 1 to MaxBlockSize do
       Begin
        ASM
         PUSH DS;
         MOV DS,ASMAddress;
         LEA SI,CurrentAddress;
         LODSW;
         MOV BX,AX;
         MOV AH,00;
         MOV DAddressL,AX;
        MOV AL,BH;
        MOV DAddressM,AX;
        LODSW;
        MOV DAddressH,AX;
        POP DS;
       END;
        Port[$302]:=DAddressH;
         For ACnt:= 0 to AddressDelay do Begin end;
        Port[$301]:=DAddressM;
         For ACnt:= 0 to AddressDelay do Begin end;
        Port[$300]:=DAddressL;
         For ACnt:= 0 to AddressDelay do Begin end;
       ReadByte:=Port[$303];
       D1^[DirectCount]:= ReadByte;
       INC(CurrentAddress);
      end;
   end;

   BlockSize:=BlockSize+1;  { Store the remainder in the next block }
   D1:=AddressPoint[BlockSize];
   For DirectCount:= 1 to BlockOver do
      Begin
       ASM
        PUSH DS;
        MOV DS,ASMAddress;
        LEA SI,CurrentAddress;
        LODSW;
        MOV BX,AX;
        MOV AH,00;
        MOV DAddressL,AX;
        MOV AL,BH;
        MOV DAddressM,AX;
        LODSW;
        MOV DAddressH,AX;
        POP DS;
       END;
        Port[$302]:=DAddressH;
         For ACnt:= 0 to AddressDelay do Begin end;
        Port[$301]:=DAddressM;
         For ACnt:= 0 to AddressDelay do Begin end;
        Port[$300]:=DAddressL;
         For ACnt:= 0 to AddressDelay do Begin end;
       ReadByte:=Port[$303];
       D1^[DirectCount]:= ReadByte;
       INC(CurrentAddress);
      end;

   D1:=AddressPoint[1];  { Restore the correct value for D1 so that is
                           pointing to d1 and not to anyting else due
                           to the above section where the same name d1 is
                           used but the variable to which it point are
                           changed. }

  ReleaseMemoryBlocks;
 end;


Procedure HyperTransFer(StartValue,StopValue:LongInt; Show:Boolean);

 Label

  OutOfHere;

 Var

  ReadValue         : Longint;   { Total amount to be Read }
  HyperReadCnt      : Longint;   { Counter to read in blocks of
                                   size of MaxBlockRead bytes }
  HyperReadBlock    : Longint;   { Amount of Main Blocks to read }
  HyperReadOver     : Longint;   { Amount not covering a full block }
  AddressStart      : Longint;   { Variables used during calculation }
  AddressStop       : Longint;   { of Address Intervals }
 WriteValue        : Longint;   { Total amount to be Write }
 HyperWriteCnt     : Longint;   { Counter to read in blocks of
                                  size of MaxBlockWrite bytes }
 HyperWriteBlock   : Longint;   { Amount of Main Blocks to write }
 HyperWriteOver    : Longint;   { Amount not covering a full block }


Begin
 GetMemoryBlocks;
 Increment:=1;
 ClearRam;
 GraphXcount:=0;
 HaltDraw:=False;
 ReadValue:= ABS(StopValue-StartValue); { Get readsize in bytes }
 HyperReadBlock:= ReadValue DIV MaxBlockRead;  { Amount of fixed Blocks }
 HyperReadOver:= ReadValue MOD MaxBlockRead;   { Left overs }

 WriteValue:= ABS(StopValue-StartValue);        { Get writesize in bytes }
 HyperWriteBlock:= WriteValue DIV MaxBlockWrite;{ Amount of fixed Blocks }
 HyperWriteOver:= WriteValue MOD MaxBlockWrite;   { Left overs }
 HyperReadcnt:=0;

 For HyperReadCnt:= 1 to HyperReadBlock do  {Read fixed sized blocks}
  Begin
   AddressStart:= ((HyperReadCnt-1) * MaxBlockRead)+StartValue;
   AddressStop:= (HyperReadCnt * MaxBlockRead);

   If DiskFlag then
    Begin
     Hyper_Read(AddressStart,AddressStop);
    end;

   If NOT(DiskFlag) then
    Begin
     Directtransfer(AddressStart,AddressStop);
    end;

   If NOT(SHOW) then
    Begin
     AddressStart:= (HyperReadCnt-1) * MaxBlockWrite;
     AddressStop:= (HyperReadCnt * MaxBlockWrite);
     Hyper_Write(AddressStart,AddressStop);
    end;

   If PCXConvert then
    Begin
     AddressStart:= (HyperReadCnt-1) * MaxBlockWrite;
     AddressStop:= (HyperReadCnt * MaxBlockWrite);
     DiskGraph(AddressStart,AddressStop);
    end;

   If Show AND NOT(PCXConvert) then
    Begin
     AddressStart:= (HyperReadCnt-1) * MaxBlockWrite;
     AddressStop:= (HyperReadCnt * MaxBlockWrite);
     DiskGraph(AddressStart,AddressStop);
     IF HaltDraw then goto OutOfhere
    end;
  end;

  INC(HyperReadCnt);                          { Read left overs }
  AddressStart:= ((HyperReadCnt-1) * MaxBlockRead)+StartValue;
  AddressStop:= AddressStart+HyperReadOver;
  If diskFlag then
   Begin
    Hyper_Read(AddressStart,AddressStop);
   end;

   If NOT(DiskFlag) then
    Begin
     Directtransfer(AddressStart,AddressStop);
    end;
```

```
     IF NOT(PCXConvert) then
      Begin
       AddressStart:= (HyperReadCnt-1) * MaxBlockWrite;
       AddressStop:= AddressStart+HyperWriteOver;
       Hyper_Write(AddressStart,AddressStop);
      end;

     If PCXConvert then
      Begin
       AddressStart:= (HyperReadCnt-1) * MaxBlockWrite;
       AddressStop:= (HyperReadCnt * MaxBlockWrite);
       DiskGraph(AddressStart,AddressStop);
      end;

     If Show AND NOT(PCXConvert) then
      begin
       AddressStart:= (HyperReadCnt-1) * MaxBlockWrite;
       AddressStop:= AddressStart+HyperWriteOver;
       DiskGraph(AddressStart,AddressStop);
       IF HaltDraw then goto OutofHere
      end;
      memBlock:=1;
OutofHere: ReleaseMemoryBlocks;
 end;


 Procedure ResetValues;
 { This procedure resets the values as used in the StartUp routine; }
  Begin
   TopStop:=0;   { Number of Frames to skip at the top of the Picture }
   SynchFlag:=False;   { Reset SynchFlag to NoSynch       }
   SynchCount:=0;      { restore Synchcounter to 0        }
   HCount:=0;          { Reset Horisontal synch counter }
   VCount:=0;          { reset Vertical synch counter     }
   HorisontalDEF:=710; { Pixels per Line                  }
   OLDEVENFLAG:=EVENFLAG;
   SynchLow:=15;
   VideoMarkMin:=30;   { Minimum Value Before a Hsynch is Possible }
   VideoMarkMax:=60;   { Maximum Duration of the HSYNCH }
   VideoLevel:=40;     { Switching point for video        }
   PrevSingle:=127;    { Set to maximum value             }
   VideoON:=False;     { Do not start with video Draw     }
   StartA:= 0;         { Startting Address Normally       }
   StopA:= 490000;     { Normal size of a FRAME           }
   Increment:= 1;      { Increment of Address Counters   }
   FileInput:='EMPTY.DAT';    { FileName for Starting }
   FileOutput:='OUTPUT.DAT'; { FileName for Output    }
   DiskFlag:=True;     { Source input from DISK for starting }
   DotFlag:=True;      { Start with a Dot Plot }
   ColorFlag:=False;   { Start in the B/W Mode }
   INVFlag:=False;     { No invertion on startup }
   GraphFlag:=False;   { Select graphics in PICTURE format display }
   FileReadFlag:=False; { Test if read from a file }
   BWLEVEL:=70;
   MaxVideoLevel:=127;
   MinVideoLevel:=127;
   MouseKeyPressed:=False;
   QuickViewFlag:=False;
   StretchFlag:=False;
   OneToOne:=False;
   PCXConvert:=False;
  end;

  Function KeytoStr(UPCASEFLAG:Boolean):String;

    Begin
    { Can Be Place outside to to check if Value in Range }

     KeyRead:=Readkey;
     SpecialKey:=True;

     IF Keyread <> #0 then
      Begin
      If KeyRead in SpecialSet then
       Begin
        Case Keyread of
         #8 : Begin KeytoStr:='BS'; end;    { BackSpace Key pressed }
         #9 : Begin KeytoStr:='TAB'; end;   { Tab key pressed       }
        #13 : Begin KeytoStr:='ENTER'; end; { Enter key pressed     }
        #27 : Begin KeytoStr:='ESC'; end;   { BackSpace key pressed }
       end; { end of case Statement }
       SpecialKey:=False;
       end
       Else
        Begin
         SpecialKey:=FALSE;
          IF UPCASEFLAG Then KeytoStr:=UPCASE(KeyRead)
                        Else KeytoStr:=KeyRead
        end;
     end;

    IF SpecialKey Then
     Begin
      KeyRead:=ReadKey;
      Case KeyRead of

    #71  : Begin KeytoStr:='HOME'; end;       { Home key         }
    #79  : Begin KeytoStr:='END'; end;        { END key          }
    #73  : Begin KeytoStr:='PGUP'; end;       { Page Up key      }
    #81  : Begin KeytoStr:='PGDN'; end;       { Page Down Key    }
    #82  : Begin KeytoStr:='INS'; end;        { INS key pressed  }
    #83  : Begin KeytoStr:='DEL'; end;        { DEL key pressed  }
    #15  : Begin KeytoStr:='-TAB'; end;       { Shift TAB key    }
    #72  : Begin KeytoStr:='UP'; end;         { Up arrow key     }
    #80  : Begin KeytoStr:='DOWN'; end;       { Down arrow key   }
    #75  : Begin KeytoStr:='LEFT'; end;       { Left arrow key   }
    #77  : Begin KeytoStr:='RIGHT'; end;      { Right arrow key  }
     #0  : Begin KeytoStr:='SPECIAL'; end;    { Special key      }

    #59  : Begin KeytoStr:='F1'; end;   {  F1 - Function Key  }
    #60  : Begin KeytoStr:='F2'; end;   {  F2 - Function Key  }
    #61  : Begin KeytoStr:='F3'; end;   {  F3 - Function Key  }
    #62  : Begin KeytoStr:='F4'; end;   {  F4 - Function Key  }
    #63  : Begin KeytoStr:='F5'; end;   {  F5 - Function Key  }
    #64  : Begin KeytoStr:='F6'; end;   {  F6 - Function Key  }
    #65  : Begin KeytoStr:='F7'; end;   {  F7 - Function Key  }
    #66  : Begin KeytoStr:='F8'; end;   {  F8 - Function Key  }
    #67  : Begin KeytoStr:='F9'; end;   {  F9 - Function Key  }
    #68  : Begin KeytoStr:='F10'; end . { F10 - Function Key  }
  end; { end of the Case statement }
  end;
 end;

 Function ReadString(CurrentX,CurrentY: Integer):String;
  Var
     Tempreadstring        : String;
     Showreadstring        : String;
     PrevShowString        : String;
     CurrentKey            : String;
     GraphCurser           : String;
     Currentchar           : Char;
     SizereadString        : Integer;
     READSTOP              : Boolean;

  Begin
   GraphCurser:='_';
   TempReadString:='';
   ShowReadString:=GraphCurser;
   PrevShowString:=GraphCurser;
   SetColor(Maxcolor);
   OuttextXY(CurrentX,CurrentY,PrevShowString);
   SetColor(0);
   ShowReadString:=TempReadString+GraphCurser;
   OuttextXY(CurrentX,CurrentY,ShowReadString);
   CurrentKey:='';
   READSTOP:=FALSE;
   Repeat
    CurrentChar:=Readkey;
    CurrentKey:=CurrentChar;
    Case CurrentChar of

 #8 : Begin
       Setcolor(MaxColor);
       OuttextXY(CurrentX,CurrentY,ShowreadString);
       SizereadString:=ORD(TempreadString[0]);
       TempreadString:=Copy(TempReadString,0,(SizeReadString-1));
       Setcolor(0);
       CurrentKey:='';
      end;
 #0 : Begin
       CurrentChar:=Readkey;
       CurrentKey :='';
      end;
 #15: Begin
       CurrentKey :='';
      end;
 #27: Begin
       CurrentKey :='';
      end;
 #13: Begin
       CurrentKey :='';
       READSTOP:=TRUE;
      end
      end; { end of the case statement }
      TempReadString:=TempReadString+CurrentKey;
      SetColor(Maxcolor);
      OuttextXY(CurrentX,CurrentY,PrevShowString);
      SetColor(0);
      ShowReadString:=TempReadString+GraphCurser;
      PrevShowString:=ShowReadString;
      OuttextXY(CurrentX,CurrentY,ShowReadString);
 Until readstop;
  readstring:=Tempreadstring;
end;

Function FileExist(ExistName:String):Boolean;

 Var
  FExist             : File;      { var to check if the file exist }
  ExistFlag          : Boolean;   { test flag }

 Begin
  Assign(FExist,ExistName);     { Assign filename to handler }
  {$I-}
  Reset(FExist);
  If IOResult <> 0 Then ExistFlag:=False else ExistFlag:=True;
  {$I+}
  If ExistFlag then begin Close(FExist) end;
  FileExist:=ExistFlag;
 end;

 Procedure RestoreGraphScreen;
  Begin
   SetViewPort(0,0,GetMaxX,GetMaxY,True);
   Xwork:=((GetmaxX +1) div 2)  - (XScreenSize div 2 );
   Ywork:=((GetmaxY +1) div 2)  - (YScreenSize div 2 );
   SetColor(0);
   SetLineStyle(Solidln,0,1); { Sets line thickness to 1, color = 0 }
   Rectangle((Xwork-1),(Ywork-1),(Xwork+XScreenSize+1),(YWork+YScreenSize+1));
   Setcolor(MaxColor);
   Rectangle(Xwork,Ywork,(Xwork+XScreenSize),YWork+YScreenSize);
   SetColor(0);
   SetViewPort(Xwork,Ywork,(Xwork+XScreenSize),YWork+YScreenSize,True);
   Xdisplay:=XscreenSize;
   Ydisplay:=YscreenSize;
   FunctionScreen(0,0,2);
   DescriptionBar(False);
  end;
```

```
Procedure ShowPICTURE;
 Begin
  If Diskflag then HyperTransfer(StartA,StopA,True);
  If Not(Diskflag) then GrabGraph(StartA,StopA,Increment);
  IF Not(PCXConvert) Then
  Begin
   ScreenSave('FRME.SCN',0,GetMaxY);
   KeyRead:=ReadKey;
   If KeyRead = #0 then Keyread:=ReadKey;
  end;
   InnerDisplay;
   FunctionScreen(0,0,2);    { Draws The function Selection screen }
   DeScriptionbar(False);
end;

Procedure PicScreen;
 Begin
  StartA:=0;
  StopA:=485000;
  TotalVShow:=560;
  ViewPicX:=TotalVShow;
  ViewPicY:=568;
  If (GetMaxX < TotalVShow) or NOT(OnetoOne) Then
  ViewPicX:=GetMaxX;
  If (GetMaxY < 568 ) or NOT(OnetoOne) Then
  ViewPicY:=GetMaxY;
  GrayScale:=MaxColor/(MaxVideoLevel-VideoLevel);
  Vcount:=0;
  SynchTest:=False;
  PrevSynchTest:=False;
  GraphPic:=1;
  FrontPorch:=77;
  VideoLength:=740;
  PrevVideoON:=True;
  XincPIC:=ViewPicX/(VideoLength-150);
  If OnetoOne then
   XincPic:=1;
  TopStop:=20;
  TopCount:=1;
  TotalVShow:=560;
  YincPic:=ViewPicY/TotalVshow;
  If OnetoOne then
  YincPic:=1;
  YposPIC:=Round(YincPic*VCount);
 end;


Procedure F1Select;
Var
  StringHeading               : String;
  TextworkX,TextworkY         : Integer;
  FileName                    : String;
  FileSave                    : Boolean;
  FileMode                    : Char;

 Begin
  FunctionScreen(1,12,1);
  FileSave:=True;
   IF FileReadFlag then
    Begin
     Close(FWrite);
   end;
  InstructionPut(18);
  Repeat
  KeyRead:=ReadKey;
  Until Keyread In ['1','2'];
  FileMode:=KeyRead;

 Repeat
  FileSave:=True;
  PutImage(17,255,PX,NormalPut);
  StatusBlock(True);
  StringHeading:='ENTER Destination FileName';
 SetColor(0); { Write in Black }
 SetTextStyle(DefaultFont,HorizDir,1);
 TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
 OUTTEXTXY((TextWorkX),258,StringHeading);
 TextWorkY:=258+(TextHeight(StringHeading)*2);
 FileName:=readString(25,TextWorkY);
 If FileExist(FileName) Then
   Begin
    PutImage(17,255,PX,NormalPut);
    StringHeading:='File exist Overwrite[Y/N]';
    SetColor(0); { Write in Black }
    SetTextStyle(DefaultFont,HorizDir,1);
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    TextWorkY:=258+(TextHeight(StringHeading)*2);
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    Repeat
     Repeat until keypressed;
     KeyRead:=UpCase(Readkey);
     If KeyRead = #0 then KeyRead:=ReadKey;
    Until KeyRead in ['Y','N'];
    If KeyRead = 'N' then FileSave:=False;
   end;
Until FileSave;

IF DiskFlag AND NOT(FileReadFlag) then
   Begin
    PutImage(17,255,PX,NormalPut);
    StringHeading:='Please SELECT a SOURCE FileName or';
    SetColor(0); { Write in Black }
    SetTextStyle(SmallFont,HorizDir,4);
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    TextWorkY:=258;
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    StringHeading:='change the INPUT source [F9]';
    SetTextStyle(DefaultFont,HorizDir,1);
    TextWorkY:=258+(TextHeight(StringHeading)*1);
    SetTextStyle(SmallFont,HorizDir,4);
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    StringHeading:='Press any Key to continue';
    SetTextStyle(DefaultFont,HorizDir,1);
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    TextWorkY:=258+(TextHeight(StringHeading)*4);
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    KeyRead:=Readkey;
    IF KeyRead = #0 then KeyRead:=Readkey;
    PutImage(17,255,PX,NormalPut);
   end
 else
  Begin
   IF  FileMode = '1' then
    Begin
     PCXConvert:=False;
     Assign(FWrite,FileName);
     Rewrite(FWrite,1);
     InstructionPut(10);
     Hypertransfer(StartA,StopA,False);
     Close(Fwrite);
     Assign(Fwrite,FileOutput);
     Rewrite(FWrite);
    end;
   IF Filemode = '2' then
    Begin
     PCXConvert:=True;
     Assign(PCX,'EVEN');
     REWRITE(PCX,1);
     Close(PCX);
     Assign(PCX,'ODD');
     Rewrite(PCX,1);
```

```
      Seek(PCX,0);
      GraphPIC:=1;
      PicScreen;
      PicScreenSetup;
      ShowPicture;
      Close(PCX);
      InstructionPut(10);
      Exec('PCX10.EXE',FileName);
      Assign(Fwrite,FileOutput);
      Rewrite(FWrite);
      PCXConvert:=False;
     end;
   end;
   InstructionPut(1);
   StatusBlock(True);
   FunctionScreen(1,12,2);

 end;

Procedure F2Select;
 { This procedure Select A File to work from }
 Var
  StringHeading                  : String;
  TextworkX,TextworkY            : Integer;
  FileName                       : String;

 Begin
  FunctionScreen(2,12,1);
  PutImage(17,255,PX,NormalPut);
  StatusBlock(True);
  IF FileReadFlag then
   Begin
    Close(F);
    Close(FWrite);
    FileReadFlag:=False;
   end;

   PutImage(17,255,PX,NormalPut);
   StringHeading:='ENTER Source FileName';
   SetColor(0); { Write in Black }
   SetTextStyle(DefaultFont,HorizDir,1);
   TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
   OUTTEXTXY((TextWorkX),258,StringHeading);
   TextWorkY:=258+(TextHeight(StringHeading)*2);
   FileName:=readString(25,TextWorkY);

  If NOT(FileExist(FileName)) Then
   Begin
    PutImage(17,255,PX,NormalPut);
    StringHeading:='File not found';
    SetColor(0); { Write in Black }
    SetTextStyle(DefaultFont,HorizDir,1);
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    TextWorkY:=258;
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    StringHeading:=FileName;
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    TextWorkY:=258+(TextHeight(StringHeading)*2);
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    StringHeading:='Press any Key to continue';
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    TextWorkY:=258+(TextHeight(StringHeading)*4);
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    KeyRead:=Readkey;
    IF KeyRead = #0 then KeyRead:=Readkey;
   end;

  If FileExist(FileName) Then
   Begin
    FileInput:=FileName;
```

```
       FileOutput:='TEMP.FRM';
       Assign(F,FileInput);
       Assign(FWrite,FileOutput);
       Reset(F,1);
       Rewrite(Fwrite,1);
       StartA:=0;
       StopA:=FileSize(F);
       DiskFlag:=True;
       FunctionScreen(9,12,2);
       FileReadFlag:=TRUE;
      end;

     InstructionPut(1);
     StatusBlock(True);
     FunctionScreen(2,12,2);

    end;

   Procedure F3Select;    { Select the Starting Address }
    Var
      StringHeading             : String;
      STRStartA                 : String;
      StartAReal                : Real;
      TextWorkX,TextWorkY       : Integer;
      STRVALERROR               : Integer;

    Begin
     FunctionScreen(3,12,1);
     PutImage(17,255,PX,NormalPut);
     StatusBlock(True);
     Repeat
      PutImage(17,255,PX,NormalPut);
      StringHeading:='ENTER new START Address';
      SetColor(0); { Write in Black }
      SetTextStyle(DefaultFont,HorizDir,1);
      TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
      OUTTEXTXY((TextWorkX),258,StringHeading);
      TextWorkY:=258+(TextHeight(StringHeading)*2);
      STRStartA:=readString(25,TextWorkY);
      Val(STRStartA,StartAReal,STRVALERROR);
      StartA:=Round(StartAReal);
     Until STRVALERROR = 0;
     InstructionPut(1);
     StatusBlock(True);
     FunctionScreen(3,12,2);
end;

   Procedure F4Select;
    Var
      StringHeading             : String;
      STRStopA                  : String;
      StopAReal                 : Real;
      TextWorkX,TextWorkY       : Integer;
      STRVALERROR               : Integer;

    Begin
     FunctionScreen(4,12,1);
     PutImage(17,255,PX,NormalPut);
     StatusBlock(True);
     Repeat
      PutImage(17,255,PX,NormalPut);
      StringHeading:='ENTER new STOP Address';
      SetColor(0); { Write in Black }
      SetTextStyle(DefaultFont,HorizDir,1);
      TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
      OUTTEXTXY((TextWorkX),258,StringHeading);
      TextWorkY:=258+(TextHeight(StringHeading)*2);
      STRStopA:=readString(25,TextWorkY);
      Val(STRStopA,StopAReal,STRVALERROR);
      StopA:=Round(StopAReal);
     Until STRVALERROR = 0;
      IF FileExist(FileInput) AND DiskFlag then
       Begin
     If StopA > FileSize(F) then StopA:=FileSize(F);
    end;
  InstructionPut(1);
  StatusBlock(True);
  FunctionScreen(4,12,2);
 end;

Procedure F5Select;
 Begin
  GraphFlag:=NOT(GraphFlag);
  FunctionScreen(5,12,0);
  StatusBlock(True);
 end;

Procedure F6Select;
Var
 MemSize           : Word;

 Begin
  QuickViewFlag:=False;
  FunctionScreen(6,12,1);
  IF GraphFlag then
   Begin
   QuickViewFlag:=True;
   InstructionPut(2);
   end
  else
   Begin
   IF FILEExist('FRME.SCN') then
    Begin
     QuickViewFlag:=True;
     ScreenSave('SAVE.SCN',0,GetMaxY);
     ScreenReread('FRME.SCN',0,GetMaxY);
     QuickViewFlag:=False;
     Repeat
     MouseMovFlag:= MouseMoveDetect;
     Until Keypressed or MouseKeyPressed;
     IF NOT(MouseKeyPressed) then
      Begin
       KeyRead:=Readkey;
       If Keyread = #0 then KeyRead:=Readkey;
      end;
     InnerDisplay;
     ScreenReRead('Save.SCN',0,GetMaxY);
     GraphFlag:=true;
     InstructionPut(1);
     StatusBlock(True);
    end;
  end;
  If Not(QuickViewFlag) then
  FunctionScreen(6,12,2);
  FunctionScreen(5,12,2);
 end;

Procedure F7Select;
 Begin
  FunctionScreen(7,12,1);
  StretchCount:=0;
  StretchFlag:=True;
  InstructionPut(4);
 end;



Procedure F8Select;
 Var
  StringHeading              : String;
  TextWorkY,TextWorkX        : Integer;

 Begin
 TopCount:=1;
 Vcount:=0;
```

```
FunctionScreen(8,12,1);
InstructionPut(9);
IF FileReadFlag or NOT(DiskFlag) then
 Begin
 IF GraphFlag then
  Begin           { Draw Graph }
   GraphScreenSetup;
   If DiskFlag then
    Begin         { Draws a graph from a file as selected }
     GraphPic:=0;   { select graph DisPlay }
     ScaleCalc(StartA,StopA,Dots,1);
     HyperTransfer(StartA,StopA,True);
    end
   Else
    Begin          { Draw graph Directly from frame grabber }
     GraphPic:=0;
     ScaleCalc(StartA,StopA,Dots,Increment);
     GrabGraph(StartA,StopA,Increment);
    end
  end
 ELSE             { Draw picture }
  Begin
   PicScreen;
   PICScreenSetup;
   If DiskFlag then
    Begin
     ShowPicture;
    end
   Else
    Begin
     ShowPicture;
    end
  End
 end
ELSE   { If no filename is selected }
 Begin
  IF DiskFlag then
   Begin
    PutImage(17,255,PX,NormalPut);
    StringHeading:='Please SELECT a FileName or';
    SetColor(0); { Write in Black }
    SetTextStyle(SmallFont,HorizDir,4);
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    TextWorkY:=258;
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    StringHeading:='change the INPUT source [F9]';
    SetTextStyle(DefaultFont,HorizDir,1);
    TextWorkY:=258+(TextHeight(StringHeading)*1);
    SetTextStyle(SmallFont,HorizDir,4);
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

    StringHeading:='Press any Key to continue';
    SetTextStyle(DefaultFont,HorizDir,1);
    TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
    TextWorkY:=258+(TextHeight(StringHeading)*4);
    OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);
    Repeat
    MouseMovFlag:= MouseMoveDetect;
    Until Keypressed or MouseKeyPressed;
    If NOT(MouseKeypressed) then
     Begin
      KeyRead:=Readkey;
      IF KeyRead = #0 then KeyRead:=Readkey;
     end;
  end;
 end;
 InstructionPut(1);
 StatusBlock(True);
 FunctionScreen(8,12,2);

end;
```

```
Procedure F9Select;
 Begin
  DISKFLAG:=NOT(DiskFLAG);
  FunctionScreen(9,12,0);
  StatusBlock(True);
 end;

Procedure F10Select;
 Begin
  FunctionScreen(10,12,1);
  If FileReadFlag then
   begin
    Close(F);
    Close(FWrite);
   { Release(PX); }
   end;
 end;

Procedure UPSelect;
 Begin
  CurserSkip:=1;
  Regs.AH:=2;
  Intr($16,Regs);
  IF (Regs.AL AND 64) = 64 then CurserSkip:=8;
  MouseMovFlag:=True;
  CurYPut:=CurYPut-(1*CurserSkip);
  If CurYPut < 0 then CurYPut:=0;
  Regs.CX:=CurXput*MousePixel;
  Regs.DX:=CurYput*MousePixel;
  Regs.AX:=4;
  Intr($33,Regs);
 end;

Procedure DOWNSelect;
 Begin
  CurserSkip:=1;
  Regs.AH:=2;
  Intr($16,Regs);
  IF (Regs.AL AND 64) = 64 then CurserSkip:=8;
  MouseMovFlag:=True;
  CurYPut:=CurYPut+(1*CurserSkip);
  If CurYPut > YscreenSize then CurYPut:=YscreenSize;
  Regs.CX:=CurXput*MousePixel;
  Regs.DX:=CurYput*MousePixel;
  Regs.AX:=4;
  Intr($33,Regs);
 end;

Procedure LEFTSelect;
 Begin
  CurserSkip:=1;
  Regs.AH:=2;
  Intr($16,Regs);
  IF (Regs.AL AND 64) = 64 then CurserSkip:=8;
  MouseMovFlag:=True;
  CurXPut:=CurXPut-(1*CurserSkip);
  If CurXPut < 0 then CurXPut:=0;
  Regs.CX:=CurXput*MousePixel;
  Regs.DX:=CurYput*MousePixel;
  Regs.AX:=4;
  Intr($33,Regs);
 end;

Procedure RIGHTSelect;
 Begin
  CurserSkip:=1;
  Regs.AH:=2;
  Intr($16,Regs);
  IF (Regs.AL AND 64) = 64 then CurserSkip:=8;
  MouseMovFlag:=True;
  CurXPut:=CurXPut+(1*CurserSkip);
  If CurXPut > XscreenSize then CurXPut:=XscreenSize;
  Regs.CX:=CurXput*MousePixel;
  Regs.DX:=CurYput*MousePixel;
  Regs.AX:=4;
  Intr($33,Regs);
 end;



Procedure PSelect;
 Begin
  OnetoOne:=Not(OnetoOne);
  FunctionScreen(12,1,1);
  Delay(1000);
  FunctionScreen(12,1,2);
  InstructionPut(1);
 end;

Procedure RSelect;
 Begin
   FunctionScreen(12,2,1);
   InstructionPut(6);
   Sound(4400);
   Delay(500);
   ResetValues;
   Sound(2200);
   Delay(500);
   Nosound;
   FunctionScreen(12,2,2);
   InstructionPut(1);
   Statusblock(True);
 end;

Procedure LSelect;
 Var
  TextWorkX          : Integer;
  TextWorkY          : Integer;
  StrValError        : Integer;
  StringHeading      : String;
  StrLine            : String;
  LineReal           : Real;
  LineInt            : LongInt;
  SampleLineLgth     : LongInt;


 Begin
  FunctionScreen(12,5,1);
  PutImage(17,255,PX,NormalPut);
  StatusBlock(True);
  Repeat
   PutImage(17,255,PX,NormalPut);
   StringHeading:='ENTER Line NUMBER to show';
   SetColor(0); { Write in Black }
   SetTextStyle(DefaultFont,HorizDir,1);
   TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
   OUTTEXTXY((TextWorkX),258,StringHeading);
   TextWorkY:=258+(TextHeight(StringHeading)*2);
   STRLine:=readString(25,TextWorkY);
   Val(STRLine,LineReal,STRVALERROR);
   LineInt:=Round(LineReal);
  Until STRVALERROR = 0;
  GraphPic:=0;
  SampleLineLgth:=768;
  IF LineInt < 4 then LineInt:=4;
  LineInt:=LineInt-3;
  StartA:=(SampleLineLgth div 2) + (SampleLineLgth*(LineInt-1))+57;
  StopA:=(SampleLineLgth div 2) + (SampleLineLgth*LineInt)+57;
  GraphScreenSetup;
  ScaleCalc(StartA,StopA,Dots,1);
  IF FileReadFlag or NOT(DiskFlag) then
   Begin
    IF diskflag then Hypertransfer(StartA,StopA,True) else
       GrabGraph(StartA,StopA,Increment);
   end
  ELSE  { If no filename is selected }
  Begin
   IF DiskFlag then
    Begin
     PutImage(17,255,PX,NormalPut);
     StringHeading:='Please SELECT a FileName or';
     SetColor(0); { Write in Black }
     SetTextStyle(SmallFont,HorizDir,4);
     TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
     TextWorkY:=258;
     OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

     StringHeading:='change the INPUT source [F9]';
     SetTextStyle(DefaultFont,HorizDir,1);
     TextWorkY:=258+(TextHeight(StringHeading)*1);
     SetTextStyle(SmallFont,HorizDir,4);
     TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
     OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);

     StringHeading:='Press any Key to continue';
     SetTextStyle(DefaultFont,HorizDir,1);
     TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
     TextWorkY:=258+(TextHeight(StringHeading)*4);
     OUTTEXTXY((TextWorkX),(TextWorkY),StringHeading);
     Repeat
     MouseMovFlag:= MouseMoveDetect;
     Until Keypressed or MouseKeyPressed;
     If NOT(MouseKeypressed) then
      Begin
       KeyRead:=Readkey;
       IF KeyRead = #0 then KeyRead:=Readkey;
      end;
   end;
  end;
  InstructionPut(1);
  StatusBlock(True);
  FunctionScreen(12,5,2);
 end;

Procedure MSelect;

Var

 KeyError : Integer;

 Begin
  FunctionScreen(12,4,1);
  Assign(VidFile,'Fr10.Set');
  Reset(VidFile);
  Instructionput(11);
  Repeat
   KeyRead:=ReadKey
  until KeyRead in ['0','1','2','3','4','5'];
  Val(KeyRead,VidData[1],KeyError);
  If VidData[1] = 5 Then
   Begin
    Instructionput(12);
    Repeat
     KeyRead:=ReadKey
    until KeyRead in ['0','1','2','3','4','5'];
    Val(KeyRead,VidData[2],KeyError);
    InstructionPut(VidData[2]+13);
    Repeat
     KeyRead:=ReadKey
    until KeyRead in ['0','1','2','3','4','5'];
    Val(KeyRead,VidData[3],KeyError);
   end

 Else
  Begin
   VidData[2]:=0;
   VidData[3]:=0;
  end;
```

```
   Write(VidFile,VidData);
   VideoModeFlag:=True;
   Instructionput(12);
   FunctionScreen(12,4,2);
   end;

  Procedure DSelect;
   Begin
   DotFlag:=NOT(DotFlag);
   FunctionScreen(12,3,0);
   StatusBlock(True);
   end;

  Procedure VSelect;
   Begin
   INVFLAG:=NOT(INVFLAG);
   FunctionScreen(12,7,0);
   StatusBlock(True);
   end;

  Procedure CSelect;
   Begin
   ColorFlag:=NOT(ColorFlag);
   FunctionScreen(12,8,0);
   StatusBlock(True);
   end;

  Procedure TSelect;
   Var
     StringHeading             : String;
     STRStopA                  : String;
     StopAReal                 : Real;
     TextWorkX,TextWorkY       : Integer;
     STRVALERROR               : Integer;

   Begin
   FunctionScreen(12,9,1);
    Repeat
     PutImage(17,255,PX,NormalPut);
     StringHeading:='ENTER new Contrast Value[0..127]';
     SetColor(0); { Write in Black }
     SetTextStyle(DefaultFont,HorizDir,1);
     TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
     OUTTEXTXY((TextWorkX),258,StringHeading);
     TextWorkY:=258+(TextHeight(StringHeading)*2);
     STRStopA:=readString(25,TextWorkY);
     Val(STRStopA,StopAReal,STRVALERROR);
     BWLevel:=Round(StopAReal);
    Until STRVALERROR = 0;
    InstructionPut(1);
    StatusBlock(True);
    FunctionScreen(12,9,2);
   end;

  Procedure Bselect;
Var
    TextWorkX          : Integer;
    TextWorkY          : Integer;
    StrValError        : Integer;
    StringHeading      : String;
    StrLine            : String;
    LineReal           : Real;
    LineInt            : LongInt;
    SampleLineLgth     : LongInt;


   Begin
    FunctionScreen(12,10,1);
    PutImage(17,255,PX,NormalPut);
    StatusBlock(True);
    Repeat
     PutImage(17,255,PX,NormalPut);
     StringHeading:='ENTER new Blacklevel [40..127]';
   SetColor(0); { Write in Black }
   SetTextStyle(DefaultFont,HorizDir,1);
   TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
   OUTTEXTXY((TextWorkX),258,StringHeading);
   TextWorkY:=258+(TextHeight(StringHeading)*2);
   STRLine:=readString(25,TextWorkY);
   Val(STRLine,LineReal,STRVALERROR);
   LineInt:=Round(LineReal);
  Until STRVALERROR = 0;
  VideoLevel:=LineInt;
  FunctionScreen(12,10,2);
  InstructionPut(1);
 end;

Procedure WSelect;
 Var
  TextWorkX            : Integer;
  TextWorkY            : Integer;
  StrValError          : Integer;
  StringHeading        : String;
  StrLine              : String;
  LineReal             : Real;
  LineInt              : LongInt;
  SampleLineLgth       : LongInt;


 Begin
  FunctionScreen(12,6,1);
  PutImage(17,255,PX,NormalPut);
  StatusBlock(True);
  Repeat
   PutImage(17,255,PX,NormalPut);
   StringHeading:='ENTER new Whitelevel [40..127]';
   SetColor(0); { Write in Black }
   SetTextStyle(DefaultFont,HorizDir,1);
   TextWorkX:=((300-TextWidth(StringHeading)) DIV 2) + 10;
   OUTTEXTXY((TextWorkX),258,StringHeading);
   TextWorkY:=258+(TextHeight(StringHeading)*2);
   STRLine:=readString(25,TextWorkY);
   Val(STRLine,LineReal,STRVALERROR);
   LineInt:=Round(LineReal);
  Until STRVALERROR = 0;
  MaxVideoLevel:=LineInt;
  FunctionScreen(12,6,2);
  InstructionPut(1);
 end;

Procedure QuickGraphView;

 Var
   MemSize              : Word;
   QStartA              : LongInt;
   QStopA               : LongInt;

 Begin
    StretchFlag:=False;
    StringKey:='';
    ScreenSave('Save.SCN',0,GetMaxY);
    IF FileReadFlag or NOT(DiskFlag) then
    Begin
     IF GraphFlag then
      Begin           { Draw Graph }
       GraphScreenSetup;
       If DiskFlag then
        Begin         { Draws a graph from a file as selected }
         GraphPic:=0;   { select graph DisPlay }
         QStartA:=DottoADR(CurXPut)+StartA;
         QStopA:=QStartA+3000;
         If QStopA > StopA then QStopA:=StopA;
         ScaleCalc(QStartA,QStopA,Dots,1);
         HyperTransfer(QStartA,QStopA,True);
         ScaleCalc(StartA,StopA,Dots,Increment);
```

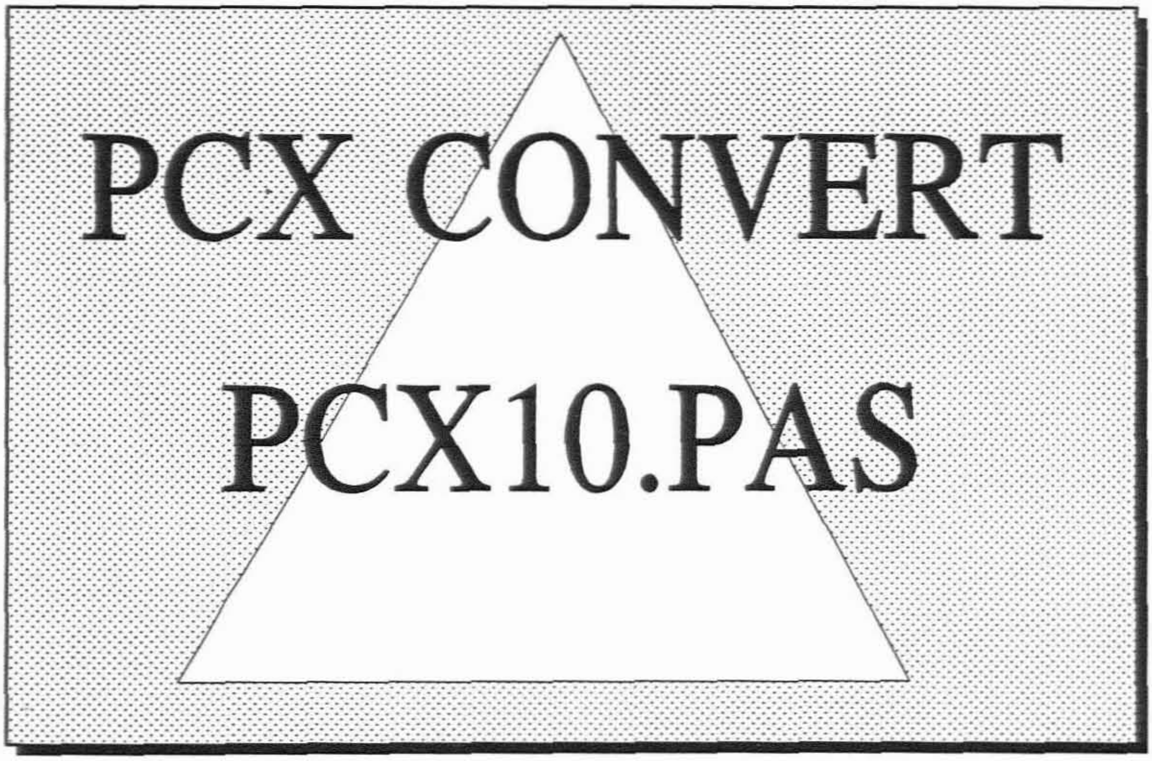
```
          end
         Else
          Begin         { Draw graph Directly from frame grabber }
           GraphPic:=0;
           QStartA:=DottoADR(CurXPut)+StartA;
           QStopA:=QStartA+3000+StartA;
           If QStopA > StopA then QStopA:=StopA;
           ScaleCalc(QStartA,QStopA,Dots,Increment);
           GrabGraph(QStartA,QStopA,Increment);
           ScaleCalc(StartA,StopA,Dots,Increment);
          end
        end
       end;

      InstructionPut(3);
      Repeat
       MouseMovFlag:= MouseMoveDetect;
      Until Keypressed or MouseKeyPressed;
       IF NOT(MouseKeyPressed) then
        Begin
         KeyRead:=Readkey;
         If Keyread = #0 then KeyRead:=Readkey;
        end;

       ScreenReRead('SAVE.SCN',0,GetMaxY);
       FunctionScreen(6,12,2);
       QuickViewFlag:=False;
     end;

Procedure GraphStretch;

   Begin
     Stringkey:='';
     INC(StretchCount);
     IF StretchCount=1 then
      Begin
       StartAVar:=StartA;
       StartA:=DottoADR(CurXPut)+StartA;
       InstructionPut(5);
      end;
     IF StretchCount=2 then
      Begin
       StopA:=DottoADR(CurXPut)+StartAVar;
       IF StopA < StartA then
        Begin
         StartAVar:= StopA;
         StopA:=StartA;
         StartA:=StartAVar;
        end;
       IF (StopA - StartA) = 0 then StartA:=StartA-1;
       IF FileReadFlag or NOT(DiskFlag) then
       Begin
        IF GraphFlag then
         Begin          { Draw Graph }
         GraphScreenSetup;
          If DiskFlag then
           Begin         { Draws a graph from a file as selected }
            GraphPic:=0;   { select graph DisPlay }
            ScaleCalc(StartA,StopA,Dots,1);
            HyperTransfer(StartA,StopA,True);
           end
          Else
           Begin         { Draw graph Directly from frame grabber }
            GraphPic:=0;
            ScaleCalc(StartA,StopA,Dots,Increment);
            GrabGraph(StartA,StopA,Increment);
           end
         end
        end;
        StatusBlock(True);
        StretchCount:=0;
        StretchFlag:=False;
      end;
   end;

Procedure MouseENTER;
 Var MemSize          : Word;
     QStartA          : LongInt;
     QStopA           : Longint;

 Begin
  IF quickViewFlag then
   Begin
    QuickGraphView;
    FunctionScreen(6,12,2);
   end;
  IF StretchFlag then
   Begin
    GraphStretch;
    IF NOT(StretchFlag) then FunctionScreen(7,12,2);
   end;
  IF NOT(QuickViewFlag) AND NOT(StretchFlag) then
   Begin
   InstructionPut(8);
   end;
 end;

Procedure MouseSelect;
Begin
   StringKey:='';
   MouseKeyPressed:=True;
 If (CurYput > Yblock1[0]) and (CurYput < Yblock1[1]) Then
  Begin
   If (CurXput > F1Mouse[0]) and (CurXput < F1Mouse[1]) then StringKey:='F1';
   If (CurXput > F2Mouse[0]) and (CurXput < F2Mouse[1]) then StringKey:='F2';
   If (CurXput > F3Mouse[0]) and (CurXput < F3Mouse[1]) then StringKey:='F3';
   If (CurXput > F4Mouse[0]) and (CurXput < F4Mouse[1]) then StringKey:='F4';
   If (CurXput > F5Mouse[0]) and (CurXput < F5Mouse[1]) then StringKey:='F5';
   If (CurXput > F6Mouse[0]) and (CurXput < F6Mouse[1]) then StringKey:='F6';
   If (CurXput > F7Mouse[0]) and (CurXput < F7Mouse[1]) then StringKey:='F7';
   If (CurXput > F8Mouse[0]) and (CurXput < F8Mouse[1]) then StringKey:='F8';
   If (CurXput > F9Mouse[0]) and (CurXput < F9Mouse[1]) then StringKey:='F9';
   If (CurXput > F10Mouse[0]) and (CurXput < F10Mouse[1]) then StringKey:='F10';
 end;

 If (CurYput > Yblock2[0]) and (CurYput < Yblock2[1]) Then
  Begin
   If (CurXput > PMouse[0]) and (CurXput < PMouse[1]) then StringKey:='P';
   If (CurXput > RMouse[0]) and (CurXput < RMouse[1]) then StringKey:='R';
   If (CurXput > DMouse[0]) and (CurXput < DMouse[1]) then StringKey:='D';
   If (CurXput > MMouse[0]) and (CurXput < MMouse[1]) then StringKey:='M';
   If (CurXput > LMouse[0]) and (CurXput < LMouse[1]) then StringKey:='L';
   If (CurXput > WMouse[0]) and (CurXput < WMouse[1]) then StringKey:='W';
   If (CurXput > VMouse[0]) and (CurXput < VMouse[1]) then StringKey:='V';
   If (CurXput > CMouse[0]) and (CurXput < CMouse[1]) then StringKey:='C';
   If (CurXput > TMouse[0]) and (CurXput < TMouse[1]) then StringKey:='T';
   If (CurXput > BMouse[0]) and (CurXput < BMouse[1]) then StringKey:='B';
 end;

 If (CurYput > Yblock3[0]) and (CurYput < Yblock3[1]) Then
 Begin
  Sound(800);
  Delay(500);
  Nosound;
  StringKey:='MouseENTER';
 end;
end;

Procedure ENTERSelect; Forward;


Procedure MenuSelection;


 Var
     KeyValid             : Boolean;  { Check if option exists }
  Begin
   IF NOT(MouseKeypressed) then
   StringKey:=KeytoStr(True);  { Get a Key from the keyboard  }
   KeyValid:=FALSE;  { Preset value of KeyValid }

   IF Stringkey = 'F1' Then Begin KeyValid:=True; F1Select; End;
   IF Stringkey = 'F2' Then Begin KeyValid:=True; F2Select; End;
   IF Stringkey = 'F3' Then Begin KeyValid:=True; F3Select; End;
   IF Stringkey = 'F4' Then Begin KeyValid:=True; F4Select; End;
   IF Stringkey = 'F5' Then Begin KeyValid:=True; F5Select; End;
   IF Stringkey = 'F6' Then Begin KeyValid:=True; F6Select; End;
   IF Stringkey = 'F7' Then Begin KeyValid:=True; F7Select; End;
   IF Stringkey = 'F8' Then Begin KeyValid:=True; F8Select; End;
   IF Stringkey = 'F9' Then Begin KeyValid:=True; F9Select; End;
   IF Stringkey = 'F10' Then Begin KeyValid:=True; F10Select; End;

   IF Stringkey = 'UP'    Then Begin KeyValid:=True; UPSelect; End;
   IF Stringkey = 'DOWN'  Then Begin KeyValid:=True; DOWNSelect; End;
   IF Stringkey = 'LEFT'  Then Begin KeyValid:=True; LEFTSelect; End;
   IF Stringkey = 'RIGHT' Then Begin KeyValid:=True; RIGHTSelect; End;

   IF Stringkey = 'P' Then Begin KeyValid:=True; PSelect; End;
   IF Stringkey = 'R' Then Begin KeyValid:=True; RSelect; End;
   IF Stringkey = 'L' Then Begin KeyValid:=True; LSelect; End;
   IF Stringkey = 'M' Then Begin KeyValid:=True; MSelect; End;
   IF Stringkey = 'D' Then Begin KeyValid:=True; DSelect; End;
   IF Stringkey = 'V' Then Begin KeyValid:=True; VSelect; End;
   IF Stringkey = 'C' Then Begin KeyValid:=True; CSelect; End;
   IF Stringkey = 'T' Then Begin KeyValid:=True; TSelect; End;
   IF Stringkey = 'W' Then Begin KeyValid:=True; WSelect; End;
   IF Stringkey = 'B' Then Begin KeyValid:=True; Bselect; End;

   IF Stringkey = 'ENTER' Then Begin KeyValid:=True; ENTERSelect; End;
   IF Stringkey = 'MouseENTER'  Then Begin KeyValid:=True; MouseENTER; End;

   IF NOT(KeyValid) Then Begin StringKey:='FAULT KEY' end;

 end;

Procedure ENTERSelect;
  Begin
   MouseSelect;
   MenuSelection;
   MouseKeyPressed:=False;
  end;



Procedure RollText;

Type

DataType      = Array[0..168] of Char;

Var

 ShowXPos              : Integer;
 ShowYPos              : Integer;
 ShowXwork             : Integer;
 ShowYWork             : Integer;
 YScaleShow            : Real;
 XShowSteps            : Integer;
 YMaxShowSize          : Integer;
 YShowSize             : Integer;
 WalkCounter           : Integer;
 ShowCounter           : Integer;
 WalkInc               : Integer;
 StartOffSet           : Integer;
 BigShowArea           : Pointer;
 BackGNDArea           : Pointer;
 MemSize               : Word;
 XShow1,XShow2          : Integer;
 YShow1,YShow2          : Integer;
 ShowF                  : File of DataType;
 ShowArray              : DataType;
 CharPixel              : Char;
 ShadowMain             : Integer;
 ViewPort               : ViewPortType;
 SoundMax               : Integer;
 SoundMin               : Integer;
 SoundScale             : Real;
 SoundValue             : Integer;

Begin
   SoundMax:=40;
   SoundMin:=10;
   GetViewSettings(ViewPort);
   SetViewPort(0,0,GetMaxX,GetMaxY,True);
   ShadowMain:=5;
   Xwork:=((GetmaxX +1) div 2)  - (600 div 2 );
   Ywork:=((GetmaxY +1) div 2)  - (180 div 2 );
   Xdisplay:=600;
   YDisplay:=180;
   SetFillStyle(0,0);
   Bar((Xwork+ShadowMain),(Ywork+ShadowMain),(Xwork+XDisplay+ShadowMain),(YWork+YDisplay+ShadowMain));
   SetFillStyle(1,MaxColor);
   SetColor(0);
   SetLineStyle(Solidln,0,1);
   Rectangle((Xwork-1),(Ywork-1),(Xwork+XDisplay+1),(YWork+YDisplay+1));
   Setcolor(MaxColor);
   Bar(Xwork,Ywork,(Xwork+XDisplay),(YWork+YDisplay));
   SetColor(0);
   SetFillStyle(SolidFill,0);
   Circle(Xwork+20,YWork+20,10);
   Circle((XWork+XDisplay-20),(Ywork+20),10);
   FloodFill(Xwork+20,YWork+20,0);
   FloodFill((XWork+XDisplay-20),(Ywork+20),0);
   SetFillStyle(SolidFill,MaxColor);
   SetColor(MaxColor);
   Circle(Xwork+18,YWork+18,10);
   Circle((XWork+XDisplay-18),(Ywork+18),10);
   FloodFill(Xwork+18,Ywork+18,MaxColor);
   FloodFill(Xwork+XDisplay-18,Ywork+18,MaxColor);
   SetColor(0);
   Circle(Xwork+18,YWork+18,10);
   Circle((XWork+XDisplay-18),(Ywork+18),10);

   YMaxShowSize:=50;
   YShowSize:=YmaxShowSize*2;
   StartOffSet:=0;
   WalkInc:=3;
   SetColor(0);
   SetFillStyle(SolidFill,0);
   SettextStyle(TriplexFont,Horizdir,4);
   XShowSteps:=GetMaxY div 2;
   ShowXWork:=GetMaxX div 2;
   ShowYWork:=GetMaxY div 2;
   SetTextJustify(CenterText,BottomText);
   SettextStyle(TriplexFont,Horizdir,7);
   SetColor(0);
   OUTTEXTXY(ShowXWork,ShowYWork,'FRAME-GRABBER');

   WalkCounter:=XWork;
   ShowYWork:=ShowYwork+5+1;
   XShow1:=WalkCounter;
   XShow2:=WalkCounter+WalkInc;
   YShow1:=ShowYWork;
   YShow2:=ShowYWork+77;
   MemSize:=ImageSize(XShow1,YShow1,XShow2,YShow2);
   GetMem(BackGNDArea,MemSize);
   GetImage(XShow1,YShow1,XShow2,YShow2,BackGNDArea^);
   Assign(ShowF,'Plane.INC');
   Reset(ShowF);
```

```
  For ShowCounter:=0 to 76 do
   begin
    Read(ShowF,ShowArray);
    For WalkCounter:= 0 to 166 do
     begin
      ShowXPos:=WalkCounter+Xwork;
      ShowYPos:=ShowCounter+ShowYWork;
      CharPixel:=ShowArray[WalkCounter];
      IF CharPixel = '1' then PutPixel(ShowXpos,ShowYPos,MaxColor)
                         else PutPixel(ShowXpos,ShowYPos,0);
     end;
    end;
    Close(ShowF);
    SetTextJustify(LeftText,topText);
    SettextStyle(DefaultFont,Horizdir,0);
    SetColor(0);
    OUTTEXTXY(XWork+36,(ShowYWork),'by N.P.Stodart');
     WalkCounter:=Xwork;
     XShow2:=WalkCounter+166;
     MemSize:=ImageSize(XShow1,YShow1,XShow2,YShow2);
     GetMem(BigShowArea,MemSize);
     GetImage(XShow1,YShow1,XShow2,YShow2,BigShowArea^);
     SoundScale:=(SoundMax-SoundMin)/(GetMaxX-(Xwork+170));

  Repeat
   XShow1:=WalkCounter;
   YShow1:=ShowYWork;
   PutImage(XShow1,YShow1,BackGNDArea^,NormalPut);
   WalkCounter:=WalkCounter+WalkInC;
   XShow1:=WalkCounter;
   YShow1:=ShowYWork;
   SoundValue:=SoundMax-Round(SoundScale*WalkCounter);
   Sound(SoundValue);
   PutImage(XShow1,YShow1,BigShowArea^,NormalPut);
  Until  (WalkCounter >= (GetMaxX-(Xwork+170))) OR KEYPRESSED;
  SetTextJustify(LeftText,topText);
  NoSound;
  IF WalkCounter > 170 then
  Begin
   SettextStyle(DefaultFont,Horizdir,0);
   SetColor(0);
   OUTTEXTXY(XWork+35,(ShowYWork),'(C) Copyright 1993');
  end;
  KeyRead:=ReadKey;
  If KeyRead =#0 then KeyRead:=ReadKey;
  SetViewPort(ViewPort.X1,ViewPort.Y1,ViewPort.X2,ViewPort.Y2,ViewPort.Clip);

  XDisplay:=XscreenSize;
  YDisplay:=YScreenSize;
 end;

Procedure GetVideoMode;

   Begin
    If FileExist('FR10.SET') then
     Begin
      Assign(VidFile,'FR10.SET');
      Reset(VidFile);
      Read(VidFile,VidData);
      Close(VidFile);
     end
    else
     Begin
      Assign(VidFile,'FR10.SET');
      Rewrite(VidFile);
      VidData[1]:=0;
      VidData[2]:=0;
      VidData[3]:=0;
      Write(VidFile,VidData);
      Close(VidFile);
     end;
    end;
  Procedure Main;

  Label VideoJump;

  Var
    GrphMode      : Integer;
    DisplayString : String;

   Begin
VideoJump: ClrScr;
           ResetValues;
           MousePixel:=7;
           StringKey:='';
           FileOutput:='FRAME.TMP';
           GetVideoMode;
           SetupGraphics;
           VideoModeflag:=False;
           MaxColor:=GetMaxColor;
           If MaxColor > 63 then MaxColor:=63;
           If MaxColor > 1 then ColorFlag:=True;
           MouseCurser;
           If FileExist('FRAME.SCN') then
           Begin
            ScreenReRead('FRAME.SCN',0,GetMaxY);
           end
           else
            ScreenSetup;
           RollText;

    If FileExist('FRAME.SCN') then
     Begin
      ScreenReRead('FRAME.SCN',0,GetMaxY);
     end
    else
    ScreenSetup;
    InnerDisplay;
    FunctionScreen(0,0,2);    { Draws The function Selection screen }
    DescriptionBar(True);
    StatusBlock(True);
    InstructionPut(1);
    HideCurser:=True;
    mouse;
    ShowMouse;
    MouseMovFlag:=False;
    MouseKeyPressed:=False;
   REPEAT
    Repeat
    StringKey:='';
    Until (MouseMoveDetect OR Keypressed) OR MouseKeyPressed;
    Mark(HeapPointer);
    IF MouseKeyPressed then MouseSelect;

    IF MouseMovFlag  then
     Begin
      PutCurser;
     end;
     IF NOT(MouseMovFlag) or MouseKeyPressed then
      Begin
       HideMouse;
       MenuSelection;
       ShowMouse;
      end;
      If VideoModeFlag then
       Begin
        CloseGraph;
        Goto VideoJump;
       end;
      Release(HeapPointer);
    UNTIL StringKey = 'F10' ;
  CloseGraph;

 end;
```

```
Begin
 Main;
end.
```

APPENDIX C

PCX Conversion program (PCX10.PAS) Program listing



A full listing of the PCX10.PAS program follows on pages C-2. It is inevitable that this program will be changed or added to for future developments in the frame grabber unit.

APPENDIX C

```
Program Grab_PCX;

{$M 51200,51200,473087}

  { This program converts a picture as Grabbed by the FR30.PAS program
                 and stores  it in the PCX-Format

                                   Written By N.P.Stodart        }


 Uses Crt,Dos;


  Type PCX_Header = Record
                     Maker, Version, Code, BPP          : Char;
                     X1, Y1, X2, Y2, Hres, Vres         : Integer;
                     Triplet                            : Array[1..16,1..3] of Char;
                     VMode,Nplanes                      : Char;
                     BPL                                : Integer;
                     UnusedSpace                        : Array[1..60] of Char;
                    end;

       DataLine   = Array[1..800] of byte;

  Const

   PCXSTOP      = 16384;         { Maximum Blocksize before the PCX file is written }


  Var
   PCXF                  : File of PCX_Header; { File Handler for the Header }
   PCXI                  : File;  { A file in the byte format for the storing of }
                                  { the image as a byte-map }
   HeaderData            : PCX_Header; { A record to define the current header to
                                       { be used }
   VideoMode             : Integer;   { Current video mode in use }
   Block_X1,Block_X2     : Integer;   { X - Coordinates of a captured block }
   Block_Y1,Block_Y2     : Integer;   { Y - Coordinates of a captured block }
   GraphDriver           : Integer;   { Variable to select video mode }
   ReadName              : String;    { Filename of the  file to besaved }
   PixelperByte          : Integer;   { Pixels for a byte in the screen memory }
   BPLArray              : DataLine;  { Array of 800 PCX pixels long }
   PCXArray              : Array[1..16400] of Byte;
   BPLCounter            : Integer;   { Counts bits per PCX Line }
   XCounter              : Integer;   { X Position counter in captured Data }
   YCounter              : Integer;   { Y Position counter in captured Data }
   XScrnScale            : Real;
   YScrnScale            : Real;
   VGAPalet              : Array[0..255,0..2] of Byte; { Array of the VGA Palet }



   Procedure PCXHeader(FileName : String);

    Var
     HeaderCount         : Integer;  { Counter use to setup data }

    Begin
     With HeaderData do
      Begin
       Maker:=#10;
       Version:=#5;
       Code:=#1;
       BPP:=CHR(LO(8 DIV PixelperByte));   { Calculate the Bytes per PCX line}
       X1:=Block_X1;         { Left X coordinate for PCX placement  }
       X2:=Block_X2;         { Right X coordinate for PCX placement }
       Y1:=Block_Y1;         { Upper Y coordinate for PCX placement }
       Y2:=Block_Y2;         { Lower Y coordinate for PCX placement }
       HRes:= ABS(X2-X1)+1; { Number of Horistontal Pixels }
    VRes:= ABS(Y2-Y1)+1; { Number of Vertical Pixels }
    VMode:=#0;           { Video Mode use 0 for VGA }
    NPlanes:=#1;         { Number of planes used 0 for VGA}
    BPL:=(ABS(X2-X1)+1) DIV PixelperByte;
    HeaderCount:=(ABS(X2-X1)+1) MOD PixelperByte;
    If HeaderCount > 0 then INC(BPL);
    For HeaderCount:= 1 to 16 do          { Set EGA Pallet to 0 }
     Begin                                { Not used for VGA PCX files}
      Triplet[HeaderCount,1]:=#0;
      Triplet[HeaderCount,2]:=#0;
      Triplet[HeaderCount,3]:=#0;
     end;
    For HeaderCount:= 1 to 60 do       { Blank spacing to identify}
     Begin                             { the starting position of the }
      UnusedSpace[HeaderCount]:=#0;    { data section}
     end;
   end;
   Assign(PCXF,FileName);
   Rewrite(PCXF);
   Write(PCXF,HeaderData);             { Write PCX Header as specified }
   Close(PCXF);
   For HeaderCount:= 1 to 8000 do      { Clear PCX variables before the }
    Begin                              { PCX Convertion is initialized  }
     BPLArray[HeaderCount]:=128;
     PCXArray[HeaderCount]:=128;
    end;
 end;


Procedure PCXPalette(FileName:String); { - VGA only }
 Var
  PaletCounter             : Integer;
  RGB                      : Array[0..2] of Byte;
  PCXPalet                 : File;
  SizePCXfile              : LongInt;
  NumWrite                 : Word;
  Fill255                  : Integer;
  SpecialPAL               : Integer;
  PalPos                   : Integer;

 Begin
  For Fill255 := 0 to 3 do             { Writes the GrayScale Palet   }
   Begin                               { At the end of the converted }
    For PaletCounter:=0 to 63 do       { PCX file }
     Begin
      PalPos:=(Fill255*64)+PaletCounter;
      SpecialPal:=PalPos DIV 4;
      VGAPalet[PalPos,0]:=SpecialPal;
      VGAPalet[PalPos,1]:=SpecialPal;
      VGAPalet[PalPos,2]:=SpecialPal;
     end;
   end;
   Assign(PCXPalet,FileName);
   Reset(PCXPalet,1);
   SizePCXFile:=FileSize(PCXPalet);
   Seek(PCXPalet,SizePCXFile);
   BlockWrite(PCXPalet,VGAPalet,768,NumWrite);
   Close(PCXPalet);
 end;


Procedure GetPCX(FileName :String);    { Convert to PCX }

 Const

  MaxLinePCX         = 30;

 Var
  RORByte            : Byte;
 PCounter           : Integer;
 PCXBYTE            : Byte;
 PrevPCXByte        : Byte;
 PCXCNT             : Byte;
 CurPix             : Byte;
 PCXCount           : Integer;
 RepeatCnt          : Boolean;
 BiggerPCX          : Boolean;
 SavePCX            : Boolean;
 ODDSTOP            : Boolean;
 NumWrite           : Word;
 NumRead            : Word;
 SWAPFILE           : Boolean;
 ODD                : File;
 EVEN               : File;
 ODDSize            : Longint;
 EVENSize           : Longint;
 ODDData            : Array[1..MaxLinePCX] of DataLine;
 EVENData           : Array[1..MaxLinePCX] of DataLine;
 BlockFull          : Integer;
 BlockOver          : Integer;
 BlockFullCnt       : Integer;
 BlockOverCnt       : Integer;
 BlockSizePCX       : LongInt;
 ArrayCount         : Integer;
 PCXBlockRead       : Boolean;


Begin
 Assign(ODD,'ODD');
 Reset(ODD,1);
 Assign(EVEN,'EVEN');
 Reset(EVEN,1);
 Seek(ODD,0);
 Seek(EVEN,0);
 ODDSize:=FileSize(ODD);
 EVENSize:=FileSize(EVEN);
 BlockFull:=EVENSize DIV (MaxLinePCX*800);
 BlockOver:=EVENSize MOD (MaxLinePCX*800);
 BlockFullCnt:=0;
 RepeatCnt:=False;
 BiggerPCX:=False;
 SavePCX:=False;
 ODDSTOP:=False;
 PCXBlockread:=True;
 RORByte:=1;
 XCounter:=Block_X1;
 YCounter:=Block_Y1;
 PCXCount:=0;
 PCXCNT:=$C0-1;
 SWAPFILE:=True;
 ASSIGN(PCXI,FileName);
 RESET(PCXI,1);
 Seek(PCXI,$80);
 Repeat
 For BPLCounter:= 1 to HeaderData.BPL do
  Begin
   IF PCXBlockRead then
    Begin
     If BlockFullCNT > BlockFull then
      Begin
       BlockSizePCX:=800*BlockOver;
       BLOCKREAD(ODD,ODDData,BlockSizePCX,Numread);
       BLOCKREAD(EVEN,EVENData,BlockSizePCX,Numread);
       ArrayCount:=1;
       BPLArray:=EVENData[ArrayCount];
       PCXBlockRead:=False;
      end
     ELSE
      Begin
     INC(BlockFullCNT);
     BlockSizePCX:=800*MaxLinePCX;
     BLOCKREAD(ODD,ODDData,BlockSizePCX,Numread);
     BLOCKREAD(EVEN,EVENData,BlockSizePCX,Numread);
     ArrayCount:=1;
     BPLArray:=EVENData[ArrayCount];
     PCXBlockRead:=False;
    end;
   end;
 PCXByte:=0;
 CurPix:=BPLArray[XCounter];
 PCXByte:=CurPix;

 IF BPLCounter = 1 then Begin
                         PrevPCXByte:=PCXByte;
                         PCXCNT:=$BF;
                        end;
 INC(XCounter);

INC(PCXCNT);
IF PCXByte <> PrevPCXByte then
 Begin
  SavePCX:=True;
  IF PCXCNT  > $C1 Then RepeatCNT:=true
                   Else RepeatCNT:=False;
  IF (PCXCNT = $C1) AND (PrevPCXByte >= $C0)
                   Then Begin
                         BiggerPCX:=True;
                         RepeatCNT:=True;
                        end
                   Else BiggerPCX:=False;
 end;

IF PCXCNT >= $FF Then Begin
                       RepeatCNT:=True;
                       SavePCX:=True;
                      end;
IF (PCXCNT >= $C1) AND (XCounter > HeaderData.X2) Then
                      Begin
                       SavePCX:=True;
                       IF PCXCNT > $C0 Then
                                        Begin
                                         RepeatCNT:=True;
                                        end
                                       Else RepeatCNT:=False;
                       IF (PCXCNT = $C0) AND (PrevPCXByte >= $C0)
                                       Then Begin
                                             BiggerPCX:=True;
                                             RepeatCNT:=True;
                                            end
                                       Else BiggerPCX:=False;
                       ODDSTOP:=FALSE;
                       IF PCXByte = PrevPCXByte then INC(PCXCNT);
                       IF PCXByte <> PrevPCXByte then ODDSTOP:=True;
                      end;

IF XCounter > HeaderData.X2 then
                      Begin
                       IF Swapfile then
                        Begin
                         BPLArray:=ODDData[ArrayCount];
                        end
                       ELSE
                        Begin
                         INC(ArrayCount);
                         IF ArrayCount > MaxLinePCX then
                          Begin
                           ArrayCount:=1;
                           PCXBlockRead:=True;
                          end;
```

```
                             BPLArray:=EVENData[ArrayCount];
                            end;
                           Swapfile:=NOT(SwapFile);
                           INC(YCounter);
                           XCounter:=Block_X1;
                          end;

    IF SavePCX Then
     Begin
      SavePCX:=False;
      IF RepeatCNT OR BiggerPCX then
       Begin
        RepeatCNT:=False;
        BiggerPCX:=False;
        INC(PCXCount);
        PCXArray[PCXCount]:=PCXCNT;
        INC(PCXCount);
        PCXArray[PCXCount]:=PrevPCXByte;
        PCXCNT:=$C0;
       end
      Else
       Begin
        INC(PCXCount);
        PCXArray[PCXCount]:=PrevPCXByte;
        PCXCNT:=$C0;
       end;
      IF ODDSTOP Then
       Begin
        IF PCXByte >= $C0 then
         Begin
          INC(PCXCount);
          PCXArray[PCXCount]:=PCXCNT+1;
          INC(PCXCount);
          PCXArray[PCXCount]:=PCXByte;
          PCXCNT:=$C0
         end
        ELSE
         Begin
          INC(PCXCount);
          PCXArray[PCXCount]:=PCXByte;
          PCXCNT:=$C0;
         end;
       end;
      IF PCXCount >= PCXSTOP then
       Begin
        BLOCKWRITE(PCXI,PCXArray,PCXCount,NumWrite);
        PCXCount:=0;
       end;
     end;
     PrevPCXByte:=PCXByte;
     ODDSTOP:=False;
  end;                                  { End of BPLCounter }
 Until YCounter > HeaderData.Y2;

 BLOCKWRITE(PCXI,PCXArray,PCXCount,NumWrite);
 PCXCount:=0;
 CLOSE(PCXI);
 CLOSE(ODD);
 CLOSE(EVEN);
 Sound(440);
 Delay(200);
 Nosound;
end;



Procedure GraphicOn;
 Begin
```

```
   Block_X1:=0;   { Set Screen Size for a full PCX file }
   Block_Y1:=0;
   Block_X2:=799;
   Block_Y2:=557;
   PixelperByte:=1;
  end;

Var
 PCXNAME      : String;

 { Main Program to Get a For converting of a PCX file }

  Begin
   If ParamCount < 1 then
    Begin
     PCXNAME:='Header.PCX';  { If no file specified use Header.PCX }
    end                      { as the output File }
   Else
    Begin
     PCXNAME:=ParamSTR(1)    { Use file as spesified on the command line }
    end;
   GraphicOn;                 { Sets PCX Size of captured data screen }
   PCXHeader(PCXNAME);        { Writes the standard PCX header }
   GetPcx(PCXNAME);           { Convert ODD,EVEN files to PCX file }
   PCXPalette(PCXNAME);       { Stores GrayScal Pallette for VGA }
  end.
```