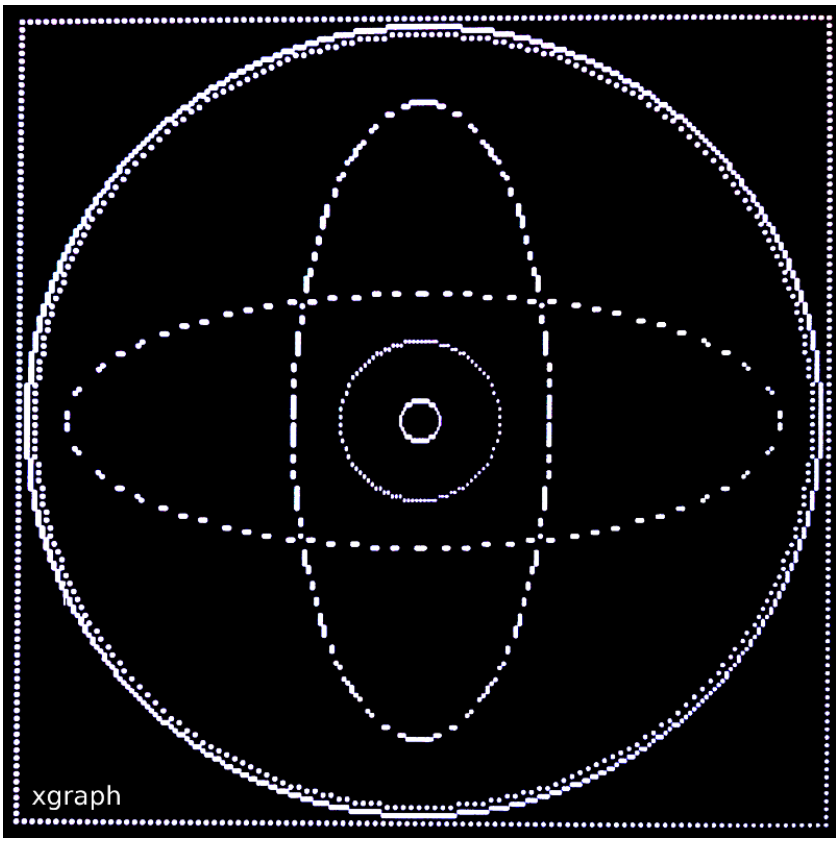
Multicomp Z80 Graphic Extension

Reference-Manual



RSX-Call-Name:	Func.-No.:	ASM-Function:
X_Hello	0	Hello
X_RSXVersion	1	RSX-Version
X_RSXName	2	RSX-Name
X_Initgraph	3	Initgraph
X_GetSTAT	4	GetStat
X_SetPxMode	5	SetPxMode
X_SetTxMode	6	SetTxMode
X_GRON	7	GRON
X_ACON	8	ACON
X_GClrScr	9	GClrScr
X_PutScrRC	10	PutScrRC
X_WrToFnROM	11	WrToFnROM
X_RdfrFnROM	12	RdfrFnROM
X_SetTxFnt	13	SetTxFNT
X_ResTxFnt	14	ResTxFnt
X_SetLnSty	15	SetLnSty
X_SetPatRot	16	SetPatRot
X_RotatePat	17	RotatePat
X_ReLoadPat	18	LoadPat
X_PrintChr	19	OutChr
X_PrintStr	20	OutStr
X_PltPix	21	PltPix
X_GetPix	22	GetPix
X_GetPxMask	23	GetPxMask
X_ScrPortRd	24	ScrPortRd
X_ScrPortWr	25	ScrPortWr
X_PutChRC	26	PutChRC
X_GetChRC	27	GetChRC
X_WrChToAddr	28	WrChToAddr
X_RdChfrAddr	29	RdChToAddr
X_GetBmpRC	30	GetBmpRC
X_PutBmpRC	31	PutBmpRC
X_WriteBmpRC	32	WriteBmpRC
X_CalcRC	33	CalcRC
X_CalcXY	34	CalcXY
X_SetQuad	35	SetQuad
X_ResQuad	36	ResQuad
X_ChkQuad	37	ChkQuadStat
X_GetStruct	38	GetStruct
X_SetRBox	39	SetRBox
X_FnLine	40	FnLine
X_FnLineWH	41	FnLine2
X_FnBox	42	FnBox
X_FnEllipse	43	FnEllipse

Inhaltsverzeichnis

Files in the xgraph-Package

Text-Files:	***File-Size:***	
Ref-Manual.pdf	631060 Byte	The same as pdf
xgraph.pdf	23600 Byte	Info text regarding the xgraph-package
xreadme.pdf	16400 Byte	Short summary of the .com files
Char-Fonts:		
CGABlkGrf.bin	2048 Byte	Semi block graphic font-rom
CGAFnIBM.bin	2048 Byte	Grants original font-rom
CGAFnSym.bin	2048 Byte	A narrower font with w/ special sym.
CGAMdrn.bin	2048 Byte	The system font-rom
CPM-Binaries:		
xcls.com	89 Byte	Clear-Scren for the graphic-screen
xdemo1.com	12928 Byte	General feature demo
xdemo2.com	11136 Byte	Specific demo regarding Box, Circle & Elipse
xdemo3.com	12032 Byte	Specific demo regarding the ellipse-func.
xdemo4.com	15360 Byte	Bitmap demo
xdemo5.com	15488 Byte	Programing the system font-rom
xdemo7.com	12672 Byte	Demo for using a ext. Font-rom
xdemo8.com	12978 Byte	Demo displays the FnEll-Struct. data
xgrapfnt.com	12800 Byte	App for Load/Save of system font-rom
xgraph.rsx	6016 Byte	The xgraph RSX
xoff.com	5 Byte	Switch graphic-screen OFF
xon.com	5 Byte	Switch graphic-screen ON
xsetfnt.com	11776 Byte	App to reload the system font-rom
xsetibm.com	11776 Byte	App to reload the Grant‘s font-rom
xsetmdrn.com	11776 Byte	App to reload a futuristic font-rom
xsetsym.com	11776 Byte	App to reload the narrower font-rom
.h Header-Files:		
xbitmap.h	4355 Byte	Header file with the bitmap-functions
xchrdef.h	3986 Byte	Header file with the chrdef-functions
xdraw.h	11781 Byte	Header file with the bitmap-functions
xgraph.h	10267 Byte	Header file with the bitmap-functions
xkeyboard.h	187 Byte	Header file with the bitmap-functions
xsys.h	5623 Byte	Header file with the bitmap-functions
xtext.h	3571 Byte	Header file with the bitmap-functions
Assembler-Sources:		
xcls.asm	2502 Byte	Asm-file of graphic clear-screen function
xgraph.asm	144909 Byte	Asm-file of the xgraph RSX
xoff.asm	106 Byte	Asm-File for graphic-screen OFF
xon.asm	105 Byte	Asm-File for graphic-screen ON
C Source-Files		
xdemo1.c	7801 Byte	Source file of xdemo1
xdemo2.c	4439 Byte	Source file of xdemo2
xdemo3.c	11622 Byte	Source file of xdemo3
xdemo4.c	29831 Byte	Source file of xdemo4
xdemo5.c	17528 Byte	Source file of xdemo5
xdemo7.c	15779 Byte	Source file of xdemo7
xdemo8.c	15779 Byte	Source file of xdemo8
xgrapfnt.c	4687 Byte	Source file of xgrapfnt
xsetfnt.c	15398 Byte	Source file of xsetfnt
xsetibm.c	13462 Byte	Source file of xsetibm
xsetkaun.c	13593 Byte	Source file of xsetkaun
xsetmdrn.c	13573 Byte	Source file of xsetmdrn
xsetsym.c	13751 Byte	Source file of xsetsym

How the Graphic is build

The following text will not give in detail information how Grant Searles hardware will work. If this is wanted, please visit his web-pages for more information. The explanations here will give only a more general view on the terminal hardware and how the graphics part is connected the ASCII-screen.

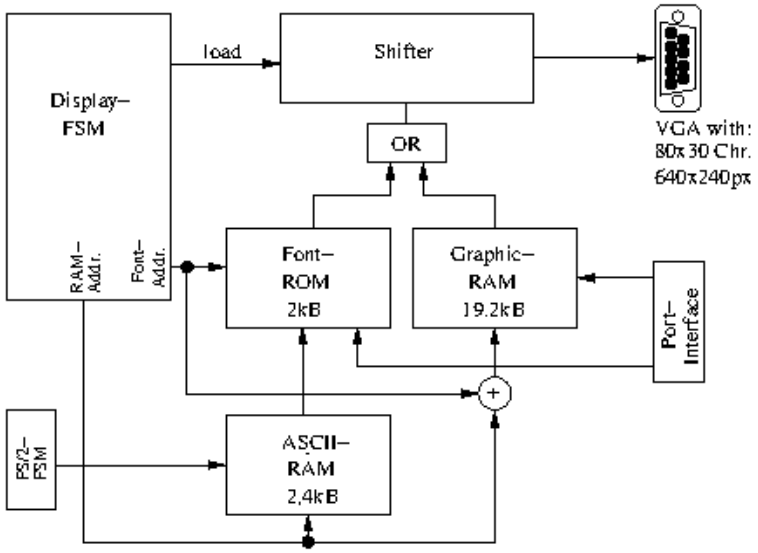


Figure 1: Hardware Model

As can be seen in the block diagramm there is no screen RAM, instead we have a very large „programmable Font-ROM“ in use. The size of the Font-ROM is determined by the number of characters to display and the „word“-size of the ASCII-RAM, here 8 bit = 256 char. The interesting

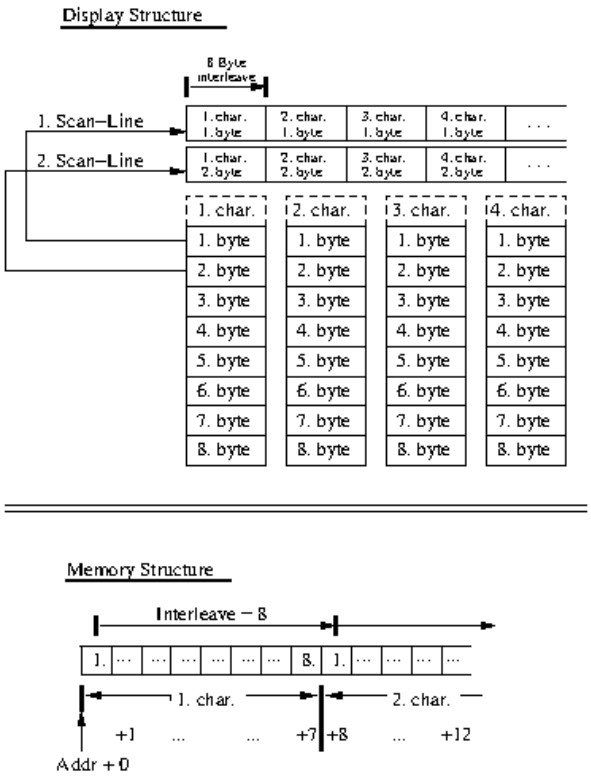


Figure 2: Memory Model

part is the char.-matrix of 8x8 bit: 8 bit horiz. = 1 byte and 8 byte vert. in hight. The multiplication of the ASCII-RAM size with the 8 byte per char. gives the graphic RAM size of 19200 bytes.

The displayed structure of the graphic RAM isn‘t linear, it‘s char-cell oriented in the same way as the Font-Rom. That means 8 byte in sequence form one charcater but when displayed within a scanline, the bytes are red with a interleave of 8. The increment from one scanline to the next is 1. That means, the display-FSM reads each character 8 times until a full text line is visible. Then the next line and so on. Finally, after 30 lines the screen is filled with all character positions. Due to the limited amount of Block-RAM in a Cyclone-2 (and Cyclone-IV) FPGA, a Scan-Doubler is used to save Block-RAM. The doubler stretches the characters (and pixel !) from squared to rectangular shape and 30 lines are enaough to fill up the whole screen (in the VHDL-Code there is a defined constant „VERT_PIXEL_SCANLINES“ to adjust this „doubling“ from 1 to n, I haven‘t played much with this, a value of „2“ is the right one for a 80x30 full screen display. One uncompfortable result of this is the mentioned rectangular nature of all pixel. If ths is not taken into account, everything on screen is distorted vertical. On the software side this is compensated in two ways: when in „HiRes“-Mode (640x240px), the distance in X is doubled, this is valid for all line graphic, or, when in „LoRes“-Mode (320x240px) the pixel is double in X-direction. All character printing on screen is done in „normal“ HiRes-Mode (regardles whether in RC- or XY-Mode), so text looks the same as on the ASCII-Screen. When switching to Double-Width-Text (only possible in XY-Mode), Pixel-Doubling in HiRes-Mode is used (speciality: to prevent the text from jumping to the left, the X-position is doubled, the reason is the general use of a 320x240px coord. system).

The Way Text is processed

All text movements are done through a 8 byte long ‚ChrBuf‘. Manipulation of single char. is done when the data is in this buffer. At present only ‚invers text‘ is possible. When requested, all 8 bytes are XORed with 0xFF before they are copied to there destination. The 8x8 Matrix of a character doesn‘t make it easy to manipulate its shape without making it unreadable. That‘s the reason why I didi not implement more text manipulation options. One way to experiment with this is a Font-Editor. My results ranged from „hard to read“ to „unusable“. Try to install fonts like ‚xsetmdrn‘ = ‚hard to read‘ or compile „xsetkaun.c“ = unusable‘. The latter characterset has only capital letters !. In the file ‚xchrdef.h‘ all text related functions are summarized.

Address- and Pixel-Mask Calculation

The confusing thing with pixel-addressing is the situation, that 8 pixel reside in 1 byte, the address of this byte is constant for 8 pixel. We have a interleave of 8 from character to character and a increment of 1 fron scanline to scanline. In 320x240px resolution 1 byte holds 4 pixel and in RC-Mode only the byte address is important. Finally we end up with 3 equiations for all this. Due to the use of a bit-flag, the address-calculation can be done in one equiation plus one for the RC-calculation…;-)

To make it short, next the equiation-list:

For Line-Graphic: (RSX-Function „CalcXY“)

```
ADDR  = ((XPOS*(640=1|320=2) AND FFF8h)+(YPOS AND 07H)+(80*(YPOS AND FFF8h)))
MASK  = BMASK((XPOS AND PMASK)+OFFSET)
          OFFSET :=   0 = 320,   4 = 640
          PMASK  := 03h = 320, 07h = 640
```

For RC-Text: (RSX-Function „CalcRC“)

```
ADDR  = (8*XPOS + 640*YPOS)
```

The Pixel-Masks are predefined, that simplifies the set/clear/invert operation. The calculated bit *number* is used as a index into a one-dimensional bit-mask array. A offset of 4 is used to distinguish between the two resolutions when indexing into the array.

The STAT-Register area

`STAT` is the most important one of several 16-bit register, they all controll or hold data that is important for proccess flow in the RSX-Module:

```
; ---------------------------------------------------------------------------
; STAT Area
; ---------------------------------------------------------------------------
oldsp:  defw 0        ; Old SP of CALLer
; ---------------------------------------------------------------------------
par1:   defw 0        ; 'X0'            Parameter #1
par2:   defw 0        ; 'Y0'            Parameter #2
par3:   defw 0        ; mode|radius     Parameter #3
par4:   defw 0        ; CHAR|'X1'|width Parameter #4
par5:   defw 0        ; ADDR|'Y1'|hight Parameter #5
par6:   defw 0        ; STAT            Bit-Flag Reg. for Graphic-Management
; ---------------------------------------------------------------------------
;addr of ext. font-rom
par7:   defw 0        ; FNTADDR     external Font address or '0' for internal
; ---------------------------------------------------------------------------
;active/selected line pattern
PATNUM:  defw 0        ; PATNUM = 0  Holds Number of selected line pattern
SELPAT:  defw LnePat0  ; SELPAT = 0  Holds Selected Line-Pattern (by 'SetLneSty')
LNEPAT:  defw LnePat0  ; LNEPAT = 0  Holds Active Line Pattern
QUADRT:  defw S_ALL    ; QUADRT = -1 Holds active quadrants of FnCircle/FnEllipse
rbwidth: defw 0        ;             Holds the width when 'PltRBox' is requested
rbhight: defw 0        ;             Holds the hight when 'PltRBox' is requested
; ---------------------------------------------------------------------------
LPattern:
        defs 16         ;8 Words for Line Pattern
; ---------------------------------------------------------------------------
```

When calling the RSX the reg. `par1..par5` are filled with the data from `x_par[0]..x_par[4]` supplied by the application. Next the function code in `x_func` is used to jump to the requested

function, which processes the data in par1..par5. par6 is the STAT-reg. Its bit-definition is as follows:

```
;############### Definition of 'STAT'-Register = 'par6' ################
; ---------------------------------------------------------------------
; From ‚Initgraph‘ ininitialized reg./bit-flags for graphics mode = '320'
; ---------------------------------------------------------------------
; Entry: --
; Exit: (GRON, ACON)    := 'ON'
;       (SCREEN, HIRES) := 0 = '320'
;       (RCXY)          := 0 = '80x30' := 'RC'
;       (TMODE, TWIDTH) := 0 = Text noninverted, 0 = normal width
;       (TxFNT, TxADDR) := 0 = use internal Font,TxADDR = 0x0 ('internal')
;       (LnePat)        := LnePat0 = active linestyle pattern
;       (USEPAT)        := 0=No Pat. in LineGraphic
;       (PATROT)        := 0=Pat. rot. ON
;       (DBLBIT)        := 0=No DblBit in 'PutScrByRC'
; ---------------------------------------------------------------------
;Bits of 'par6' = Reg. 'STAT':
; ---------------------------------------------------------------------
;=== STAT+0:
; BIT(0)      = SCREEN          (0=320|1=640)      CalcXY Resol. Control
; BIT(1)      = RCXY            (0=80x30|1=320x240|640x240) Text-Graphic
; BIT(2)      = TWIDTH          (0=8x8-Nwidth|1=16x8-Dwidth)Text-Graphic
; BIT(3)      = TMODE           (0=norm|1=inv)              Text-Graphic
; BIT(4)      = HIRES           (0=320|1=640)               Text-Graphic
; BIT(5..6)   = PXMODE = '0..3' (0=SET|1=CLR|2=INV|3=2)  Pixel-Operation
; BIT(7)      = TXTFNT          (0=internal|1= external)     Font-Source
; ---------------------------------------------------------------------
;=== STAT+1:
; BIT(8)      = USEPAT          (0=No Pat.|1=Use Pat.        LineGraphic
; BIT(9)      = PATROT          (0=Pat. rot. ON|1=Pat. rot. OFF LineGra.
; BIT(10)     = DBLBIT          (0=No DblBit|1=DblBit, used in 'PutScrByRC'
; BIT(11..15) = ......           Reserved for future use
; ---------------------------------------------------------------------
```

The Data-Struct. of FnEllipse

When calling FnEllipse or its aliases, a special structure is filled with data regarding the plotted ellipse. The returned pointer from FnEllipse points to `O_ELL: the „Ground-Zero"` of the array. The function `SearchStruct()` skips the first 20 BYTEs and starts its run at label `OctCoord0:` because the first value of `O_ELL:` or `O_BXEL:` might be misinterpreted and can cause trouble. In MESCC these first 10 WORD‘s can be simply made excessable as follows:

```
int *O_ELL, *O_BXEL;
```

```
int *OctCoord;

SetQuad(XO_NULL); /* Do not plot anything */
OctCoord = FnEllipse( x0, y0, r1,r1);
O_ELL  = OctCoord;
O_BXEL = OctCoord + 10;
Array-Length   = O_ELL[0] >> 8; /* Struct-Length in Hbyte of O_ELL[0] */
X-Center-Coord = O_ELL[1];
[...]
Rbox-Flag  = O_BXEL[0] & 0x00FF; IF 255 THEN Rbox  ELSE 0 = No
HiRes-Flag = O_BXEL[0] >> 8;     IF 255 THEN HiRes ELSE 0 = No
Rbox-Width = O_BXEL[3];          View the sketch shown in Function SetRBox()
                                 for detailed information !
[...]
```

Other C-Compiler will have simular ways to extract the data of the first 10 WORD‘s.

```
;-------------------------------------------------------------------------
;Coord.-Array base address for pointer to CALLer... All Data in this
;Array is ONLY valid for the actual plotted ellipse. Any new ellipse plot
;updates this data for the newly plotted ellipse ! If this data is needed
;later, it should be copied using the returned pointer. Valid data is from
;label 'OctCoord:' to 'OctCoord1:' = 10 * 5 WORD's= 50 WORD in total !
;
;Remark: SRhight & SRwidth are only half the value. That simplifies X/Y-coord.
;======= calc. of the box-edges when drawing a surrounding box or parts of that.
;
;           +==+==+==+==+==+
OctCoord:;  |+0|+2|+4|+6|+8| <= Offset in Bytes !
;===========+==+==+==+==+==+=============================================
O_ELL: defw   0, 0, 0, 0, 0  ; Data of Ellipse when called
;             |  |  |  |  +--- Hight Radius
;             |  |  |  +------ Width Radius
;             |  |  +--------- Y-Center Coord.
;             |  +------------ X-Center Coord.
;             +--------------- LByte= plotted Oct. Mask, HByte = Array-Length
;                              HByte is set in 'ClrOctArray' Routine !
O_BXEL: defw  0, 0, 0, 0, 0  ; Data of RBOX-Ellipse (when requested) ELSE '0'
;             |  |  |  |  +--- RBhight = RBox-Hight between rounded corners
;             |  |  |  +------ RBwidth = RBox-Width between rounded corners
;             |  |  +--------- SRhight of surrounding rectangular box
;             |  +------------ SRwidth of surrounding rectangular box
;             +--------------- LByte = -1 if RBox  used ELSE '0',
;                              HByte = -1 if HiRes used ELSE '0'
;-------------------------------------------------------------------------
;           +==+==+==+==+==+
;           |OL|X0|Y0|X1|Y1|   'O' = LByte:  Oct.-Num, 'L' = HByte:Linestyle
OctCoord0: ;|+0|+2|+4|+6|+8| <= Offset in Bytes !
```

```
;==========+==+==+==+==+==+=========================================
O_NNW:  defw  0, 0, 0, 0, 0  ; Coord. North-North-West Octant
O_NNE:  defw  0, 0, 0, 0, 0  ; Coord. North-North-East Octant
O_WWN:  defw  0, 0, 0, 0, 0  ; Coord. West-West-North  Octant
O_EEN:  defw  0, 0, 0, 0, 0  ; Coord. East-East-North  Octant
O_SSW:  defw  0, 0, 0, 0, 0  ; Coord. South-South-West Octant
O_SSE:  defw  0, 0, 0, 0, 0  ; Coord. South-South-East Octant
O_WWS:  defw  0, 0, 0, 0, 0  ; Coord. West-West-South  Octant
O_EES:  defw  0, 0, 0, 0, 0  ; Coord. East-East-South  Octant
;             |  |  |  |  +--- Oct. End Y-Coord.
;             |  |  |  +------ Oct. End X-Coord.
;             |  |  +--------- Oct. Start Y-Coord.
;             |  +------------ Oct. Start X_Coord.
;             +--------------- LByte= Oct. Number, HByte = Linestyle
;-------------------------------------------------------------
OctCoord1:;Coord.-Struct End
;-----------------------------------------------------------------------
```

Const-Names of Func-Parameters

The following listing shows all names that can be used as param. in the param field of the c-functions. Which names are when valid can be find in the function declaration of each function.

```
/* STAT-Register Bit-Names
// ----------------------
*/
#define XS_ALL        -1
#define XS_RCXY        2
#define XS_TWIDTH      4
#define XS_TMODE       8
#define XS_HIRES      16
#define XS_PxMODSC    32
#define XS_PxMODIN    64
#define XS_TxTFNT    128

/* Draw modes
// ----------
*/
#define XM_HiRes 0
#define XM_LoRes 1
#define XM_SET   2
#define XM_CLR   3
#define XM_INV   4

/* Line Graphic
// ---------------------------------
// Circle, Ellipse, Box, Triangle
```

```
// Bit-Flags for Sector switching
// Beware: also defined in 'xgraph.asm' ! Defiintion
//         precedence is here, 'xgraph.asm' follows !
*/
#define XO_NULL   0  /* Plot nothing (for testing etc.) */
#define XO_ALL   -1  /* Plot all Octants             */
#define XO_NNW    1  /* Plot North-North-West Octant */
#define XO_NNE    2  /* Plot North-North-East Octant */
#define XO_WWN    4  /* Plot West-West-North  Octant */
#define XO_EEN    8  /* Plot East-East-North  Octant */
#define XO_SSW   16  /* Plot South-South-West Octant */
#define XO_SSE   32  /* Plot South-South-East Octant */
#define XO_WWS   64  /* Plot West-West-South  Octant */
#define XO_EES  128  /* Plot East-East-South  Octant */

/* Def's of Octant-Numbers for searching the FnEllipse Data-Structure */
#define XO_OctLnNum 0  /* Plotted Oct. + Linestyle-Number */
#define XO_OctStXCo 1  /* Oct. Start X_Coord.     */
#define XO_OctStYCo 2  /* Oct. Start Y-Coord.     */
#define XO_OctEnXCo 3  /* Oct. End X-Coord.       */
#define XO_OctEnYCo 4  /* Oct. End Y-Coord.       */

/*Definition of Hemisphere's as addition of single Octants */
#define XO_NOH  (XO_NNW + XO_NNE + XO_WWN + XO_EEN) /* Northern Hem. */
#define XO_SOH  (XO_SSW + XO_SSE + XO_WWS + XO_EES) /* Southern Hem. */
#define XO_WEH  (XO_NNW + XO_SSW + XO_WWS + XO_WWN) /* Western  Hem. */
#define XO_EAH  (XO_NNE + XO_EEN + XO_EES + XO_SSE) /* Eastern  Hem. */

/*Definition of Quadrant's as addition of single Octants */
#define XO_NWQ  (XO_NNW + XO_WWN) /* Plot North-West Quadr. */
#define XO_NEQ  (XO_NNE + XO_EEN) /* Plot North-East Quadr. */
#define XO_SWQ  (XO_SSW + XO_WWS) /* Plot South-West Quadr. */
#define XO_SEQ  (XO_SSE + XO_EES) /* Plot South-East Quadr. */

/* Line Style PatNumbers
// ----------------------
*/
#define XP_Pat0   0  /* '****************' */
#define XP_Pat1   1  /* '-*-*-*-*-*-*-*-' */
#define XP_Pat2   2  /* '-**--**--**--**-' */
#define XP_Pat3   3  /* '-**----**---**--' */
#define XP_Pat4   4  /* '-**------**-----' */
#define XP_Pat5   5  /* '-******--**--**-' */
#define XP_Pat6   6  /* '-******--******-' */
#define XP_Pat7   7  /* '-**-------------' */
```

```
/* Text attributes
// ---------------
*/
#define XA_RC      0  /* set RC-coord. mode     */
#define XA_XY      1  /* set XY-coord. mode     */
#define XA_DW      2  /* Prt. Double-Width Text */
#define XA_SW      3  /* Prt. Single-Width Text */
#define XA_TI      4  /* Prt. text invers       */
#define XA_TN      5  /* Prt. text noninvers    */

/* Other
// -----
*/
#define X_ON           0       /* Set to "ON"  */
#define X_OFF          -1      /* Set to "OFF" */
#define X_BLACK        0       /* Set to "Black" */
#define X_WHITE        -1      /* Set to "White" */
```

The I/O-Ports of the Graphic-Screen

The used port-range is $92...$9F. There is nothing much to explain, simply read the comments in the table below.

Data-Width:	Port-Addr.:	Comment:
1 Byte	$92	Graphic BYTE write to RAM by address
1 Byte	$93	Graphic BYTE read from RAM by address
1 Byte	$94	Switch Graphic-Screen OFF
1 Byte	$95	Switch Graphic-Screen ON
1 Byte	$96	Set Graphic-Address LOW-Byte
1 Byte	$97	Set Graphic-Address HIGH-Byte
1 Byte	$98	__reserved for future use__
1 Byte	$99	__reserved for future use__
1 Byte	$9A	Cursor flashing OFF
1 Byte	$9B	Cursor flashing ON
1 Byte	$9C	Set Char-ROM Address LOW-Byte
1 Byte	$9D	Set Char-ROM Address HIGH-Byte
1 Byte	$9E	Write Byte to Char-ROM
1 Byte	$9F	Read Byte from Char-ROM

Specialities worth to know

When programming new linegraphic functions, it might be helpfull to have access to some normaly not needed Low-Level internals. I‘ve learned that when programming the circle-algorithm on C-

Level. I present here only the function-list, more detailed explanations w/ examples can be found in the function-reference, section.“File xsys.h“.

Special function list

- CalcRC()
- CalcXY()
- GetStat()
- GetPixMask()
- ReLoadPat()
- RotatePat()
- ScrPortWr()
- ScrPortRd()
- SetPatRot()

These functions are collected in the file „xsys.h".

The xgraph internal tb_bench

Originally ment for testing xgraph in a stand alone mode, this part may become handy when writing asm-programms that need graphic but shouldn‘t run as RSX. I copied this tb_bench version from one of my intermidiate archives. It‘s a little bit lengthy but shows in a nice way how easy the xgraph functions can be used in assembler. All values in par1..5 stay unaltered when a function is called. This minimizes asignments of new values from function call to function call. Only one thing regarding the return value of some functions should be kept in mind: some c-functions do slightly rework the return value, this is not the case when using the function on assembler level ! Which function can be used is listed in the func.-table a the beginning of xgraph.asm (see line 400…).

tb_bench example

```
;-------------------------------------------------------
;                       tb_bench
;-------------------------------------------------------
; =======================================================
; Test Bench for Subroutines in xgraph. Parameters and
; subroutine calls are inserted by hand as needed. 'STAT'
; is GENERELLY modified by the corresponding Bit-Flip
; routines ! ONLY 'par1...par5' should be loaded by hand.
; When "Test-Benching" the following lines at the beginning
; have to be commented out/uncommented:
;
; Line (1,5):    comment out when running as '.com'
; Line (2..4,6): uncomment when running as '.com'
;
; (1)   cseg              ;deactivate, when tb_xgraph is used as
;                         ;stand-alone program
```

```
;       ;#########################################################
;       ;# activate these 3 lines when tb_xgraph runs as stand- #
; (2)   ;org  0100h       ;                        alone program #
; (3)   ;jp   tb_xgraph   ;####### Shortcut for debugging #######
; (4)   ;ld   de,par1     ;#### DE to Param.-Array        #######
;       ;#########################################################
;
; ------------------------------------------------
; Param:  par1   par2   par3   par4   par5   par6
; Entry:   x      y     mode   char   addr   STAT
; Exit:   depending on test
; ================================================
tb_xgraph: ;test-bench routine for Lines with Pattern
; ================================================
; Save the SP
        ld      (oldsp),sp      ;Use RSX Stack-Area
        ld      sp,newsp
; ================================================
;Graphic Init.
        call    Initgraph
        call    ResTMODE        ;noninverted text
        call    SetXY           ;set XY-Mode
        call    ResTWIDTH       ;Double width char
        call    SetHiRES        ;Set HiRes for Graphic
        call    SetRotON        ;Rotation = ON
        call    ResQuad         ;All Octants = ON
        call    GClrScr         ;clear graphic screen

; == FnEllipse ==================================
;------------------------------------------------
;Set Quadr./Oct. param.

        ld      hl,S_ALL        ;ALL
        ld      (par3),hl       ;Quadrants
        call    SetQuad         ;Set Octants
;------------------------------------------------
;RBox Param.
        ld      hl,60
        ld      (par4),hl       ;rbwidth
        ld      hl,05
        ld      (par5),hl       ;rbhight
        call    SetRBox
;------------------------------------------------
;Ellipse Param.
        ld      hl,160
        ld      (par1),hl       ;X
        ld      hl,120
        ld      (par2),hl       ;Y
```

```
        ld      hl,60
        ld      (par4),hl       ;width
        ld      hl,20
        ld      (par5),hl       ;hight

        ld      hl,2
        ld      (par3),hl       ;Linestyle
        call    SetLnSty

        call    SetLoRES        ;Set Resolution for Graphic
        call    FnElipse
; == FnBox ========================================
; ------------------------------------------------
FnBox Param.

        ld      hl,160-101
        ld      (par1),hl       ;X
        ld      hl,120-101
        ld      (par2),hl       ;Y
        ld      hl,065h*2+1
        ld      (par4),hl       ;width
        ld      hl,065h*2+1
        ld      (par5),hl       ;hight
        ld      hl,1
        ld      (par3),hl       ;Linestyle

        call    SetLnSty
        call    SetLoRES        ;Set Resolution for Graphic
        call    FnBox
        call    FnLine2
; ================================================
;        ret               ;(5) ;uncomment when called from 'xdemo.com'
                                ;as RSX-Extension
; ======================================================================
Reload old SP
        ld      sp,(oldsp) ;(6) ;Restore the SP
        jp      0x0000     ;(6) ;uncomment when assembled as stand-alone
;                               ;programm
;
;TestStr: defb   "Hello World !",0
;
;======  E N D - O F - T E S T B E N C H - A R E A  =======
```

Function Reference

All functions are collected in the following .h-files:

```
- xbitmap.h      functions for moving bitmaps between memory & screen
- xchrdef.h      functions regarding font- & character definitions
- xdraw.h        functions regarding line-drawing & support functions
- xsys.h         special support functions checking internal RSX-states
- xtext.h        functions for screen handling & char. printing on screen
- xkeyboard.h    minimal keyboard input functions
- xgraph.h       constant definitions, RSX data-structure, BDOS-Call. Read
                 xgraph.txt for more.
```

File xbitmap.h

Function summary:

- `GetBmpRC();`
- `PutBmpRC();`
- `WriteBmpRC();`

Function GetBmpRC()

Reads a raw bitmap from screen, start point is the top left corner at Pc(col,row) and expands down to (width,hight) of the picture. Minimal error checking is done for:

0 >= width < 80 and
0 >= hight < 30

Error-codes are:

```
width    = 0 -> -1; > 79 -> -3
hight    = 0 -> -2; > 29 -> -4
addr     = no checks !
row, col = no checks !
```

The bitmap must begin and end at a character-cell position. The interleaved organisation of the screen memory is converted to a linear bitmap as mentioned above in „How the Graphic is build“. No additional information regarding width,hight,position etc. is stored with the copied bitmap. If this is wanted, it must be done by the application. It is possible to invert the bitmap on the fly, when the mode „`XM_TI`“ is set. For a non-inverted copy the mode „`XM_TN`“ should be set, if not already done. Use function „`SetTxMode()`“ for this.

Function Declaration:

```
GetBmpRC(row, col, width, hight, addr)
int row, col, width, hight;
```

```
BYTE *addr;
{
    x_func = X_GetBmpRC;

    x_par[0] = row;
    x_par[1] = col;
    x_par[2] = width;
    x_par[3] = hight;
    x_par[4] = addr;

    return x_call();
}
```

Example:

In this example `width` and `hight` are preceding the bitmap as 16bit words. For storage size calc.: Take into account that each character needs 8 byte to store ! „`GetBmpRC()`" calc. for itself the bitmap size in bytes from (xwidth,yhight).

```
/* Variable definitions */
    int  xwidth,yhight;
    int  xc, yc;
    int  *bitmapRC;

/* Write screen bitmap to 'xgr_bmp4' */
      xwidth = 16;     yhight = 10;     /* bitmap size */
      xc = 2;    yc = 2;    /* top left edge of bitmap */

       bitmapRC = xgr_bmp3;
       bitmapRC[0] = xwidth; /* Columns: 1..80 */
       bitmapRC[1] = yhight; /* Rows:    1..30 */

    SetTxMode(XA_TI);  /* read bitmap inverted */
      GetBmpRC(xc, yc, xwidth, yhight, xgr_bmp4);
      (more application code follows here… )

/* The bitmap is stored in a assembler declaration */
#asm
xgr_bmp3:
      DEFW  0                  ;xwidth
      DEFW  0                  ;yhight
xgr_bmp4:
      DEFS  (xwidth*8)*yhight  ;storage area for the bitmap
#endasm
```

Function PutBmpRC()

This fuction is the counterpart of „GetBmpRC()". It writes a raw bitmap from memory to the screen. The startpoint on screen is the top left corner at Pc(col,row) and expands down to (width,hight) on screen. Minimal error checking is done for the size param.:

```
0 >= width < 80        and
0 >= hight < 30
```

Error-codes are:

```
width    = 0 -> -1; > 79 -> -3
hight    = 0 -< -2; > 29 -> -4
addr     = no checks !
row, col = no checks !
```

The bitmap must begin and end at a character-cell position. The interleaved organisation of the screen memory is converted to a linear bitmap as mentioned above in „How the Graphic is build". No additional information regarding width,hight,position etc. is stored with the copied bitmap. If this is needed, it must be done by the application. It is possible to invert the bitmap on the fly, when the mode „XM_TI" is set. For a non-inverted copy the mode „XM_TN" should be set before, if not already done. Use function „SetTxMode()" for this.

Function Declaration:

```
PutBmpRC(row, col, width, hight, addr)
int row, col, width, hight;
BYTE *addr;
{
    x_func = X_PutBmpRC;

    x_par[0] = row;
    x_par[1] = col;
    x_par[2] = width;
    x_par[3] = hight;
    x_par[4] = addr;

    return x_call();
}
```

Example:

In this example width and hight are preceding the bitmap as 16bit words. For storage size calc.: Take into account that each character needs 8 byte to store ! „GetBmpRC()" calc. for itself the bitmap size in bytes from (xwidth,yhight).

```
/* Variable definitions */
    int  xwidth,yhight;
    int  xc, yc;
```

```
    int  *bitmapRC;

/* Read bitmap from 'xgr_bmp4' to screen */
      xc = 2;    yc = 2;    /* top left edge of bitmap */

       bitmapRC = xgr_bmp3;
        xwidth = bitmapRC[0]; /* Columns: 1..80 */
        yhight = bitmapRC[1]; /* Rows:    1..30 */

    SetTxMode(XA_TI);  /* read bitmap inverted */
      PutBmpRC(xc, yc, xwidth, yhight, xgr_bmp4);
      (more application code following here… )

/* The bitmap is red from a assembler declaration */
#asm
xgr_bmp3:
   DEFB 020h,000h,00Ah,000h ;xwidth,yhight => image size
xgr_bmp4:
   DEFB 042h,04Dh,07Eh,00Bh,000h,000h,000h,000h
   DEFB 000h,000h,03Eh,000h,000h,000h,028h,000h
   . . .
   DEFB 0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
   DEFB 0FFh,0FFh,0FFh,0FFh,000h,000h,000h,000h
#endasm
```

Function WriteBmpRC()

`WriteBmpRC()` behaves in the same way as `PutBmpRC()`, the important difference is the build-in aspect-ratio correction when ihe image is written to the screen. This correction doubles the width of a pixel and hence the image. This should be taken into account when choosing the origin on screen. `(width,hight)` are valid only for the source size when copying the image from memory. For getting a aspect-ratio corrected image, first write it to screen, then read it back to memory. Use for readback `GetBmpRC()` with same origin but with **`width`** doubled !

Function Declaration:

```
WriteBmpRC(row, col, width, hight, addr)
int row, col, width, hight;
BYTE *addr;
{
    x_func = X_WriteBmpRC;

    x_par[0] = row;
    x_par[1] = col;
    x_par[2] = width;
```

```
    x_par[3] = hight;
    x_par[4] = addr;

    return x_call();
}
```

Example:

See PutBmpRC().

File xchrdef.h

Function summary:

- GetChRC();
- PutChRC();
- RdChfrAddr();
- RdFntROM();
- ResTxFnt();
- SetTxFnt();
- WrChToAddr();
- WrFntROM();

Function GetChRC()

GetChRC() is a Low-Level function that reads the char.-data from a (row,col)-position on screen to ‚ChrBuf‘ only. If used, a companion function like WrChToAddr() is needed to form a complete operation. It abstracts the char. access on screen and is used from other funktions RSX-Internal. The address calculation is done by ‚CalcRC()‘, so only the usual 80x30 chr-positions are possible. No error checking is done on (x,y).

Function Declaration:

```
GetChRC(x, y)
int x, y;
{
    x_func   = X_GetChRC;
    x_par[0] = x;
    x_par[1] = y;

    x_call();
}
```

Example:

```
Int   x, y, chr;
BYTE *addr;

x     = 10;       /* col 10 */
y     = 12;       /* row 12 */
chr   = 0x00      /* use font-table offset = 0 as start-value  */
addr  = ProgEnd; /* use RAM at program end for char-storage */

GetChRC(x++, y); /* read 1 char. from Pc(x,y) on screen to ChrBuf*/
WrChToAddr(chr++, addr);               /* write ChrBuf to buffer */

GetChRC(x++, y); /* read next char. from Pc(x,y) on screen to ChrBuf */
WrChToAddr(chr++, addr);  /* write ChrBuf to next position in buffer */

#asm
ProgEnd:
      DEFS 8*3          ;24 byte = 3 char to buffer
#endasm
```

Function PutChRC()

`PutChRC()` is a Low-Level function that writes the char.-data from `ChrBuf` to `Pc(col,row)`-position on screen only. If used, a companion function like `RdChToAddr()` is needed to form a complete operation. It abstracts the char. access on screen and is used from other funktions RSX-Internal. The address calculation is done by ‚`CalcRC()`‘, so only the usual 80x30 positions are possible. No error checking is done on `(x,y)`.

Function Declaration:

```
PutChRC(x, y)
int x, y;
{
    x_func   = X_PutChRC;
    x_par[0] = x;
    x_par[1] = y;

    x_call();
}
```

Example:

```
Int   x, y, chr;
BYTE *addr;
```

```
x     = 10;       /* col 10 */
y     = 12;       /* row 12 */
chr   = 0x00      /* use font-table offset = chr*8 as start-value  */
addr  = ProgEnd; /* use RAM at program end for char-storage */

RdChfrAddr(chr++, addr);                /* write 1 char to ChrBuf */
PutChRC(x++, y); /* write 1 char. from ChrBuf to Pc(x,y) on screen */

RdChfrAddr(chr++, addr);                /* write next char to ChrBuf */
PutChRC(x++, y); /* write next char. from ChrBuf to Pc(x,y) on screen */

#asm
ProgEnd:
      DEFS 8*3          ;24 byte = 3 char to buffer
#endasm
```

Function RdChfrAddr()

Low-Level function that reads a char. from `addr` to internal `ChrBuf`. Needs acompanion function to form a complete operation, see `PutChRC()`. No error checking on `addr` is done. Uses `chr*8` as offset into char-table. The return-value points to next 8 byte chr-definition after read of a char. This enables consecutive char-read operations, if chr is set to 0x00.

`chr` = ASCII-Code of character, `char` = 8 byte char-cell

Function Declaration:

```
RdChfrAddr(chr, addr)
int chr; BYTE *addr;
{
    x_func   = X_RdChfrAddr;
    x_par[3] = chr;
    x_par[4] = addr;

    return x_call();
}
```

Example:

```
Int   x, y, chr;
BYTE *addr, *nextchar;

x     = 10;       /* col 10 */
y     = 12;       /* row 12 */
chr   = 0x00      /* use font-table offset = 0 as value  */
```

```
addr  = ProgEnd; /* use RAM at program end for char-storage */

nextchar = RdChfrAddr(chr, addr);        /* write 1 char to ChrBuf */
PutChRC(x++, y); /* write 1 char. from ChrBuf to Pc(x,y) on screen */

nextchar = RdChfrAddr(chr, nextchar); /* write next char to ChrBuf */
PutChRC(x++, y); /* write char. from ChrBuf to Pc(x,y) on screen */

#asm
ProgEnd:
      DEFS 8*3          ;24 byte = 3 char to buffer
#endasm
```

Function WrChToAddr()

Low-Level function that writes a `char` from internal `ChrBuf` to `addr`. Needs a companion function for complete operation, see `GetChRC()`. No error checking on `addr` is done. Uses `chr`*8 as offset into char-table. The return-value points to next 8 byte char-definition after writing all data. This enables consecutive char-write operations, if `chr` is set to 0x00.

Function Declaration:

```
WrChToAddr(chr, addr)
int chr; BYTE *addr;
{
    x_func   = X_WrChToAddr;
    x_par[3] = chr;
    x_par[4] = addr;

    return x_call();
}
```

Example:

```
See GetChRC()
```

Function RdFntROM()

Read `n` character from Font-ROM, starting at position `(0x0000 + (chr * 8))` and store data at `(addr+(chr*8))` in memory. If a external Font-ROM with `SetTxFnt()` is defined, char‘s are red from there, using the font base-address defined with `SetTxFnt()`. The CALLer is responsible for garanteeing that `n` does not go beyond the Font_ROM end !

Function Declaration:

```
RdFntROM(n, chr, addr)
int n, chr; BYTE *addr;
{
    x_func   = X_RdfrFnROM;
    x_par[2] = n;
    x_par[3] = chr;
    x_par[4] = addr;

    x_call();
}
```

Example:

```
Int   n, chr;

n = 255;   /* Read complete Font-ROM */
chr = 0x00;/* We start from the beginning */

SetTxFnt(ExtFontROM); /* Define ext. Font-ROM, if not needed, omit ! */
RdFntROM(n, chr, FontBuffer);
ResTxFnt();           /* Set Font-ROM back to internal */

#asm
ExtFontROM:      ;External Font-ROM data
   DEFB 000h,000h,000h,000h,000h,000h,000h,000h
   DEFB 07Eh,081h,0A5h,081h,0BDh,099h,081h,07Eh
   DEFB 07Eh,0FFh,0DBh,0FFh,0C3h,0E7h,0FFh,07Eh
   DEFB 06Ch,0FEh,0FEh,0FEh,07Ch,038h,010h,000h
   [...]
#endasm

#asm
FontBuffer:
   DEFS 2048     ;Storage area for Font-Data
#endasm
```

Function WrFntROM()

Write n character to Font-ROM, starting at position `(0x0000 + (chr * 8))` and read data from `(addr+(chr*8))` in memory. If a external Font-ROM is defined with `SetTxFnt()`, char‘s are written to this ROM, using the font base-address defined with `SetTxFnt()`. The CALLer is responsible for garanteeing that n does not go beyond the Font_ROM end !

Function Declaration:

```
WrFntROM(n, chr, addr)
int n, chr; BYTE *addr;
{
    x_func   = X_WrToFnROM;
    x_par[2] = n;
    x_par[3] = chr;
    x_par[4] = addr;

    x_call();
}
```

Example:

```
Int   chr;

n = 255;   /* Read complete Font-ROM */
chr = 0x00;/* We start from the beginning */

SetTxFnt(ExtFontROM); /* Define ext. Font-ROM, if not needed, omit ! */
WrFntROM(n, chr, FontBuffer); /* Write  FontBuffer to ExtFontROM */
ResTxFnt();           /* Set Font-ROM back to internal */

#asm
ExtFontROM:      ;External Font-ROM
   DEFS 2048
#endasm

#asm
FontBuffer:
   DEFB 000h,000h,000h,000h,000h,000h,000h,000h
   DEFB 07Eh,081h,0A5h,081h,0BDh,099h,081h,07Eh
   DEFB 07Eh,0FFh,0DBh,0FFh,0C3h,0E7h,0FFh,07Eh
   DEFB 06Ch,0FEh,0FEh,0FEh,07Ch,038h,010h,000h
   […]
#endasm
```

Function SetTxFnt()

Redirects the Font-ROM base address from internal to a external (=> in memory). This is valid until `ResTxFnt()` is called. The ASCII-Screen takes its Font-Data always from the internal Font-ROM. This is defined in hardware and cannot be changed. If something like that is wanted, the internal Font-ROM has to be rewritten with new Font-Data. If this rewrite ends up with unreadable text on ASCII-Screen → good luck with your new Klingon Font-Set ! The redirection is valid until

ResTxFnt() is called. If for any reason the addr is 0x0000, the redirection will silently be rejected ! In that case the internal Font-ROM remains in use.

Function Declaration:

```
SetTxFnt(addr)
int addr;
{
    x_func   = X_SetTxFnt;
    x_par[4] = addr;

    x_call();
}
```

Example:

See RdFntROM() or WrFntROM().

Function ResTxFnt()

Resets the Font-ROM redirection from external to internal. From now on all graphic Print-Instructions will use the internal Font-ROM for there output again.

Function Declaration:

```
ResTxFnt()
{
    x_func   = X_ResTxFnt;

    x_call();
}
```

Example:

See RdFntROM() or WrFntROM().

File xdraw.h

Function summary:

- InitGraphic()
- GScreen()

- GclrScr()
- SetLnStyle()
- SetQuad()
- CopyStruct()
- SearchStruct()
- GetPltOct()
- SetRBox()
- SetPixMode()
- PlotPixel()
- GetPixel()
- FnLine()
- FnLineWH()
- FnBox()
- FnTriangle()
- FnCircle()
- FnEllipse()
- FnRBCircle()
- FnRBEllipse()

Function InitGraphic()

Initializes the xgraph-system to the following state:

- clear the parameter-array par1...5 to zero
- set LineStyle to LNEPAT0
- set Pattern-Rotation to X_ON
- set the LinePattern array to their definition values
- set graphic screen to X_ON
- set ASCII-Curor flashing to X_ON
- set pixel-coord. to 320x240px
- set text-coord. to XA_RC for text output
- set plot pixel mode to XM_SET
- set text mode to XA_TN output
- set text width to XA_SW (same width as ASCII-screen)
- set Font-ROM for graphic text to internal
- set octant plotting to XO_ALL
- set RBox mode to X_OFF (auto-OFF at end of FnEllipse/FnCircle)
- set aspect-ratio correction to X_OFF (WriteBmpRC enables/disables this automatically)

Function Declaration:

```
InitGraphic()
{
    x_func   = X_Initgraph;
```

```
    x_call();
}
```

Example:

```
InitGraphic();
[...]
```

Function Gscreen()

Switch graphic screen ON or OFF

Possible Parameters:

- X_ON
- X_OFF

Function Declaration:

```
GScreen(mode)
int mode;
{
    x_func   = X_GRON;
    x_par[2] = mode;

    x_call();
}
```

Function GClrScr()

Clear the graphic screen according to function parameter to X_BLACK or to X_WHITE background.

Possible Parameters:

- X_BLACK
- X_WHITE

Function Declaration:

```
GClrScr(byte)
int byte;
{
    x_func   = X_GClrScr;
```

```
    x_par[3] = byte;

    x_call();
}
```

Example:

```
GClrScr(X_BLACK);
```

Function SetLnStyle()

Sets the pattern for all line graphic functions (`FnLine()`, `FnLineWH()`,`FnEllipse/FnCircle` and the `Rbox` variants). The pattern is used only in `SET`-mode. `CLR` & `INV` doesn‘t work with this. They do not „generate“ pixel, therefore the pattern can‘t be applied. The pattern stays active until set to `‚XP_Pat0‘` or its redefinition. Text mode is not effected.

Possible Parameters:

```
- XP_Pat0 := '****************' = 'OFF'
- XP_Pat1 := '-*-*-*-*-*-*-*-*'
- XP_Pat2 := '-**--**--**--**-'
- XP_Pat3 := '-**----**----**-'
- XP_Pat4 := '-**------**-----'
- XP_Pat5 := '-******--**--**-'
- XP_Pat6 := '-******--******-'
- XP_Pat7 := '-**------------'
```

Function Declaration:

```
SetLnStyle(PatNo)
int PatNo;
{
    x_func   = X_SetLnSty;
    x_par[2] = PatNo;

    x_call();
}
```

Example:

```
  SetLnStyle(XP_Pat5);       /* '-******--**--**-' */
  FnTriangle(x, y, w, h);
  SetLnStyle(XP_Pat0);       /* '****************' = OFF */
```

Function SetQuad()

Sets a mask that defines for `FnEllipse/FnCircle` which Octants/Quadrants/Hemispheres should be plotted. Will be overwritten when `SetRBox()` prior to the call of FnEllipse is used, because `SetRBox()` has its own octant-mask parameter which acts in the same way as `SetQuad()` do.! Each octant is defined by one bit within the mask. Quadrants/Hemispheres are a combination of octants, as can be seen below. Ellipse & Circles are plotted „Head-Down“, which means „SOUTH“ is top-up, „NORTH“ is top-down. WEST and EAST are on their expected sides ! Octants are plotted with end-coord. „Back-to-Back“. When selecting one of these end-points, they own the same position on screen ! See „xdemo3“. The octant definition is auto-resetted at the end of `FnEllipse()` and aslias functions to „`XO_ALL`".

Possible Parameters:

For Octants:

```
- XO_NULL /* Plot nothing (for testing etc.) */
- XO_ALL  /* Plot all Quadrants              */
- XO_NNW  /* Plot North-North-West Octant */
- XO_NNE  /* Plot North-North-East Octant */
- XO_WWN  /* Plot West-West-North  Octant */
- XO_EEN  /* Plot East-East-North  Octant */
- XO_SSW  /* Plot South-South-West Octant */
- XO_SSE  /* Plot South-South-East Octant */
- XO_WWS  /* Plot West-West-South  Octant */
- XO_EES  /* Plot East-East-South  Octant */
```

For Quadrants:

```
/*Definition of Quadrant's as addition of single Octants */
- XO_NWQ = (XO_NNW + XO_WWN) /* Plot North-West Quadr. */
- XO_NEQ = (XO_NNE + XO_EEN) /* Plot North-East Quadr. */
- XO_SWQ = (XO_SSW + XO_WWS) /* Plot South-West Quadr. */
- XO_SEQ = (XO_SSE + XO_EES) /* Plot South-East Quadr. */
```

For Hemispheres:

```
/*Definition of Hemisphere's as addition of single Octants */
- XO_NOH = (XO_NNW + XO_NNE + XO_WWN + XO_EEN) /* Northern Hem. */
- XO_SOH = (XO_SSW + XO_SSE + XO_WWS + XO_EES) /* Southern Hem. */
- XO_WEH = (XO_NNW + XO_SSW + XO_WWS + XO_WWN) /* Western  Hem. */
- XO_EAH = (XO_NNE + XO_EEN + XO_EES + XO_SSE) /* Eastern  Hem. */
```

Function Declaration:

```
SetQuad(mask)
int mask;
{
```

```
    x_func   = X_SetQuad;
    x_par[2] = mask;

    x_call();
}
```

Example:

```
SetQuad(XO_NNW);            /* Plot North-North-West Octant */
FnCircle( 180, 120, 40);

SetQuad(XO_NWQ);            /* Plot North-West Quadrant */
FnEllipse(230, 200, 23, 18);

SetQuad(XO_NOH);            /* Plot Northern Hemisphere */
FnEllipse(230, 200, 23, 18);
```

Function CopyStruct()

FnEllipse collects coord.- and other data in a special structure during its operation. This structure is return by a pointer. `CopyStruct()` can copy this structure to `addr` when called. This way the collected data can be preserved for later use, because the structure is only valid until `FnEllipse()` is called again. If `CopyStruct()` is called, preserve 100 byte for that or better use ‚`SearchStruct(XO_ALL,0,0)`‘ to get the size of the structure as return value. For a more detailed description of the structure itself, see „`The Data-Struct. of FnEllipse`". ‚`CopyStruct(addr)`‘ is a short form of ‚`SearchStruct(XO_NULL,0,addr)`‘.

Function Declaration:

```
CopyStruct(addr)
int *addr;
{
    x_func   = X_GetStruct;
    x_par[2] = XO_NULL;     /* ‚XO_NULL‘ selects ‚COPY‘-Operation */
    x_par[3] = 0;           /* <= ZERO, needed RSX internaly */
    x_par[4] = addr;        /* where the data goes to */

    x_call();
}
```

Example:

```
FnEllipse( x, y, r1, r2);
```

```
CopyStruct(OctCoord); /* Should be called directly after FnEllipse() */

FnLine(SearchStruct(XO_SSW, XO_OctStXCo, OctCoord),
       SearchStruct(XO_SSW, XO_OctStYCo, OctCoord),
       SearchStruct(XO_NNE, XO_OctStXCo, OctCoord),
       SearchStruct(XO_NNE, XO_OctStYCo, OctCoord));

#asm
OctCoord:
   DEFS  100
#endasm
```

Function SearchStruct()

Searches the structure for a specific octant or other data contained in the structure. If the searched octant is found the requested coord.-data is extracted and returned. If the octant could not be found ‚0x0000' is returned.

Special function of 1st parameter (set 2nd & 3rd param. to ‚0'):

```
XO_NULL     = Copy struct. To ‚addr'
XO_ALL      = Return size of struct. in ‚bytes'
```

For extracting Start/End-Point coord. Use these octant names as 1st parameter:

```
XO_NNW, XO_NNE, XO_WWN, XO_EEN,
XO_SSW, XO_SSE, XO_WWS, XO_EES
```

Quadrant and Hemisphere names are not possible, because they are bit combinations of octant names. Each octant must be searched on its own.

Possible 2nd param. value, coord. names:

```
XO_OctLnNum,                /* Oct. Linestyle-Number */
XO_OctStXCo, XO_OctStYCo    /* X-Start/Y-Start coord. of oct. */
XO_OctEnXCo, XO_OctEnYCo    /* X-End/Y-End coord. of oct. */
```

Buffer-Address as 3rd param.:

```
addr       /* Pointer to buffer with size of Structure */
```

Function Declaration:

```
SearchStruct(oct, param, addr)
int oct, param, *addr;
{
```

```
    x_func   = X_GetStruct;
    x_par[2] = oct;
    x_par[3] = param;
    x_par[4] = addr;

    if(oct == XO_ALL)
       return x_call() >> 8;
    else
       return x_call();
}
```

Example:

Plot a rectangular box with rounded edges. RBox-plotting is done in two steps. First the circle with the quadrant-offset is plotted. Next is to find the quadrant start-coord. Last step is drawing the connecting lines between the quadrants. The octant end-coord. are not needed, because they are ending back-to-back in the middle of the quadrant.

```
    x=160;  y=120;
    SetLnStyle(0);
    SetPixMode(XM_LoRes);
    SetRBox(XO_ALL,13,5);   /* see sketch in „SetRBox()" ! */
      OctCoord = FnCircle( x, y>>1, 10);

    SetLnStyle(0);
      FnLine(SearchStruct(XO_SSW, XO_OctStXCo, OctCoord),
             SearchStruct(XO_SSW, XO_OctStYCo, OctCoord),
             SearchStruct(XO_SSE, XO_OctStXCo, OctCoord),
             SearchStruct(XO_SSE, XO_OctStYCo, OctCoord));

      FnLine(SearchStruct(XO_NNE, XO_OctStXCo, OctCoord),
             SearchStruct(XO_NNE, XO_OctStYCo, OctCoord),
             SearchStruct(XO_NNW, XO_OctStXCo, OctCoord),
             SearchStruct(XO_NNW, XO_OctStYCo, OctCoord));

      FnLine(SearchStruct(XO_EES, XO_OctStXCo, OctCoord),
             SearchStruct(XO_EES, XO_OctStYCo, OctCoord),
             SearchStruct(XO_EEN, XO_OctStXCo, OctCoord),
             SearchStruct(XO_EEN, XO_OctStYCo, OctCoord));

      FnLine(SearchStruct(XO_WWS, XO_OctStXCo, OctCoord),
             SearchStruct(XO_WWS, XO_OctStYCo, OctCoord),
             SearchStruct(XO_WWN, XO_OctStXCo, OctCoord),
             SearchStruct(XO_WWN, XO_OctStYCo, OctCoord));
```

Function GetPltOct()

`GetPltOct()` extracts the plotted octants mask from the struct. that `FnEllipse()` returns or from a buffered struct in memory. This function is the short form of `SearchStruct(XO_ALL,0,addr)`.

The returned 16-bit value contains the oct.-mask in the LByte.

Function Declaration:

```
GetPltOct(addr)
int *addr;      /* pointer to buffer with struct or from FnEllipse() */
{
    x_func   = X_GetStruct;
    x_par[2] = XO_ALL;
    x_par[3] = 0;
    x_par[4] = addr;   /* if addr = 0 then use RSX-internal struct. */

    return x_call() & 0x00FF;
}
```

Example:

```
Int OctCoord, OctMask;
Int x, y, width, hight;
int buffer[SearchStruct(XO_ALL,0,0)]; /* define the struct-buffer */

x = 160; y = 120;
width = 30; hight = 10;
SetQuad(XO_NOH);                  /* set Northern-Hemisphere to plot */
/* Use FnEllipse-Struct */
OctCoord = FnEllipse(x, y, width, hight);
OctMask  = OctCoord[0] & 0x00FF;  /* the short way */

/* use struct in external buffer */
FnEllipse(x, y, width, hight);
CopyStruct(*buffer[]);               /* copy struct to buffer */
OctMask = GetPltOct(*buffer[]);      /* a bit more complicated */
```

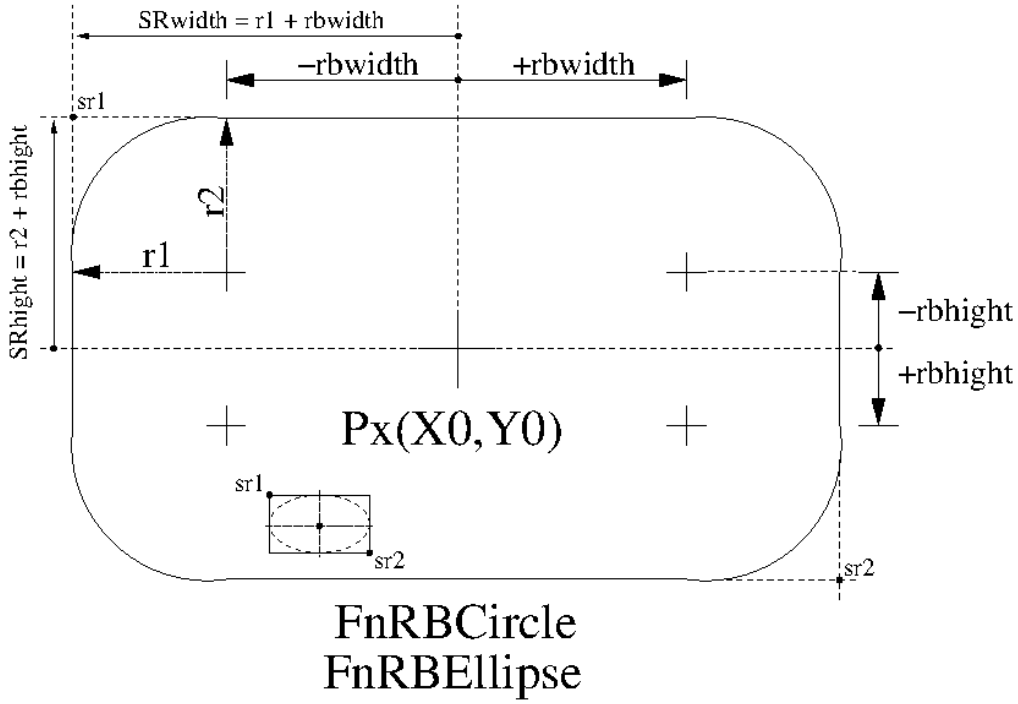
Function SetRBox()

Set the width and hight of the straight section between the rounded corners for FnCircle/FnEllipse. The overal dimesnions of the total width & hight of the Rbox is then:

```
Vertical-hight   := 2 * rbhight + 2 * r2
```

```
Horizontal-width := 2 * rbwidth + 2 * r1
```

If the corners are circle quadrant then set r1 = r2. See also the following sketch:



The `mask` for Octant/Quadrant/Hemisphere definition has the same purpose as in the SetQuad()-Function, so a extra SetQuad() isn‘t needed. The RBox-Setting is only valid for the following FnEllipse() (and alias-functions) call. The oct.-mask and rb* values are then reset to `mask = XO_ALL & rb* = XO_NULL`

Possible Parameters:

For `mask` see `SetQuad()`-Function.

`width` & `hight` are in pixels, range is `0..319px` & `0..239px`, but the overall value of `Vertical-hight` should not exceed `239px`, same for `Horizontal-width` that should not exceed `319px`. There is no limit checking. No negativ values for `rbwidth` & `rbhight` are allowed !.

Function Declaration:

```
SetRBox(mask, rbwidth, rbhight)
int mask, rbwidth, rbhight;
{
    x_func   = X_SetRBox;
    x_par[2] = mask;
    x_par[3] = rbwidth;
    x_par[4] = rbhight;

    x_call();
}
```

Example:

```
SetLnStyle(0);
SetRBox(XO_ALL,rbwidth,rbhight);
FnEllipse(x0, y0, r1, r2);
```

Function SetPixMode()

This function defines whether pixels are SET, CLRed or INVerted when the Fn...-Functions are called. If a Linepattern is set (with SetLnSty()), the pattern will only be active in SET-Mode. CLR- & INV-Mode are no pixel generating modes. To invert a line with a pattern first plot it in SET-Mode and then replot it in INV-Mode (the LineStyle-Setting will be ignored). The SetPixMode() setting is valid until it is redefined.

This function can be used to switch the resolution too, see below.

Possible Parameters:

- XM_HiRes
- XM_LoRes
- XM_SET
- XM_CLR
- XM_INV

Function Declaration:

```
SetPixMode(mode)
int mode;
{
    x_func   = X_SetPxMode;
    x_par[2] = mode;

    x_call();
}
```

Example:

```
SetPixMode(XM_HiRes);
SetPixMode(XM_SET);
SetLnStyle(0);
FnLine(10, 20, 309, 20);
```

Function PlotPixel()

Plots a pixel at Px(X,Y), according to the settings of `SetPixMode() & SetLnStyle()`. The range is for `X = 0..319px` and for `Y = 0..239px`. No limit checks are done.

Function Declaration:

```
PlotPixel(x, y)
int x, y;
{
    x_func   = X_PltPix;
    x_par[0] = x;
    x_par[1] = y;

    x_call();
}
```

Example:

```
SetPixMode(XM_LoRes);
SetPixMode(XM_SET);
SetLnStyle(0);
[...]
while (b2*x <= a2*y) {
      PlotPixel(xpos=xc + x, ypos=yc + y);
      PlotPixel(xpos=xc - x, ypos=yc + y);
      PlotPixel(xpos=xc + x, ypos=yc - y);
      PlotPixel(xpos=xc - x, ypos=yc - y);
      RotatePat();
      if(sigma >= 0) {
         sigma = sigma + fa2 * (1 - y);
         y=y-1;
      }
      sigma = sigma + b2 * ((4 * x) + 6);
      x=x+1;
   }
[...]
```

Function GetPixel()

This function is meant for requesting the pixel-status (`SET` or `CLR`) at `Px(X;Y)`. If `SET` the return value is `TRUE` otherwise `FALSE`. The value ranges for X & Y is as for `PlotPixel()`. All coord. values have to stay within `x = 0..319px, y = 0..239px`.

Function Declaration:

```
GetPixel(x, y)
```

```
int x, y;
{
    x_func   = X_GetPix;
    x_par[0] = x;
    x_par[1] = y;

    return x_call();
}
```

Example:

```
If(GetPixel(x, y))
   printf(„Pixel is set\n\r");
else
   printf(„Pixel is not set\n\r");
```

Function FnLine()

Draw a line in the current draw mode set by `SetPixMode() & SetLnStyle()` from `Px(X0,Y0)` to `Px(X1,Y1)`. All coord. values have to stay within `x = 0..319px, y = 0..239px`.

Function Declaration:

```
FnLine(x0, y0, x1, y1)
int x0, y0, x1, y1;
{
    x_func   = X_FnLine;
    x_par[0] = x0;
    x_par[1] = y0;
    x_par[3] = x1;
    x_par[4] = y1;

    x_call();
}
```

Example:

```
[...]
SetPixMode(XM_LoRes);
SetLnStyle(XP_Pat1);
FnLine(x0, y0, x1, y1);
[...]
```

Function FnLineWH()

Draw a line in the current draw mode set by `SetPixMode()` & `SetLnStyle()` from `Px(X0,Y0)` to `Px(X0 + width,Y0 + hight)`. `width & hight` may be negativ. All coord. values have to stay within `x = 0..319px, y = 0..239px`.

Function Declaration:

```
FnLineWH(x0, y0, width, hight)
int x0, y0, width, hight;
{
    x_func   = X_FnLine;
    x_par[0] = x0;
    x_par[1] = y0;
    x_par[3] = width;
    x_par[4] = hight;

    x_call();
}
```

Example:

```
/* Plot a triangle */
FnLineWH(x,y,(width >> 1)+1,hight);
FnLineWH(x+width,y,-(width >> 1),hight);
FnLine(x,y,x+width,y);
[...]
```

Function FnBox()

Draw a Box in the current draw mode from `Px(x0,y0)` to `Px(x0 + width, y0 + hight)`. All coord. values have to stay within `x = 0..319px, y = 0..239px`. `width & hight` may be negativ.

Function Declaration:

```
FnBox(x, y, width, hight)
int x, y, width, hight;
{
    x_func   = X_FnBox;
    x_par[0] = x;
    x_par[1] = y;
    x_par[3] = width;
    x_par[4] = hight;
```

```
    x_call();
}
```

Example:

```
SetPixMode(XM_LoRes);
SetLnStyle(XP_Pat3);
FnBox( x, y, width, hight);
```

Function FnTriangle()

Draw a Triangle in the current draw mode from `Px(x0,y0)` to `Px(x0+width, y0) to Px(x0+width/2, y0+hight)`. All coord. values have to stay within `x = 0..319px, y = 0..239px. width & hight` may be negativ.

Function Declaration:

```
FnTriangle(x, y, width, hight)
int x, y, width, hight;
{
    FnLineWH(x,y,(width >> 1)+1,hight);
    FnLineWH(x+width,y,-(width >> 1),hight);
    FnLine(x,y,x+width,y);
}
```

Example:

```
[...]
SetLnStyle(XP_Pat0);
SetPixMode(XM_LoRes);
FnTriangle( x, y, width, hight);
[...]
```

Function FnCircle()

Draw a circle in the current draw mode at Px(x0,y0) with radius `r1`. Return-value in `hl = Pointer to Oct-Coord. Array`. All coord. values have to stay within `x = 0..319px, y =`

0..239px. r1 has be positiv. The Quad.-Mask will be reset to XO_ALL at the end of FnEllipse().

Function Declaration:

```
FnCircle(x0, y0, r1)
int x0, y0, r1;
{
    return FnEllipse(x0, y0, r1, r1);
}
```

Example:

```
[...]
SetLnStyle(XP_Pat0);

SetQuad(XO_NWQ);
FnCircle( x, y, r1);   /* plot NW-Quadr. */
FnCircle( x1, y1, r2); /* will plot a full circle */
[...]
```

Function FnEllipse()

Draw a ellipse in the current draw mode at Px(x0,y0) with radius r1 & r2. r1 is the horizontal radius, r2 the vertical one. All coord. values have to stay within x = 0..319px, y = 0..239px. r1 & r2 has be positiv. Return-value in hl = Pointer to Oct-Coord. Array. The Quad.-Mask will be reset to XO_ALL at the end of FnCircle(). In the byte at address (O_BXEL+1) flag is stored whether HiRes or LoRes mode was used for plotting.

Function Declaration:

```
FnEllipse(x, y, width, hight)
int x, y, width, hight;
{
    x_func   = X_FnEllipse;
    x_par[0] = x;
    x_par[1] = y;
    x_par[3] = width;
    x_par[4] = hight;

    return x_call();
}
```

Example:

```
[...]
SetLnStyle(XP_Pat1);
SetQuad(XO_NEQ);
FnEllipse(x, y, r1, r2);    /* plot NE-Quadr. */
FnEllipse(x1, y1, r3, r4);  /* will plot all quadrants */
[...]
```

Function FnRBCircle()

Draw a box with round edges in the current draw mode at `Px(x0,y0)` with radius `r1`. Return-value is `Pointer_to_Oct-Coord.-Array`. All coord. values have to stay within `x = 0..319px, y = 0..239px`. `r1` has be positiv. The Quad-Mask is set to `XO_ALL` in the function declaration . The `RB-Values` will be reset to `XM_NULL` at the end of `FnRBCircle()`.

Function Declaration

```
FnRBCircle(x0, y0, r1, rbwidth, rbhight)
int x0, y0, r1, rbwidth, rbhight;
{
    int OctCoord;

    SetRBox(XO_ALL,rbwidth,rbhight);
    OctCoord = FnEllipse( x0, y0, r1,r1);

    FnLine(SearchStruct(XO_SSW, XO_OctStXCo, OctCoord),
           SearchStruct(XO_SSW, XO_OctStYCo, OctCoord),
           SearchStruct(XO_SSE, XO_OctStXCo, OctCoord),
           SearchStruct(XO_SSE, XO_OctStYCo, OctCoord));

    FnLine(SearchStruct(XO_NNE, XO_OctStXCo, OctCoord),
           SearchStruct(XO_NNE, XO_OctStYCo, OctCoord),
           SearchStruct(XO_NNW, XO_OctStXCo, OctCoord),
           SearchStruct(XO_NNW, XO_OctStYCo, OctCoord));

    FnLine(SearchStruct(XO_EES, XO_OctStXCo, OctCoord),
           SearchStruct(XO_EES, XO_OctStYCo, OctCoord),
           SearchStruct(XO_EEN, XO_OctStXCo, OctCoord),
           SearchStruct(XO_EEN, XO_OctStYCo, OctCoord));

    FnLine(SearchStruct(XO_WWS, XO_OctStXCo, OctCoord),
           SearchStruct(XO_WWS, XO_OctStYCo, OctCoord),
           SearchStruct(XO_WWN, XO_OctStXCo, OctCoord),
           SearchStruct(XO_WWN, XO_OctStYCo, OctCoord));
```

```
    return OctCoord;
}
```

Example:

```
SetPixMode(XM_SET);
SetPixMode(XM_LoRes);
SetLnStyle(XP_Pat0);
FnRBCircle( x, y, r1, rbwidth, rbhight);
```

Function FnRBEllipse()

Draw a RBox with ellipsoid edges in the current draw mode at Px(x0,y0) with radius r1 & r2. Return-value in hl = Pointer to Oct-Coord. Array. All coord. values have to stay within x = 0..319px, y = 0..239px. r1 & r2 have be positiv. The Quad-Mask is function internal set to XO_ALL. The RB-Values will be reset to XM_NULL at the end of FnRBEllipsle().

Function Declaration:

```
FnRBEllipse(x0, y0, r1, r2, rbwidth, rbhight)
int x0, y0, r1, r2, rbwidth, rbhight;
{
    int OctCoord;

    SetRBox(XO_ALL,rbwidth,rbhight);
    OctCoord = FnEllipse( x0, y0, r1,r2);

    FnLine(SearchStruct(XO_SSW, XO_OctStXCo, OctCoord),
           SearchStruct(XO_SSW, XO_OctStYCo, OctCoord),
           SearchStruct(XO_SSE, XO_OctStXCo, OctCoord),
           SearchStruct(XO_SSE, XO_OctStYCo, OctCoord));

    FnLine(SearchStruct(XO_NNE, XO_OctStXCo, OctCoord),
           SearchStruct(XO_NNE, XO_OctStYCo, OctCoord),
           SearchStruct(XO_NNW, XO_OctStXCo, OctCoord),
           SearchStruct(XO_NNW, XO_OctStYCo, OctCoord));

    FnLine(SearchStruct(XO_EES, XO_OctStXCo, OctCoord),
           SearchStruct(XO_EES, XO_OctStYCo, OctCoord),
           SearchStruct(XO_EEN, XO_OctStXCo, OctCoord),
           SearchStruct(XO_EEN, XO_OctStYCo, OctCoord));

    FnLine(SearchStruct(XO_WWS, XO_OctStXCo, OctCoord),
           SearchStruct(XO_WWS, XO_OctStYCo, OctCoord),
           SearchStruct(XO_WWN, XO_OctStXCo, OctCoord),
```

```
            SearchStruct(XO_WWN, XO_OctStYCo, OctCoord));

    return OctCoord;
}
```

Example:

```
SetPixMode(XM_SET);
SetPixMode(XM_LoRes);
SetLnStyle(XP_Pat0);
FnRBEllipse(x,y,r1,r2,rbwidth,rbhight);
```

File xsys.h

Function summary:

- `CalcXY()`
- `CalcRC()`
- `GetStat()`
- `GetPixMask()`
- `ReLoadPat()`
- `RotatePat()`
- `ScrPortWr()`
- `ScrPortRd()`
- `SetPatRot()`

Function `CalcXY()`

`CalcXY()` returns the XY-Coord. `Px(x,y)` of a pixel on screen. `Px(x,y)` has to be positiv and in the range `x = 0..319px, y = 0..239px`. When called on assembler level, hl holds the `addr` of byte with the pixel. The pixel `mask` is in `a`. Depending on the resolution set with SetPixMode(), the mask is for 8 pixel in a byte (Resol. = 640px) or for 4 pixel in a byte (Resol. = 320px). On C-Level only the `addr` in `hl` is returned. The pixel `mask` must be determined separately with `GetPixMask()`. `CalcXY()` isn‘t really needed on C-Level programming. Each discussed function that needs address calculation implicitly calls `CalcXY()`. Its usefullness comes up when programming new things on C-Level and speed is a concern. Doing the address calculation in high level will be much slower.

Function Declaration:

```
CalcXY(x, y)
int x, y;
```

```
{
    x_func   = X_CalcXY;
    x_par[0] = x;
    x_par[1] = y;

    return x_call();
}
```

Example:

```
addr = CalcXY(x, y);
mask = GetPixMask(x, y);
```

Function `CalcRC()`

`CalcRC()` plays in the same category as `CalcXY()`. On return the address of `Pc(col,row)` position is in `hl` (= `return value` in C). A pixel-mask isn‘t needed. Usefullness = see `CalcXY()`. The return value points the first byte of the 8 byte long char-cell on screen. `Pc(col,row)` must be positiv and in the range `col = 0..79, row = 0..29.`

Function Declaration:

```
CalcRC(x, y)
int x, y;
{
    x_func   = X_CalcXY;
    x_par[0] = x;
    x_par[1] = y;

    return x_call();
}
```

Example:

```
addr = CalcRC(x,y);
```

Function `GetStat()`

`GetStat()` return the status of single bits in STAT-Register. Its main purpose is to support situations on C-Level programming where resolution dependend x-coord. correction is needed. For example programming a circle algorithm dealing with `HiRes` & `LoRes` → see example.

Possible Parameters:

```
IF STAT-Bit = 'SET' then 'TRUE' ELSE 'FALSE';
Possible Mask-Param.:
 XS_ALLBITS  =  0 Return all Status-Bits (No TRUE / FALSE !)
 XS_RCXY     =  2   0=RC       or 1=XY mode for text
 XS_TWIDTH   =  4   0=Single   or 1=Double width text
 XS_TMODE    =  8   0=invers   or 1=noninvers text
 XS_HIRES    =  16  0=LoRes    or 1=HiRes for Pixel-Graphic
 XS_PxMODSC  =  32  0=SET      or 1=CLR pixel mode
 XS_PxMODIN  =  64  0=nonINV   or 1=INV (1=INV pixel mode, 'INV'
                                  has precedence over SET/CLR)
 XS_TxTFNT   =  128 0=internal or 1=external text font
                    if 'TxTFNT' is set and 'addr' for ext. text-font
                    is 0x0000, 'TxTFNT' will be reset by RSX-system
 XS_USEPAT   =  256 0=No Line-Pattern    or 1=Use Line-Pattern
 XS_PATROT   =  512 0=Patter-Rot. active or 1=Pattern-Rot. inactive
 XS_RBOX     = 1024 0=No RBox plotting   or 1=RBox plotting requested
                    This Bit is auto-reset to '0' after plotting !
=======================================================================
The Return-Value is: 'NOT SET' = '0' = FALSE | 'SET' = '-1' = TRUE,
except otherwise noted !
=======================================================================
```

Pixel Masks for 320px resolution:

```
==================================
BitMask320:             ; BIT-MASKs FOR 320 PIXEL-Mode
        defb 0C0H       ; BIT(7,6)=0C0H Index(0)
        defb 030H       ; BIT(5,4)=060H Index(1)
        defb 00CH       ; BIT(3,2)=030H Index(2)
        defb 003H       ; BIT(1,0)=018H Index(3)
```

Pixel Masks for 640px resolution:

```
==================================
BitMask640:             ; read with (BitMask320 + 4) + index
        defb 080H       ; BIT(7)=080H   Index(0)
        defb 040H       ; BIT(6)=040H   Index(1)
        defb 020H       ; BIT(5)=020H   Index(2)
        defb 010H       ; BIT(4)=010H   Index(3)
        defb 008H       ; BIT(3)=008H   Index(4)
        defb 004H       ; BIT(2)=004H   Index(5)
        defb 002H       ; BIT(1)=002H   Index(6)
        defb 001H       ; BIT(0)=001H   Index(7)
```

Example:

```
FnCircle(x0, y0, r1)
int x0, y0, r1;
{
    int x, xh, xx, yy;
    int y, yh;
    int decision;

    if (GetStat(XS_HIRES)) {
       x0 = x0 << 1;
    }

    x = r1;
    y = 0;
    decision = 1 - x;

    ReLoadPat();
    RotatePat();
    SetPatRot(X_OFF);
    while(x >= y) {

        if (GetStat(XS_HIRES)) {
           yy = y << 1;
           xx = x << 1;
        }
        else {
           xx = x;
           yy = y;
        }

        xh = x;
        yh = y;

        PlotPixel( xx + x0,  yh + y0);
        PlotPixel( yy + x0,  xh + y0);
        PlotPixel(-xx + x0,  yh + y0);
        PlotPixel(-yy + x0,  xh + y0);
        PlotPixel(-xx + x0, -yh + y0);
        PlotPixel(-yy + x0, -xh + y0);
        PlotPixel( xx + x0, -yh + y0);
        PlotPixel( yy + x0, -xh + y0);
        RotatePat();
        y++;

        if(decision <= 0) {
```

```
            decision += y+y + 1;
        }
        else {
            x--;
            decision += (y+y - x-x) + 1;
        }
    }
  SetPatRot(X_ON);
}
```

Function GetPixMask()

`GetPixMask()` is the companion function for `CalcXY()`. It returns the pixel `mask` corresponding to the `addr` calculated by `CalcXY()`. The retured mask depends on the active resolution setting done with `SetPixMode()`. `Px(x,y)` has to be positive, the coord. values must be in the range `x = 0..319px, y = 0..239px.`

Function Declaration:

```
GetPixMask(x, y)
int x, y;
{
    x_func   = X_GetPxMask;
    x_par[0] = x;
    x_par[1] = y;

    return x_call();
}
```

Example:

```
addr = CalcXY(x, y);
mask = GetPixMask(x, y);
```

Function ReLoadPat()

`ReLoadPat()` is a Low-Level function and reloads the selected Pattern, so it starts from its initial bit-position. For example, if a straight line is plotted with a patter <> `XP_Pat0`, this init is done once at the beginning so each line starts with bit set to ‚1‘ at the starting point, regardless how the pattern has ended in the previous plot command. All line plotting functions do a implicit `ReLoadPat()`. In pricipal this function is only usefull when programing things like the Circle-Algorithm shown at `GetStat()` where pixel are set with `PlotPixel()` and pattern control is explicitly needed.

Function Declaration:

```
ReLoadPat()
{
    x_func   = X_ReLoadPat;

    x_call();
}
```

Example:

See `GetStat()`

Function `RotatePat()`

`RotatePat()` is a Low-Level function as `ReLoadPat()`. If called, it rotates the active pattern one position to the left. If direct pattern controll in a pixel plotting function needed `RotatePat()` might become important. `RotatePat()` has no parameters.

Function Declaration:

```
RotatePat()
{
    x_func   = X_RotatePat;

    x_call();
}
```

Example:

See `GetStat()`

Function `SetPatRot()`

`SetPatRot() is` a Low-Level function as `RotatePat()`. If called it can stop or start pattern rotation. It is only useful in functions, where pattern controll is needed.

Possible Parameters:

```
- X_ON     /* for Pattern Rotation = ON */
- X_OFF    /* for Pattern Rotation = OFF */

ON/OFF doesn't change pattern rotation state.
```

Function Declaration:

```
SetPatRot(stat)
int stat;
{
    x_func   = X_SetPatRot;
    x_par[2] = stat;

    x_call();
}
```

Example:

See `GetStat()`

Function `ScrPortWr()`

`ScrPortWr()` writes a `Byte` to the Graphic Screen using the `addr` from `CalcXY()` or `CalcRC()`. `ScrPortWr()` is a Low-Level function and gives direct access to the byte of the graphic screen. It should be mentioned, that using this function for manipulating the graphic screen gives generally slower result compared to the `FnLine()` function because of the BDOS overhead when calling the function. The reason is the transfer of the ‚`x_par[]`‘ array to the RSX before the function can be executed.

Function Declaration:

```
ScrPortWr(addr, data)
int addr, data;
{
    x_func   = X_ScrPortWr;
    x_par[3] = data;
    x_par[4] = addr;

    x_call();
}
```

Example:

```
SetPixMode(LoRes);
addr = CalcXY(x, y);
mask = GetPixMask(x, y);
data = ScrPortRd(addr) & mask;
ScrPortWr(addr, data);
```

Function ScrPortRd()

ScrPortRdr() reads a Byte from the Graphic Screen using the addr from CalcXY() or CalcRC(). The description of ScrPortWr() is valid for ScrPortRdr() too. The returned value is 16 bit uint but in the range 0..255 only.

Function Declaration:

```
ScrPortRd(addr)
int addr;
{
    x_func   = X_ScrPortRd;
    x_par[4] = addr;

    return x_call();
}
```

Example:

```
SetPixMode(HiRes);
addr = CalcXY(x, y);
mask = GetPixMask(x, y);
data = ScrPortRd(addr) & mask;
ScrPortWr(addr, data);
```

Function RSXName()

Returns a pointer to a NULL terminated string containing the RSX-Name.

Function Declaration:

```
RSXName()
{
    x_func   = X_RSXName;

    return x_call();
}
```

Example:

```
SetTxMode(XA_RC);
PrintStr(1,0,"RSX-Name: ");
PrintStr(11,0,RSXName());
```

Function `RSXVersion()`

Returns the RSX Version-Number als binary value.

Function Declaration:

```
RSXVersion()
{
    x_func   = X_RSXVersion;
    return x_call();
}
```

Example:

```
printf("RSX-Version: %d :",RSXName());
```

File xtext.h

Function summary:

- `AclrScr()`
- `HideCursor()`
- `ShowCursor()`
- `SetIntFnt()`
- `SetExtFnt()`
- `SetTxMode()`
- `PrintChRpt()`
- `PrintChr()`
- `PrintStr()`

Function `AclrScr()`

Clears simply the ASCI-Screen. For this a ESC-Sequence is used, but, this will make it terminal type dependend. If it does not work on a PC with terminal emulation, the ESC-Sequence might be the cause. Then you have to modify it.

Function Declaration:

```
AClrScr()
{
    putch(27); putch('['); putch('H');             /* Home */
    putch(27); putch('['); putch('2'); putch('J'); /* Cls  */
```

```
}
```

Example:

```
AClrScr();
```

Function `HideCursor()`

Disables the cursor flashing. The cursor position isn‘t changed by this. `HideCursor()` is only valid for the Multicomp VGA-Output. If a PC Terminal emulation is used the cursor of the PC-Terminal will be still flashing !

Function Declaration:

```
{
    x_func   = X_ACON;
    x_par[2] = X_OFF;

    x_call();
}
```

Example:

```
AClrScr();
```

Function Show`Cursor()`

Disables the cursor flashing. The cursor position isn‘t changed by this. `ShowCursor()` is only valid for the `Multicomp VGA-Output`. If a PC Terminal emulation is used the cursor of the PC-Terminal isn‘t influenced by this !

Function Declaration:

```
ShowCursor()
{
    x_func   = X_ACON;
    x_par[2] = X_ON;

    x_call();
}
```

Example:

```
ShowCursor();
```

Function `SetIntFnt()`

Sets the font for graphic text to the internal Font-ROM of the ASCII-Screen.

Function Declaration:

```
SetIntFnt()
{
    x_func   = X_ResTxFnt;

    x_call();
}
```

Example:

```
SetIntFnt();
```

Function `SetExtFnt()`

Sets the font for graphic text to the the `addr` of a external Font-ROM. All subsequent graphic text output will use the char-data from this ROM. The text output on the ASCII-Screen isn‘t influenced. A ext.-Font -Address of 0x0000 is silently rejected and no change takes place. If new char-data is written to the ext.-ROM be shure that this a really your ext.-Font-ROM, no checks or something else are done to enshure this !

Function Declaration:

```
SetExtFnt(addr)
int addr;
{
    x_func   = X_SetTxFnt;
    x_par[4] = addr;

    x_call();
}
```

Example:

```
[...]
```

```
SetTxMode(XA_XY);
SetTxFnt(addr_ext_Font);
PrintStr(x,y,"Show some external ext. Font char...");
ResTxFnt();
[...]
```

Function `SetTxMode()`

Define the way text is printed on graphic screen. It‘s possible to define in witch coord. system this should be done (`XY-` or `RC-Coord`.), the text width or normal or invers text. Either definition stay active until it is redefined. If calculating text-length keep in mind that the `320x240px` coord. system counts the char-cell width as `4px` despite that a byte has 8 bit. Text in `XA_DW` mode counts as `8px` in width and spans 2 char-cells per chr – good luck !

Possible Parameters:

```
- XA_RC = set RC-coord. mode
- XA_XY = set XY-coord. mode
- XA_DW = Prt. Double-Width Text
- XA_SW = Prt. Single-Width Text
- XA_TI = Prt. text invers
- XA_TN = Prt. text noninvers
```

Function Declaration:

```
SetTxMode(mode)
int mode;
{
    x_func   = X_SetTxMode;
    x_par[2] = mode;

    x_call();
}
```

Example:

```
SetTxMode(XA_XY);
SetTxMode(XA_TI);
SetTxMode(XA_DW);
PrintStr(x, y, "XY: Hello World !");
```

Function `PrintChRpt()`

Print `chr` `n`-times repeatedly on graphic screen. `n` must be positiv and can be in the range `0..255`. `PrintChRpt()` respects `SetTxMode()` settings. For RC-Mode: `x = 0..79, y = 0..29` or in XY-Mode: `x = 0..319px, y = 0..239px.` `Chr` can be in the range: `0..255`.

Function Declaration:

```
PrintChRpt(x,y,n,chr)
int x, y, n, chr;
{
    n &= 0xFF
    while(n--) {
        PrintChr(x++, y, chr);
    }
}
```

Example:

```
SetTxMode(XA_RC);           /* or XA_XY */
SetTxMode(XA_TN);
PrintChRpt(x,y, n, '=');
```

Function `PrintChr()`

Print single `chr` on graphic screen. `PrintChr()` respects `SetTxMode()` settings. For RC-Mode: `x = 0..79, y = 0..29` or in XY-Mode: `x = 0..319px, y = 0..239px.` `Chr` can be in the range: `0..255`.

Function Declaration:

```
PrintChr(x, y, chr)
int x, y, chr;
{
    x_func   = X_PrintChr;
    x_par[0] = x;
    x_par[1] = y;
    x_par[3] = chr;

    x_call();
}
```

Example:

```
SetTxMode(XA_RC);           /* or XA_XY */
SetTxMode(XA_TI);
PrintChr(x, y, '9');
```

Function `PrintStr()`

Prints null-terminated `chr-string` on graphic screen. `PrintStr()` respects `SetTxMode()` settings. For RC-Mode: `x = 0..79, y = 0..29` or in XY-Mode: `x = 0..319px, y = 0..239px`. `Chr` can be in the range: `0..255`.

Function Declaration:

```
PrintStr(x, y, str)
int x, y;
char *str;
{
    x_func   = X_PrintStr;
    x_par[0] = x;
    x_par[1] = y;
    x_par[4] = str;

    x_call();
}
```

Example:

```
SetTxMode(XA_RC);           /* or XA_XY */
SetTxMode(XA_TN);
PrintStr(x, y, "PrintChr. '9'");
```

File xkeyboard.h

Function summary:

- `xkey()`

Function `xkey()`

Check if key is pressed, if key pressed then get key, otherwise return 0. This function is a quick hack and may be MESCC dependent.

Function Declaration:

```
xkey()
{
    if(kbhit()) {
       return getch();
    }
    return 0;
}
```

Example:

```
- none -
```

File xgraph.h

Function summary:

```
      - Const-Definitions    \
      - #asm x_dat            | No explanation, see xgraph.h itself
      - #asm x_call          /
      - HelloRsx()
```

Function HelloRsx()

Checks whether the RSX is in memory or not - Returns NZ if true, else Z. This funktion should be called first (among others) and abort the programm if no RSX is found.

Function Declaration:

```
HelloRsx()
{
    x_func = X_Hello;

    return x_call() == X_SIGNATURE;
}
```

Example:

```
/* Check if the RSX is in memory */
if(!HelloRsx()) {
    puts("The RSX is not in memory!");
    return -1;
}
```