GDS-II - Graphic Display Subsystem

Hardware Description

GDS-II - Graphic Display Subsystem

Author :
Torsten Wiese

Technical Documentation

Version 1.1 (G300 B02)

June 1990

This documentation is written for a GDS-II equipped with a G300B (IMSG300G-11SA ENG B07).

Contents

Preface

When I finally finished this handbook, I realized that it had become somewhat longer than intended. I'm sure that everyone will be thrilled about such detailed instructions (at least at first). This will change abruptly as soon as you start looking for some certain detail among this 100-plus page opus. Therefore, a few opening instructions for using this instruction manual... .

The first two chapters describe the transputer node's basic functions without explaining the GDS-II's graphic functions in any great detail.

Chapter three will be the most tedious of all, as it explains the basics of G300 programming as well as the GDS-II's architecture in depths. I tried to structure this chapter in such a way as to make it possible for a beginner to slowly grow acquainted with the system and its programming. Skip this chapter if you don't need to know (or care) exactly where a given pixel is located within the video memory. This is even more so the case if your going to be using the GDS-II with an existing software (X-Windows under HELIOS etc.).

Chapter four contains detailed descriptions of each jumper's functions. Advanced/experienced GDS-II users, however, will find the jumper listings in chapter 9.6 to be more useful in everyday work.

I would especially like to point out the index at the end of the manual,- it should make digging out info a lot easier... .

Since I spent a lot of time and effort writing this manual I'd be grateful for any corrections, comments or criticisms concerning it.

Enjoy!

T. Wiese

P.S.

Thanks for Jacques Beckman for translating the german version into " readable " English !

1. Introduction To The GDS - II

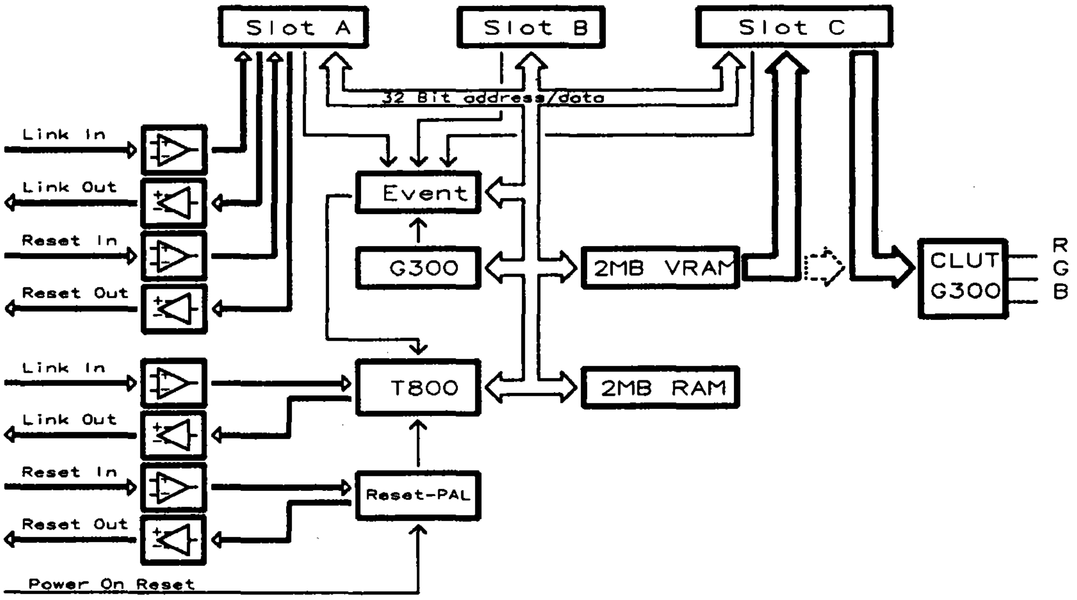


Figure 1.0 Block wiring diagram of the GDS-II

Figure 1.0 shows the basic parts of a GDS-II in a block diagram. The transputer node consists of a T800 transputer with a 2 MByte dynamic working memory (DRAM), a 2 MByte dynamic video memory (VRAM) and 4 Parsytec UniLinks. The 4 additional UniLinks connected to slot A make it possible to add a DBT-x or a DB-DMA-Module. DB-DMA-Module in turn is also equipped with a transputer capable of writing data into the video memory. With two transputers (one on the GDS-II & one on the DB-DMA-Module) working, the effective data transfer rate from an external transputer network into the GDS-II's video memory is, of course, considerably greater.

The G300 Video-Controller with its internal Colour Lookup Table (CLUT) is " I/O mapped " within the transputer's address space. The VRAM's video data outputs are bridged into the G300's inputs. If the bridge is removed, you can install an external 13 bit CLUT (DB-CLUT) here.

2. Transputer-Node Description

2.1 The Processor

The GDS-II is built around a 32 bit, T800 transputer. The " *Processor Clock* " depends upon the exact processor type used, and can be changed by rearranging jumper " J3 " (see chapter 4).
The GDS-II is equipped with 2 MBytes of dynamic working memory and 2 MBytes of " *dual ported* " video memory. The working memory starts at the address #00000 (0x80000000) and ends at #7FFFF (0x801FFFFF). In the standard option, the video memory starts where the working memory ends - beginning at address #80000 (0x80200000) in other words. The processor can therefore use both memories as one continuous memory.
If a continuous memory is not desired due to the specific uses, the video memory's start address can be shifted to a higher value by exchanging a PAL. This, however, may only be done by the manufacturers.

The " *Link Speed* " can be set at either 10 or 20 MB/s. A rate of 5 MB/s cannot be supported. The transfer rates of pins 1-3 can now additionally be set using pin 26B of the 96-way VG-connector (see chapter 4 - Jumper J4).

2.2 The Parsytec UniLinks

A total of 7 UniLinks are connected to the backplane via the (96-way) VG-connector. Each Link, in turn, consists of four differential signals e.g. eight wires. These signals are accessible on the backplane by using (10-way) BERG-connectors.

Reset Out:	Reset-Register's programmable reset outputs
Reset In:	Reset input for resetting the transputer
Link Out:	
Link In:	

```
Link-connector pinout                      Video connector pinout

Reset out +  || o o || Reset out -          HSYNC  || o o || GND
Link  out +  || o o || Link  out -          blue   || o o || green
GND          || o o    GND                  GND    || o o    GND
Link  in -   || o o || Link  in +           red    || o o || HCLK 1)
Reset in -   || o o || Reset in +           VSync  || o o || GND
```

By wiring the signals to the connector in this type of an array, you'll need only one type of cable to connect PARSYTEC-Boards among each other (crossed lines).

Link In +	═	Link Out +
Link In -	═	Link Out -
Link Out +	═	Link In +
Link Out -	═	Link In -
Reset In +	═	Reset Out +
Reset In -	═	Reset Out -
Reset Out +	═	Reset In +
Reset Out -	═	Reset In -
Link connector	Cable	Link connector

[1] The GDS-II's " *Video Clock* "; running either at 5 MHz or 32 MHz depending on jumper J5

2.3 The Backplane Link Layout

Backplane Links 0, 1, 2 and 3 correspond to the GDS-II transputer Links. Links 4, 5 and 7 are routed to the slot A connector. Slot A's Link 6 is only accessible via the GDS-II's onboard (10-way) BERG connector - its not routed to the Backplane.

The BERG connector normally corresponding to Link 6 is instead used, analogous to the GDS-I, for video output. The following figure demonstrates the backplane Link layout.

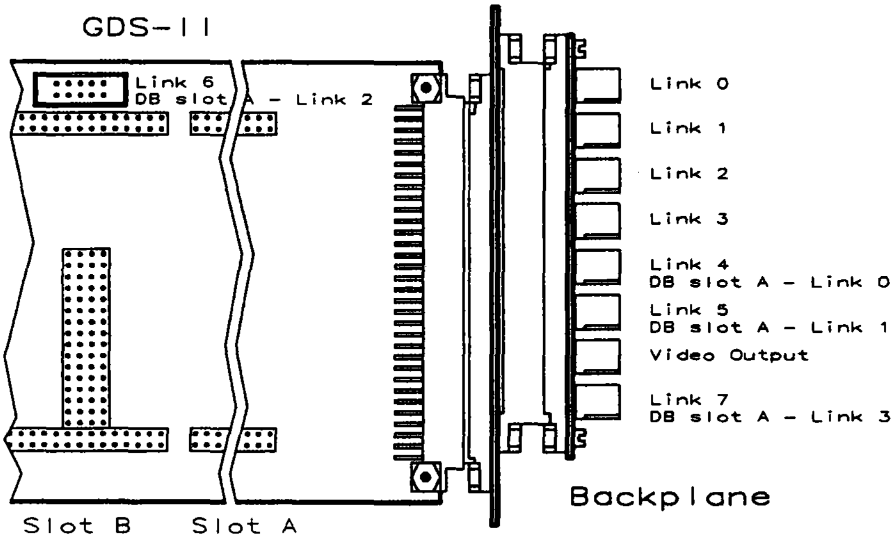


Figure 2.0 The backplane Link layout

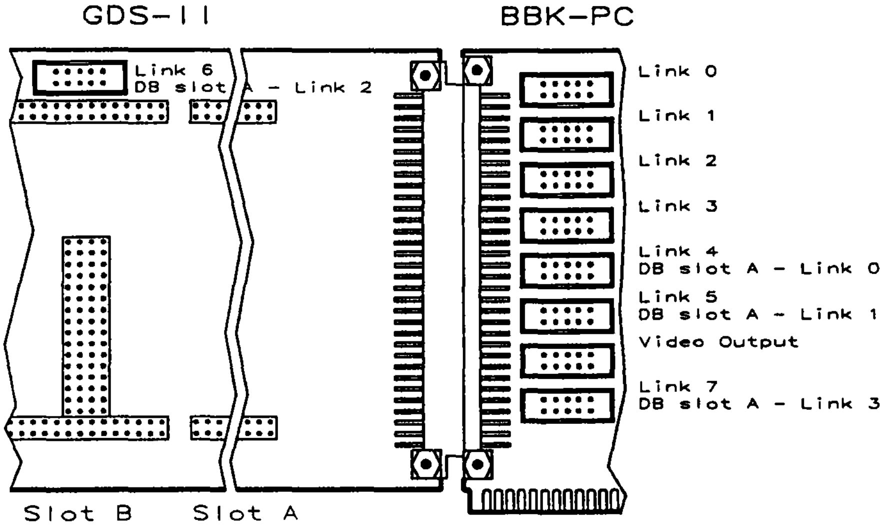


Figure 2.1 The BBK-PC Link layout

2.4 The Reset Mechanism

There are three different ways to reset the transputer and " *Video Controller* ".

- *Power-On-Reset:* When starting the system, the GDS-II will automatically generate a reset signal which will affect all components on the board.

- *Master-Reset:* The " *Master-Reset* " has the same effect as the " *Power-On-Reset* ". It enters the system via the (96-way) VG-connector's A24 pin which is connected to the reset button of the various MULTICLUSTER systems.
  The " *Master Reset* " cannot be applied if the GDS-II is used together with a BBK-PC or an ADAPT-PC.

- *Link-Reset-In:* In this case, the processor can be resetted using any of the four Links' " *Reset-In* " channels (they're logical OR arranged). **The Link-Reset-In procedure will not reset the G300 (!)**[2]. It is therefore possible to initialize the G300 independently from the main program by using a separate configuration program.

The GDS-II is also equipped with four programmable " *Link-Reset-Out* " channels in addition to a programmable reset for the G300. When this is used, a code word (following a predetermined start sequence to avoid an accidental reset) is sent to the address #20000030h (0xC0). The code word bits have the following functions:

bit 0:	Reset-Out via Link 0
bit 1:	Reset-Out via Link 1
bit 2:	Reset-Out via Link 2
bit 3:	Reset-Out via Link 3
bit 4:	**G300 Video-Controller Reset**

This design enables the user to reset, load a new program code into and start any one of the network's transputers individually via one of the four Links. These functions are what makes large, error tolerant transputer systems possible.

[2] If the G300 is " *hanging* " during a transfer cycle, it is impossible to reset the G300 and the transputer via software. You have to use the " *Power-On-Reset* " or the " *Master Reset* ". If you want your G300 reset by " *Link-Reset-In* " contact us for modifying your GDS-II.

The following OCCAM 2 listing demonstrates the programming and the necessary start sequence.

```
PROC reset  ( VAL INT link )
  -- reset channel 0: link = 0
  -- reset channel 1: link = 1
  -- reset channel 2: link = 2
  -- reset channel 3: link = 3
  -- reset G300     : link = 4
  INT addr.reset, time :
  PLACE addr.reset AT #20000030  : -- Address Reset-Registers
  TIMER clock       :
  VAL INT wait IS 2 :
  SEQ
    addr.reset := 0                -- Release Sequence for
    addr.reset := 1                -- the Reset Register
    addr.reset := 2
    addr.reset := 3
    addr.reset := 1 << link        -- Reset Code word
    clock ? time
    clock ? AFTER time PLUS wait   -- 128 us Delay
    addr.reset := 0
:
```

The same procedure in 'C' under HELIOS:

```
void reset ( int link )
{
    int * addr_reset = (int *)0xC0
    *addr_reset = 0 ;
    *addr_reset = 1 ;
    *addr_reset = 2 ;
    *addr_reset = 3 ;
    *addr_reset = ( 1 << link) ;
    Delay (128)     ;
    *addr.reset = 0 ;
}
```

3. The GDS-II Video Section

3.1 Graphic System Basics

In addition to the usual working memory, a graphic system also needs a special video memory for storing the image as a pixel pattern. The transputer uses the video memory just as it would a ordinary dynamic working memory. The video memory is not only accessed by the transputer (which creates, rotates, magnifies, shifts, etc. images), but also by the Video Controller (G300). The Video Controller will periodically address a screen line and transfer it to one of the video RAM's internal shift registers from where it's sent to the screen (e.g. called Screen Refresh). When the G300 is addressing the video memory the transputer is cut off (DMA-cycle) and forced to rely on its own internal registers and On-Chip-RAM. This explains the GDS-II transputer node's lower performance (approx. 8% less) in comparison to similar transputer nodes without a video section.

3.1.1 How An Image Is Built Up

As stated above, an image is stored as a pixel pattern in the video memory. The G300 will read a complete video line at a line frequency of 15 kHz - 64 kHz and convert the information into a serial " pixel flow ". In order to do this, the binary coded colour information stored in the video memory first has to be converted into analog voltage values. This is done by the G300's three internal 8 bit D/A Converters.

In addition to all this, the G300 generates the driving signals for the monitor (*Horizontal Sync and Vertical Sync or Composite Sync*) and if you want it to, will combine them with the colour outputs accordingly.

3.1.2 The Number Of Available Colours

The width of a pixel determines the number of colours that can be used simultaneously. An 8 bit wide pixel can encode $2^8=256$ different colours, a 24 bit pixel, 2^{24}.

Both modes are possible with a GDS-II. In addition, a 13 bit mode can also be employed when using the appropriate added board (DB-CLUT).

3.1.3 The G300's Colour-Lookup-Table

Using the 8 bit/pixel mode, a total of 256 colours can be used by the GDS-II simultaneously. The monitor, however, can only use the data if the colours have been separated into the three basic colours. In order to do this, the G300 is equipped with three separate D/A-Converters (red, green & blue). Each D/A Converter is able to convert an 8 bit wide signal in up to 256 different voltage values. Each basic colour can therefore be shown in 256 different shades. The three D/A converters make it possible to show 768 different basic colour shades which, in turn, can be combined for a total of 16,776,448 mixtures.

The CLUT enables the user to choose 256 colours from the wide range possible, and address them with an 8 bit wide pixel. It's essentially a memory for 256 words at 24 bits each. The following figure will demonstrate how it works. The CLUT address and the pixel's value are identical.

CLUT Address		Memory contents			Colour seen on the monitor
		red	green	blue	
#00	0x00	00h	00h	00h	black
#01	0x04	FFh	FFh	FFh	white
#FF	0x3FC	80h	80h	00h	medium yellow brightness

Colours can be coded directly when using the 24 bit/pixel mode.

Bits 0-7 stand for the red, bits 8-15 for the green, bits 16-23 for the blue colour component and bits 24-31 are ignored.

The older G300A automatically switches off the CLUT in mode 2 (13, 24 bit/pixel). The newer G300B, however, enables you use your CLUT.

The user can then use a 256 word á 8 bit table for **each basic colour**, making quick monitor colour changes possible without having to alter the video memory.

3.1.4 Programming The G300

The G300's registers are located in the transputer's address space from the base address #20100000 (0x400000) onwards. There's a summary with all registers and functions in chapters 3.3 and 3.4.

The registers fall into several categories.

- The CLUT uses 256 words (1 kByte) of memory and can be reprogrammed while an image is running. To avoid disturbances, try to program only during the " *Vertical Blank* " cycles. The appropriate signal is sent to the transputer's Event-Pin. This is explained in detail in chapter 7.8[3].

- Registers that determine the G300's function. This includes choosing the pixel width, " *Pixel Clock* ", etc.
  These registers are loaded every time the system is started, then usually left alone.

- Registers that determine the monitor timing thereby also determining the resolution. The parameters are set by the monitor manufacturer and should be followed exactly. The parameters for several widespread models were tested (!) and are listed in the appendix along with the appropriate driver disc. Don't change the parameters while your monitor is in use, as false programming might damage it.

- Registers that can be modified during use in order to shift portions of the image or to switch from one image to another within the video memory (Double Buffering).

[3] Software examples ; chapter 7.8 " *Synchronizing With The Vertical Blank* "

3.2 The Organization Of The Video Memory

The GDS-II has a 2 MByte video memory that starts at the hardware address 0x80200000. This corresponds to OCCAM word address #80000.

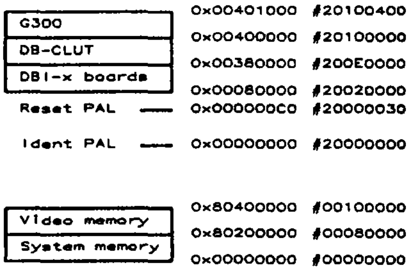


Figure 3.0 The video memory's position in the address space

The video memory is equipped with dual-ported Video RAMs, 4 * 256 kBit in size, and consists of two banks. Each bank is organized as a 512 * 512 word matrix. Two banks result in a 512 * 1024 word organisation.

To address a certain word within the matrix, you need a 9 bit " *Column-Address* " and a 10 bit " *Row-Address* ".

The number of pixels per memory cell depends on the selected mode:

24 bit/pixel Mode:

1 pixel a' 24 bit and 8 bits for additional information

Byte 0 = red ; Byte 1 = green ; Byte 2 = blue

13 bit/pixel Mode: (can only be used with the DB-CLUT)

2 pixel a' 13 bit and 2 * 3 bits for additional information

Byte 0 and Byte 1 comprise pixel number 2n

Byte 2 and Byte 3 comprise pixel number 2n+1

8 bit/pixel Mode:

4 pixel a' 8 bit

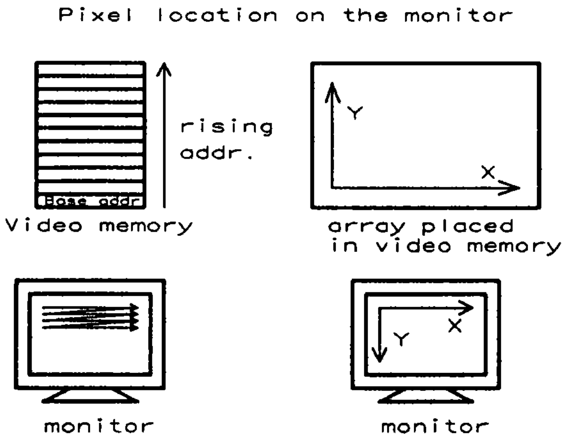
Byte 0 comprises pixel number 4n

Byte 1 comprises pixel number 4n+1

Byte 2 comprises pixel number 4n+2

Byte 3 comprises pixel number 4n+3

Pixel location on the monitor:



This figure demonstrates how the pixel's location on the monitor corresponds its location in the video memory. As opposed to the GDS, the GDS-II does in fact use normal graphic standards as far as video memory addressing are concerned. The left upper monitor pixel is located at the bottom-most video memory address.

Figure 3.1 Pixel location on the monitor

As the addresses of the memory cells rise, rows are built up from left to right and the screen is filled with the resulting lines from top to bottom.

When using the G300 Video-Controller there are two ways to address the video memory; - " **Row-Oriented Addressing** " which is explained in chapter 3.2.1 and " **Linear Addressing** " which is explained in chapter 3.2.2.

3.2.1 Row-Oriented Addressing

The simplest memory organisation consists in mapping *VideoRAM* - rows to monitor rows and *VideoRAM* - columns to monitor columns. As a consequence, each VideoRAM-row can only correspond to one monitor row. The figures 3.2 - 3.4 will illustrate just how an image is stored in the video memory and the dependency on the mode of representation (24, 13 or 8 bit/pixel). They are based on the assumption that the first monitor pixel is located at the lowest video memory address[4] (0x00200000 or #80000).

[4] " *TopScreen Register* ", see 3.2.1.1 and 3.4.4, - set to zero in this case.

The location of an image in the video memory when using " Row-Oriented Addressing ":

24 bit/pixele:

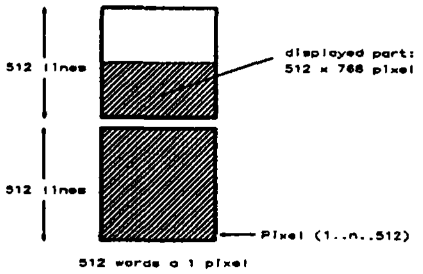


Fig. 3.2

Up to 512 pixels (horizontal resolution) by 1024 pixels (vertical resolution) are possible. The lower left position in the video memory (0x200000) corresponds to the first pixel in the first monitor row. The right pixel in the bottom monitor row corresponds to the last memory position (0x400000).

13 bit/pixele: (only with a DB-CLUT)

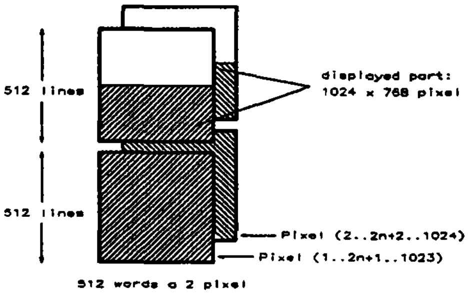


Fig. 3.3

Up to 1024 pixels (horizontal resolution) by 1024 pixels (vertical resolution) are possible. Since 2 pixels are stored per word in this mode, two planes are used (the front one's for bytes 0 and 1, the rear one's for bytes 2 and 3).

8 bit/pixele:

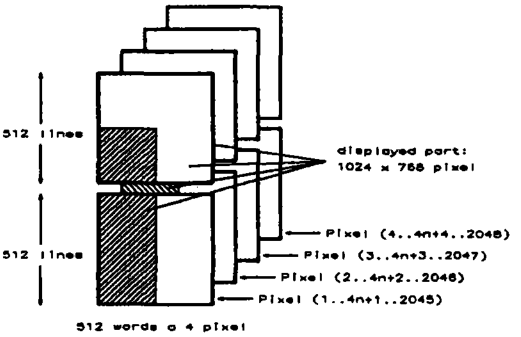


Fig. 3.4

Up to 2048 pixels (horizontal resolution) by 1024 pixels (vertical resolution) are possible. Since 4 pixels are stored per word in this mode, a total of four planes are used. The front plane corresponds to byte 0, the rear plane to byte 3.

At this point one realizes that the 24 bit/pixel mode used in this configuration isn't very useful (a maximum of 512 pixel in horizontal orientation).

The following examples should demonstrate how to find the address of a specific pixel depending on the chosen mode.

Example[5]:	$x = 400$	$x_{8bit} \in \{ 0,.., 2047 \}$	$x_{13bit} \in \{ 0,.., 1023 \}$	$x_{24bit} \in \{ 0,.., 512 \}$
	$y = 64$	$y_{8bit} \in \{ 0,.., 1023 \}$	$y_{13bit} \in \{ 0,.., 1023 \}$	$y_{24bit} \in \{ 0,.., 1023 \}$

8 bit/Pixel Mode

Each word consists of four pixels in this mode. The pixel in question is located in the first byte of the memory cell in row 64 and column 100.

$$\begin{aligned} Adr_{Col} &= 100 = 001100100 = 64h \\ Adr_{Row} &= 64 = 0001000000 = 40h \end{aligned}$$

$$\begin{aligned} Adr &= (Adr_{Row} * 512) + Adr_{Col} + Adr_{Base} \\ &= (Adr_{Row} << 9) + Adr_{Col} + Adr_{Base} \\ &= \# 8064 + \#80000 \\ &= \#88064 \end{aligned}$$

This address is an OCCAM word address.
The byte address generated by the hardware is therefore
$Adr_{Hardware} = 0x80220190$

General equation 3.1a: (to determine the OCCAM word address for 8 bit/pixel)

$$\begin{aligned} Adr &= Adr_{Base} + \tfrac{1}{4}x + 512y \\ &= Adr_{Base} + (x >> 2) + (y << 9) ; \qquad Adr_{Base} = \#80000 \end{aligned}$$

General equation 3.1b: (to determine the hardware address for 8 bit/pixel)

$$Adr = Adr_{Base} + x + (y << 11) ; \qquad Adr_{Base} = 0x80200000$$

[5] The origin of the coordinate system lies at lower left corner of the video memory (see figure 3.1).

13 bit/Pixel Mode

Each word contains exactly two pixels in this mode.

$$\begin{aligned} Adr_{Col} &= 200 = 011001000 = C8h \\ Adr_{Row} &= 64 = 0001000000 = 40h \end{aligned}$$

$$\begin{aligned} Adr &= (Adr_{Row} * 512) + Adr_{Col} \quad + Adr_{Base} \\ &= (Adr_{Row} << 9) + Adr_{Col} \quad + Adr_{Base} \\ &= \#\ 80C8 \quad + \#80000 \\ &= \#880C8 \end{aligned}$$

$$Adr_{Hardware} = 0x80220320$$

General equation 3.2a: (to determine the OCCAM word address for 13 bit/pixel)

$$\begin{aligned} Adr &= Adr_{Base} + \tfrac{1}{2}x + 512y \\ &= Adr_{Base} + (x >> 1) + (y << 9) \end{aligned}$$

$$Adr_{Base} = \#80000$$

General equation 3.2b: (to determine the hardware address for 13 bit/pixel)

$$Adr = Adr_{Base} + (x << 1) + (y << 11)$$

$$Adr_{Base} = 0x80200000$$

24 bit/pixel Mode

Each word represents exactly one pixel.

$$\begin{aligned} Adr_{Col} &= 400 = 110010000 = 190h \\ Adr_{Row} &= 64 = 0001000000 = 40h \end{aligned}$$

$$\begin{aligned} Adr &= (Adr_{Row} * 512) + Adr_{Col} \quad + Adr_{Base} \\ &= (Adr_{Row} << 9) + Adr_{Col} \quad + Adr_{Base} \\ &= \#\ 8190 \quad + \#80000 \\ &= \#88190 \end{aligned}$$

$$Adr_{Hardware} = 0x80220640$$

General equation 3.3a: (to determine the OCCAM word address for 24 bit/pixel)

$$\begin{aligned} Adr &= Adr_{Base} + x + 512y \\ &= Adr_{Base} + x + (y \ll 9) \end{aligned}$$

$$Adr_{Base} = \#80000$$

General equation 3.3b: (to determine the hardware address for 24 bit/pixel)

$$Adr = Adr_{Base} + (x \ll 2) + (y \ll 11)$$

$$Adr_{Base} = 0x80200000$$

Initialisation examples

The smart array definition can make addressing specific pixels a whole lot easier.

8 bit/pixel Mode: (OCCAM 2) ('C')

```
VAL  YSIZE  IS 1024         :     #define YSIZE 1024
VAL  XSIZE  IS 2048         :     #define XSIZE 2048
[YSIZE][XSIZE] BYTE VRAM    :     char *VRAM =
PLACE VRAM AT #80000        :       (char *) 0x80200000;
```

13 bit/pixel Mode: (OCCAM 2) ('C')

```
VAL  YSIZE  IS 1024         :     #define YSIZE 1024
VAL  XSIZE  IS 1024         :     #define XSIZE 1024
[YSIZE][XSIZE] INT16 VRAM   :     short *VRAM =
PLACE VRAM AT #80000        :       (short *) 0x80200000;
```

24 bit/pixel Mode: (OCCAM 2) ('C')

```
VAL  YSIZE  IS 1024         :     #define YSIZE 1024
VAL  XSIZE  IS  512        :      #define XSIZE 512
[YSIZE][XSIZE] INT VRAM     :     int *VRAM =
PLACE VRAM AT #80000        :       (int *) 0x80200000;
```

The following OCCAM sequence draws a vertical white (if the CLUT is programmed that way) line at the position x.

8 bit/pixel Mode :

```
SEQ y = 0 FOR 1024
  VRAM [y][x] := BYTE (0)
```

13 bit/pixel Mode:

```
SEQ y = 0 FOR 1024
  VRAM [y][x] := INT16 (0)
```

24 bit/pixel Mode:

```
SEQ y = 0 FOR 1024
  VRAM [y][x] := 0
```

The following is an example of how, when in the 8 bit/pixel mode, the definitions can be incorporated into a program under HELIOS 'C', and how to draw a white vertical line at the position x = 20.

```
#define  XSIZE  2048
#define  YSIZE  1024
char *VRAM = (char *) 0x80200000 ;
/*   function prototype */
void putpixel ( int x, int y, int colour );

main ( )
{
    int i ;
    int x = 20 ;
    for ( i = 0; i < 1024; i++ ) {
        putpixel ( x, i, 0 ) ;
    }
}

void putpixel ( int x, int y, int colour )
{
    if (( x < XSIZE ) && ( y < YSIZE ))
        VRAM [x+ (XSIZE * y)] = colour ;
}
```

3.2.1.1 Hardware Panning With The G300: (Row-Oriented Addressing)

Since the picture will usually be smaller than the video memory, the G300 Video Controller will have to be told where to look for it.
This can be done by programming the G300's " **TopSreen Register** ". This register will then contain the image's start address within the video memory (i.e. the address of the upper left pixel on the monitor, see Fig. 3.1).

The G300 generates the " *Row-Address* " on bits 2-11 and the " *Column-Address* " on bits 12 - 20. The address sequence is switched compared to the sequence used by the transputer[6]. (bits 0 and 1 are not used when generating the word address.)

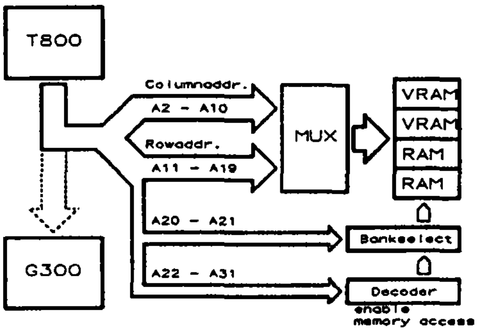


Figure 3.5 T800 Memory access

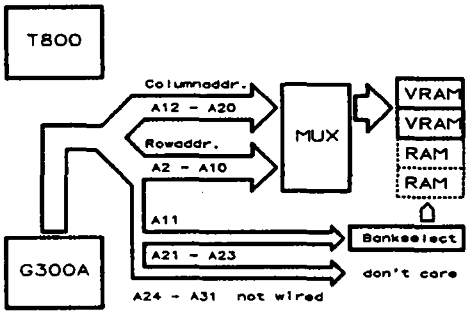


Figure 3.6 G300 Memory access

Examples for calculating G300's TopScreen Register

<u>8 bit/pixel Mode: 1024 * 768 pixel from a 2048 * 1024 pixel plane</u>

Top of Screen[7] = 0: The area ranging from { x , y = 0 , 0 } to { x , y = 1023, 767 } will be shown.

Now you want to move the image in the video memory 100 pixels to the right and 40 pixels **upwards**[8]. To do this, the " *TopScreen Register* " has to be loaded with the **pixel address** { x , y = 100 , 40 }. (The pixel is in the first byte of the 32 bit word with the coordinates { x , y = 25 , 40 })

[6] Only affects the programming of the " *TopScreen Register* ".

[7] " *Top of Screen* " is the value stored in the " *TopScreen Register* "

[8] The screen image will be shifted 100 pixels to the right and 40 pixels downward.

Adr_{Col} = 25 = 000011001 = 19H ≡ A12 .. A20

Adr_{Row} = 40 = 0000101000 = 28H ≡ A2 .. A11

(A11 is used for switching between the two video memory banks)

The value stored in the " TopScreen Register " is put on the address/data bus and interpreted as a word address by the Video-RAMs, - that's why the two bottom bits have to be set to zero.

$$\text{Top of Screen} = [(\text{Adr}_{\text{Col}} * 1024) + \text{Adr}_{\text{Row}}] * 4$$
$$= [(\text{Adr}_{\text{Col}} << 10) + \text{Adr}_{\text{Row}}] * 4$$
$$= 190A0h$$

Using an OCCAM-Word address in connection with the " TopScreen Register " won't make any sense since values stored in the TopScreen Register consist of normal " data words ". These have to interpreted as addresses by the address decoder in the GDS-II before being usefuL

General equation 3.4a: (determining the Top of Screen for 8 bit/pixel)[9]

Top of Screen	= 1024x + 4y	for x=4n
	= 4[1024 * Int(¼x)] + 4y	for all x
	= (x << 10) + y << 2	for x=4n
	= ((x >> 2) << 12) + (y << 2)	for all x

Note:

Since each word represents four pixels in the 8 bit/pixel mode and the G300 is only able to address word by word, horizontal Hardware-Panning is only possible using even multiples of 4.

Note: The highest allowable " Row-Address " in the " Row-Oriented Addressing Mode " is: 1023 - vertical pixel resolution.

Applying the above example, you get: Top of Screen < = 1023 - 768 = 255

If you were to, for instance, program in a value of 257, the top monitor row (768th) would receive the " Row-Address " 1025. The topmost bit in this value is then interpreted as the bottom " Column-Address " bit. This in turn causes the top two monitor rows to be shifted to the right by four pixels (= 1 word).

[9] Let x , y be the coordinates of the lower left screen pixel in the video memory after shifting.

13 bit/pixel Mode:

The 13 bit/pixel mode equation can easily be derived from equation 3.4a. This time, the user has access to a video memory 1024 * 1024 pixels in size. As in the 8 bit/pixel mode you can only shift horizontally word by word. As a result one can only move to the left or right by steps of multiples of two.

General equation 3.4b: (determining the Top of Screen for 13 bit/pixel)[10]

$$\begin{array}{lll} \text{Top of Screen} & = 2048x + 4y & \text{for } x=2n \\ & = 2[2048 * \mathrm{Int}(\tfrac{1}{2}x)] + 4y & \text{for all } x \\ & = (x \ll 11) + \; y \ll 2 & \text{for } x=2n \\ & = ((x \gg 1) \ll 12) + (y \ll 2) & \text{for all } x \end{array}$$

24 bit/pixel Mode:

Again, the 24 bit/pixel mode equation can easily be derived from equation 3.4a and is only explained for the sake of being complete. When using the G300's " *Row-Oriented Addressing* " in this mode, the user has access to a video memory 512 * 1024 monitor pixels in size. Since, however, one is limited to 512 pixels in horizontal orientation, you will probably not want to use " *Hardware - Panning* " in the 24 bit/pixel mode, but use the G300 in its " *Linear Addressing Mode* "[11] instead.

General equation 3.4c: (determining the Top of Screen for 24 bit/pixel)[8]

$$\begin{array}{lll} \text{Top of Screen} & = 4096x + 4y & \text{for all } x \\ & = (x \ll 12) + \; y \ll 2 & \text{for all } x \end{array}$$

[10] Let x , y be the coordinates of the lower left screen pixel in the video memory after shifting.

[11] See chapter 3.2.2

3.2.1.2 The Importance Of The Registers MemInit And TransferDelay

Two other registers are relevant for " *Hardware - Panning* ". The G300 not only has to know how long an image row really is, it also has to know how many pixels from the video memory's next row it has to have access to. This is achieved by programming the registers **TransferDelay** and **MemInit**. When the values stored in these registers are divided by 1/4th of the " *Pixel Clock* " you get a time span in seconds.

MemInit tells the G300 at what point in time it has to start its transfer cycle, **TransferDelay** tells you how long it takes the G300 to complete the cycle. The sum of both registers, multiplied by four[12], results in the number of pixels after which the G300 has to complete a transfer cycle (i.e. access the next video memory row).

The way the video memory rows and monitor rows are linked to each other (ridged, that is) the sum must always be identical to one quarter of the horizontal pixel resolution (this is what makes Hardware-Panning possible).

In our example[13]*:*
*TransferDelay + MemInit = ¼ * 1024 = 256 !!*

Some additional explanations concerning the registers TransferDelay and MemInit:
(for the more advanced user)

When creating an image, one complete VRAM-row (2048 pixel for 8 bit/pixel, 1024 pixel for 13 bit/pixel and 512 pixel for 24 bit/pixel) is always read and loaded into the VRAMs' internal shift register. This process is referred to as a **Transfer Cycle**. *The sum of the register contents of TransferDelay and MemInit determines when (after how many pixels) a new transfer cycle has to be ended.*

Note that all time spans have to be stated as a multiple of 4 pixels.

The ridged coupling of VRAM-rows and monitor rows described here amounts to the following correlation:

MemInit + TransferDelay = XSIZE[14] */ 4*

This ensures that a new Transfer cycle will be started every time the end of a monitor row is reached. Make sure that the number of pixels shown in horizontal orientation on the screen does not exceed four x *the length of the VRAMs' shift register (one* x *the length when using 24 bit/pixel).*

*The VRAMs used (organized 256k * 4) have a 512 * 512 word matrix, i. e. every row, including the VRAMs' internal shift registers, is 512 words long.*

[12] The multiplication factor is determined by the G300

[13] 1024 * 768 pixels in the 8 bit/pixel mode

[14] Video memory's horizontal pixel resolution !

The effective shift register length depends on the mode used.

The shift register length in the 24 bit/pixel mode is 512 pixels. Consequently, the sum of MemInit and TransferDelay **may not exceed 128** *(¼ * 512).*

When using the 8 bit/pixel mode, each word contains four pixels, each of which are demultiplext in the G300. This results in an effective shift register length of 2048 pixels in this mode. The sum of MemInit and TransferDelay **may not exceed 512** *(¼ * 2048).*

*Each word has two pixels in the 13 bit/pixel mode. The G300, however, will be working in its 24 bit/pixel mode and is therefore expecting one pixel per word. The DB-CLUT Daughterboard is equipped with a 2:1 Multiplexer and, with the help of the 8192 * 24 bit CLUT, can take two 13 bit pixels (these are read from the video memory at half the " Pixel Clock " rate) and convert them into two 24 bit pixels which are then fed to the G300 at the rate of the normal " Pixel Clock ". The effective shift register length in this mode is therefore 1024 pixels. The sum of MemInit and TransferDelay* **may not exceed 256** *(¼ * 1024).*

The contents of the register **TransferDelay** tells the G300 how much time it takes to complete a transfer cycle.

Too short a time span (the TransferDelay value's too small) leads to bus conflicts on the board. When the transputer has BLOCKMOVE running one will then see **short horizontal stripes on the monitor.** The Transputer is trying to access the working memory while the G300 is still in the middle of a transfer cycle. **This may eventually lead to hardware damage.** TransferDelay values that are too large will not damage the system, but will reduce its performance.

The following table (middle column) lists the necessary values, depending on the "*Pixel Clock* ", for a 17.5 MHz Transputer in a 8 bit/pixel mode. (These are " *worst case* " values, safe & sufficient for all modes and any transputer types.)

	TrDelay		
pixelClock	24bpp	8bpp	8bpp
	17.5 MHz	17.5 MHz	25 MHz
30 MHz	-	10	9
32 MHz	11	-	-
35 MHz	-	11	9
40 MHz	-	12	10
45 MHz	-	13	11
50 MHz	-	14	11
55 MHz	-	15	12
60 MHz	-	16	12
65 MHz	-	16	13
70 MHz	-	17	14
75 MHz	-	18	14
80 MHz	-	19	15
85 MHz	-	20	16
90 MHz	-	21	16
95 MHz	-	22	17
100 MHz	-	23	17
105 MHz	-	24	18
110 MHz	-	24	19

The right column lists the values that can be used for a 25 MHz transputer in an 8 bit/pixel mode. These values are lower, and the resulting performance, higher. The necessary TransferDelay value can be calculated with the following equation.

```
TrDelay = [( 28 TState + 20ns ) * ¼ pixelclock ]/[s] + 4.5 (24,12bpp)
TrDelay = [( 24 TState + 20ns ) * ¼ pixelclock ]/[s] + 4.5 (8bpp)
```

```
TSTate = 20ns for 25 MHz Transputer
       = 25ns for 20 MHz Transputer
       = 28.6ns for 17.5 MHz Transputer
```

... which result in the following equations ... :

```
TrDelay = [3.5 * PClk / TClk + 0.005 * PClk] + 4.5  (24,12bpp)
TrDelay = [3   * PClk / TClk + 0.005 * PClk] + 4.5  (8bpp)
```

bpp = bits per pixel
[s] = second
TClk = Transputer Clock (" Processor Clock ") in MHz
PClk = " Pixel Clock " in MHz

(Dis-) Advantages of Hardware Panning

The ridged correspondence between the image shown and the video memory architecture (dictated by the system) makes it possible to move the monitor image " across and over " a larger total picture, but it costs additional video memory...

> *Using our example again (1024 * 768 pixel), we get the following situation:*
> *Beside the image you already have, you still have an equally large " space " for a second image. If, however, you don't want to store a second image, you will hardly be able to use this memory (768 kByte) since it consists of 768 separate 1 kByte sections (as opposed to linear addressable block).*

3.2.2 Linear Addressing

The video memory can be used most effectively if the shift registers are used to their full length. This means:

24 bit/pixel:	**MemInit + TransferDelay =: 128**
13 bit/pixel:	**MemInit + TransferDelay =: 256**
8 bit/pixel:	**MemInit + TransferDelay =: 512**

A new transfer cycle, independent of the horizontal timing, will take place 128 pixels (256 pixels for 13 bit/pixels ; 512 pixels for 8 bit/pixel). Using this type of programming enables you to use more than 512 pixels in horizontal orientation when in the 24 bit/pixel mode. The image format may be chosen freely as long as it does not exceed a total of 512 * 1024 = 524,288 pixels (1024 * 1024 = 1,048,576 pixel for 13 bit/pixel ; 2048 * 1024 = 2,097,152 pixel for 8 bit/pixel). The following figures will try to demonstrate where in the video memory the image can be found (compare this to figures 2.2 , 2.3 , 2.4).

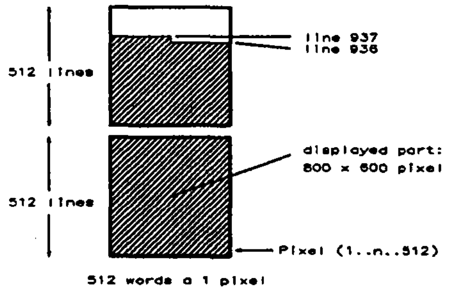


Figure 3.8 800 * 600 * 24 bit (Linear Addressing: Top of Screen =: 0)

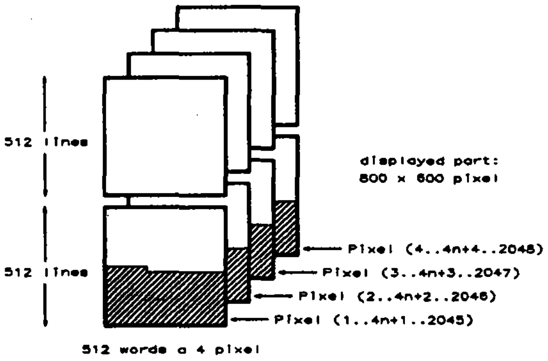


Figure 3.9 800 * 600 * 8 bit (Linear Addressing: Top of Screen =: 0)

The two above figures show the memory occupancy for " *Top of Screen* " =: 0 and should (!) make it clear why Hardware-Panning cannot be used in this mode.

Note:

> **The image's first pixel always (!) has to be placed at the beginning of a VRAM-row when using the " Linear Addressing Mode ". This means setting the " Column-Address Component " to zero when programming the " TopScreen Register " !**

3.2.2.1 Calculating Top Of Screen: (Linear Addressing)

24 bit/pixel

Image size:	800 * 600 pixels = 480,000 pixels.
Memory needed:	480,000 * 4 Byte = 1875 kByte

This corresponds to 937.5 VRAM rows at 512 pixels each. The last pixel will be placed in the middle of the last VRAM-row used.

Start address:	#8.0000H (Top of Screen =: 0)
End address:	#F5.2FFH (Top of Screen =: 0)

The unused section at the end of the video memory is 44,288 words (173 kByte) in size. By raising " *Top of Screen* "[15], however, you can move the starting and end address so that the " excess " memory (unused part of video meory) starts at the address #8.0000H. The unused memory section " starts " where the working memory " lets off " and can therefore be used as additional working memory.

Calculating the optimal value for Top of Screen

The easiest way to calculate your best " *TopScreen Register* " value would be:

Top of Screen ≡ #8.0000H - number of pixels per picture

This will only work if the number of pixels per image is dividable by 512...
If this should not be the case (*The image's last pixel is not identical to the last pixel of a VRAM row*) you're going to have to correct the equation, so that partially used rows are also considered.

Image size:	800 * 600 pixel = 480,000 pixel.
Memory needed:	937.5 Rows á 512 pixel

The starting address of the image has to also be the starting address of a VRAM row. This is what you get:

Top of Screen ≡ #0x80000 - 938 * 512 = #AC00h

[15] Value stored in the " *TopScreen Register* "

The calculated value represents the starting address and cannot be used directly for the " *TopScreen Register* ". (see chapter 3 and figure 3.5)

Use the following equation:

Top of Screen =: (1024 - 938) * 4 = 344 = 158h

44,032 words (172 kByte) of previously unusable memory are now accessible. The remaining 1.0 kByte, however, are at the top of the video memory and still can't be reached.

The following equations, listed according to the mode used, will deliver the optimal " *Top of Screen* " values.

General equation 3.6: (Top of Screen for " **Linear Addressing** " and **24 bit/pixel**)

```
Top of Screen =4 * Int[((1024 * 512)[16]- (XSIZE * YSIZE)[17])[18]/ 512][19]
              = (((1024 * 512)[16]- (XSIZE * YSIZE)[17])[18]>> 9)[19]<< 2[20]
```

Initialising the monitor for 24 bit/pixel: (example in OCCAM 2)

```
VAL  XSIZE       IS 800   :
VAL  YSIZE       IS 600   :
VAL  TopScreen   IS  .... :   ( above equation )
[YSIZE][XSIZE] INT VRAM   :
PLACE VRAM AT #80000 + (( TopScreen >> 2 )[19]<< 9) :
```

Initialising the monitor for 24 bit/pixel: (example in 'C')

```
#define  XSIZE    800
#define  YSIZE    600
#define  TopScreen   ... above equation
int *VRAM = 0x80200000 + ( TopScreen << 9 );
```

[16] Number of pixels in the video memory

[17] Number of pixels per screen

[18] Number of unused words in the video memory (VRAM)

[19] Number of unused Video-RAM rows

[20] The starting row (*Row-Address*) has to be stored from address bit A2 onward, e.g. it has to be shifted 2 bits
Note : (A > > 9) < < 2 isn't identical to A > > 7 !!

13 bit/pixel

Analogous to what's listed above. In this case, however, the unused memory (at a resolution of 800 * 600 pixels) is 1.08 MByte in size, a VRAM row contains 1024 pixels and the equation is:

General equation 3.7: (Top of Screen for " **Linear Addressing** " and **13 bit/pixel**)

Top of Screen = (((1024 * 1024) - (XSIZE * YSIZE)[21]) >> 10) << 2

Initialising the monitor for 13 bit/pixel: (example in OCCAM 2)

```
VAL  XSIZE       IS 800     :
VAL  YSIZE       IS 600     :
VAL  TopScreen   IS  .  ... :  ( above equation )
[YSIZE][XSIZE] INT16 VRAM   :
PLACE VRAM AT #80000 + ( TopScreen << 7 ) :
```

Initialising the monitor for 13 bit/pixel: (example in 'C')

```
#define  XSIZE    800
#define  YSIZE    600
#define  TopScreen  ... above equation
short *VRAM = 0x80200000 + ( TopScreen << 9 );
```

[21] Number of pixels per screen

8 bit/pixel

Again, analogous to the 13 bit/pixel mode. The unused memory is 1.54 MByte large at a resolution of 800 * 600 pixels, and a VRAM row has exactly 2048 pixels.
The corresponding equation:

General equation 3.8a: (Top of Screen for " **Linear Addressing** " and 8 **bit/pixel**)

```
Top of Screen = ((( 2048 * 1024 ) - ( XSIZE * YSIZE )) >> 11) << 2
```

Initialising the monitor for 8 bit/pixel: (example in OCCAM 2)

```
VAL  XSIZE       IS 800   :
VAL  YSIZE       IS 600   :
VAL  TopScreen   IS  .... :   ( above equation )
[YSIZE][XSIZE] BYTE VRAM   :
PLACE VRAM AT #80000 + ( TopScreen << 7 ) :
```

Initialising the monitor for 8 bit/pixel: (example in 'C')

```
#define  YSIZE    800
#define  XSIZE    600
#define  TopScreen   ... above equation
char *VRAM = 0x80200000 + ( TopScreen << 9 );
```

3.2.2.2 " Linear Addressing A Summary

When using the G300 in its " *Linear Addressing Mode* ", you are free to choose any image format you like. You are therefore also free to choose formats larger than 512 pixels in horizontal orientation when using the 24 bit/pixel mode.
Hardware-Panning is not possible in the " Linear Addressing Mode ", since the video memory is used in its entirety.
If you want to use excess video memory (video memory without pixel data) as working memory, you will have to move the image in the video memory from the " bottom " to the " top ". The necessary " *Top of Screen* " values can be calculated with the equations 3.6 - 3.8. The equation 3.9 summarizes the results.

General equation 3.9: (calculating the Top of Screen value when using " **Linear Addressing** ")

```
Top of Screen = ((1024 * (2048/btpp)) - (XSIZE * YSIZE))*(btpp / 2048))<< 2
```

XSIZE: horizontal pixel resolution
YSIZE: vertical pixel reolution
btpp: Bytes per pixel (btpp = 4 for 24 bit/pixel, btpp = 2 for 13 bit/pixel, btpp = 1 for 8 bit/pixel)

3.2.3 Double Buffering

The principals of " *Double Buffering* " are easily explained once the results of the chapters 3.2.1 and 3.2.2 are understood.

Row-Oriented Addressing in the 8 bit/pixel mode

The video memory is 2048 * 1024 pixels in size. The following figures demonstrate several possible ways to pack smaller images into the video memory.

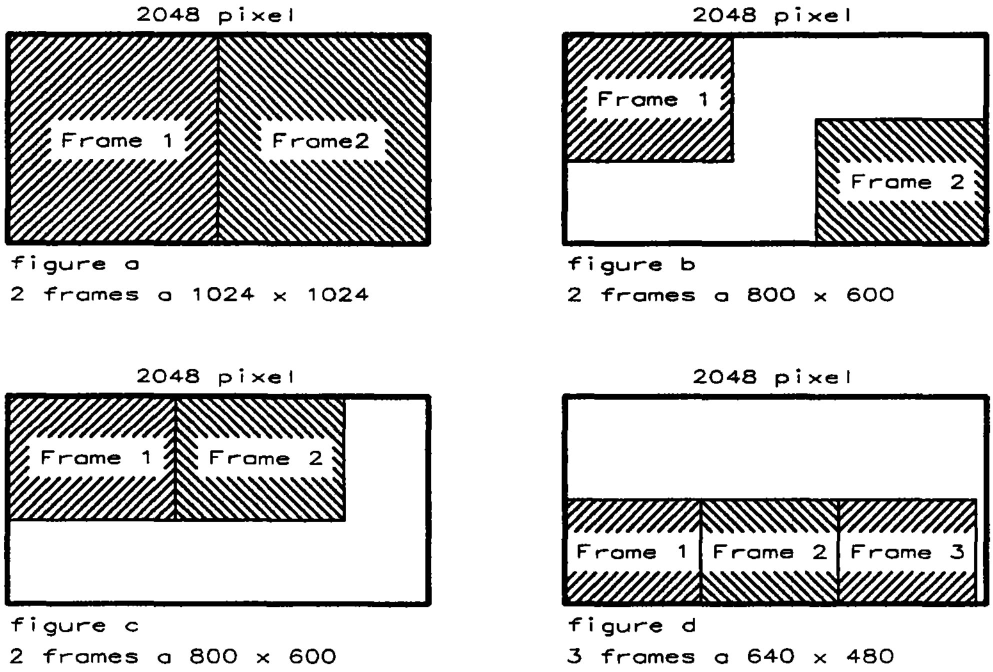


figure a
2 frames a 1024 x 1024

figure b
2 frames a 800 x 600

figure c
2 frames a 800 x 600

figure d
3 frames a 640 x 480

Figure 3.10 Examples of Double Buffering

Figure a shows two " Frames " at 1024 * 1024 pixels within the video memory. If the " **TopScreen Register** " is given the value zero (pixel address { x, y = 0, 0 }), " frame 1 " will be shown on the monitor. While " 1 " is being shown, " 2 " can be worked on by the transputer. Showing " 2 " on the monitor so that " 1 " can be manipulated by the transputer corresponds to moving everything to the right by 1024 pixels. The " *TopScreen Register* " contains the address of the pixel { x , y = 1024 , 0 }. Equation 3.4a .

The process of having two frames being alternately shown and manipulated by the transputer is known as " *Double Buffering* ".

This results in the following Top of Screen values.

Figure a	Frame 1 {x,y = 0,0}	Top of Screen = 0	
	Frame 2 {x,y = 1024,0}	Top of Screen = 1024 * 1024	= 100000h
Figure b	Frame 1 {x,y = 0,424}	Top of Screen = 4 * 424 =	6A0h
	Frame 2 {x,y = 1248,0}	Top of Screen = 1024 * 1248	= 138000h
Figure c	Frame 1 {x,y = 0,424}	Top of Screen = 4 * 424 =	6A0h
	Frame 2 {x,y = 800,424}	Top of Screen = 1024 * 800 + 4 * 424 =	C86A0h
Figure c	Frame 1 {x,y = 0,0}	Top of Screen = 0	= 0h
	Frame 2 {x,y = 640,0}	Top of Screen = 1024 * 640	= A0000h
	Frame 3 {x,y = 1280,0}	Top of Screen = 1024 * 1280	= 140000h

Linear Addressing in the 8 bit/pixel mode

Its not as easy to illustrate the distribution of more than two images within the video memory, but it is possible to get optimal memory efficiency when using " *Double Buffering* " in this mode. Just make sure that the sum of all the images' pixels does not exceed 2 * 1024^2.

Example:

A SUN-Workstation's monitor has a resolution of 1152 * 900 pixels = 1,036,800 pixels. This requires 1012.5 kBytes of video memory, or 506.25 VRAM rows.
The first image will use up 507 VRAM rows (1014 kBytes) of video memory, leaving a total of 517 rows free for storing a second image.
The question is: how to determine the TopScreen register values for optimal efficiency ? Both images have to be stored as far to the " top " of the video memory as possible.

Equation for calculating the Top of Screen values for both frames:

1. Frame:

```
Top of Screen = 4 * Int(((2048*1024)-(1152*900))/2048)
              = 4 * 517 = 814h
```

2. Frame:

```
Top of Screen = 4 * Int(((2048*517)-(1152*900))/2048)
              = 4 * 10 = 28h
```

The second frame starts at VRAM row 10 and ends at the first quarter of VRAM row 516. The first frame starts at VRAM row 517 and ends at the first quarter of VRAM row 1023. This configuration leaves you with two 1536 Byte ($^{3}/_{4}$ VRAM row) large sections of video memory that can't be accessed, but the space at the beginning of the video memory (20 kByte) can now be added to the working memory.

Storing two frames with a horizontal resolution of more than 1024 pixels in the video memory simultaneously is only possible with " Linear Addressing " !

Programming the " *TopScreen Register* " to switch between several frames in the memory results in Hardware-Panning. If you know the frame coordinates you can use the equations 3.4a-c directly, depending on the monitor mode used.

3.3 The GDS-II's Address Space

The following lists the addresses of the various resisters. Both the hardware- and the OCCAM-word addresses are listed.

Hardware-addr.	OCCAM-word-addr.	Name	function
8000.0000	#0000.0000	DRAM - Start	2 MByte working memory
801F.FFFF	#0007.FFFF	DRAM - End	
8020.0000	#0008.0000	VRAM - Start	2 MByte video memory
803F.FFFF	#0009.FFFF	VRAM - End	
0000.0000	#2000.0000	Ident-register	bit0 = 0
		(read-only)	bit8 = 1: Event by VSYNC ("Frame" = active)
0000.00C0	#2000.0030	Reset-register	bit0 - 3: Link Reset Out
		(write-only)	bit4: Reset for G300
0008.0000	#2002.0000	PCS0 - Start	Chipselect 0 for DB-Slot A (1/2 MByte)
000F.FFFF	#2003.FFFF	PCS0 - End	
0010.0000	#2004.0000	PCS1 - Start	Chipselect 1 for DB-Slot A (1/2 MByte)
0017.FFFF	#2005.FFFF	PCS1 - End	
0018.0000	#2006.0000	PCS2 - Start	Chipselect 0 for DB-Slot B (1/2 MByte)
001F.FFFF	#2007.FFFF	PCS2 - End	
0020.0000	#2008.0000	PCS3 - Start	Chipselect 1 for DB-Slot B (1/2 MByte)
0027.FFFF	#2009.FFFF	PCS3 - End	
0028.0000	#200A.0000	PCS4 - Start	Chipselect 0 for DB-Slot C (1/2 MByte)
002F.FFFF	#200B.FFFF	PCS4 - End	
0030.0000	#200C.0000	PCS5 - Start	Chipselect 1 for DB-Slot C (1/2 MByte)
0037.FFFF	#200D.FFFF	PCS5 - End	
0038.0000	#200E.0000	CS-CLUT - Start	ChipSelect for 12 bit-expansion (DBCLUT)
003F.FFFF	#200F.FFFF	CS-CLUT - End	
0040.0000	#2010.0000	Base address G300	
0040.0000	#2010.0000	CLUT - Start	G300's Internal CLUT (256 Words á 24 bit)[22]
0040.03FC	#2010.00FF	CLUT - End	
0040.0500	#2010.0140	Mask register	Read/write
0040.0580	#2010.0160	Control Register	Read/write
0040.0600	#2010.0180	Top of Screen	Read-only
0040.0680	#2010.01A0	Boot Location[23]	Write-only !
0040.0484	#2010.0121	Datapath Register - Start[24]	
0040.04B0	#2010.012C	Datapath Register - End	

[22] The tranputers addresses 256 words á 32 bit but only the lowest three bytes of every word are used !

[23] Once programmed " *Boot Location* " can't be changed.

[24] Access to Datapath Registers only possible with enabled VTG !

3.4 The G300Registers

The following chapters will describe the G300's various registers. The chapter " *Datapath Register* "[25] is more or less a " handbook " for the more experianced user. Connections between monitor parameters, monitor types and resolution will not be explained in this section. " Beginners " who do not yet fully understand the details, but still want to study and/or modify (- mess around with) the monitor drivers should read the chapter " *Programming The Video Timings* "[26].

3.4.1 Boot Location (#201001A0)

The " **read-only** " register " *Boot Location* " is the first to be programmed after resetting the system. Bits 0-4 at the address #201001A0 (0x400680) represent the multiplication factor for the internal PLL, thereby determining the " *Pixel Clock* ". The PLL's input frequency is set at 5 MHz and can only be changed by installing a different oscillator.
Bit 5 is used to tell the G300B[27] whether the internal PLL is active (bit 5 = 1) oder not (bit 5 = 0).
All other bits have to be set to zero.

8 bit/pixel Mode:

" Clock Input " for G300 jumpered to 5 MHz (J5).

(Jumper J10 ist not set)
The " *Pixel Clock* " can be programmed in steps of 5 MHz; Boot Location gives you the neccessary faktor.

```
VAL  PLL           IS 22          :
VAL  Boot_Location IS PLL \/ #20  :  -- set bit 5 since the G300
                                     -- works in the PLL mode
PLACE Boot_Location AT #201001A0  :
```

In this example, the " *Pixel Clock* " is set at 22 * 5 MHz = 110 MHz.

[25] See chapter 3.4.5
[26] See chapter 6
[27] Bit 5 is always set to zero when using a G300A

Pixel clock rates of less than 32 MHz may cause problems, since the first four pixel of the video memory will not show up on the monitor.

" Clock Input " for G300 jumpered to 32 MHz (J5).

(Jumper J10 is closed)
The " *Pixel Clock* " is set to 32 MHz, i.e. the G300's " *Clock Input* " is jumpered to 32 MHz and the PLL multiplication factor is set to zero. Bit 5 has to be set to zero too.

Boot Location ≡ 0

24 bit/pixel Modus

The " *Pixel Clock* " is set at 32 MHz, i.e. the G300's " *Clock Input* " is jumpered to 32 MHz and the PLL multiplication factor is set to zero.

Boot Location ≡ 0

Once programmed, the register " Boot Location " cannot be changed. You'll have to reset the G300 via the " Reset Register ".

The register " Boot Location " cannot be read-out !

3.4.2 Control Register (#20100160)

The " *Control Register* " is a read/write register and can be programmed at the address #20100160 (0x400580). The various bits represents a number of functions which will be explained in this chapter.

Do not attempt to program the " Control Register " before the register " Boot Location " has been programmed.

The G300B's " Control Register "

bit	Function	Comments	
31-24	Not wired	Don't care when reading the Control Register	
23	Blank function switch	Write zero for GDS-II	
22	Pixel repeat	Write zero	
21	Interlace standard	1 = CCIR interlace format (europe)	0 = EIA interlace format (US)
20	Address step control	Write Zero for GDS-II	
19	Address step control	1 = interlaced	0 = non interlaced
18-17	Bits per pixel	Set pixel port to required pixel depth	
16	Blank I/O	Write Zero for GDS-II	
15	Turn off blanking	1 = blanking disabled for test	0 = blanking enabled
14-13	Reserved	Write zero	
12	Black level	Selects blanking level	0 = Blank = Black level
11-9	Delay value	Write zero for GDS-II	
8	Pixel port mode	1 = mode 2 (13-24 bit/pixel)	0 = mode 1 (8 bit/pixel)
7	Micro port mode	Write zero for GDS-II	
6	Reserved	Write zero	
3	Analogue video format	1 = video only	0 = video and composite sync
4	Digital sync format	1 = seperate sync.	0 = mixed sync.
5	Frame flyback pattern	1 = plain synchronizing waveform	0 = tesselated synchronizing
2	Device operating mode	Write zero for GDS-II	
1	Screen format	1 = interlaced	0 = non interlaced
0	Enable VTG	1 = VTG running	0 = VTG disabled

Details concerning the various bits:

Bit 0 = 0 stops the Video Timing-Generator (VTG)

Bit 0 = 1 starts the Video Timing-Generator

Setting or deleting this bit will start or stop the Video Timing-Generator (VTG). You usually finish initialising the G300 by setting this bit.

```
PLACE Control AT #20100160:
                 .
SEQ
  Control := Control /\ #FFFFFFFE
```

This example " **should** " stop the Video Timing-Generator, but sometimes you will have problems with this program sequence. If the VTG is stopped when a " *Transfer cycle* " is pending, the G300 may " hang ". In this situation it is not possible to reset the system via " *Reset PAL* " because the G300 blocks the " *Transputer Bus* "; you have to use the " *Master Reset*[28] " or " *Power-On-Reset* ".
If you want to stop the VTG (neccessary when accessing the " *Datapath Registers*[29] ") you have to alternatives.

a.) Reset the G300 using the " *Reset Pal* " and reprogram the chip.[30]

b.) Synchronize your access to the " *Control Register* " with the " *Vertical Blank Pulse* ". During VBlank no transfer cycle will take place.[31]
Chapter 7.9 gives you an example for disabling the VTG and reading some of the " *Datapath Registers* ".

[28] See chapter 2.4

[29] See chapter 3.4.5

[30] This is what INMOS suggests. It works, but what happens if you don't know the initialisation parameters .....

[31] Here is what I suggest....

Bit 1,19 = 0 The GDS - II is working non-interlaced

Bit 1,19 = 1 The GDS - II is working interlaced

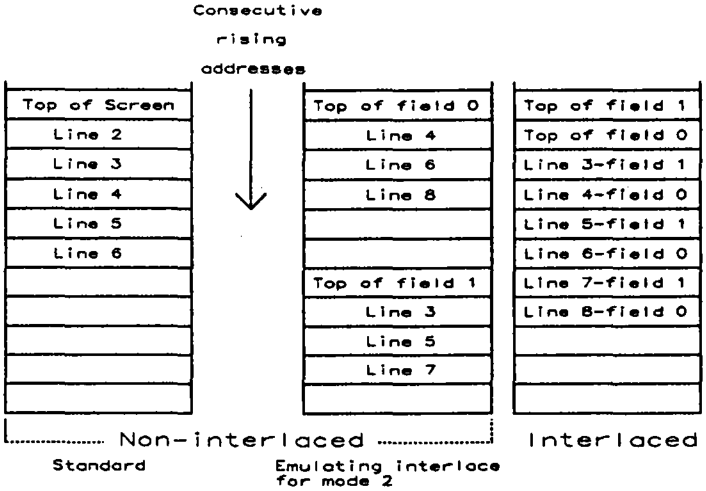


Figure 3.11 video memory organisation

When using the "*Interlaced Mode*", the complete video image is put together out of two half images. The "*Pixel Clock*" only has to be half as big in order to get the (seemingly) same resolution, but image repetition rate will sink. Large areas have the same repetition rate as in the "*Non-Interlaced Mode*", but edges and pixel wide lines will flicker at half the repetition rate.

You don't have to change the storage of your image in the video memory when choosing the interlace mode since the G300 increments the "*Row-Address*" for every field by 2 ! The framestore format for interlace is identical to that for non-interlace. Address ordering depends on the standard selected. CCIR scans even lines first, NTSC scans odd lines first[32].

Don't forget to divide the value stored in the datapath register "*VDisplay*" by 2.

The " Interlace Mode " is only available in the " Row-Oriented Addressing Mode ".

If you try to use the interlace mode in the "*Linear Addressing Mode*" (e.g. mode 2 with more than 512 pixels horizontal resolution) G300 will fail to generate the transfer address. If you want to realize a non-interlace system in mode 2 you can do it the other way round. Set the G300 to non-interlace screen format and store your two half-frames at two different areas of the video memory. With every received VBlank[33] you have to change the "*TopScreen Register*" so that you see alternating two different parts of the video memory.

Bit 2 = 0 Always set bit 2 to zero.

[32] Refer to bit 21; "*Control Register*"

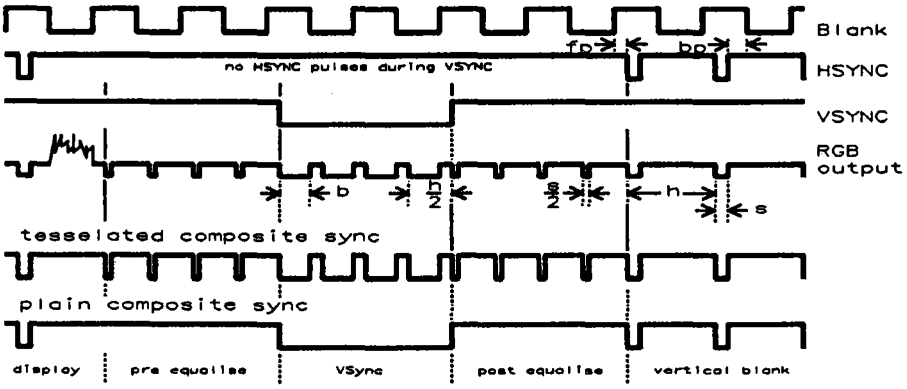
[33] See chapter 7.8 how to use the event channel.

Bit 3 = 0 A " *Tesselate Composite Sync* " is delivered via the HSYNC output

Bit 3 = 1 A " *Plain Composite Sync* " is delivered via the HSYNC output

A VSYNC will always be sent to the VSYNC output, regardless of how bits 3, 4 and 5 . The HSYNC output will only change if bit 4 is set to zero. Check the table for a complete list.

Control bits			output	
G300				
5	4	3	VSYNC	HSYNC
x	0	0	VSync	Tesselate Comp. Sync.
x	0	1	VSync	Plain Comp. Sync.
x	1	0	VSync	HSync[34]
x	1	1	VSync	HSync[34]



s = horizontal sync. pulsewidth
h = linetime = 1/horizontal scan frequency
b = broadpulse width = linetime/2 - fp
fp = horizontal front porch
bp = horizontal backporch

Figure 3.12 " Tesselated/Plain Composite Sync. "

Bit 4 = 0 " *Mixed Synchronisation* "; a Composite SYNC is delivered via the HSYNC output (" *Plain Comp. Sync* " for bit 3 = 1 - " *Tesselate Comp.Sync* " for bit 3 = 0).

Bit 4 = 1 HSYNC[34] and VSYNC are delivered separately

Bit 4 has no influence on the " *Composite Sync* " modulated on to RGB output (bit 5 = 0 !).

[34] No HSYNC pulses will be generated during VSYNC

Bit 5 = 0 " *Composite Sync* " is modulated to the RGB-output

Bit 5 = 1 Sync-signals will only be delivered via HSYNC, VSYNC outputs

Most analog monitors are synchronized by a " *Composite Sync* " modulated to the RGB-outputs. Here, a few typical monitors:

EIZO FLEXSCAN 8060, 9070, 9500
NEC MULTISYNC GSII, XL
SONY GDM 1601, 1602, 1901, 1950, 1952

When in doubt, try reading the instructions. Most monitors can be controlled by either a modulated " *Composite Sync.* " or a separate Sync-signal.

The G300's outputs HSYNC and VSYNC are always " active low "[35] (negative Sync.). Its not possible to change the polarity.

The external SYNC signal's polarity will usually not matter, since most monitors will switch automatically. (remember to check your monitor's manual)

Bit 6, 7 = 0 Always set bits 6 and 7 to zero !

Bit 8 = 0 The G300 will work in mode 1; i.e. 2 ,4 or 8 bit/pixel mode[36].

Bit 8 = 1 The G300 will work in mode 2; i.e. 24 bit/pixel mode (13 bit/pixel mode with DB-CLUT expansion).

Changing between the two modes per software is not enough, jumpers J5, J6, J7 and J10 have to be changed accordingly too.[37]

Bit 9-15 = 0 In normal use, bits 9-15 should be set to zero.

[35] HSYNC and VSYNC are active high, when using the G300A chip.

[36] 1, 2 and 4 bit/pixel are only available with the G300B chip

[37] Jumper 8 has to be changed too, when using the G300A

The following bits 16 - 23 can only be set (and read) in a G300B. If you are using a G300A - set them to zero !

Bit 16 = 0 In normal use, bit 16 should be set to zero

Bit 17, 18 The G300B offers the option to store more than 1 pixel in each word. The following table tells you how to do this.

Control			Address
bits		mode	of CLUT
18	17		(in Hex)
0	0	1 bpp	#00 - #01
0	1	2 bpp	#00 - #03
1	0	4 bpp	#00 - #0F
1	1	8 bpp	#00 - #FF

(bpp = bits per pixel)
(Offset of CLUT-address = #2010.0000)

The last column shows the part of the CLUT that is used for programming. In the **8 bpp** mode you can choose between 256 different colours - the whole CLUT is used (256 words with 24 Bit each).
In the **2 bpp** mode 4 colours out of a 2^{24} palette can be chosen. Each pixel is 2 bits wide but you need only 4 of the CLUT's memory cells[38].
Using the **1 bpp** mode you can store 8 pixels in every byte; only two different colours are available then.

Bit 19,1 = 0 The GDS - II is working non-interlaced
Bit 19,1 = 1 The GDS - II is working interlaced

Bit 20 = 0 In normal use, bit 20 should be set to zero

[38] The first four memory cells with the addresses #20100000 - #20100003

Bit 21 = 0 Output signal performs the " *EIA-343 studio television standard* "

Bit 21 = 1 Output signals performs the " *CCIR studio television standard* ".

Bit 21 is only acted on when using the G300 in the " *Interlace Mode* ".
The EIA-343 standard is used in the US and well known as the " *NTSC - standard* ", whereas the CCIR standard or " *PAL - standard* " is used in europe.
CCIR scans even lines first (even field), NTSC scans the odd lines first (odd field).

Bit 22 = 0 In normal use, bit 22 should be set to zero

Bit 22 was reserved for implementing a " *Hardware Zoom* " option. This mode didn't work on the first G300B chips[39]. Instead of correcting this bug, INMOS now tells us to program it to zero.
Setting this bit is not dangerous, but gets interesting and amazing results.....

Bit 23 = 0 In normal use, bit 23 should be set to zero

Bit 24-31 = 0 Bits 24-31 are not wired and will be ignored.

[39] G300B01

3.4.3 Mask Register (#20100140)

The " *Mask Register* " can be used to mask incoming pixels. The value stored in the " *Mask Register* " is combined with every pixel through an AND-function. If you forget to program the " *Mask Register* " after resetting the G300[40], the entire monitor will assume the colour listed in the first CLUT word (address #2010.0000). If the CLUT's still empty, then the screen will be , too (dark).

3.4.4 Top Of Screen (#20100180)

The register " *Top Screen* " is explained (functions & programming) in full detail in chapter 3.2. There are two methods of accessing this register. The first method is by the name of " *Top of Screen* " at address #2010.0180 and, the second, by the name of " *Line Start* " at address #2010.012A (see " *Datapath Register* "[41]). The difference is, that access via the register " *Line Start* " is only possible if the Video Timing-Generator (VTG) is inactive, whereas writing into the register " *Top of Screen* " is only possible if the VTG is running.... .

As a consequence, the basis address of an image has to be programmed into the " *Line Start Register* ", when the system is being initialised. After the VTG is running, however, it has to be modified by way of the register " *Top Screen* " (panning, Double Buffering, etc).

It's a good idea to synchronize access to the " *Top Screen Register* " with the " *Vertical Blank Pulse* " to prevent flickering on the screen.[42]

The register at the location of " Top Screen " is not readable !

[40] All registers, even the " *Mask Register* ", are set to zero after a reset.

[41] See chapter 3.4.5

[42] See chapter 7.8

3.4.5 Datapath Register (#20100121 - #2010012C)

During initialising (running VTG prevents access), the " *Datapath Registers* " are fed all parameters necessary for the G300 to generate " *video timings* ". The timing parameters for some of the more widespread monitor models are listed in the appendix and also delivered with the GDS-II in the form of test programs on a diskette. There's an example in chapter 5.1 explaining how to figure out the correct register values from the monitor parameters supplied by the monitor manufacturer.

All values are given in periods of " *Serial Clock* " or in half-lines. Note:

Serial clock = $^1/_4$ Pixel Clock = $^1/_4$ Dot Clock

Example:

A monitor requires a Pixel Clock of 80 MHz. This means that all time values will be stated as multiple of 50 ns.

Register	Address	Unit	Note
HalfSync	#121	SClk	
BackPorch	#122	SClk	
Display	#123	SClk	
ShortDisplay	#124	SClk	
BroadPulse	#125	SClk	
VSync	#126	Half-line	
VBlank	#127	Half-line	
VDisplay	#128	Half-line	
Linetime	#129	SClk	
LineStart	#12A	SClk	chapter 3.4.4
MemInit	#12B	SClk	chapter 3.2.1.3
TransferDelay	#12C	SClk	chapter 3.2.1.3

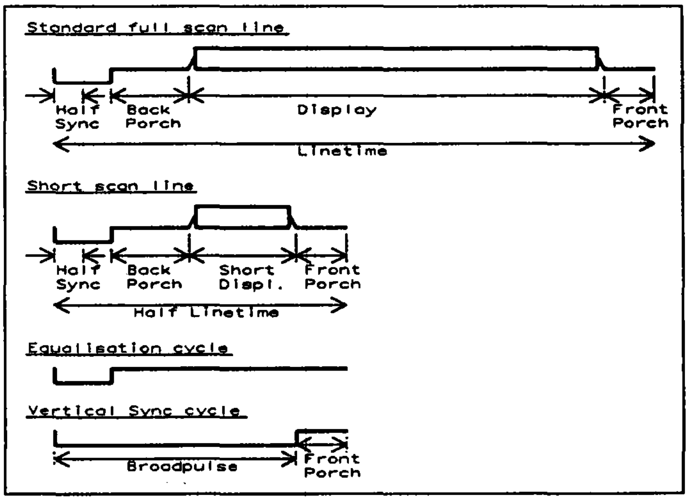


Figure 3.13 Horizontal Timing

This figure demonstrates the different cycles which compose the video timing.

" *Standard Full Scan Line* " represents a standard row whose timing is determined by the registers **HalfSync**, **BackPorch**, **Display** and **Linetime**. (Linetime has to be an even number.) There's no need for a register named FrontPorch since its value can be calculated using the other values. (FrontPorch = Linetime - Display - Backporch - 2 * HalfSync).

note the following restrictions: **FrontPorch < ½ Linetime**
2 * HalfSync + BackPorch < ½ Linetime

" *Short Scan Line* " is generated in the " *Interlaced Mode* " at the beginning of the first half image and at the end of the second half image. Even though the value of the register **ShortDisplay** is a direct result of the other registers, it still has to be programmed into the appropriate register separately. This register must be programmed **even when the G300 is working in the " Non-Interlaced Mode ".**

```
ShortDisplay = ½ Linetime - 2 * HalfSync - FrontPorch - Backporch
FrontPorch   = Linetime - 2 * HalfSync - BackPorch - Display
ShortDisplay = Display - ½ Linetime
```

" *Equalisation Cycle* " is generated prior to and after the actual " *Vertical Sync* " is generated, and is just as long as the " Short scan line ".

" *Vertical Sync Cycle* " This is the VSYNC. You need the length of " **BroadPulse** " in order to generate this cycle. The equation for calculating the value of the register of that name is:

```
BroadPulse = ½ Linetime - FrontPorch
BroadPulse = Display + 2 * HalfSync + BackPorch - ½ Linetime
```

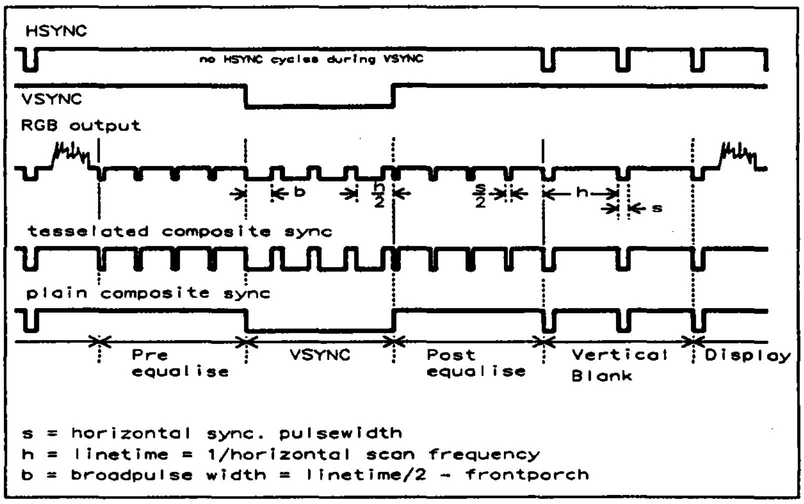

HSYNC
no HSYNC cycles during VSYNC
VSYNC
RGB output
b
h/2
s/2
h
s
tesselated composite sync
plain composite sync
Pre equalise
VSYNC
Post equalise
Vertical Blank
Display
s = horizontal sync. pulsewidth
h = linetime = 1/horizontal scan frequency
b = broadpulse width = linetime/2 - frontporch

Figure 3.14 Vertical Timing

This figure demonstrates the manor in which the various rows are put together. The number of " *Standard Full Scan Lines* ", i.e. the vertical pixel resolution, is listed in the register **VDisplay** as a multiple of half-lines.

After the last image row, the " *Pre Equalise Cycle* " (also known as the " *Vertical Backporch* ") starts. This has the same length as the **VSYNC**. An equally long " *Post Equalise Cycle* " is attached to the **VSYNC**. Then, at last, comes the actual **VBlank**.

The term VBlank is used a little different in this context (see below).

Standard term	Definition for G300 use
Vertical Blank	= 3 * VSync + VBlank
Vertical Backporch	PreEqualise = VSYNC
Vertical Frontporch	PostEqualise + VBlank = VSYNC + VBlank

Transfer Delay

" *Transfer Delay* " determines the time span the G300 will need for a complete " *Transfer cycle* ". It is only determined[43] by the GDS-II and does not depend on the monitor used..

Always pay attention to the following restrictions for " *Transfer Delay* " if you want your G300 to function properly.

TransferDelay < BackPorch
TransferDelay < ShortDisplay

That means you can't make the BackPorch as short as you want.

For the sake of being complete - a table for 8 bit/pixel and 17.5 MHz " *Processor Clock* ". The smallest possible value you can store in BackPorch is:

$$\textbf{BackPorch}_{\textbf{Min}} = \textbf{TransferDelay + 1}$$

PixelClock	TrDelay
30 MHz	10
35 MHz	11
40 MHz	12
45 MHz	13
50 MHz	14
55 MHz	15
66 MHz	16
65 MHz	16
70 MHz	17
75 MHz	18
80 MHZ	19
85 MHz	20
90 MHz	21
95 MHz	22
100 MHz	23
105 MHz	24
110 MHz	24

[43] See chapter 3.2.1.3

MemInit

The sum of " *MemInit* " and " *Transfer Delay* " corresponds to the length of the VRAM's shift register in the " *Linear Addressing Mode* ". The sum is otherwise, i.e. " *Row-Oriented Addressing* ", identical to the number of horizontal pixels.

Note:

Access to the Datapath Registers is only possible, when the Video Timing Generator is not running !

If you want to check one or more of the datapath registers, it is necessary to stop the VTG by setting bit 0 of the Control Register to zero.[44]

3.4.6 CLUT (#20100000)

The CLUT is located at address #20100000 (0x400000). If you want to modify the CLUT when the VTG is running refer to chapter 7.8.2.

Note for " C " programmers:

```
red information, 1st word :     0x400000
green information, 1st word :   0x400001
blue information, 1st word:     0x400002
red information, 2nd word :     0x400004
red information, 3rd word :     0x400008
```

[44] It's more complicated as it sounds. Please check chapter 7.9.

3.4.7 Ident Register (#20000000)

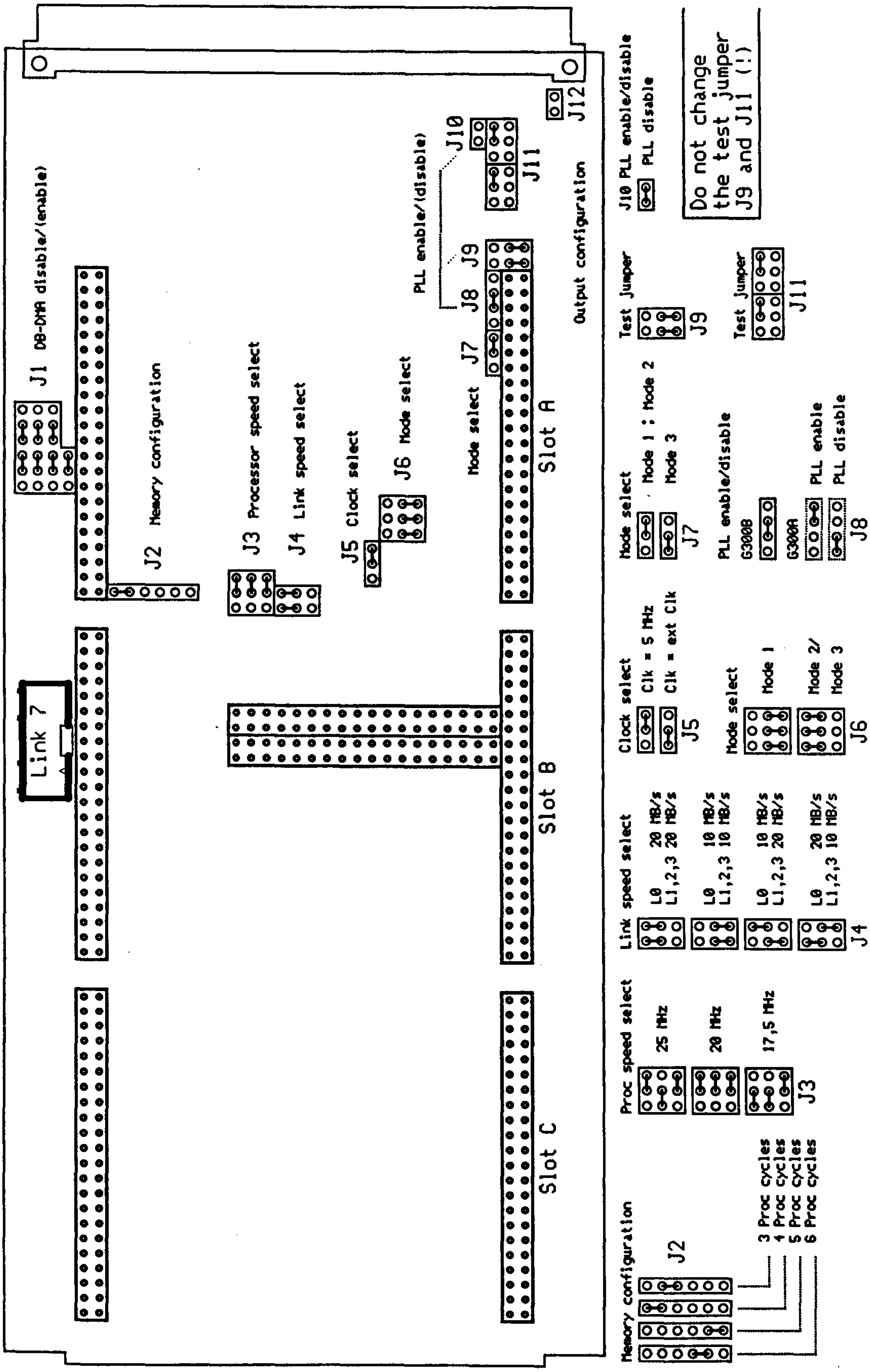
The " *Ident Register* ". This register can only be read, and is in contrast to other Parsytec boards (MTM 2-10, MTM 2-11), limited in its functions. If an "*Ident Register*" is present on the board, you can use it to identify a given board within a transputer network. There is, however, an easier way to do this on a GDS-II. Write a series of random numbers into the CLUT and then read it out. Compare what you got out to what you wrote in and check for write/read errors. If there are none then you are dealing with a GDS-II since no other Parsytec board has memory or register placed at the address #20100000.

Bit 0 in the " Ident-Register " is always zero.
This makes sure that it will be compatible to the INMOS " TDS-development system ".

Bit 8 in the " Ident-Registers " shows you if the signal VSYNC is active at the moment.
This tells you if an event is caused[45] by a connected Daughterboard or by a VSYNC.

[45] See chapter 7.8 " *Synchronizing With The Vertical Blank* "

4. Jumper layout:



J1: Configuration Of Slot A

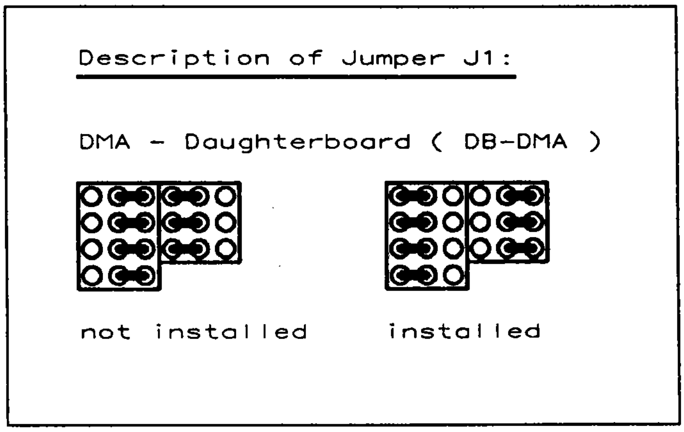


Figure 4.1 Jumper J1

Jumper J1 determines how slot A can be used..

DB-DMA Disable:

- Slot A can be used for either a DBI-x module (I/O Daughterboard) or a DBT-x Module (Processor Daughterboard).

DB-DMA Enable:

- Slot A can be used for a DBT-x Module (Processor Daughterboard) or for a DB-DMA (DMA Daughterboard).

J2: Memory Configuration

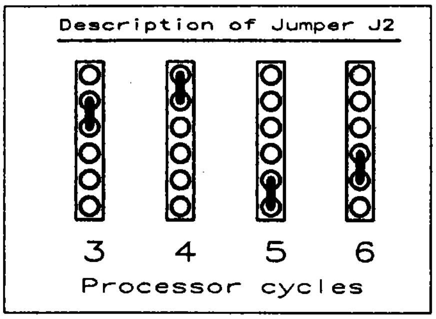


Figure 4.2 Jumper J2

Jumper J2 determines the number of " *Processor Cycles* " per memory access. The possible values are based on the " *Processor Clock* " and the type of RAMs used. The following table lists possible resulting cycle times.

Proc cycles	3	4	5	6
17.5 MHz PCO	(171 ns)	**229 ns**	286 ns	343 ns
20 MHz PCO	(150 ns)	**200 ns**	250 ns	300 ns
25 MHz PCO	(120 ns)	(160 ns)	**200 ns**	240 ns
30 MHz PCO	(100 ns)	(133 ns)	(167 ns)	**200 ns**

PCO = Processor Clock Out

The cycle duration is one of a RAMs characteristics and is influenced by its access time. Sine the GDS-II's video memory is equipped RAMs with access times of 100 ns and cycle times of 190 ns one should try to stick to the times listed in boldface type.

Using faster RAMs for the working memory is possible, but not very practicable as long as the video RAMs are only available with access times of 100 ns. **Both the working- and the video memory have to have the same memory configuration.**

Its unwise to try to use shorter cycle time than 190 ns. Even if there are no recognizable memory errors, the higher power losses will cause the RAMs to age faster and the (resulting) higher temperatures may damage other components.

J3 Processor Speed Select

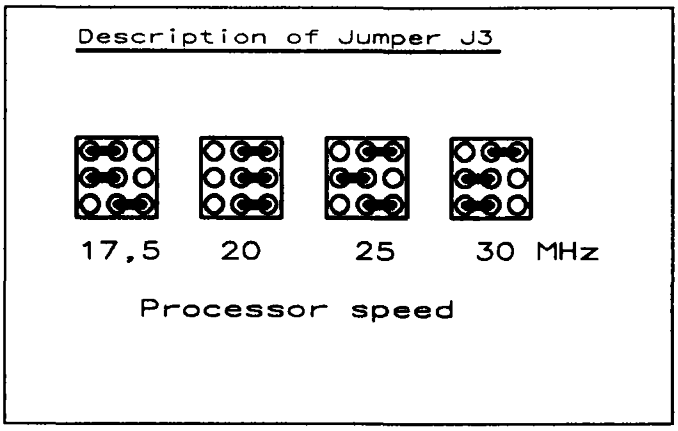


Figure 4.3 Jumper J3

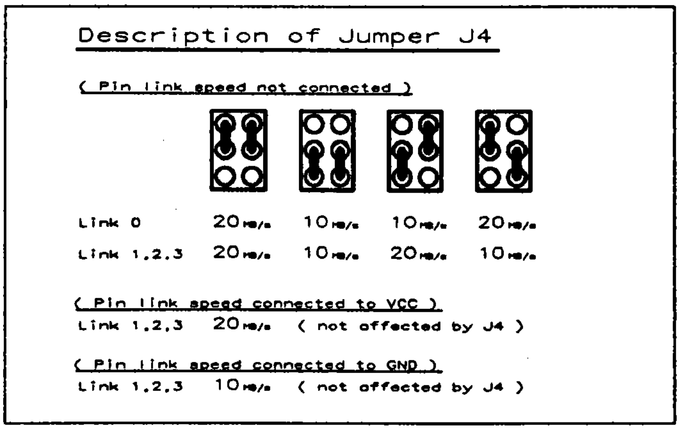
J3 sets the transputers " *Processor Clock* ". The transputer is constantly supplied with a 5 MHz input frequency from which it generates its own " *Processor Clock* " (using a PLL circuit). Setting jumper J3 determines the factor with which the 5 MHz input frequency is multiplied.

The maximal " *Processor Clock* ", allowable for a transputer is printed on its housing; IMST800D G25S means : INMOS T800; revision D; 25 MHz (the letter S signifies the housing type).

Never drive a transputer at a " Processor Clock " greater than the value printed on its housing.

Of course its possible to slow a transputer down with a lower frequency (e.g. running a 25 MHz transputer at 17.5 MHz Processor Clock), but this rarely makes sense. You can run a GDS-II with 25 MHz " *Processor Clock* " even though all other network transputers are running at 20 MHz or even 17.5 MHz. Link communication between processors will function trouble-free as long as they are all supplied with an identical input frequency (5 MHz).

J4 Link Speed Select



Description of Jumper J4

(Pin link speed not connected)

Link 0	20MB/s	10MB/s	10MB/s	20MB/s
Link 1,2,3	20MB/s	10MB/s	20MB/s	10MB/s

(Pin link speed connected to VCC)

Link 1,2,3 20MB/s (not offected by J4)

(Pin link speed connected to GND)

Link 1,2,3 10MB/s (not offected by J4)

Figure 4.4 Jumper J4

By setting jumper J4 you can determine wether the Link data transfer rate will be 10 or 20 Mbit/s. 5 Mbit/s will not be supported. The data transfer rates for Links 1, 2 and 3 cannot be set separately. The "*Link Speed*" signal represents a new development on Parsytec boards. This signal can be accessed on the (96-way) VG connector's 26B pin and used for external Link speed adjustments. The data transfer rates of Links 1, 2 and 3 are set to 20 Mbits/s if this pin is set to VCC, and to 10 Mbits/s if the pin is set to GND. The transfer rate can only be set via J4 if this pin is not connected Link 0's transfer rate is not affected by the " *Link Speed* " pin and can only be changed by resetting J4. The " *Link Speed* " pin is an expansion that will be available in future systems.

Up till now, all backplanes have been delivered with a high resistance pin 26B (the pin is not connected). The Link speed is therefore determined by the J4 setting.

J5 G300 Clock Select

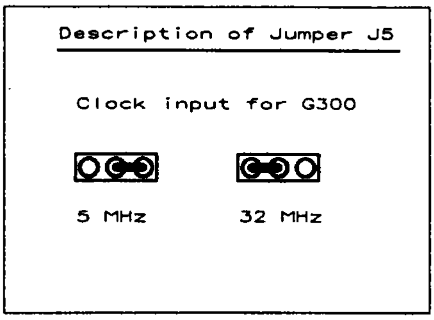


Figure 4.5 Jumper J5

The GDS-II is equipped with two frequency generators. One of these generates the 5 MHz signal used by the transputer to time its work cycles. This 5 MHz can also be delivered to the G300 by setting J5 appropriately. If this is the case, the G300 will use the signal to generate its own internal "*Pixel Clock*" which can then be programmed in steps of 5 MHz.[1].

J5, however, can also be used to rout the second frequency generator's signal to the G300 (usually 32 MHz). This is done in modes 2 and 3 (24 bit/pixel & 13 bit/pixel).

J6, J7 Mode Select

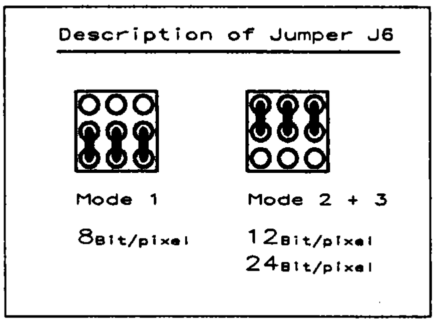


Figure 4.7 Jumper J6

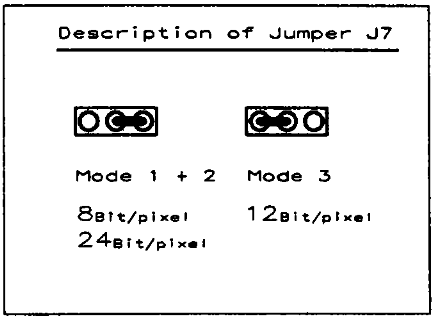


Figure 4.8 Jumper J7

Use jumper J6 for internal switching. In the G300's mode 1 there are 8 bit/pixel whereas in mode 2 (at a reduced pixel resolution) you get 24 bit/pixel.

As the G300 does not differentiate between modes 2 and 3 (13 bit/pixel) you will have to " tell " it by setting J7 appropriately.

[1] Register "*Boot Location*" at address #201001A0 ; see chapter 3.4.1

J8, J10 Mode Select

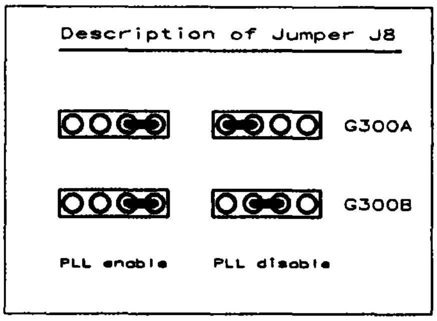


Figure 4.9 Jumper J8

J8 is only used with the G300A chip. The G300A chip has two different " *Clock Inputs* " and J8 determinates which input is used.

When using the G300B it is not neccessary to change jumper J8 when setting your GDS-II to another mode.

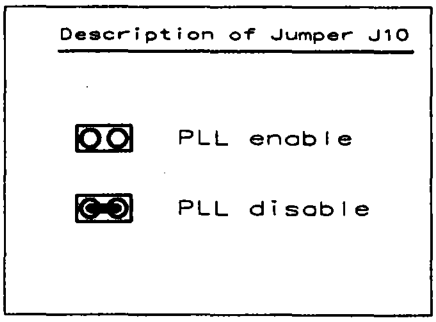


Figure 4.10 Jumper J10

J10 is used to short circuit the PLL's external condensator. Every time you use the external 32 MHz oscillator for generating the " *Pixel Clock* " (mode 2: 24 bit/pixel and mode 3: 13 bit/pixel) J10 you have to set.

One of the following chapters will contain the jumper setting for various modes in detail[2].

J9, J11

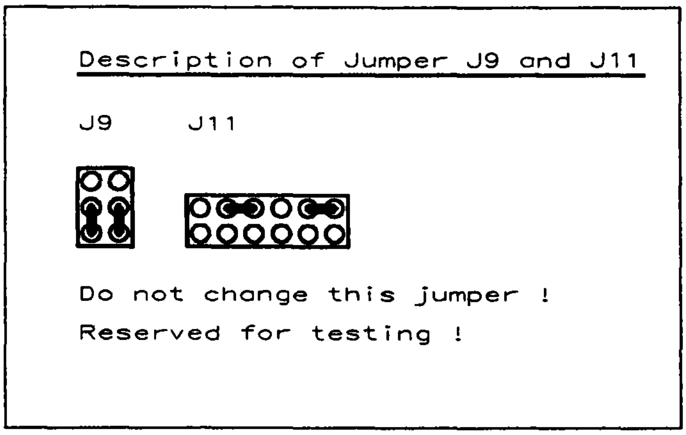


Figure 4.11 Jumper J9 and J11

Jumpers J9 and Jumper J11 are only used when starting the GDS-II for the first time or when testing it. Please, try not to change their settings.

[2] Chapter 9.1 " *Appendix* "

J12 Output Configuration

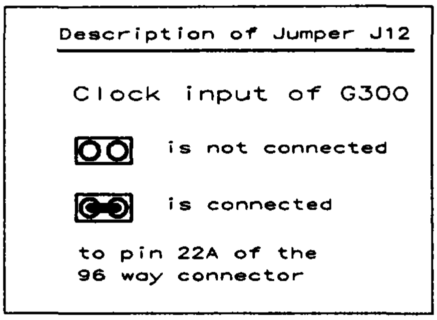


Figure 4.12 Jumper J12

By setting jumper J12 you can rout the " *Video Clock* " (5 MHz or 32 MHz depending on the setting of J5) via the Link backplane to other systems (take a look at the video output's pin layout)[3].

[3] See chapter 2.2 and 8.8 - signal HClk

5. Daughterboard Connectors

The following table lists, which additional modules can be inserted into which slots.

Slot C	Slot B	Slot A
DBI-x	DBI-x	DB-DMA
---- DB-CLUT ----		DB-DMA
---- DB-CLUT ----		DBT-1
DBI-x	DBI-x	DBT-1
DBI-x	---- DBT-4 ----	

DB-DMA:

This is an additional board that can **only** be used together with a GDS-II. It may only be inserted into slot A, which means that you will have to configure the slot accordingly with jumper J1. The DB-DMA is equipped with a second transputer that has no external working memory[1]. This transputer also (together with the base board transputer) has access to the video memory, making it possible to transfer data to the video memory via a total of eight Links. The DB-DMA is connected to Links 4 - 7 (see figures 2.0 and 2.1).

DB-CLUT:

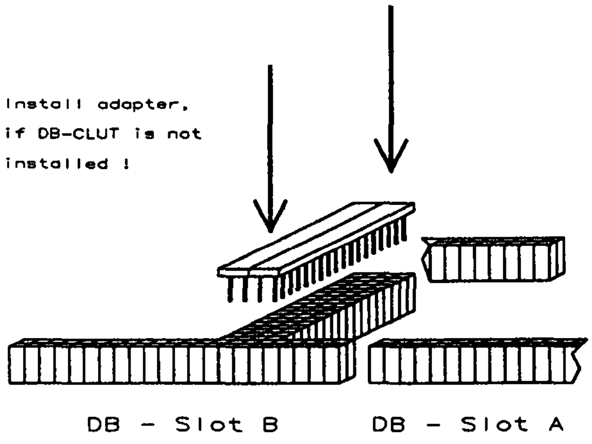


Figure 5.0 Installing an adapter board if a DB-CLUT is not used

Another additional board that can **only** be used on a GDS-II. It has to be inserted into both slots B **and** C. Since a DB-CLUT needs signals that are not obtainable on a standard slot connector, slot B is equipped with an additional (72-way) connector. An adapter is delivered together with the GDS-II especially for this connector, and

[1] From revision 1.1 on also with 1 MByte or 4 MByte working memory.

should always be used when not using a DB-CLUT. The adapter also has to used if a DBI-x is inserted in slot C or B.

A DB-CLUT makes an additional mode (13 bit/pixel) possible. In this mode, two pixels are stored per word, thus representing a compromise between high colour resolution (24 bit/pixel) and high processing speeds (8 bit/pixel).

DBT-1:

This is a standard transputer node with a 1 MByte RAM[2] that may only be inserted in slot A. Its Links are connected to the GDS-II's Links 4 - 7. Links 4, 5 and 7 are accessible on the Backplane, Link 6, directly on the GDS-II (see Figures 2.0 and 2.1).

DBT-4:

Just like the DBT-1, except that it has a 4 MByte RAM[3]. It has to be inserted into both slots A and B, since it needs two slot connectors and only slot A has the necessary Link connections.

DBI-x:

DBI-x stands for DBI-1, DBI-2, etc. Any time you insert a DBI-x module in slot A, you will have to set jumper J1 appropriately.

[2] 4 MByte working memory if equipped with 4 Mbit DRAMs.

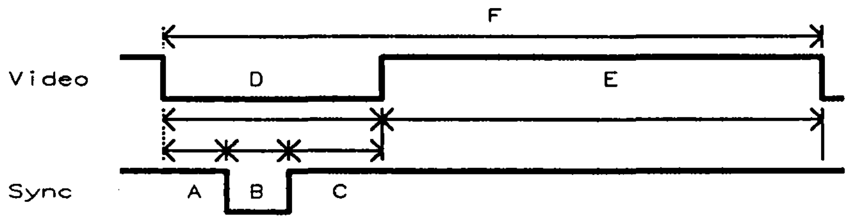
[3] 16 MByte working memory if equipped with 4 MBit DRAMs.

6. Calculating a Video Timing:

Establishing a video timing is explained on hand of the following example using an EIZO 9070H-S.

The appropriate manual page copied in figure 5.0.

Timing Charts



Preset Timing			A	B	C	D	E	F
Enhanced Graphics	H	us	-0.1	4.9	1.6	6.4	39.4	45.8
fH:21.85kHz	V	ms	0.05	0.6	0.1	0.7	16.0	16.75
Personal System2	H	us	0.6	3.8	1.9	6.3	25.48	31.78
fH:31.5kHz,350Lines	V	ms	1.2	0.06	1.9	3.16	11.11	14.27
Personal System2 (Analog)	H	us	0.6	3.9	1.9	6.3	25.48	31.78
fH:31.5kHz,350Lines	V	ms	0.4	0.06	1.1	1.56	12.71	14.27
Personal System2 (Analog)	H	us	0.6	3.9	1.9	6.3	25.48	31.78
fH:31.5kHz,350Lines	V	ms	0.35	0.06	1.0	1.41	15.26	16.67
CAD.CAM use (Analog)	H	us	0.68	1.77	3.04	5.49	14.62	20.11
fH:49.8kHz,768Lines	V	ms	0.48	0.12	0.60	1.2	15.44	16.64

Figure 5.0 The EIZO 9070's timing diagram

We want to use the monitor at its highest reolution (1024 * 768), so only the bottom line will be relevant.

Line time (F): t_z = 20.11 us

Display length (E): t_d = 14.62 us

Pixel Clock: f_c = 1024 / 14.62 us = 70.04 MHz

Horizontal resolution = 1024 pixel

The " *Pixel Clock* " is set at 70 MHz. This results in:

Boot Location = 14

The horizontal resolution is 1024 pixel.

Display = ¼ * 1024 = 256

Linetime = ¼ * t_z * f_c = ¼ * 20.11 us * 70 MHz = 351.9
Linetime = 352

HSync (B): t_h = 1.77 us
BackPorch (C): t_{bp} = 3.04 us

HalfSync = ½ * t_h * ¼ * f_c = $^1/_8$ * 1.77 us * 70 MHz = 15.5
HalfSync = 15

BackPorch = t_{bp} * ¼ * f_h = ¼ * 3.04 us * 70 MHz = 53.2
BackPorch = 53

The vertical resolution is 768 lines.

VDisplay = 768 * 2 = 1536

VSync (B): t_v = 0.12 ms
VBackPorch (C): t_{vbp} = 0.6 ms
VFrontPorch (A): t_{vfp} = 0.48 ms
Vert. Blank (D): t_{vb} = 1.2 ms

It is not possible to adjust " *Vertical Frontporch* " and " *Vertical Backporch* " separately. The G300 only allows you to alter the length of the " *Vertical Blank* ".
Note:

Vertical Blank = Pre Equalisation + VSync + Post Equalisation + VBlank

VSync = Pre Equalisation = VSync = Post Equalisation

For the parameters calculated at this point you get a line time of:

t_z = 4 * Display /f_z = 4 * 352 / 70 MHz = 20.11 us

VSync = t_v / t_z = 120 us / 20.11 us = 6 lines
VSync = 12

Vertical Blank = t_{vb} / t_z = 60 lines

VBlank = Vertical Blank - 3 * VSync
VBlank = 60 - (3 * 6) Rows = 42 lines

VBlank = 84

The other parameters result from the values established so far.

ShortDisplay = Display - ½ Linetime = 256 - 176 = 80
BroadPulse = Display - ½ Linetime + 2 HalfSync + BackPorch
BroadPulse = 80 + 30 + 53 = 163

The calculation of the following values is explained in chapter 3.2.1:

TransferDelay	= 14	(25 MHz Transputer, 8 bit/pixel)
MemInit	= 512 - 14 = 498	(Linear Addressing - panning not possible)
MemInit	= 256 - 14 = 242	(Row-Oriented Addressing - panning possible)
LineStart	= 0	(image starts at address #8.0000)

The screen refresh rate can be calculated using the following equation:

fv = [½ * (VBlank + 3 * HalfSync + VDisplay) * rowlength] $^{-1}$
= [½ * 1656 * 20.11 us] $^{-1}$ = 60 Hz

Checking to see if all conditions were met:

ShortDisplay > TransferDelay	80 > 14
BackPorch > TransferDelay	53 > 14
2 * HalfSync + BackPorch + Display > ½ * Linetime	339 > 176

The table on the left summarizes our calculated parameters, the table on the right compares the resulting values with the values theoretically required by the monitor.

Calculated parameters		
Linetime	=	352
Display	=	256
HalfSync	=	15
BackPorch	=	53
VDisplay	=	1536
VSync	=	12
VBlank	=	84

Segments		resulting Time	required Time
Rowslength	:	20.11 us	20.11 us
HSYNC	:	1.71 us	1.77 us
HFrontPorch	:	0.74 us	0.68 us
HBackPorch	:	3.03 us	3.04 us
VSync	:	0.12 ms	0.12 ms
VFrontPorch	:	0.12 ms	0.48 ms
VBackPorch	:	0.97 ms	0.60 ms
Vert.Blank	:	1.21 ms	1.20 ms

As you can see in the right hand table, the calculated values are more or less equal to those required by the monitor. The only major differences are in the time spans for VFrontPorch and VBackPorch. The critical values for Vert.Blank and VSync, however, are exactly what they should be.

In everyday use, most monitors can usually handle moderate deviations in timing quite well. The timing listed in the appendix for the EIZO 9070 is in some points a lot different from what the manufacturer suggests, but is easily tolerated and shortens the time necessary for adapting to the horizontal scan frequency.

Error analysis (finding the glitch):

The monitor screen stays black:

- The monitor's not synchronizing - horizontal scan frequency
- The G300's Mask Register isn't programmed and is blanking each pixel
- The VTG isn't running (Control Register bit 0 = 0)
- HSync is too short
- TransferDelay > BackPorch
- Linetime too short or BackPorch too long, this results in a negative horizontal FrontPorch.
- An error in the calculation of ShortDisplay or Broadpulse
- VBlank or VSync too short

The monitor's " whistling ":

- The monitor's not synchronizing - horizontal scan frequency

The picture's left edge is hazy or unrecognizable:

- The Horizontal BackPorch is too short

Pale or wrong colours:

- The Horizontal BackPorch is too short; the monitor isn't able to manage the colour balance.

The upper edge of the picture is in a transient state:

- Either VSync, VBlank or HSync are too short

Colour shifts within the picture or picture distortion:

- The monitor has to be rebalanced or replaced.

7. Software Examples

7.1 Booting The GDS-II

The way the GDS-II's transputer is configured, it can be booted via any of its four Links. After resetting (" *PowerOn-Reset* ", " *Master Reset* " or " *Link Reset* "), all Links have the same function and are waiting for the boot-code. The first data that the transputer receives by any of the Links will be interpreted as a programcode and executed.

The GDS-II can be used as a normal transputer node under MultiTool. The user will then have a total of 4 MByte of memory with an access time of 100 ns. Since the working memory and the video memory are immediately next to each other, the transputer will treat them as one.
The G300 will not perform any " *Transfer Cycles* " until the Video Timing-Generator has been started by setting the " *Control Register's* " bit 0. As a normal transputer node, the GDS-II with perform exactly like any other 4 MByte transputer node (e.g. TPM-4).

7.2 The GDS As A Host Under MultiTool

If you boot a GDS-II as a host with MultiTool, MultiTool will take up the entire memory, including the video memory.
MultiTool 5.0 offers an parameter that will limit the size of memory it will take over.

```
MTOOL -s #200000
```

If you start MultiTool in this manor, it will only use the bottom 2 MByte of memory e.g. the working memory. (#200000 HEX = 2,097,152 bit = 2 MByte)

Your " EXE " may not work, if you don't to start MultiTool this way.

7.3 The GDS As A Slave Under Multitool

The GDS-II is booted just like any other standard transputer node via one of its four links. The G300 can be initialized in advance by a separate program or else during the main program by a appropriate process. The complete parameter sets can be used directly[1] or you can install a library access[1] into the program.

7.4 X - Windows Under Helios On A GDS-II

There are only minor differences between using X-Windows on a GDS-II and using it on a GDS:

- Copy the GDS-II specific *device driver* " **gds2.d** " into the directory "**/helios/lib** ".

e.g.:

```
cp  /a/lib/gds2.d  /helios/lib
```

- The attribute " **SELECT_MODE** " has been added to the configuration directory "xrc", making it easier to choose a certain mode of operation. Chapter 3.4.5 explains the various parameters.

- To have full access to the GDS-II's working memory, specify it correctly in the appropriate " *Resource Map* ". A node " GDS-II " could, for instance, be defined like this (excerpt from the " *Resource Map* ")

```
terminal GDS2 {~00,,,; SYSTEM ;
                       memory # 200000 ;
                       ptype T800 ;
              }
```

In this example, the video memory is used solely as a " *Video Buffer* ". Inserting the line (memory # 300000) will cause X-Windows to use the video memory's bottom Megabyte as a working memory too, thus managing a total of 3 MBytes of free working memory.

Using a GDS-II as a Host node under X-Windows is theoretically possible, but you don't want to try it.

[1] See chapter 9.2

7.5 Link Software Addresses

After declaring the channels, use the following PLACE statements to coordinate the channels to the physical links.

```
PLACE Link0.Output AT #0 :
PLACE Link1.Output AT #1 :
PLACE Link2.Output AT #2 :
PLACE Link3.Output AT #3 :
PLACE Link0.Input  AT #4 :
PLACE Link1.Input  AT #5 :
PLACE Link2.Input  AT #6 :
PLACE Link3.Input  AT #7 :
```

7.6 Initialising The G300

After resetting[2)] the system, you first have to program the register " *Boot Location* ". This determines the " *Pixel Clock* ".
Then program the " *Control Register* " choose the G300's operation mode. At this point, bit 0 of the " *Control Registers* " has to be zero to keep the VTG from running.
The next step consists of programming the " *Datapath Register* " and the " *Mask Register* ".
Since it will take a while for the monitor to get synchronized after starting the VTG, its a good idea to leave the screen dark at the beginning. This can be done by setting the " *Mask Register* " to zero before starting the VTG, and then loading its correct value a little later.

2) Master Reset, Power-on Reset or Software Reset via Reset-Pal

7.7 The Transputer's Graphic Operations

The transputer offers a few interesting additional instructions for graphic applications. When using OCCAM 2 their names are **Move2D**, **Draw2D** and **Clip2D** (see the following chapters for details)[3].
When using Helios 'C' you have similar functions under " *system call* " **bytblt!**[4]

7.7.1 Move2D

This command, also known as 2-dimensional Blockmove, can be used any time you want to copy part of a 2-dimensional array into another array.

Move2D's function can illustrated by the following OCCAM2 procedure.

```
PROC Move2D (VAL[][]BYTE Source, VAL INT sx, sy,
             [][]BYTE Dest, VAL INT dx, dy, width, length)
  SEQ y = 0 FOR length
    [Dest[y+dy] FROM dx FOR width] :=
      [Source[y+sy] FROM sx FOR width]
:
```

Source:	Block you want (partially) copied.
sx, sy:	Coordinates of the first bytes you want copied out of Source .[5]
Dest:	Destination array
dx, dy:	Coordinates of the destination.[6]
width:	Width of the array you want to copy.
length:	Length (number of rows) of the array you want to copy.

[3] See " INMOS Technical note 26: < *Notes on Graphics Support and Performance Improvements on the IMS T800* > " for detailed description !

[4] See " *HELIOS operating system part II*; chapter 8 < *calling helios* > "

[5] Coordinates of the lower left pixel of the array you want to copy

[6] Source pixel { sx, sy } is copied to the coordinates { dx, dy } of Dest

7.7.2 Draw2D

This command can also be used any time you want to copy part of a 2-dimensional array into another array. The difference to **Move2D** is that this time, only bytes with values different from zero are copied. This, for instance, makes it very easy to place letters on a given background. Define the letter in a 2-dimensional array so that everything except the letter is given the value zero. **Draw2D** will then ignore everything in the array except those parts belonging to the letter.

Draw2D's function can illustrated by the following OCCAM2 procedure.

```
PROC Draw2D (VAL[][]BYTE Source, VAL INT sx, sy,
             [][]BYTE Dest, VAL INT dx, dy, width, length)
  BYTE temp:
  SEQ line := 0 FOR length
    SEQ point = 0 FOR width
      SEQ
        temp := Source[line+sy][point+sx]
        IF
          temp = (BYTE 0)
            SKIP
          TRUE
            Dest[line+dy][point+dx] := temp
:
```

Source:	Block you want (partially) copied.
sx, sy:	Coordinates of the first bytes you want copied out of Source .[7]
Dest:	Destination array
dx, dy:	Coordinates of the destination.[8]
width:	Width of the array you want to copy,
length:	Length (number of rows) of the array you want to copy.

[7] Coordinates of the lower left pixel of the array you want to copy

[8] Source pixel { sx, sy } is copied to the coordinates { dx, dy } of Dest

7.7.3 Clip2D

This is the **inverse function** to **Draw2D**. **Clip2D** will only copy bytes with the value zero.

Clip2D's function can illustrated by the following OCCAM2 procedure.

```
PROC Draw2D (VAL[][]BYTE Source, VAL INT sx, sy,
             [][]BYTE Dest, VAL INT dx, dy, width, length)
  BYTE temp:
  SEQ line := 0 FOR length
    SEQ point = 0 FOR width
      SEQ
        temp := Source[line+sy][point+sx]
        IF
          temp = (BYTE 0)
            Dest[line+dy][point+dx] := temp
          TRUE
            SKIP
:
```

Source:	Block you want (partially) copied.
sx, sy:	Coordinates of the first bytes you want copied out of Source .[9]
Dest:	Destination array
dx, dy:	Coordinates of the destination.[10]
width:	Width of the array you want to copy,
length:	Length (number of rows) of the array you want to copy.

[9] Coordinates of the lower left pixel of the array you want to copy

[10] Source pixel { sx, sy } is copied to the coordinates { dx, dy } of Dest

7.8 Synchronizing With The Vertical Blank

The G300 sends out a signal that is active throughout the entire duration of the " *Vertical Blank* ". This signal is connected to the transputer's " *event input* ".

By accessing the " *event channel* " you can time certain procedures, so that they will only be active during the " *Vertical Blank* ". This would make it possible to avoid disturbances they might otherwise cause.

A few examples:

Programming the Colour Lookup Table:

When programming the CLUT, it may happened that the CLUT's bits change their values just when CLUT's output signal (single pixels) is being latched into the G300. This causes short disturbances seen on the monitor screen.
You can avoid this be programming the CLUT only during " *Vertical Blank* ".

Switching between two images (" *Double Buffering* ")

You have to modify the " *TopScreen Register* " if you want to switch between images (see chapter 3.2.3). To avoid disturbances, try programming the " *TopScreen Register* " during the " *Vertical Blank* ".

Moving images (Hardware Panning)

Moving images is also only possible by " *TopScreen Register* " modification (see chapter 3.2.2).

How the Event channel works:

The transputer's event input is controlled by pulse edges. In other words, a rising edge at this input (beginning of the VBlank) activates the event channel[11)]. It stays active till inactivated by an access. As a result, the transputer will only be able to react to a further rising pulse edge after being accessed.

[11)] See INMOS T800 Data Sheet: " *When an external event takes EventReq (Event Input of T800) high the external event channel is made ready to communicate with a process.* "

7.8.1 Programming the CLUT (one interrupt source)

Programming is fairly simple if interrupts can only be generated by the G300. The following OCCAM2 excerpt should illustrate this :

```
PROC event ( )
  TIMER clock         :
  VAL timeout IS 780  :       -- 50 ms
  INT time            :
  CHAN OF ANY event   :       -- Event channel definition
  PLACE event AT #8   :

  SEQ
    clock ? time
    ALT
      event ? x               -- Wait until event channel is activated
        SKIP
      clock ? time PLUS timeout    -- Wait 50 ms
        SKIP
:

PROC set.CLUT ( [256] INT dummy.array )
  [256] INT CLUT          :
  PLACE CLUT  AT #20100000:

  SEQ
    event ( )                  -- Dummy access activates the Event channel
    event ( )                  -- Waiting for the next VBlank to begin
    CLUT := dummy.array        -- Copy CLUT values into CLUT
:

PROC grey.colour.scale ( )
  [256] INT dummy.array   :

  SEQ
    SEQ i = 0 FOR 256
      dummy.array[i] := i + (( i << 8 ) + ( i << 16 ))
    set.CLUT ( dummy.array )
:
```

A " *VBlank* " usually lasts at least 1 ms. Always consider this time span when planning lengthier operations. You also have to consider the " reaction time " for an event. This is influenced by the number and priorities of any background pocesses.

In the example above, the event channel is accessed twice in a row. Synchronization is not achieved the first time.

Since the event channel is controlled by pulse edges, it will be activated by the first " *VBlank* " to hit it. All further " *VBlanks* " will be ignored until an access sets it back. The above process will not wait for a " *VBlank* " after the first access - it will immediately deactivate the event channel and proceed to the next command. The second time around, however, it will wait until the event channel (deactivated at that point) is reactivated by a new " *VBlank* ".

7.8.2 Programming the CLUT (several interrupt source)

If the GDS-II is connected to further interrupt causes (Daughterboards) you will be forced to check what caused the event (VBlank or something else).

Example 1: (OCCAM2)

```
PROC waiting.for.event ( )
  INT x, ident :
  BOOL running :
  CHAN OF ANY event    :          -- Event channel definition
  PLACE ident AT #20000000:
  PLACE event AT #8    :

  TIMER clock          :
  VAL timeout IS 1560 :           -- 100 ms
  INT time             :

  SEQ
    running := TRUE
    clock ? time
    ALT
      WHILE running
        SEQ
          event ? x
          IF
            (ident/\#100) = #100    -- Check Ident Register's bit 8
              running := FALSE      -- Event caused by VBlank
            TRUE
              SKIP                  -- Event caused by a Daughterboard
      clock ? time PLUS timeout     -- No event for 100 ms
        SKIP
:

PROC set.CLUT ( [256] INT dummy.array )
  [256] INT CLUT          :
  PLACE CLUT  AT #20100000:

  SEQ
    waiting.for.event ( )     -- Dummy access activates the Event channel
    waiting.for.event ( )     -- Waiting for the next VBlank to begin
    CLUT := dummy.array       -- Copy CLUT values into CLUT[12]
:
```

[12] Definition of " dummy.array " in chapter 7.8.1

Example 2: (Helios 'C')

Testroutine for VBlank - Pulses recognition (several interrupt causes)

```
/***************************************************************/
/*                                                             */
/* Program demonstrates the use of the transputer event line */
/* on a GDS2 for synchronization with the VBlank              */
/*                                                             */
/* Main waits on VBlank (Event line) and increments a counter*/
/* Every 50 VBlanks a message is displayed.                   */
/*                                                             */
/***************************************************************/
#include <syslib.h>
#include <stdio.h>
#include <event.h>
#include <sem.h>

/*          function prototyping                                */
void SetEvent (Event *handler);     /* should be in event.h but isn't  */
void RemEvent (Event *handler);     /* should be in event.h but isn't  */

void evhandler(word *data, Event *event);

/* initialize event handler struct                              */
Event handler = {                  /* struct for HELIOS event handling */
                                   /* see technical note #13 ' Use of  */
                                   /* the Transputer Event Line from   */
                                   /* Helios ' Jan. 89                 */
    NULL, NULL,                    /* list pointer (internal use)      */
    0,                             /* priority (in list)               */
    evhandler,                     /* event handler function pointer   */
    NULL,                          /* data vector                      */
    NULL                           /* reserved!                        */
};

Semaphore ev;

int main ()
{
    int i = 1;

    InitSemaphore (&ev, 0);
    SetEvent (&handler);           /* install event handler in list    */
    printf ("Waiting for VSync of GDS2...\n");
    while (i < 1000) {             /* get 1000 events                  */
            Wait (&ev);            /* wait on event signal             */
            if ((i % 50)==0) {
            printf ("Event %d recognized.\n", i);    /* print          */
     }
     i++;
    }
    RemEvent (&handler);           /* remove event handler from list   */
    printf ("Program terminating.\n");
    return (0);
 }
```

```
void evhandler (word *data, Event *event)
{
    if (gds2event())                /* Event from GDS2 ??                        */
        Signal (&ev);               /* yes: signal to main... and terminate      */
    return;
}


int gds2event()
{
    int *pident = 0x0;              /* pointer to ident register                 */
    return ((*pident & 0x100)); /* return bit 8 of ident register (VBlank)
*/

}
```

7.9 Reading The Datapath Register

The following example in OCCAM 2 should give you an idea of how to disable the VTG.
The problem:

After G300's initialization you want to know the screen format.

```
PROC event ( )
  TIMER clock          :
  CHAN OF ANY event    :
  PLACE event AT #8    :
  VAL timeout IS 780   :          -- 50 ms
  INT time             :
  SEQ
    clock ? time
    ALT
      event ? x                          -- Event received
        SKIP
      clock ? time PLUS timeout          -- No event since 50 ms
        SKIP
:


PROC disable.VTG ( )
  INT   Control              :
  PLACE Control AT #20100160 :
  SEQ
    event ( )                            -- Reset event channel
    event ( )                            -- First received VBlank
    Control := Control /\ #FFFFFE        -- Disable VTG
:


PROC enable.VTG ( )
  INT   Control              :
  PLACE Control AT #20100160 :
  SEQ
    Control := Control \/ #1             -- Enable VTG
:


PROC screen.format ( INT XSIZE, YSIZE )
  INT   Display  :
  INT   VDisplay :
  PLACE Display  AT #20100123 :
  PLACE VDisplay AT #20100128 :
  SEQ
    disable.VTG ( )
    XSIZE := Display  * 4
    YSIZE := VDisplay / 2
    enable.VTG ( )
:
```

8. GDS-II vs. GDS - A Comparison

	GDS-II	GDS
Working memory:	2 MByte	1 MByte
Video memory:	2 MByte	1 MByte
Video Timing Controller:	G300 (INMOS)	TS 68483 (Thomson)
Memory cycle time:	200 ns	250 ns
Backplane link layout:	a: Link 0 b: Link 1 c: Link 2 d: Link 3 e: Link 0[1] f: Link 1[1] g: video out h: Link 3[1]	a: Link 0 b: not used c:. Link 1 d: not used e: Link 2 f: not used g: video out h: Link 3
Colour lookup table:[2]	3 * 8 bit	3 * 6 bit
Number of possible colours:	16,777,216	262,144
Colours used simultaneously (13 bpp):	8192	--
Colours used simultaneously (8 bpp):	256	256
Colours used simultaneously (4 bpp):	16	--
Colours used simultaneously (2 bpp):	4	--
Colours used simultaneously (1 bpp):	2	--
Number of basic colours:	256	256
" *Pixel Clock* "[2]:	In 5 MHz steps up to 110 MHz	32 or 64 MHz
Maximal resolution (1, 2, 4, 8 bpp):	1280 * 1024 pixel	1024 * 768 pixel
Maximal resolution (13, 24 bpp):	800 * 600 pixel	--
Video memory size (1 bpp):	16,384 * 1024 pixel	--
Video memory size (2 bpp):	8192 * 1024 pixel	--
Video memory size (4 bpp):	4096 * 1024 pixel	--
Video memory size (8 bpp):	2048 * 1024 pixel	1024 * 1024 pixel
Video memory size (13 bpp):	1024 * 1024 pixel	--
Video memory size (24 bpp):	524,288 pixel	--
Location of the left topmost monitor pixel in the video memory:	bottom video memory address	top video memory address
Generation of the Transferaddresses by the Video Timing Controller:	Addresses are incremented	Addresses are decremented

[1] Links are connected to Daughterboard-Slot A

[2] Reffers to the 8 bits per pixel mode. 13 and 24 bit/pixel are only possible on a GDS-II (!)

9. Appendix

9.1 Configuration examples

Three examples for possible configurations are given on the next pages.

A jumper overview is given for the following configurations:

9.1.1 - 8 bit/pixel

- variable " *Pixel Clock* " via PLL
- 20 MHz Transputer (4 processor cycles)
- Slot A is configured for a DBI-x module

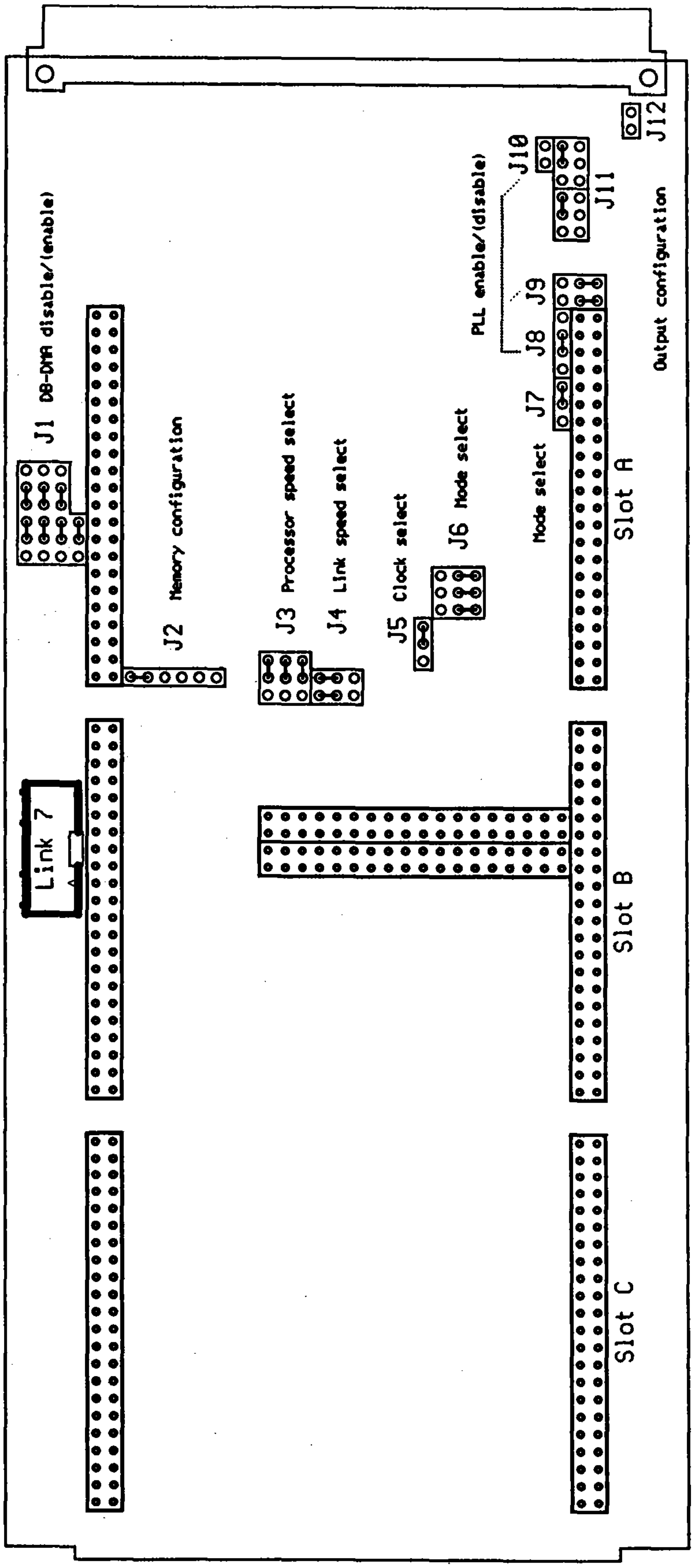
9.1.2 - 13 bit/pixel

- 32 MHz " *Pixel Clock* "
- 25 MHz Transputer (5 processor cycles)
- Slot A is configured for a DBI-x module

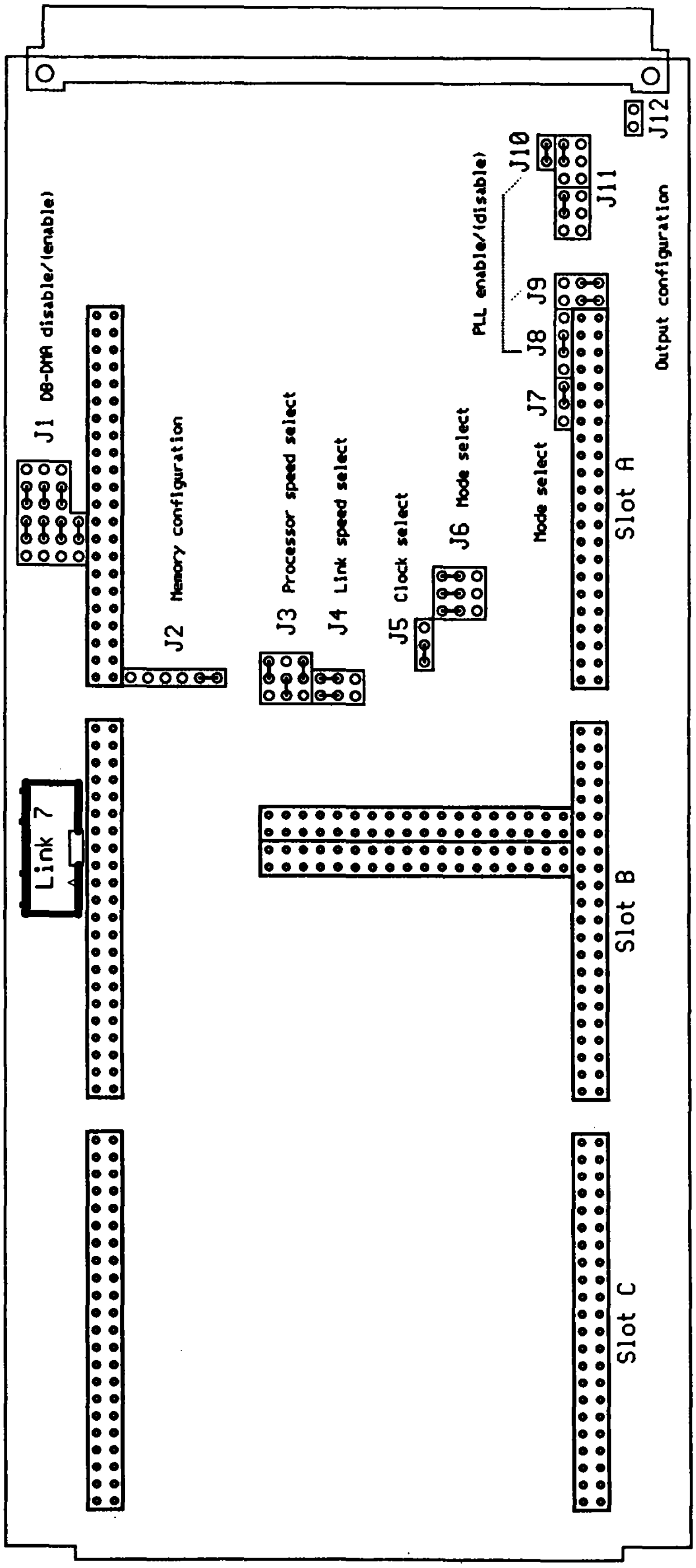
9.1.3 - 24 bit/pixel

- 32 MHz " *Pixel Clock* "
- 20 MHz Transputer (5 processor cycles)
- Slot A is configured for a DBI-x module

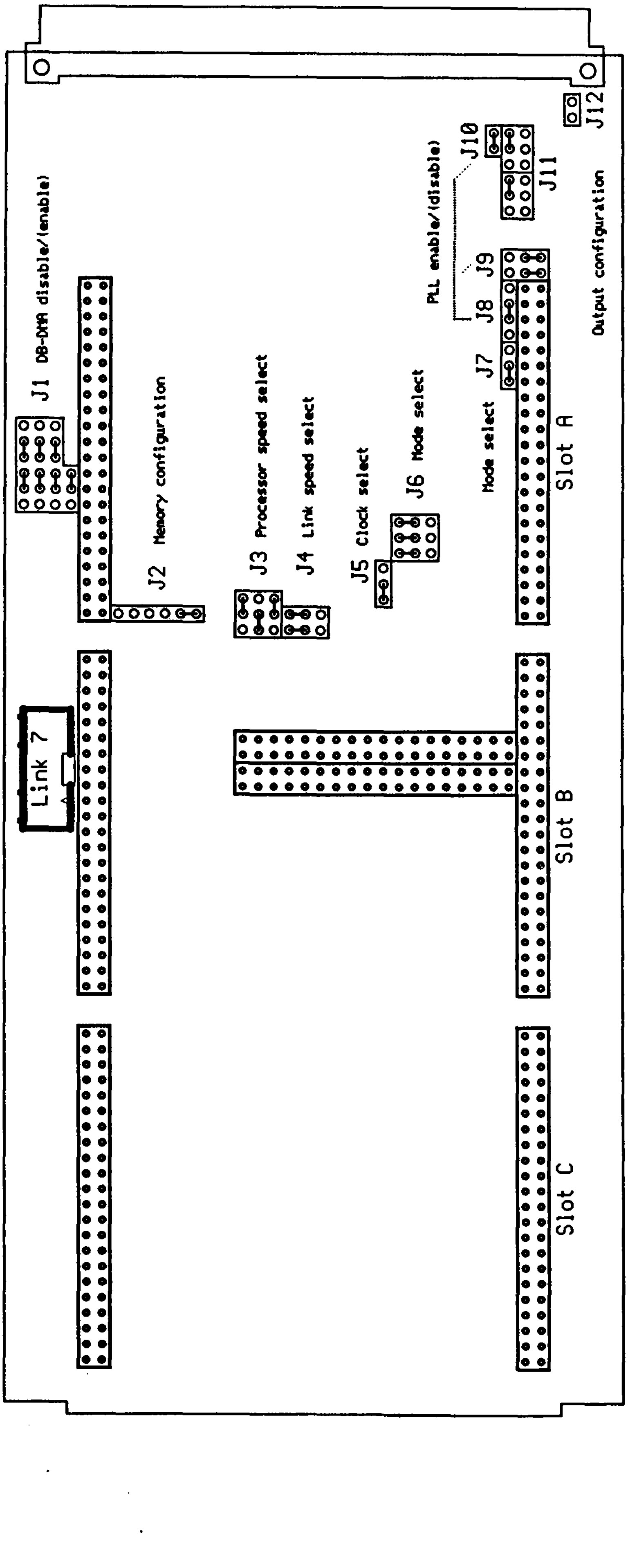
9.1.1 8 Bit/Pixel



9.1.2 13 Bit/Pixel



9.1.3 24 Bit/Pixel



9.2 Video Timings

9.2.1 EIZO FLEXSCAN 8060 - mode 1

```
-- EIZO 8060 - 30 MHz dot clock - 640*480 dots - 68.5 Hz

                    -- pll = 6
LINETIME  := 210    -- 35.7 Khz Horiz.Scanning freq.
DISPLAY   := 160    -- 640 dots
HALFSYNC  := 12     -- Horiz. Sync        = 3.2  us
BACKPORCH := 24     -- Horiz. Backporch   = 3.2  us
--           2         Horiz. Frontporch = 0.3  us

VDISPLAY  := 960    -- 480 lines
VSYNC     := 6      -- 3  lines     = 0.084ms
VBLANK    := 64     -- 32 lines
                    -- VFP = PreEqualisation = VSync   = 0.084 ms
                    -- VBP = VBlank + PostEqualisation = 0.98  ms
                    -- total lines a 28 us            = 521 lines
                    -- total time for each image       =  14.6 ms
TRDELAY   := 12     -- 1.6 us
```

```
-- EIZO 8060 - 35 MHz dot clock - 720*540 dots - 64.1 Hz

                     -- pll = 73
LINETIME   := 240   -- 36.5 Khz Horiz.Scanning freq.
DISPLAY    := 180   -- 720 dots
HALFSYNC   := 12    -- Horiz. Sync        = 2.7  us
BACKPORCH  := 34    -- Horiz. Backporch   = 3.9  us
--            2        Horiz. Frontporch = 0.2  us

VDISPLAY   := 1080 -- 540 lines
VSYNC      := 6     -- 3  lines      = 0.082ms
VBLANK     := 40    -- 20 lines
                     -- VFP = PreEqualisation = VSync   = 0.082 ms
                     -- VBP = VBlank + PostEqualisation = 0.631 ms
                     -- total lines a 27.4us            = 569 lines
                     -- total time for each image       =  15.6 ms
TRDELAY    := 13    -- 1.5 us
```

```
-- EIZO 8060 - 35 MHz dot clock - 768*576 dots - 59.3 Hz

                     -- pll = 7
LINETIME   := 244   -- 35.86 Khz Horiz.Scanning freq.
DISPLAY    := 192   -- 768 dots
HALFSYNC   := 12    -- Horiz. Sync        = 2.7  us
BACKPORCH  := 26    -- Horiz. Backporch   = 3.0  us
--            2        Horiz. Frontporch = 0.2  us

VDISPLAY   := 1152 -- 576 lines
VSYNC      := 6     -- 3  lines      = 0.083ms
VBLANK     := 40    -- 20 lines
                     -- VFP = PreEqualisation = VSync   = 0.083 ms
                     -- VBP = VBlank + PostEqualisation = 0.891 ms
                     -- total lines a 27.9us            = 605 lines
                     -- total time for each image       = 16.86 ms
TRDELAY    := 13    -- 1.5 us
```

```
-- EIZO 8060 - 40 MHz dot clock - 800*600 dots - 62.0 Hz

                     -- pll = 8
LINETIME  := 258     -- 38.8 kHz Horiz.Scanning freq.
DISPLAY   := 200     -- 800 dots
HALFSYNC  := 12      -- Horiz. Sync       = 2.4  us
BACKPORCH := 33      -- Horiz. Backporch  = 3.3  us
--           1          Horiz. Frontporch = 0.1  us

VDISPLAY  := 1200 -- 600 lines
VSYNC     := 6       -- 3  lines     = 0.077ms
VBLANK    := 32      -- 16 lines
                     -- VFP = PreEqualisation = VSync   = 0.077 ms
                     -- VBP = VBlank + PostEqualisation = 0.722 ms
                     -- total lines a 28.8us           = 625 lines
                     -- total time for each image      =  16.13 ms
TRDELAY   := 14      -- 1.4 us
```

```
-- EIZO 8060 - 40 MHz dot clock - 800*600 dots - 59.8 Hz

                     -- pll = 8
LINETIME  := 266     -- 37.6 kHz Horiz.Scanning freq.
DISPLAY   := 200     -- 800 dots
HALFSYNC  := 12      -- Horiz. Sync       = 2.4  us
BACKPORCH := 41      -- Horiz. Backporch  = 4.1  us
--           1          Horiz. Frontporch = 0.1  us

VDISPLAY  := 1200 -- 600 lines
VSYNC     := 6       -- 3  lines     = 0.079ms
VBLANK    := 36      -- 18 lines
                     -- VFP = PreEqualisation = VSync   = 0.079 ms
                     -- VBP = VBlank + PostEqualisation = 0.797 ms
                     -- total lines a 26.6us           = 627 lines
                     -- total time for each image      =  16.7 ms
TRDELAY   := 14      -- 1.4 us
```

9.2.2 EIZO FLEXSCAN 8060 - mode 2

```
-- EIZO 8060 - 32 MHz dot clock - 640*480 dots - 70.0 Hz
                 -- pll = 0
LINETIME   := 220  -- 36.36 Khz Horiz.Scanning freq.
DISPLAY    := 160  -- 640 dots
HALFSYNC   := 12   -- Horiz. Sync       = 3   us
BACKPORCH  := 34   -- Horiz. Backporch  = 4.3 us
--            2       Horiz. Frontporch = 0.3 us

VDISPLAY   := 960  -- 480 lines
VSYNC      := 6    -- 3  lines     = 0.082ms
VBLANK     := 64   -- 31 lines
                   -- VFP = PreEqualisation = VSync   = 0.082 ms
                   -- VBP = VBlank + PostEqualisation = 0.935 ms
                   -- total lines a 27.5us           = 520 lines
                   -- total time for each image       =  14.3 ms
TRDELAY    := 11   -- 1.4 us
```

```
-- EIZO 8060 - 32 MHz dot clock - 720*540 dots - 61.4 Hz
                 -- pll = 0
LINETIME   := 232  -- 34.5 Khz Horiz.Scanning freq.
DISPLAY    := 180  -- 720 dots
HALFSYNC   := 12   -- Horiz. Sync       = 3    us
BACKPORCH  := 26   -- Horiz. Backporch  = 3.3  us
--            2       Horiz. Frontporch = 0.3  us

VDISPLAY   := 1080 -- 540 lines
VSYNC      := 6    -- 3  lines     = 0.087ms
VBLANK     := 26   -- 13 lines
                   -- VFP = PreEqualisation = VSync   = 0.087 ms
                   -- VBP = VBlank + PostEqualisation = 0.464 ms
                   -- total lines a 29us             = 562 lines
                   -- total time for each image       =  16.3 ms
TRDELAY    := 11   -- 1.4 us
```

```
- EIZO 8060 - 32 MHz dot clock - 768*576 dots - 56.2 Hz
                    -- pll = 0
LINETIME   := 240   -- 33.3 kHz Horiz.Scanning freq.
DISPLAY    := 192   -- 768 dots
HALFSYNC   := 10    -- Horiz. Sync       = 2.5  us
BACKPORCH  := 27    -- Horiz. Backporch  = 3.4  us
--            1        Horiz. Frontporch = 0.1  us

VDISPLAY   := 1152 -- 576 lines
VSYNC      := 4    -- 2  lines      = 0.060ms
VBLANK     := 26   -- 13 lines
                   -- VFP = PreEqualisation = VSync    = 0.060 ms
                   -- VBP = VBlank + PostEqualisation = 0.450 ms
                   -- total lines a 30us              = 595 lines
                   -- total time for each image        =  17.8 ms
TRDELAY    := 12   -- 1.5 us
```

```
-- EIZO 8060 - 32 MHz dot clock - 800*600 dots - 51.8 Hz

                    -- pll = 0
LINETIME   := 248   -- 32.3 kHz Horiz.Scanning freq.
DISPLAY    := 200   -- 800 dots
HALFSYNC   := 10    -- Horiz. Sync       = 2.5  us
BACKPORCH  := 26    -- Horiz. Backporch  = 3.3  us
--            2        Horiz. Frontporch = 0.3  us

VDISPLAY   := 1200 -- 600 lines
VSYNC      := 6    -- 3  lines      = 0.093ms
VBLANK     := 26   -- 13 lines
                   -- VFP = PreEqualisation = VSync    = 0.093 ms
                   -- VBP = VBlank + PostEqualisation = 0.496 ms
                   -- total lines a 31us              = 622 lines
                   -- total time for each image        =  19.3 ms
TRDELAY    := 11   -- 1.4 us
```

9.2.3 EIZO FLEXSCAN 9070 - mode 1

```
-- EIZO 9070 - 35 MHz dot clock - 640*480 dots - 80.0 Hz

                  -- pll = 7
LINETIME  := 212  -- 41.2 Khz Horiz.Scanning freq.
DISPLAY   := 160  -- 640 dots
HALFSYNC  := 12   -- Horiz. Sync       = 2.7  us
BACKPORCH := 18   -- Horiz. Backporch  = 2.1  us
--           10      Horiz. Frontporch = 1.1  us

VDISPLAY  := 960  -- 480 lines
VSYNC     := 8    -- 4  lines     = 0.096ms
VBLANK    := 50   -- 25 lines
                  -- VFP = PreEqualisation = VSync   = 0.096 ms
                  -- VBP = VBlank + PostEqualisation = 0.702 ms
                  -- total lines a 24.2us          = 517 lines
                  -- total time for each image       =  12.5 ms
TRDELAY   := 10


-- EIZO 9070 - 40 MHz dot clock - 720*540 dots - 76.3 Hz

                  -- pll = 8
LINETIME  := 228  -- 43.9 Khz Horiz.Scanning freq.
DISPLAY   := 180  -- 720 dots
HALFSYNC  := 6    -- Horiz. Sync       = 1.2  us
BACKPORCH := 28   -- Horiz. Backporch  = 2.8  us
--           8       Horiz. Frontporch = 0.8  us

VDISPLAY  := 1080 -- 540 lines
VSYNC     := 6    -- 3  lines     = 0.068ms
VBLANK    := 50   -- 25 lines
                  -- VFP = PreEqualisation = VSync   = 0.068 ms
                  -- VBP = VBlank + PostEqualisation = 0.638 ms
                  -- total lines a 22.8us          = 574 lines
                  -- total time for each image       =  13.1 ms
TRDELAY   := 12   -- 1.2 us
```

```
-- EIZO 9070 - 45 MHz dot clock - 768*576 dots - 74.1 Hz

                  -- pll = 9
LINETIME   := 250 -- 45.0 kHz Horiz.Scanning freq.
DISPLAY    := 192 -- 768 dots
HALFSYNC   := 6   -- Horiz. Sync       = 1.1  us
BACKPORCH  := 36  -- Horiz. Backporch  = 3.2  us
--            10     Horiz. Frontporch = 0.9  us

VDISPLAY   := 1152 -- 576 lines
VSYNC      := 6   -- 3  lines     = 0.066ms
VBLANK     := 44  -- 22 lines
                  -- VFP = PreEqualisation = VSync   = 0.066 ms
                  -- VBP = VBlank + PostEqualisation = 0.556 ms
                  -- total lines a 22.2us           = 607 lines
                  -- total time for each image       =  13.5 ms
TRDELAY    := 13  -- 1.2 us
```

```
-- EIZO 9070 - 45 MHz dot clock - 800*600 dots - 70.4 Hz

                  -- pll = 9
LINETIME   := 254 -- 44.3 kHz Horiz.Scanning freq.
DISPLAY    := 200 -- 800 dots
HALFSYNC   := 6   -- Horiz. Sync       = 1.1  us
BACKPORCH  := 34  -- Horiz. Backporch  = 3.0  us
--            8      Horiz. Frontporch = 0.7  us

VDISPLAY   := 1200 -- 600 lines
VSYNC      := 6   -- 3  lines     = 0.067ms
VBLANK     := 44  -- 22 lines
                  -- VFP = PreEqualisation = VSync   = 0.067 ms
                  -- VBP = VBlank + PostEqualisation = 0.565 ms
                  -- total lines a 22.6us           = 631 lines
                  -- total time for each image       =  14.2 ms
TRDELAY    := 14  -- 1.2 us
```

```
-- EIZO 9070 - 65 MHz dot clock - 1024*768 dots - 61.7 Hz

                     -- pll = 13
LINETIME   := 324   -- 50.2 Khz Horiz.Scanning freq.
DISPLAY    := 256   -- 1024 dots
HALFSYNC   := 11    -- Horiz. Sync       = 1.4 us
BACKPORCH  := 40    -- Horiz. Backporch  = 2.5 us
--             6       Horiz. Frontporch = 0.4 us

VDISPLAY   := 1536 -- 768 lines
VSYNC      := 8     -- 4   lines    = 0.079ms
VBLANK     := 60    -- 30  lines
                     -- VFP = PreEqualisation = VSync    = 0.079 ms
                     -- VBP = VBlank + PostEqualisation = 0.678 ms
                     -- total lines a 19.9us            = 810 lines
                     -- total time for each image        =  16.2 ms
TRDELAY    := 18    -- 1 us


-- EIZO 9070 - 70 MHz dot clock - 1080*810 dots - 59.9 Hz

                     -- pll = 14
LINETIME   := 344   -- 50.9 Khz Horiz.Scanning freq.
DISPLAY    := 270   -- 1080 dots
HALFSYNC   := 12    -- Horiz. Sync       = 1.4 us
BACKPORCH  := 44    -- Horiz. Backporch  = 2.5 us
--             6       Horiz. Frontporch = 0.3 us

VDISPLAY   := 1620 -- 810 lines
VSYNC      := 6     -- 3   lines    = 0.058ms
VBLANK     := 60    -- 30  lines
                     -- VFP = PreEqualisation = VSync    = 0.058 ms
                     -- VBP = VBlank + PostEqualisation = 0.649 ms
                     -- total lines a 19.7us            = 849 lines
                     -- total time for each image        =  16.7 ms
TRDELAY    := 18    -- 1 us
```

9.2.4 EIZO FLEXSCAN 9070, 9500 - mode 2

```
-- EIZO 9070, 9500 - 32 MHz dot clock - 640*480 dots - 78.1 Hz

                      -- pll = 0
LINETIME  := 200      -- 40.0 Khz Horiz.Scanning freq.
DISPLAY   := 160      -- 640 dots
HALFSYNC  := 10       -- Horiz. Sync       = 2.5  us
BACKPORCH := 12       -- Horiz. Backporch  = 1.5  us
--           8           Horiz. Frontporch = 1.0  us

VDISPLAY  := 960      -- 480 lines
VSYNC     := 6        -- 3  lines     = 0.075ms
VBLANK    := 50       -- 25 lines
                      -- VFP = PreEqualisation = VSync    = 0.075 ms
                      -- VBP = VBlank + PostEqualisation = 0.700 ms
                      -- total lines a 25us              = 514 lines
                      -- total time for each image        =  12.8 ms
TRDELAY   := 10


-- EIZO 9070, 9500 - 32 MHz dot clock - 720*540 dots - 63.7 Hz

                      -- pll = 0
LINETIME  := 222      -- 36.0 Khz Horiz.Scanning freq.
DISPLAY   := 180      -- 720 dots
HALFSYNC  := 7        -- Horiz. Sync       = 1.8  us
BACKPORCH := 26       -- Horiz. Backporch  = 3.3  us
--           2           Horiz. Frontporch = 0.3  us

VDISPLAY  := 1080     -- 540 lines
VSYNC     := 6        -- 3  lines     = 0.083ms
VBLANK    := 34       -- 17 lines
                      -- VFP = PreEqualisation = VSync    = 0.083 ms
                      -- VBP = VBlank + PostEqualisation = 0.555 ms
                      -- total lines a 27.8us            = 566 lines
                      -- total time for each image        =  15.7 ms
TRDELAY   := 10
```

```
-- EIZO 9070, 9500 - 32 MHz dot clock - 768*576 dots - 57.3 Hz

                       -- pll = 0
LINETIME  := 232   -- 34.5 kHz Horiz.Scanning freq.
DISPLAY   := 192   -- 768 dots
HALFSYNC  := 6     -- Horiz. Sync       = 1.5  us
BACKPORCH := 26    -- Horiz. Backporch  = 3.3  us
--           2        Horiz. Frontporch = 0.3  us

VDISPLAY  := 1152  -- 576 lines
VSYNC     := 6     -- 3  lines     = 0.087ms
VBLANK    := 34    -- 17 lines
                   -- VFP = PreEqualisation = VSync   = 0.087 ms
                   -- VBP = VBlank + PostEqualisation = 0.580 ms
                   -- total lines a 29us             = 602 lines
                   -- total time for each image       = 17.45 ms
TRDELAY   := 10    -- 1.3 us



-- EIZO 9070, 9500 - 32 MHz dot clock - 800*600 dots - 53.2 Hz

                       -- pll = 0
LINETIME  := 240   -- 33.3 kHz Horiz.Scanning freq.
DISPLAY   := 200   -- 800 dots
HALFSYNC  := 6     -- Horiz. Sync       = 1.5  us
BACKPORCH := 26    -- Horiz. Backporch  = 3.3  us
--           2        Horiz. Frontporch = 0.3  us

VDISPLAY  := 1200  -- 600 lines
VSYNC     := 6     -- 3  lines     = 0.090ms
VBLANK    := 34    -- 17 lines
                   -- VFP = PreEqualisation = VSync   = 0.090 ms
                   -- VBP = VBlank + PostEqualisation = 0.600 ms
                   -- total lines a 30us             = 626 lines
                   -- total time for each image       =  18.8 ms
TRDELAY   := 10    -- 1.3 us
```

```
-- EIZO 9070, 9500 - 32 MHz dot clock - 800*600 dots - 56.8 Hz

                  -- pll = 0
LINETIME   := 228 -- 35.1 kHz Horiz.Scanning freq.
DISPLAY    := 200 -- 800 dots
HALFSYNC   := 6   -- Horiz. Sync       = 1.5  us
BACKPORCH  := 14  -- Horiz. Backporch  = 1.8  us
--            2      Horiz. Frontporch = 0.3  us

VDISPLAY   := 1200 -- 600 lines
VSYNC      := 4   -- 2  lines     = 0.057ms
VBLANK     := 20  -- 10 lines
                  -- VFP = PreEqualisation = VSync   = 0.057 ms
                  -- VBP = VBlank + PostEqualisation = 0.342 ms
                  -- total lines a 28.5us           = 616 lines
                  -- total time for each image        =  17.6 ms

TRDELAY    := 10
```

9.2.5 EIZO FLEXSCAN 9500 - mode 1

(EIZO 9500 timings only tested with the new monitor version. The new version has two different inputs: RGB and 9-pin D)

```
-- EIZO 9500 - 35 MHz dot clock - 640*480 dots - 80.0 Hz

                     -- pll = 7
LINETIME   := 212    -- 41.2 Khz Horiz.Scanning freq.
DISPLAY    := 160    -- 640 dots
HALFSYNC   := 12     -- Horiz. Sync       = 2.7  us
BACKPORCH  := 18     -- Horiz. Backporch  = 2.1  us
--            10        Horiz. Frontporch = 1.1  us

VDISPLAY   := 960    -- 480 lines
VSYNC      := 8      -- 4  lines     = 0.096ms
VBLANK     := 50     -- 25 lines
                     -- VFP = PreEqualisation = VSync    = 0.096 ms
                     -- VBP = VBlank + PostEqualisation = 0.702 ms
                     -- total lines a 24.2us           = 517 lines
                     -- total time for each image       =  12.5 ms
TRDELAY    := 10
```

```
-- EIZO 9500 - 40 MHz dot clock - 720*540 dots - 76.3 Hz

                     -- pll = 8
LINETIME   := 228    -- 43.9 Khz Horiz.Scanning freq.
DISPLAY    := 180    -- 720 dots
HALFSYNC   := 6      -- Horiz. Sync       = 1.2  us
BACKPORCH  := 28     -- Horiz. Backporch  = 2.8  us
--            8         Horiz. Frontporch = 0.8  us

VDISPLAY   := 1080 -- 540 lines
VSYNC      := 6      -- 3  lines     = 0.068ms
VBLANK     := 50     -- 25 lines
                     -- VFP = PreEqualisation = VSync    = 0.068 ms
                     -- VBP = VBlank + PostEqualisation = 0.638 ms
                     -- total lines a 22.8us           = 574 lines
                     -- total time for each image       =  13.1 ms
TRDELAY    := 12     -- 1.2 us
```

```
-- EIZO 9500 - 45 MHz dot clock - 768*576 dots - 74.1 Hz

                   -- pll = 9
LINETIME  := 250   -- 45.0 kHz Horiz.Scanning freq.
DISPLAY   := 192   -- 768 dots
HALFSYNC  := 6     -- Horiz. Sync       = 1.1  us
BACKPORCH := 36    -- Horiz. Backporch  = 3.2  us
--           10       Horiz. Frontporch = 0.9  us

VDISPLAY  := 1152  -- 576 lines
VSYNC     := 6     -- 3  lines     = 0.066ms
VBLANK    := 44    -- 22 lines
                   -- VFP = PreEqualisation = VSync   = 0.066 ms
                   -- VBP = VBlank + PostEqualisation = 0.556 ms
                   -- total lines a 22.2us           = 607 lines
                   -- total time for each image       =  13.5 ms
TRDELAY   := 13    -- 1.2 us
```

```
-- EIZO 9500 - 45 MHz dot clock - 800*600 dots - 70.4 Hz

                   -- pll = 9
LINETIME  := 254   -- 44.3 kHz Horiz.Scanning freq.
DISPLAY   := 200   -- 800 dots
HALFSYNC  := 6     -- Horiz. Sync       = 1.1  us
BACKPORCH := 34    -- Horiz. Backporch  = 3.0  us
--           8        Horiz. Frontporch = 0.7  us

VDISPLAY  := 1200  -- 600 lines
VSYNC     := 6     -- 3  lines     = 0.067ms
VBLANK    := 44    -- 22 lines
                   -- VFP = PreEqualisation = VSync   = 0.067 ms
                   -- VBP = VBlank + PostEqualisation = 0.565 ms
                   -- total lines a 22.6us           = 631 lines
                   -- total time for each image       =  14.2 ms
TRDELAY   := 14    -- 1.2 us
```

```
-- EIZO 9500 - 65 MHz dot clock - 1024*768 dots - 61.7 Hz

                     -- pll = 13
LINETIME  := 324     -- 50.2 Khz Horiz.Scanning freq.
DISPLAY   := 256     -- 1024 dots
HALFSYNC  := 11      -- Horiz. Sync        = 1.4 us
BACKPORCH := 40      -- Horiz. Backporch   = 2.5 us
--           6          Horiz. Frontporch = 0.4 us

VDISPLAY  := 1536 -- 768 lines
VSYNC     := 8       -- 4   lines      = 0.079ms
VBLANK    := 60      -- 30  lines
                     -- VFP = PreEqualisation = VSync    = 0.079 ms
                     -- VBP = VBlank + PostEqualisation = 0.678 ms
                     -- total lines a 19.9us             = 810 lines
                     -- total time for each image        =  16.2 ms
TRDELAY   := 18      -- 1 us



-- EIZO 9500 - 70 MHz dot clock - 1280*1024 dots - 59.9 Hz

                     -- pll = 14
LINETIME  := 344     -- 50.9 Khz Horiz.Scanning freq.
DISPLAY   := 270     -- 1080 dots
HALFSYNC  := 12      -- Horiz. Sync        = 1.4 us
BACKPORCH := 44      -- Horiz. Backporch   = 2.5 us
--           6          Horiz. Frontporch = 0.3 us

VDISPLAY  := 1620 -- 810 lines
VSYNC     := 6       -- 3   lines      = 0.058ms
VBLANK    := 60      -- 30  lines
                     -- VFP = PreEqualisation = VSync    = 0.058 ms
                     -- VBP = VBlank + PostEqualisation = 0.649 ms
                     -- total lines a 19.7us             = 849 lines
                     -- total time for each image        =  16.7 ms
TRDELAY   := 18      -- 1 us
```

9.2.6 NEC MULTISYNC GSII - mode 1

```
-- NEC GSII - 30 MHz dot clock - 640*480 dots - 68.0 Hz

                         -- pll = 6
LINETIME  := 210         -- 35.7 Khz Horiz.Scanning freq.
DISPLAY   := 160         -- 640 dots
HALFSYNC  := 10          -- Horiz. Sync       = 2.7 us
BACKPORCH := 22          -- Horiz. Backporch  = 2.9 us
--           8              Horiz. Frontporch = 1.1 us

VDISPLAY  := 960         -- 480 lines
VSYNC     := 6           -- 3  lines     = 0.084ms
VBLANK    := 70          -- 35 lines
                         -- VFP = PreEqualisation = VSync   = 0.084 ms
                         -- VBP = VBlank + PostEqualisation = 1.06  ms
                         -- total lines a 28us              = 524 lines
                         -- total time for each image       =   14.7ms
TRDELAY   := 10          -- 1.3 us
```

```
-- NEC GSII - 35 MHz dot clock - 720*540 dots - 68.0 Hz

                         -- pll = 7
LINETIME  := 224         -- 39.6 Khz Horiz.Scanning freq.
DISPLAY   := 180         -- 720 dots
HALFSYNC  := 10          -- Horiz. Sync       = 2.3  us
BACKPORCH := 16          -- Horiz. Backporch  = 1.8  us
--           8              Horiz. Frontporch = 0.9  us

VDISPLAY  := 1080        -- 540 lines
VSYNC     := 4           -- 2  lines     = 0.051ms
VBLANK    := 56          -- 28 lines
                         -- VFP = PreEqualisation = VSync   = 0.051 ms
                         -- VBP = VBlank + PostEqualisation = 0.768 ms
                         -- total lines a 25.6us            = 574 lines
                         -- total time for each image       =  14.7 ms
TRDELAY   := 10          -- 1.1 us
```

```
-- NEC GSII - 40 MHz dot clock - 800*600 dots - 66.2 Hz

                  -- pll = 8
LINETIME  := 242  -- 41.32 kHz Horiz.Scanning freq.
DISPLAY   := 200  -- 800 dots
HALFSYNC  := 10   -- Horiz. Sync       = 2.0  us
BACKPORCH := 14   -- Horiz. Backporch  = 1.4  us
--           8       Horiz. Frontporch = 0.8  us

VDISPLAY  := 1200 -- 600 lines
VSYNC     := 4    -- 2  lines     = 0.048ms
VBLANK    := 36   -- 18 lines
                  -- VFP = PreEqualisation = VSync    = 0.048 ms
                  -- VBP = VBlank + PostEqualisation = 0.484 ms
                  -- total lines a 24.2us           = 624 lines
                  -- total time for each image       =  15.1 ms
TRDELAY   := 10   -- 1.1 us
```

9.2.7 NEC MULTISYNC GSII - mode 2

```
--   NEC GSII - 32 MHz dot clock - 640*480 dots - 72.5 Hz

                                -- pll = 0
LINETIME   := 210   -- 38.1 Khz Horiz.Scanning freq.
DISPLAY    := 160   -- 640 dots
HALFSYNC   := 10    -- Horiz. Sync       = 2.5 us
BACKPORCH  := 22    -- Horiz. Backporch  = 2.8 us
--            8        Horiz. Frontporch = 1.0 us

VDISPLAY   := 960   -- 480 lines
VSYNC      := 6     -- 3  lines     = 0.078ms
VBLANK     := 70    -- 35 lines
                    -- VFP = PreEqualisation = VSync   = 0.078 ms
                    -- VBP = VBlank + PostEqualisation = 0.998 ms
                    - total lines a 26.25us          = 524 lines
                    -- total time for each image       =  13.8 ms
TRDELAY    := 10    -- 1.3 us
```

```
--   NEC GSII - 32 MHz dot clock - 720*540 dots - 62.1 Hz

                                -- pll = 0
LINETIME   := 224   -- 35.71 Khz Horiz.Scanning freq.
DISPLAY    := 180   -- 720 dots
HALFSYNC   := 10    -- Horiz. Sync       = 2.5  us
BACKPORCH  := 16    -- Horiz. Backporch  = 2.0  us
--            8        Horiz. Frontporch = 1.0  us

VDISPLAY   := 1080 -- 540 lines
VSYNC      := 4     -- 2  lines     = 0.056ms
VBLANK     := 56    -- 28 lines
                    -- VFP = PreEqualisation = VSync   = 0.056 ms
                    -- VBP = VBlank + PostEqualisation = 0.840 ms
                    -- total lines a 28us             = 574 lines
                    -- total time for each image       =  16.1 ms
TRDELAY    := 10    -- 1.3 us
```

```
--   NEC GSII - 32 MHz dot clock - 800*600 dots - 52.9 Hz

                    -- pll = 0
LINETIME   := 242   -- 33.1 kHz Horiz.Scanning freq.
DISPLAY    := 200   -- 800 dots
HALFSYNC   := 10    -- Horiz. Sync         = 2.5  us
BACKPORCH  := 14    -- Horiz. Backporch    = 1.8  us
--            8        Horiz. Frontporch = 1.0  us

VDISPLAY   := 1200 -- 600 lines
VSYNC      := 4     -- 2  lines      = 0.060ms
VBLANK     := 36    -- 18 lines
                    -- VFP = PreEqualisation = VSync    = 0.060 ms
                    -- VBP = VBlank + PostEqualisation = 0.605 ms
                    -- total lines a 30.25us           = 624 lines
                    -- total time for each image        =  18.9 ms
TRDELAY    := 10    -- 1.3 us
```

9.2.8 NEC MULTISYNC XL

```
--   NEC XL - 65 MHz dot clock - 1024*768 dots - 60.2 Hz

                  -- pll = 13
LINETIME  := 334  -- 48.65 kHz Horiz.Scanning freq.
DISPLAY   := 256  -- 1024 dots
HALFSYNC  := 8    -- Horiz. Sync        = 1.0   us
BACKPORCH := 47   -- Horiz. Backporch   = 2.9   us
--           15      Horiz. Frontporch  = 0.9   us

VDISPLAY  := 1536 -- 768 lines
VSYNC     := 8    -- 4   lines    =  82.2 us
VBLANK    := 56   -- 28  lines
                  -- VFP = PreEqualisation = VSync   =  82.2 us
                  -- VBP = VBlank + PostEqualisation = 657.7 us
                  -- total lines a 20.55 us         = 808 lines
                  -- total time for each image       =  16.6 ms
TRDELAY   := 18
```

9.2.9 SONY GDM 1602

```
-- SONY GDM 1602 - 65 MHz dot clock - 1024*768 dots - 60.2 Hz

                  -- pll = 13
LINETIME  := 332  -- 48.95 kHz Horiz.Scanning freq.
DISPLAY   := 256  -- 1024 dots
HALFSYNC  := 12   -- Horiz. Sync       = 1.5  us
BACKPORCH := 32   -- Horiz. Backporch  = 2.0  us
--           20      Horiz. Frontporch = 1.2  us

VDISPLAY  := 1536 -- 768 lines
VSYNC     := 6    -- 3   lines    = 61 us
VBLANK    := 72   -- 36  lines
                  -- VFP = PreEqualisation = VSync     =   61 us
                  -- VBP = VBlank + PostEqualisation  =  796 us
                  -- total lines a 20.4us           = 813 lines
                  -- total time for each image        = 16.6 ms
TRDELAY   := 18   -- 1.1 us
```

9.2.10 Philips CT 2064

```
-- Philips CT 2064 - 110 MHz dot clock - 1280*1024 dots - 60 Hz

                  -- pll = 22
LINETIME  := 430  -- 64 kHz Horiz.Scanning freq.
DISPLAY   := 320  -- 1280 dots
HALFSYNC  := 23   -- Horiz. Sync       = 1.7  us
BACKPORCH := 50   -- Horiz. Backporch  = 1.8  us
--           14      Horiz. Frontporch = 0.5 us

VDISPLAY  := 2048 -- 1024 lines
VSYNC     := 6    -- 3   lines    = 46 us
VBLANK    := 60   -- 30  lines
                  -- VFP = PreEqualisation = VSync     =   46 us
                  -- VBP = VBlank + PostEqualisation  =  516 us
                  -- total lines a 15.63us         = 1063 lines
                  -- total time for each image        = 16.6 ms
TRDELAY   := 28   -- 1.0 us
```

9.2.11 SONY GDM 1950

```
-- SONY GDM1601,1950 - 110 MHz dot clk - 1280*1024 dots - 60Hz
                  -- pll = 22
LINETIME  := 434  -- 63.36 kHz Horiz.Scanning freq.
DISPLAY   := 320  -- 1280 dots
HALFSYNC  := 23   -- Horiz. Sync       = 1.7  us
BACKPORCH := 57   -- Horiz. Backporch  = 2.1  us
--           11      Horiz. Frontporch = 0.4 us

VDISPLAY  := 2048 -- 1024 lines
VSYNC     := 6    -- 3   lines    = 47 us
VBLANK    := 46   -- 23  lines
                  -- VFP = PreEqualisation = VSync    =   47 us
                  -- VBP = VBlank + PostEqualisation =  410 us
                  -- total lines a 15.78us          = 1056 lines
                  -- total time for each image        =  16.7 ms
TRDELAY   := 28   -- 1.0 us
```

9.2.12 Silicon Graphics

```
-- Silicon Graphics - 105 MHz dot clk - 1280*1024 dots - 60 Hz

                       -- pll = 21
LINETIME  := 410  -- 64 kHz Horiz.Scanning freq.
DISPLAY   := 320  -- 1280 dots
HALFSYNC  := 15   -- Horiz. Sync       = 1.1  us
BACKPORCH := 58   -- Horiz. Backporch  = 2.2  us
--           2       Horiz. Frontporch = 0.08  us

VDISPLAY  := 2048 -- 1024 lines
VSYNC     := 6    -- 3   lines    = 46.0 us
VBLANK    := 64   -- 32  lines
                  -- VFP = PreEqualisation = VSync    =  46.0 us
                  -- VBP = VBlank + PostEqualisation =   546 us
                  -- total lines a 15.6us           = 1065 lines
                  -- total time for each image        =  16.6 ms
TRDELAY   := 28   -- 1.1 us
```

```
-- Silicon Graphics - 110 MHz dot clk - 1376*1024 dots - 60 Hz

                       -- pll = 22
LINETIME  := 430  -- 64 kHz Horiz.Scanning freq.
DISPLAY   := 344  -- 1376 dots
HALFSYNC  := 16   -- Horiz. Sync       = 1.2  us
BACKPORCH := 53   -- Horiz. Backporch  = 1.9  us
--           1       Horiz. Frontporch = 0.04  us

VDISPLAY  := 2048 -- 1024 lines
VSYNC     := 6    -- 3   lines    = 46.0 us
VBLANK    := 64   -- 32  lines
                  -- VFP = PreEqualisation = VSync     = 46.0 us
                  -- VBP = VBlank + PostEqualisation  = 547  us
                  -- total lines a 15.6us           = 1065 lines
                  -- total time for each image        =  16.6 ms
TRDELAY   := 28   -- 1.0 us
```

9.2.13 TV-Monitor - interlaced

```
-- TV Monitor - 20 MHz dot clock - 768*512 interlaced - 50.0 Hz
                 -- interlaced yet to set by control register!!
                  -- pll = 4
LINETIME  := 320  -- 15.625 Khz Horiz.Scanning freq.
DISPLAY   := 192  -- 768 dots
HALFSYNC  := 15   -- Horiz. Sync       = 6.0 us
BACKPORCH := 58   -- Horiz. Backporch  = 11.6 us
--           40      Horiz. Frontporch = 8.0 us

VDISPLAY  := 512  -- 256 lines
VSYNC     := 26   -- 13  lines     = 0.832ms
VBLANK    := 34   -- 17  lines
                  -- VFP = PreEqualisation = VSync    = 0.832 ms
                  -- VBP = VBlank + PostEqualisation =  1.92 ms
                  -- total lines a 64 us             = 312 lines
                  -- total time for each image       =    20 ms
TRDELAY   := 10


-- TV Monitor - 15 MHz dot clock - 512*512 interlaced - 50.0 Hz
                 -- interlaced yet to set by control register!!
                  -- pll = 3
LINETIME  := 240  -- 15.625 Khz Horiz.Scanning freq.
DISPLAY   := 128  -- 512 dots
HALFSYNC  := 15   -- Horiz. Sync       = 8.0 us
BACKPORCH := 46   -- Horiz. Backporch  = 12.3 us
--           36      Horiz. Frontporch = 9.6 us

VDISPLAY  := 512  -- 256 lines
VSYNC     := 26   -- 13  lines     = 0.832ms
VBLANK    := 34   -- 17  lines
                  -- VFP = PreEqualisation = VSync    = 0.832 ms
                  -- VBP = VBlank + PostEqualisation =  1.92 ms
                  -- total lines a 64 us             = 312 lines
                  -- total time for each image       =    20 ms
TRDELAY   := 10
```

9.3 Testprograms

Introduction to the test- and driver software

Concept

The supplied software is designed to explain the handling of GDS II and the programming of GDS II specific functions.
All given software is programmed under MULTITOOL OCCAM II.

Hardware and Software needed to run demo programs:

- IBM PC/XT/AT with BBK-PC adaptor
- Standard transputer-node as HOST (MTM2, MTMPC, TPM4, TPMPC, TPMIO, etc.)
- Parsytec Multitool (formerly named Megatool).

The main purpose of the given routines is to demonstrate the operation and programming of the board. With the exception of the library 'GDSII.util' most of the given files may not even be used in your application, so don't stick too close to the way we programed but understand the principles of the board and write your own !

How to use the software:

You will find a PC formated disk that holds a batch file and all utilities you need.
Put the supplied diskette in your drive, log to that drive and simply call " `install <sourcedrive>` " (e.g. " `install c` " for installation on drive c:). The batchfile will copy everything to a subdirectory named "GDSDEMO" under MTOOL

Functionality of given programs

GDSII.util:

GDSII.util is the central library file and supports the user with commands (that means the necessary procedures and functions) to directly access the hardware. These library routines operate on the lowest level. Some of the supplied functionalities are:

```
<x> = input variable    x = output variable

event (EventError)                              wait for event
check.for.event (EventError,IdentError)         see if event has come
vertical.scan (EventError,IdentError,Scan)      calculates vertical scan freq.
        EventError (BOOL) := TRUE               no event since 50ms
        IdentError (BOOL) := TRUE               error checking " Ident Register "
        Scan (REAL32)                           vertical scan freq.

set.dot.clock (<pll>)                           set pixel dot clock
        pll (INT)                               multiplication factor for PLL

different.palettes                              set CLUT with some palettes

delay (<x>)                                     just a delay routine
        x (INT)                                 delay in x times of 10ms

simple pictures                                 draw patterns

G300 register operations                        G300 register operations

        read.Control (Contr)                    read " Control Register "
        write.Control (<Contr>)                 set " Control Register "
                Contr (INT)                     value stored in " Control Register "

        disable.VTG ( )                         starts the " Video Timing Generator "
        enable.VTG ( )                          stops the " Video Timing Generator "

        read.datapath (datapath)                read " Datapath Registers "
        write.datapath (<datapath>)             write " Datapath Registers "
                datapath ([20]INT)              values stored in the " Datapath Registers "

ask.for.interlace (interlace)                   check for interlace mode
        interlace (BOOL) := TRUE                interlaced mode detected
```

Be careful on changing the way the hardware is called, since the way most operations are handled is highly hardware dependant. We suggest that you program your own graphic drivers on using these routines (which are easy to call and mostly optimized on speed) or that your alter these routines only after a very close analysis of the given examples !
Nevertheless, don't be afraid of opening the given folds and read through all routines.

DATAPOOL

Datapool is a library file and holds all monitor specific data (such as timing, horizontal and vertical frequencies, etc.) and contains no active routines. Its simply a library and can be extended to suit your needs by adding more datasets. To create a new dataset for your monitor or your specific resolution, just copy an existing fold and change the timing data to your desired (or more likely ' tested ') parameters.

MONITOR.util

MONITOR.util is a procedure called by the actual demo program and selects the desired dataset from datapool. The calling parameters are: monitor.type, resolution, mode, pll, linear.
After giving (in integer values) the monitor.type, resolution number, mode number and the Boolean value 'linear' (TRUE for "*Linear Addressing*"[1] and FALSE for "*Row-Oriented Addressing*"[2]), *MONITOR.util* loads the corresponding data from *datapool* and programs the GDS registers by using routines from the library file *GDSII.util.*
PLL is returned by procedure and represents the multiplication factor for the internal pll loop of the G300[3].

The general call syntax is:

MONITOR.util (<monitor.type>,<resolution>,<mode>,pll,<linear>)

Description of parameters:

monitor.type (INT)

0	TV Monitor
1	EIZO Flexscan 8060
2	EIZO Flexscan 9070
3	EIZO Flexscan 9500[4]
4	NEC Multisync GSII
10	NEC Multisync XL[5]
12	Sony GDM 1601[5]

[1] See chapter 3.2.2

[2] See chapter 3.2.1

[3] See chapter 3.4.1

[4] Only the new version of the EIZO 9500 was tested !

[5] Only mode 1 (8 bit/pixel) : "*Pixel Clock*" > 32 MHz

14	Sony GDM 1602,1650[6]
16	Philips CCT 2064[6]
18	Silicon Graphics HR Monitor[6]

resolution (INT)

The resolution depends on the monitor.type. Look inside the fold *datapool* and choose the right value:

(example for fixed Pixel Clock with EIZO 8060)

1	640 * 480
2	720 * 540
3	768 * 576
4	800 * 600

linear

set linear to TRUE for " *Linear Addressing Mode* "
set linear to false for " *Row-Oriented Addressing Mode* "

Example for a EIZO 8060 with a 640 * 480 resolution and " Row-Oriented Addressing Mode " (Hardware Panning possible). GDS-II jumpered to 8 bit per pixel.

```
#USE GDSII.util
#USE MONITOR.util
VAL monitor.type IS 1     :
VAL resolution   IS 1     :
VAL mode         IS 1     :
VAL linear       IS FALSE :
INT pll                   :
.
.
MONITOR.util ( monitor.type, resolution, mode, pll, TRUE )
```

[6] Only mode 1 (8 bit/pixel) : " Pixel Clock " > 32 MHz

INIT.util

INIT.util sets " *Pixel Clock* ", " *Control Register* ", " *Mask Register* ", " *Datapath Register* ", CLUT and starts the VTG. No additional commands are required to initialize the G300.

The general call syntax is:

INIT.util (<monitor.type>,<resolution>,<mode>,pll,<linear>,Error)

monitor.type, resolution, mode, pll, linear: see *MONITOR.util*
Error: Integer value, gives you back an error code.
Error := 0: No errors during *INIT.util*

DEMO programs

Every program needs an additional transputer node as a host. Connect link 2 of your host transputer to link 0 of the GDS-II. Enter the directory " \MTOOL\GDSDEMO " and start " *multitool* " with the command " MTOOL ". Than you can enter the several folds and load the program code by pressing " <ALT>4 ". The program can be started by loading the EXE " *Link Monitor* " with key " <F5> and by starting the EXE with key " <F6> ".

Real colour demo:

This program only runs in mode 2. The resolution is fixed to 768 * 576 dots.

Demo:

This program is menu driven. You can choose between mode 1, mode 2, mode 3, linear- and " *Row-Oriented Addressing Mode* ", different monitor types and different resolutions. You will also find some simple drawing commands for testing the GDS-II and your monitor.

Programming a new video timing:

After choosing resolution and monitor type restart the program with " *back to main menu* ". You are asked once again for a monitor type. Choose now the command " *user defined* " and you will enter a menu where you can change all the different datapath registers. The old values will be used as default for every new cycle.

Testing the DB-DMA

Connect link 0 of the DB-DMA to link 2 of the GDS-II.
Enter the fold " *demo* " and find the definition for processor 1. Remove the " *comment fold* " and recompile the program " *demo* ".

9.4 Installation

9.4.1 Installation on Multicluster system units

When installing the GDS-II module within Multicluster system or expansion units you may use the BNC-connectors at the rear of the unit for feeding the video signals to your graphic monitor.

The installation should only be done, if the unit is switched off !

- you can insert the GDS-II module into any desired slot, even though the left slots are preferred

- the GDS-II module will normally be used as a network processor, so connect its link 0 to link 2 of your host transputer

- The following step depends on your pcb rear backplane version, which is easy to identify. In one version, the rear backplane contains a few integrated circuits (refered to as " *active rear backplane* "). The other version does not contain any active devices (referred to as " *passive rear backplane* ").

- *active rear backplane:*
  Connect the video output of the GDS-II module (this is link 6 of the backplane) to the active rear backplane's video link plug (this is the left-most connector when viewed from the front of the MultiCluster system / expansion unit) with a standard internal link cable.

- *passive rear backplane:*
  Connect the video output of the GDS-II module (this is link 6 of the backplane) to the video link plug of the active rear backplane (this is the left-most connector when viewed from the front of the MultiCluster system / expansion unit) using a special video link cable.
  This video cable has a 1 to 1 plug connection instead of the inverted plug connection used in the standard link cables for internal use. In addition, lines 6 and 7 are crossed in the special cable. This special video link cable should only be used within MultiCluster units !

- Connect your monitor to the BNC-connectors. All RGB-signals also carry the synchronization signals. When using the "*passive rear backplane*" you should be careful about a few incorrectly labelled signals at the rear of the unit. Youd better switch the labels HSYNC and VSYNC !

9.4.2 Installation In IBM PC/XT/AT Or Compatibles

When installing the GDS module within IBM PC/XT/AT or compatible computers you will need a BBK-PC adaptor. This has to be fixed to the GDS-II. After removing the front cover of the GDS-II both modules form an add-on board that can be inserted into a long IBM slot. The GDS is normally used as a network processor, so it may be necessary to disable the BBK-PC adaptor to avoid addressing conflicts with other adaptors (see the technical documentation of the BBK-PC adaptor).
To feed the video signals to your graphic monitor, connect the video output of the GDS-II module (Link 6 of the BBK-PC) to one of the external 8-pin Link connectors at the back side of the adaptor using a standard flat Link cable.

If you've got a new version of the BBK-PC, watch for the additional power connectors on the upper side of the board. Connect the two power connectors to one of the 4-way power connector inside the PC.

9.5 Power Requirements

Operating Temperature[6]:	0 to 70 °C
Power Supply Voltage:	+ 5 Volt $^{+}/_{-}$ 5%

Power Supply Current; GDS-II:

-	standby; G300 is not working:	ca. 1.4 A
-	standby; G300 / mode1 runs at 35 MHz:	ca. 1.5 A
-	standby; G300 / mode1 runs at 110 MHz:	ca. 1.8 A
-	standby; G300 / mode2 runs at 32 MHz:	ca. 1.85 A
-	blockmove; G300 / mode2 runs at 32 MHz:	ca. 2.2 A

Power Supply Current; GDS-II + DB-DMA:

-	standby; G300 is not working:	ca. 1.9 A
-	standby; G300 / mode2 runs at 32 MHz:	ca. 2.3 A
-	blockmove; G300 / mode2 runs at 32 MHz:	ca. 2.6 A

Power Supply Current; GDS-II + DB-DMA + DB-CLUT:

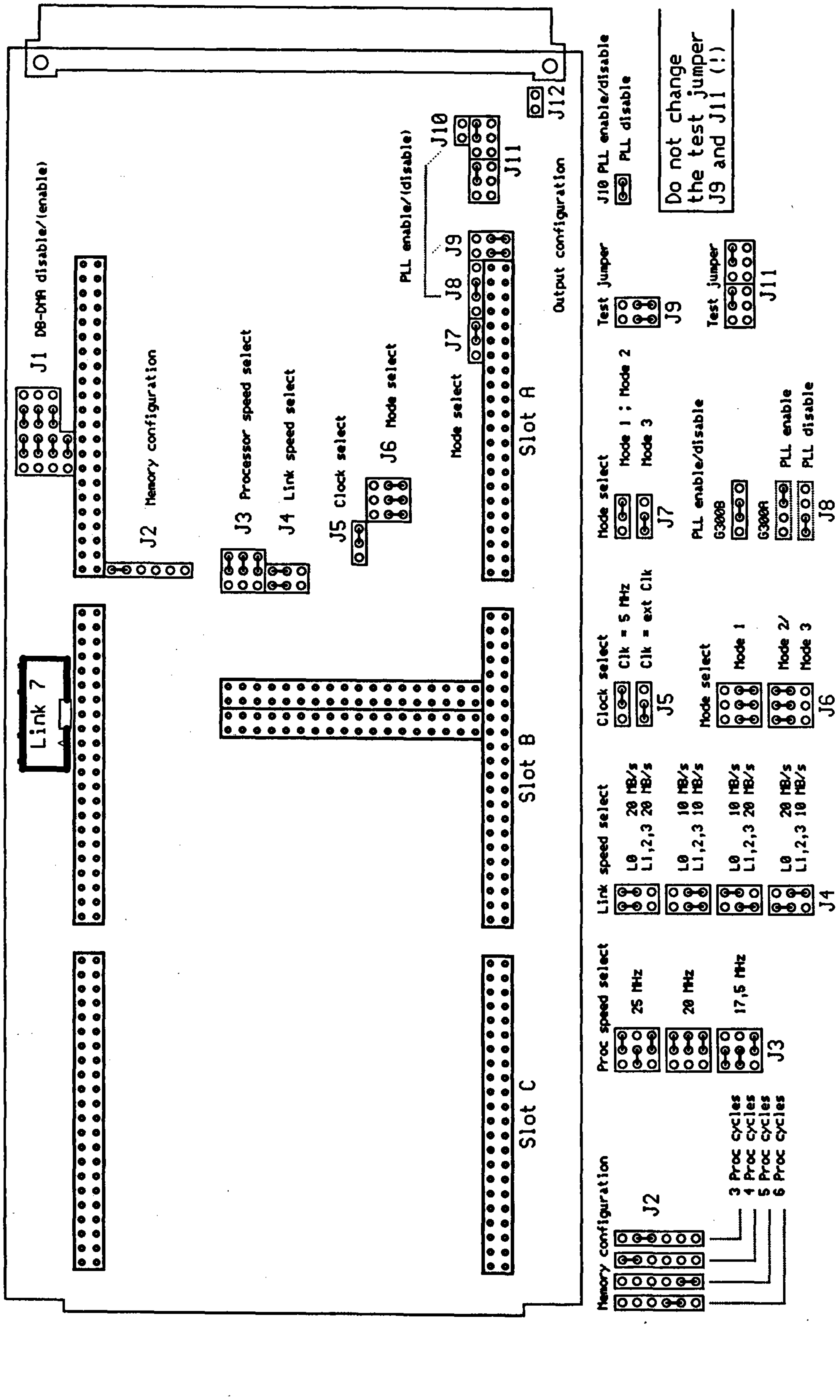
-	standby; G300 is not working:	ca. 2.5 A
-	standby; G300 / mode3 runs at 32 MHz:	ca. 2.9 A
-	blockmove; G300 / mode3 runs at 32 MHz:	ca. 3.2 A

When installing your GDS-II in an IBM PC/XT/AT make sure, that your Power Supply is " strong enough ". Use the additional Power Connectors on your BBK-PC.

The best way to run your GDS-II is using a " MultiCluster " unit. Power supply and cooling fans are provided for these kind of boards !

[6] Sometimes additional cooling is required !

9.6 Jumper overview



9.7 GDS-II Addressmap

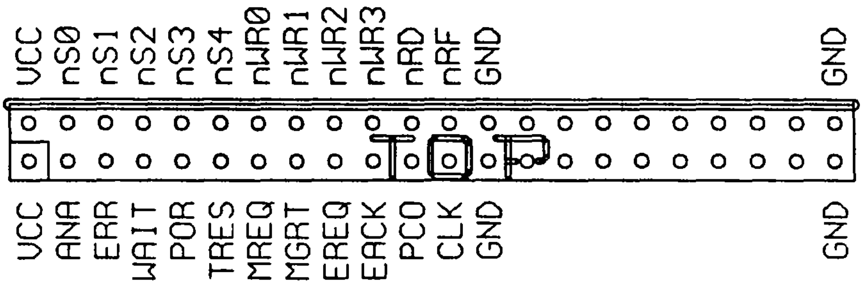
Hardware-addr.	OCCAM-word-addr.	Name	function
8000.0000	#0000.0000	DRAM - Start	2 MByte working memory
801F.FFFF	#0007.FFFF	DRAM - End	
8020.0000	#0008.0000	VRAM - Start	2 MByte video memory
803F.FFFF	#0009.FFFF	VRAM - End	
0000.0000	#2000.0000	Ident-register	bit0 = 0
		(read-only)	bit8 = 1: Event by VSYNC ("Frame" = active)
0000.00C0	#2000.0030	Reset-register	bit0 - 3: Link Reset Out
		(write-only)	bit4: Reset for G300
0008.0000	#2002.0000	PCS0 - Start	Chipselect 0 for DB-Slot A (1/2 MByte)
000F.FFFF	#2003.FFFF	PCS0 - End	
0010.0000	#2004.0000	PCS1 - Start	Chipselect 1 for DB-Slot A (1/2 MByte)
0017.FFFF	#2005.FFFF	PCS1 - End	
0018.0000	#2006.0000	PCS2 - Start	Chipselect 0 for DB-Slot B (1/2 MByte)
001F.FFFF	#2007.FFFF	PCS2 - End	
0020.0000	#2008.0000	PCS3 - Start	Chipselect 1 for DB-Slot B (1/2 MByte)
0027.FFFF	#2009.FFFF	PCS3 - End	
0028.0000	#200A.0000	PCS4 - Start	Chipselect 0 for DB-Slot C (1/2 MByte)
002F.FFFF	#200B.FFFF	PCS4 - End	
0030.0000	#200C.0000	PCS5 - Start	Chipselect 1 for DB-Slot C (1/2 MByte)
0037.FFFF	#200D.FFFF	PCS5 - End	
0038.0000	#200E.0000	CS-CLUT - Start	ChipSelect for 12 bit-expansion (DBCLUT)
003F.FFFF	#200F.FFFF	CS-CLUT - End	
0040.0000	#2010.0000	Base address G300	
0040.0000	#2010.0000	CLUT - Start	G300's Internal CLUT (256 Words á 24 bit)[7]
0040.03FC	#2010.00FF	CLUT - End	
0040.0500	#2010.0140	Mask register	Read/write
0040.0580	#2010.0160	Control Register	Read/write
0040.0600	#2010.0180	Top of Screen	Read-only
0040.0680	#2010.01A0	Boot Location[8]	Write-only !
0040.0484	#2010.0121	Datapath Register - Start[9]	
0040.04B0	#2010.012C	Datapath Register - End	

[7] The tranputers addresses 256 words á 32 bit but only the lowest three bytes of every word are used !

[8] Once programmed " *Boot Location* " can't be changed.

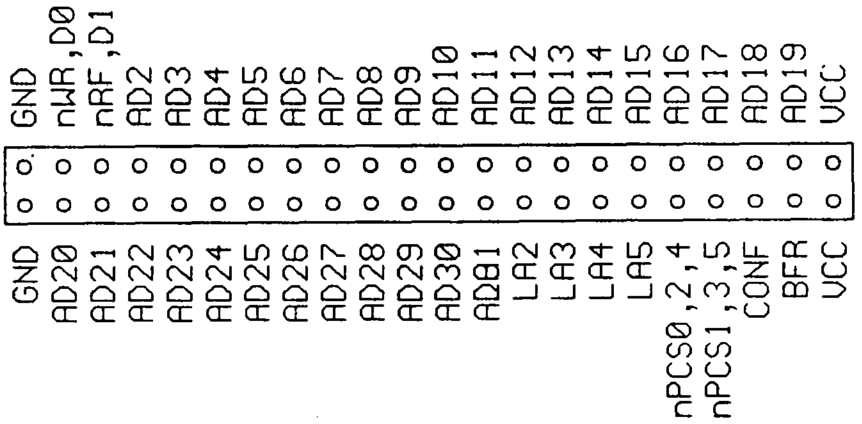
[9] Access to Datapath Registers only possible with enabled VTG !

9.8 Pinout of the daughterboard slot



DBIx Slot Connector

Pin location



9.9 Pinout Of The (96-Way) DIN 41612 Connector

The pinout is compatible to the 8 Link PARSYTEC boards (MTM2, TPM-IO). The video signals are placed at the position of Link 6. The signals of Link 6 are connected to an additional 10-pin BERG-connector on the board.

	c	b	a
1	Reset 0 out +	Reset 1 out +	Reset 0 out -
2	Link 0 out +	Reset 1 out -	Link 0 out -
3	GND	Link 1 out +	GND
4	Link 0 in -	Link 1 out -	Link 0 in +
5	Reset 0 in -	Link 1 in -	Reset 0 in +
6	Link 1 in +	Reset 1 in -	Reset 1 in +
7	Reset 2 out +	Reset 3 out +	Reset 2 out -
8	Link 2 out +	Reset 3 out -	Link 2 out -
9	GND	Link 3 out +	GND
10	Link 2 in -	Link 3 out -	Link 2 in +
11	Reset 2 in -	Link 3 in -	Reset 2 in +
12	Link 3 in +	Reset 3 in -	Reset 3 in +
13	Reset 4 out +	Reset 5 out +	Reset 4 out -
14	Link 4 out +	Reset 5 out -	Link 4 out -
15	GND	Link 5 out +	GND
16	Link 4 in -	Link 5 out -	Link 4 in +
17	Reset 4 in -	Link 5 in -	Reset 4 in +
18	Link 5 in +	Reset 5 in -	Reset 5 in +
19	HSYNC	Reset 7 out +	GND
20	BLUE	Reset 7 out -	GREEN
21	GND	Link 7 out +	GND
22	RED	Link 7 out -	HCLK
23	VSYNC	Link 7 in -	GND
24	Link 7 in +	Reset 7 in +	Master Reset
25		Reset 7 in -	
26		LinkSpeed	
27	+ 5V	+ 5V	+ 5V
28	+ 5V	+ 5V	+ 5V
29	+ 5V	+ 5V	+ 5V
30	GND	GND	GND
31	GND	GND	GND
32	GND	GND	GND

9.10 Index - Register